DigiFlow User Guide

Version 4.0, January 2017

© Dalziel Research Partners
2000 – 2017

Contents

1 Introduction **1**
1.1 History 1
1.2 Key features 2
1.3 User guide 2
2 Installation **3**
2.1 Basic installation 3
2.2 DigiFlow onfiguration 3
2.2.1 Basic configuration *4*
2.2.2 Encapsulated PostScript configuration *4*
2.2.3 Additional memory *6*
2.3 Installation with framegrabber 7
2.3.1 Framegrabber installation *7*
2.3.2 Camera configuration *7*
2.3.3 Local security policy *7*
2.3.4 Video capture configuration *8*
3 Basics **9**
3.1 Starting DigiFlow 9
3.2 Main Window 12
3.3 Obtaining help 13
3.4 Image Selectors 14
3.5 Archive files 14
3.6 Sifting 15
3.7 DigiFlow Macros 15
3.8 Threads 15
3.9 Text output 15
4 Common dialogs **18**
4.1 Open Image 18
4.2 Save Image As 22
4.3 Sifting input streams 23
4.3.1 Selector timing *23*
4.3.2 Selector region *24*
4.3.3 Matching intensities *26*
4.4 Modifying output streams 28
4.4.1 Setting output stream colour *28*
4.4.2 Full colour *29*
4.4.3 File format *29*
4.4.4 First index *29*
4.4.5 Resampling *29*
4.4.6 Save user comments *30*
4.4.7 Encapsulated PostScript streams *30*
4.5 dfc Help 31
4.6 Code library 32
5 Menus **34**
5.1 File 34
5.1.1 Open Image *34*
5.1.2 Run Code *34*
5.1.3 Save As *34*
5.1.4 Live Video *34*

5.1.5 Edit Stream......*44*
5.1.6 Merge Streams......*46*
5.1.7 Export AVI......*47*
5.1.8 Print View......*48*
5.1.9 Print Visible View......*49*
5.1.10 Export to EPS......*49*
5.1.11 Export Visible to EPS......*51*
5.1.12 Export to simple EPS......*52*
5.1.13 Close......*52*
5.1.14 Close All......*52*
5.1.15 Exit......*52*
5.2 Edit......52
5.2.1 Copy......*52*
5.2.2 Copy as bitmap......*53*
5.2.3 Zoomed Copy......*53*
5.2.4 Properties......*54*
5.2.5 Coordinates......*55*
5.2.6 Region......*60*
5.2.7 Process again......*61*
5.2.8 Dialog responses......*61*
5.2.9 dfcConsole......*62*
5.3 View......64
5.3.1 Zoom......*64*
5.3.2 Fit Window......*66*
5.3.3 Cursor......*66*
5.3.4 Vectors......*68*
5.3.5 Appearance......*68*
5.3.6 Colour scheme......*69*
5.3.7 Toggle colour......*71*
5.3.8 Toolbar......*71*
5.3.9 Slaves......*71*
5.3.10 Threads......*73*
5.3.11 Pause all threads......*74*
5.3.12 Refresh......*74*
5.3.13 In Parallel......*74*
5.4 Create......75
5.5 Sequence......75
5.5.1 Animate......*76*
5.6 Analyse......80
5.6.1 Time information......*80*
5.6.2 Ensembles......*92*
5.6.3 Dye images......*94*
5.6.4 Synthetic schlieren......*103*
5.6.5 Particles......*126*
5.6.6 Particle Tracking Velocimetry......*145*
5.6.7 Optical flow......*167*
5.7 Tools......171
5.7.1 Recipe......*171*
5.7.2 Transform intensity......*172*

5.7.3 Combine images ... *180*
5.7.4 Accumulate ... *185*
5.7.5 Slave process ... *186*
5.7.6 To world coordinates ... *188*
5.8 Window ... 189
5.9 Help ... 189
5.9.1 Help (browser) ... *189*
5.9.2 dfc Help ... *189*
5.9.3 Auto help ... *189*
5.9.4 About DigiFlow ... *189*
6 Techniques ... 191
6.1 Determining black ... 191
7 Chaining processes ... 193
8 Interpreter basics ... 196
8.1 Syntax ... 196
8.2 Variables ... 197
8.2.1 Simple variables ... *197*
8.2.2 Compound variables ... *198*
8.2.3 Type query functions ... *199*
8.3 Assignment ... 199
8.4 Arrays ... 199
8.5 Lists ... 202
8.6 Operators ... 203
8.7 Constants ... 204
8.8 Execution control ... 205
8.9 User-defined functions ... 207
8.10 User input and ouput ... 209
8.11 Input of code from files ... 210
8.12 Debugging ... 211
8.12.1 Error handling ... *211*
8.12.2 View variables ... *212*
8.12.3 Messages ... *213*
8.12.4 Queries ... *214*
8.12.5 Break points ... *214*
8.12.6 Tracing execution ... *215*
8.12.7 dfcConsole ... *215*
9 Functions ... 218
9.1 Basic mathematical functions ... 219
9.2 String functions ... 219
9.3 Array functions ... 219
9.4 Type manipulation functions ... 220
9.5 Information functions ... 220
9.6 Variable functions ... 221
9.7 File handling ... 221
9.8 Reading and writing images ... 222
9.9 Windows and views ... 223
9.10 Timing functions ... 224
9.11 Statistical functions ... 224
9.12 Image processing functions ... 224

9.13 Flow functions......225
9.14 Coordinate functions......225
9.15 Bit-wise operations......226
9.16 Camera control......226
9.17 Array plotting functions......228
9.18 Numerical functions......228
9.19 Differential functions......229
9.20 Handling threads......229
9.21 Web browsing......229
9.22 ftp functions......230
9.23 DirectDraw functions......230
9.24 Data acquisition functions......230
9.25 Serial communications......231
9.26 GhostScript functions......231
9.27 Particle tracking functions......231
9.28 Logging......232
9.29 Registry functions......233
9.30 Configuration and licence functions......233
9.31 Miscellaneous functions......233
9.32 All functions......233
10 Macros......238
10.1 DigiFlow command files......238
10.1.1 Running processes......238
10.1.2 Control of input streams......239
10.1.3 Control of output streams......243
10.1.4 Chaining responses......245
10.1.5 Multiple output streams......246
10.1.6 Accessing dialogs......247
10.2 Recording user input......248
11 Plotting and drawing......249
11.1 Drawing commands......249
11.2 The DigiFlow Drawing format......250
11.3 Simple plot......252
11.4 Text......253
11.5 LaTeX macros......253
12 Image file formats......256
12.1 Windows bitmap files (.bmp)......256
12.2 TIFF files (.tif)......256
12.3 GIF files (.gif)......256
12.4 Enhanced metafiles (.emf)......256
12.5 Windows metafiles (.wmf)......257
12.6 Encapsulated PostScript (.eps)......257
12.7 DigiFlow floating point image format (.dfi)......257
12.7.1 Header......257
12.7.2 Tag......258
12.7.3 8 bit image (DataType = #1001)......258
12.7.4 8 bit multi-plane image (DataType = #11001)......258
12.7.5 Compressed 8 bit image (DataType = #12001)......258
12.7.6 32 bit image (DataType = #1004)......259

12.7.7 32 bit multi-plane image (DataType = #11004) *259*
12.7.8 Compressed 32 bit image (DataType = #12004)... *260*
12.7.9 64 bit image (DataType = #1008)... *260*
12.7.10 64 bit multi-plane image (DataType = #11008) *260*
12.7.11 Compressed 64 bit image (DataType = #12008).. *261*
12.7.12 32 bit range (DataType = #1014) *261*
12.7.13 64 bit range (DataType = #1018) *261*
12.7.14 Rescale image (DataType = #1100) *262*
12.7.15 Rescale image rectangle (DataType = #1101) *262*
12.7.16 Colour scheme (DataType = #2000).. *263*
12.7.17 Colour scheme name (DataType = #2001)... *263*
12.7.18 Colour scheme name variable (DataType = #2002)................................. *263*
12.7.19 Description (DataType = #3000)... *263*
12.7.20 User comments (DataType = #3001)... *264*
12.7.21 Creating process (DataType = #3002) *264*
12.7.22 Creator details (DataType = #3003) *264*
12.7.23 Image time (DataType = #3018)... *264*
12.7.24 Image coordinates (DataType = #4008)... *264*
12.7.25 Image plane details (DataType = #4108) *265*
12.8 DigiFlow Particle tracking format.. 266
12.9 DigiFlow pixel data format (.dfp) 266
12.10 DigiFlow drawing format (.dfd)... 266
12.11 DigiFlow archive format (.dfa) 267
12.12 DigImage raw format (.pic)... 268
12.13 DigImage compressed format (.pic) 269
12.14 DigImage movie format(.mov or .dfm)... 270
13 Configuration files.. 272
13.1 DigiFlow_Licence.dfc.. 272
13.2 DigiFlow_LocalData.dfc.. 272
13.3 DigiFlow_Cameras.dfc 274
13.4 DigiFlow_Dialogs.dfs 278
13.5 DigiFlow_Status.dfs... 279
14 Extending DigiFlow... 281
14.1 Installing extensions... 281
15 Miscellaneous publications... 282
References 284
Index.. 285
16 Licence Agreement.. 291
Licence: 291
Warranty:.. 292
Other Conditions: 292

1 Introduction

DigiFlow provides a range of image processing features designed specifically for analysing fluid flows. The package is designed to be easy to use, yet flexible and efficient, and includes a powerful yet flexible macro language. Whereas most image processing systems are intended for analysing or processing single images, DigiFlow is designed from the start for dealing with sequences or collections of images in a straightforward manner.

Before installing or using DigiFlow, please read the Licence Agreement (see §16) and ensure you have completed the registration requirements.

1.1 History

The origins of DigiFlow lie in an earlier system by the same author: DigImage. This earlier system, with its origins in 1988 and first released commercially in 1992, pioneered many uses of image processing in fluid dynamics. Utilising its own DOS-extender technology, DigImage existed in the base 640kB of DOS memory (and later from the command prompt under Windows 3.x and 9x), accessing around 12MB of extended memory for image storage and interface with the framegrabber hardware.

To obtain the necessary performance in these early days of image processing on desktop computers, DigImage required a framegrabber card to be installed to provide not only image capture, but also image display and some of the processing. While this close coupling allowed efficient real-time processing and frame-accurate control of a video recorder, it ultimately restricted the development and deployment of the technology. The original ISA bus based Data Translation DT2861 and DT2862 frame grabber cards remained available until 2001, but by that time suitable motherboards had become difficult to source. At time of writing (2007) and despite its reliance on outdated technology, DigImage is still used in many laboratories around the world.

The development of DigiFlow began in 1994, although the project had a number of false starts and development put on hold a number of times due to other commitments. The code of this version has its origins in 1997 as part of the development of synthetic schlieren (see §5.6.4). The computational and resolution requirements for synthetic schlieren could not be accommodated efficiently within the framework of DigImage.

Despite sharing many approaches, algorithms and techniques, DigiFlow does not re-use any of DigImage's 8Mbytes Fortran 77 and 2MB Assembler source code. The design goals for power, flexibility and efficiency in DigiFlow could only be achieved by starting again from scratch.

DigiFlow builds on experience with DigImage from the user view point to provide a more powerful, more flexible, but simpler interface. It also builds on the programming experience to provide a more flexible, powerful and maintainable code base (now in excess of 15MB of source).

A central feature of DigiFlow is a powerful macro language (dfc) and interpreter. This provides users with an efficient and flexible environment in which to automate and customise processing, as well as proving to be a very useful general computational and plotting tool.

Versions of DigiFlow have been in use in Cambridge since 2000, and at other selected laboratories since 2002. Its wider dissemination began in late 2003 with a series of beta releases. The first commercial release (version 1.0) dates from February 2005, with parallel processing and other technologies providing substantial speed increases being introduced with version 2.0 during 2007. Version 3.0, released in 2008, provides further performance improvements plus a wealth of new processing features.

1.2 Key features

DigiFlow has been designed from the outset to provide a powerful yet efficient environment for acquiring and processing a broad range of experimental flows to obtain both accurate quantitative and qualitative output.

Central to design philosophy is the idea that an image stream may be processed as simply as a single image. Image streams may consist of a sequence of images (*e.g.* from a 'movie'), or a collection of images related in some other manner.

Efficiency is obtained through the use of advanced algorithms (many of them unique to DigiFlow/DigImage) for built in processing options.

Power and flexibility are obtained through an advanced fully integrated macro interpreter (using DigiFlow's dfc macro language) providing a similar level of functionality to industry standard applications such as MatLab. This interpreter is available to the user either to directly run macros, or as part of the various DigiFlow tools to allow more flexible and creative use. Commercial versions of DigiFlow include additional features such as partial compilation to further improve performance.

Although not an essential component, DigiFlow retains the potential DigImage released by the control of a framegrabber. Not only does this greatly simplify the process of running experiments, acquiring images, processing them, extracting and plotting data, but it also enables real-time processing of particle streaks and synthetic schlieren, for example.

1.3 User guide

This User Guide is designed to provide the primary reference for DigiFlow. The User Guide is supplied in both .html and .pdf formats and is linked to the help system within DigiFlow. Pressing the F1 function key within DigiFlow will start a web browser and take you to the most appropriate point in the .html version of the User Guide.

The User Guide is not in itself complete: detailed descriptions of the many functions provided by the macro interpreter may be found in the interactive help system (Help: *dfc* Functions). The User Guide is also supplemented by a variety of scientific publications that expand on some of the underlying technologies.

The typographical convention used in the User Guide is described below:

Typography	Description
Analyse	Windows elements such as prompts, menu items and dialogs.
Expt_A.dfi	File names, *etc.*
read_image()	Interpreter commands and functions.
:=	Interpreter operators and syntax.
"string"	Interpreter operators and syntax.
# comment	Formal argument names for interpreter functions.
my_image	Variables, numbers, *etc.*, for the interpreter.
*file*0	Formal argument names for interpreter functions.

2 Installation

Although DigiFlow will work on any Windows XP or later machine, we recommend that you avoid using Windows Vista if possible as the performance of Vista is significantly worse than either Windows XP or Windows 7. There are versions of DigiFlow that can operate under both 32-bit and 64-bit implementations, although at present it cannot control a digital video camera under a 64-bit implementation of Windows.

2.1 Basic installation

DigiFlow is a typical Windows application with a graphical user interface, menus, dialog boxes and toolbars. However, unlike many applications, DigiFlow does not require a special installation procedure, but can simply be copied to the desired directory. In most cases DigiFlow will be delivered in a .zip or self-extracting (.exe) archive file, downloaded from the web. This should simply be unzipped into your selected directory. However, to make the best of DigiFlow, there are some additional settings and tasks to be completed. The setup.bat file that is copied to the installation folder will help with this process. Refer to GettingStarted.pdf for further details.

The installed part of DigiFlow consists of DigiFlow.exe, which contains the core functionality, and a range of DLL files that handle specific menu options. DigiFlow also makes use of various global start-up files stored in the same directory.

During use, DigiFlow generates two status files in the directory in which it is started. These are DigiFlow_Status.dfs (§13.5), which contains a range of information describing the settings, and DigiFlow_Dialogs.dfs (§13.4), which records your last responses to many of the prompts, *etc.* By storing this information in the directory in which DigiFlow is started, DigiFlow is able to keep a separate set of information for each user, or for each specific task, without polluting the registry. Additionally, these status files can be deleted or moved as the user wishes. In some circumstances, DigiFlow_Status.dfs may become corrupted. If DigiFlow fails to start, or exhibits unexpected behaviour, you should try removing (or renaming) DigiFlow_Status.dfs to see if this cures the problem.

It is recommended that you use a new directory for each new set of experiments and for each new project. In this way the DigiFlow strategy of storing localised status files will facilitate use of DigiFlow in the various different contexts. In such an environment it is frequently most convenient to start DigiFlow from the command prompt within the appropriate directory structure, although other strategies such as multiple shortcuts or setting up associations for Windows Explorer are also possible.

If you wish to run DigiFlow from a command prompt (strongly recommended), it is worth putting this directory on the path so that DigiFlow may be started by simply typing `DigiFlow` at the prompt (DigiFlow will normally add itself to the search path the first time it is run to enable this). If you prefer to start DigiFlow from the desktop or start menu, you will need to create a shortcut at that point and set the Start in directory appropriately. It is strongly recommended that you do not run DigiFlow from the directory in which the program resides, exept during the set-up procedure.

2.2 DigiFlow onfiguration

Details of the basic setup and configuration of DigiFlow under Windows is covered in GettingStarted.pdf. This section reiterates some of the key points and highlights other considerations that may facilitate your use of DigiFlow. Note that DigiFlow can also be installed to run under Wine on a Linux machine, although it is not possible to control a digital

video camera and .eps (Encapsulated PostScript) files do not have access to the normal range of fonts and appear visually less satisfying.

2.2.1 Basic configuration

Specification of the file extension for file names within DigiFlow is mandatory in most circumstances as DigiFlow utilises this extension to determine the file type for output. However, by default, Windows XP and later hide the extensions to files of known types, a feature that can cause problems with DigiFlow. We recommend, therefore, that you turn off this feature. DigiFlow will attempt to do this for itself, but this may not work on some systems. If DigiFlow does not make all extensions visible automatically, then you may achieve this manually through the View tab of Tools: Folder Options under Windows Explorer. Simply remove the check mark from Hide extensions for known file types. Note that this will need to be done for each DigiFlow user.

By default, DigiFlow will not be associated with any file types or extensions, unless you install it using setup.bat (in which case .dfc, .dfd, .dfi, .dfm, .dfs and .dft will be associated with DigiFlow). The easy way to make or add such associations is to right-click on a file with such an association then select Open with (or Open if Open with is not visible) and choose the default program from the Open With dialog and check the Always use… box. If DigiFlow is not listed in this dialog, then locate it using the Browse button.

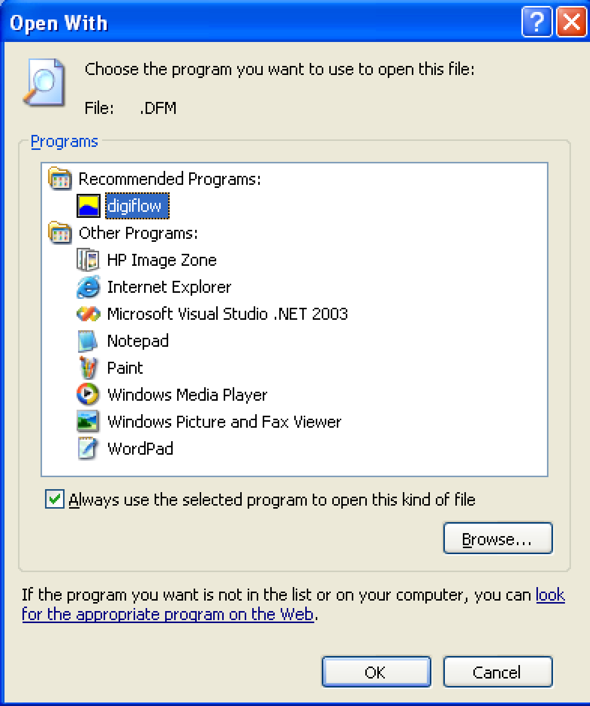


Figure 1: The Open With dialog for selecting the default program.

We recommend that the following extensions are associated with DigiFlow on all installations: .dfc, .dfd, .dfi, .dft and .dfs. You may also wish to set up associations for other standard image formats such as .bmp, .tif, .png and .jpg.

2.2.2 Encapsulated PostScript configuration

DigiFlow can create Encapsulated PostScript (.eps) files from image and graphical output for incorporation into documents in packages such as LaTeX and Word. This can be achieved either through DigiFlow's inbuilt .eps facility, or using a Windows printer driver. The former is restricted to bit images (or a rasterised version of graphics), whereas the latter can produce both bit image and vector graphics.

By default, DigiFlow searches for a printer named EPS to use to create the .eps files. Creation of this printer is relatively straight forwards. Start the Add Printer Wizard from the Printers and faxes window, selecting Local printer attached to this computer and using the File: (print to file) port. Select a PostScript printer driver (we recommend the `HP C LaserJet 4500-PS` if you are using Windows XP, or the `Xerox Phaser 6120 PS` if you are using Windows 7) and name the printer "EPS". (You do not want to make this the default printer, you may, however, wish to share the printer to simplify the setting up of further machines.) For Windows Vista, it is recommended that you download an Adobe PostScript driver from www.adobe.com as some of the drivers distributed with Windows Vista format their PostScript in a manner that inhibits the use of LaTeX packages such as psfrag.

Once the wizard has finished, right-click on the new EPS printer and select Printing preferences. Click on the Advanced button expand Document Options and PostScript Options within it. Under PostScript Output Option select Encapsulated PostScript (EPS), as indicated in figure 2.

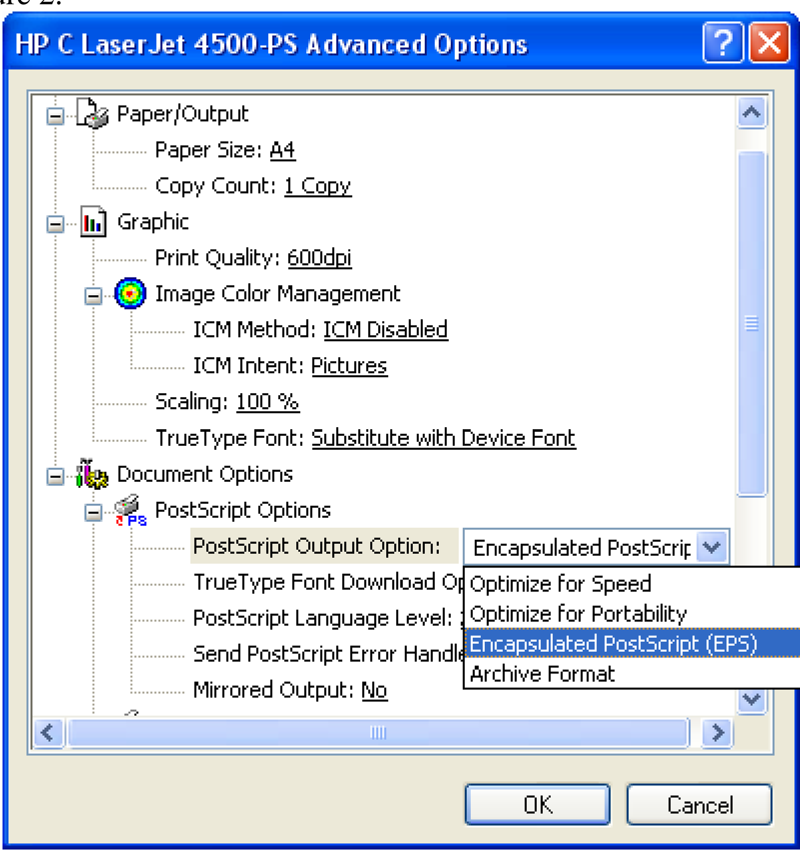


Figure 2: Encapsulated PostScript (.eps) printer setup.

Note: if you are using Remote Desktop to access the computer with DigiFlow installed, you are best to disable the feature making local printers available to the remote session as this can cause problems if the EPS printer exists on your local machine.

DigiFlow cannot itself read back in an Encapsulated PostScript file it produces. However, if DigiFlow detects that GhostScript is installed on the machine, then DigiFlow will attempt to use GhostScript to help it load the .eps file in an appropriate format. For this to be achieved, then GhostScript must be on the system `PATH` and the `GS_LIB` environment variable must be set up to point to the GhostScript libraries.

Note that GhostScript is **not** distributed with or required by DigiFlow. Use of GhostScript is governed entirely by the licence of that product and not by the DigiFlow Licence.

2.2.3 Additional memory

The maximum linear address range under 32 bit Windows is 32 bits or 4GB. By default under Windows this is subdivided into two ranges for each process. The first 2GB of memory is for the process's own use, while the second is for the operating system. Although 2GB superficially appears a lot, there are times when it would be useful to have more. (At the time Windows was designed, 2GB was considered a good approximation to an unlimited memory resource, but things have moved on…) With Windows XP and later it is possible to change the 50:50 default split to reserve 3GB for processes, restricting the system. Not all software, particularly some drivers, support this extension. DigiFlow, however, is able to and so if you start running low on virtual memory, it may be worth a try.

To install the 3GB process memory option, select System Properties (right click on My Computer and select Properties) then the Advanced tab. Click the Startup and Recovery Settings button, then the Edit button to open NotePad to make the necessary changes. Note that you need to have Administrative access rights to be able to do this.

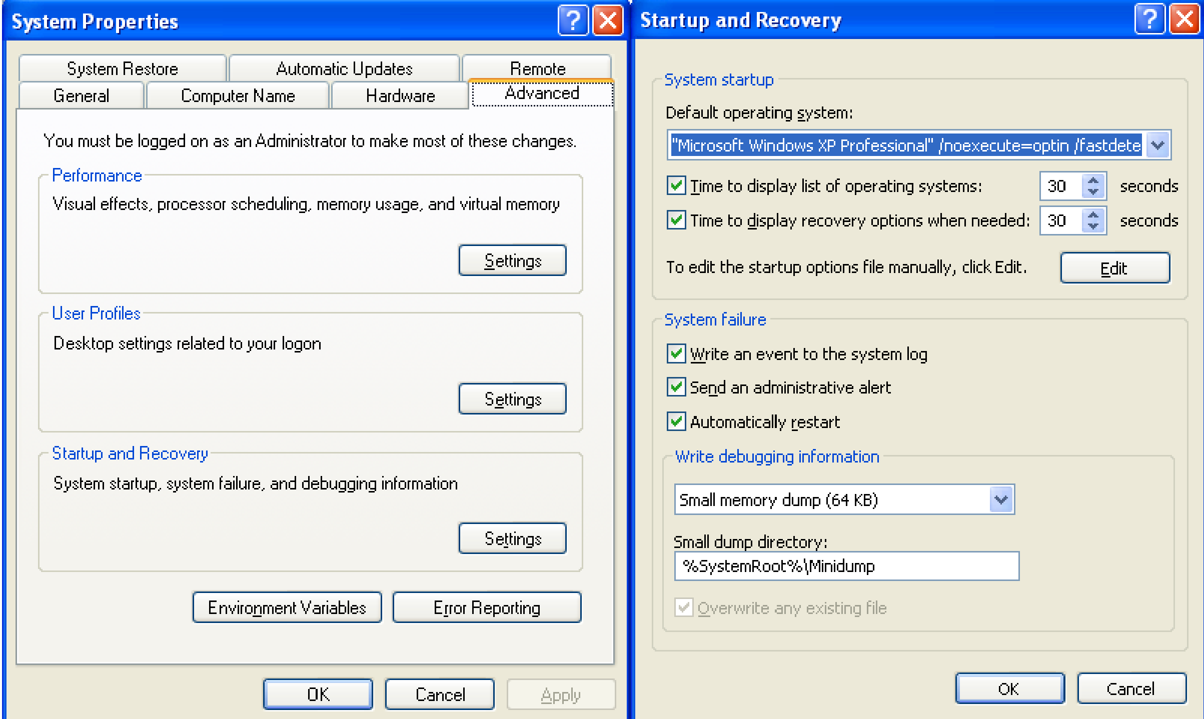


Figure 3: Dialogs for setting the /3GB option to increase available virtual memory.

NotePad will allow you to edit the boot.ini file that controls the startup of Windows. Typically, this will look like

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP
      Professional" /noexecute=optin /fastdetect
```

To enable the 3GB option, you need to add the /3GB switch to the end of the line specifying Windows startup. It is best to do this by adding an additional startup option so that you can boot your machine in either standard 2GB or 3GB modes. The resulting boot.ini should look something like this:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
```

```
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP
    Professional" /noexecute=optin /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP
    Professional (3GB)" /noexecute=optin /fastdetect /3GB
```

Note that the order of these two lines determines whether the default boot is 2GB or 3GB. In the above example, the standard 2GB boot is the default. Reverse the order of the two lines to make the 3GB boot the default.

2.3 Installation with framegrabber

If you are installing DigiFlow in a machine equipped with a BitFlow R2, R3, R64 or R64e series framegrabber then some additional steps are required. These require administrative access to implement.

2.3.1 Framegrabber installation

The framegrabber should be installed and tested using the BitFlow installation procedure. You will require the BitFlow drivers for version 5.00, 5.20 or 5.30. Later versions may also be supported (contact Dalziel Research Partners for details). You should note that on some systems the BitFlow installation procedure can hang; if this occurs, try installing after rebooting Windows in Safe Mode.

The BitFlow framegrabber requires a configuration file (.cam, .rcl or .r64) for the camera being used. Configuration files for cameras known to work with DigiFlow may be found at http://www.dalzielresearch.com/digiflow/cameras/.

If you have a multi-user system where most users do not have administrative access, we recommend that you change the permissions on the BitFlow software to allow all users to change the camera configuration file if and when they need to. This is achieved using the Registry Editor (regedit.exe; accessible from the command prompt) to adjust the permissions on all keys in the registry relating to 'BitFlow' by adding the 'Authenticated Users' security principle with 'Full control'. Failure to do this would mean that only users with administrative access could change the camera configuration.

2.3.2 Camera configuration

DigiFlow requires information over and above what is provided in the configuration file for the framegrabber card. This additional information about the camera capabilities and users preferences is stored in DigiFlow_Cameras.dfc; consult §13.3 for details of the format of this file. Cameras not listed in this file have not been tested, although there is a reasonable chance that all that is required (for a camera supported by the BitFlow frame grabber) is the addition of appropriate entries, provided a suitable camera configuration file is also available for the BitFlow framegrabber. Please contact Dalziel Research Partners if you require any help or guidance with this.

2.3.3 Local security policy

In the 'Local security policy' (found in the 'Administrative tools' section of the 'Control Panel'), open the 'Local Policies: User Rights Assignment' option. You need to add permission for all DigiFlow users to the following items:

- Adjust memory quotas for a process
- Increase scheduling priority
- Lock pages in memory

It is suggested that you do this by giving full control to 'Authenticated users'

These adjustments are necessary to ensure that DigiFlow is able to manage the machine performance adequately to ensure trouble-free capture.

2.3.4 Video capture configuration

It is strongly recommended that video capture is to a disk other than that containing the operating system in order to obtain adequate performance. The necessary disk system bandwidth may be in excess of 240MB/s in some cases (*e.g.* with a Dalsa 4M60 camera), thus requiring a Mode 0 RAID array, or using Windows to 'stripe' across multiple disks. However, for most cameras 40MB/s is sufficient and this may be achieved via a fast IDE or SATA disk (but not the one the operating system is on!).

The capture process in DigiFlow can be configured in two ways. Either you can directly specify the capture file and location each time (risking the user specifying a disk system with insufficient bandwidth), or setting up DigiFlow to capture to a fixed location and require the user to 'review' (and possibly edit) the sequence in order to copy it into their own directory space. For multi-user systems, this second is generally preferred as it allows users to utilise the capture facility like a video recorder while preventing retention of unwanted video footage.

The default configuration takes the second option, and assumes that the capture location is V:\Cache\CaptureVideo.dfm. We recommend that you configure your system so that this directory exists (either by appropriate naming of the capture disk, or by setting up a share to an appropriate point and then connecting to it). This directory must not be compressed and must have full access for all DigiFlow users. Once you have created this directory, you should run File: Live Video: Setup (see §5.1.5.3 for further details) to create the initial V:\Cache\CaptureVideo.dfm. **It is strongly recommended that you do this before writing any other data to the capture disk.** Details on how to change the name or location of the cache file may be found in 13.2.

It is important that the space DigiFlow reserves in this file remains as a single contiguous block on the disk drive. If it becomes fragmented for any reason then, due to the very high data transfer rates required, DigiFlow may not be able to write to the disk as fast as data becomes available from the camera and so timing errors may result.

Once created, V:\Cache\CaptureVideo.dfm will be flagged as Read only by the operating system (although DigiFlow will still be able to write to it). The file will not shrink if a smaller sequence is captured, but may grow if one larger than that specified during File: Live Video: Setup is requested (note that there is a risk of fragmentation if this occurs). It is important, therefore, that you go through the review process outlined in §5.1.5.2, rather than simply copying this file, as in general only a part of the file will contain valid data.

Consult §13.2 on DigiFlow_LocalData.dfc should you wish to change the name or location of V:\Cache\CaptureVideo.dfm.

3 Basics

3.1 Starting DigiFlow

It is recommended that you use a new directory for each new set of experiments and for each new project. In this way the DigiFlow strategy of storing localised status files will facilitate use of DigiFlow in the various different contexts. In such an environment it is frequently most convenient to start DigiFlow from the command prompt (see figure 4) within the appropriate directory structure, although other strategies such as multiple shortcuts or setting up associations for Windows Explorer are also possible.

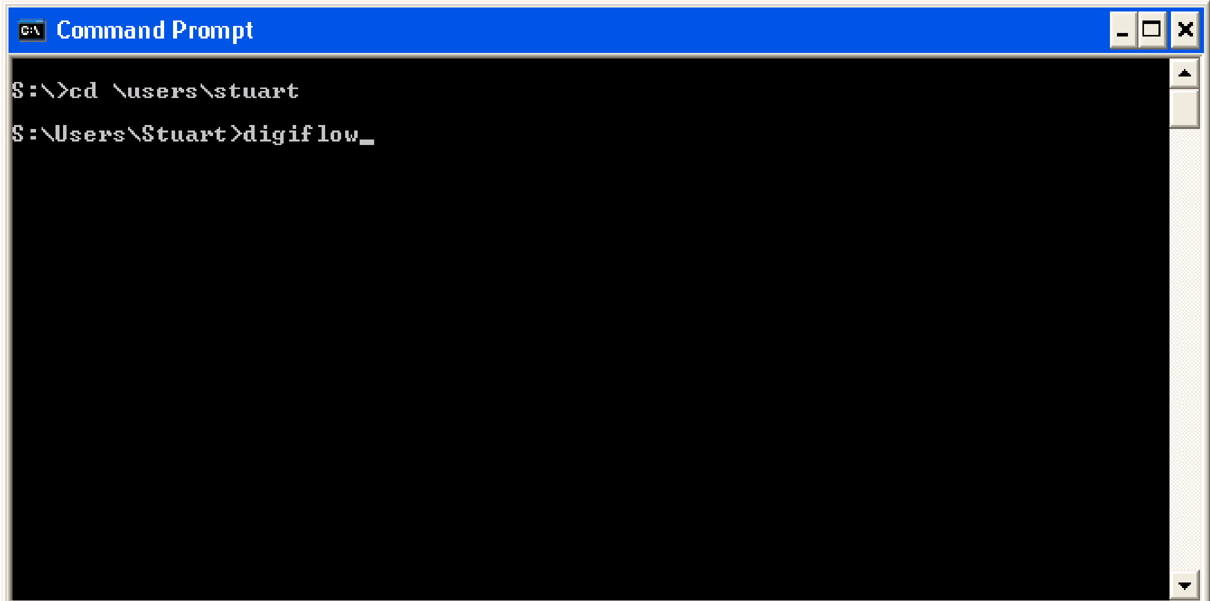


Figure 4: It is frequently most convenient to start DigiFlow from the command prompt.

If you wish to run DigiFlow from a command prompt (strongly recommended), it is worth putting this directory on the path so that DigiFlow may be started by simply typing `DigiFlow` at the prompt, and placing a shortcut to the command prompt on the topmost level of the Start button. It is also worth pinning a shortcut to the command prompt to the start menu (and/or task bar for Windows 7), and in the properties setting the Start in path to an appropriate location.

If you are not familiar with the use of the command prompt, then the following brief list of the most useful commands may be of some value.

Command	**Description**
`cd` *folder*	Changes to the directory (folder) located within the current directory.
`cd ..`	Move up one directory level.
`cd \`	Move to the topmost (root) directory on the current drive
`dir`	List all files in the current directory.
`dir *.dfi`	List all .dfi files in the current directory.
`dir *.dfc /s`	List all dfc files located either in the current directory, or any subdirectories.
`move` *file dest*	Move *file* to a new directory *dest*. Can also be used to rename folders.
`copy` *srce dest*	Copy the file *srce* to *dest*.
`xcopy` *srce dest*	A more flexible form of `copy`.

xcopy *srce dest* /s/d	If *srce* and *dest* are directories, then will copy all the files in the directory and any subdirectories to *dest*, but only if the *srce* version is newer. This might be used, for example, as xcopy expt*.dfd results /s/d to update a collection of DigiFlow drawings (dfd files) in a results folder.
ren *file new*	Rename *file* with the name *new*.

A very useful feature of Windows XP and later is that the <tab> key will expand a file name. For example, if you are in a directory that contains subdirectories named Expt1, Expt2 and Expt3, then typing cd e followed by <tab> will expand this to cd Expt1. Pressing <tab> a second time will change this to cd Expt2, and so on. To find out more about the commands available at the command prompt, then search for command prompt in the Window's Help and Support Centre and select Using Command Prompt. Alternatively, if you know the name of the command but want more details of its options, type the command followed by /? at the command prompt (see figure 5 for an example).

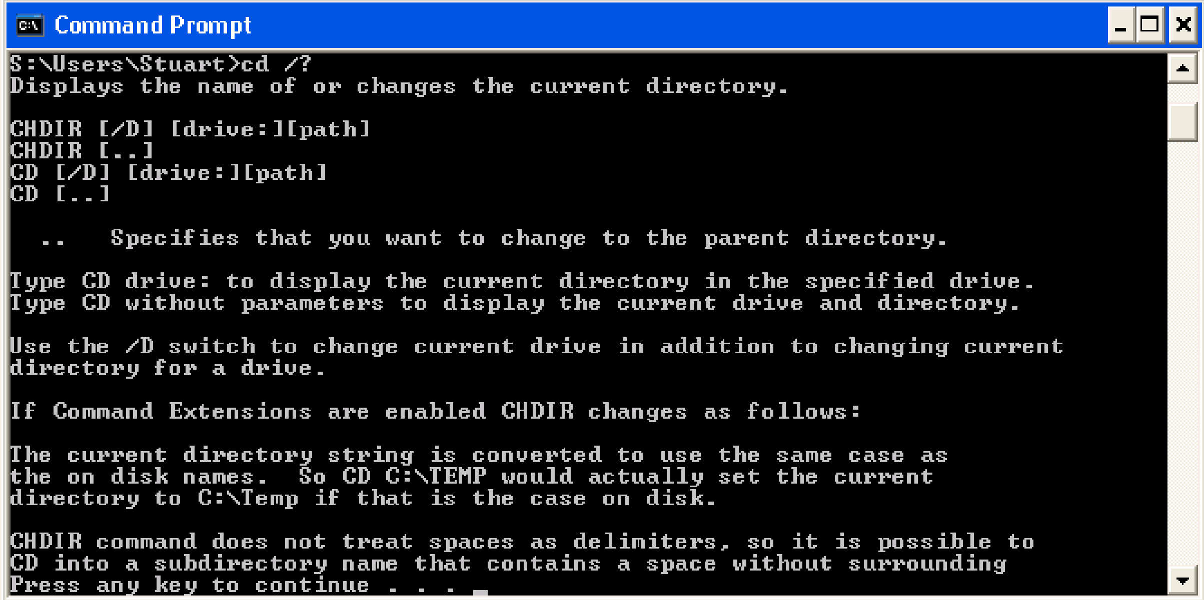


Figure 5: Help is available for a command at the command prompt by adding /? after the command.

If you prefer to start DigiFlow directly from the desktop or start menu, you will need to create a shortcut at that point and set the Start in directory appropriately (see figure 6). It is strongly recommended that you do not normally run DigiFlow from the directory in which the program resides.

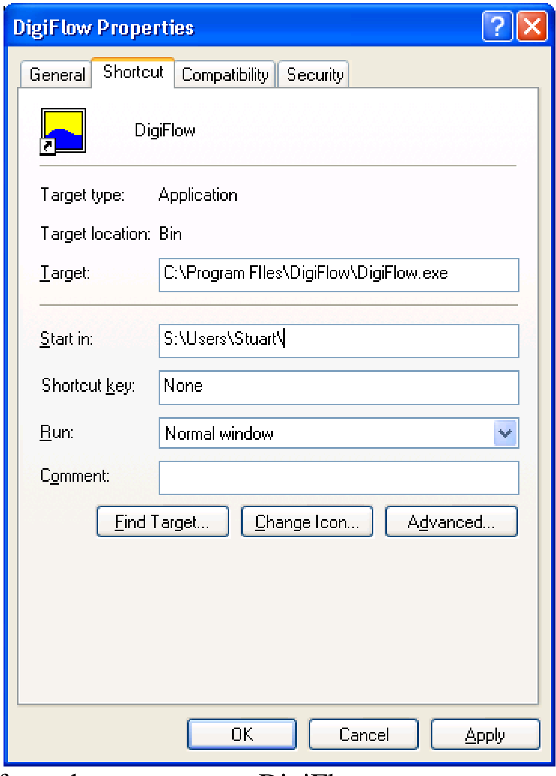


Figure 6: Properties dialog for a short cut to start DigiFlow.

DigiFlow supports a number of command line arguments. The most common use is to specify either an image file or movie to be loaded when DigiFlow starts, or a dfc file to be run. In both these cases, simply type `DigiFlow` at a command prompt, followed by the name of the image file/movie or dfc file.

Additionally, there are a number of command line switches that can be used in special circumstances. These are given in the following table.

Switch	**Description**
`/?`	Give the command line options for starting DigiFlow.
`/allowautorun`	Causes files named _autorun.dfc to be started automatically when they are created in the current folder. Note that they will be deleted once they have been run.
`/archive`	Turns on the .dfa archive file generation system when not enabled by default.
`/noarchive`	Turns off the .dfa archive file generation system.
`/dfa`	Identical to `/archive`.
`/autopreprocess`	Turns on the auto-preprocess mechanism (using .dfb filter files).
`/noautopreprocess`	Turns off the auto-preprocess mechanism (using .dfb filter files).
`/bitflow:`*n*	For machines with more than one BitFlow framegrabber installed, specifies which board *this* instance of DigiFlow is to use. Note: there should be no space between the colon and the number.
`/camera:`*camerafile*	Specifies that a particular camera configuration file should be used rather than the default (specified through the BitFlow `SysReg` utility. Note: there should be no space between the colon and the name of the camera file.
`/camerabuffer:`*option*	Specifies the type of buffering to be used for the capture file. One of: `"(default)"`, `"buffer"`, `"nobuffer"` or `"writethrough"`.
`/con`	Start up a conole window at the same time as starting DigiFlow.

	(The console window can also be started from dfc code using `open_console()` or `open_file()`.)
`/debug`	Turns on all logging options (to DigiFlow.log) for debug purposes.
`/dev`	Enable certain features related to internal performance monitoring. Intended for use by the developer.
`/disablewritequeue`	Disable the threaded writing of images, thus forcing the process generating the image to wait until the image has been written before moving on to the next stage.
`/enablewritequeue`	Enable the threaded writing of images. This speeds up processing by allowing the process generating the images to proceed to the next image before the write is complete. Enabled by default.
`/disableparallelwrite`	Certain aspects of reading and writing images may have problems if accessed simultaneously by multiple threads. Use of this flag introduces mutexes (blocks of code where a thread has mutually exclusive execution) to prevent simultaneous access. This has a detramental impact on performance.
`/enableparallelwrite`	Certain aspects of reading and writing images may have problems if accessed simultaneously by multiple threads. Use of this flag turns off the mutexes (blocks of code where a thread has mutually exclusive execution) to prevent simultaneous access. Allowing simultaneous access improves performance.
`/dpi:`*n*	Tells DigiFlow to assume the display has *n* pixels per inch. This is used to work out the size of some of the visual elements in the user interface. Note that including `DigiFlow.Options.Display.dpi` and `DigiFlow.Options.Display.Scaling` in DigiFlow_LocalData.dfc sets the assumed pixels per inch and scaling (respectively) of the screen.
`/stack:`*multiple*	Sets the multiple for stack size allocation. If not specified, thenthe default multiple is 1.0. This switch can be used in an attempt to decrease or increase the size of the stack created for separate computational threads in event of memory problems.
`/timing`	This turns on a performance timing feature incorporated in some of the DigiFlow facilities.
`/wine`	Changes some features to improve performance when running DigiFlow under Wine on Linux.

The final option for starting DigiFlow is to double-click in Windows Explorer on a file associated with DigiFlow.

3.2 Main Window

The main DigiFlow window follows that common for most applications with a Multiple Document Interface (MDI). The menu bar at the top provides access to the majority of the facilities, while the toolbar underneath gives a more convenient method of accessing the more widely used functions. A typical example is shown in figure 7.

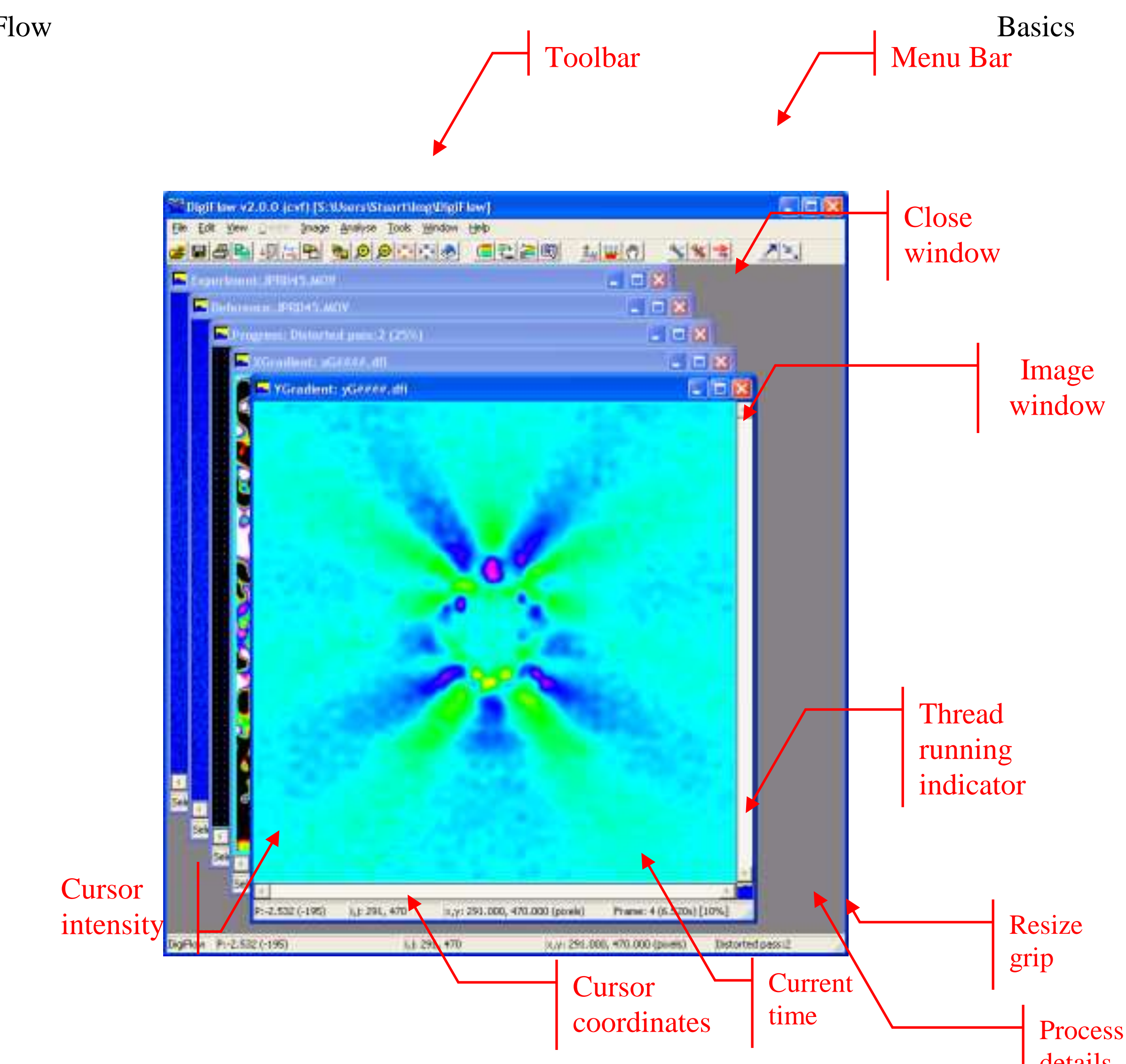


Figure 7: The basic DigiFlow window.

As is normally the case for Windows applications, the main window and the client windows may be resized by dragging the frame of the window. Holding down the control key, while dragging the boundary of a client window, will cause the contents of the window to be zoomed so as to make the best use of the available space. If you do not hold down the control key, then the window size is changed without changing the zoom applied to its contents.

3.3 Obtaining help

As is common with most Windows applications, help while using DigiFlow can be obtained by pressing the f1 key. This will start an instance of Internet Explorer and bring up an html copy of this manual. DigiFlow will automatically scroll to the position within the manual that is most relevant to the dialog or process you have open at the time. Subsequent presses of f1 will utilise the same tab in Internet Explorer, provided this remains open.

Some users, particularly when first starting, may prefer to have the manual automatically keeping track with their activities. This can be achieved by turning on the AutoHelp facility, either from the Help menu, or by clicking the questionmark button () on the toolbar. When activated, an instance of the Internet Explorer browser will be activated and track your activity, providing timely help.

As an option to using the html version of this manual, the manual is also provided as DigiFlow.pdf.

3.4 Image Selectors

DigiFlow uses image selectors to specify image streams for input to and output from a given process. Four types of image stream are supported:

Single images. These contain just a single image.

Movie. A movie contains multiple images stored in a single file.

Sequence. A sequence is a collection of related files, typically identified by a numeric part of the file name that increases by one between neighbouring images in the sequence.

Collection. A collection is a group of image files that have no special relationship to each other. Collections may be subdivided into two groups: *homogeneous collections* and *heterogeneous collections*. In a homogeneous collection, all the images within the collection have the same format (same size, colour depth, file type, *etc.*). With a heterogeneous collection, the format may vary from one image to another. At present, most processes within DigiFlow do not support heterogeneous collections.

Image selectors may specify not only raster format image files, but also vector format files. DigiFlow supports many standard raster formats, including .bmp, .tif, .gif, .png, .jpg, and .avi along with special formats to provide backward compatibility with DigImage (.pic and .mov, the latter now renamed .dfm in DigiFlow). DigiFlow also introduces the new DigiFlow Image format, .dfi, to allow images to be saved with full floating point precision, and the DigiFlow Pixel format (.dfp) provides text output specifically tailored for raster images.

Vector format files include Enhance Meta Files (.emf) Windows Meta Files (.wmf) and DigiFlow Drawing format (.dfd). The last of these provides output formatted as plain text containing both data and drawing commands. This text may be imported into other applications, or read back into DigiFlow to reconstruct the image or drawing it represents. If GhostScript is installed on the system (see GettingStarted.pdf), then Encapsulated PostScript (.eps) files can also be opened with DigiFlow.

DigiFlow also provides a specialised file format (.dft) for storing particle tracking data. While these may be treated as images, in general the functionality available through the specialised particle tracking facilities is to be preferred.

The specialised DigiFlow and DigImage formats (.dfm, .dfi, .dfp, dft and .dfd) are described more thoroughly in §11.2.

3.5 Archive files

The concept of an archive file was introduced to DigiFlow in version 3.4 as a method of both collecting multiple image files that form part of a sequence, and of storing additional details about an image or a sequence of images when the file format being used does not provide a mechanism for storing this information.

DigiFlow archive files use a .dfa extension appended on to the corresponding image file or sequence name. For example, if a .dfa file is generated for a sequence of images image000.png, image001.png, … image732.png, then the name of the .dfa file will be image###.png.dfa.

If the use of archive files is enabled, then when reading the image sequence back in as image###.png, additional information not present in the .png files themselves (such as the method used to construct the sequence and the timing of the sequence) will be recovered from the .dfa file. Alternatively, attempting to open image###.png.dfa will both read the sequence image###.png and recover the additional information from the .dfa file.

The use of DigiFlow archive files is enabled or disabled through the Open Image and Save Image dialogs (see §4.1 and §4.2). For some builds of DigiFlow, .dfa generation is disabled by default. Under these circumstances, .dfa support may be turned on by including the switch `/dfa` on the command line.

For further details on the format of .dfa files, refer to §12.11.

3.6 Sifting

A key concept associated with input image streams is *sifting*. In DigiFlow, sifting is the process by which images are extracted from in input stream. The extraction process may result in all the images being extracted, or only a subset of images (typically specified by a start number, an end number and a step). It may also result in a subregion of the image (a rectangular *window* within the image) being returned, or, in the image being modified to conform to some reference. Further details of the sifting process are given in §4.3.

3.7 DigiFlow Macros

DigiFlow includes a powerful interpreter and associated macro language. The language is referred to as dfc code. While the programming language for dfc code is specific to DigiFlow, it follows the general syntax and conventions of many other modern high-level languages. In addition to the basic functionality expected of such languages, DigiFlow provides a vast range of functions tailored specifically to tasks for which DigiFlow is ideal. This includes not only image processing functions (ranging from contour tracing to Fast Fourier Transforms), and data analysis functions (such as statistics, least squares fits), to numerical solution of the equations of motion (*e.g.* Goudnov solution of shallow water equations and stream function-vorticity formulation for two-dimensional Boussinesq flows).

The present manual contains introductory documentation for the use of dfc functions and code. However, much of the detailed documentation for the individual dfc functions is to be found in the interactive help system Help: dfc Functions. The most convenient way of accessing this is frequently through the dfcConsole feature described in §5.2.10. The DigiFlow macros\ subdirectory (found in the folder where DigiFlow is installed) contains a number of documented examples of macro code.

3.8 Threads

One important aspect of DigiFlow is that it supports not only multiple image windows, but also multiple processing threads. This has two important benefits. First, it allows DigiFlow to continue to be used interactively while it is processing simultaneously one or more sequence of images, thus allowing real-time inspection of the progress. Second, for PCs with multiple processors, the execution time of a single process can be greatly reduced. (It should also be noted that more than one copy of DigiFlow may be used simultaneously).

If the user attempts to close a window that is in use with an active thread, then the system will warn the user that closing the window will also kill the thread. Depending on the version of DigiFlow you are using, windows that are playing a role in an active thread have the name of the thread indicated in the window status bar at the bottom of the window and have a 'sloshing tank' symbol in the bottom right-hand corner.

The user may also control the individual threads more directly, stopping them, pausing or resuming them, or changing their priority. This is achieved through the View Threads menu item (§5.3.10), or the corresponding button on the main toolbar. (All active process threads may be suspended by clicking on the toolbar.)

3.9 Text output

Some of DigiFlow's features include graphical and text output. In such cases it may be desirable to include more than simple plain text. To achieve this, fully licenced copies of DigiFlow support LaTex-like math-mode text formatting. For example, in Analyse: Time Series: Summarise (see §5.6.1.6) it is possible to specify the titles of the axes of the graph produced. Specifying it as the string `"Dimensionless height $\big(\frac{h}/{\alpha^2H_0}\big)$"` would produce the label

$$\text{Dimensionless height } \left(\frac{h}{\alpha^2 H_0}\right).$$

Although DigiFlow does not understand the full range of LaTeX commands and macros, it can interpret those most likely to be of use in figures and graphs. The list includes:

Upper and lower case greek letters (*e.g.* \Alpha or \zeta)

$	\ddot	\lbrace	\P	\supset
\!	\div	\lbrack	\partial	\supseteq
\#	\dot	\le	\phantom	\surd
\$	\dots	\left(	\pm	\tan
\%	\downarrow	\left[	\pounds	\tanh
\&	\Downarrow	\Leftarrow	\prime	\textbf
\,	\ell	\leftarrow	\prod	\textit
\2dots	\equiv	\Leftrightarro	\propto	\textnormal
\:	\euro	w	\qquad	\textrm
\;	\exists	\leftrightarro	\quad	\therefore
\aleph	\exp	w	\rangle	\tilde
\angle	\footnotesize	\leq	\rbrace	\times
\approx	\forall	\ll	\rbrack	\tiny
\backslash	\frac	\ln	\Re	\underline
\bar	\ge	\log	\right)	\Uparrow
\bf	\geq	\mathbf	\right]	\uparrow
\big	\gg	\mathit	\rightarrow	\wedge
\BIG	\hat	\mathrm	\Rightarrow	\wp
\Big	\HUGE	\minus	\S	\yen
\bigsizes	\huge	\mp	\scriptsize	\\
\bullet	\Im	\nabla	\sim	\^
\cdot	\in	\ne	\simeq	_
\circ	\infty	\neq	\sin	\{
\copyright	\int	\normalsize	\sinh	\}
\cos	\it	\notin	\small	\~
\cosh	\langle	\oplus	\sqrt	
\dagger	\LARGE	\oslash	\subset	
\ddagger	\large	\otimes	\subseteq	
\dddot	\Large	\overchar	\sum	

Thereare some minor restrictions and additional requirements for the DigiFlow LaTeX-like syntax compared with standard LaTeX. For example, the standard LaTeX `$\left(\frac{a}{b}\right)$` should be stated in DigiFlow as `$\left({\frac{a}{b}}\right)$`. The additional pair of braces tells DigiFlow that the fraction *a*/*b* is controlling the size of the large left bracket. This additional pair of braces does not affect the processing of the string by LaTeX. A further example is that DigiFlow accepts LaTeX macros such as `\alpha` whether or not it is in 'maths mode' (*i.e.* between `$..$`).

This LaTeX-like text formatting (available only with fully licensed copies of DigiFlow) may also be used in dfc code, for example through the **draw_text**(..), **draw_axes**(..) and **plot_titles**(..) commands.

It is not possible to define additional LaTeX-like macros from within LaTeX-like formatted text. However, additional macros may be defined from within dfc code; see §11.4 and the dfc help for further details.

A powerful feature of this component of DigiFlow is the way it works to support the use of Encapsulated PostScript (.eps) files in LaTeX through the psfrag macro package. See §5.1.12 for further details.

4 Common dialogs

4.1 Open Image

The Open Image dialog box is used throughout DigiFlow to open source image selectors (§3.1).

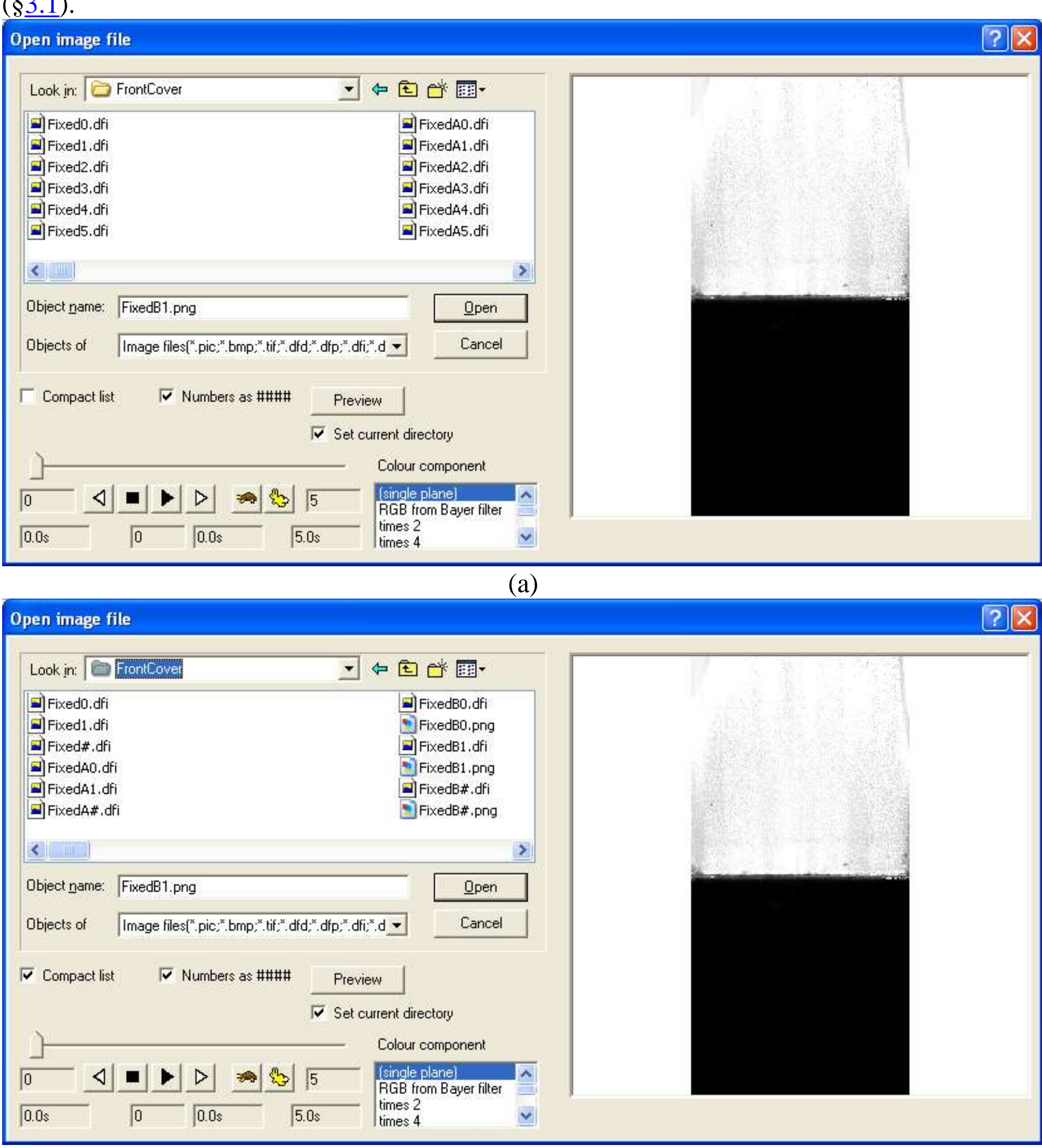


Figure 8: The Open Image dialog box under Windows XP. (a) Showing all files and (b) using Compact List option.

The Open Image dialog box consists of a standard Explorer-style display of folders, files, file types, *etc.*, along with a preview pane on the right-hand side. This preview pane will attempt to display the currently selected file.

DigiFlow supports a range of industry standard image formats, plus some special formats. The special formats both provide compatibility with the earlier DigImage system, and provide

facilities (*e.g.* floating point data representation) not found in industry-standard formats. These non-standard formats are described in more detail in §11.2 (DigiFlow drawing format) and §12 (DigiFlow image file formats). Note that DigiFlow expects the user to specify the extension of the file. It is therefore important that all extensions are visible in the dialog (refer to §2.2 for how to achieve this).

To select a single image or a movie, simply click on the name of the file containing this object. If you prefer, the name of the file may be typed at the File name prompt. If you type in the file name a preview will not be generated automatically, but can be requested by clicking the Preview button. If manually entering the file name, then it is important that you specify the file extension to remove any potential ambiguity.

To select a sequence, the name of the sequence must be typed at the File name prompt, using hashes (#) to indicate the varying numeric part of the file name. Alternatively, click on any member of the sequence and check the Numbers as #### box. This will convert (starting from the right-hand end of the file name) any digits found into the appropriate number of hash characters, thus allowing easy specification of the sequence. However, numbers enclosed in parentheses or square brackets (*i.e.* (..) or [..]) will not be converted to hashes. This allows numeric data to be included unambiguously in the file name. Again, the Preview button may be used to generate a preview if it is not generated automatically.

The precise behaviour of the Compact list check box depends on which version of Windows you are using. The motivation, however, is to provide a more compact way of accessing a large number of numbered images in a given folder. Under Windows XP, the Compact list check box will provide a more compact summary of those present by displaying the name of the first few in a given sequence, and using the compact hash notation to summarise the rest. An example of this is given in figure 8, where figure 8a shows all the files (without Compact list checked) and figure 8b shows how the number of files visible is decreased and sequences are replaced by hashes in the file name when the Compact list box is checked.. Selecting the summary containing hashes is equivalent to selecting the entire series. (Note that clicking on Compact list will retain the files specified at the Object name prompt, but remove any selection in the view window.)

Unfortunately, this simple Compact list option is not available under Windows Vista or Windows 7. Instead, checking Compact list searches for .dfa DigiFlow archive files (see §3.5) and displays only them. Provided all the images and sequences have been created with the archive facility enabled, then the net effect is very similar. An example of this is shown in figure 9.

Whichever version of Windows you are using, if you select an image file (rather than a .dfa file) then the .dfa file is read only if the Read .dfa archive box is checked. However, if you select the .dfa file itself, then it will always be read, along with the image or sequence of images.

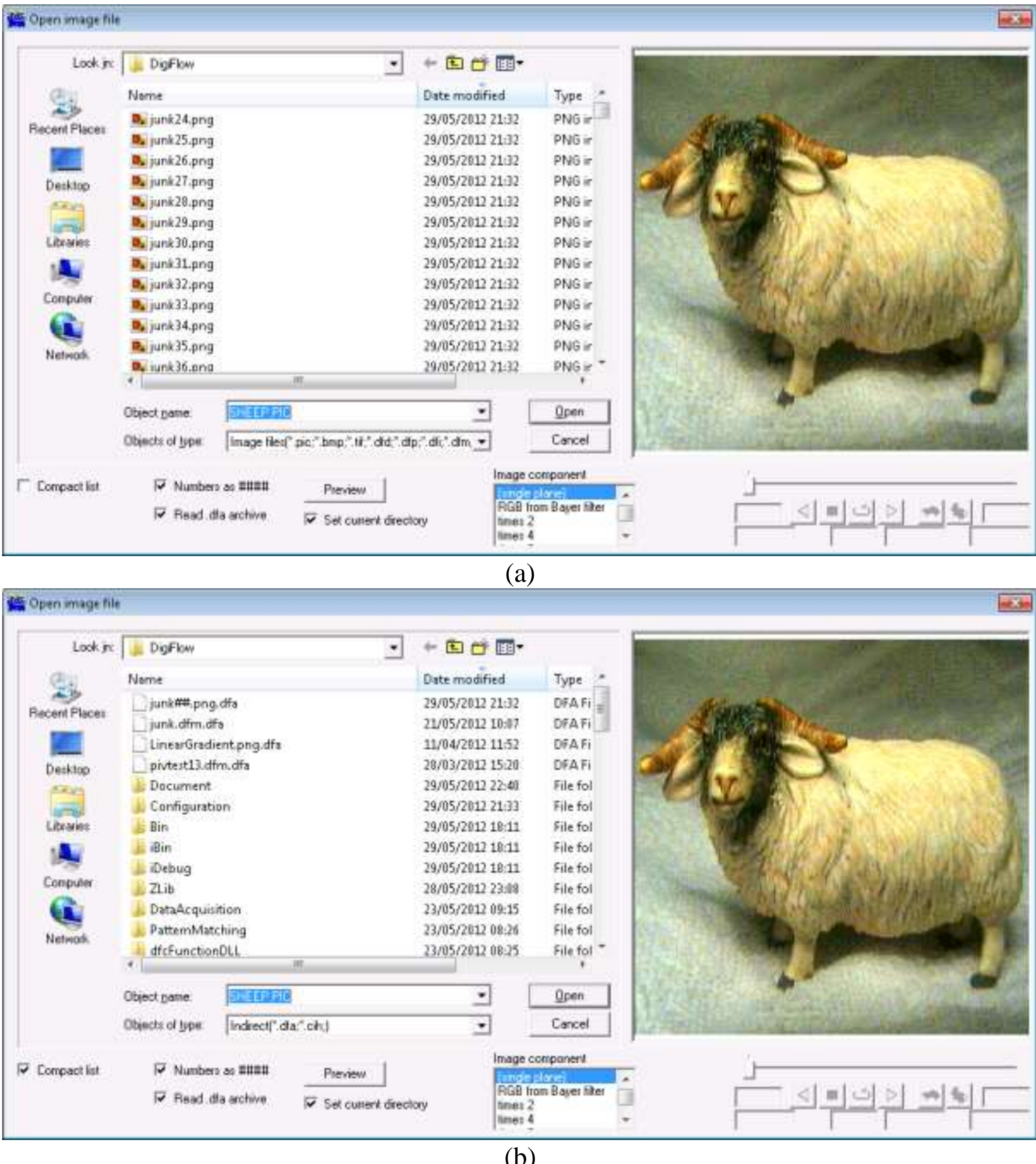


Figure 9: The Open Image dialog box under Windows 7. (a) Showing all files and (b) using Compact List option. Note that only thos files for which a .dfa archive file was generated will be shown when the Compact list box is checked.

Note that the default settings of the Number as ####, Compact list and Read .dfa archive check boxes is remembered from one invocation of the dialog to the next.

A collection of images may be specified using the mouse in combination with the <shift> key to select a range of files, or the ctrl keys to select or deselect individual files. Alternatively, the names may be typed at the File name prompt, each name enclosed by double quotation marks. The collection is sorted into alphabetical order for display and processing. (If a collection is specified in this manner then any hash characters will be interpreted as hashes. Similarly, checking Number as #### will be ignored.) In general, a sequence is preferable to a collection as it offers a greater level of control.

A collection of images may also be selected using wildcards. This may be achieved in two ways. If you use the standard Windows wild cards (? to represent a single character, and * to represent a variable number of characters) then the dialog will display only those files that fit the description; you may then select them in the normal manner. Alternatively, you may use % in place of ? and $ in place of * to do the selection directly. For example, typing Sheep*.* will cause the dialog to display sheep2.tif, sheep.bmp, sheep.jpg, sheep.pic and sheep.tif to be displayed in the dialog box, which may then be selected using the mouse and shift key. Alternatively, Sheep$.$ will achieve the same result, selecting all five files.

If the selected image contains true colour, then the Colour component list box is enabled. This list box allows selection of whether the image is to be treated as full colour, or how the colour information is converted to a greyscale for processing by DigiFlow. For example, selecting RGB will allow DigiFlow to process the red, green and blue image planes separately (where this makes sense), while green will take the green component of the colour image and treat it as a greyscale image, or hue will process the colour using a hue/saturation/intensity representation of the image. The options greyscale and mean all produce a similar effect, although precise details of how the resulting image is constructed from the red, green and blue components differs. The table below gives the relationships.

Key	**Returns**	**Comments**
RGB	Three colour planes	Full colour image
Mono	0.11**red* + 0.59**green* + 0.30**blue*	Same as grey.
Red	*red*	Red component only.
Green	*green*	Green component only.
Blue	*blue*	Blue component only.
hue		Image hue (colour)
saturation		Image saturation (purity)
intensity		Image intensity (brightness)
cyan	1 – *red*	
magenta	1 – *green*	
yellow	1 – *blue*	
grey	0.11**red* + 0.59**green* + 0.30**blue*	Same as mono.
mean	(*red* + *green* + *blue*)/3	Mean of three components.
max	max(*red*, *green*, *blue*)	The brightest component.
min	min(*red*, *green*, *blue*)	The darkest component.

An image containing only a single plane of data may contain colour information if captured from a camera fitted with a Bayer colour mosaic filter. To provide support for this and since it is unlikely that the image file will contain information that DigiFlow can use to automatically detect such an image, when DigiFlow detects a single plane of data in the image to be opened it provides the following supporting options:

Key	**Returns**	**Comments**
(single plane)	*P*	Standard image
RGB from Bayer filter	Three colour planes	Interpret as a full colour image using a standard Bayer filter layout
times 2	2**P*	Intensities rescaled
times 4	4**P*	Intensities rescaled
times 8	8**P*	Intensities rescaled
times 16	16**P*	Intensities rescaled

times 32	32*P	Intensities rescaled
times 64	64*P	Intensities rescaled
div 2	$P/2$	Intensities rescaled
div 4	$P/4$	Intensities rescaled
div 8	$P/8$	Intensities rescaled
div 16	$P/16$	Intensities rescaled

4.2 Save Image As

The Save Image As dialog is essentially the same as the Open Image dialog (§4.1), but is produced when the name of the output image selector (§3.1) is required.

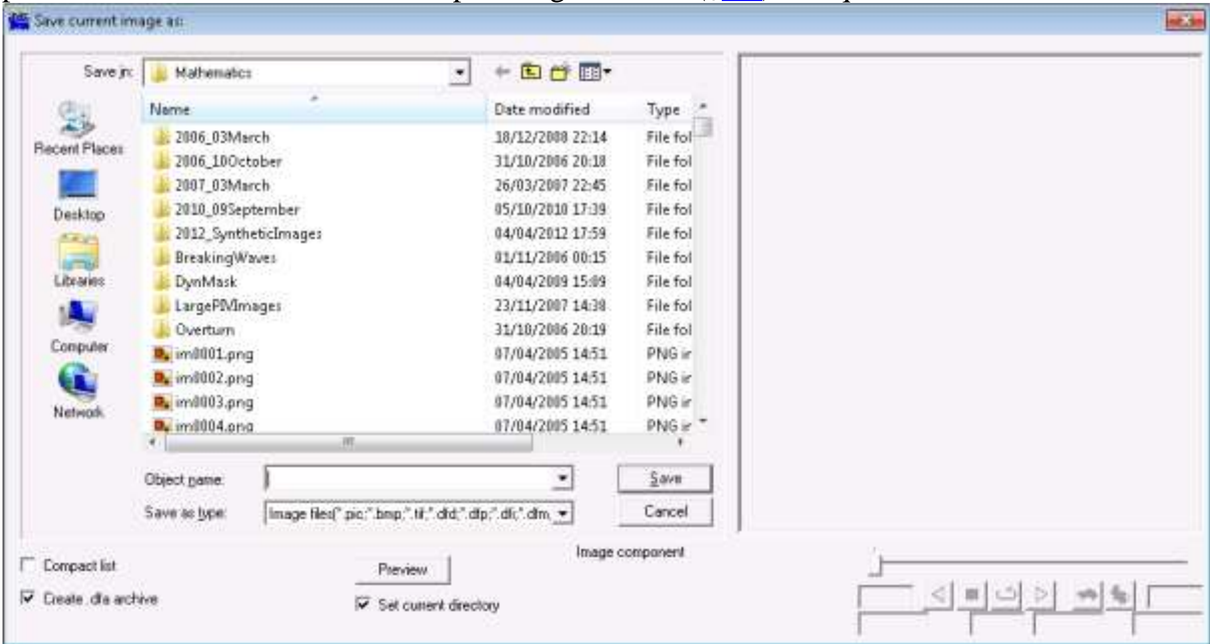


Figure 10: The Save Image As dialog box.

If an image selector of the same name does not exist already, then the file name must be entered by typing at the File name prompt. The extension to be used should be specified explicitly as DigiFlow uses this to determine the file type to be created. It is therefore important that all extensions are visible in the dialog (refer to §2.2 for how to achieve this). Simply selecting a type from the Save as type list will not necessarily have the desired effect if more than one possible type is indicated.

Note that some file types have a range of options such as bit depth and compression. These are normally controlled from outside the Save Image As dialog box using the Options… button in the parent dialog. Refer to §4.4 for further details.

DigiFlow supports a range of industry standard image formats, plus some special formats. The special formats both provide compatibility with the earlier DigImage system, and provide facilities (*e.g.* floating point data representation) not found in industry-standard formats. These non-standard formats are described in more detail in §11.2 (DigiFlow drawing format) and §12 (DigiFlow image file formats).

The Compact list check box operates in the same way as for the Open Image dialog described in §4.1. Here, the Create .dfa archive check box replaces the Read .dfa archive and causes DigiFlow to create a .dfa archive for the output it produces (see §3.5).

4.3 Sifting input streams

When processing an image stream it is often desirable to select only a subset of the stream for processing. This subset may contain only some of the images from the stream, and/or it may contain only part of each image. Within DigiFlow this process of selecting a specific part of an image stream for processing is referred to as ‘sifting’. When sifting is available, the corresponding dialog will have a Sift… button (typically one for each input selector) that starts a tabbed dialog box controlling the sifting process. The following subsections describe the various sifting options.

4.3.1 Selector timing

The Selector Timing tab of the Sift dialog allows the user to specify which times from a multi-image image selector (§3.4) will be used for a process.

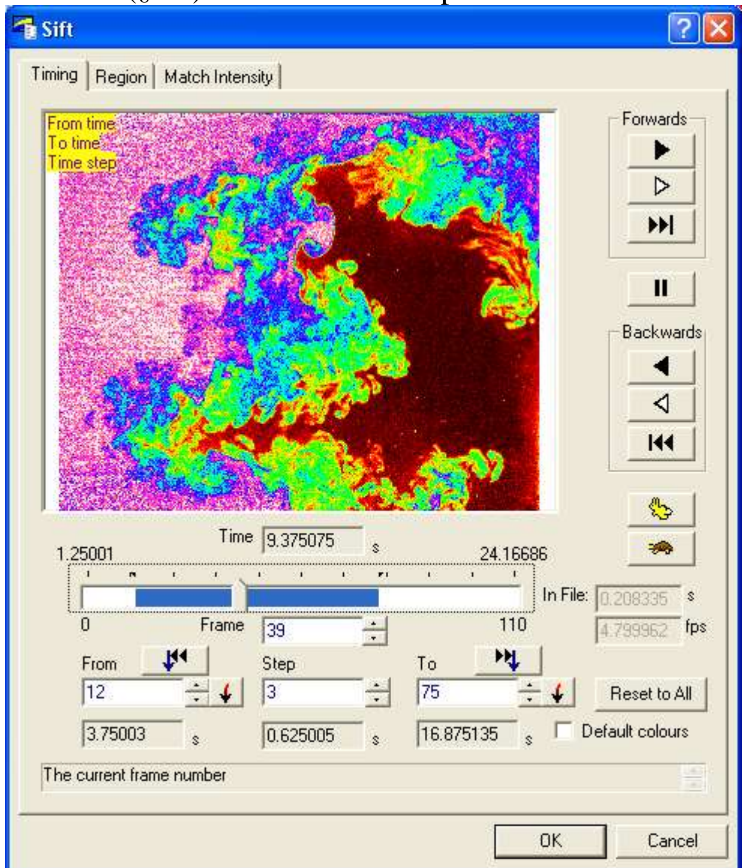


Figure 11: The standard Selector Timing tab of the Sift dialog.

This tab allows the preview of the image selector and specification of the processing start and end points as well as the step between the images to be processed.

The buttons down the right-hand side allow the image selector to be played, the speed of this preview controlled by the hare and tortoise buttons. The slider allows the currently visible frame to be dragged to any time. The Frame edit box and spin control allow more precise movement of the preview frame. The and buttons move to the currently specified limits for the processing.

The frame numbers for the start and end points may be typed in the From and To edit boxes, and the spacing in the Step edit box. The corresponding time boxes below will be updated automatically.

Clicking the buttons adjacent to the From or To edit boxes will set the corresponding from or to position to the current position, shown by the slider and the edit boxes immediately above (time) and below (frame).

Alternatively (but less precisely), holding <shift> while dragging the slider will allow specification of the timings.

When the From and To times are set, or Step is not unity, then this information is displayed on a yellow background at the top of the image preview.

For files that do not store timing information, the DigiFlow assumes by default that the files are separated in time by one second. This may be changed using In file, in which the image spacing may be specified in either seconds or, using the lower of the two controls, in frames per second. These two controls are disabled for files that store time information, but display the relevant details.

Reset to All resets the start and end points to include the entire selector.

Checking the Default colours control will cause the preview image to be displayed using the DigiFlow default colour scheme rather than the colour scheme stored in the image file.

4.3.2 Selector region

The Selector Region tab of the Sift dialog allows the user to specify a region within an image selector (§3.4) that will be used for a process.

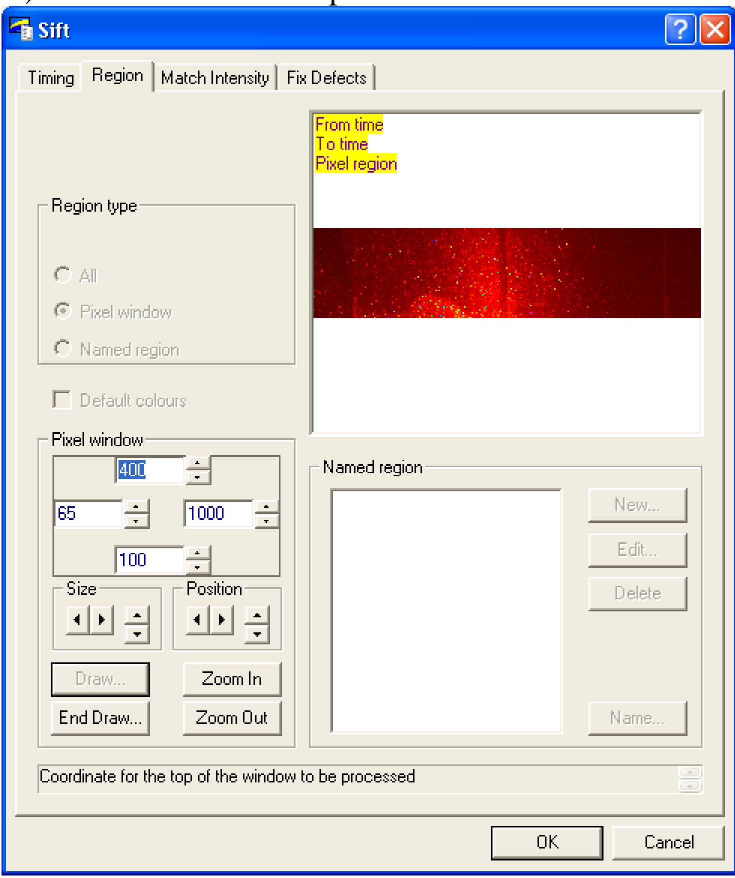


Figure 12: The standard Selector Region tab of the Sift dialog.

For a process requiring more than one input stream (and hence having more than one image selector in its dialog box), one of the streams (typically the first in the dialog box) will be the *master* stream. If the region for this stream is changed, then the region for the other (slave) streams will be changed automatically to *conform* to (typically made the same as) that for the master stream. It remains possible, however, to change independently the region for the slave selectors, provided the size of the region for the slave selector is compatible with that for the master selector.

The type of region is selected by the Region type group of radio buttons. The example shown in figure 12 is for a master selector; the Conform option is not available here, but would be visible above All when sifting slave selectors.

If Pixel window is selected, the pixel coordinates of the left, right, top and bottom of the window may be specified in the edit controls within the black rectangle. If preferred, the size may be increased without shifting the centre of the region, or the location of the region may be changed without adjusting the size, using the Size and Position controls, respectively.

Alternatively, clicking the Draw button opens a full size window that allows the window to be moved and resized dynamically using the mouse (see figure 13). (Hint: it is sometimes worth dragging a corner of the window to increase its size and thus make it easier to grab the edge of the region window.) The Zoom In and Zoom Out buttons may be used to control the magnification while drawing. Similarly, you may swap between this window and the Sift dialog box to use the various edit and spin controls to move the region around. Click on the End Draw… button to close the drawing window and re-enable the other controls on the Sift dialog.

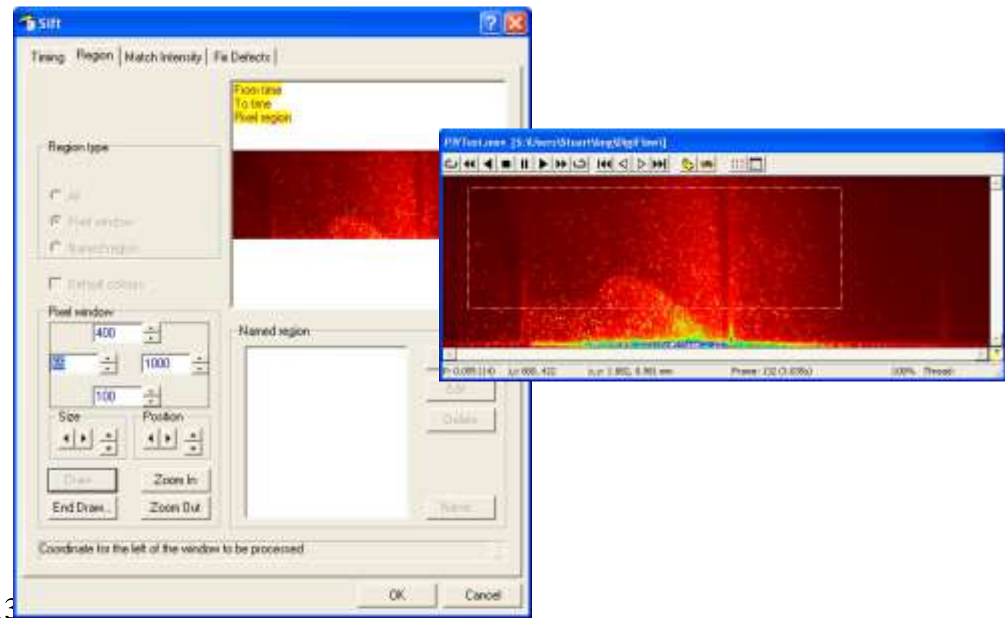


Figure 13

If Named region is selected, then previously saved regions are displayed and may be selected. This provides a convenient method of using the same region in a range of different processes. The four buttons to the right of the list box may be used to manage these named regions. New named regions may be created either by clicking the New button, in which case a subdialog is produced to allow specification of the region, or by clicking the Name button (when Pixel window is selected) to give a name to a pixel window. The Edit button allows alteration of an existing window, while Delete removes the region from the list. Note that selecting a named region that is a Pixel window will update the controls in the Pixel window group. Switching back to Pixel window allows editing of these values, while Name may be used to overwrite the old values with the new ones, or to create a copy.

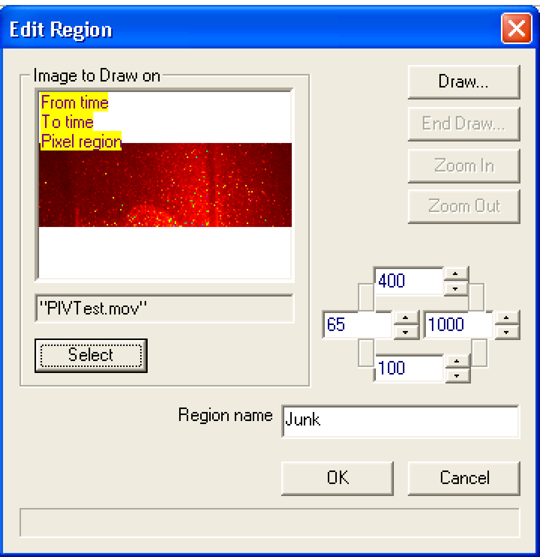


Figure 14: Editing a region.

4.3.3 Matching intensities

Quantitative measurements often require that the intensities are matched between different frames and sequences. The intensities of the raw image streams may fluctuate due to a number of reasons. One common one is the mismatch in frequencies between the illumination and the camera frame rate. Depending on the type of light source and the shutter speed of the camera, this mismatch may lead to a modulation of nearly 50% of the signal amplitude, while automatic gain features can lead to similar results. While it is in general best to avoid these problems by using continuous or high frequency light sources, this is not always practical.

The Match Intensity tab in the Sift dialog (figure 15) provides a basic mechanism for correcting the intensities of input image streams to match them to some fixed reference. The basic strategy is for the image to contain two reference regions that contain approximately uniform intensities that should not change with time. These two regions are then used to generate a linear mapping between the input image and a reference intensity, thereby adjusting the intensities in preparation for processing.

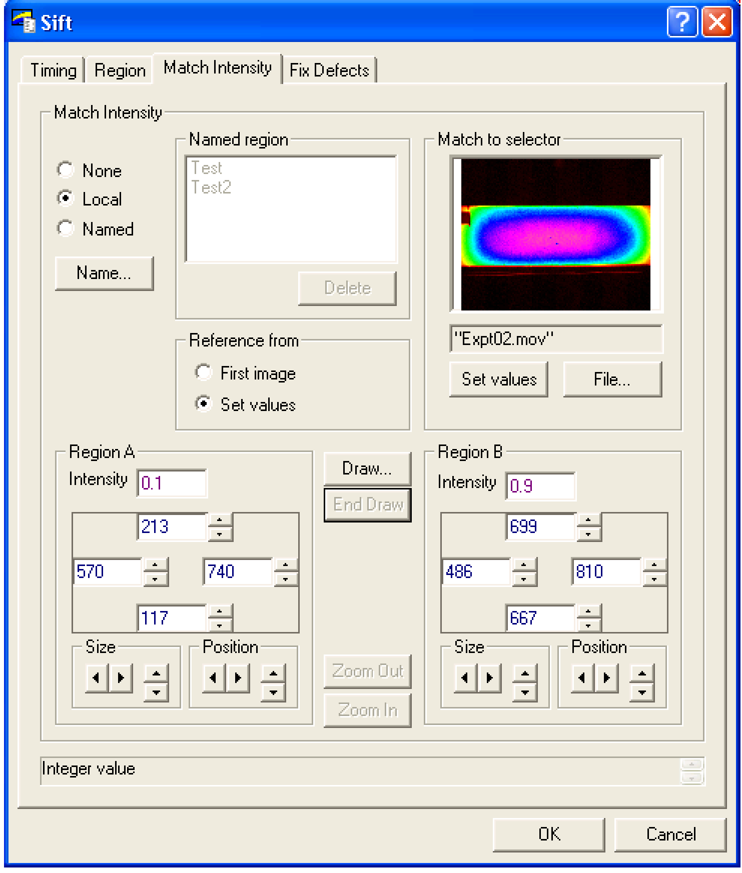


Figure 15: The Match Intensities tab provides the ability to directly relate an image to reference values.

The Match Intensity facility is turned on and off using the radio button group in the top-left; .when off (None), then the intensities are read without alteration. The Match Intensity facility can be enabled either using details provided locally (Local), or with details saved previously (Named), in a similar manner to that used for Regions.

A locally defined Match Intensity reference consists of a pair of rectangular regions, Region A and Region B. The location and size of these regions is controlled by a variety of controls for specifying the left, right, top and bottom of each of the rectangles. Additionally, as with the Regions dialog, the regions may be drawn on an image and dragged to their desired location by clicking the Draw… button (see figure 16).

Each region requires an intensity to be associated with it. When Reference from is set to Values, then the Intensity controls in the Region A and Region B groups is enabled. The user may directly enter the desired (target) reference here, or by using File in Match to selector to select a suitable image, then the Match button will read the intensities from the specified image. Alternatively, if Reference from is set to First image, then the reference intensities are not entered at this point, but rather they are determined automatically from the first image in the stream to be processed.

Once the various controls for a Local Match Intensity have been set, their values may be saved for use elsewhere by clicking Name…. This prompts for a user-supplied descriptive name, saves the settings, and switches the dialog into Named mode.

Selecting an entry from Named matches loads the corresponding settings for use. If you wish to alter the settings of a saved match, load it by selecting from the list, then switch to Local mode. Make any necessary changes, then click again on Name to name and save it (you may re-use an existing name).

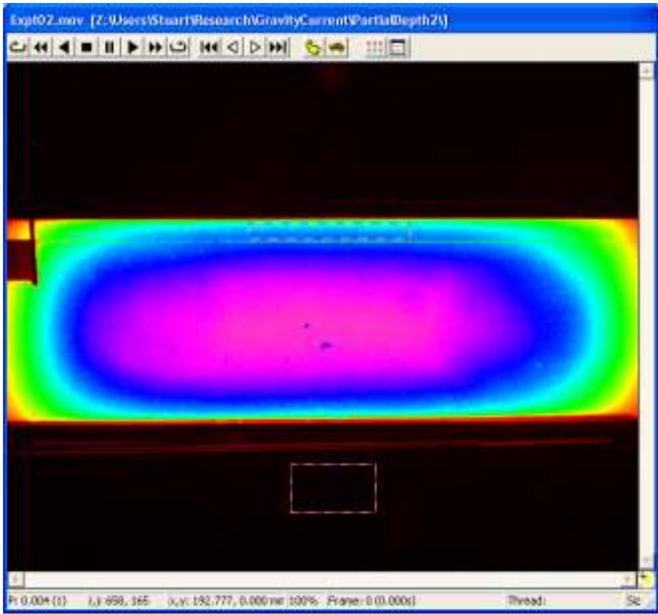


Figure 16: Drawing regions for intensity matching.

4.4 Modifying output streams

This section describes the various modifications that may be made to the output streams. These modifications are accessed via the Options… button in the output stream select group. The precise contents of this dialog will vary depending on the output file type that has been selected.

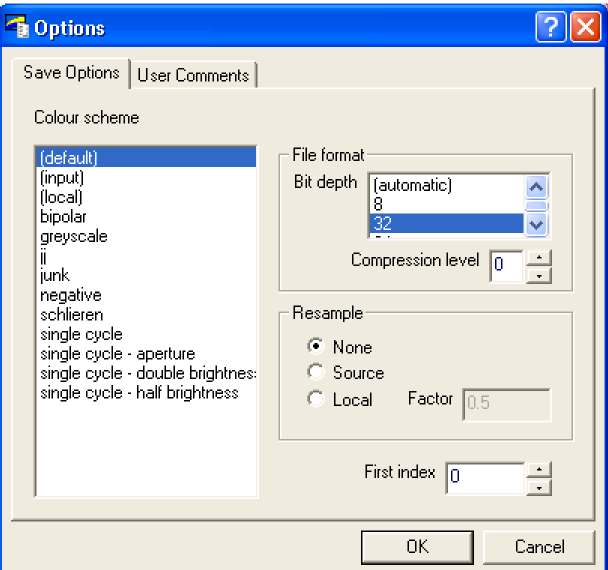


Figure 17: The Save Options dialog.

4.4.1 Setting output stream colour

The colour scheme for the output stream is selected from the list of known colour schemes in the Colour scheme list box. Selecting the (input) member will set the colour scheme to be the same as for the master input stream.

If you wish to add a new colour scheme or modify an existing scheme, you must use the View: Colour Scheme… menu option. Refer to §5.3.6 for further details.

4.4.2 Full colour

For output formats such as .bmp, .png or .jpg that support true colour images, a Full colour checkbox is produced. If checked, then the output is saved as in a 24 bit true colour format. If not checked, then a greyscale version of the output is saved, along with the selected false colour map (the false colour map is not saved for .jpg files).

4.4.3 File format

The File format group invokes various options that may exist for the specified file type. The contents of this group will depend on the file type specified: in many cases there are no options and so the group is left empty.

The Bit depth field determines the number of significant bits saved for each pixel in the image. Most image formats use 8 bits, but for high resolution images, or images that result from numerical computations, a greater depth may be desired. If the .dfi format is specified for the file type, then bit depths of 8, 32 and 64 bits are possible.

When available, the Compression level edit and spin control will determine whether or not the image is to be compressed using a lossless compression. A value of zero indicates no compression, with positive integers giving various levels of compression. Typically compressing an image reduces its size by around a factor of two, but at the cost of slower access (although for a very slow hard disk the access speed may improve with compression). The additional time taken to compress an image will depend in part on the level of compression requested, and in part on the structure of the image. If a process seems particularly slow, but still producing the correct answer, try reducing the level of compression.

In the case of an .avi file, selecting zero causes full, uncompressed images to be saved, whereas setting Compression level to 1 will use the Cinepak compression (installed by default with Windows). For other compressions specify a value of 2 for Compression level which will then cause the standard Windows Video Compression dialog to be produced when DigiFlow is ready to save the first frame of the output stream. (Note that most of the .avi compression options are 'lossy' in the sense that only an approximation to each image is saved.)

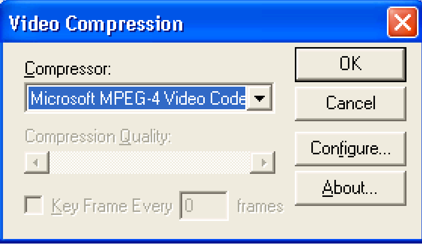


Figure 18: Standard Windows Video Compression dialog.

For .jpg images, the compression applied is 'lossy'. The higher the level of compression, the greater the fraction of information lost. This is controlled by the Quality control. Note that in general the lossy nature of the compression in .jpg images means that they should not be used for the storage of intermediate results.

4.4.4 First index

By default, the first image in a sequence produced by DigiFlow will be given a zero index (numerical part of the file name). The First index control may be used to change the index for this first image. In either case, subsequent images will always be produced with unit increments from this value.

4.4.5 Resampling

When the .dfi image format is selected, it is possible to rescale the output stream before it is saved and then reverse this rescaling when the image is subsequently read in. Typically this

option is used to reduce the resolution of the saved image, but maintain its size by interpolating back to the original size before using the image again.

This feature is enabled using the Resample check box. When enabled, the resolution of the saved image is controlled by the Factor edit control which accepts a floating point value for the relative resolution of the saved image. For example, a value of 0.5 will cause the saved image to have only ¼ of the number of pixels of the original in the file, but through interpolation the missing pixels are reconstructed when the image is read in again. This option is particularly valuable for use with images produced by the synthetic schlieren (§5.6.4.3) and PIV (§5.6.5.2) facilities.

4.4.6 Save user comments

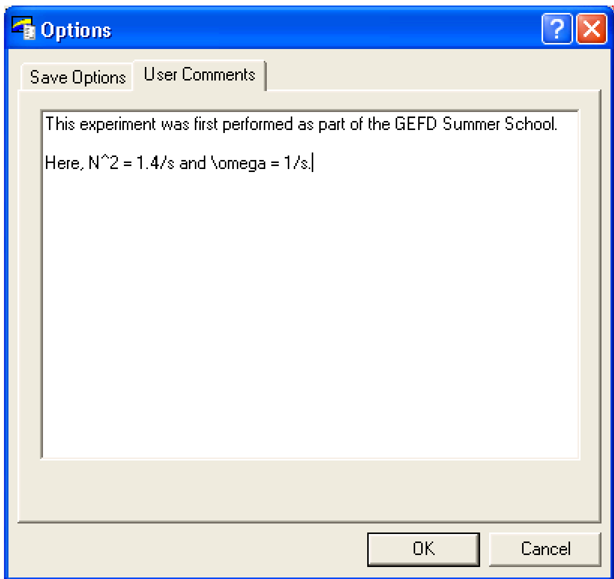


Figure 19: User comments tab.

Some file formats (*e.g.* .dfi, .dfd and .dft) allow user comments to be saved along with the images. These comments are specified using the User Comments tab of the Save Options dialog.

4.4.7 Encapsulated PostScript streams

DigiFlow can produce Encapsulated PostScript (.eps) output either using the Export to EPS option in the File menu (see §5.1.12) or by specifying an .eps file as the output stream. In the latter case the normal Options dialog has an additional EPS button that invokes the dialog shown in figure 20.

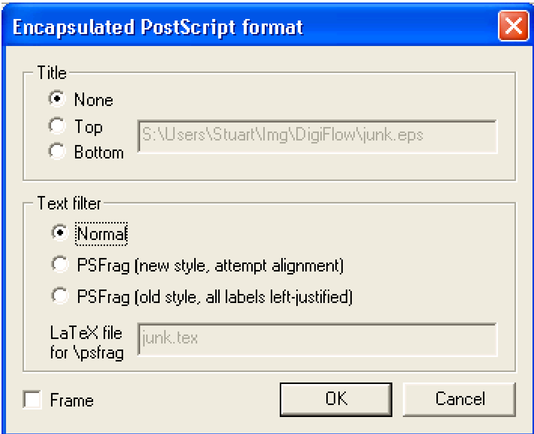


Figure 20: The output options for Encapsulated PostScript (.eps) files.

The PostScript options provide the ability to add a title either above (Top) or below (Bottom) the image or graphic output, and to add a frame (Frame) around the output. DigiFlow (commercial version only) provides support for the LaTeX \psfrag macro package. This enables the text produced by DigiFlow to be readily replaced with text generated by LaTeX, thus keeping font and style information consistenet and allowing post-plotting adjustment of the text labels, *etc*. Selecting Normal produces the eps containing the original labels, whereas with either of the PSFrag options the text is replaced by a unique character for each element. At the same time, DigiFlow creates a .tex file that contains the mapping between these characters and the original text. This .tex file can then be embedded in included in the main LaTeX document to reproduce the figure. See §5.1.12 for further information on the Encapsulated PostScript formatting options.

4.5 dfc Help

As will be seen in §5, a large part of DigiFlow's power and flexibility is gained by the use of user-supplied macro code. This code is known as dfc code. Examples of facilities that require such code include Analyse: Time: Extract (§5.6.1.5), Analyse: Time: Summarise (§5.6.1.6), Tools: Transform Intensity (§5.7.2) and Tools: Combine Images (§5.7.3). Details of the macro code itself are given in §§8 and 9. However, this manual gives only a relatively brief introduction to a subset of the dfc functions available within DigiFlow. Instead, the bulk of the documentation is provided within an interactive help facility available from within DigiFlow itself in the Help: dfc Functions menu item, and from the fx? button within dialogs where such information is of value.

The help facility takes the form of the dialog illustrated in figure 21. To find a function performing a given task, simply type some information about that task into the Search for box. For example, if you want to find functions that have something to do with drawing, enter "`draw`". You will notice that as you enter "`draw`", the Look up list changes as each letter is typed. When you type the "`d`", the size of the items in the list is reduced so that it only includes those with a "`d`" somewhere in their names. Similarly, "`dr`" leads to a further reduction, excluding those that do not have this pattern, and so on.

Spaces in the Search for box are interpreted as "and" criteria for the search. For example, entering "`dr ma`" would reduce the list to those functions with both "`dr`" and "`ma`" in their names, but without the two patterns needing to be adjacent. This, combined with the logical and descriptive (if somewhat verbose) naming conventions for DigiFlow functions, provides a very powerful search facility.

At all stages the Look up list is sorted alphabetically. (Note that if Search for is left blank, then Look up contains all possible functions.)

Selecting an item in the Look up list then brings up the documentation for the function in the three boxes below. The top of these identifies the role played by the entry within dfc code. The list box below gives the range of possible entry points to the function. As we shall see later, many DigiFlow functions are "overloaded" (*i.e.* they accept more than one type of data), and may have optional parameters. This list itemises the full range of possibilities. Selecting an entry point from this list and clicking the Copy button copies this entry point into the clipboard.

The bottom control on the dialog provides the detailed documentation for the selected function. This documentation should be read in conjunction with the entry point documentation. The help system is hyperlinked (*e.g.* **`draw_start`**`(..)` in figure 21): clicking on a hyperlink will take you to the corresponding help. Similarly, backward (⇦) and forward (⇨) buttons will move through previously selected hyperlinks.

The General items list box provides access to more general information, such as modifiers for input and output streams, recent changes to DigiFlow, how to return images from code specified for tools such as Transform intenstiy, and how to produce simple plots.

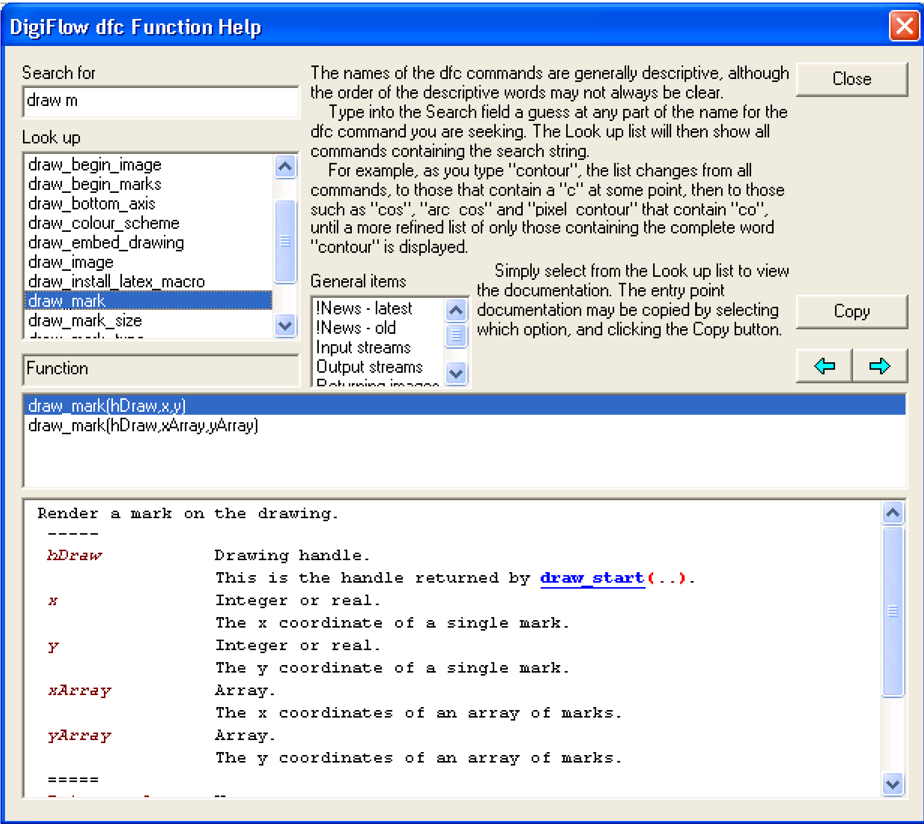


Figure 21: The help dialog for dfc code.

The help facility may also be started from within a code edit box by right-clicking. Doing so will cause the word under the cursor to be pre-loaded into Search for field. Moreover, if that word is a known DigiFlow command, the details will be looked up automatically.

4.6 Code library

DigiFlow incorporates a number of features that will facilitate the re-use of the dfc code used in facilities such as Analyse: Time: Extract (§5.6.1.5), Analyse: Time: Summarise (§5.6.1.6), Tools: Transform Intensity (§5.7.2) and Tools: Combine Images (§5.7.3). This section describes the DigiFlow Code Library. Details of the macro code itself are given in §§8 and 9.

The dfc Code Library provides convenient method of storing and retrieving user-developed code. The library itself is stored in a file named DigiFlow_Library.dfs in the directory in which DigiFlow is started. Note that this file is re-read from the current directory every time the Code Library is invoked. The DigiFlow_Library.dfs file may be copied from one directory to another, if the user desires.

The library is accessed via the Code Library button in appropriate dialogs. Central to the Code Library dialog, shown in figure 22, is the Entry list that itemises all previously saved items of code for this DigiFlow facility (a separate list is maintained within the same file for each different facility). Any code currently specified in the parent dialog box is recorded under the _current key; this will be the default selection upon entry.

To retrieve a previously stored code item, simply select it from the Entry list and click OK to insert it in the parent dialog. The Code edit box will show the code, while Description will

show any previously saved description. Clicking Cancel will return to the parent dialog without changing the code in that dialog.

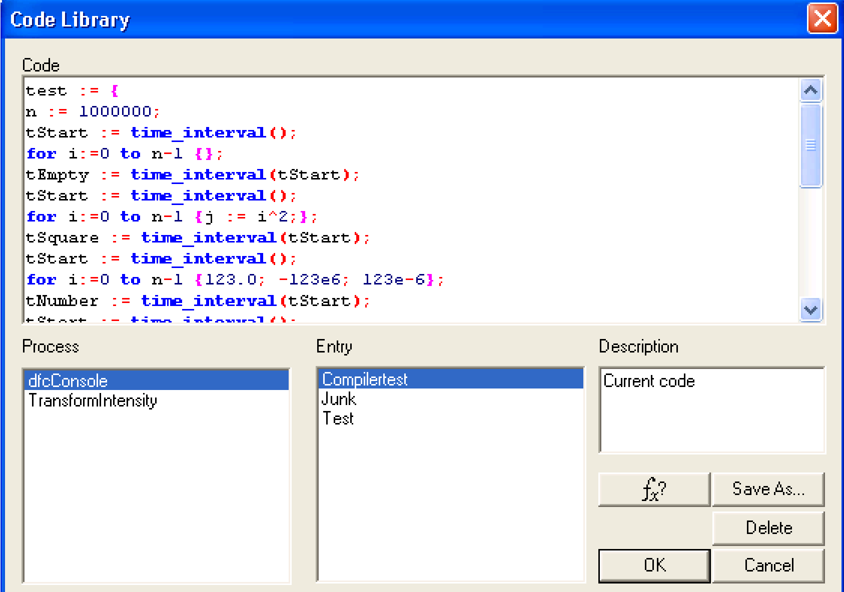


Figure 22: The code library dialog.

The Code and Description may be edited before returning to the parent dialog. The Process list allows code to be selected from different processes. The Delete button may be used to remove an entry from the Code Library, and the *fx?* button gains access to the dfc Help facility. Finally, the Save As button allows code to be saved into the data base (see figure 23) under any of the processes.

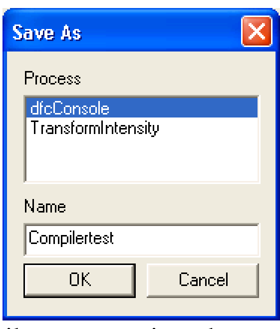


Figure 23: Name under which a Code Library entry is to be saved.

5 Menus

This section describes the main menu options. Some of these will be familiar as they follow standard Windows conventions, whereas others are specific to DigiFlow. Many of these menu options can be strung together to create processes that are more complex. Details of how to achieve this are given in §6.

5.1 File

5.1.1 Open Image

Toolbutton:
Shortcut: ctrl+O
Related commands: open_image(..), read_image(..), read_image_details(..), view(..)

Allows an image selector (§3.4) to be opened for viewing. The image is selected through the Open Image dialog box (§4.1). Both images and drawing formats may be opened. Encapsulated PostScript (.eps) may also be opened if DigiFlow is able to find an installed copy of GhostScript (see §2.2.2).

5.1.2 Run Code

Toolbutton:
Shortcut: ctrl+R
Related commands: include(..)

Opens and runs a DigiFlow dfc macro. Refer to §10 for further details.

5.1.3 Run Macro

Toolbutton: ctrl+shift+R
Shortcut:
Related commands:

Opens and runs a DigiFlow dfc macro from the Macros subfolder within the folder where DigiFlow is installed. This folder contains various macros and wizards that are of general value. Refer to §10 for further details.

5.1.4 Save As

Toolbutton:
Shortcut: ctrl+S
Related commands: save_image(..), write_image(..)

This option allows the contents of the active window to be saved. Note that if the active window contains a sequence or other collection of images, only the currently displayed image will be saved. To copy an entire sequence use File Edit stream (see §5.1.6) or one of the related transformation tools.

5.1.5 Live Video

5.1.5.1 Show Live Video

Toolbutton:
Shortcut:
Related commands: process File_ShowLiveVideo(..), camera_live_view(..)

This option creates a new window and streams live video directly to it. Whilst the live view is intended primarily for previewing camera output, it may be used in conjunction with macros such as **camera_grab**(..) to acquire single or multiple frames, or with

camera_capture_file(..), camera_start_capture and camera_stop_capture(..) to acquire entire sequences.

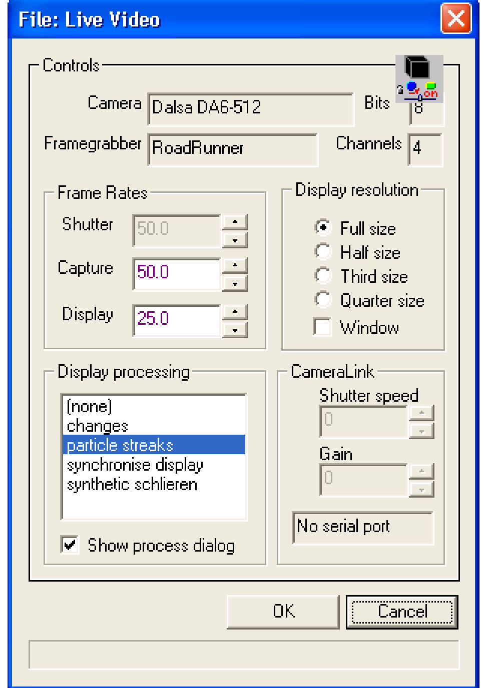


Figure 24: The dialog controlling what is seen in a live video window.

Video captured using this option is fed continuously to the display until stopped by the user; it is not saved to hard disk (except via the use of dfc code). For this reason, no duration can be specified. For some cameras, it is possible to set the shutter speed (Shutter, in frames per second; n frames per second is equivalent to a shutter speed of $1/n$ seconds) independently of the acquisition rate (set by Capture, in frames per second). However, many digital cameras force the two rates to be equal. For some supported CameraLink cameras, the Shutter speed and camera Gain can be set as integer indices into the range of possible values. The meaning and acceptable range of values varies between different makes and models of cameras. (Note that a value of zero indicates unit gain on some cameras, but on others, such as the UniqVision UP1830CL, unit gain corresponds to a value of 128. The default value is obtained from the entry for a specific camera in DigiFlow_Cameras.dfc.)

The frame rate for updating the display is independent of the shutter and capture rates. Typically Display is set to a lower frame rate (there is little point exceeding around 12 frames per second). Any necessary processing of the incoming data stream to correct the format is undertaken automatically.

The Display resolution group controls how much of the original image being captured by the framegrabber will be displayed on the screen (note that this does not affect the data available through dfc functions such as camera_grab(..)). The meaning of the various options is self-explanatory.

In some cases, simple real-time processing of the image prior to display will greatly assist with the setting up and running of the experiment. The Display processing group controls the type of processing that will be done. These are described in more detail for the Capture Video option in §5.1.5.2. To suppress processing, select none from the list box. For large images, it may be best to use Display resolution to reduce the resolution and thus the computational burden of undertaking any processing.

The Tools: Slave Process family of functions (see §5.7.5) provides a convenient way of accessing an even broader range of additional functionality, ranging from focusing tools to real-time optical flow calculation.

Particle Streaks

If Display processing is set to particle streaks then the dialog shown in figure 25 is displayed to provide processing of display output while at the same time capturing the raw video to a file. There are four display options: Threshold, Maximum, Minimum and Direct.

For Threshold the incoming image is segmented into *particles* (bright points) and *background* (dark points) by varying the Threshold control. Using relatively simple processing, the particles so identified may be converted into comet-like streaks that slowly fade with time. The length (in time) of these streaks is determined by the Length control.

The Maximum and Minimum options work in a similar way except that rather than segmenting the incoming image, the brighter (Maximum) or dimmer (Minimum) of the incoming and stored images is used. Again the length of the streaks can be set using the Length control.

When Direct is selected, then the incoming images are displayed without any further processing.

The Reset button clears the display of all earlier times.

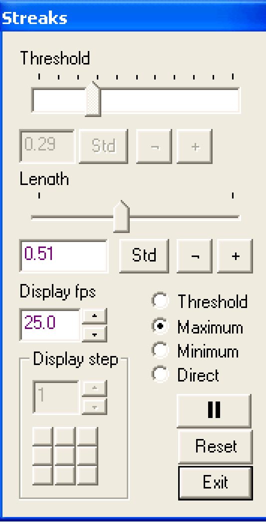


Figure 25: Particle streaks preview dialog.

The rate at which the screen is refreshed is set by Display fps. Note, however, that the desired rate may not be achieved if the computational load is too great.

The Display step group of controls is used to aid the viewing of very large images that may be larger than the available display area. The edit and spin controls set the step between displayed pixels (hence a value of 2, for example, will give a half-resolution image). The grid of buttons in the bottom left allows the view port into a larger image to be moved around in a manner that is efficient to display. These options are only enabled if the Window option in the Live Video dialog is checked.

Synthetic Schlieren

If Display processing is set to synthetic schlieren then the dialog shown in figure 26 is displayed to provide processing of display output while at the same time capturing the raw video to a file. This allows real time visualisation of a synthetic schlieren image (see §5.6.4). There are three processing options: Difference is the simplest (and computationally fastest) technique that provides a qualitative output proportional to the magnitude of the gradient in

the density perturbation. The Horizontal gradient and Vertical gradient options perform more a more sophisticated analysis that returns a semi-quantitative output of the specified component of the gradient in the density perturbation. Note that these two options distinguish between positive and negative gradients.

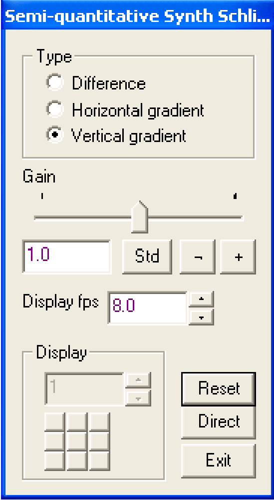


Figure 26: Synthetic schlieren preview dialog.

The Gain control determines the relationship between the gradient and the intensity of the display. The Reset button forces the reference image to be recaptured.

The rate at which the screen is refreshed is set by Display fps. Note, however, that the desired rate may not be achieved if the computational load is too great. The Direct button turns off the streaks processing and displays directly the camera input.

The Display step group of controls is used to aid the viewing of very large images that may be larger than the available display area. The edit and spin controls set the step between displayed pixels (hence a value of 2, for example, will give a half-resolution image). The grid of buttons in the bottom left allows the view port into a larger image to be moved around in a manner that is efficient to display. These options are only enabled if the Window option in the Live Video dialog is checked.

5.1.5.2 Capture Video

Toolbutton:

Shortcut:

Related commands: `process File_CaptureVideo(..)`, `camera_live_view(..)`, `camera_capture_file(..)`, `camera_start_capture(..)`, `camera_stop_capture(..)`

Using this facility, a video sequence may be captured from one of the digital video cameras supported by DigiFlow.

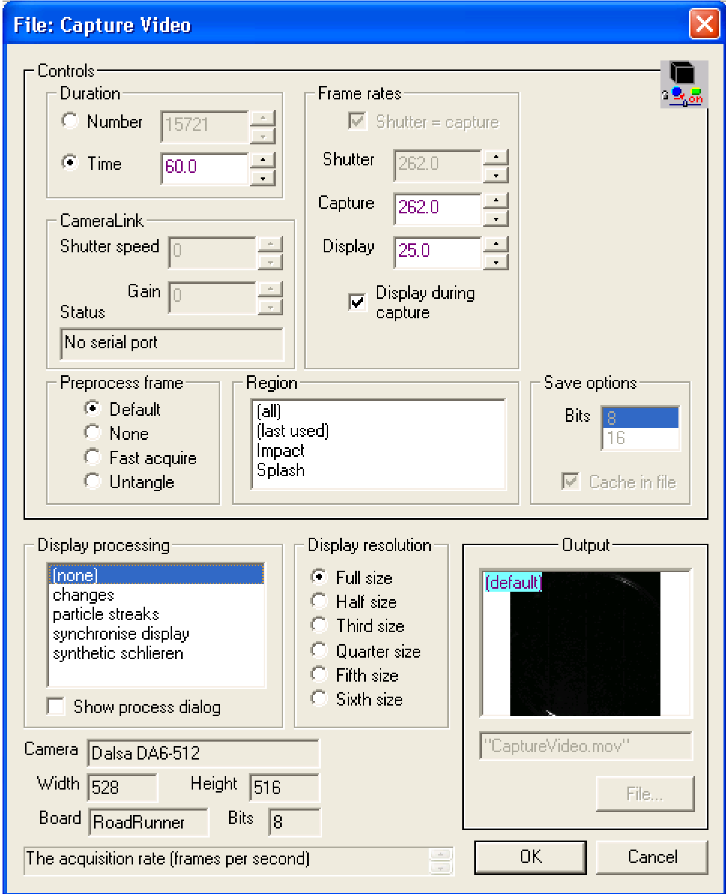


Figure 27: Dialog box controlling the capture of video.

The basic timing for the video sequence is controlled by a combination of the Duration and Frame Rates groups. The first of these sets the length of the sequence, either as a specified number of frames (if Number selected), or as time in seconds (Time selected). For some cameras, it is possible to set the shutter speed (Shutter, in frames per second; n frames per second is equivalent to a shutter speed of $1/n$ seconds) independently of the acquisition rate (set by Capture, in frames per second). By checking Shutter = capture, then the two of these are forced to be equal. For some supported CameraLink cameras, the Shutter speed and camera Gain can be set as integer indices into the range of possible values. The meaning and acceptable range of values varies between different makes and models of cameras. (Note that a value of zero indicates unit gain on some cameras, but on others, such as the UniqVision UP1830CL, unit gain corresponds to a value of 128.)

The frame rate for updating the display is independent of the capture rate. Typically Display is set to a lower frame rate (there is little point exceeding around 12 frames per second). This setting does not affect the data stored to disk. For some systems the bandwidth requirements of displaying the image while acquiring to hard disk exceeds that available. In such cases the Display during capture check box should be cleared, thus suppressing the display during the capture, although the display is still updated before capture begins and after capture finishes.

For some camera and framegrabber combinations, the raw data coming from the framegrabber may not be in the correct format for display. This typically occurs with multi-tap cameras; most single-tap cameras produce data in the correct format and require no additional processing. If additional processing is required, the Preprocess frame group determines what should be done in this situation. The Untangle option forces the data to be untangled before displaying or saving to hard disk. This option is the most processor and memory bandwidth intensive, and so may not function adequately on all systems, especially during the capture

process when much of the bandwidth is already taken up. To overcome this, the Fast acquire option untangles the images before and after the capture process, thus giving an intelligible preview, but turns off the untangling during the capture. DigiFlow will automatically untangle the image subsequently when it reads the image file produced in this manner. The remaining option, None, turns off all processing.

The Display resolution group controls how much of the original image being captured by the framegrabber will be displayed on the screen (note that this does not affect the data stored to disk). The meaning of the various options is self-explanatory.

In some cases, simple real-time processing of the image prior to display will greatly assist with the setting up and running of the experiment. The Display processing group controls the type of processing that will be done. Note that this does not affect the data written to disk. To suppress processing, select none from the list box. For large images, it may be best to use Display resolution to reduce the resolution and thus the computational burden of undertaking any processing.

The Output group provides a standard interface to select the destination for the captured image. However, by default DigiFlow will be configured to always capture to a fixed location (see §13 for details) to avoid the user having to select a disk drive with appropriate characteristics and to force the user to go through a compulsory review process to extract only those parts of the image stream that are of value. If this feature is enabled, then the Output group will be disabled (the file name CaptureVideo.dfm will be visible) and following completion of the sequence capture, the Edit Stream dialog (see §5.1.6) will be started automatically to control this second step.

If you do not want to capture the entire field of view of the camera then you may choose the region you wish to save through the Region list. The region must have been defined within DigiFlow previously (*e.g.* using Edit: Regions; see §5.2.7, or as part of a Sift operation). Note that there is a time overhead in extracting a region from an image. Consequently, although the amount of data to be stored is reduced, the total time taken may be increased in some circumstances. However, if the region to be saved is more than around 20% smaller than the full view, then it could well be worth capturing a more limited region.

After pressing OK, DigiFlow opens an image preview, creates a Display processing dialog (see below) if display processing was selected, and then prompts the user to start the acquisition as illustrated in figure 28.

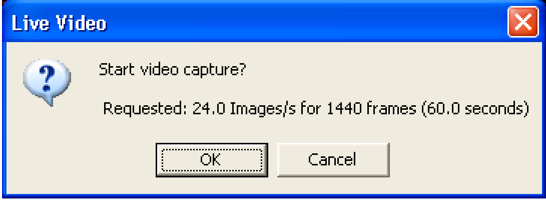


Figure 28: Message box starting the video capture.

After the sequence capture has finished, the performance is displayed in a message box, such as shown in figure 29. The precise text in the message box will depend on whether or not the compulsory review feature is enabled. Note that the 'achieved' timings may not be precise, especially for relatively short sequences.

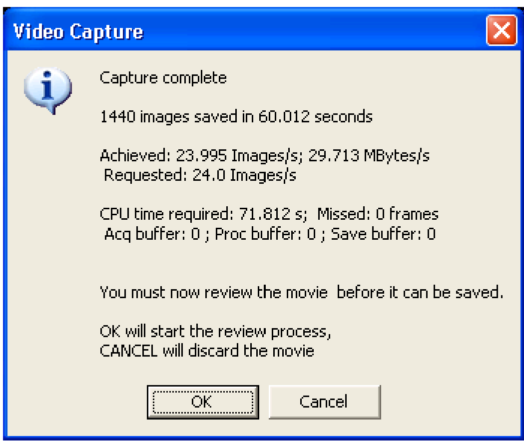


Figure 29: Message box at end of video capture.

If the computer fails to keep up with the bandwidth requirements of the capture process, then this will be indicated by there being some missed frames, and a lower than expected frame rate being achieved. The amount of CPU time required is a strong function of any display processing required. The synthetic schlieren option was selected for the example illustrated in figure 29. This was performed on a 1GHz dual processor machine. Clearly more CPU time was required than the capture time, but each processor was busy only around 60% of the time. A single processor machine, however, would not have managed to keep up with the bandwidth requirements.

The review process will utilise either the File: Edit Streams (see §5.1.6) or File: Camera File (see §5.1.7) dialogs. For most cameras, the former will be used as all that is necessary is to copy across the parts of the captured sequence that are actually needed. However, for some cameras, it can be desirable to impose a flat-field correction during this copy process. Where the DigiFlow camera database suggests this is desirable, DigiFlow will start the File: Camera FIle dialog instead. In either case, if Cancel is selected in response to the dialog (figure 29) at the end of the capture process, you can still access the captured sequence by manually accessing one of these two dialogs and selecting the Input from capture cache check box.

Further information on how to generate a flat-field correction may be found in §5.1.5.4.

5.1.5.3 Setup

Toolbutton:

Shortcut:

Related commands:

This menu item controls the configuration of the cache file used when DigiFlow is capturing a digital video sequence directly from an attached camera. As noted in §2.3.4, it is important that the cache file is located on a disk other than that containing the operating system, and that the capture file is in a single large contiguous block, rather than many fragments scattered over the disk. Typically, the disk drive will only be able to keep up with the camera if the drive can devote all its time to writing the video data. This will not be the case if there is other disk activity occurring for that drive (as would be the case if it contained the operating system), or if the heads of disk drive have to continually move backwards and forwards across the disk as would occur if the file becomes fragmented.

Ideally, this menu item will be run when the capture disk is empty (*e.g.* following a reformat of the disk) or at least nearly so. By default the disk should be assigned the drive letter V: (either through the Disk Manager or by `net use V:` … onto a share) and the directory V:\Cache should be created before running File: Live Video: Setup. (Details on how to change the name or location of the cache file may be found in 13.2.)

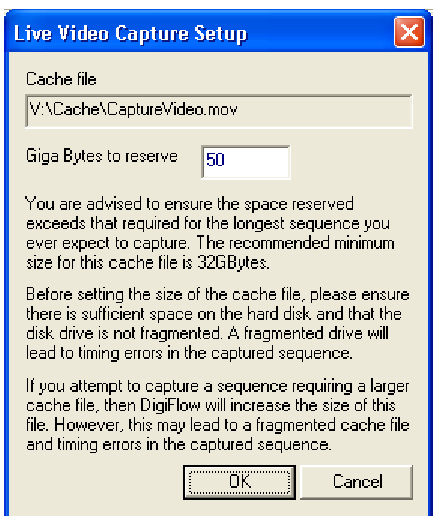


Figure 30: Configuration of the cache file for capturing video sequences.

Figure 30 shows the dialog that is produced. It is recommended that the default size of 50GB is used. While this may seem excessively large, if you make the file too small you may need to offload all your data and reformat the hard disk to be able to create a single large contiguous file at a later stage.

Note that if DigiFlow detects an existing V:\Cache\CaptureVideo.dfm, then this dialog will not be produced. DigiFlow does not provide a mechanism for you to remove or change the size of the cache file. If you must change the size, then you should delete it from Windows Explorer then run File: Live Video: Setup again. Note, however, that doing so may lead to fragmentation of this file, which may in turn prevent the hard disks from keeping up with the bandwidth from the camera.

5.1.5.4 Calibrating a camera

Calibration of a video camera falls at two different levels. On the one-hand, for some forms of processing, it is necessary to have a known linear relationship between actual intensity and the digitised values. Typically, all that is required here is to know the value to which black digitises. This form of calibration is dealt with later in §6.1. However, for some cameras, an additional low-level calibration is required to take into account details of the image sensor where these are not ideal.

This section is divided into two parts: determining the optimal black value for a camera, and generation of a flat-field correction. Most cameras do not require either of these processes to be undertaken.

Optimising black

Most image sensors are sensitive not only to visible light, but also to heat. Consequently, the output potentially depends on camera temperature as well as the incident light. To counteract this, many cameras have a built in method for eliminating or at least reducing the temperature-dependent signal. For example, the image sensor may have some rows of blacked off pixels to assess the temperature-induced component of the signal from the sensor. For most cameras their built-in approach works fairly well, although there are some where an additional calibration, adjustment or correction is necessary.

Related to this is the desire, for most forms of processing, to maximise the dynamic range available on the sensor. This includes ensuring black is digitised to a low *but nonzero* value. The reason why having black digitise to zero is that one cannot be sure whether it is digitised to zero or a negative value. Some methods of determining black are discussed in §6.1. Here we concentrate on the subset of cameras where the black offset can be set in software but the

camera's own internal mechanism for setting this is not sufficient. The DigiFlow cameras webpage (www.dalzielresearch.com/digiflow/cameras/) indicates which cameras require such calibration.

Once completed, the calibration is stored in the Cameras folder within the DigiFlow installation process and subsequently picked up automatically when DigiFlow is started. (Note that in order to complete this calibration, you must have write permission for this folder.) Unfortunately, for some cameras, it may be necessary to redo this calibration from time to time if there are significant differences in ambient temperatures, or if the camera is mounted differently so that its equilibrium temperature changes.

To complete the calibration, start by opening a live view (see §5.1.5.1) then use File: Run Macro (§5.1.3) to navigate to the Wizards subfolder and run Wizard_Camera_OptimiseBlack.dfc. The wizard will lead you through the process (see figure 31), which is typically completed with the lens cap on so that no light is falling on the image sensor.

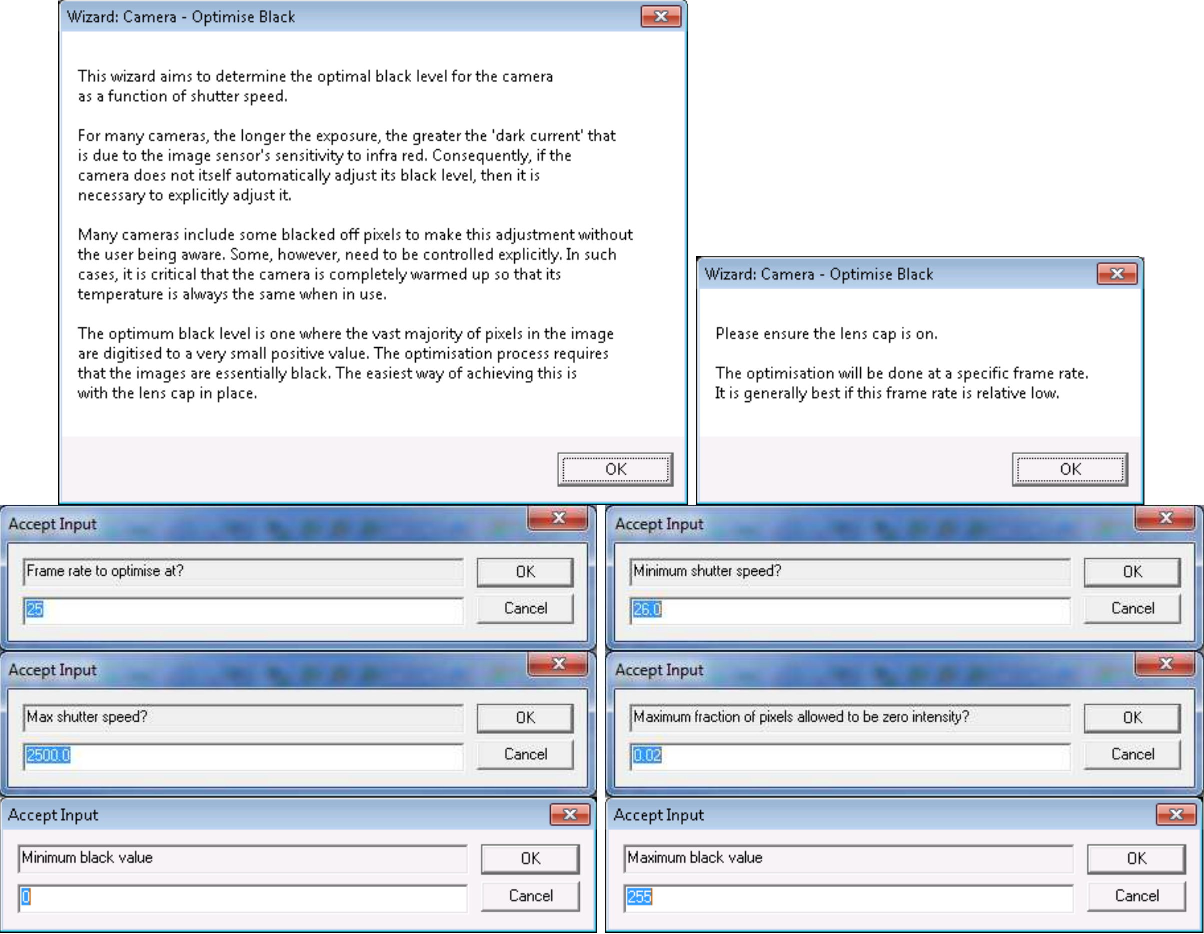


Figure 31: Dialogs for Wizard_OptimiseBlack.dfc.

Although the wizard operates on the assumption that it is the exposure time and not the frame rate which is of paramount importance, the wizard lets you choose the frame rate to be used. It is recommended that this is no less than one fifth of the maximum frame rate for the camera. The output of the wizard is fit of the best possible black level over a range of exposure times, the default maximum exposure time is slightly less than the reciprocal of the frame rate, whereas the default minim is 1/5000 s. Again, the user has the ability to adjust these.

For a given exposure, the definition of the optimal black is based on an instantaneous count of the number of pixels that have an exactly zero digitised value. By default, 2% (a fraction of 0.02) of the pixels are permitted to be zero. The final pair of questions prior to starting the

calculation specifies the range of black values that the wizard should search over. Note that the wizard can complete the task more rapidly if the lower bound is set close to the actual level needed.

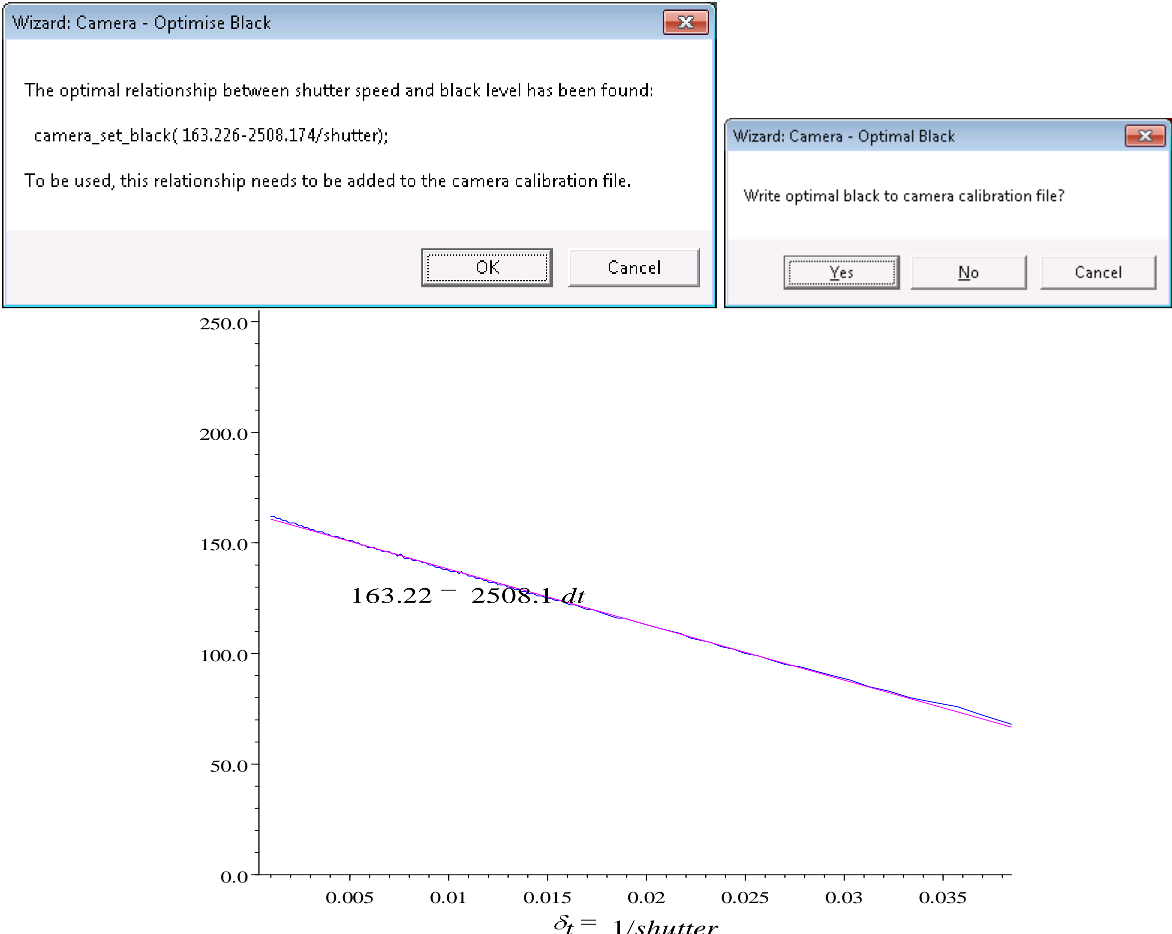


Figure 32: The output from Wizard_OptimiseBlack.dfc.

While it is working the macro will show in the status bar the various shutter speeds and black levels it is using. At the end, it provides a plot of the data (in blue) and the best fit (in magenta) for the optimal black. If desired, this best fit can be written automatically to the appropriate folder (based on the BitFlow camera file name) within the DigiFlow\Cameras folder.

Flat-field correction

The available flat-field corrections are stored in the Cameras folder within the folder in which DigiFlow is installed. These comprise of a fixed field noise and a gain. The wizard Wizard_Camera_FlatFieldCorrection.dfc provides a mechanism for determining and setting up the flat field correction. To run this wizard, open a live view with File: Live Video: Show Live Video (see §5.1.5.1) then select File: Run Macro (see §5.1.3) and navigate to the Wizards subfolder and run Wizard_Camera_FlatFieldCorrection.dfc.

Where a flat-field correction is necessary, it will generally be valid only for a particular shutter speed, hence the first question asked by the wizard is what shutter speed you will be using. The wizard will then set the camera to that shutter speed (if possible) and ask the user to put the lens cap on so that the camera sees only black. Click OK once this has been done, and the wizard will determine the time average black value over fifty frames. The user is then asked to make the view white. This is the most difficult and critical step. The user needs to illuminate the camera as uniformly as possible, preferably using the same lens and aperture as

will be used subsequently. A good diffuser paced immediately in front of the lens is essential for this (*e.g.*, a number of thicknesses of tracing film or a piece of opal acrylic), as is a fairly uniform light source to illuminate the diffuser. The intensity of the digitised image should be in the upper half of the range, but without any pixels saturating. Once this illumination has been set up, click OK and DigiFlow will again average the live video over fifty frames.

Once the averaging is complete, DigiFlow converts the 'white' image into a measure of the gain required to achieve a uniform intensity relative to the 'black' image, and the user will be prompted to use this to update the stored calibration. The default camera name under which to save the calibration is based on the name of the camera file currently in use, whereas the default name for the correction is a combination of the shutter speed selected at the start and the date on which the calibration is made.

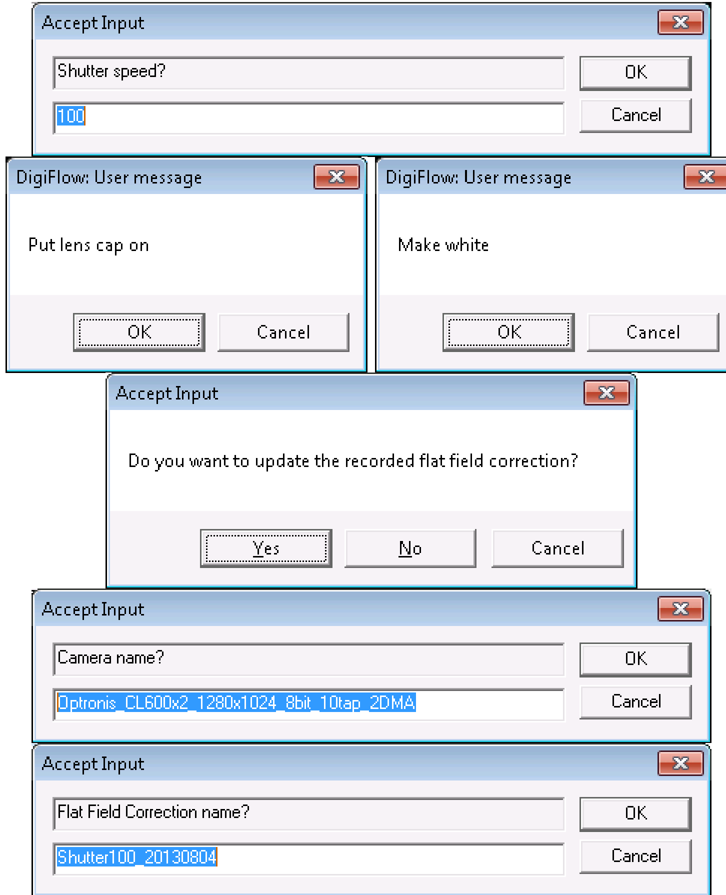


Figure 33: Prompts controlling the generation of a flat field correction.

5.1.6 Edit Stream

Toolbutton:

Shortcut:

Related commands: `process File_EditStream(..)`

This option provides efficient editing of a single video stream.

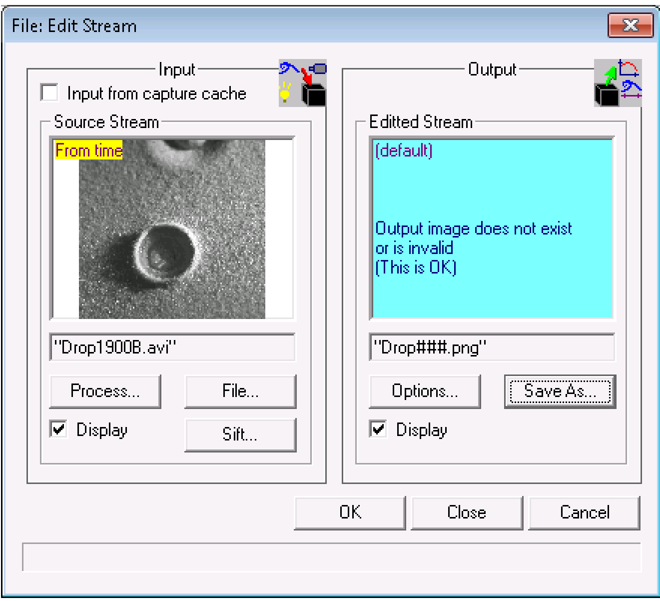


Figure 34: The Edit Stream dialog for editing a single video stream.

Parts of the Source Stream are copied to the Edited Stream; the parts to be copied are determined by the Sift button (see §4.3). Typically this is used to change image file format, reduce the time period, select only specific frames, and/or extract a subregion of the input stream. Note: if you do not click the Sift button when setting up the input stream, then DigiFlow might prompt you by starting up the sift dialog itself if it detects no or minimal changes are to be applied during the editing process.

The Input from capture cache check box disables the File and Process buttons, connecting instead the input to the capture cache file (see §5.1.5.2), allowing you to extract additional sequences from the cache file, if desired.

5.1.7 CameraFile

Toolbutton:

Shortcut:

Related commands: `process File_CameraFile(..)`

This option is a variant on File: Edit Stream (see §5.1.6). The principal difference is that it provides the opportunity to apply a flat field correction to the image while importing it from the cache used during a video acquisition process. For many video cameras, there is little point using this facility rather than File: Edit Stream either as a flat field correction is undertaken in the camera, or relatively little change is justified.

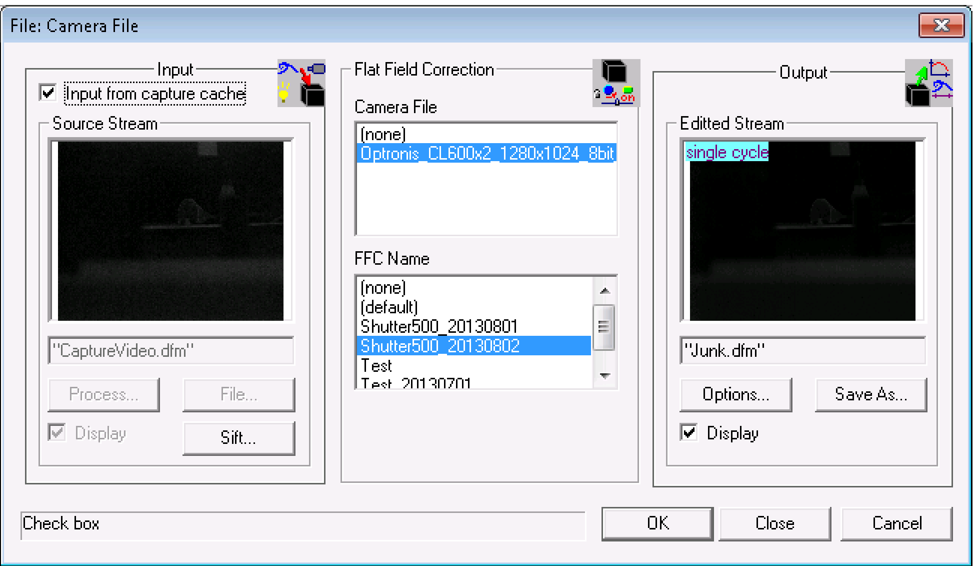


Figure 35: The Camera File dialog for copying a video stream, optionally transforming it using a flat field correction for the camera.

Parts of the Source Stream are copied to the Edited Stream; the parts to be copied are determined by the Sift button (see §4.3). If selected, the intensities will be transformed by a flat-field correction, specified by the combination of the Camera File and the FFC Name. Typically, the input is taken from the capture file by checking Input from capture cache and written out to a file format of the users choosing. Note: if you do not click the Sift button when setting up the input stream, then DigiFlow might prompt you by starting up the sift dialog itself if it detects no or minimal changes are to be applied during the editing process.

The Input from capture cache check box disables the File and Process buttons, connecting instead the input to the capture cache file (see §5.1.5.2), allowing you to extract additional sequences from the cache file, if desired.

Details of how to generate a flat-field correction may be found in §5.1.5.4.

5.1.8 Merge Streams

Toolbutton:

Shortcut:

Related commands: `process File_MergeStreams(..)`

This option allows two video streams to be merged into a single stream to provide an extended sequence.

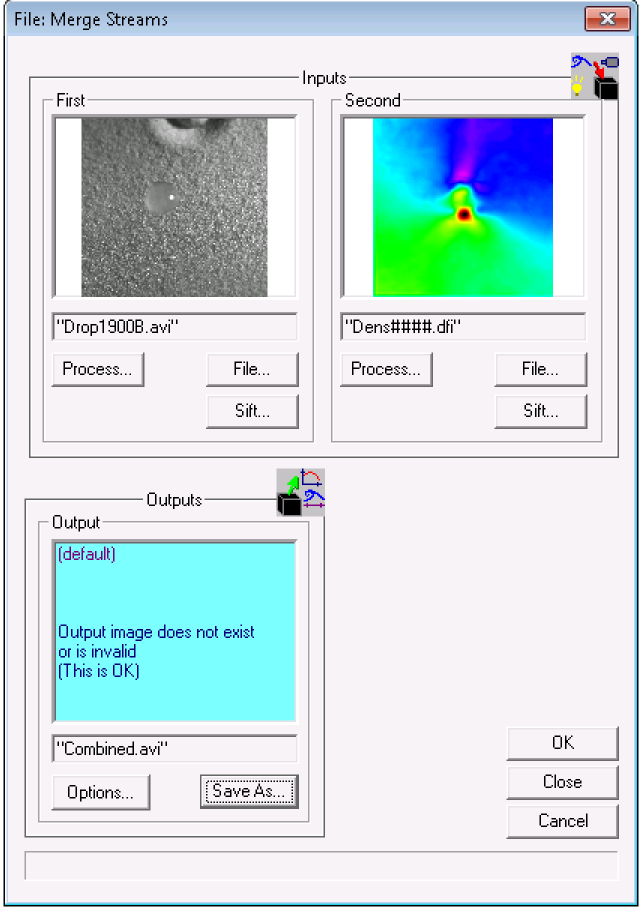


Figure 36: The Merge Streams dialog for combining image streams sequentially.

Two input selectors are provided: First and Second. These are written to the Output selector in the order suggested by their names. The timings of the two input selectors need not correspond, but the regions must conform. The First selector is the master, dictating the region to be used.

5.1.9 Export AVI

Toolbutton:

Shortcut:

Related commands: `process File_ExportAVI(..)`

This option provides an efficient mechanism for exporting a sequence of images, drawings or a DigiFlow movie to a standard Windows AVI movie file.

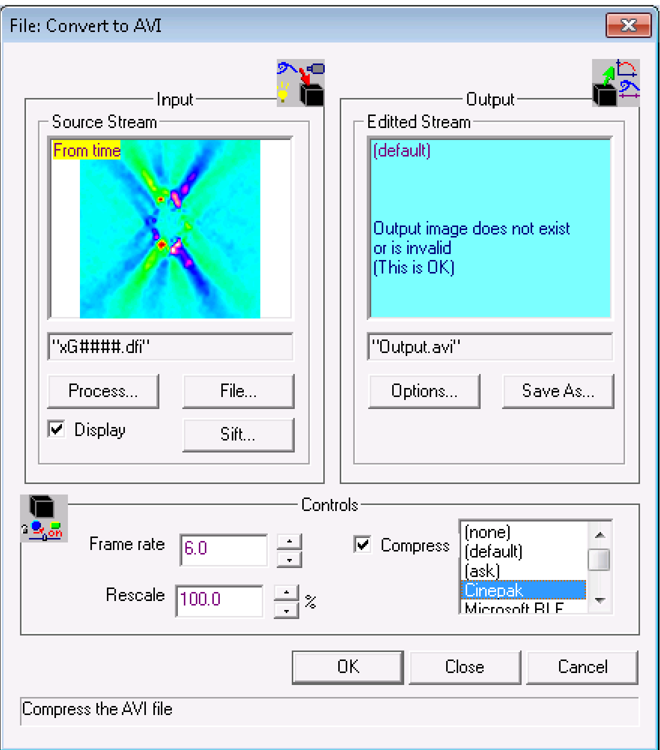


Figure 37: The Export AVI dialog.

The dialog consists of a single input and a single output selector. The latter should be directed to an .avi file. The playback speed of the resulting .avi file is set by Frame rate, while Rescale allows the input image to be rescaled (typically reduced in size) before copying to the movie. If Compress is not checked, then the full raw data is saved to the .avi file. If Compress is checked, then the type of compression is determined by the list box to the right. Only a subset of the compressions methods available are listed explicitly in this box. However, specifying (ask) will cause DigiFlow to prompt the user with the complete range of methods available at the point where DigiFlow is ready to save the first image in the output stream (*i.e.* the user will be prompted *after* OK has been clicked; see figure 38).

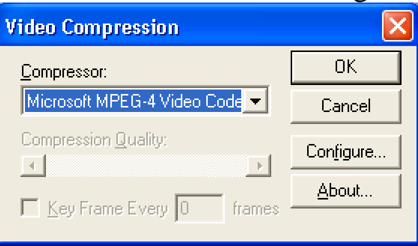


Figure 38: Compression options when exporting to an .avi file.

5.1.10 Print View

Toolbutton:

Shortcut: ctrl+P

Related commands: `print_view(..), ask_printer(..)`

Print out the currently selected viewing window. The menu version of this facility produces a dialog box allowing the user to select the printer, whereas the toolbar version simply prints to the currently selected printer.

5.1.11 Print Visible View

Toolbutton:

Shortcut: shift+ctrl+P

Related commands: `print_view(..)`, `ask_printer(..)`

This command is the same as File: Print View (see §5.1.10) except that only the currently visible part of the view is printed.

5.1.12 Export to EPS

Toolbutton:

Shortcut:

Related commands: `export_to_eps(..)`

Converts the currently selected viewing window into an Encapsulated PostScript (.eps) file. Section 2.2.2 describes how to set up an .eps printer driver that allows both bit image and vector graphics to be converted to .eps format. If the .eps printer is not set up, then DigiFlow will convert vector graphics to a bit image before generating the .eps file.

When using the printer driver, not only can the user specify the name of the output file (figure 39a), but some control over the format is also provided (figure 39b). In particular, a title may be added either above or below the figure, and the figure may be given a frame. Further, using the Text filter group, it is possible to suppress all text on a figure, or to replace each element of text with a unique letter combination. These text filtering options are provided for convenience with manuscript submissions where some journals wish all text removed from figures, while others use systems such as the LaTeX \psfrag package to replace the original text and fonts. Selecting Normal produces the eps containing the original labels, whereas with either of the PSFrag options the text is replaced by a unique character for each element. At the same time, DigiFlow creates a .tex file that contains the mapping between these characters and the original text. This .tex file can then be embedded in included in the main LaTeX document to reproduce the figure.

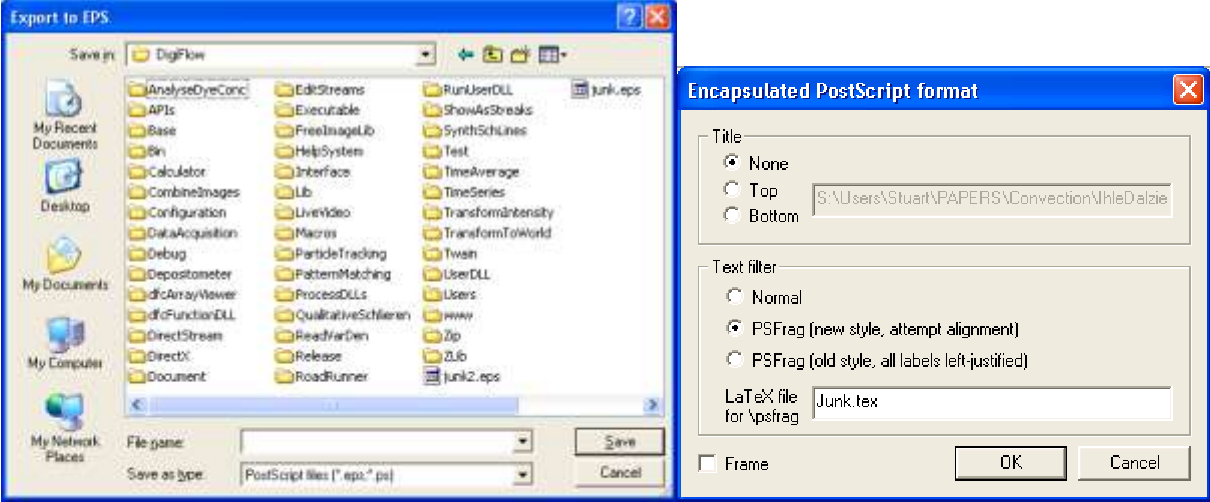
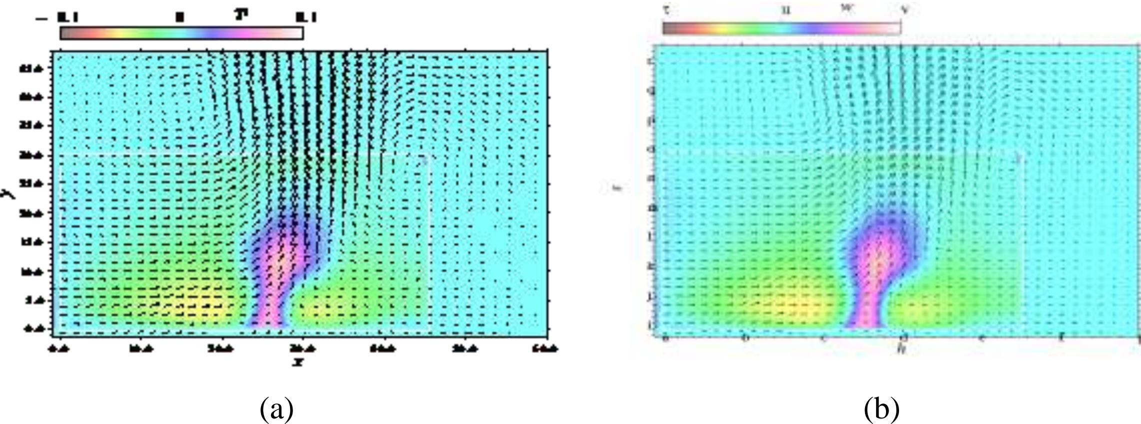


Figure 39: The dialogs controlling the saving and formatting of the exported eps file.

For example, when the drawing shown in figure 40a is exported, all the text is replaced by single characters. Close inspection shows that these characters appear to be positioned incorrectly, but this is necessary to resolve differences between the way Windows positions characters, and the way psfrag determines their positioning from the eps file.



(a) (b)

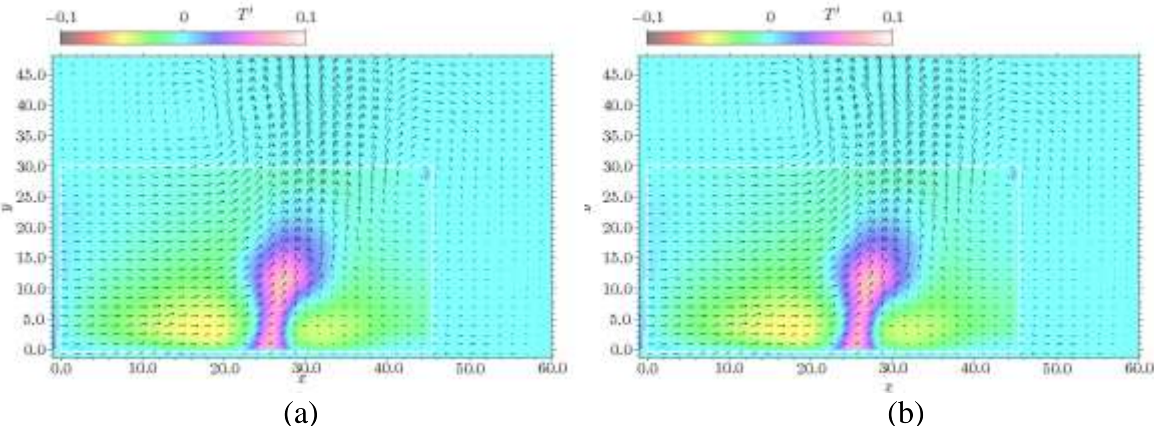
Figure 40: Example of exporting a drawing through psfrag with the PSFrag (new sytle) option. (a) Before exporting. (b) The resulting eps file.

At the same time as producing the eps file, DigiFlow also generates the following LaTeX file that contains a psfrag wrapper:

```
\begin{figure}
 \psfrag{a}[cc][Bl]{$0.0$}
 \psfrag{b}[cc][Bl]{$10.0$}
 \psfrag{c}[cc][Bl]{$20.0$}
 \psfrag{d}[cc][Bl]{$30.0$}
 \psfrag{e}[cc][Bl]{$40.0$}
 \psfrag{f}[cc][Bl]{$50.0$}
 \psfrag{g}[cc][Bl]{$60.0$}
 \psfrag{h}[cc][Bl]{$x$}
 \psfrag{i}[cr][Bl]{$0.0$}
 \psfrag{j}[cr][Bl]{$5.0$}
 \psfrag{k}[cr][Bl]{$10.0$}
 \psfrag{l}[cr][Bl]{$15.0$}
 \psfrag{m}[cr][Bl]{$20.0$}
 \psfrag{n}[cr][Bl]{$25.0$}
 \psfrag{o}[cr][Bl]{$30.0$}
 \psfrag{p}[cr][Bl]{$35.0$}
 \psfrag{q}[cr][Bl]{$40.0$}
 \psfrag{r}[cr][Bl]{$45.0$}
 \psfrag{s}[Bc][Bl]{$y$}
 \psfrag{t}[Bc][Bl]{$-0.1$}
 \psfrag{u}[Bc][Bl]{$0$}
 \psfrag{v}[Bc][Bl]{$  0.1$}
 \psfrag{w}[Bc][Bl]{$T'$}
  \includegraphics{junk2.eps}
 \end{figure}
```

Each of the `\psfrag{..}[..][..]{..}` statements gives the letter code, the required alignment relative to the point specified in the eps file, how the point specified in the eps file relates to the character rendered in the eps file, and finally the text to be typese in place of the letter code. Due to incompatibilities between Windows and psfrag, it is necessary for DigiFlow to position the characters in the eps file above and to the right of the reference point or else confusion arises in the height and width of the string. Thus each of the entries has `[Bl]` as the second of the positioning codes. The LaTeX text, however, is rendered in the correct place via the first of the positioning codes, as illustrated in figure 41a, although if there is a change in the size of the font some further adjustments may be necessary to improve the spacing between some elements. Here, for example, the font size has been increased and the x axis and y axis titles would look better if they were moved away from the axes. This may be achieved, for example, by adding `\raisebox(..)(..)` in the above example. In particular,

```
\psfrag{h}[tc][Bl]{\raisebox{-10 pt}{$x$}}
\psfrag{s}[Bc][Bl]{\raisebox{5pt}{$y$}}
```



(a) (b)

Figure 41: LaTeX output using automatically generated psfrag replacements. (a) Straight from the LaTeX generated by DigiFlow. (b) After small adjustments to the positions of the x and y axis labels.

Such adjustments are less frequently necessary if the original graphic is produced with text of much the same size as will be used in the final version. With DigiFlow drawings, this may be achieved readily by including a call to **draw_set_base_scales**(..) near the start of the drawing.

Note that .eps files may also be specified for the output stream from most of DigiFlow's menu options. If this is done, then the dialog in figure 39b replaces that normally produced by the Options button.

Some journals require the figures to be complete as .eps files rather than relying on \psfrag within the LaTeX. This does not prevent you, however, from using \psfrag in their production. The following process allows you to process the text with LaTex and end up with a stand-alone .eps file.

1. 1. Include the graphic plus all the psfrag commands in a LaTeX file with `\thispagestyle{empty}`. Probably best if standard PostScript fonts are used, so have something like:

```
\documentclass{article}
\usepackage{psfrag}
\usepackage{mathptmx} % PostScript fonts
\usepackage[T1]{fontenc}
\thispagestyle{empty}

\begin{document}
\input{myfig} % The figure in myfig.tex

\end{document}
```

2. Compile to create the .dvi and .ps, as normal. The output should all be on a single page.
3. Open the output .ps with GhostView. Select PS to EPS from the File menu. Select Automatically calculate BoundingBox, give the file a name (*e.g.* myfig.eps) and save.
4. Check that the myfig.eps can be read OK and has the correct bounding box.

5.1.13 Export Visible to EPS

Toolbutton: ☒

Shortcut:

Related commands: close_view(..)

This command is equivalent to File: Export to EPS (see §5.1.12) except that only the currently visible part of the view is used to generate Encapsulated PostScript.

5.1.14 Export to simple EPS

Toolbutton:
Shortcut:
Related commands: `export_to_simple_eps(..)`

The standard method of generating PostScript, described in §5.1.12, utilises a Windows printer driver to make the conversion. Vector graphics remain as vectors, while raster images remain as rasters. However, the Encapsulated PostScript produced tends to be cumbersome. Occasionally, Encapsulated PostScript with a simpler structure is desired. The present Export to simple EPS option does not utilise the Windows printer driver, but generates the Encapsulated PostScript directly. The big limitation of this option, however, is that it only produces raster formatted files. If applied to vector drawings, then these are first converted to a raster format.

5.1.15 Close

Toolbutton: ☒
Shortcut: ctrl+W
Related commands: `close_view(..)`

Close the active window. This is equivalent to clicking on the close button ☒ at the top right corner of the document window.

5.1.16 Close All

Toolbutton:
Shortcut:
Related commands: `close_all_views(..)`

Close all open views.

5.1.17 Exit

Toolbutton:
Shortcut:
Related commands: `exit_digiflow(..)`

Closes DigiFlow and all open image windows. This is equivalent to clicking on the close button ☒ at the top right corner of the main DigiFlow window. If DigiFlow detects that any processes are currently running then it will prompt the user to ensure DigiFlow should still be closed as this will terminate those processes.

5.2 Edit

5.2.1 Copy

Toolbutton:
Shortcut: shift+ctrl+C
Related commands:

Copies the currently selected image or drawing to the system clipboard. The image or drawing is available to other applications in both raster and metafile formats.

5.2.2 Copy as Bitmap

Toolbutton:
Shortcut: ctrl+alt+C
Related commands:
Copies the currently selected image or drawing to the system clipboard, placing it there as a bitmap regardless of its initial form.

5.2.3 Copy as Text

Toolbutton:
Shortcut: shift+ctrl+alt+C
Related commands:
Copies the file name or description of currently selected image to the system clipboard.

5.2.4 Zoomed Copy

Provides a group of options that allow images to be copied to the clipboard at a size that differs from the full resolution image.

5.2.4.1 Double size

Copies the currently selected image to the system clipboard, doubling the size of the image using bicubic interpolation, where appropriate.

5.2.4.2 Full size

Toolbutton:
Shortcut: shift+ctrl+C
Related commands:
Identical to Edit Copy (§5.2.1). Copies the currently selected image to the system clipboard, doubling the size of the image using bicubic interpolation, where appropriate.

5.2.4.3 Half size

Toolbutton:
Shortcut: shift+ctrl+alt+C
Related commands:
Copies the currently selected image to the system clipboard, halving the size of the image first.

Copies the currently selected image to the system clipboard, reducing the linear resolution of the image by a factor of three.

5.2.4.4 Quarter size

Copies the currently selected image to the system clipboard, reducing the linear resolution of the image by a factor of four.

5.2.4.5 Zoom

Copies the currently selected image to the system clipboard, adjusting the resolution using a user-specified factor (see figure 42).

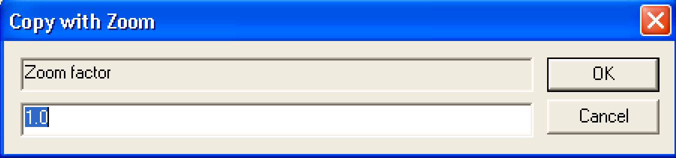


Figure 42: Dialog selecting scale factor for image to be placed on clipboard.

5.2.5 Properties

Toolbutton:
Shortcut: ctrl+\
Related commands: `read_image_details(..)`
Displays the properties for the selected window.

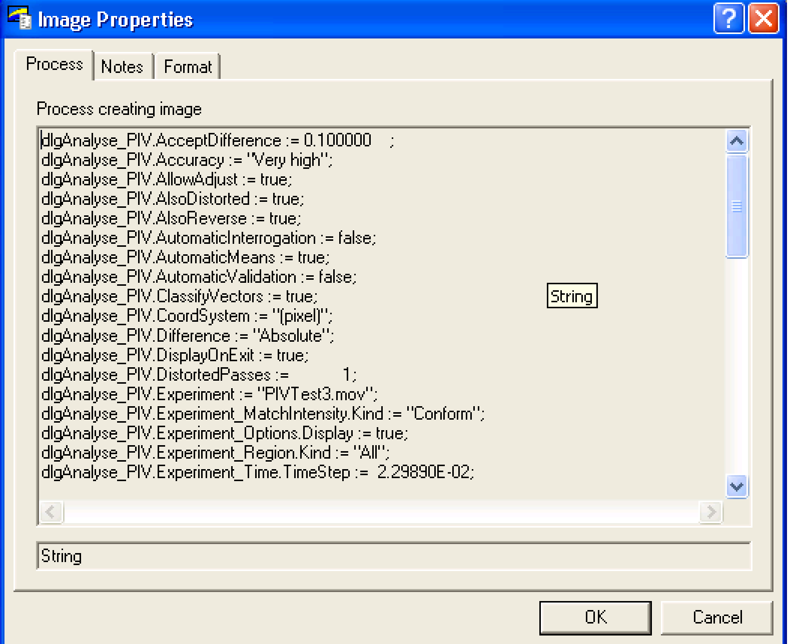


Figure 43: Process that created the image.

The Process tab contains comments supplied by the user at the time when the image was created. Note that this tab is only available when the image is supplied by a file format that supports the storage of this information. The contents here is exactly that that invoked the command (either interactively or from a dfc macro).

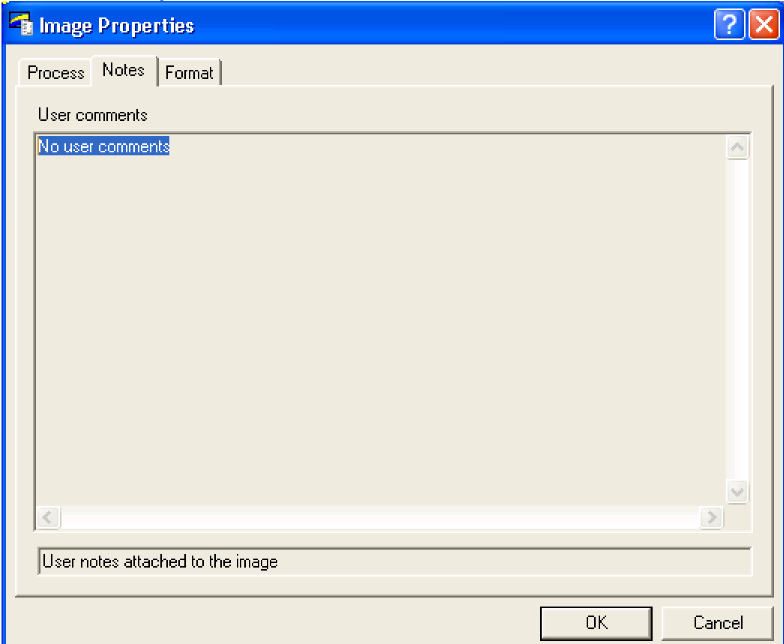


Figure 44: User comments.

The Notes tab contains comments supplied by the user at the time when the image was created. Note that this tab is only available when the image is supplied by a file format that supports comments.

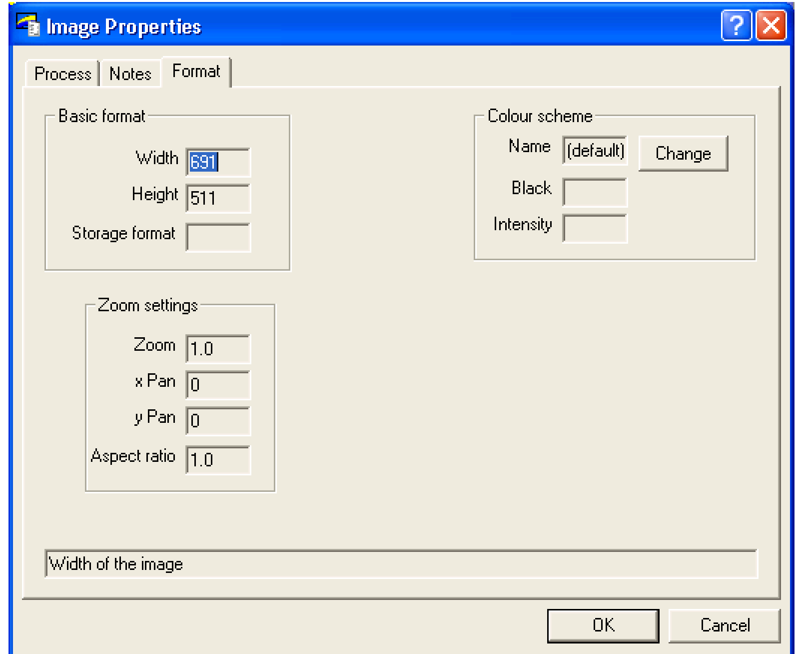


Figure 45: Image format.

The Format tab contains information describing the format of the image. This information is available for all image types.

5.2.6 Coordinates

Toolbutton:

Shortcut:

Related commands: coord_system_create(..), coord_system_mapping(..), coord_system_add_point(..),coord_system_destroy(..), coord_system_set_default(..),coord_system_list(..), coord_system_mapping(..),coord_system_get_mapping(..), coord_system_get_points(..),coord_system_translate(..), coord_system_translate_pixel(..), coord_system_units(..), coord_system_apply_region(..), pixel_coordinate(..), world_coordinate(..)

Provides the ability to define, edit and delete coordinate systems providing a mapping between the pixel coordinates of an image and some user-defined coordinate system.

Note that once defined, the coordinate system is stored in the DigiFlow_Status.dfs file in the directory from which DigiFlow was started. This file uses the standard dfc syntax but is run automatically upon starting DigiFlow. In some cases you may wish to make a copy of the relevant part of the coordinate system for backup purposes.

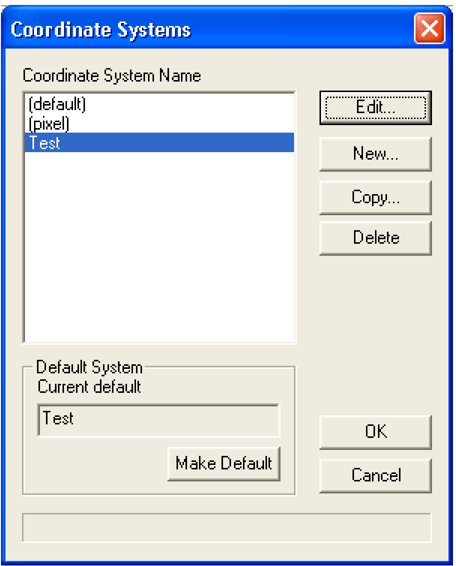


Figure 46: Selection of coordinate system.

The top-level dialog provides the ability to select the active coordinate system for the current window by clicking on the desired entry in the list box, or to make changes to the available coordinates.

The Edit, New and Copy buttons are used to adjust an existing coordinate system, create a new coordinate system, or create a copy of an existing coordinate system (respectively). A more complete description of these buttons is given below. The Delete button will remove the currently selected coordinate system from DigiFlow, while Make Default will register the selected system as the default for other operations.

Note: details of all coordinate systems defined, and which system is the current default, are stored in the DigiFlow_Status.dfs file located in the directory from which you started DigiFlow. These details are local to instances of DigiFlow started from that directory, although you may either copy the DigiFlow_Status.dfs file to another directory, or open it in an editor and extract the details of the coordinate system for use in a dfc macro. Refer to §

5.2.6.1 New coordinate system button

To create a new coordinate system, click on New. This starts a dialog allowing the name, type and units of the new coordinate system to be specified.

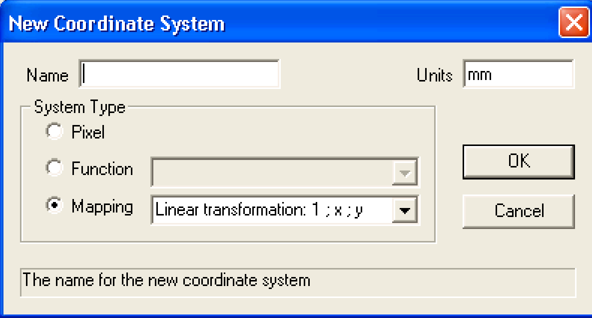


Figure 47: Give a name and type to the new coordinate system.

The Name of the coordinate system is arbitrary. The user should select a name that is meaningful to the task at hand. This name will subsequently be used for identifying the coordinate system. The Units are also arbitrary. They may refer to some standard measure of length, or to a dimensionless unit. The name of these units is recorded for later use in captions, titles, *etc*.

There are three possible types of coordinate system that may be specified. Pixel coordinates have a one to one correspondence with the pixels in the image and are the least flexible.

Function coordinates have a user-specified mapping between the pixel and user coordinates. This form of mapping is the most precise, but will only be of use where there is some external method of determining the required mapping functions. Four functions are required, separated by semicolons. The first two functions give the world x and y coordinates as functions of the pixel coordinates i and j, while the third and fourth give the pixel coordinates as functions of the world coordinates.

Mapping coordinates are generally the most useful. These systems are specified through a combination of mapping functions and identification points where both the pixel and user coordinates are known. A least squares mapping is then used to generate the unknown coefficients in the mapping functions and complete the transformation. There are a number of pre-defined mapping functions, or the user may specify their own. The format of the mapping function specification is an arbitrary name followed by a colon then a list of basis functions, each separated by a semicolon and expressed in terms of the generalised coordinates x and y, and optionally the time t. The points defining the unknown coefficients are specified using the Edit button of the parent dialog.

5.2.6.2 Edit coordinate system button

The Edit button starts a dialog that may be used to edit the Units and Mapping functions for the coordinate system. As noted above, the coordinate system may also be time dependent, in which case Use Time-dependent Mapping should be checked and the points defining the coordinate system should span both space and time.

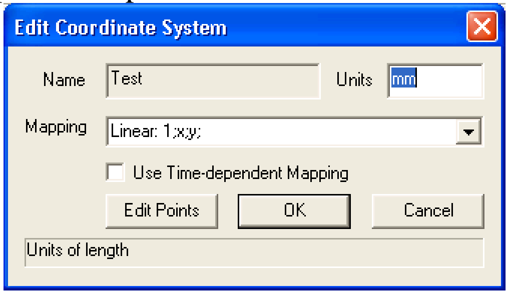


Figure 48: Editing coordinate systems.

To specify the points used for the mapping, the Edit Points button should be clicked, which will allow the user to specify points in the window that was active before entering the coordinate system dialogs. At the same time a modeless dialog box, which should be used to indicate the specification of the points is complete, is started.

Coordinate points are specified within the window by clicking at the desired location. This places a plus mark (+) at the position. The plus mark may then be dragged to a new location, if desired. Double-clicking the plus mark activates a dialog for specifying the user coordinates for this point.

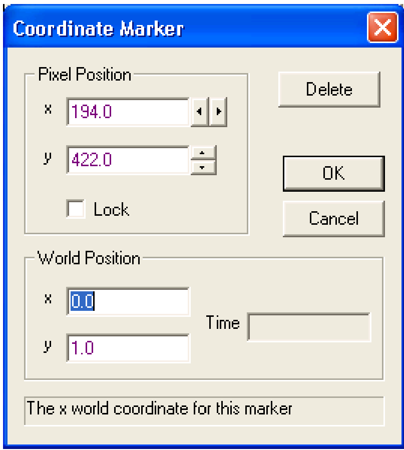


Figure 49: World coordinates for a point when defining a coordinate system.

This dialog gives the current Pixel Position of the point (and allows this to be edited), and provides the ability for the user coordinates to be defined in the World Position group. If a time-dependent mapping were specified, then the Time for this point must also be specified.

Clicking Delete will remove the point, while checking Lock will prevent the point being dragged around the image accidentally.

Figure 50: Indicate that you have finished editing the coordinate system markers.

When you have finished adding and/or editing the coordinate system markers, click the Finished button in the dialog shown in figure 50.

5.2.6.3 Copy coordinate system button

The Copy button provides the ability to make a copy of an existing coordinate system.

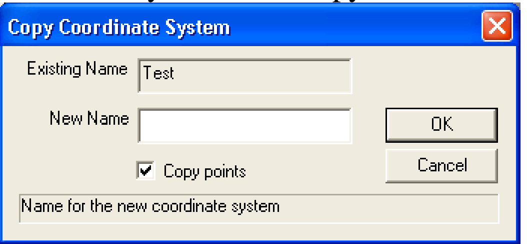


Figure 51: Copy a coordinate system.

5.2.6.4 Coordinate system creation wizard

As an alternative to manually defining a coordinate system as outlined in the above subsections, DigiFlow provides a wizard for this purpose. This wizard, Wizard_CoordSystem.dfc, takes the form of a dfc macro and can be found in the folder in which DigiFlow is installed. The purpose of the wizard is to take an image of a regular grid of features, location the features and form a coordinate system from them.

One way of forming the grid of regular features is to use one of the CoordinateSystemGrid_*.pdf files found in the DigiFlow installation folder. When printing these out, make sure they are printed at the correct scale rather than 'shrink to fit'! These grids, constructed from the following PostScript segment, can readily be printed out on paper or overhead transparency and placed in the flow. (It may be necessary to laminate a paper grid!)

```
%!PS-Adobe-3.0
```

```
% Select paper size (e.g. a4 or a2)
a4

/inch {72 mul} def
/mm   { 25.4 div inch } def

% Make all subsequent measurements in mm
1 mm dup scale

% Set the spacing of the features
/FeatureSpacing 10 def

/radius 0.75 def

% Page size: can be much bigger than the actual page size!
/Width 1000 def
/Height 1000 def

0 setgray % Make black

0 FeatureSpacing Height
  {/y exch def
     0 FeatureSpacing Width
       {/x exch def
        x y radius
        0 360 arc
        closepath % make sure start and end are connected
        fill}
   for}
for

# Set the number of identical copies you want and print
/#pages 1 def
showpage
%%EOF
```

Figure 52 shows such an image captured from such a coordinate system grid. Here the grid was printed with features spaced at 20mm on an A2 format with the subsequent pages laminated and then carefully taped together. Once you have an image of your grid, simply start the wizard and it will lead you step-by-step through the definition of your coordinate system. Note, however, that the wizard requires the actual coordinate system to be closely aligned with the pixels and that the mapping required for the coordinate system is close to linear.

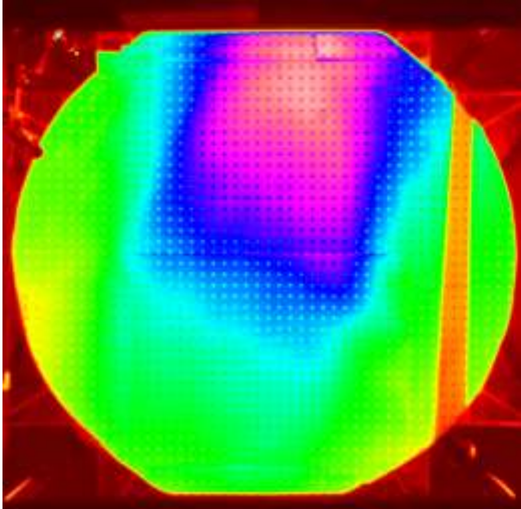


Figure 52: Image of coordinate system grid used with the Coordinate System Wizard.

5.2.6.5 Coordinate system test wizard

This wizard, Wizard_TestCoordinateSystem.dfc, is intended as a tool for checking the consistency and accuracy of a coordinate system and, optionally, removing points defining the coordinate system that may be outliers (*e.g.* due to them having incorrect data assigned to them). When run, the wizard will lead the user through the process.

5.2.7 Region

Toolbutton:
Shortcut:
Related commands: region_create(..), region_destroy(..), get_region(..), region_list(..)

In most cases the creation and modification of regions is handled as part of the sifting process started with the Sift button when specifying an input image stream, as described in §4.3.2. In some cases, however, it may be desirable to specify a region independently from processing any images (*e.g.* creating a region used during the capture of live video, as seen in §5.1.5.2). Selecting Edit: Region will start up a dialog containing only the Regions tab from the normal Sift dialog, as shown in figure 53. The only point in invoking this is to look at, define or modify a named region. The only lasting effect of this dialog is any changes in the definition(s) of named regions.

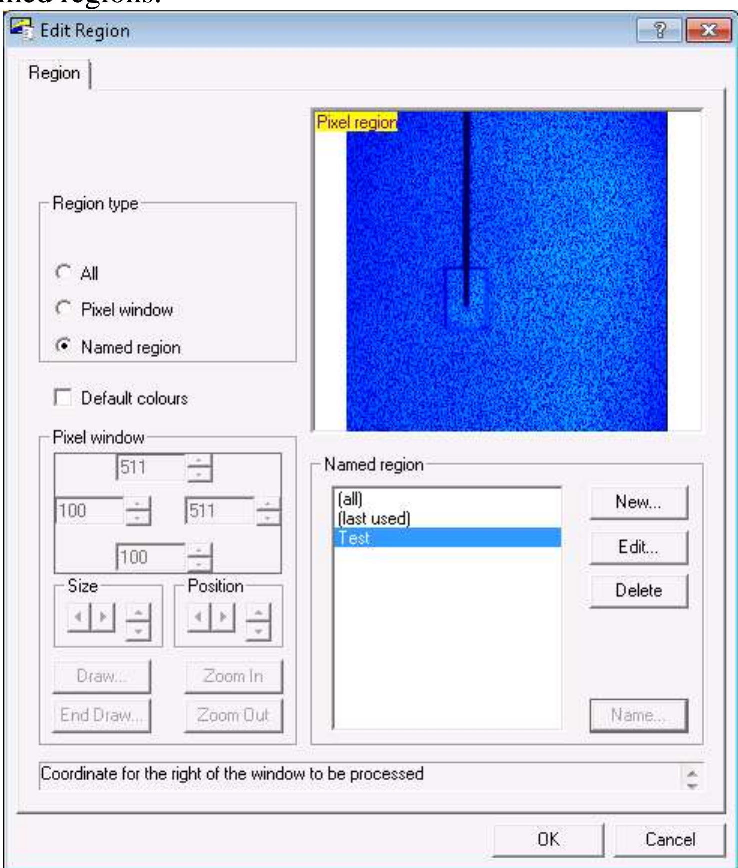


Figure 53: The Region tab from the Sift dialog is produced in response to Edit: Dialog.

5.2.8 Process again

Toolbutton:
Shortcut:
Related commands: `get_process_details(..)`

Users often wish to reprocess an image, perhaps making minor changes to the control settings, or maybe to apply the same process to a different set of images. The *Process again* facility provides a convenient method for doing this.

To use this feature, simply open the image for which you wish to replicate the process, and click on the Process again button (or select from the Edit menu). DigiFlow will then recover the process settings from the image and, where possible, use them to initialise the dialog that was initially used to create the image.

Note that this feature only works with DigiFlow-specific formats such as .dfi, .dfd or .dft files as other formats do not provide an appropriate mechanism for storing the settings used to create the image.

5.2.9 Dialog responses

The purpose of this facility is to provide an aid for those trying to create dfc files (see §8) to run processes, and to provide an alternative user interface to many of the DigiFlow processing facilities.

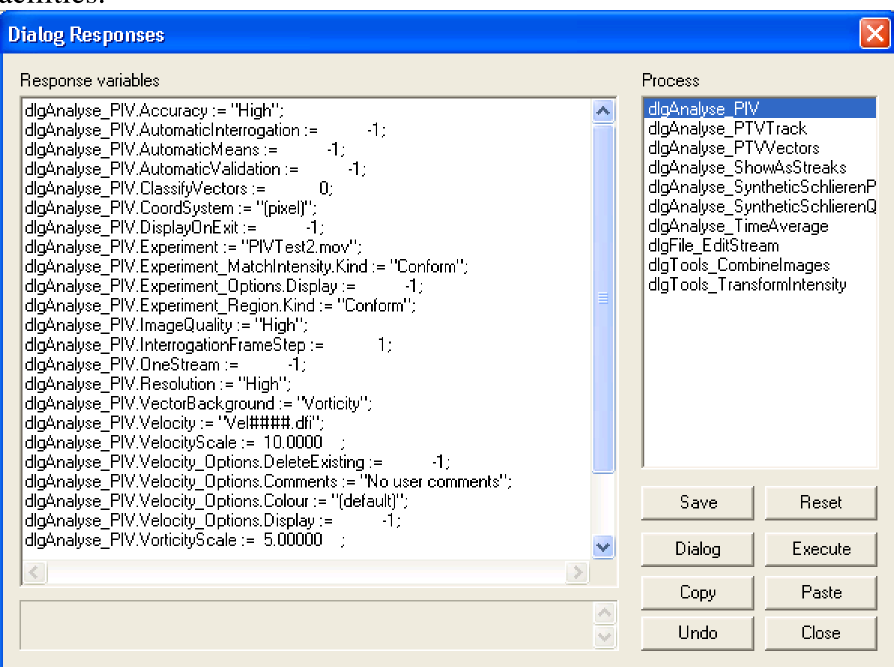


Figure 54: The Dialog Responses dialog that contains details of the responses corresponding to the latest invocations of the dialogs.

The Process list box indicates the dialog for which responses are required. This list is empty when DigiFlow is first started in a directory, but gradually fills as more DigiFlow features are used. Upon exit from DigiFlow, all this information is saved in the DigiFlow_Dialogs.dfs file that is created in the DigiFlow start directory.

Selecting a dialog from the Process list causes the corresponding response lines to be displayed in the Response variables edit box on the left-hand side of the dialog. Note that the entries in this edit control are always displayed in alphabetical order, and the list will only contain assignment statements. Entries in the edit box may be edited, selected, copied, *etc.*, as is standard for edit boxes. Users may find it useful to copy the contents of this edit box to *.dfc* files they are creating.

If the responses variables are edited, then they may be saved by clicking Save; alternatively Reset restores them to their previous values. The user will also be prompted to save any alterations if a different dialog is selected from the list.

The corresponding dialog may be started (*e.g.* to provide updated values) by clicking the Dialog button, while clicking Execute will cause the corresponding process to be started.

5.2.10 dfcConsole

Toolbutton: for j:=1;

Shortcut: ctrl+E

Related commands:

The dfcConsole provides an interactive tool for writing, editing and debugging dfc macro code.

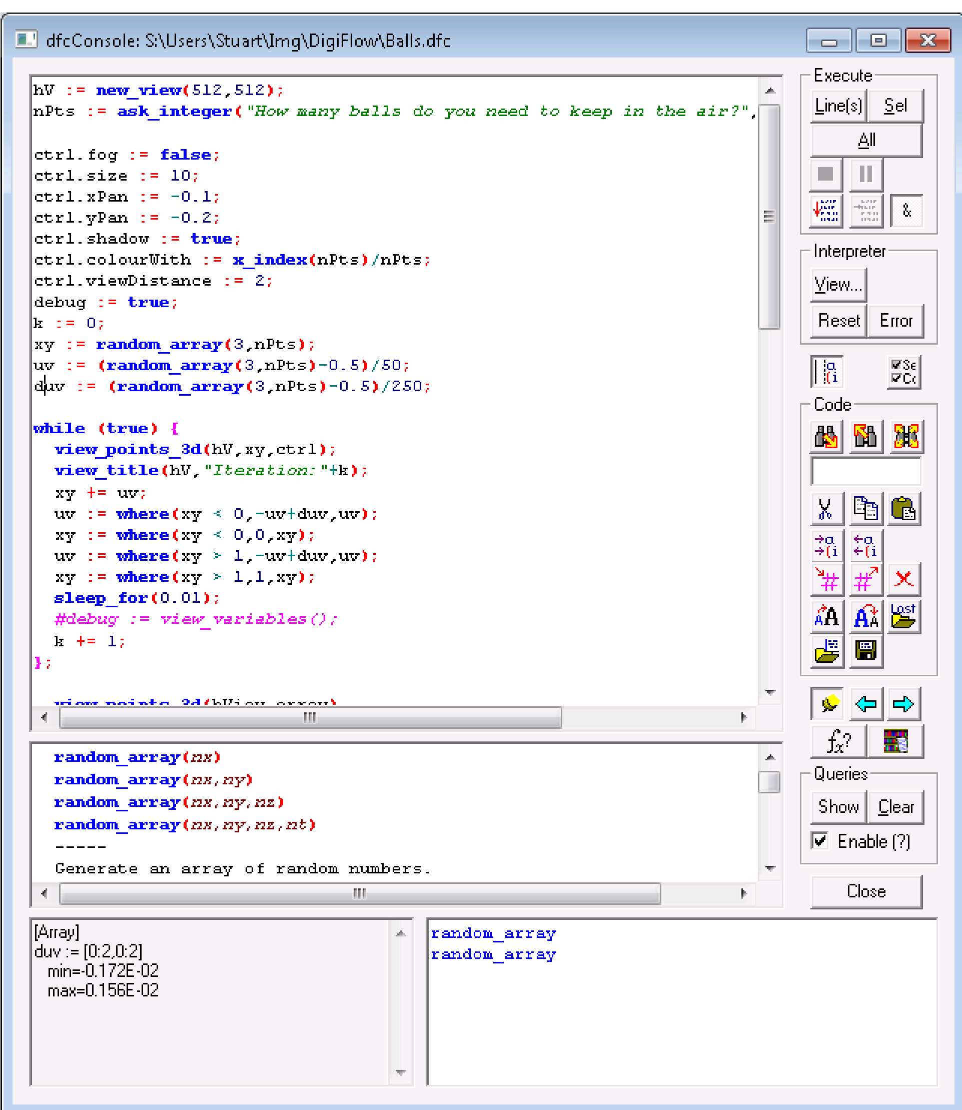


Figure 55: The dfcConsole dialog is resizable.

This resizable window contains an edit control allowing interactive editing of the dfc code to be run, alongside a series of controls allowing control over the execution environment and providing timely information.

The main window (top left) in the dfcConsole allows interactive editing of dfc code. Syntax is colourised and, matching parentheses, braces and square brackets are highlighted as they are entered (or when the <shift> key is depressed adjacent to a bracket). The buttons in the Code

group down the right-hand side provide the basic editing functionality in conjunction with standard short cuts such as ctrl+Z for undo (or shift+ctrl+Z for redo).

The Execute group may be used to selectively execute code. If there is no text selected, then Line(s) will execute the current line. If there is an active selection, then Selection will execute the selected code, and Line(s) will execute not only the selected text, but all the lines on which some text is selected. Regardless of the selection, All will cause the entire code to be executed. If Auto reset is checked, then using All will first discard any existing variables, *etc.*, from the interpreter. Note that `<alt><enter>` is equivalent to clicking Line(s).

Most of the control buttons are disabled while the code is executing. Amongst the exceptions are the stop and pause buttons. Clicking the stop button will abort the currently executing code, while the pause button will temporarily suspend execution. When toggled, the will cause the line currently being executed to be displayed regularly (at intervals of about 1s) in the bottom-left control of the console. If not toggled, then the button will cause the line currently being executed to be displayed when it is clicked. Checking Breaks (&) causes break points, indicated by an ampersand in the code (see §8.12.5) to be executed as and when they are found by the interpreter. If cleared, then the break points are ignored. Note that the status of the Breaks (&) control may be changed by the user as the dfc program runs.

The Interpreter group controls the internal state of the DigiFlow interpreter. Reset will clear all variables and functions from the interpreter, while View displays the variables and objects defined within the interpreter using the **view_variables**(..) interface. If an error occurs, then Last Error will redisplay the last error message.

Below the code window is a documentation window. When DigiFlow detects a function name under the cursor in the code window then it will display the documentation for this function. This documentation (which is also accessible through the dfc help facility) is hyperlinked to aid navigation. Standard forward and back navigation buttons are provided to the right of the window. A given help topic may be pinned (or unpinned) using , or by double tapping the <alt> key on the keyboard. Double-clicking on a line from within the documentation window will cause the corresponding text (*e.g.* the definition of the entry point for a function) to be inserted at the cursor in the main edit window.

The two windows at the bottom of the dfcConsole provide information about the contents of variables. When the cursor is on a variable name, and that variable has a defined value (typically the result of executing part of the code), then a summary of its contents will be displayed in the left-hand window.

The bottom right-hand window serves two separate functions. As the user types in code, a list of possible corresponding function names will be shown in this window. Double-clicking on any of the entries in the list will cause the corresponding help text to be displayed in the documentation window. This provides a convenient method of determining the name and usage of the most appropriate function if you do not know its name in advance. At run-time the bottom right-hand window takes on a different role and displays the output resulting from a Query (see §8.12.4 for further details). The Show button in the Queries group may be used to switch between the possible function list and the query output whilst editing code.

Standard file open and save buttons are provided to handle dfc code, along with a dedicated button to reopen the last dfc code edited. The dfcConsole will automatically reopen the last code edited if it did not complete cleanly its execution, and also saves snapshots of the code prior to execution, *etc.*, in DlgResponses.log.

The search facility is provided by a group of three buttons, , and , that search forwards or backwards for, or highlight all the text that matches that in the edit box beneath

the buttons. Cut (ctrl+X or), copy (ctrl+C or) and paste (ctrl+V or) operate in the standard way. Additionally, indenting (ctrl+space or), unindenting (shift+ctrl+space or), commenting (ctrl+/ or) and uncommenting (shift+ctrl+/ or) can help with the laying out and testing/documenting of dfc code.

Utility buttons to delete all code , help and code library buttons are provided for convenience. Similarly, the font size may be increased or decreased .

Information about where a running code is currently executing can be found by clicking the button. This will cause the line currently being executed to be displayed in the dfc help window at the bottom of the console. Similarly, clicking turns on (or off) automatic display of the line currently being executed, updated typically every second. Note that these facilities can have a significant impact on the execution speed.

When toggled, the smart indent button, , will attempt to align elements of the code in an intelligent way. The settings button, , opens the subdialog shown in figure 56. This dialog controls syntax and variable highlighting, and whether the calculator interpreter is automatically reset each time All is clicked.

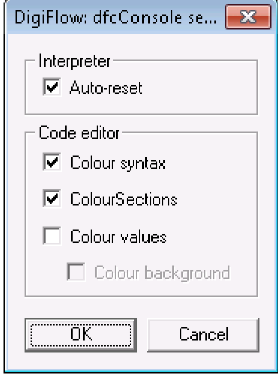


Figure 56: dfcConsole settings for controlling syntax highlighting, *etc.*

Note that using either the Close or ☒ buttons to close the window will prompt to save the dfc code it contains. This feature is bypassed if you close the main DigiFlow window which will simply close the window without prompting for the dfc code to be saved.

5.3 View

5.3.1 Zoom

5.3.1.1 In

Toolbutton:
Shortcut: alt+Z, *or* **up_arrow**
Related commands: view_zoom(..)

Zoom in the current window by a factor of two. Note: if the ctrl key is held down while clicking the toolbutton or using the up arrow, then the window is resized to fit the new zoom.

5.3.1.2 Out

Toolbutton:
Shortcut: shift+alt+z, *or* **down_arrow**
Related commands: view_zoom(..)

Zoom out the current window by a factor of two. Note: if the ctrl key is held down while clicking the toolbutton or using the down arrow, then the window is resized to fit the new zoom.

5.3.1.3 Full size

Toolbutton:
Shortcut: ctrl+1
Related commands: view_zoom(..)

Zoom the current window to full size (one pixel on the display for each pixel in the stored image). Note: if the ctrl key is held down while clicking the toolbutton, then the window is resized to fit the new zoom.

5.3.1.4 Custom

Toolbutton:
Shortcut:
Related commands: view_zoom(..)

Starts the zoom dialog box that allows a broader range of zooms to be selected, and also allows specification of the aspect ratio for the displayed image.

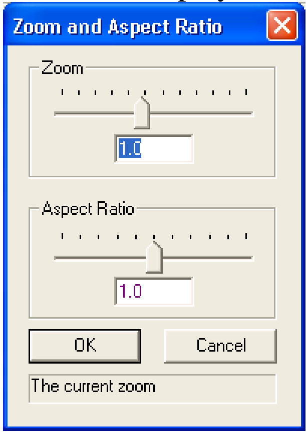


Figure 57: The custom zoom dialog box.

The two slider controls are linked with the two edit boxes. The Zoom setting controls the number of pixels on the display used to display a single pixel in the stored image *in the horizontal direction*. In contrast, the Aspect Ratio control determines the shape of the virtual pixels to be displayed. For images captured through DigImage, the aspect ratio should be set to 0.68 for PAL systems, or 0.75 for NTSC, thus recovering the original aspect ratio of the images.

5.3.1.5 To Window

Toolbutton:
Shortcut: shift+ctrl+Q
Related commands: view_zoom_to_fit(..), view_zoom_all_to_fit(..)

Changes the zoom to fit the current window.

5.3.1.6 All Full Size

Toolbutton:
Shortcut: shift+ctrl+1
Related commands: view_zoom_all(..)

Changes the zoom of all windows to 100%, and fits the windows to the size of the images.

5.3.1.7 All Half Size

Toolbutton:
Shortcut: shift+ctrl+2
Related commands: view_zoom_all(..)

Changes the zoom of all windows to 50%, and fits the windows to the size of the images.

5.3.1.8 All Third Size

Toolbutton:

Shortcut: shift+ctrl+3

Related commands: view_zoom_all(..)

Changes the zoom of all windows to 33%, and fits the windows to the size of the images.

5.3.1.9 All Quarter Size

Toolbutton:

Shortcut: shift+ctrl+4

Related commands: view_zoom_all(..)

Changes the zoom of all windows to 25%, and fits the windows to the size of the images.

5.3.2 Fit Window

Toolbutton:

Shortcut: ctrl+Q

Related commands: view_fit_to_zoom(..), view_fit_all_to_zoom(..)

Resizes the current window so that it fits the zoom of its contents.

5.3.3 Cursor

Under all cursor modes, holding down the left mouse button will cause a duplicate cursor to be displayed at the same pixel location on all the other open image windows. At the same time, the current intensity and other related data (*e.g.* velocity) will be displayed in the status bar for that window. This feature is valuable when trying to assess the relationship between features in different images. As described below, the cursor can also be set to display other information or to perform other tasks when the buttons are clicked.

If you hold down alt and click the left mouse button, then the duplicate cursor will continue to be displayed in the location of the click on all windows (and on the window you clicked) until you click again without the ctrl key held down. (Clicking again while holding down ctrl will cause a further duplicate to be placed on all the windows, and so on.)

	Left	Middle	Wheel	Right
	Click: Show duplicate cursor on all windows.	Click: Play/pause movie or sequence. Drag: Move rapidly through a movie or sequence.	Step through movie or sequence	
shift	Drag: Scroll (pan) current view.		Zoom current view.	
ctrl		Click: Play/pause movie or sequence in all views*. Drag: Move rapidly through all movies or sequences*.	Step through movie or sequence in all views.	
alt	Click: Add cursor marker to all windows (removed by next click)			
shift+ctrl	Drag: Scroll (pan) all views.	Click: Move all movies or sequences	Zoom all vies.	

	to frame of the clicked view.*
shift+alt	
ctrl+alt	
shift+ctrl+alt	

*Sequence animation controls only operate on views that do not have their synchronisation button activated.

5.3.3.1 Show Where

Toolbutton:

Shortcut: ctrl+alt+M (ctrl+M to turn off)

When the left mouse button is held down on an image, a popup window will appear next to the cursor showing the current pixel and (if defined) world coordinates. Clicking the right button (while the left button is still depressed) will produce a message box showing the coordinates, as seen in figure.

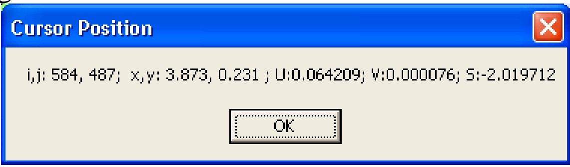


Figure 58: The message box produced when the right-hand mouse button is clicked whilst the left-hand button is held down. The coordinate i,j is in pixel coordinates, while x,y are in the current world coordinates. In this case a velocity field was being explored and U and V represent the velocity, and S gives the scalar field (here vorticity) on which the velocity is displayed.

5.3.3.2 Measure Distance

Toolbutton:

Shortcut: shift+ctrl+M (ctrl+M to turn off)

When the left mouse button is held down on an image, a popup window will appear next to the cursor showing the distance between where the left-hand button was depressed and the current location of the mouse pointer. Clicking the right button (while the left button is still depressed) will produce a message box showing the distance.

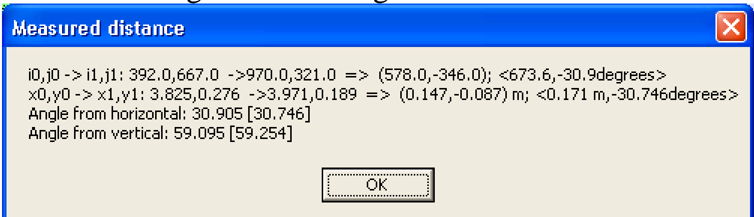


Figure 59: The message box produced when measuring distances with the cursor. The first line gives the pixel coordinates of the start and end points, along with the Δx and Δy in pixels (within brackets) and the distance and angle (within <..> pairs). The second line repeats this for world coordinates. The last two lines give the principal angle from horizontal and vertical, respectively, for the line with the first value in each case being in pixel coordinates, whist the second gives the same information with reference to the world coordinate system.

5.3.3.3 Move Image

Toolbutton:

Shortcut: shift+ctrl+alt+M (ctrl+M to turn off)

When the left mouse button is held down on an image, moving the mouse will pan the image. The mouse wheel can be used to zoom the image.

Note that holding down shift while using the mouse will temporarily activate this feature regardless of the toolbar settings.

5.3.3.4 Move All Images

Toolbutton:
Shortcut:
Similar to Move Image (§5.3.3.3), but moves and zooms all open images.

Note that holding down shift+alt while using the mouse will temporarily activate this feature regardless of the toolbar settings.

5.3.4 Vectors

5.3.4.1 Increase Length

Toolbutton:
Shortcut:
Related commands:
Increase the length of velocity vectors and similar arrows. For finer control, see View: Appearance (§5.3.5).

5.3.4.2 Decrease Length

Toolbutton:
Shortcut:
Related commands:
Decrease the length of velocity vectors and similar arrows. For finer control, see View: Appearance (§5.3.5).

5.3.4.3 Reset Length

Toolbutton:
Shortcut:
Related commands:
Resets the length of velocity vectors and similar arrows to their defaults. For finer control, see View: Appearance (§5.3.5).

5.3.5 Appearance

Toolbutton:
Shortcut: ctrl+A
Related commands:
This tool provides a variety of tools for adjusting the appearance of an image. The scope of the tools depends on the format of the image; in particular, if the image is integer or floating point, and whether it contains a single plane of information or multiple planes.

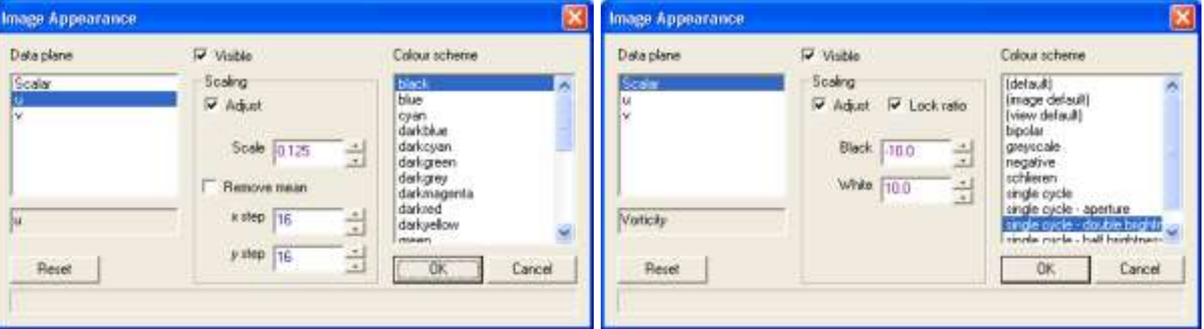


Figure 60: Dialog for adjusting the appearance of an image. This example shows the features available for an image containing a velocity field.

The Data plane control will show the data planes available within the image. For simple images, only one data plane will be listed, but for velocity fields, for example, both velocity (vector data) and a background image (scalar data) will be listed. The appearance of each data plane type can be changed by selecting it in the Data plane control, then making the required adjustments. For example, if a velocity plane is selected, then the scale of the vectors, the spacing between the vectors, and the colour in which they are plotted may all be changed. Additionally, a check box allows the removal of any mean velocity. For scalar data, the colour scheme and the mapping between the scalar values and the limits of the colour scheme can be changed. Some of this latter functionality is also available through the Colour scheme dialog – see §5.3.6.

5.3.6 Colour scheme

Toolbutton:
Shortcut: shift+ctrl+C
Related commands: `view_colour(..)`, `colour_scheme(..)`, `add_colour_scheme(..)`, `delete_colour_scheme(..)`

Used to select the colour scheme for the active image, or define a new colour scheme. The Select Colour Scheme dialog box invoked by this option provides the ability to add, remove and alter colour schemes.

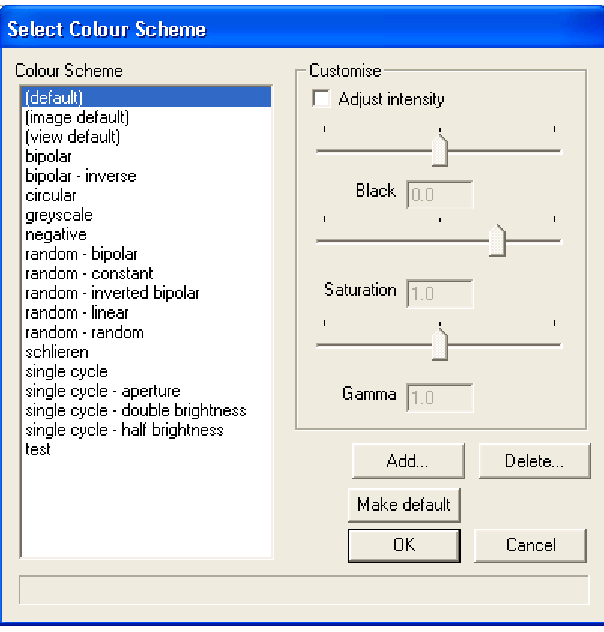


Figure 61: Dialog used for selecting the colour scheme for an image.

A specific colour scheme may be selected by clicking on the name in the list box. The (image default) scheme is the scheme in use when the window was created. The (view default) is that in use for the image when this dialog was entered.

Note: This dialog is modal for free versions of the DigiFlow licence, but modeless (allowing switching between other elements of DigiFlow) for full licences.

Checking Adjust intensity and moving the sliders or typing a value into the associated edit boxes may alter the appearance of a given scheme. For example, by setting Black to 1.0 and Saturation to –1.0, a scheme with the negative colours may be produced. Colour schemes created and added through the dfc function **`add_colour_scheme(..)`** are saved in DigiFlow_Status.dfs and will be included in the list of available schemes.

Clicking on Add brings up a dialog for adding new colour schemes. Details of these new schemes are added to the DigiFlow_Status.dfs file and thus remain available the next time DigiFlow is started in the same directory.

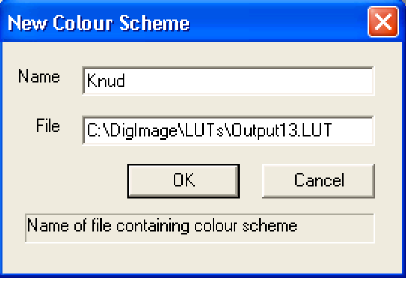


Figure 62: Add a new colour scheme.

DigiFlow understands the DigImage colour schemes. These may be added, as illustrated in figure 62, by simply giving them a name and typing the full path specification of the colour scheme. For DigImage, the colour schemes are stored in the %DigImage%\LUTs directory under the name Output##.lut, where ## represents a two digit number, starting with 00 for the first DigImage colour scheme.

Alternatively, specifying an image in the File edit box will install the colour scheme stored in that image, or construct a colour scheme of your own. The following example uses a short piece of dfc code to construct and install a colour scheme that is white in the middle tending towards red for low values and blue for high values. In this particular case, the scheme is tweaked so that its equivalent greyscale intensity varies parabolically. (It is often useful to have a colour scheme that works in both colour and monochrome.) The parabolic nature will emphasise extreme values more than small values. Such a scheme can be useful for displaying vorticity, for example.

```
# Initialise arrays to white
red := make_array(1,256);
green := red;
blue := red;

# Make basic colour scheme as two linear ramps
for k:=0 to 63 {
  z := k/64;
  p := 128 + k;
  q := 127 - k;

  red[p] := 1 - z; # Remove red
  blue[q] := 1 - z; # Remove blue
};
for k:=64 to 127 {
  z := (k-64)/64;
  p := 128 + k;
  q := 127 - k;

  red[p] := 0; # No red
  green[p] := 1 - z; # Remove green

  blue[q] := 0; # No blue
  green[q] := 1 - z; # Remove blue
};

# Make parabolic in grey value
grey := 0.299*red + 0.587*green + 0.114*blue; # Current grey
     equivalent
p := x_index(256)/255;
limit := 0.8; # Limiting grey value for ends
target := 1 - 4*limit*(p - 0.5)^2; # Target grey equivalent
```

```
scale := target/(grey max 1e-8); # The required scaling
red *= scale;
green *= scale;
blue *= scale;

# Install the colour scheme
add_colour_scheme("test",red,green,blue);

# Now test the colour scheme
im := x_index(512,64)/511;
hV := view(im);
view_colour(hV,"test");
```

5.3.7 Toggle colour

Toolbutton:
Shortcut: ctrl+B
Related commands: `view_toggle_colour(..)`**,**

When an image is displayed using false colour, clicking this button will temporarily switch to greyscale. Clicking a second time will return the image to the original false colour scheme.

5.3.8 Toolbar

Turns on or off the toolbar at the top of the main DigiFlow window. It is recommended that you leave this turned on.

5.3.9 Slaves

Slave windows are a special type of window that are tied to a normal window – the master window – in such a way that when the master window is updated, any changes are reflected in the slave window. For example, if the master window is part of a sequence, then stepping through the sequence will update not only the master window, but also its slave.

A given master window may have one or more slave windows. When a master window is closed, its slaves are closed automatically (without prompting). Closing a slave window does not alter the state of the master nor force it to close.

For a more comprehensive array of slaves, refer to Tools: Slave Process in §5.7.5.

5.3.9.1 3D View

Toolbutton:
Shortcut:
Related commands: `slave_view_3d(..)`

This option takes a copy of the image in the current active window and uses the values it contains to create a three-dimensional surface plot in a new slave window. The special slave window has its own toolbar that controls the three-dimensional rendering and allows re-orientation and other visual changes. This window is illustrated in figure 63.

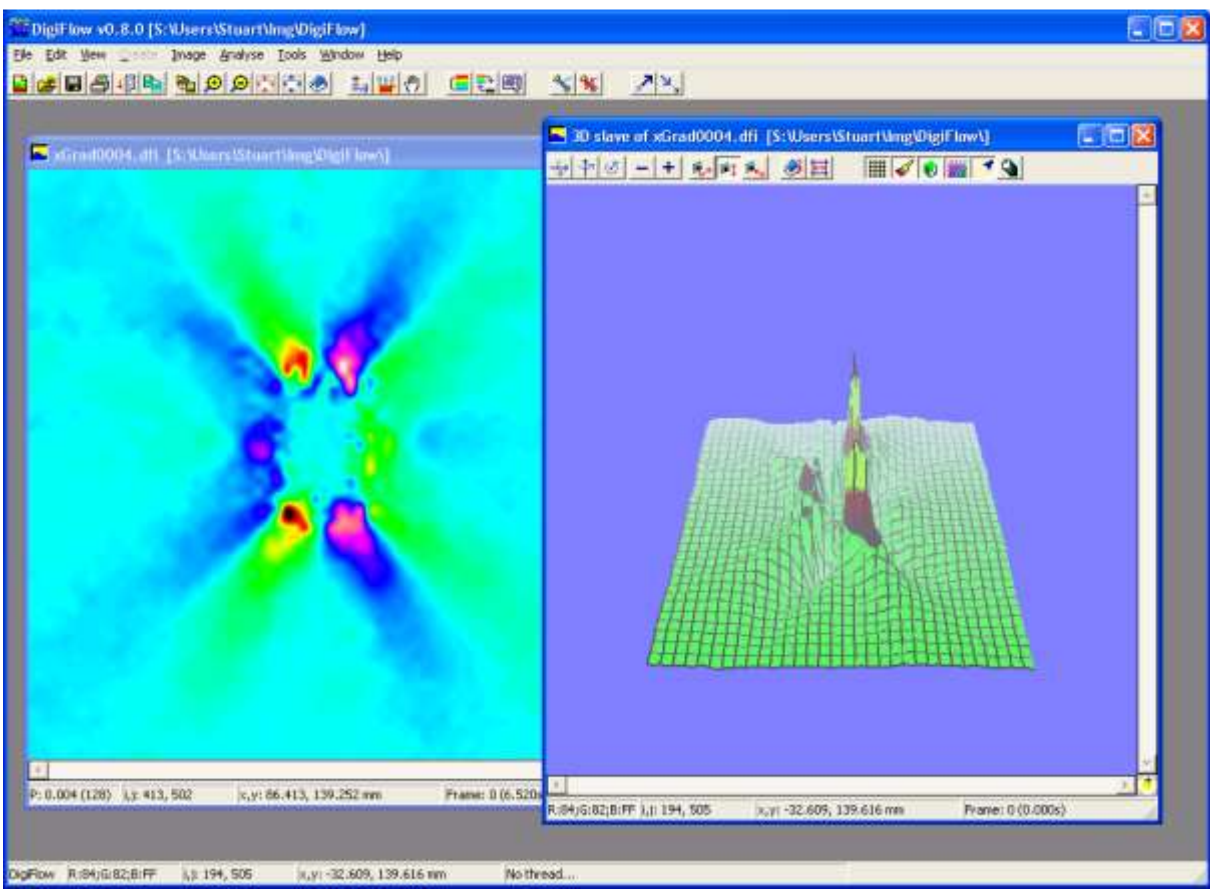


Figure 63: A three-dimensional slave view.

The buttons on the toolbar are divided into five groups. With the exception of the + and buttons (the second group), the first four groups of buttons act as a set of radio buttons in which only one button may be pressed at a time. The pressed button then selects the drawing attribute that is to be changed using the plus or minus buttons. A brief summary of the buttons is given below.

Button	Description
	Enable rotations about the *x* axis. Once enabled, then the − and + buttons rotate the plot about the x axis
	Enable rotations about the *y* axis. Once enabled, then the − and + buttons rotate the plot about the x axis
	Enable rotations about the *y* axis. Once enabled, then the − and + buttons rotate the plot about the x axis
−	Decrements the number associated with a render setting. The render setting is selected by clicking on the corresponding radio button in the toolbar menu.
+	Increments the number associated with a render setting. The render setting is selected by clicking on the corresponding radio button in the toolbar menu.
	Enable panning the three-dimensional view left or right. Once enabled, then the − and + buttons pan the plot left or right.
	Enable panning the three-dimensional view up or down. Once enabled, then the − and + buttons pan the plot up and down.

	Enable moving the camera closer or further. Once enabled, then the − and + buttons move the view position in and out.
	Enable changing the vertical scale of the plotted data. Once enabled, then the − and + buttons decrease and increase the scale.
	Enable changing the spacing of the grid lines plotted on the surface. Once enabled, then the − and + buttons change the spacing..
	Toggle the grid on and off.
	Paint the surface
	Toggle hidden line removal
	Toggle depth fog on and off.
	Activate spotlight.
	Change the colour of the background.
	Toggle display of axes.
	Reset settings.

Note that the view produced is displayed as a bitmap. This may be saved, printed and/or converted to an Encapsulated PostScript plot.

5.3.10 Threads

Toolbutton:
Shortcut:
Related commands: as_thread(..), is_running(..), wait_for_end(..), pause_thread(..), unpause_thread(..), kill_thread(..), get_thread_prioritiy(..), set_thread_priority(..), process, process_as_thread

Starts the dialog box showing and controlling the various active processing threads.

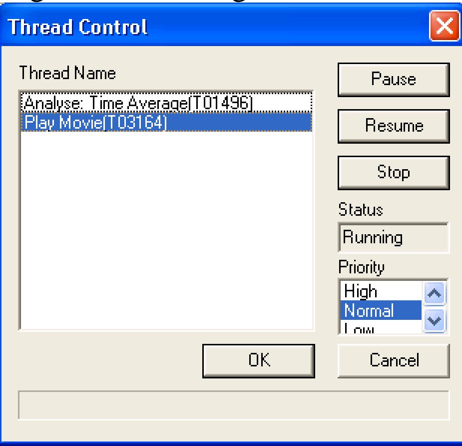


Figure 64: The thread control dialog box.

Each thread is given a name composed of a brief description of the process responsible for the thread and an identification number. The latter is used to provide a unique identification of a particular thread.

To make any changes to a thread, first select it from the list. The current status and execution priority of the thread will then be displayed. The thread may be paused or resumed by clicking the corresponding buttons. Alternatively, clicking Stop will close the thread, terminating the associated process in a relatively graceful manner. Once a process thread has been stopped, it may only be restarted by starting the process again from the beginning. In contrast, a thread that has been paused may always be resumed.

Any threads still running when DigiFlow is exited will be stopped and cannot be restarted.

Note that Normal priority is one step lower than the default priority for most Windows applications, thus preventing a DigiFlow process from unacceptably impacting the overall performance of Windows.

5.3.11 Pause all threads

Toolbutton:
Shortcut:
Related commands:

This tool causes all threads currently running in DigiFlow to be paused until the OK button is pressed. Note that pausing these threads does not prevent you from opening images, changing colour schemes, or even starting new processes: it is only threads that were running at the time the tool was activated that are paused.

Figure 65: Message box indicating all processing within DigiFlow has been paused.

Note: it is advisable not to use this tool while using the File: Live Video features.

5.3.12 Refresh

Toolbutton:
Shortcut:
Related commands:

Causes the currently selected view to be refreshed from the corresponding file, if one exists. This is necessary if you wish to see any changes that have been made to the file since it was originally displayed. This is particularly valuable when editing .dfd files or viewing images made by external programs, for example.

5.3.13 In Parallel

Toolbutton:
Shortcut:
Related commands: `in_parallel(..)`

From DigiFlow version 2.0 (excluding the free version), DigiFlow is able to execute some facilities in parallel when it detects multiple processors, allowing a significant speed-up of these facilities. Note, however, that due to overheads in the parallelisation, plus limited memory and disk bandwidth, the speed-up is less than the increased cpu usage. In general, when doing standard processing, better performance can be achieved by running two jobs at the same time on DigiFlow *without* the In Parallel facility invoked, than will be achieved by running the two jobs in succession using the In Parallel facility.

Note that the current cpu usage is shown in the main DigiFlow status bar at the bottom of DigiFlow. With multiple processors, invoking In Parallel should increase the cpu usage beyond 100%. In version 2.0 the parallelisation is coded explicitly in only a small number of facilities, although it is expected that in future versions parallel execution will be much more widely available within DigiFlow.

5.4 Create

This menu is currently disabled.

5.5 Sequence

This menu is only available when the active window contains a movie, sequence or collection of images. The menu largely replicates the functionality available from the toolbar along the top of these windows.

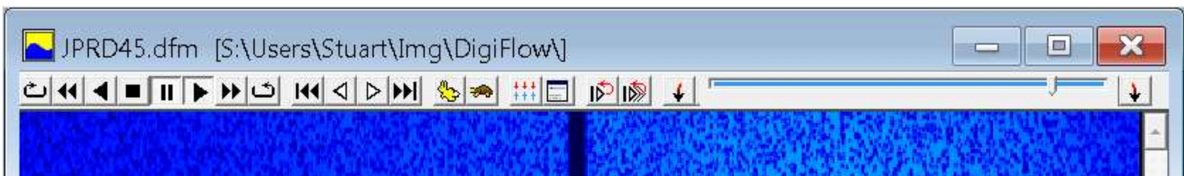


Figure 66: The movie tool bar.

Button	Section	Description
	§5.5.1.12	Play backwards in a loop
	§5.5.1.6	Play backwards quickly (Review)
	§5.5.1.2	Play backwards
	§5.5.1.3	Stop playing sequence
	§5.5.1.4	Pause current movie animation
	§5.5.1.1	Play forwards
	§5.5.1.5	Play forwards quickly (Cue)
	§5.5.1.11	Play forwards in a loop
	§5.5.1.9	Jump to the start
	§5.5.1.8	Move back one frame
	§5.5.1.7	Move forwards one frame
	§5.5.1.10	Jump to the end
	§	Speed up the playback
	§	Slow down the playback
	§5.5.1.13	Synchronise with another view
	§5.5.1.14	Control dialog for additional control over
	§5.5.1.15	Joggle between two frames

	§5.5.1.16	Joggle across multiple frames
	§5.5.1.17	Set first frame for loop
	§5.5.1.18	Frame track bar showing location in sequence
	§5.5.1.19	Set last frame for loop

5.5.1 Animate

5.5.1.1 Play

Toolbutton:
Shortcut: alt+up; Click middle mouse button
Related commands: `animate_view(..,"play")`
Plays the image selector (§3.4) from the current location onwards.

Clicking the middle mouse button will play the movie (unless it is already playing, forwards or backwards, in which case it will pause it). Holding the middle button down while dragging the mouse to the right (left) will move rapidly forwards (backwards) through the movie.

5.5.1.2 Play Backwards

Toolbutton:
Shortcut: alt+down; Double click middle mouse button
Related commands: `animate_view(..,"playbackward")`
Plays backwards the image selector (§3.4) from the current location.

Double clicking the middle mouse button will play the movie backwards. Holding the middle button down while dragging the mouse to the right (left) will move rapidly forwards (backwards) through the movie.

5.5.1.2.1

5.5.1.3 Stop

Toolbutton:
Shortcut:
Related commands: `animate_view(..,"stop")`
Stops the playing of the image selector (§3.4) from the current location. The sequence is left with the final frame played visible, but internally the movie, sequence or collection is returned to its starting point. Playing the movie forwards again will start from the beginning, or backwards will start from the end.

5.5.1.4 Pause

Toolbutton:
Shortcut: alt+space; Click middle mouse button
Related commands: `animate_view(..,"pause")`
Pauses the playing of the image selector (§3.4) from the current location. The sequence is left with the final frame played visible, and play operations will restart from this point.

Clicking the middle mouse button will pause the movie if it is already playing (either forwards or backwards. If the movie is not already playing, then clicking the middle mouse button will play it.

5.5.1.5 Cue

Toolbutton:

Shortcut:

Related commands: `animate_view(..,"cue")`

Plays the image selector (§3.4) at ten times the normal speed (showing only every tenth frame).

5.5.1.6 Review

Toolbutton:

Shortcut:

Related commands: `animate_view(..,"review")`

Plays the image selector (§3.4) backwards at ten times the normal speed (showing only every tenth frame).

5.5.1.7 Step Forwards

Toolbutton:

Shortcut: alt+right; Mouse wheel, middle mouse button

Related commands: `animate_view(..,"step")`

Step forwards one frame in the image selector (§3.4), starting from the current location.

Rotating the mouse wheel will step the movie forwards or backwards, depending on the direction of rotation. Holding the middle button down while dragging the mouse to the right (left) will move rapidly forwards (backwards) through the movie.

If the ctrl key is held down at the same time as using the above mouse controls, then the command is applied to all open windows.

5.5.1.8 Step Backwards

Toolbutton:

Shortcut: alt+left; Mouse wheel, middle mouse button

Related commands: `animate_view(..,"stepbackward")`

Step backwards one frame in the image selector (§3.4), starting from the current location.

Rotating the mouse wheel will step the movie forwards or backwards, depending on the direction of rotation. Holding the middle button down while dragging the mouse to the right (left) will move rapidly forwards (backwards) through the movie.

If the ctrl key is held down at the same time as using the above mouse controls, then the command is applied to all open windows.

5.5.1.9 Start of Movie

Toolbutton:

Shortcut:

Related commands: `animate_view(..,"start")`

Move to the start of the image selector.

5.5.1.10 End of Movie

Toolbutton:

Shortcut:

Related commands: `animate_view(..,"end")`

Move to the end of the image selector.

5.5.1.11 Loop

Toolbutton:
Shortcut:
Related commands: `animate_view(..,"loop")`
Plays the image selector (§3.4) forwards in a continuous loop.

5.5.1.12 Loop backward

Toolbutton:
Shortcut:
Related commands: `animate_view(..,"loopbackward")`
Plays the image selector (§3.4) backwards in a continuous loop.

5.5.1.13 Synchronise

Toolbutton:
Shortcut:
Related commands: `animate_view(..,"sync")`
Causes this image selector (§3.4) to be slaved to another selector. The other selector will provide the time information for synchronous advancement of this selector.

5.5.1.14 Control dialog

Toolbutton:
Shortcut:
Related commands:
This option fires up a dialog providing more detailed control over the animation of the image selector.

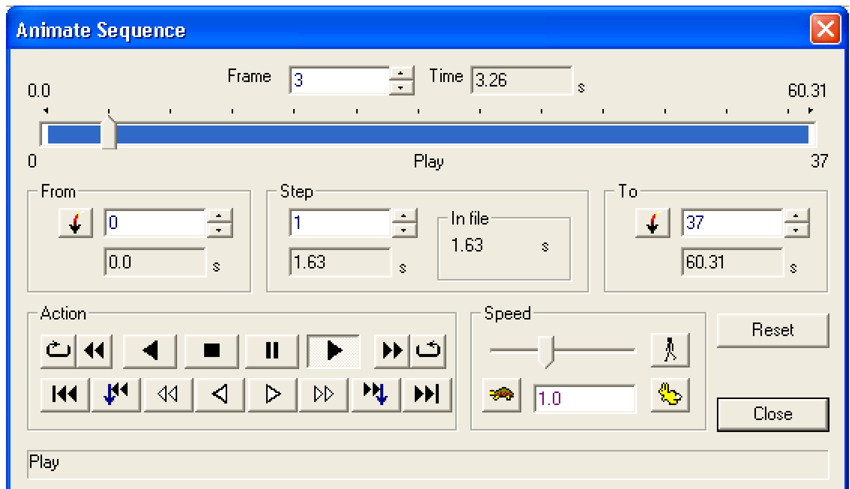


Figure 67: The dialog box that provides detailed control of image animation.

The buttons and controls in this dialog box are similar to those found for the selector timing tab in the Sift dialog in §4.3.1. In addition, the dialog provides more detailed control over the animation speed.

Note: This dialog is modal for free versions of the DigiFlow licence, but modeless (allowing switching between other elements of DigiFlow) for full licences.

5.5.1.15 Joggle between two frames

Toolbutton:
Shortcut:
Related commands: animate_view(..,"joggle2")

This causes the sequence to flip back and forwards between two consecuitive images. This can be useful for looking at the relative movement or relationship between the images.

5.5.1.16 Joggle across multiple frames

Toolbutton:
Shortcut:
Related commands: animate_view(..,"joggle6")

This causes the sequence to repeatedly play multiple (six) images and then jump back to the first. This can be useful for looking at the relative movement or relationship between the images.

5.5.1.17 Set first frame for loop

Toolbutton:
Shortcut:
Related commands: animate_view(..,"setfrom")
animate_view(..,"resetfromto")

Sets the first image (the From image in the control dialog in §5.5.1.14) to be the current frame. Any subsequent playing of the sequence will start at this image. Note the range over which the sequence is played can also be set by selecting a range by holding down the shift button while using the frame trackbar (see §5.5.1.18).

5.5.1.18 Frame track bar

Toolbutton:
Shortcut:
Related commands:

The trackbar shows the current position of the image within the sequence. Grabbing the track bar with the mouse allows rapid movement through the sequence. A range of images may be selected by using the shift key in conjunction with the mouse. Any subsequent play or loop operation will be restricted to the selected range, although the track bar can still be used to access images outside that range. Clicking the bar while holding down the ctrl key will reset the range to the entire sequence.

The range of the track bar may also be set using the buttons detailed in §§5.5.1.17 and 5.5.1.19.

5.5.1.19 Set last frame for loop

Toolbutton:
Shortcut:
Related commands: animate_view(..,"setto")
animate_view(..,"resetfromto")

Sets the last image (the To image in the control dialog in §5.5.1.14) to be the current frame. Any subsequent playing of the sequence will start at this image. Note the range over which the sequence is played can also be set by selecting a range by holding down the shift button while using the frame trackbar (see §5.5.1.18).

5.6 Analyse

5.6.1 Time information

5.6.1.1 Time average

Toolbutton:

Shortcut:

Related commands: `process Analyse_TimeAverage(..)`

Calculates a variety of averages and other statistics for an image selector.

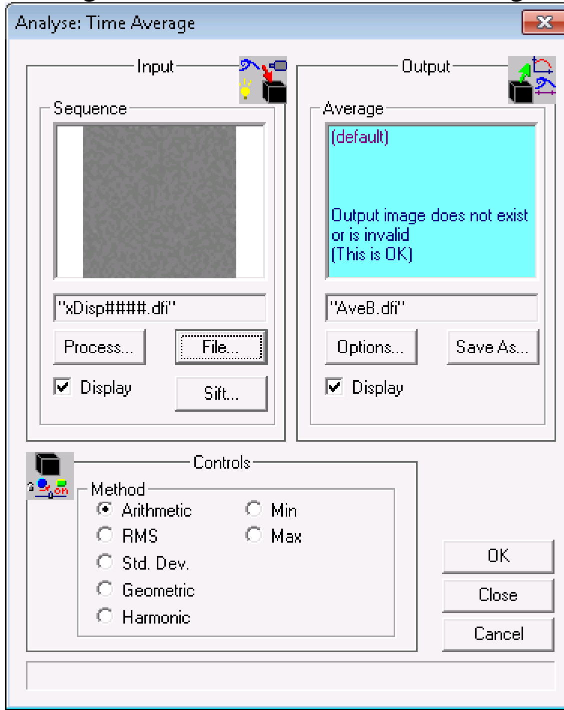


Figure 68: Compute the average of an image selector.

The Input group contains the controls used to determine the image selector to be sampled. This selector may be specified from a file by clicking the File button, in which case the standard Open Image dialog box is produced. Finer control over which parts of the input stream are to be processed are determined via the Sift button; see §4.3. Alternatively, the output of a different process may be utilised by clicking the Process button (refer to §7 on chaining processes for further details).

Note that this tool can process not only images from any DigiFlow supported format, but also velocity fields and other complex data stored in .dfi files. The output stream preserves the data format of the input stream. For example, if the input stream is a velocity field, then the output stream will also contain velocity information.

The time average image is saved to the file specified in the Outputs group by clicking the Save As button. If this process is acting as the source for another process, the Save As button is suppressed (refer to §7 for further details). The colour scheme and compression options to be used for the output stream is set by clicking the Options button (§4.4).

The Method radio buttons select the averaging procedure adopted. Arithmetic returns the standard arithmetic mean, while RMS calculates the root mean square image. The image fluctuations are represented by the Std. Dev. option, while Geometric and Harmonic provide the other forms of averaging. These are summarised in the table below.

Method	Formula	Comments
Arithmetic	$\frac{1}{n}\sum_{i=0}^{n-1} P_i$	This is the standard mean value.
RMS	$\left(\frac{1}{n}\sum_{i=0}^{n-1} P_i^2\right)^{1/2}$	The root mean square value.
Std. Dev.	$\left(\frac{1}{n}\sum_{i=0}^{n-1} P_i^2 - \left(\frac{1}{n}\sum_{i=0}^{n-1} P_i\right)^2\right)^{1/2}$	Standard deviation of the image series.
Geometric	$\left(\prod_{i=0}^{n-1} P_i\right)^{1/n}$	Geometric mean.
Harmonic	$n \Big/ \sum_{i=0}^{n-1} \frac{1}{P_i}$	Harmonic mean.
Min	$\min_{i=0}^{n-1} P_i$	Minimum value.
Max	$\max_{i=0}^{n-1} P_i$	Maximum value.

5.6.1.2 Weighted time average

The weighted time average facility provides an extension to that provided by the more simple averaging. In particular, the following table defines the weighted means. Here, P_i is the intensity at a given pixel from the image at time i, and A_i is the weighting applied to that pixel/image.

Method	Formula	Comments
Arithmetic	$\frac{\sum_{i=0}^{n-1} A_i P_i}{\sum_{i=0}^{n-1} A_i}$	This is the standard mean value.
RMS	$\left(\frac{\sum_{i=0}^{n-1} A_i P_i^2}{\sum_{i=0}^{n-1} A_i}\right)^{1/2}$	The root mean square value.
Std. Dev.	$\left(\frac{\sum_{i=0}^{n-1} A_i P_i^2}{\sum_{i=0}^{n-1} A_i} - \left(\frac{\sum_{i=0}^{n-1} A_i P_i}{\sum_{i=0}^{n-1} A_i}\right)^2\right)^{1/2}$	Standard deviation of the image series.
Geometric	$\left(\prod_{i=0}^{n-1} P_i^{A_i}\right)^{1 \big/ \sum_{i=0}^{n-1} A_i}$	Geometric mean.
Harmonic	$\sum_{i=0}^{n-1} A_i \Big/ \sum_{i=0}^{n-1} \frac{1}{A_i P_i}$	Harmonic mean.
Min	$\min_{i=0}^{n-1} P_i$, only for $A_i \neq 0$	Minimum value.

Max	$\max_{i=0}^{n-1} P_i$, only for $A_i \neq 0$	Maximum value.

The user interface for the weighted mean is similar to that for the simple mean facility (see §5.6.1.1), but has an additional (optional) Weighting selector for specifying a second input stream to provide information with which to construct the weighting A_i. The weighting itself is constructed by the code specified in Expression. This code should return an array of the same dimensions as the input Sequence, or a two-dimensional array of the same size in those two dimensions. This array can be constructed from the input Sequence and (when specified) the input Weighting. If Weighting is specified, then this image is available to the code through the array variable `A` and the compound variable `B`. If Weighting is not specified, then `A` and `B` point to the input Sequence. In both cases the input Sequence is also available through the array variable `P` and compound variable `Q`.

For simple images, the array variables `A` and `P` will have two dimensions, whereas for more complex images `A` and `P` will have more than two dimensions. The information in these multidimensional arrays will also be available through individual components of the compound variables `B` and `Q`. For example, if the first input Sequence contains a velocity field generated by the PIV facility (see §5.6.5.2) then `Q.u` and `Q.v` will contain the two components of the velocity field, and (depending on the options selected during the processing) `Q.Scalar` may contain the vorticity field. Full colour images are supplied as their red, green and blue components with a three-dimensional `P` array: `P[:,:,0]` contains the red component, `P[:,:,1]` contains the green component, and `P[:,:,2]` contains the blue component. For convenience, these are also supplied as `Q.Red`, `Q.Green` and `Q.Blue`. The button may be used to search for or provide information on specific DigiFlow functions. The button may be used to search for or provide information on specific DigiFlow functions.

One common use of the weighted average facility is for computing the temporal average of velocity fields where there may be an incomplete spatial coverage for the velocity at any particular time. In such cases, only points with valid velocity vectors should contribute to the temporal mean. This may be achieved by specifying the sequence of velocity fields to the input Sequence and simply specifying `A<>0;` as the Expression as the PIV subsystem sets the velocity to exactly zero when it is unable to determine a velocity. More sophisticated averaging can be achieved by using the Quality output from the PIV subsystem to construct a more continuous measure of quality to be used as the weighting.

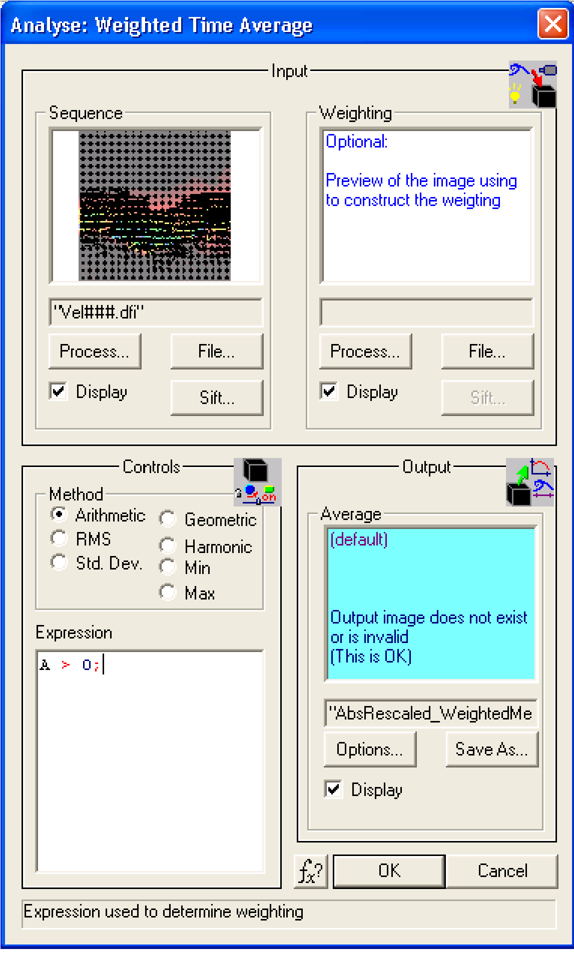


Figure 69: Dialog controlling the weighted time average.

5.6.1.3 Harmonic analysis

Toolbutton:

Shortcut:

Related commands: `process Analyse_HarmonicAnalysis(..)`

One valuable method of analysing periodic signals, such as waves, is by harmonic analysis to determine the phase and amplitude of the component of the signal at a given frequency or set of frequencies.

DigiFlow provides a convenient method of analysing a sequence for a given frequency and harmonics (integer multiples) of that frequency. The Input sequence should span one or preferably more periods of the frequency you wish to analyse. The Sift button should be used to ensure the period being analysed represents a time of steady oscillation. In general, the more images available within this period the better the results.

The Fundamental frequency to be analysed can be specified in a number of ways, with the Period stated in either time or frames, or the Frequency in Hertz (cycles per second) or radians per second. For best results, the input sequence should correspond to an exact multiple of the fundamental period. If the Conform input to period box is checked, then DigiFlow will automatically truncate the input sequence so that it is as close as possible to a multiple of the period.

In addition to analysing the fundamental frequency, DigiFlow can simultaneously analyse harmonics of this frequency, plus the mean (zero frequency) component. Unlike many other DigiFlow facilities, the Harmonic Analysis tool will generate the names of the harmonic files automatically from the files specified in Output for the fundamental. If the name for the

fundamental file is amp.dfi, then the ×2 harmonic is saved in amp[x2].dfi. Similarly for other harmonics. (Note that even if the Number as #### or Compact list boxes are checked in the Open Image dialog then DigiFlow will not treat the number within the square brackets as part of a sequence number.)

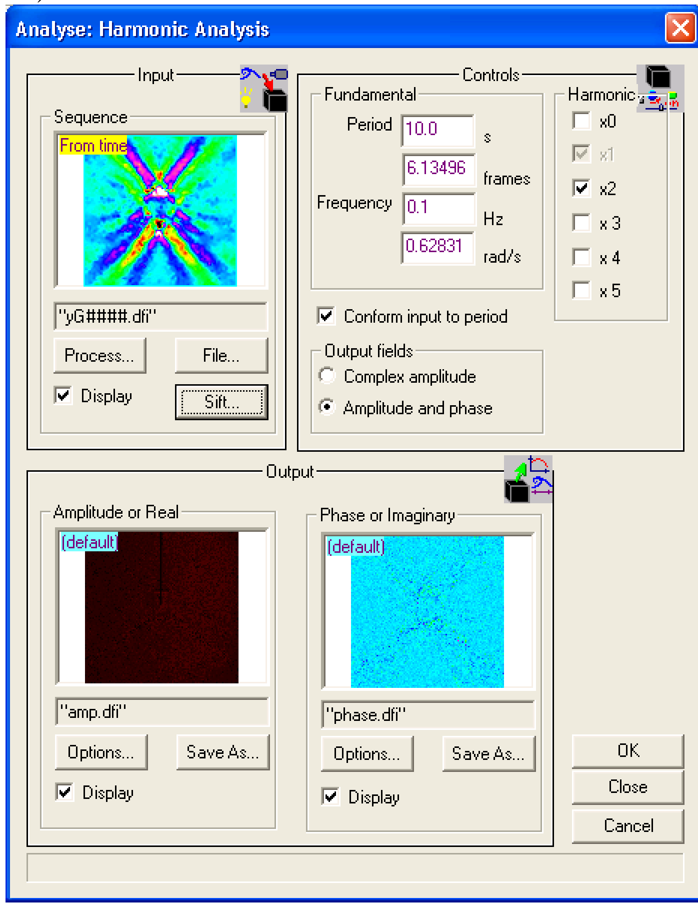
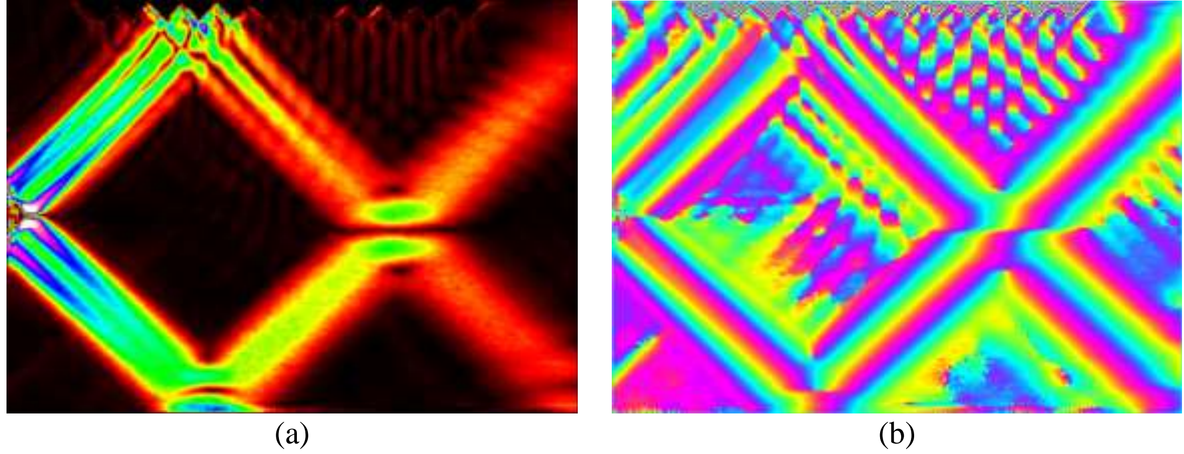


Figure 70: Dialog controlling harmonic analysis within DigiFlow.



(a) (b)

Figure 71: Example of harmonic analysis, showing (a) amplitude and (b) phase of the fundamental frequency of an internal gravity wave.

Once the harmonic analysis has been completed, it is a simple matter to reconstruct the flow field at an instant in time. Alternatively, the field may be decomposed into the different wave components using a Hilbert Transform. A tool for achieving this is found under the Spectral recipes in Tools Transform Recipes. Figure ?? illustrates the results.

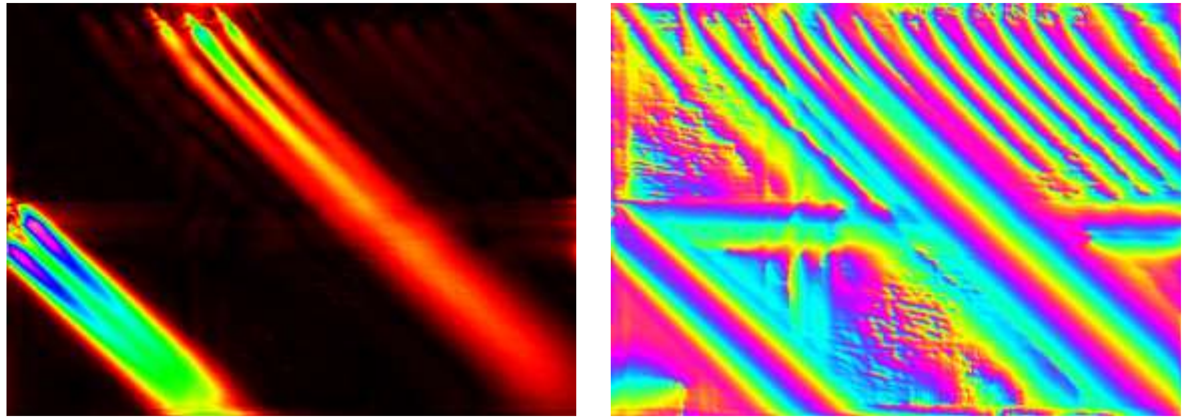


Figure 72: Hilbert Transform yielding waves with $k_x > 0$ and $k_y > 0$. (a) Amplitude. (b) Phase.

5.6.1.4 Time series

Toolbutton:

Shortcut:

Related commands: `process Analyse_TimeSeries(..)`

Extracts a time series of the intensity along some line or curve and forms an image with one spatial and one temporal dimension.

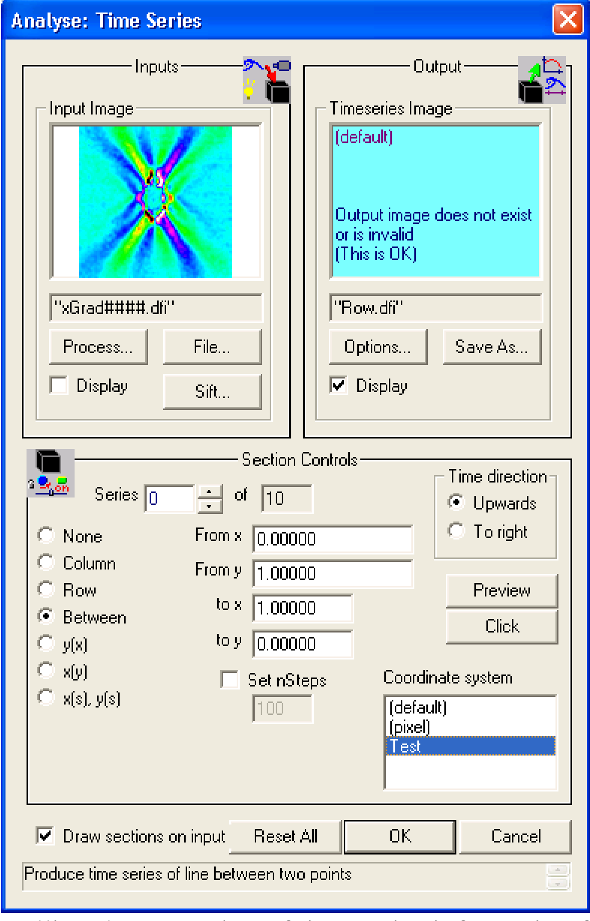


Figure 73: Dialog box controlling the extraction of time series information from an image sequence.

This facility takes a single input stream, and creates one or more output streams. The input streams are normally sequences, while the output streams are normally individual images.

The Input group contains the controls used to determine the image selector to be sampled. This selector may be specified from a file by clicking the File button, in which case the standard Open Image dialog box is produced. Finer control over which parts of the input

stream are to be processed are determined via the Sift button; see §4.3 for further details. Alternatively, the output of a different process may be utilised by clicking the Process button (refer to §5.7.5 for further details).

Note that this tool can process not only images from any DigiFlow supported format, but also velocity fields and other complex data stored in .dfi files. The output stream preserves the data format of the input stream. For example, if the input stream is a velocity field, then the output stream will also contain velocity information.

A number of time series may be generated simultaneous in this manner, each representing a different section through time image, and each written to a different Timeseries Image. You may move between each of the possible time series using the Series edit control and associated spin control. Individual extraction codes are enabled or disabled via the Use check box, while the Reset All button may be used to turn off all and reset all extraction time series.

For each time series, the section through the image may be specified in a variety of ways. The Column and Row radio buttons allow data to be extracted from a given column or row within the image. In both cases, this data is written to the output image as a row of pixels, with each successive time being placed above the previous one.

The Between radio button allows two points to be specified, and the data extracted from the line joining the two points using a specified number of steps. The points may be specified in either pixel or world coordinates by selecting the appropriate Coordinate sytem.

Alternatively, expressions may be given to determine the line or curve along which data is to be sampled. These curves may be specified either as y(x), x(y), or parametrically as x(s) and y(s). Depending on which of these is selected, the formula supplied by the user should be cast in terms of x, y or s. The formula may also include time Time.tNow and/or the frame number Time.fNow (the limits on the selector times and frames are also available through Time.tFrom, Time.tTo, Time.tStep, Time.fFrom, Time.fTo and Time.fStep). In addition, the variable Time.iNow provides an iteration counter. This will always start at zero and increase by one for each image processed (in contrast, the first value for Time.fNow depends on where in the input sequence the sequence to be processed starts, and its increment depends on the stepping between the images to be processed). In all cases, the user can specify the number of sample points and the coordinate system to be used.

The direction of the time axis on the resulting images may be specified using the Time direction group.

Each Timeseries Image created has the samples taken across its width (from first to last left to right), and time increasing from bottom to top. The file that receives this image is specified in the Outputs group by clicking the Save As button. Note that a different destination is provided for each time series activated. If this process is acting as the source for another process, the Save As button is suppressed (refer to §7 for further details). The colour scheme and compression options to be used for the output stream is set by clicking the Options button (§4.4).

In addition to the standard image formats for the output of each time series, this facility supports simple ASCII data files with a .dat extension that provides a more convenient format, including precise details of the pixel or world coordinates from where the data came.

Note that subpixel precision is obtained for all the samples by using bilinear interpolation, where appropriate.

5.6.1.5 Time extract

Toolbutton:

Shortcut:

Related commands: `process Analyse_TimeExtract(..)`

Using a user-specified formula, extract a one-dimensional array of data from each image in a sequence, and use this to construct an image with one spatial and one temporal dimension.

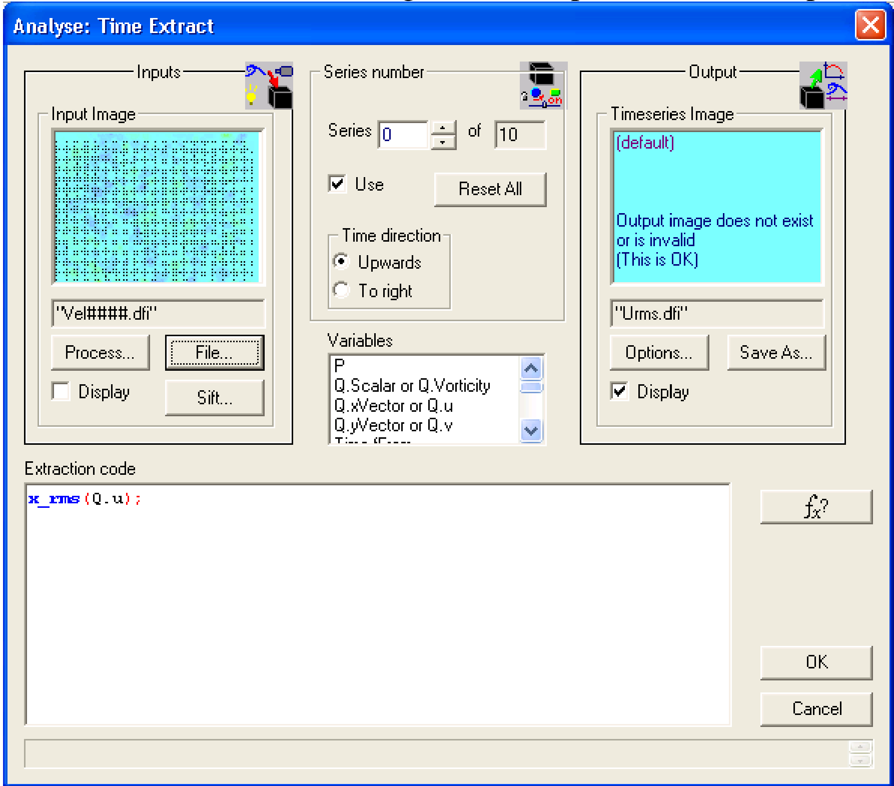


Figure 74: Dialog controlling the extraction of calculated data from an image to form a time series of this data.

This facility takes a single input stream, specified through, and creates one or more output streams. The input streams are normally sequences, while the output streams are normally individual images.

The Input group contains the controls used to determine the image sequence to be sampled. This selector may be specified from a file by clicking the File button, in which case the standard Open Image dialog box is produced. Finer control over which parts of the input stream are to be processed are determined via the Sift button; see §4.3 for further details. Alternatively, the output of a different process may be utilised by clicking the Process button (refer to §7 for further details).

Note that this tool can process not only images from any DigiFlow supported format, but also velocity fields and other complex data stored in .dfi files. The output stream preserves the data format of the input stream. For example, if the input stream is a velocity field, then the output stream will also contain velocity information.

The Extraction code should take the current image and return a one-dimensional array of data to be added to the time series, and the Variables box lists some of the variables describing the image that are available for use in the code; a more comprehensive list may be viewed by clicking the Variables button.. A number of time series may be generated simultaneous in this manner, each with a different Extraction code, and each written to a different Timeseries image. You may move between each of the possible time series using the eSeries edit control and associated spin control. Individual extraction codes are enabled or

disabled via the Use check box, while the Reset All button may be used to turn off all and reset all extraction time series.

Each Timeseries Image created as one axis as time and the other as the ordinal position of the one-dimensional array returned by the Extraction code. The direction of the time axis is specified by the Time direction group. The file that receives this image is specified in the Outputs group by clicking the Save As button. Note that a different destination is provided for each time series activated. If this process is acting as the source for another process, the Save As button is suppressed (refer to §7 for further details). The colour scheme and other output options to be used for the output stream is set by clicking the Options button (§4.4).

The basic image from the input stream is supplied to the Extraction code in the array variable `P`. For simple images this will be a two-dimensional array. However, for more complex image formats (such as velocity fields stored in .dfi files), `P` will contain more than two dimensions. In such cases DigiFlow will also provide the same data split into its individual component two-dimensional arrays in the compound variable `Q`. For example, if the input stream contains a velocity field generated by the PIV facility (see §5.6.5.2) then `Q.u` and `Q.v` will contain the two components of the velocity field, and (depending on the options selected during the processing) `Q.Scalar` may contain the vorticity field. If the input stream contains a DigiFlow drawing (typically one or more .dfd files), then DigiFlow provides the drawing is available through its handle `hD`, in addition to a bitmap version of it in the array variable `P`. Additional drawing commands may be added to the drawing handle, or it may be incorporated into a compound drawing using **`draw_embed_drawing`**`(..)`.

Full colour images are supplied as their red, green and blue components with a three-dimensional `P` array: `P[:,:,0]` contains the red component, `P[:,:,1]` contains the green component, and `P[:,:,2]` contains the blue component. For convenience, these are also supplied as `Q.Red`, `Q.Green` and `Q.Blue`.

DigiFlow also provides time information about the input stream through the `Time` compound variable. Typically this contains `Time.fNow` and `Time.tNow` giving the current frame number and time (in seconds) relative to the start of the entire input stream. An additional variable `Time.iNow` gives an iteration counter that is the frame number relative to the start of those that are actually being processed. Details of the entire input stream are provided through `Time.fFirst`, `Time.fLast` and `Time.tFirst`, `Time.tLast` that provide details of the first and last frame/time that exist in the input stream. Moreover, `Time.fFrom`, `Time.fTo` and `Time.tFrom`, `Time.tTo` provide information about which part of the stream is being processed.

Although the main variables available are listed in the Variables list box this list does not include any additional modifiers for the individual data plane variables beginning with `Q`. These modifiers include the description, scaling and (where appropriate) spacing of the data. A more comprehensive list may be viewed by clicking the Variables button. For further details, refer to the PIV data example in §5.7.2.

The Extraction code may be as simple as returning a subarray (*e.g.* `P[100,10:50]`), or it may be the result of a complex calculation on the image. The fx? button may be used to search for or provide information on specific DigiFlow functions. Examples of more complex processing are given below.

Depth of gravity current

For example, suppose you have an experiment of a gravity current propagating along a channel and want to produce a time history of the depth of the current. The first question is how to measure the depth. There are a number of possibilities.

The simplest measure of the depth would be the height from the bottom to the point where the density fell below some threshold. Suppose we have previously processed a sequence using the dye attenuation facility described in §5.6.3.1 and have an image stream that represents the concentration/density of the current. This could be defined as the number of pixels that exceed some threshold in intensity. In this case the Extraction code would be `y_count(P > 0.1)/y_size(P)`, where the threshold is 0.1. Dividing by `y_size(P)` means that the resulting *depth* will be normalised by the height of the input stream.

A more robust measure would be to use the integral of the concentration over the depth. This is achieved simply by looking at the vertical mean as a function of position by `y_mean(P)`. This gives a measure of the hydrostatic pressure excess at the base of the current.

Concentration power spectrum

As a more complex example, suppose we have a series of LIF images from turbulent flow (these may have been processed using the LIF facilities described in §5.6.3.2), and you would like to know how the power spectrum of some region evolves in time. The region might vary for each image. In particular here we are looking at Rayleigh-Taylor instability and are interested only in the region where the two layers are mixing. In this case the camera was turned on its side. We could rotate the image (*e.g.* using `rotate_image_clockwise(..)` or `transpose(..)`), but for this example we will work in the rotated space with x vertical and y horizontal. A suitable code segment is given below:

```
# Threshold for fluctuations
thresh := 0.05;
# Determine fluctuations
Fluct2 := y_rms(P)^2 - y_mean(P)^2;
# Fird first location where threshold exceeded
iStart := -1;
for i:=0 to x_size(P)-1 {
  if (iStart = -1 and Fluct2[i] > thresh) {
    iStart := i;
  };
};
# Fird last location where threshold exceeded
iEnd := -1;
for i:=x_size(P)-1 to 0 step -1 {
  if (iEnd = -1 and Fluct2[i] > thresh) {
    iEnd := i;
  };
};
# Compute power spectrum within this region
Spect := power_spectrum_column(P[iStart:iEnd,:]);
# Determine and return mean
x_mean(Spect);
```

To determine the region over which we will compute the concentration power spectrum we probe the magnitude of the concentration power spectrum, calculated from the root mean square and mean intensities (concentrations) in the y direction, looking for the first and last columns that satisfy a threshold condition. (Note we could use the function `x_transition_index(..)` in place of the loops for improved computational efficiency.)

5.6.1.6 Time summarise

Toolbutton:

Shortcut:

Related commands: `process Analyse_TimeSummarise(..)`

The Time summarise facility is similar to the Time extract facility, except that it is tailored towards extracting and graphing scalar quantities from an image sequence.

Figure 75 shows the dialog controlling this facility. The Input group contains the controls used to determine the image sequence to be sampled. This selector may be specified from a file by clicking the File button, in which case the standard Open Image dialog box is produced. Finer control over which parts of the input stream are to be processed are determined via the Sift button; see §4.3 for further details. Alternatively, the output of a different process may be utilised by clicking the Process button (refer to §7 for further details).

The Extraction code should take the current image (provided in `P for simple images)` and return a scalar value to be added to the time series. This code may be as simple as returning the intensity at a specific point (*e.g.* `P[100,10]`), or it may be the result of a complex calculation on the image (see below for further details).

The result of the Extraction code is rendered on a graph against time. The method of representing the individual data points is determined by Draw with to specify line or mark type, and Colour to set the colour to be used. Multiple data sets may be plotted on the same graph by specifying different Extraction code, Draw with and Colour for each Series that is selected by Use.

The extracted data are all rendered on the same graph, specified by Output graph in the normal way. In this case, Output graph would normally be a vector format image (.dfd, .emf or .wmf file), but this may be saved as a raster image, if preferred. The x axis represents time, while the y axis is used for the extracted data. The limits on the y axis are set by yMin and yMax. The titles for the two axes are given by x Title and y Title. With fully licensed copies of DigiFlow these may contain LaTeX-like text formatting commands. For example, the string `Dimensionless height $\big(\frac{h}/{\alpha^2H_0}\big)$` would produce the label

$$\text{Dimensionless height}\left(\frac{h}{\alpha^2 H_0}\right).$$

See §3.9 for further details.

A number of time series may be generated simultaneous in this manner, each with a different Extraction code, and each written to a different Timeseries image. You may move between each of the possible time series using the Series edit control and associated spin control. Individual extraction codes are enabled or disabled via the Use check box, while the Reset All button may be used to turn off all and reset all extraction time series.

The basic image from the input stream is supplied to the Extraction code in the array variable `P`. For simple images this will be a two-dimensional array. However, for more complex image formats (such as velocity fields stored in .dfi files), `P` will contain more than two dimensions. In such cases DigiFlow will also provide the same data split into its individual component two-dimensional arrays in the compound variable `Q`. For example, if the input stream contains a velocity field generated by the PIV facility (see §5.6.5.2) then `Q.u` and `Q.v` will contain the two components of the velocity field, and (depending on the options selected during the processing) `Q.Scalar` may contain the vorticity field. Full colour images are supplied as their red, green and blue components with a three-dimensional `P` array: `P[:,:,0]` contains the red component, `P[:,:,1]` contains the green component, and `P[:,:,2]` contains the blue component. For convenience, these are also supplied as `Q.Red`, `Q.Green` and `Q.Blue`. The fx? button may be used to search for or provide information on specific DigiFlow functions.

DigiFlow also provides time information about the input stream through the `Time` compound variable. Typically this contains `Time.fNow` and `Time.tNow` giving the current frame number and time (in seconds) relative to the start of the entire input stream. An additional variable `Time.iNow` gives an iteration counter that is the frame number relative to the start of those that are actually being processed. Details of the entire input stream are

provided through Time.fFirst, Time.fLast and Time.tFirst, Time.tLast that provide details of the first and last frame/time that exist in the input stream. Moreover, Time.fFrom, Time.fTo and Time.tFrom, Time.tTo provide information about which part of the stream is being processed.

The main variables available are listed in the Variables list box. This list does not, however, include any additional modifiers for the individual data plane variables beginning with Q. These modifiers include the description, scaling and (where appropriate) spacing of the data. A more comprehensive list may be viewed by clicking the Variables button. For further details, refer to the PIV data example in §5.7.2.

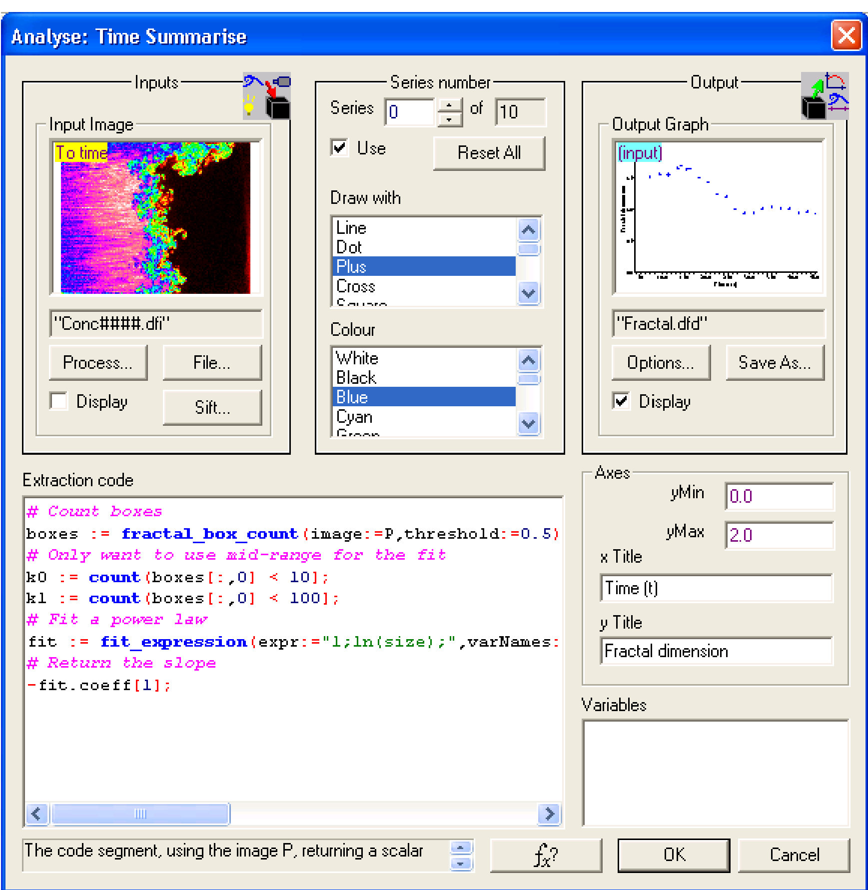


Figure 75: The Analyse: Time Summarise dialog.

Evolution of fractal dimension

In the example shown in figure 75 (the code is repeated below for clarity) we want determine the evolution of the fractal dimension of a contour from a series of LIF images. Here we can make use of DigiFlow's in-built box counting algorithm in conjunction with its ability to fit least squares curves.

```
# Count boxes
boxes := fractal_box_count(image:=P,threshold:=0.5);
# Only want to use mid-range for the fit
k0 := count(boxes[:,0] < 10];
k1 := count(boxes[:,0] < 100];
# Fit a power law
fit := fit_expression(expr:="1;ln(size);",
     varNames:="size;",values:=boxes[k0:k1,0],rhs:=boxes[k0:k1,1],
     rhsExpr:="ln(n);",rhsNames:="n;");
# Return the slope
-fit.coeff[1];
```

The net result (just visible in the preview of the output image) is a time series showing the evolution of the fractal dimension. In reality, it is advisable to plot the individual box count verses box size curves for individual images before embarking on processing such as that described above in order to ensure a power law relationship exists in the range of box sizes selected (here between 10 and 100 pixels).

Evolution of mean intensity along line

Suppose we want to know the mean intensity along some line within an image. Obviously, if the line is simply a line or column then we need simply specify `mean(P[10,:])`, for example, for the mean intensity of column 10. However, if the user wishes to specify the line interactively, we might use

```
if (Time.iNow = 0) {
  # For first iteration, find line and determine points
  hView := get_active_view();
  line := get_mouse_line(hView);
  dx := line.x1 - line.x0;
  dy := line.y1 - line.y0;
  if (abs(dx) > abs(dy)) {
    x := make_array(0,abs(dx));
    y := line.y0 + x*dy/dx;
    x := line.x0 + dx/abs(dx)*x_index(x);
  } elseif (abs(dy) > 0) {
    y := make_array(0,abs(dy));
    x := line.x0 + y*dx/dy;
    y := line.y0 + dy/abs(dy)*x_index(y);
  } else {
    x := make_array(line.x0,1);
    y := make_array(line.y0,1);
  };
};
# Extract values for specified points and determine the mean
points := sample_values(P,x,y);
mean(points);
```

In this case, we use `Time.iNow` to detect the first time through and set up the `x` and `y` arrays to contain the points on the line specified by the user drawing it on the input image stream. Here we rely on the input view being active at the time the code segment starts, allowing its handle to be determined by `get_active_view(..)`.

5.6.2 Ensembles

5.6.2.1 Ensemble mean

Toolbutton:

Shortcut:

Related commands: `process Analyse_EnsembleMean(..)`

It is frequently desirable to determine the behaviour of flows across an ensemble of experiments. The Ensemble Mean facility provides one of the basic building blocks for analysing an ensemble of experiments.

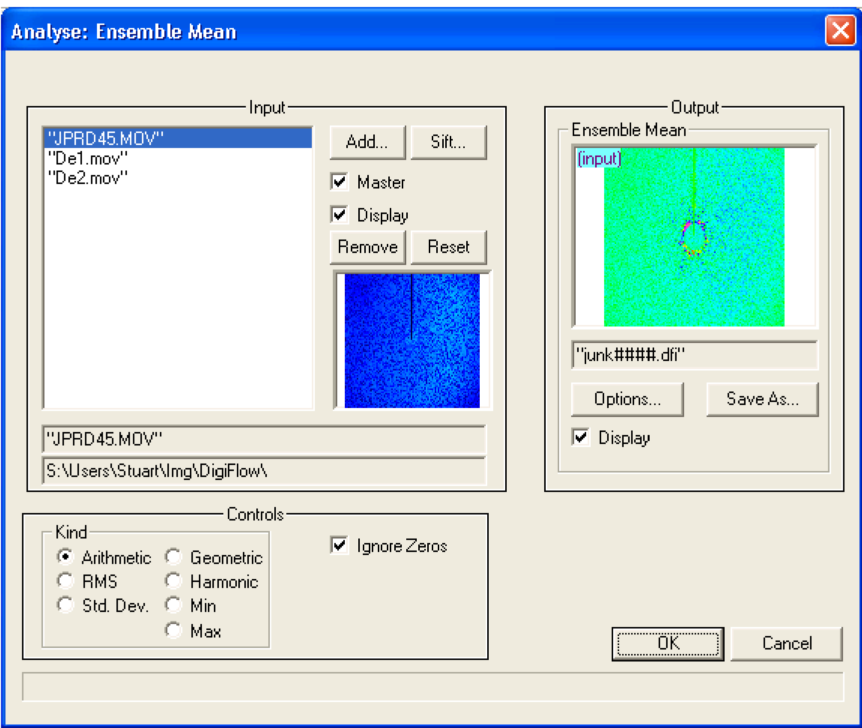


Figure 76: Dialog controlling the calculation of ensemble means.

The interface for specifying an ensemble of experiments differs slightly from the normal mechanism of specifying input streams in DigiFlow in order to provide a more compact and convenient specification process, although this is at the cost of some of the functionality of the standard interface. The Input group provides the various controls needed to specify the members of the ensemble. Add will fire up the standard Open Image dialog for specifying an input stream. Once specified, the name of the input stream is added to the list on the left of the group. An input stream may be sifted by selecting it from the list then clicking the Sift button (see §4.3). While a given member of the ensemble is selected, its name and directory are displayed at the bottom of the Input group, with a preview just above. The Master checkbox indicates if the selected stream is the master (controls the timing, region, *etc.*). This checkbox may be used to specify the currently selected stream as the master, but not to deselect it (to deselect a stream you must select another stream as the master). Streams may be removed from the ensemble by selecting from the list then clicking the Remove button. Alternatively, all members of the ensemble may be removed using the Reset button.

The ensemble mean image is saved to the file specified in the Outputs group by clicking the Save As button. If this process is acting as the source for another process, the Save As button is suppressed (refer to §6 for further details). The colour scheme and compression options to be used for the output stream is set by clicking the Options button (§4.4).

The Controls group allows specification of the type of average to be computed, and whether or not to include zero values in the average. Seven types of average are provided through the Kind group. The meaning of each of these is identical to that given for the time averaging in §5.6.1.1. If the Ignore zeros box is checked, then only those points which are not identically zero are included in the averaging. DigiFlow's synthetic schlieren and PIV facilities both flag missing data with identical zeros, thus checking Ignore zeros provides a convenient way of calculating a mean that is not contaminated by missing data.

5.6.3 Dye images

5.6.3.1 Dye attenuation

Toolbutton:

Shortcut:

Related commands: `process Analyse_DyeAttenuation(..)`

Correct a back-illuminated image for variations in the intensity of the back-illumination. Pre- and post-correction manipulations allow for easy implementation of camera calibration and dye calibration procedures.

Principle of operation

Let us consider polychromatic light from a source with an intensity $i_0(x,y,k) = I_0(x,y)\,J(k)$, where x,y are the location on the source and k the wavenumber of the light emitted. For simplicity, we assume the source colour is independent of the position within the source. Here, $I_0(x,y)$ represents the spatial variation in the intensity of the illumination, while $J(k)$ gives the spectral colour. Assuming the spectral response of a linear monochrome camera (with no black offset) is described by $S(k)$, then the intensity perceived by the camera viewing this source directly is

$$P_0(x,y) = I_0(x,y)\int_0^\infty J(k)S(k)\,dk\,.$$

For a colour camera, there will (typically) be three such expressions, one each for the red, green and blue components (with corresponding $S_R(k)$, $S_G(k)$ and $S_B(k)$ leading to P_{0R}, P_{0G} and P_{0B}). For simplicity, we shall concentrate on a camera yielding a single (monochrome) component. Moreover, we shall assume that the only light received by the camera is that from the source I_0 and that the source is 'in focus'.

Suppose we conduct an experiment using coloured (non-fluorescing) dye of concentration $c(x,y,z,t)$ such that the attenuation of light from the source passing in the z direction through the dye is governed by

$$\frac{di}{dz} = -\alpha c i\,,$$

where $\alpha = \alpha(k)$ describes the colour of the dye. The intensity falling on the camera is therefore

$$i_C(x,y,t,k) = i_0\exp\left(-\alpha\int c\,dz\right) = I_0 J\exp\left(-\alpha\int c\,dz\right) = I_0 J\exp\left(-\alpha\bar{c}L\right),$$

where $\bar{c}(x,y,t) = \frac{1}{L}\int c\,dz$ is the mean concentration along the light ray connecting the source and camera, and $L(x,y,t)$ is the thickness of the region in which we are interested in associating with the mean $\bar{c}(x,y,t)$. Here we have assumed that the light rays are parallel (or nearly parallel) with the z axis.

The intensity perceived by the camera is therefore

$$\begin{aligned} P(x,y,t) &= \int_0^\infty i_C(x,y,t,k)S(k)\,dk \\ &= I_0(x,y)\int_0^\infty J(k)S(k)\exp\left(-\alpha(k)\bar{c}(x,y)L\right)dk \end{aligned} \qquad (1)$$

We proceed by considering three classes of dyes, catagorised by $\alpha(k)$ in conjunction with the $J(k)S(k)$ combination.

Ideal dyes

If the dye is neutral density ($\alpha(k) = \alpha_0 = const$) then (1) reduces to

$$Q = \frac{P}{P_0} = \exp\left(-\alpha_0 \bar{c} L\right).$$

Here we have defined Q as the normalised intensity P/P_0. Similarly, if the light is monochromatic of wavenumber k_1 at which $\alpha(k_1) = \alpha_1$, or the camera's response is monochromatic of wavenumber k_1, then

$$Q = \frac{P}{P_0} = \exp\left(-\alpha_1 \bar{c} L\right).$$

We shall call these two situations as the 'ideal' case. The first can be achieved using a black dye such as nigrosin, while the second can be achieved either using a monochromatic light source or by placing in front of the camera lens.

In this 'ideal' case, if L is known *a priori*, then we can readily determine the mean concentration as

$$\bar{c} = -\frac{1}{\alpha_i L} \ln Q,$$

where the combination $\alpha_i L$ is typically determined by calibration. Similarly, if $\bar{c}$ is know *a priori* (typically by setting $c(x,y,z,t) = c_0 = const$), then we can use the same relation to determine the thickness (depth) of the layer from

$$L = -\frac{1}{\alpha_i \bar{c}} \ln Q.$$

If L is constant in time but varies in space, it is often more convenient to determine this from an image of a constant concentration of dye, rather than attempting to measure it everywhere. Suppose the camera perceives an image of intensity P_L when the experiment is filled with a constant concentration of dye c_L. From this we can calculate

$$L = -\frac{1}{\alpha_i c_L} \ln \frac{P_L}{P_0} = -\frac{1}{\alpha_i c_L} \ln Q_L,$$

where $Q_L = P_L/P_0$, and substitute back to determine

$$\bar{c}\left(x, y, t\right) = c_L \frac{\ln Q}{\ln Q_L}.$$

Non-ideal dyes

If the dye is not ideal (*i.e.* it is neither neutral density nor illuminated by a monochromatic light source) then additional calibration is required. Recall that the camera perceives

$$Q = \frac{P}{P_0} = \frac{\int_0^\infty J\,S \exp\left(-\alpha \bar{c} L\right) dk}{\int_0^\infty J\,S\,dk}.$$

For a given illumination and camera response, we can write $Q = g\left(\bar{c}L\right) = g\left(\varphi\right)$ where $g(\varphi)$ (which is bounded above by one) characterises the response of the dye as

$$g(\varphi)=\frac{\int_0^\infty J\,S\exp(-\alpha\varphi)\,dk}{\int_0^\infty J\,S\,dk}.$$

If $g(\varphi)$ is invertible then we can determine the concentration as

$$\bar{c}=\frac{1}{L}g^{-1}(Q).$$

As we saw above, for an ideal dye $g^{-1}(Q)$ is $-\frac{1}{\alpha_i}\ln Q$.

In general, however, we may not have a detailed knowledge of the spectrum of the light source $J(k)$, the spectral response of the camera $S(k)$ or of the dye absorption $\alpha(k)$, but must instead determine by calibration $g(\varphi)$ or, more usefully, the inverse $g^{-1}(Q)$.

Suppose we determine, through a calibration procedure, that $g^{-1}(Q) \approx Dye(Q)$, then we can proceed to compute $\bar{c}$ or L. In particular, if L is known *a priori*, then

$$\bar{c}=\frac{1}{L}Dye(Q),$$

while if $\bar{c}$ is know *a priori* then

$$L=\frac{1}{\bar{c}}Dye(Q).$$

As before, if L is constant in time but varies in space we can compute $\bar{c}$ by introducing a calibration image P_L of a uniform dye concentration c_L to obtain

$$\bar{c}=c_L\frac{Dye(Q)}{Dye(Q_L)},$$

where $Q_L = P_L/P_0$.

Band-pass dyes

Some dyes can be approximated as being transparent to some wavelengths of light while strongly filtering other wavelengths. The spectral response of such a dye could be approximated by $\alpha(k) = \alpha_0 + \alpha_1 H(k-k_0)$, where $H(\cdot)$ is the Heaviside step function, α_0 (≥ 0) and α_1 are constant attenuation rates and k_0 is a constant wavenumber. If, further, we assume the product $J(k)S(k)$ is constant for $k_1 \leq k \leq k_2$¸and zero outside this range, then for such a dye $g(\varphi)$ becomes

$$\begin{aligned}g(\varphi)&=\frac{\int_{k_1}^{k_0} J\,S\exp(-\alpha_0\varphi)\,dk+\int_{k_0}^{k_2} J\,S\exp\left(-(\alpha_0+\alpha_1)\varphi\right)dk}{\int_{k_1}^{k_2} J\,S\,dk}\\&=e^{-\alpha_0\varphi}+\frac{k_2-k_0}{k_2-k_1}e^{-\alpha_1\varphi}\end{aligned}$$

In the limits of $\alpha_1 \gg \alpha_2$ and of $\alpha_1 \sim -\alpha_0$, then $g(\varphi)$ is well approximated by a constant plus a decaying exponential. The constant term is due to little attenuation of light over some bandwidth, while the exponential is due to a rapid (nearly constant) attenuation of light over the remainder of the spectrum. As seen by a number of previous authors, this model is a good approximation to the behaviour of some food colourings when illuminated by white light.

DigiFlow interface

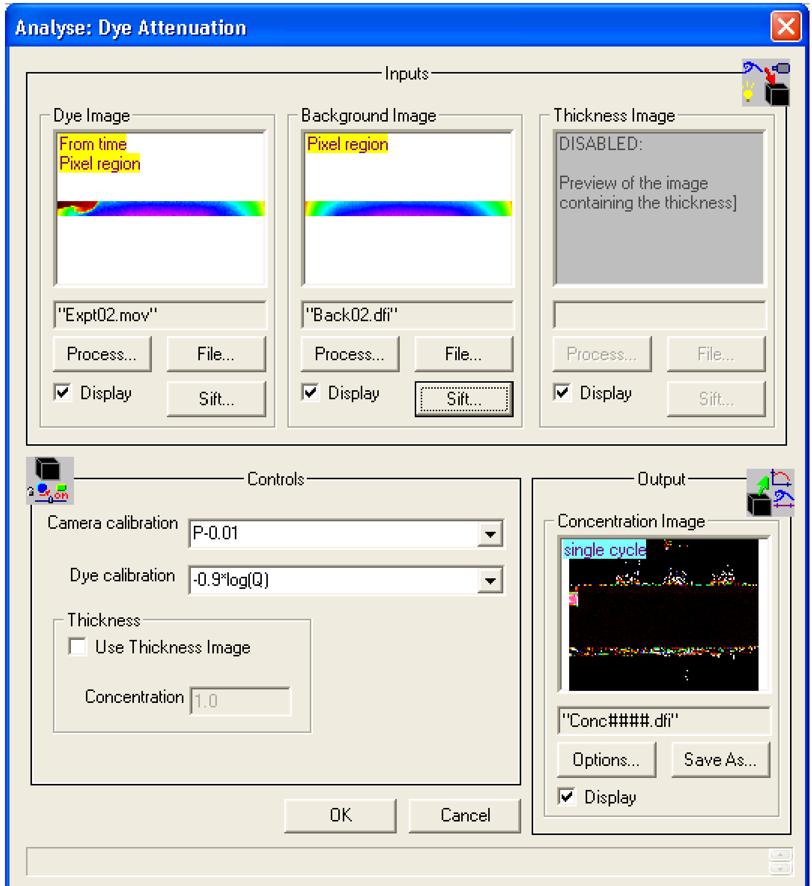


Figure 77: Analysis of variations in background illumination to determine the dye concentration.

This process takes either two or three source image selectors, depending on whether the base optical thickness needs to be calculated.

The Dye Image group determines the image selector to be corrected. This image selector may be selected from a file by clicking the File button, in which case the standard Open Image dialog box (§4.1) is produced. Precise details of the region and times to be used may be set using the Sift button (§4.3). Alternatively, clicking the Process button will allow a source process to be used (refer to §7 on chaining processes for further details).

The Background Image group determines the image selector containing the background illumination. Only the first image will be used if an image selector containing multiple images is selected, although the particular image from a sequence may be specified using the Sift button. As with the Dye Image group, clicking File activates the Open Image dialog box (§4.1), whereas clicking Process allows a source process (§7) to be used.

The resulting image selector is saved to the file specified in the Concentration Image group by clicking the Save As button. If this process is acting as the source for another process, the Save As button is suppressed (refer to §7 for further details).

The Controls group allows user specification of the camera and dye calibration.

The Camera calibration is applied to both the Dye Image and Background Image prior to their processing. The calibration is specified as a function of the intensity in the input image selectors, represented in the expression as the variable P (upper case). Note that regardless of the format of the input selectors, all processing is performed in floating point arithmetic and normally the images will be scaled between an intensity of 0.0 for the darkest parts and 1.0 for the brightest parts. Thus the default `P - 16/255` would set an intensity of 16 in an eight-bit greyscale image to zero. Refer to §8 for further details on the interpreter.

The Dye calibration mapping is applied to the image(s) resulting from this procedure. The mapping function should be specified in terms of the intensity of the corrected image. Here this corrected image intensity is expressed through the variable Q (upper case), which is again in the range 0.0 to 1.0. The default for this calibration is `dye_deal(Q)`, which is simply `-ln(Q)`

The normal processing undertaken by this feature may be described as

$$P_{Conc} = Dye\left(\frac{Camera(P_{Dye})}{Camera(P_{Back})}\right),$$

where $Camera$`(..)` represents the Camera calibration, Dye`(..)` represents the Dye calibration, P_{Dye} is the Dye Image, P_{Back} is the Background Image and P_{Conc} is the Concentration Image. The result of $Camera(P_{Dye})/Camera(P_{Back})$ is what is provided in the variable Q, thus $P_{Conc} = Dye(Q)$.

However, strictly speaking, P_{Conc} is proportional to the integral of the dye concentration over the length of the light ray seeing dye as it passes through the flow. Thus, if the length of this light ray varies (*e.g.* due to tank geometry or camera parallax), the P_{Conc} image is contaminated by this variation.

As noted above, by using an additional image, P_L, of the tank containing a uniform concentration of dye C_L, it is possible to correct for this variation in the length of the light rays. In such a case the required processing is

$$P_{Conc} = C_{Uniform}\, Dye\left(\frac{Camera(P_{Dye})}{Camera(P_{Back})}\right) \Big/ Dye\left(\frac{Camera(P_{Thick})}{Camera(P_{Back})}\right).$$

This more advanced processing is enabled by checking Use Thickness Image. This then enables the Thickness Image group to determine the image selector containing the background illumination. Only the first image will be used if an image selector containing multiple images is selected. As with the Dye Image group, clicking File activates the Open Image dialog box, whereas clicking Process allows a source process to be used.

5.6.3.2 Light Induced Fluorescence (LIF)

LIF, often referred to as Laser Induced Fluorescence but more generally can stand for Light Induced Fluorescence, describes the family of techniques where a sheet of light is used to stimulate emission from a fluorescent dye. Typically this dye is dissolved in the fluid at very low concentrations, rendering it a passive tracer, but which is used to tag some other species (*e.g.* salt concentration) thereby providing a means of visualising and quantifying an otherwise invisible component of the flow.

Fluorescent dyes are often used in fluids experiments to obtain an image of the concentration field on a single plane of a flow. The name often given to such techniques is LIF or Laser Induced Fluorescence. However, the use of a laser is not obligatory, and white light may be used to produce comparable results, provided the colour temperature of the light source is sufficiently high. Xenon arc lamps, for example, provide an excellent and safer alternative to the high cost of lasers.

LIF principles

The fluorescent dyes used in LIF typically absorb energy from incident light over a range of wave lengths and radiate it at a single or well defined range of wave lengths. Typically the absorption in the range of wave lengths radiated is relatively small so that radiated light passing through regions of fluid containing the fluorescent dye is not attenuated significantly by that dye. Clearly the illuminating light must attenuate as it passes through the dye. For most useful dyes the efficiency of this fluorescence is relatively high so that only very weak solutions are required and the attenuation of the illuminating light is small.

If the flow is illuminated by monochromatic light (such as a laser, or at least coloured light with a narrow power spectrum) with a wave length significantly different from that of the fluoresced light, then it is possible to eliminate the effect of any light scattered directly from the experimental apparatus or contaminants in the water by introducing a filter in front of the camera to cut the wave length(s) of the light source. However for reasons of cost, availability and safety, a laser was not employed for these experiments. Thus our LIF images contain a component of directly scattered light despite efforts to minimise this.

In all the LIF experiments reported here, sodium fluorescein was used as the fluorescent dye. Its choice was based on its high efficiency, low cost and relative safety. The light fluoresced typically appears green, with the dye responding better to the blue end of the visible spectrum (this is one of the reasons the blue-white light of the arc lamp was better than the yellow-white light of the halogen light source).

Correction for illumination

As mentioned above, as the illuminating light sheet passes through the dyed fluid, some of the light is absorbed thus reducing the intensity of the light sheet. In addition, the light sheet will typically diverge slightly, effectively reducing the intensity further. In order to obtain quantitative information about the density field (as marked by dye concentration) it is necessary to correct the LIF images for this attenuation and divergence. In this subsection we briefly outline the technique used in this work for performing this correction.

Consider an image $p = p(\boldsymbol{x})$ of a flow containing a fluorescent dye of concentration $C = C(\boldsymbol{x})$. We define a virtual light sheet $P = P(\boldsymbol{x})$ such that

$$P = C\,P. \tag{2}$$

Assume that the attenuation of the virtual light sheet as it passes through the dye can be described by

$$\frac{dP}{ds} = -\sigma CP = -\sigma p \tag{3}$$

where s describes the path of the light rays and $\sigma = \sigma(\boldsymbol{x})$ is the attenuation of the virtual light sheet. Suppose we have a calibration image p_0 of a constant concentration C_0. Now we may estimate the spatial structure of the attenuation from

$$\tilde{\sigma} = -\frac{1}{C_0 \hat{p}_0}\frac{d\hat{p}_0}{ds}, \tag{4}$$

where $\hat{p}_0$ is the least squares fit of

$$\hat{p}_0 = a_0 e^{a_1 s + a_2 s^2 + a_3 s^3} \tag{5}$$

to the calibration image p_0 (it is often necessary only to include the linear term in the exponential). Using our estimate of the attenuation we can calculate an estimate $\tilde{P}_0$ (say) for the virtual sheet from

$$\frac{d\tilde{P}_0}{ds} = -\tilde{\sigma} p_0 \tag{6}$$

and thus obtain our estimate of the concentration field

$$C = \frac{p}{\tilde{P}} C_0, \tag{7}$$

where $\tilde{P}$ is the estimate for the virtual light sheet evaluated from equation (3).

We may determine how accurate this process is by performing this process on the calibration image, and then comparing the result with the known virtual sheet $P_0 = p_0/C_0$ to obtain the defect ratio

$$R_{defect} = \frac{\tilde{P}_0}{P_0} = \frac{\tilde{P}_0}{p_0} C_0 . \tag{8}$$

In some cases we may wish to adjust our calculation for other images using this defect ratio by determining

$$C = \frac{p}{\tilde{P}} \frac{\tilde{P}_0}{p_0} C_0 . \tag{9}$$

This approximation guarantees perfect reconstruction of the calibration image, but does not necessarily ensure an improved concentration field for other images.

Another technique that can be useful is to use R_{defect} to improve the estimate to the decay law fit rather than as a direct modification to the concentration field. To achieve this, a least squares fit of the same form as (5) is applied to R_{defect} (which we would hope was a constant), and the coefficients a_1, a_2, *etc.*, are used as a correction to those obtained from (5). Applying this correction iteratively will ensure that in the mean R_{defect} is unity.

LIF processing in DigiFlow

The processing of LIF images in DigiFlow is somewhat more sophisticated than that described above, extending the basic idea to include multiple point light sources, distributed light sources, and additional models for the behaviour of light rays.

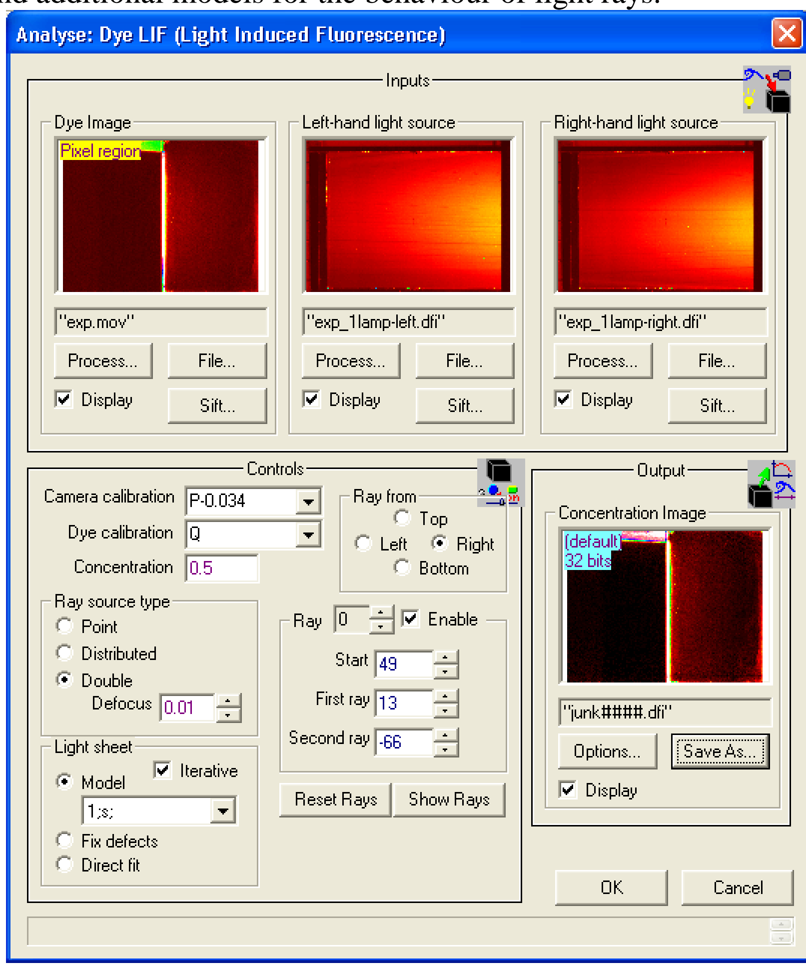


Figure 78: Dialog controlling correction of LIF images.

The example illustrated in figure 78 is for Rayleigh-Taylor instability. In this case, the camera was rotated by 90 degrees so that the initially dense layer is to the left and the light layer to the right. The flow was illuminated from below by a sheet of white light. Fluorescent

dye (disodium fluorescein) was present in the upper layer. The two layers were initially separated by a barrier at half the depth of the tank.

The Dye Image input image stream is the raw footage of the experiment. The attenuation and divergence of the illuminating light sheet as it passes through the fluorescent dye is clearly visible. In this case, the light sheet is generated by a pair of 300W arc lamps, each effectively a point source. The Left-hand light source and Right-hand light source input streams give images of the two separate light sources illuminating the tank when it contains a uniform concentration of dye. These images are used for calibration purposes. Each of these three input selectors is specified in the normal manner using the associated File and Sift buttons.

The Controls group contains the various parameters that affect the modelling of the light passing through the dye. For this correction procedure to operate effectively, it is important that the experiment is carefully set up, that there are no stray reflections reaching the camera, and that all the necessary details are recorded at the time of the experiment.

The Camera calibration should specify the relationship between the digitised values and real intensities. With most modern scientific CCD and CMOS cameras the relationship is close to being linear. However, 'black' seldom digitises to zero. Here, we assume a linear relationship with black digitising to a value of 0.034. A number of methods for determining this black level are described in §6.1.

Provided the concentration of the fluorescent dye is sufficiently low, then the assumption that the fluoresced signal is linear in its concentration is reasonable. Deviations from this may be entered in the Dye calibration control.

The Concentration value is the (arbitrary) concentration used in the calibration images Left-hand light source and Right-hand light source. Note that these images are used not only to calibrate the response of the light sheet as it passes through the dye, but also to calibrate the intensity of the light sheet entering the dyed region. In this latter context it is important that these images (and also the experimental images) extent to the tank boundaries where the light first enters the dye.

The Ray from group indicates the direction from which the light rays enter the image. Here it is the bottom of the tank which corresponds to the right-hand side of the image. The Ray source type allows selection between single point light sources, distributed light sources, and double point light sources. (The Right-hand light source input stream is enabled only for the last of these.) The Defocus control recognises that the lights might not in fact be true point sources and that they will become slightly defocused as they pass through the flow.

Alongside the origin of the light rays is their direction which will not normally be aligned exactly with the pixel coordinates. Indeed, the light rays will typically be diverging. The Ray *n* group provides a means of specifying the orientation of light rays. This process is typically achieved by capturing an additional image of the uniform concentration in which a grid has been imposed on the light sheet in order to show clearly the direction of the light rays. This additional image (or two additional images when double light sources are used) is temporarily loaded into the Left-hand light source stream. Clicking Show rays then superimposes the ray definitions from the Ray *n* group on that image. Subsequent use of the controls within this group allows interactive specification of the orientation of the rays. It is recommended that three or four such light rays are specified as a minimum.

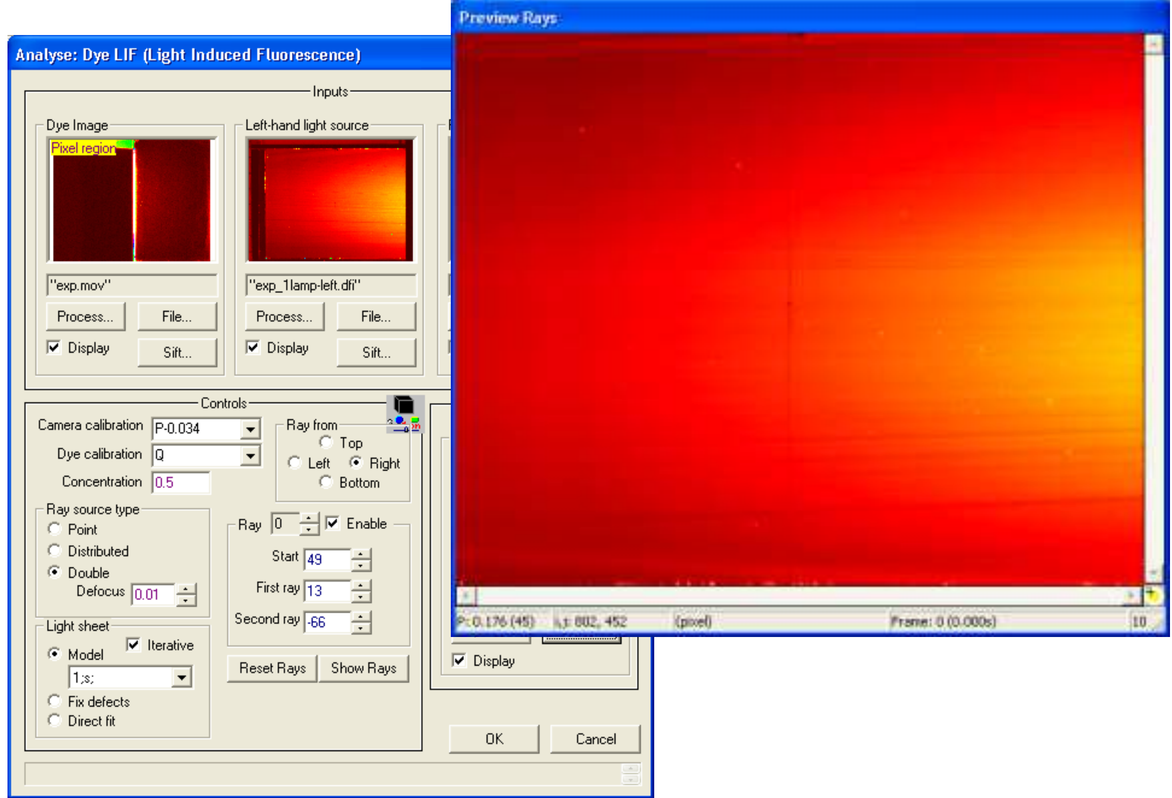


Figure 79: Light ray definition for LIF processing.

Up to ten sets of light rays may specified. Each set contains either one or two directions, depending on the Ray source type. The rays in each set share a common Start, but can have different rotations through the First ray and Second ray controls. The ray definitions may be reset using the Reset Rays button.

Modelling of the light sheet can be handled in a number of different ways using the Light sheet group. Both Model and Fix defects begin by fitting a model decay law (using least squares) to the calibration image(s) to determine the relationship between fluorescence and attenuation. If Iterative is checked, then this process is performed iteratively to improve the model by taking into account the directions of the light rays. In many cases the linear model 1;s; (corresponding to an exponential decay of light through the calibration image) is most appropriate. However, in some circumstances higher order terms can improve performance (a combination of visual inspection and tests for mass conservation should be used to determine the optimal model).

The difference between Model and Fix defects is that the latter compares the calculation performed on the calibration image with the 'known' constant concentration it contains and develops a multiplicative correction to force a uniform concentration in the output. This correction may be appropriate in cases where optical imperfections alter the apparent lighting in a static manner that does not coincide with the simple attenuation modelling.

Calculations using the Direct fit light sheet model are similar to the other two, but rather than fitting an exponential decay law, the data in the calibration image is used directly, pixel by pixel, to determine the relationship between fluorescence and absorption. This approach is likely to lead to a higher noise level in most situations, but may have advantages in cases with complex optical effects such as reflections.

For most good quality experiments, the combination of Fix defects with 1;s; and Iterative selected will yield the best results.

Finally, the output stream may be specified in the normal way using the Save As and Options buttons.

5.6.4 Synthetic schlieren

Theory

Synthetic schlieren is a novel technique for producing qualitative visualisations of density fluctuations and for obtaining quantitative whole-field density measurements in two-dimensional density-stratified flows. This set of techniques is outlined in detail in Dalziel *et al.* (2000) (and a subset in Sutherland *et al.* 1999). In this section, we discuss only the most advanced of these techniques, 'pattern matching refractometry', and how this may be applied to provide accurate quantitative measurements of a two-dimensional density field.

While synthetic schlieren has its origins in the classical optical schlieren and moiré fringe techniques, synthetic schlieren is much simpler to set up than the classical schlieren and interferometry methods, and provide useful information in situations where shadowgraph is of little or no value. Moreover, they may be set-up to analyse much larger domains than is feasible with the classical approaches, and do not require high quality optical windows in the experimental apparatus. Ultimately the greatest strength of these techniques is the ability to extract accurate, quantitative measurements of the density field.

The basic setup for synthetic schlieren is illustrated in figure 80. An illuminated mask (normally simply a piece of paper printed with a pattern) with a strong pattern is placed to the rear of the experiment. This mask is then viewed by the video camera looking *through* the experiment.

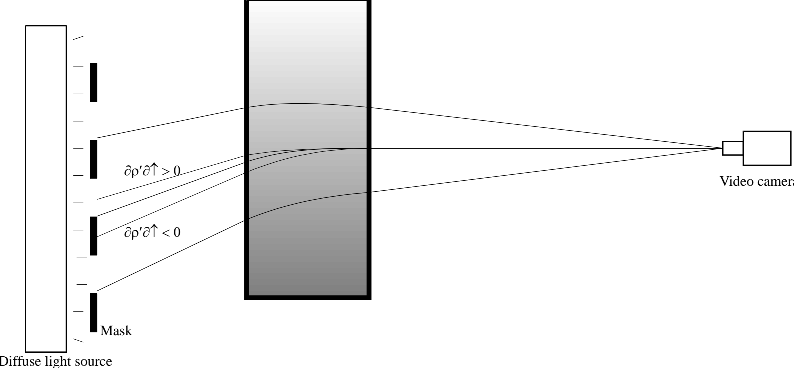


Figure 80: Basic setup for synthetic schlieren. An illuminated textured mask is placed on one side of the experiment, and is viewed by the camera on the other side of the experiment.

The key idea behind synthetic schlieren is the bending of rays of light by fluctuations in the refractive index field. A detailed description of this process is required in order to make quantitative measurements. This description may start from a number of points. Sutherland *et al.* (1999) analysed a ray of light by invoking Snell's law. Here we shall start from Fermat's variational principle for the behaviour of light in an inhomogeneous medium

$$\delta \int n(x, y, z)\, ds = 0, \tag{10}$$

where s is oriented along the light ray and $n(x,y,z)$ is the refractive index field (the ratio of the speed of light through a vacuum to that through the medium). We select our coordinate system (x,y,z) with x along the length of the tank, y across the width (the direction in which variations in the flow are negligible) and z vertically upward.

Rather than solving the full variational problem, we restrict ourselves to rays of light which always have a component in the y direction so that their paths may be described by $x = \xi(y)$ and $z = \zeta(y)$. This restriction is simply a requirement that light is able to cross the tank, a fundamental requirement for synthetic schlieren. The variational principle then gives rise to a

pair of coupled ordinary differential equations (Weyl 1954) relating the light path to the gradients of n in the x–z plane:

$$\frac{d^2\xi}{dy^2} = \left[1 + \left(\frac{d\xi}{dy}\right)^2 + \left(\frac{d\zeta}{dy}\right)^2\right]\frac{1}{n}\frac{\partial n}{\partial x}, \qquad (11a)$$

$$\frac{d^2\zeta}{dy^2} = \left[1 + \left(\frac{d\xi}{dy}\right)^2 + \left(\frac{d\zeta}{dy}\right)^2\right]\frac{1}{n}\frac{\partial n}{\partial z}. \qquad (11b)$$

For synthetic schlieren we are interested primarily in light rays which remain approximately parallel to the y direction. Under this restriction the terms $(d\xi/dy)^2$ and $(d\zeta/dy)^2$ may be neglected, effectively decoupling (11a) and ((11b). For a two-dimensional flow where there are only weak varitions in density (and hence weak variations in the refractive index) along the ray path, we may integrate these expressions across the width W of the tank to obtain the path of the light ray across the tank:

$$\xi = \xi_i + y\tan\phi_\xi + \tfrac{1}{2}y^2\frac{1}{n}\frac{\partial n}{\partial x}, \qquad (12a)$$

$$\zeta = \zeta_i + y\tan\phi_\zeta + \tfrac{1}{2}y^2\frac{1}{n}\frac{\partial n}{\partial z}, \qquad (12b)$$

where ξ_i, ζ_i describe the incident location and $\tan\phi_\xi = d\xi/dy(y{=}y_0)$ and $\tan\phi_\zeta = d\zeta/dy(y{=}y_0)$ describe the horizontal and vertical components (respectively) of the angle at which the light ray enters the tank (measured relative to the y direction).

With synthetic schlieren, we are interested in how an image of a mask placed some distance B behind the tank appears to change as the result of flow-induced refractive index variations relative to the refractive index variations in the absence of the flow. Specifically we wish to analyse the changes in the image formed by the camera as a shift in the origin of the light ray reaching the camera. By back tracking the light rays received by the camera the apparent shift $(\Delta\xi,\Delta\zeta)$ in the origin of the light ray is given by

$$\Delta\xi = \xi_i + y\tan\phi_\xi + \frac{1}{2}y^2\frac{1}{n}\frac{\partial n}{\partial x}, \qquad (13a)$$

$$\Delta\zeta = \zeta_i + y\tan\phi_\zeta + \frac{1}{2}y^2\frac{1}{n}\frac{\partial n}{\partial z}. \qquad (13b)$$

Here we have decomposed the refractive index field n into $n_0 + n_{base} + n'$, where n_0 is the nominal refractive index of the medium (*e.g.* $n_0 = 1.3332$ for water), n_{base} represents spatial variations associated with the "known" base state (*e.g.* the changes introduced by adding a quiescent linear background stratification) and n' is the variation caused by the flow under consideration (*e.g.* the internal wave field). In obtaining (13) we have assumed the variations n_{base} and n' in the refractive index field are small compared with the nominal value n_0. (As we will see below, a correction is necessary to take into account the refractive index contrasts between air, the material the tank is made of and the working fluid.)

In many cases, it is more convenient to consider the apparent displacement of the origin of the light rays in terms of their projection on the experiment in the absence of any fluctuations in the density field. This then allows us to use a common coordinate system for both the coordinates within the experiment and for the texture mask located behind the experiment. Taking the distance between the texture mask and the camera as L, and assuming that the experiment is 'thin' (*i.e.* $W/L \ll 1$), then we may use simple projective geometry to show that the apparent displacements in experiment coordinates are

$$\Delta\xi_{exp} = -\frac{1}{2}\left(1-\frac{B+\frac{1}{2}W}{L}\right)W\left(W+2B\right)\frac{1}{n_0}\frac{\partial n'}{\partial x}, \tag{14a}$$

$$\Delta\zeta_{exp} = -\frac{1}{2}\left(1-\frac{B+\frac{1}{2}W}{L}\right)W\left(W+2B\right)\frac{1}{n_0}\frac{\partial n'}{\partial z}. \tag{14b}$$

Here we have defined the experiment coordinate system to be at the mid-plane of the experiment. If $W/B \ll 1$ or $(B+\frac{1}{2}W)/L \ll 1$, then the precise location of the coordinate system within the experiment is unimportant. Note also that the optical gain provided by increasing B is greatest for large L.

The above expression, however, ignores the effect of the refractive index change between the tank and the (presumably) air between the tank and the mask. Taking the refractive index of air as n_{air}, then this amplifies the slope on exit from the tank by n_0/n_{air}. An additional correction can also be made for the refractive index of the tank wall, n_{wall}. This does not change the slope within the air, but does provide an additional offset. If the tank wall has thickness T and we measure B from the *outside* of the tank wall, then (14) becomes

$$\Delta\xi = -\frac{1}{2}W\left(W+2\frac{n_0}{n_{air}}B+2\frac{n_0}{n_{wall}}T\right)\frac{1}{n_0}\frac{\partial n'}{\partial x} \tag{15a}$$

$$\Delta\zeta = -\frac{1}{2}W\left(W+2\frac{n_0}{n_{air}}B+2\frac{n_0}{n_{wall}}T\right)\frac{1}{n_0}\frac{\partial n'}{\partial z} \tag{15b}$$

in the coordinate system of the textured mask. Similarly, if the experiment is not thin, then the magnification term projecting this onto the central plane must take into account the refractive index variations for rays entering and leaving the tank. The net result of this is that

$$\Delta\xi_{exp} = -\frac{1}{2}\left[\frac{L-B-\left(1-\frac{n_{air}}{2n_0}\right)W-2\left(1-\frac{n_{air}}{2n_{wall}}\right)T}{L-\left(1-\frac{n_{air}}{n_0}\right)W-2\left(1-\frac{n_{air}}{n_{wall}}\right)T}\right]W\left(W+2\frac{n_0}{n_{air}}B+2\frac{n_0}{n_{wall}}T\right)\frac{1}{n_0}\frac{\partial n'}{\partial x}, \tag{16a}$$

$$\Delta\zeta_{exp} = -\frac{1}{2}\left[\frac{L-B-\left(1-\frac{n_{air}}{2n_0}\right)W-2\left(1-\frac{n_{air}}{2n_{wall}}\right)T}{L-\left(1-\frac{n_{air}}{n_0}\right)W-2\left(1-\frac{n_{air}}{n_{wall}}\right)T}\right]W\left(W+2\frac{n_0}{n_{air}}B+2\frac{n_0}{n_{wall}}T\right)\frac{1}{n_0}\frac{\partial n'}{\partial z}. \tag{16b}$$

As stated above, there is normally a constitutive relationship between the density of the fluid and the refractive index. To a good approximation the relationship between refractive index and density for salt water is linear (Weast 1981), allowing us to write

$$\nabla n = \frac{dn}{d\rho}\nabla\rho = \beta\frac{n_0}{\rho_0}\nabla\rho, \tag{17}$$

where

$$\beta = \frac{\rho_0}{n_0}\frac{dn}{d\rho} \approx 0.184, \tag{18}$$

and ρ_0 is the nominal reference density (1000kg m^{-3}). Substitution into (13) then gives the relationship between density fluctuations ρ' and apparent movement of the source of a light ray

$$\Delta\xi_{exp} = -\frac{1}{2}\left[\frac{L-B-\left(1-\frac{n_{air}}{2n_0}\right)W-2\left(1-\frac{n_{air}}{2n_{wall}}\right)T}{L-\left(1-\frac{n_{air}}{n_0}\right)W-2\left(1-\frac{n_{air}}{n_{wall}}\right)T}\right]W\left(W+2\frac{n_0}{n_{air}}B+2\frac{n_0}{n_{wall}}T\right)\frac{\beta}{\rho_0}\frac{\partial\rho'}{\partial x}, \quad (19a)$$

$$\Delta\zeta_{exp} = -\frac{1}{2}\left[\frac{L-B-\left(1-\frac{n_{air}}{2n_0}\right)W-2\left(1-\frac{n_{air}}{2n_{wall}}\right)T}{L-\left(1-\frac{n_{air}}{n_0}\right)W-2\left(1-\frac{n_{air}}{n_{wall}}\right)T}\right]W\left(W+2\frac{n_0}{n_{air}}B+2\frac{n_0}{n_{wall}}T\right)\frac{\beta}{\rho_0}\frac{\partial\rho'}{\partial z}. \quad (19b)$$

Simply measuring the apparent displacements and inverting (19) then allows us to determine the perturbation density gradient. This may, in turn, be integrated once to return the density perturbation itself.

The main difficulty is determining the apparent displacements $\Delta\xi_{exp}$ and $\Delta\zeta_{exp}$ with sufficient accuracy for the whole process to be meaningful. Often the apparent displacements are only a small fraction of a pixel. DigiFlow employs a range of techniques to achieve this. The most accurate, but computationally expensive, use powerful pattern matching techniques to determine the apparent displacement as accurately as possible: the design of this part of the system has concentrated more on accuracy than speed. DigiFlow also offers faster (but less accurate) techniques to provide a reasonable approximation relatively quickly.

5.6.4.1 Qualitative Preview

Toolbutton:

Shortcut:

Related commands: `process Analyse_SyntheticSchlierenQualitative(..)`

This option provides a qualitative or semi-quantitative preview of an image sequence using relative simple processing to determine the gradient of the perturbation density field. The processing used here is similar to that provided during Video Capture (see §5.1.5).

Starting the option provides access to a simple dialog box (see figure 81) for selecting the input image stream (Experiment selector) and, optionally, a Background Image. If the latter is not specified, then the first frame of the Experiment file is utilised. Both these selectors have the standard Sift button (see §4.3).

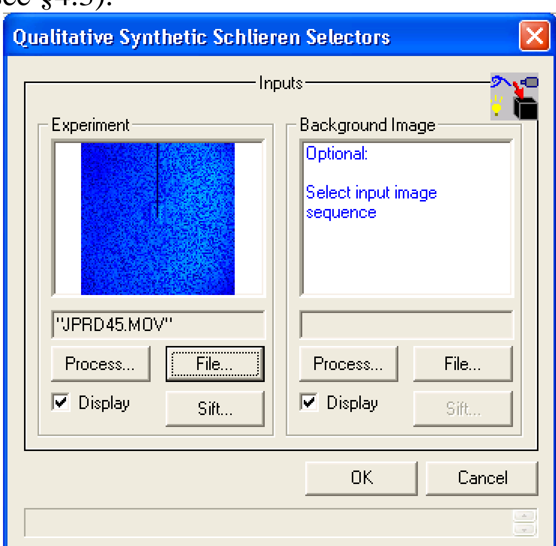


Figure 81: Dialog for determining which sequence is to be previewed with synthetic schlieren.

Once the sequence has been identified, an image window is opened to show the preview. The preview itself is controlled by a second dialog. Some of the controls on this dialog are reminiscent of those seen in §5.5.1.14 for controlling the animation of sequences. Controls specific to synthetic schlieren are found in the Processing and Gain groups. The first of these determines the type of processing to be performed.

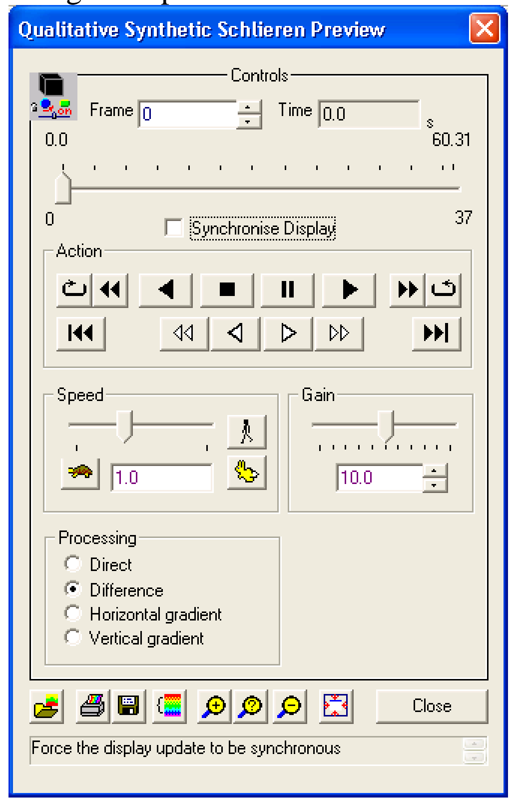


Figure 82: Control dialog for qualitative synthetic schlieren preview.

There are four processing options: Direct simply shows the input stream without any processing, while Difference is the simplest (and computationally fastest) technique that provides a qualitative output proportional to the magnitude of the gradient in the density perturbation. The Horizontal gradient and Vertical gradient options perform more a more sophisticated analysis that returns a semi-quantitative output of the specified component of the gradient in the density perturbation. Note that these two options distinguish between positive and negative gradients.

The Gain control determines the relationship between the gradient and the intensity of the display. The display colour scheme may be changed using Colours, and a different set of input streams may be used by clicking Selectors.

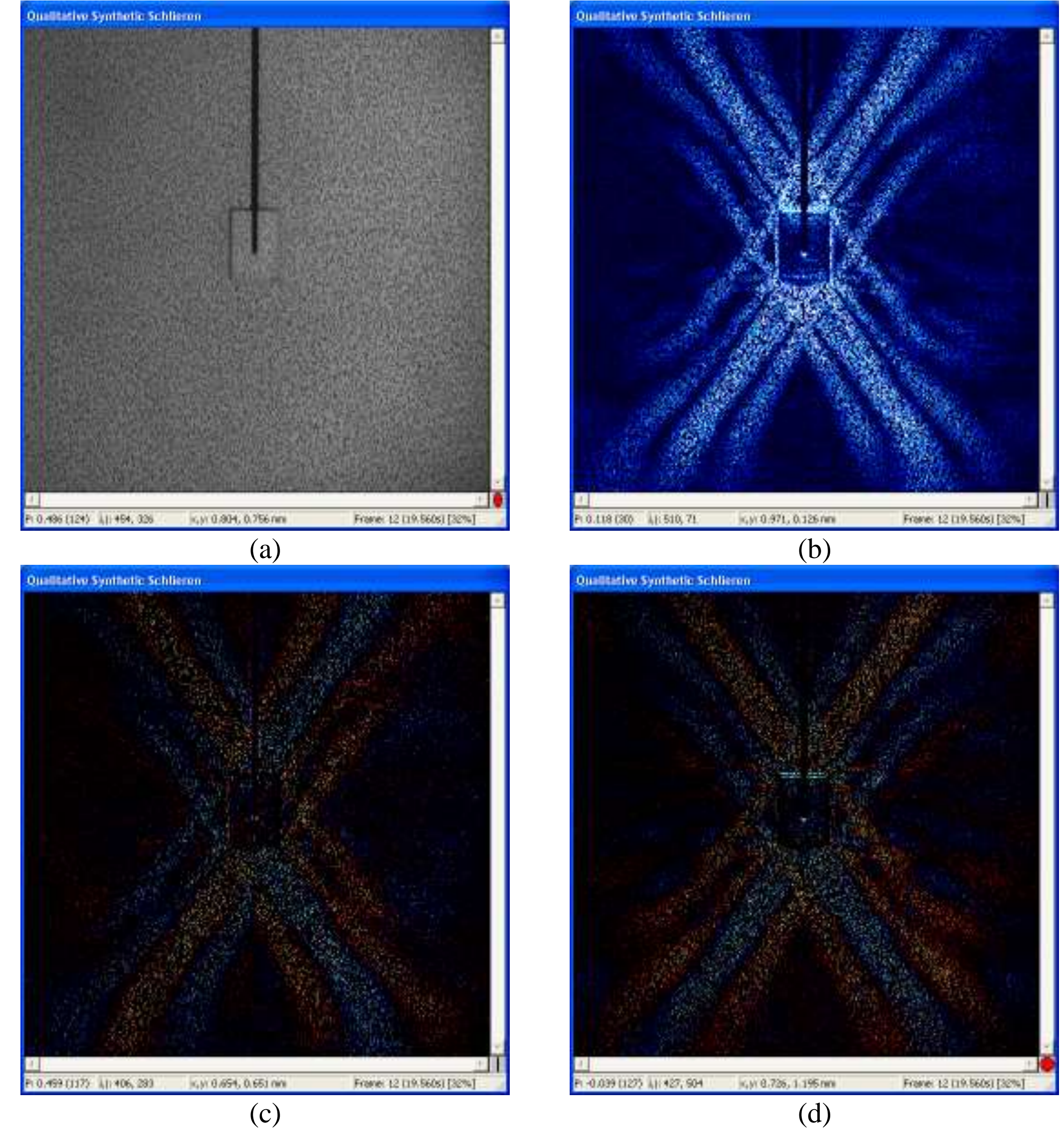


Figure 83: Examples of output from Qualitative Synthetic Schlieren. (a) Direct, (b) Difference, (c) Horizontal gradient and (d) Vertical gradient.

The Synchronise display check box forces synchronisation such that each and every frame in the sequence is displayed, even if this slows the update rate below the desired frame rate. If Synchronise display is cleared and the computer cannot keep up with the desired frame rate, then frames are skipped to maintain that frame rate.

5.6.4.2 Interpolative

Toolbutton:

Shortcut:

Related commands: `process Analyse_SyntheticSchlierenInterpolative(..)`

The method of calculating the synthetic schlieren image used in this option is a compromise between speed and accuracy. While it is based on a similar technology to that used in the Qualitative Preview version of synthetic schlieren discussed in §5.6.4.1, the algorithm is tuned to give a wider dynamic range, greater accuracy, and more complete coverage of data. However, the resulting measurements remain less accurate than those obtained using Pattern Matching version of synthetic schlieren (see §5.6.4.3).

The processing here was conceived for masks located behind the experiment containing lines, however experience has shown that it also provides reasonable semi-quantitative measurements for other mask geometries (*e.g.* random dots).

Suppose the changes in the refractive index gradient give an apparent vertical displacement of the mask by some amount $\Delta\zeta$ at time t. We shall assume the curvature in ρ' is small so that $\Delta\zeta$ varies only over length scales large compared to the features contained in the mask.

As we have seen, the intensity of a pixel is related to the mean of the (unknown) intensity falling on the CCD sensor by

$$P_{ij}(t) = \frac{1}{\Delta x \Delta z} \int_{x_i - \Delta x/2}^{x_i + \Delta x/2} \left[\int_{z_j - \Delta z/2}^{z_j + \Delta z/2} p(x, z; t) dz \right] dx . \quad (20)$$

The combination of optical imperfections, noise and imperfections in the mask will ensure that $p(x,z;t)$ is a continuous function, even when the mask contains discrete steps. We may approximate $p(x,z;t)$ using a piece-wise quadratic interpolation in a manner similar to that employed for numerical solution of the advection equation in control volume techniques. The idea here is that the approximation $P_{ij}(t) = p(x_i,z_j;t)$ (approximating the integral in (20)) by the so-called *mid-point rule* for numerical integration) has an error $O(\Delta z^3)$ which is of the same order as the error in a quadratic interpolation of the intensity (x_i,z_j). More specifically, if $\hat{P}_{0,ij}(z-z_j)$ is the quadratic interpolation of the unperturbed image around (x_i,z_j), we look to solve for the value $z-z_j = \Delta\zeta_{ij}$ such that $\hat{P}_{0,ij}(\Delta\zeta_{ij}) = P_{ij}(t)$. Thus the apparent displacement (in the z direction) of the mask $\Delta\zeta_{ij}$ is given by the roots of

$$P_{0,0} - P + ½(P_{0,1} - P_{0,-1})\Delta\zeta + ½(P_{0,1} - 2P_{0,0} + P_{0,-1})\Delta\zeta^2 = 0. \quad (21)$$

Here we have used the shorthand $P = P_{ij}(t)$, $P_{0,0} = P_{0,ij}$, $P_{0,-1} = P_{0,i,j-1}$ and $P_{0,1} = P_{0,i,j+1}$. To avoid ambiguity as to which root of (21) should be taken, we solve (21) only if $P_{0,0}$ is intermediate between $P_{0,-1}$ and $P_{0,1}$, and the intensity contrast across the three lines is sufficiently large (*i.e.* $|P_{0,1}-P_{0,-1}| > \Delta P_{min}$). Further, we select the root of (21) with smallest $|\Delta\zeta|$, effectively limiting $\Delta\zeta$ to be less than the spacing of the lines on the mask.

As an alternative to solving the quadratic expression for $\Delta\zeta$ given by (21), we may utilise a binomial expansion to show that this process has the same $O(\Delta z^2)$ accuracy as assuming $\Delta\zeta$ is quadratic in $P_{0,ij}$. This latter approach was used by Sutherland *et al.* (1999) and gives

$$\Delta\zeta = \left[\frac{(P - P_{0,0})(P - P_{0,-1})}{(P_{0,1} - P_{0,0})(P_{0,1} - P_{0,-1})} - \frac{(P - P_{0,0})(P - P_{0,1})}{(P_{0,-1} - P_{0,0})(P_{0,-1} - P_{0,1})} \right] \Delta z . \quad (22)$$

As with (21), $\Delta\zeta$ is calculated from (22) only if $P_{0,0}$ is intermediate between $P_{0,-1}$ and $P_{0,1}$, and there is sufficient intensity contrast across the three lines.

Once $\Delta\zeta$ has been determined from either (21) or (22), it is mapped from pixel space into physical space and (19) is applied to determine $\partial\rho'/\partial z$. Points for which Δz could not be calculated (typically points where $\partial P/\partial z$ is too small, as may occur if a line is centred on a pixel and would lead to an ambiguity in the sign of the displacement) are replaced by interpolated values using a Gaussian weighting function. The final result is scaled and used to construct an image representing $\partial\rho'/\partial z$.

In the present implementation, if the value of $\Delta\zeta$ determined from (22) exceeds one pixel then the reference image intensities are themselves displaced so as to avoid extrapolation. This effectively increases the accuracy and dynamic range of the technique.

Inputs tab

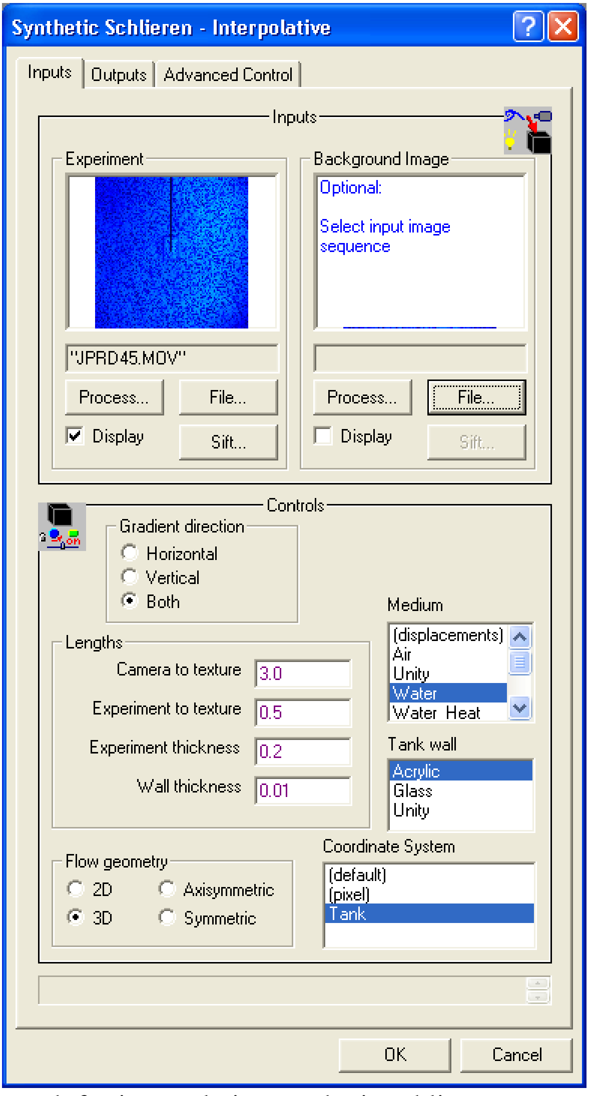


Figure 84: The inputs dialog tab for interpolative synthetic schlieren.

The Experiment stream, specified using File and optionally sifted with Sift, should contain an image of a strong texture located behind the experiment. DigiFlow then compares this input stream with a reference image to determine the apparent displacements. The reference image may be specified using the optional Background image input stream. If no stream is specified, then the first frame of the Experiment input stream is used instead.

As noted above, this method was conceived for masks containing lines, in which case Gradient direction should be set normal to the lines. The most accurate results will be obtained in this configuration. However, if the mask contains two-dimensional features (such as random dots), then it is possible to generate Both in-plane components of the gradient.

The Flow geometry group enables internal processing options that attempt to ensure the result is consistent with the underlying geometry of the flow.

Details of the experimental setup are required in the Lengths group to allow interpretation of the apparent movements of the dots. The units for these should be consistent with the units for the density gradient that will be determined. Ultimately, the output will be $(1/\rho_0)\nabla\rho'$, which has dimensions of $1/length$. Specifying the distances here in metres will give units of m^{-1} for the final result.

Note that a distance of zero is acceptable for Experiment to texture, the distance from the back of the experiment to the texture mask, but not for Experiment thickness. The Experiment

thickness should be the internal measurement of the tank, while Experiment to texture should be measured from the outside of the tank. The Wall thickness should be specified for the wall closer to the texture, and its corresponding Tank wall material selected.

Camera to texture is the distance between the effective focal plane of the camera and the texture mask. It is generally sufficient to measure the distance from the base of the lens to the texture. Experiment to texture is the distance from the back of the experiment to the texture mask. This distance can be zero for some set ups. Experiment thickness is the width of the flow through which the light rays experience density fluctuations. This cannot be zero.

The Medium list box allows selection of different media for the experiment. The key detail, picked up from a DigiFlow data base, is the relationship between refractive index and density changes. In addition to the normal media, two pseudo media are also included: Unity returns refractive index gradients rather than density gradients, while (displacements) returns the calculated apparent displacements (with units of the selected coordinate system) rather than density gradients.

The coordinate system required to interpret the experiment is specified in the Coordinate system list box.

Outputs tab

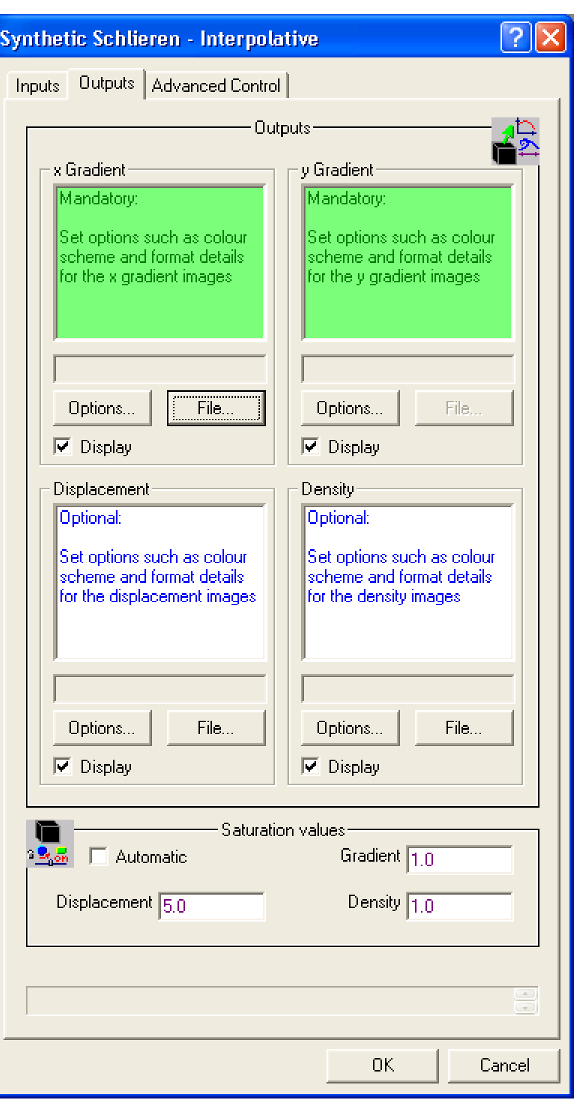


Figure 85: The outputs tab for interpolative synthetic schlieren.

The most important controls on the Outputs tab are for selecting the main output streams x Gradient and/or y Gradient. Whether one or both of these is required depends on the selected option in the Gradient direction group on the Inputs tab. The visual scaling of the images produced is determined by the Gradient entry in the Saturation values group. This value sets the gradient that will produce a saturated image. For most image formats, getting this wrong will require reprocessing of the image due to quantisation errors introduced. For this reason the use of the .dfi file format is recommended as this does not sacrifice dynamic range and the scaling may be subsequently changed at a later date.

Advanced control tab

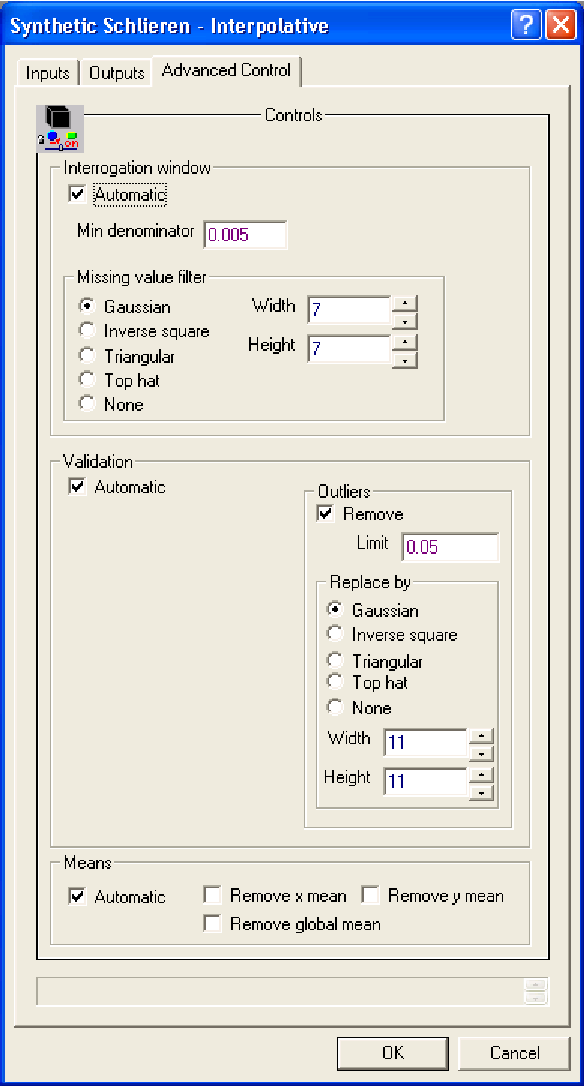


Figure 86: The advanced control tab for interpolative synthetic schlieren.

In most cases the controls on the Advanced control tab should be left on Automatic. The Interrogation window group controls the limits on the quadratic interpolation that must be satisfied before the results can be used, and also controls how to fill in any missing values. The Validation group determines how to check for consistency with neighbouring points.

5.6.4.3 Pattern Matching

Toolbutton:

Shortcut:

Related commands: `process Analyse_SyntheticSchlierenPatternMatch(..)`

The most sophisticated (and slowest) of the synthetic schlieren algorithms is based on an advanced pattern matching algorithm that has its origins in PIV (Particle Image Velocimetry; see §5.6.5.2).

The mask behind the flow giving the texture to the image is typically constructed from random features of high contrast. The simplest way of generating this is by printing a pattern onto overhead projector transparencies and then tiling these up into a sheet of the required side with clear adhesive tape. The following PostScript file may be used to generate a suitable pattern. The pattern is a basic square grid of dots, with each of the dots perturbed by a random amount. The randomness helps prevent aliasing errors and ensures that the pattern is robust against any defects produced when you overlap slightly multiple tiles of transparency.

```
%!PostScript
% Generate dot pattern for synthetic schlieren
/mm {25.4 div 72 mul} def

% Set the basic size of the pattern (mean spacing in mm)
/Size 2 def

% Set the size of the sheet
/Sheet 300 Size mul def

% Relative size of dots to their mean spacing
/DotFraction 0.25 def

% Scale for randomnesss
/Randomness 0.6 def

% Draw black background
0 0 moveto
Sheet mm 0 rlineto
0 Sheet mm rlineto
Sheet mm neg 0 rlineto
closepath
0 setgray
fill

% Draw grid of white dots with random perturbations
1 setgray
0 Size Sheet
  {/y exch mm def
   0 Size Sheet
     {/x exch mm def
      gsave
       x rand 0.25e9 div Size mul Randomness mul add
       y rand 0.25e9 div Size mul Randomness mul add
%x y
       translate
       0 0 Size DotFraction mul mm 0 360 arc
       closepath
       fill
      grestore}
   for}
for

% Set number of copies of sheet to be made
/#copies 2 def
showpage
```

This PostScript file can be simply copied to any PostScript printer. If a PostScript printer is not available, an interpreter such as GhostScript/GhostView could be used. The pattern should be scaled (using the `/Size` definition) so that the dots are close to the limit of what the camera can resolve. Some trial and error may be required to determine the optimal size for a given experimental setup.

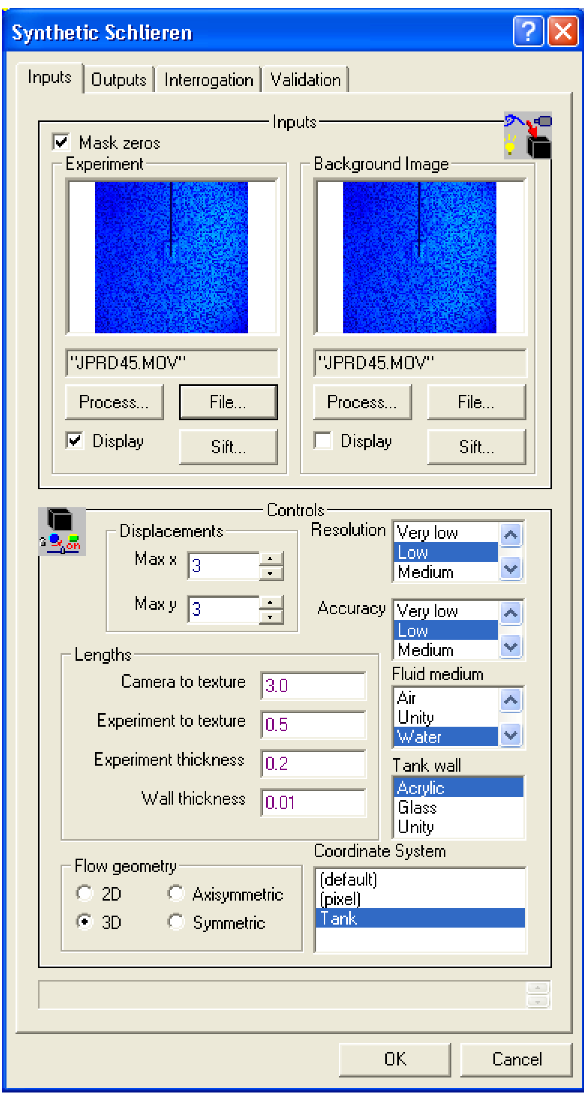


Figure 87: The Inputs tab of the synthetic schlieren dialog.

The synthetic schlieren interface is divided into six tabbed dialogs. The first two, Inputs and Outputs are mandatory and control what is to be processed, and where the results are to be stored, respectively. The other four tabs, Interrogation, Validation, Quality and User Interpolation may be used by experienced users to tweak the process to yield better performance in some situations. The more advanced controls on these tabs are not available under the Free DigiFlow Licence.

Inputs page

The standard synthetic schlieren process takes two input streams. The first, specified by the Experiment group, is the video sequence of the experiment itself. As normal, an image selector is used to specify the stream.. This selector may be specified from a file by clicking the File button, in which case the standard Open Image dialog box (see §4.1) is produced.

Alternatively, clicking the Process button will allow a source process to be used (refer to §7 on chaining processes for further details).

The Background input takes a single image (specified in the normal manner). This image should be of the background texture mask *before* the experiment introduces any density perturbations. Typically, this image is taken just prior to the experiment, and contains all the ambient refractive index variations due to, for example, a background density stratification.

The Mask zeros check box causes DigiFlow to ignore all pixels with an identically zero intensity. This feature is designed to allow simple masking of images. Such masking may be used to remove parts of the field of view that do not contain the flow. For example, it could be a static boundary to the flow, a free surface, or possibly an object moving through the flow. In each case external processing of the image sequence should be made to apply the mask prior to starting the synthetic schlieren processing.

There are eight groups of controls on the Inputs tab. The first controls maximum apparent displacement that will be searched for. The values Max x and Max y are specified in pixels and are assumed symmetric about zero. These values should be set to represent slightly more than the maximum expected apparent displacement of the mask. In most circumstances this will be limited to two or three pixels, and will generally be isotropic (hence specify the same values for Max x and Max y). Note that the computation required to determine the displacement increases approximately as the product of these two values, hence specifying excessively large values is counterproductive.

The Flow Geometry group is used to indicate the basic geometry of the flow under consideration, and control the invocation of processes optimised for the specific geometry. The entries 2D and 3D have the obvious meaning. Similarly, Axisymmetric is for flows where the symmetry axis lies in the mid-plane (normal to the viewing axis) of the experiment, and Symmetric is for flows where the mid-plane is a plane of symmetry, but the flow is not axisymmetric.

The Lengths group specifies the geometric setup of the experiment. **The distances should be specified in the same units as the selected coordinate system** (see below), or it will be difficult to interpret the results of the calculation. Note that a distance of zero is acceptable for Experiment to texture, the distance from the back of the experiment to the texture mask, but not for Experiment thickness. The Experiment thickness should be the internal measurement of the tank, while Experiment to texture should be measured from the outside of the tank. The Wall thickness should be specified for the wall closer to the texture, and its corresponding Tank wall material selected.

Both input streams may be sifted (§4.3) to extract the desired subregion and times. This feature is activated using the Sift button associated with each of the input streams.

To provide a simplified interface to the internal workings of the synthetic schlieren algorithm, DigiFlow provides a range of predefined settings that have the effect of producing different resolutions and accuracies. The Resolution and Accuracy list boxes both have six possible settings: Very low, Low, Medium, High, Very high and Best. The choice will depend on a combination of the intended purpose of the results, and the time available to undertake the processing. The fastest processing is achieved at the Very low end of both scales, while the most detailed and accurate measurements are obtained with both Resolution and Accuracy set to Best. In the latter case, even with relatively basic analogue video equipment, the accuracy with which the apparent movement of the texture mask may be detected can be better than 1/100 of a pixel, and the spatial resolution of the measurements is a few pixels. By default, the Resolution and Accuracy controls will be enabled. However, if the Automatic check box for the Interrogation window group on the Advanced Control tab is cleared, then the Resolution and Accuracy controls will be disabled.

To determine the relationship between refractive index and density, DigiFlow requires that a fluid medium is specified with the Medium list box. This box contains a range of standard fluids (*e.g.* Water and Air), plus the special fluid Unity in which all the physical constants are set to unit values. In the context of synthetic schlieren, DigiFlow extracts the value of $\beta = (\rho_0/n_0)(dn/d\rho)$ (see (17) and (18)) for the selected medium. Additionally, (displacements) will cause the pattern matching process to return the apparent displacements of the mask rather than the density gradient.

The final input on this tab is Coordinate System. This specifies the coordinate system that will be used to relate pixel to world coordinates. The coordinate system is assumed to have been defined in the mid-plane of the experiment (not the plane of the dots). See §5.2.6 for further details on setting up a coordinate system.

Outputs page

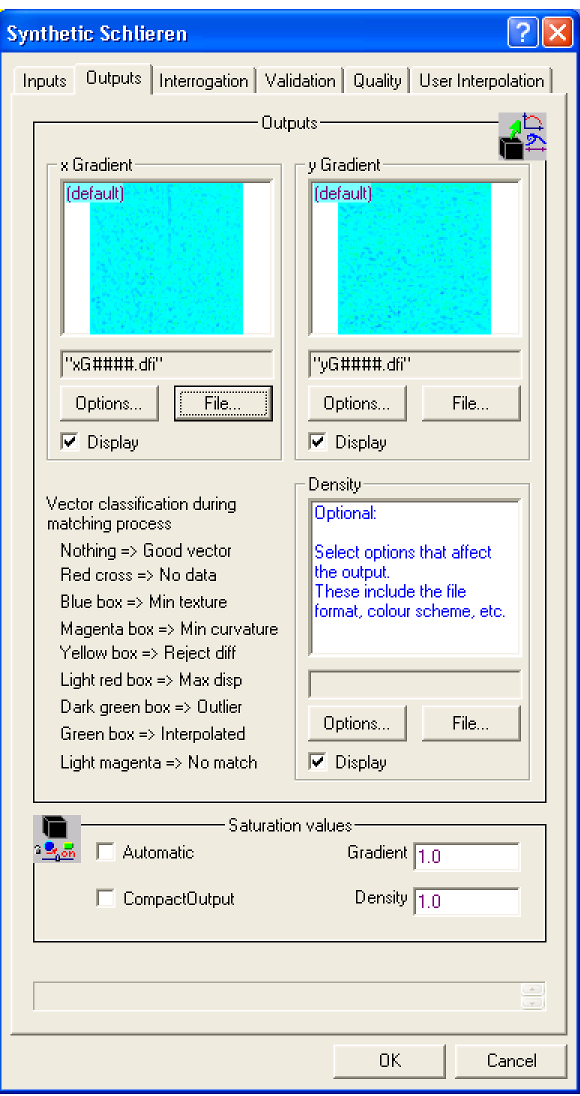


Figure 88: The Outputs tab of the synthetic schlieren dialog.

The Outputs tab controls the destination and scale of the output from the synthetic schlieren calculation. This dialog page consists of three image selectors, each with its own Options and File buttons and Display check boxes. The destination for the output stream is selected by clicking the File button (thus starting the standard Open Image dialog box; see §4.2), while the colour scheme and other related details are selected with the Options button (see §4.4).

The x Gradient, y Gradient and Density images are centred with a zero value corresponding to half the intensity range (*i.e.* 128 for an 8 bit image format). Saturation corresponds to the values given in the Scales group for Gradient and Density. The gradient images have the units of ‘per unit length’ (what the unit length is depends on the coordinate system selected), and represent $\nabla\rho'/\rho_0$, unless (displacements) was specified for .the Fluid Medium on the Inputs tab.

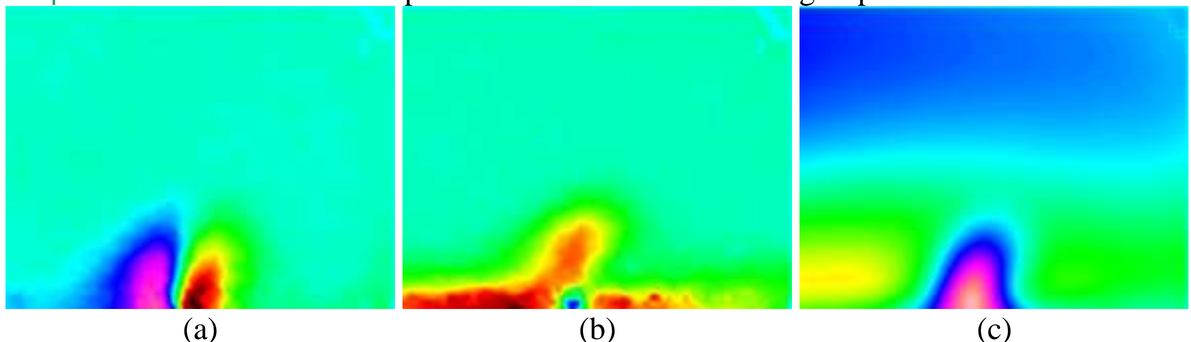
The optional Density output (not available with free licences) is calculated by a least squares integration of the density gradient field. In general, integration of a vector field to find a scalar potential is not unique, as the vector field will contain both irrotational and rotational parts. With synthetic schlieren, the density gradient field should be irrotational, which would make the integration unique, but inevitable measurement noise renders some rotational component. The integration procedure used in DigiFlow aims to find the scalar potential (here ρ'/ρ_0) that minimise the root mean square of this rotational part (effectively minimising the enstrophy), hence is a least squares solution. The solution process is achieved iteratively using a multigrid approach that is aware of any missing data or masked regions in the synthetic schlieren results. (Inevitably, there is some data loss in the neighbourhood of any masked regions.) The integration procedure leaves one unknown arbitrary constant of integration which DigiFlow sets by forcing the spatial mean density perturbation to vanish. Of course, this may not always be appropriate.

The DigiFlow Data format (.dfd) or DigiFlow Pixel format (.dfp) may be specified for any of these output images so that the data is readable in other applications. However, it is recommended that the .dfi floating point format is used if you wish to make quantitative use of the data.

Selecting the Compact check box causes DigiFlow to save an approximation to the calculated density gradient field by only saving the gradient at the nominal location of the interrogation windows used to calculate the gradient. DigiFlow will automatically expand out this approximate gradient field, when it is reloaded, to produce one that is very close to that saved without the Compact option. The files produced, however, are much smaller.

Figure 89 shows an example of the density gradient fields and the density perturbation. Note that in all cases they are normalised by the reference density ρ_0. The density gradient fields therefore have dimensions of per length; it is important that you use the same units for the Lengths group on the Inputs tab as you use in the chosen coordinate system, or else it will be difficult to interpret the output!

You may choose not to calculate the density perturbation while doing the synthetic schlieren computation, but instead calculate it later from the density gradient field. Tools: Recipies contains a suitable recipe to do this in the Differential group.



(a) (b) (c)

Figure 89: Example of synthetic schlieren output. (a) $(1/\rho_0)\ \partial\rho'/\partial x$, (b) $(1/\rho_0)\ \partial\rho'/\partial x$ and (c) ρ'/ρ_0 for a thermal plume erupting from a boundary layer.

Interrogation page

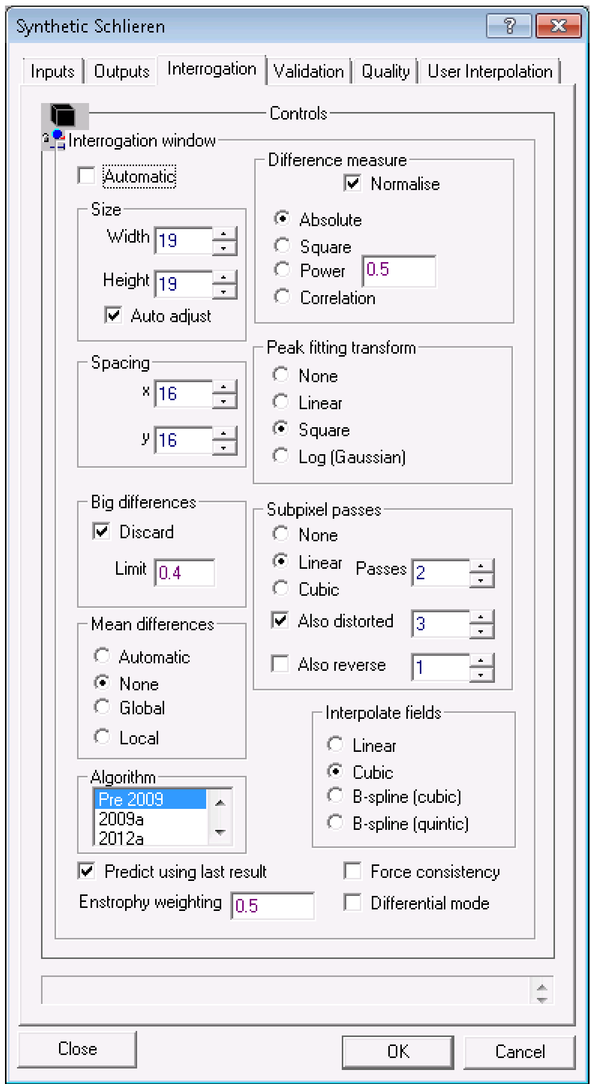


Figure 90: The Interrogation tab of the synthetic schlieren dialog.

The Interrogation page enables direct control over many of the underlying values of the synthetic schlieren calculation. For most users, checking the Automatic box yields optimal performance, with the Resolution and Accuracy controls on the Inputs tab providing all the performance tuning necessary. Advanced users, however, may wish to fine tune the processing manually; clearing the Automatic check box enables the remaining controls and disables the Resolution and Accuracy controls on the Inputs tab.

The Size subgroup controls the Width and Height of the interrogation window. Increasing the size of the window increases the accuracy, but decreases the spatial resolution and slows the computation. If the window is too small, relative to the pattern size, then very poor results are achieved. The Adjust check box specifies whether DigiFlow can adjust the size of the window if it thinks this is necessary to produce more reliable data.

The Spacing subgroup has the most direct control on the spatial resolution of the synthetic schlieren computation. This specifies the x and y spacing between points where the pattern matching process is undertaken. Clearly reducing this value, specified in pixels, increases the amount of computation, but may not always increase the spatial resolution due to the interplay with the Size of the interrogation window.

The Difference measure subgroup specifies the type of difference calculation. This is the function that DigiFlow minimises as it searches for the correct apparent shift. In practice,

there is little to choose between the three functions. The Absolute option is computationally a little cheaper, while Correlation is that frequently used in PIV techniques. Table 1 summarises the various difference measures f that may be used in DigiFlow. Note that in all cases the summation is over N valid pixels in the interrogation region. The Power option is simply a generalisation of Absolute and Square. The Normalise check box rescales each of the measures, based on the strength of the texture in the interrogation region.

f	Standard	Normalised
Absolute	$\sum\lvert A-B\rvert$	$\dfrac{\sum\lvert A-B\rvert}{\sqrt{\left(\sum\lvert A\rvert\right)\left(\sum\lvert B\rvert\right)}}$
Square	$\sum(A-B)^2$	$\dfrac{\sum(A-B)^2}{\sqrt{\left(\sum A^2\right)\left(\sum B^2\right)}}$
Power	$\sum\lvert A-B\rvert^p$	$\dfrac{\sum\lvert A-B\rvert^p}{\sqrt{\left(\sum\lvert A\rvert^p\right)\left(\sum\lvert B\rvert^p\right)}}$
Correlation	$\sum AB-\dfrac{\left(\sum A\right)\left(\sum B\right)}{N}$	$\dfrac{\sum AB-\dfrac{\left(\sum A\right)\left(\sum B\right)}{N}}{\sqrt{\left(\sum A^2-\dfrac{\left(\sum A\right)^2}{N}\right)\left(\sum B^2-\dfrac{\left(\sum B\right)^2}{N}\right)}}$

Table 1: The difference measures used in DigiFlow pattern matching.

The Peak fitting transform group determines the method of processing interpolating in the neighbourhood of the smallest value of the difference measure f (or largest value, for the case of the correlation measure) in order to provide an improved estimate. In all cases a bi-quadratic least squares procedure using nine points is employed. However, DigiFlow provides the option of transforming the difference measure prior to undertaking the fitting. The possible transformations are shown in table 2. Note that the logarithmic option effectively assumes a Gaussian form for the difference measure in the neighbourhood of the optimal shift.

	None	Linear	Square	Log
Absolute	f	f	f^2	$\log(f)$
Square	f	$\sqrt{f}$	F	$\log(f)$
Power	f	$f^{1/p}$	$f^{2/p}$	$\log(f)$
Correlation	f	f	f^2	$\log(f)$

Table 2: Transformation of the difference measure f prior to computing bi-quadratic least squares fit.

The Subpixel passes subgroup has a pronounced effect on the accuracy, resolution and speed of the calculation. The radio buttons determine the basic type of treatment to obtain improved subpixel accuracy: None is the fastest but least accurate. Linear offers a good compromise between speed and accuracy, while Cubic provides the best results, but is substantially slower. The Passes edit box controls the number of levels of subpixel treatment. For Linear a value of 1 to 3 is recommended, while Cubic normally only requires 1.

The Interpolate fields radio group controls how the data, initially obtained only at the centres of the interrogation zones, is expanded to fill the complete image. The simplest option of Linear, which uses a bilinear interpolation, tends to end up with an artificial appearance. The next level of sophistication, Cubic, produces a good balance between speed and accuracy.

While the resulting fields are continuous, they are not continuously differentiable. This problem is overcome by the computationally more expensive cubic b-spline and the quintic b-spline. For most circumstances either the cubic or cubic spline provides the best compromise between computational efficiency and accuracy.

If the interrogation window Spacing is small to improve the spatial resolution, then it is recommended that Also distorted is checked. This enables DigiFlow's unique image distortion technology to substantially increase the spatial resolution. A further improvement in both resolution and accuracy may obtained in some circumstances by also checking Also reverse.However, for high quality images, undertaking the reverse pass may lead to a deterioration in the quality of the results.

Checking Discard in Big differences will dynamically discard pixels that DigiFlow determines may not belong to the pattern it is trying to match. While discarding valid pixels can detrimentally affect the signal to noise ratio, retaining invalid ones can have an even more serious effect. The Limit controls the level at which pixels are discarded.

For images that have a poor signal to noise ratio, a fluctuating level of illumination, and strong spatial gradient in intensity, a spurious signal can be obtained from the interaction between the spatial gradient and the temporal fluctuations. The Mean differences group controls whether DigiFlow will attempt to correct for this by rescaling the image intensities to remove this signal. Selecting None will turn off the image rescaling to deal with mean differences between images, while Global will force the mean intensity of the two images (excluding any pixels of zero intensity) to be the same. The processing invoked by Local is similar to that of Global except that it does it locally for individual interrogation windows. While Local may superficially seem the most attractive, the results are much more sensitive to noise and should only be used when there is no other solution. The Automatic setting will attempt to assess which of the other three settings is most appropriate.

The Algorithm control provides access to different internal versions of the pattern matching algorithm, thus ensuring backward compatibility.

The Enstrophy weighting controls the weighting applied to the condition that the apparent displacements must be expressed as the gradient of a scalar when determining the optimal apparent shift of the dots. The way in which this weighting is used depends on the Flow geometry setting in the Inputs tab.

Setting Predict using last result will suppress the initial pixel pass for points where a result has been calculated previously. This reduces the time required to converge on a solution.

Validation page

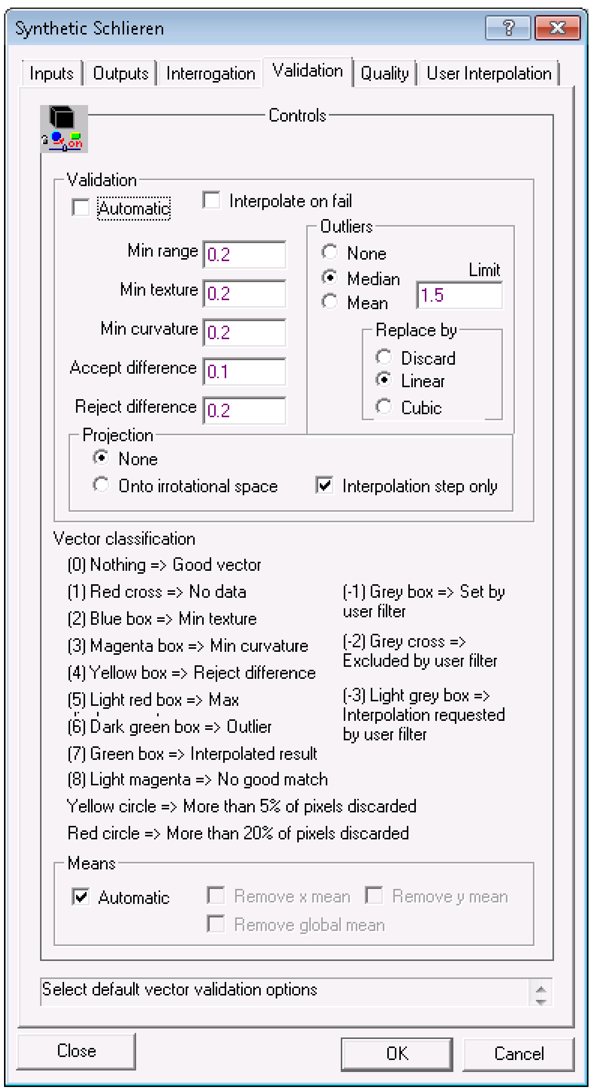


Figure 92: The Validation tab of the synthetic schlieren dialog.

Clearing the Automatic check box in the Validation group allows direct user control over the parameters that control validation of the individual apparent displacement vectors. Min texture prohibits computation when the standard deviation of intensity within the interrogation window is less than the specified value. Similarly, Min range sets the minimum range (maximum minus minimum values) the intensity within the interrogation window must have before computation is permitted. Min curvature imposes a lower limit on how sharp the difference measure around optimal apparent displacement must be, while Reject difference imposes an upper bound on the difference measure. If the difference measure exceeds Accept difference multiplied by the range of intensities within the interrogation window (but is less than Reject difference, again multiplied by the range within the interrogation window) then it is subject to additional checks and processing to try to improve and ensure the quality of the resulting data.

The Outliers subgroup handles the identification and resolution of apparently erroneous data. For a well set up experiment, there should not be any erroneous data to be corrected. This feature is enabled by the Remove check box, with Limit applying to the difference between the value at the point and the mean of the neighbouring four vectors. This limit is expressed in terms of the apparent pixel displacements. If the Limit is exceeded, then the value is either Discarded, or relaxed towards a Linear or Cubic interpolation.

The apparent displacement field visualised by the synthetic schlieren is due to the gradient of the refractive index field. As such, the apparent displacement field should be irrotational since curl(grad(·)) ≡ 0. The Projection group attempts to make use of this as part of the validation process by projecting the measured displacement field onto an irrotational space when Onto irrotational space is selected. This projection will be made for every iterated value of the measured displacement field unless the Interpolation step only box is checked. When checked, the projection will only be applied to displacement fields used to the steps used to distort the images.

The final group, Means, is enabled by clearing its Automatic check box. The three check boxes within this allow for the removal of apparent mean gradients in the measurements. The Remove x mean control scans the data for each y and removes any mean apparent displacement for that y. The Remove y mean performs a similar calculation for each x, while Remove global mean simply calculates the mean apparent displacement for all the data, and subtracts this from the data. In most circumstances it is unnecessary to remove the means, but there are times when extraneous optical effects, or experimental setups such as having the camera and texture mask mounted on a traverse, will make this facility desirable.

Quality

DigiFlow determines a range of additional information about the apparent displacement field that represents the density gradient during the processing of the experimental images. While for most users this additional information is of little value, the Quality tab makes it possible to output some of this for advanced users.

This output requires a .dfi file if all the information is to be retained as there are multiple planes of data available. In particular, these planes contain the following information:

Plane	Description
0	**Difference measure.** This is the value of the difference measure (see Table 1) for the final match.
1	**x curvature.** The curvature in the difference measure in the neighbourhood of the final match.
2	**y curvature.** The curvature in the difference measure in the neighbourhood of the final match.
3	**State**. Indicates the state of the pattern matching. State values are integer (although stored as floating point) as follows: 0 Good vector 1 No data 2 Insufficient texture 3 Insufficient curvature in difference measure 4 Difference too great – rejected 5 Displacement too great 6 Outlier 7 Interpolated value 8 No match found –1 Value set by user dfc code –2 Value set by user interpolation code –3 Value excluded by user dfc code.
4	**Fraction discarded.** The fraction of the pixels in the interrogation window that were discarded due to them being too different between the two images.
5	**x vector position.** Stores the location (in world units) at which the vector was

	determined.
6	**y vector position.** Stores the location (in world units) at which the vector was determined.
7	**x displacement.** Stores the actual displacement (in pixels) determined.
8	**y displacement.** Stores the actual displacement (in pixels) determined.

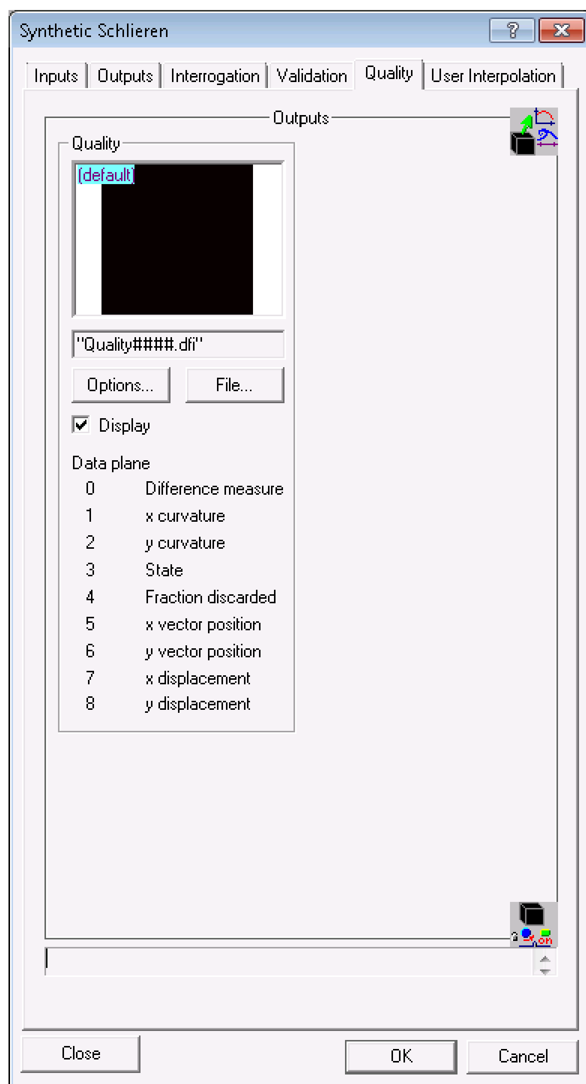


Figure 93: The Quality tab for synthetic schlieren.

User Interpolation

As part of the pattern matching procedure, DigiFlow interpolates the apparent displacement field determined at discrete points to the entire image plane in order to distort the images. The performance of the pattern matching depends on the quality of this interpolation, but the default techniques may not always be optimal. Hence, the User Interpolation tab provides the user with a way of bypassing the default mechanism and supplying their own customised scheme.

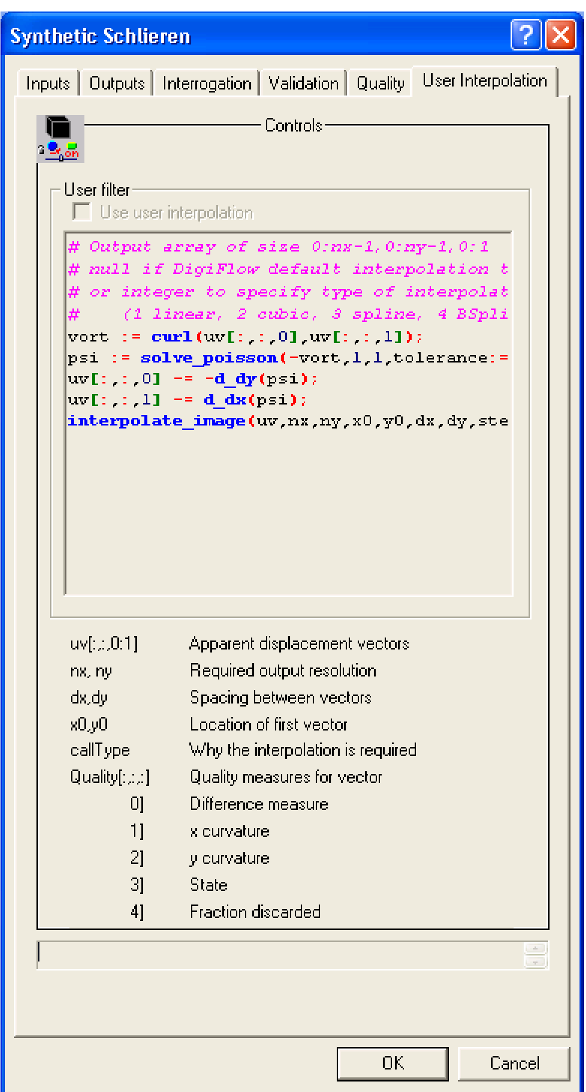


Figure 94: The User Interpolation tab for synthetic schlieren.

The tab is activated only if Automatic interrogation is turned off on the Interrogation tab, and the Use user interpolation check box is checked on the User Interpolation tab. The default sample code uses a weighted least squares approach to completing the interpolation. This is more computationally costly than the default method, but provides a more robust method of handling missing data.

Parameters passed to this code include the interrogation-window apparent displacement vector `uv`, the required output resolution `nx`, `ny`, the nominal interrogation window size `wx`, `wy` (in pixels), the spacing between the vectors `dx`, `dy` (in pixels), the location of the first vector `x0`, `x1`, and the quality measure `Quality`. The string `calltype` indicates the point in the algorithm when the call to the filter is made. This may take one of the values `"Pixel"`, `"SubPixel"`, `"Reverse"`, `"Distorted"` or `"Final"`. The experimental image `Pa` and background image `Pb` are also available.

Processing

When OK is pressed, the dialog box will check that all mandatory values have been entered. If they have not, then the focus will return to the page and control of the first missing value.

The progress of the processing may be viewed by selecting the Progress window that appears once synthetic schlieren has started. The contents of this window are updated

periodically during the processing of each of the images from the Experiment stream. Most of the time, this window provides information on the apparent vertical shift of the texture mask. The title bar on this window and the 'thread message' panel on the main status bar provide details of the individual calculations as they are performed.

The basic processing algorithm may be summarised by the following steps:

1 Determine optimal pixel shift.

- ◊ Determine optimal pixel shift by moving interrogation window around on Experiment image, measuring the difference between this and the comparable unshifted window on the Background image.
- ◊ For each optimal pixel shift, use a bi-quadratic least squares to obtain subpixel resolution.
- ◊ Repeat for each grid point on this level.
- ◊ Refine grid to next level by bi-linear interpolation, in a multi-grid-like process.

2 Determine optimal subpixel shift.

- ◊ Determine optimal shift in a manner analogous to the pixel shift, but using an interpolated version of the Experiment image to allow smaller shifts to be probed.
- ◊ For each optimal subpixel shift, use a bi-quadratic least squares to obtain an improved estimate of the optimal shift.

3 Determine optimal distorted shift.

- ◊ Use the current estimates of the apparent displacement to distort the Experiment image back to the Background image (*i.e.* try to undo the apparent movements).
- ◊ Repeat the optimal subpixel shift process, using this distorted image.
- ◊ The optimal shift from this process should be small (it represents the error in the previous optimal shift) is used to correct the optimal subpixel shift

4 Repeat steps 3 the required number of times, each with a smaller subpixel shift of the Experiment image.

5 Repeat steps 2 and 3, but shifting the Backround image rather than the Experiment image.

- ◊ The optimal shift that is produced by this *reverse* shift is inverted and itself distorted to shift it back to the Background frame of reference.

6 The forwards and distorted reverse shifts are combined to produce the ultimate optimal subpixel shift.

7 The shift is transformed to world coordinates.

8 The world coordinate shift is transformed into gradients in the density perturbation.

9 The density perturbation is computed by integrating the gradient field.

- ◊ The direct integration method used does not require boundary conditions, and the arbitrary constant of integration is defined so that the mean perturbation vanishes.

During the processing, DigiFlow will display a *Progress* window that provides feedback on the performance of the pattern matching algorithm. One of the key components of this is the

classification of displacement vectors by drawing boxes at their roots if there is some potential problem. The table below lists these classifications and gives a description of the various categories and an indication of the control that can affect this.

Symbol	Description	Advanced Control page
Red cross	No valid displacement vector	
Blue box	The image does not contain an adequate texture for the matching to be reliable.	Advanced: Min range Advanced: Min texture
Magenta box	The difference function being minimised does not have a well-defined peak.	Advanced: Min curvature
Yellow box	The value of the difference function is too large.	Advanced: Reject difference
Light red	The best match is found beyond the limit of the permissible shifts.	Inputs: Displacements
Dark green box	The optimal match produced an outlier. This has been replaced by an interpolated value.	Advanced: Outliers
Green box	Vector is the result of interpolation from surrounding vectors.	
Light magenta	A best match could not be found.	Inputs: Displacements: Max x and Max y

5.6.5 Particles

Processing of particles is split between this submenu, which includes particle streaks (§5.6.5.1) and Particle Image Velocimetry (§5.6.5.2), and the Particle Tracking submenu (§5.6.6).

5.6.5.1 Show as Streaks

Toolbutton:

Shortcut:

Related commands: `process Analyse_ShowAsStreaks(..)`

The Show as Streaks option provides a convenient method of reviewing and presenting image sequences containing particles. Such sequences will often be subsequently analysed in more detail using either Particle Tracking Velocimetry (PTV) or Particle Image Velocimetry (PIV). However, it is normally worth reviewing the sequence first as streaks as this will often give significant insight into the structure of the flow, regions where things are steady, and where the flow is unsteady, and where the contrast is adequate to proceed with quantitative measurements.

Two dialog boxes are produced as standard during the Show as Streaks process. The first (see figure 95) allows selection of the input data stream in the standard manner. Under most circumstances there will be no need to use Sift to change the timings, as this can be done subsequently. However, the exception to this is when dealing with image sequences that are interleaved so that images at different levels in a flow (for example) are stored adjacently and

only every *n*th image is at the same level. In this case it may be desirable to set the time step using Sift.

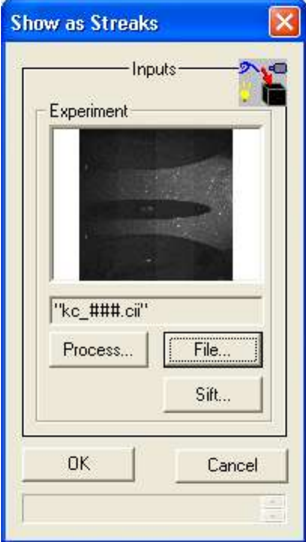


Figure 95: Dialog box used to specify the input stream for the Show as Streaks facility.

The main control dialog (see figure 96) consists of standard video controls, track bar and speed control. This dialog sits alongside a floating window containing the processed streaks image. Note that you may swap between these with the mouse to pan the image around, if desired. Note that both windows are floating (*i.e.* they are not required to remain within the main DigiFlow window).

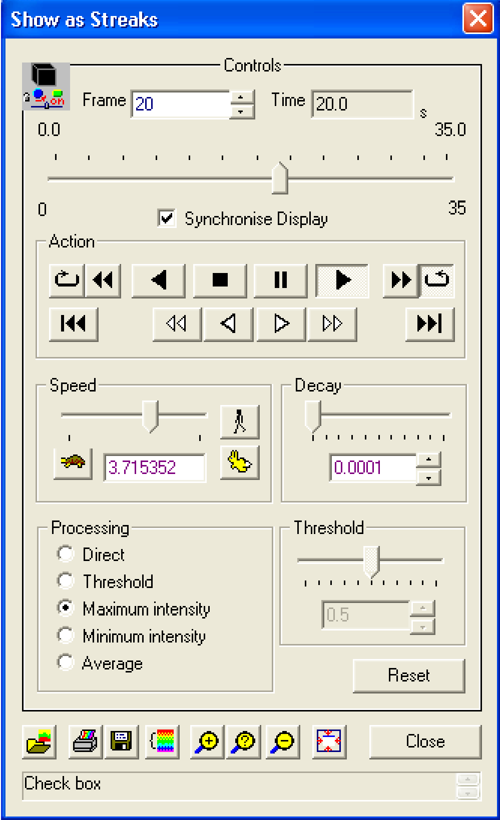


Figure 96: Dialog controlling the Show as Streaks facility.

The Processing group determines how the displayed image is to be constructed from the raw image stream. Direct simply shows the raw image, Threshold segments the raw image into a binary image prior to combining with a stored image (the Threshold group determining

the intensity level for this split), Maximum intensity will take the greater of the intensity in the current image and the corresponding pixel in the stored image, Minimum intensity will take the smaller of the intensity in the current image and the corresponding pixel in the stored image, and Average will generate the streaks using a simple arithmetic averaging process. In all cases (except for Direct) the intensity of the stored image is reduced by the amount specified by the Decay group each time a new image is added, thus providing a fading memory of the flow. The Reset button clears the stored image, thus resetting the streaks.

Which processing option produces the best results depends in part on the quality of the original images. For clean images with uniform illumination and good contrast, Threshold is likely to produce the best results. However, if the images have strong variations in illumination, such as shown in figure 97, the Maximum intensity option produces more satisfactory results.

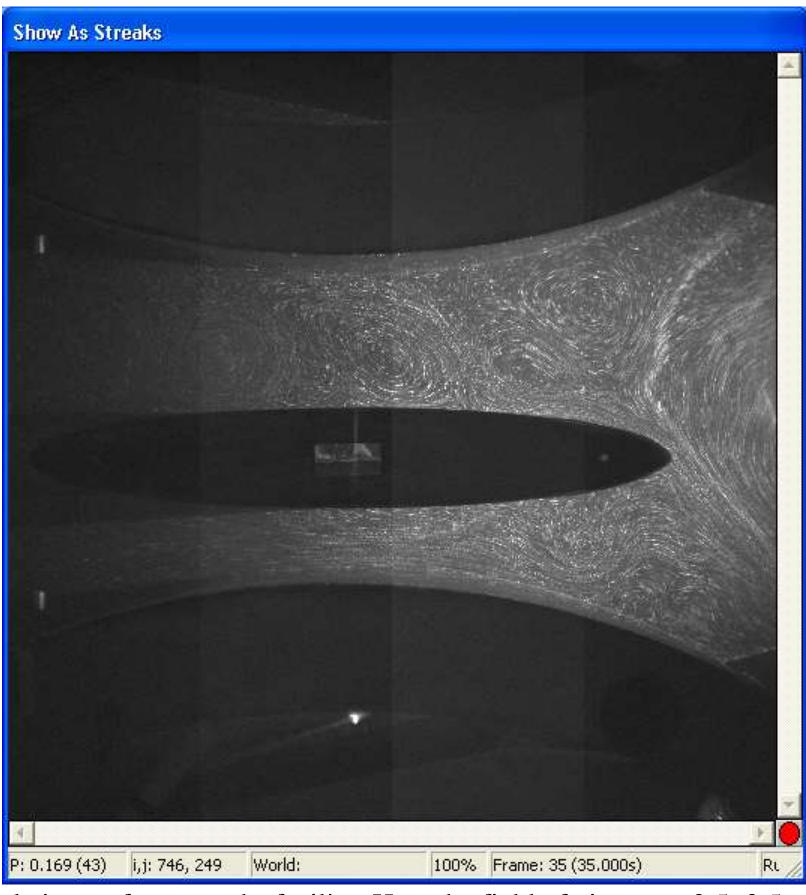


Figure 97: Example image from streaks facility. Here the field of view was 2.5×2.5m and particles illuminated by a 5W argon laser. The streaks show barotropic vortices interacting with the baroclinic hydraulic exchange through a strait containing an island.

The Synchronise display check box forces synchronisation such that each and every frame in the sequence is displayed, even if this slows the update rate below the desired frame rate. If Synchronise display is cleared and the computer cannot keep up with the desired frame rate, then frames are skipped to maintain that frame rate. In most cases streak images work best if frames are not skipped (*i.e.* you should normally have Synchronise display checked).

5.6.5.2 Particle Image Velocimetry

Toolbutton:

Shortcut:

Related commands: process Analyse_PIV(..)

Theory

The Particle Image Velocimetry (PIV) component of DigiFlow has a great deal in common with the pattern matching synthetic schlieren component (§5.6.4.3), and indeed many of the unique features in the PIV system owe their development to synthetic schlieren.

The PIV interface is divided into a number tabbed dialogs. The first two, Inputs and Outputs are mandatory and control what is to be processed, and where the results are to be stored, respectively. The remaining tabs may be used by experienced users to tweak the process to yield better performance in some situations. Only the first two tabs are available under Free DigiFlow licences.

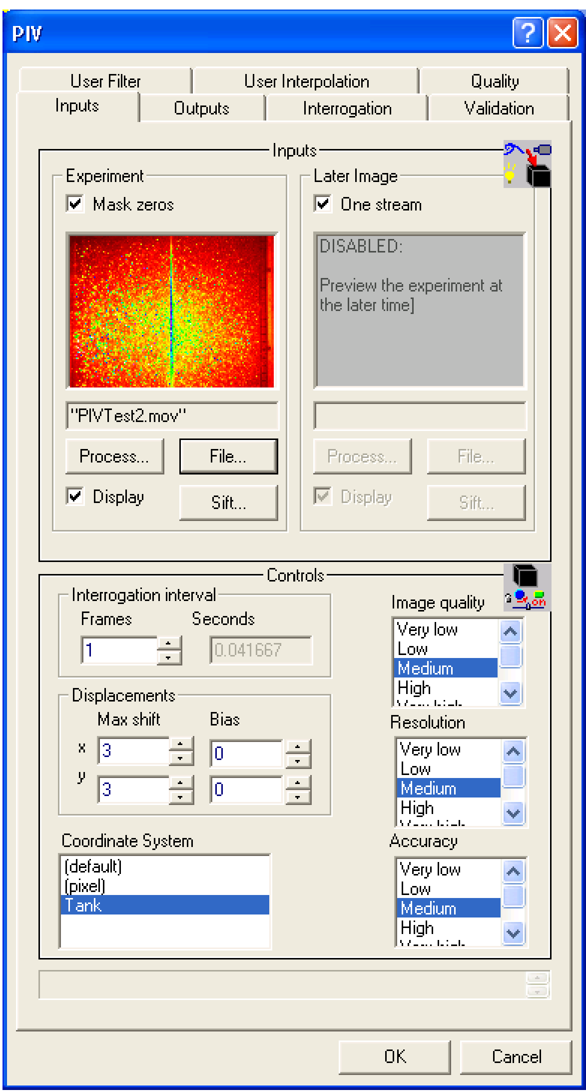


Figure 98: The Inputs tab of the PIV dialog.

Intputs page

The standard PIV process takes two input streams. The first, specified by the Experiment group, is the video sequence of the experiment itself. As normal, an image selector is used to specify the stream. This selector may be specified from a file by clicking the File button, in

which case the standard Open Image dialog box is produced. Alternatively, clicking the Process button will allow a source process to be used (refer to §6 on chaining processes for further details).

The second stream, the Earlier Image input, may be tied to the Experiment stream by the One stream check box, or taken from an independent data source. In either case the interval between these two streams should be specified in the Interrogation interval group. If One stream is used, then Interrogation interval is specified in frames. If separate image streams are used, then the Interrogation interval is specified as the time interval between the two streams.

The Mask zeros check box causes DigiFlow to ignore all pixels with an identically zero intensity. This feature is designed to allow simple masking of images. Such masking may be used to remove parts of the field of view that do not contain the flow. For example, it could be a static boundary to the flow, a free surface, or possibly an object moving through the flow. In each case external processing of the image sequence should be made to apply the mask prior to starting the PIV processing.

There are four groups of controls on the Inputs tab. The first controls maximum displacement that will be searched for. The values x Max shift and y Max shift are specified in pixels and are assumed symmetric about zero. These values relate to the maximum expected particle displacement but need to be as large as that shift (they parameterise the initial search space for the particle displacement, but DigiFlow will search a larger space if necessary). In most circumstances the default 3 pixels is adequate. Note that the computation required to determine the displacement (and hence velocities) increases approximately as the product of these two values, hence specifying excessively large values is counterproductive.

If the velocity field has a significant bias in one direction (*e.g.* there is a mean flow), then specifying a nonzero x Bias and/or y Bias will allow greater computational efficiency by permitting smaller values for x Max shift and y Max shift. The units of x Bias and y Bias are pixel displacements and have an effect similar to shifting the second image by negative the specified amount. For example, if there is a mean velocity down and to the right, then you would specify x Bias as positive and y Bias as negative.

To provide a simplified interface to the internal workings of the PIV algorithm, DigiFlow provides a range of predefined settings that have the effect of producing different resolutions and accuracies. The Image quality, Resolution and Accuracy list boxes both have six possible settings: Very low, Low, Medium, High, Very high and Best. The choice will depend on a combination of the intended purpose of the results, and the time available to undertake the processing, and the quality of the original images. The fastest processing is achieved at the Very low end of both scales, while the most detailed and accurate measurements are obtained with both Resolution and Accuracy set to Best. In the latter case, even with relatively basic analogue video equipment, the accuracy with which the particle displacement may be detected can be better than 1/100 of a pixel in ideal circumstances (*e.g.* no particles disappearing), and the spatial resolution of the measurements is a few pixels. By default, the Image quality, Resolution and Accuracy controls will be enabled. However, if the Automatic check box for the Validation group on the Advanced tab is cleared, then Image quality is disabled. Similarly, the Interrogation window group on the Advanced tab is cleared, then the Resolution and Accuracy controls will be disabled.

The final input on this tab is Coordinate System. This specifies the coordinate system that will be used to relate pixel to world coordinates. The coordinate system is assumed to have been defined in the mid-plane of the experiment. See §5.2.6 for further details on setting up a coordinate system. Note that if you select a pixel coordinate system, then the To World Coordinates tool (§5.7.6) may be used to retrospectively convert the pixel PIV results to a

world coordinate system. (It is generally better, however, to compute the PIV using the appropriate world coordinate system in the first instance.)

Outputs page

The Outputs tab controls the destination and scale of the output from the PIV calculation. This dialog page consists of two image selectors, each with its own Colour and File buttons. The destination for the output stream is selected by clicking the File button (thus starting the standard Open Image dialog box), while the colour scheme to be used with the stream is (optionally) selected with the Colour button. At least one of the four output streams must be given a file name before OK will close the dialog box and start the process.

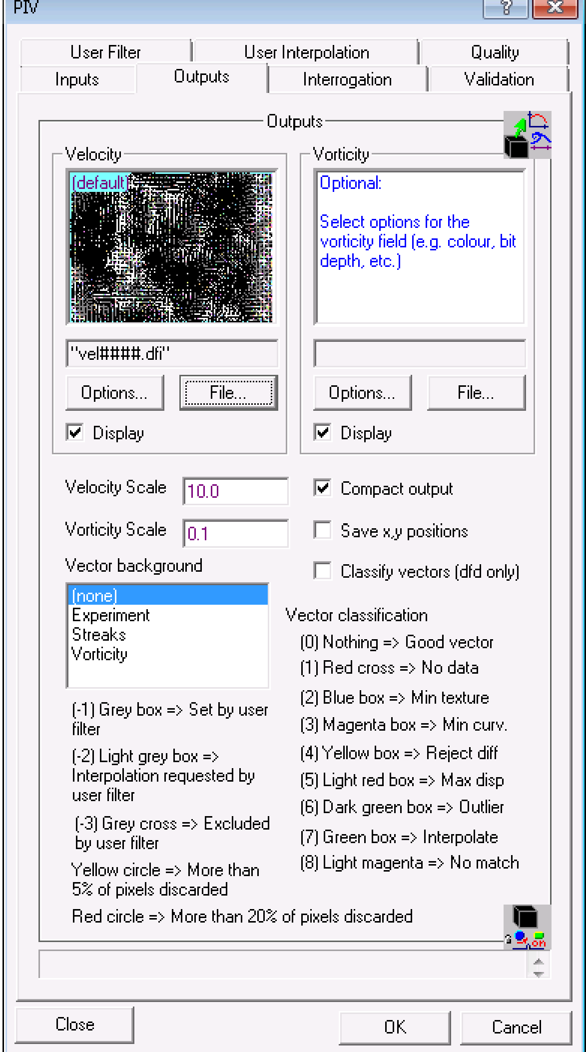


Figure 100: The Outputs tab of the PIV dialog.

The mandatory Velocity output selector will contain an image of the velocity field calculated. It is recommended that you specify either a DigiFlow Drawing file (.dfd) or a DigiFlow Image file (.dfi) rather than an industry standard raster image format for this output. In general, a .dfi file is to be preferred. In a .dfd the velocity data as ASCII data in conjunction with a series of drawing commands. This format is very convenient if you are using other software to process the results as reading these files is straight forward. They are, however, not very compact. Using a .dfi file stores the velocity data as velocity data, but allows this to be processed by DigiFlow as though it were an image. For example, the time average facility (see §5.6.1.1) and most of the other manipulation tools can be used to process the velocity

data. In general, saving the data in .dfi format will be preferable until you have finished all processing.

The scale length of the velocity arrows is determined by the Velocity scale setting. A unit value draws the arrows of a length equal to the distance the particles have moved in the time interval between the two images used in the PIV calculation. Increasing Velocity scale causes the length of the velocity to increase, *etc*. This approach allows Velocity scale to be largely independent of the coordinate system used. For many flows, a value of 2 to 10 is appropriate. Note that if Velocity scale is negative, then the arrows are drawn in the reverse direction.

The background to the velocity field may be selected through the Vector background control. Selecting (none) gives a plane white background for the velocity field map, and the output stream stores only the velocity field itself. When Vorticity is selected for the Vector background, then the vorticity field is calculated and stored in the output stream; the vorticity field is also displayed as a colour map behind the velocity field. The Experiment and Streaks options place an image of the experiment behind the velocity field (also storing it in the output stream). The Experiment option is self-explanatory, while the Streaks option synthesises a streak image (see §5.6.5.1 for an example of a streak image) to be displayed.

Selecting the Compact check box causes DigiFlow to save an approximation to the calculated velocity field by only saving the gradient at the nominal location of the interrogation windows used to calculate the gradient. DigiFlow will automatically expand out this approximate velocity field, when it is reloaded, to produce one that is very close to that saved without the Compact option. The files produced, however, are much smaller. This option works well with either no background, or using the vorticity field as a background. However, as the background is compressed in the same way as the velocity field, this option does not work so well when selecting either the experiment or particle streaks as the background.

The Save x,y positions is not normally necessary. This will add two data planes to each .dfi output by the PIV process, with one plane containing the x coordinate of every pixel, and the other the corresponding y coordinate. Note that the Quality output (see the discussion below on the Quality tab) also provides access to the location of each of the interrogation vectors.

If Classify vectors is checked, then the velocity vectors produced include an indication of the quality of the vector. This is indicated by a box or cross drawn at the base of any suspect vector, as per the table below. Note that the Progress window (which is always produced to show the progress of the PIV calculation) will also show this information, even if Classify vectors is turned off. At present, classification will only be indicated on output to .dfd files.

Symbol	**Description**	**Advanced Control page**
Red cross	No valid velocity vector	
Blue box	The image does not contain an adequate texture for the matching to be reliable.	Interrogation: Min range Interrogation: Min texture
Magenta box	The difference function being minimised does not have a well-defined peak.	Interrogation: Min curvature
Yellow box	The value of the difference function is too large.	Validation: Reject difference
Light red	The best match is found beyond the limit of the permissible shifts.	Inputs: Displacements
Dark green box	The optimal match produced	Validation: Outliers

	an outlier. This has been replaced by an interpolated value.	
Green box	Vector is the result of interpolation from surrounding vectors.	
Light magenta box	A best match could not be found.	
Yellow circle	At least 5% of pixels discarded	Interrogation: Big differences
Red circle	At least 20% of pixels discarded	Interrogation: Big differences

The vector field may be superimposed on a range of backgrounds. These are selected by the Vector background list box. If (none) is specified, then a plain, white background is used, whereas Experiment leads to the vectors being superimposed on the corresponding experimental image. Similarly Vorticity draws the arrows on an image of the vorticity field.

The Vorticity output selector is optional, and should normally specify a raster image format file. The Scale setting controls the rendering of the vorticity as a colour map. The value specified here will be taken as the saturation limit of the false colour map produced. Thus decreasing Scale amplifies the vorticity map. Note that this scale is used to determine the scaling of the vorticity map behind the velocity vectors if Vorticity is specified for Vector background, regardless of whether a separate vorticity output file is being created. Since vorticity has dimensions of inverse time (and so does not have a length scale), the scaling of vorticity is largely independent of the coordinate system selected.

Note that outputting the velocity field to a .dfi file works best when the coordinate system is essentially aligned with the image. In all cases the velocities are determined on a regular grid in pixel space. When output to a .dfd file, the velocity vectors will be displayed in world coordinates with a standard Cartesian grid in physical space; this may mean that the original pixel coordinates are no longer Cartesian. When output to a .dfi file, the original pixel coordinates remain Cartesian, and the world system may remain distorted.

Interrogation page

The Interrogation tab is identical to that for synthetic schlieren described in §5.6.4.3. Most users will not need to disable the automatic settings on any of the controls.

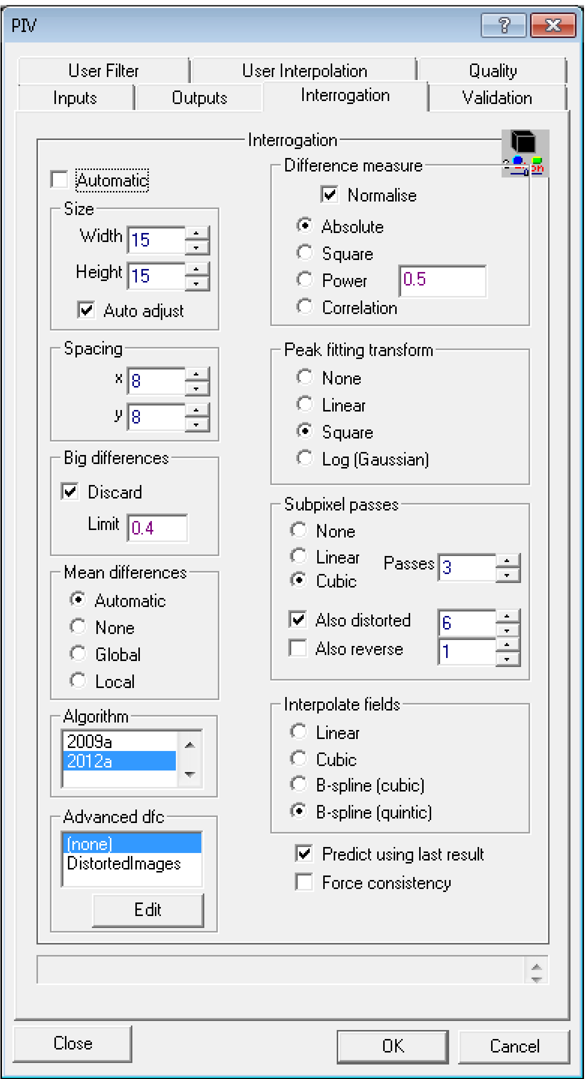


Figure 103: The Interrogation tab of the PIV dialog.

A unique feature of DigiFlow, available only on fully licensed copies, is the ability to use dfc macro code to fine tune various aspects of the pattern matching process. The Advanced dfc group controls the specification of this code. To provide code, select the required category from the drop-down list then click Edit. This will start up a dfcConsole (see §5.2.10 for details) to edit the code.

The precise requirements for the dfc code vary depending on the task required of it. In all cases, information is passed to the code through specific variables. The code can then change the contents of the variables, but must not redefine their type or location. Consequently, assignment to one of the arrays passed to the code should be through specified elements (*e.g.* `A[1,2] := …;`) or ranges (*e.g.* `A[:,:] := …;`) and not directly to the name (*i.e.* not `A := …;`). Some example codes for specific purposes are given below.

The easiest way to determine what variables are available in a given code segment is to include a call to `view_variables(..)` while developing the code. Note, however, that the memory space and variables names are shared by *all* code segments. Thus, a variable defined by the user in one section of code will be available to other sections of code.

To suppress a given piece of code, either ensure that it is blank or have

```
# Do not use
```

as the first line. This will suppress execution of the code. Having `quit` as the first statement (without the 'Do not use' comment) start the code executing, but stop it again immediately. While this approach works, there is an added computational overhead to it.

The following variables are passed to the code for the different calls:

For any code

`callFor`	String	Identifies the purpose of the call. See below for specific calls.
`invokedBy`	String	Specifies the condition under which the call is made.
`dlg`	Compound	The dialog structure used to define the PIV process
`A`	`nx`×`ny` array	The image (undistorted) of the flow at the earlier time
`B`	`nx`×`ny` array	The `nx`×`ny` image (undistorted) of the flow at the later time
`hA`	Handle	Handle for the window showing the flow at the earlier time
`hB`	Handle	Handle for the window showing the flow at the later time
`Time`	Compound	Contains time information relating to the earlier time
`nx`	Integer	The width of the image stream being processed
`ny`	Integer	The height of the image stream being processed
`X`	`nx`×`ny` array	World x coordinate for each pixel
`Y`	`nx`×`ny` array	World y coordinate for each pixel
`xMin`	Real	The minimum world coordinate in the x direction (real)
`xMax`	Real	The maximum world coordinate in the x direction (real)
`yMin`	Real	The minimum world coordinate in the y direction (real)
`yMax`	Real	The maximum world coordinate in the y direction (real)
`u`	`nx`×`ny` array	Current estimate of the displacement in x direction
`v`	`nx`×`ny` array	Current estimate of the displacement in y direction
`nxZ`	Integer	The number of interrogation regions horizontally the image
`nyZ`	Integer	The number of interrogation regions vertically in the image
`xyCentre`	`nxZ`×`nyZ` array	The centres, in world coordinates, of each interrogation region
`ijCentre`	`nxZ`×`nyZ` array	The centres, in pixel coordinates, of each interrogation region
`iteration`	Integer	Counter for iterative processes; −1 if not an iterative process

callFor = "GetImages" – all algorithms

callFor = "PredictDisplacement" – 2014a and later algorithms

callFor = "InterpolateDisplacement" – all algorithms

`invokedBy`	String	One of `"Forward"`, `"Reverse"`, `"Distorted"`,

		"FinalOutliers", "FinalField"
uInterrog	nxZ×nyZ array	Stores the x component of the current displacement on the coarse grid.
vInterrog	nxZ×nyZ array	Stores the y component of the current displacement on the coarse grid.
uField	nx×ny array	Stores the interpolated x component of the current displacement at the full image resolution..
vField	nx×ny array	Stores the interpolated y component of the current displacement at the full image resolution..
rect	Compound	The rectangle within U,V to which u,v must be interpolated.
callFor = "DistortionField" – 2017a and later algorithms		
xDisp	nxZ×nyZ array	The x-component of the current estimate of the displacement field.
yDisp	nxZ×nyZ array	The y-component of the current estimate of the displacement field.
xDist	nxZ×nyZ array	The x-component of the displacement field used to distort the images during a distortion pass. On entry, this will be the same as xDisp. Typically, this hook is used to filter this distortion field.
yDist	nxZ×nyZ array	The y-component of the displacement field used to distort the images during a distortion pass. On entry, this will be the same as yDisp. Typically, this hook is used to filter this distortionfield.
xZSize	Integer	The horizontal size of the interrogation window used for pattern matchine the distorted image.
yZSize	Integer	The horizontal size of the interrogation window used for pattern matchine the distorted image.
nMicrosteps	Integer	The distortion process is achieved by an advection equation. This specifies the number of intermediate steps used.
weightUnfilteredDisplacement	Real	The weighting applied to the current estimate of the displacement (xDisp) compared with the displacement field used to distort the image (xDist).
scaleDisort	Real	The weighting to be applied to disorting the image. A value of 1.0 will try to distort the images using the current estimate of the displacement so that they match.
relaxation	Real	The correction determined with the distortion pass is applied with a relaxation factor.
callFor = "DifferenceFilter" – 2014a and later algorithms		
angle	nx×ny array	The angle for the major axis to be applied to the difference filter
ellipt	nx×ny array	The ellipticity to be applied to the difference filter
callFor = "DistortedImages" – 2012a and later algorithms		
P	nx×ny array	The distorted image at the earlier time
Q	nx×ny array	The distorted image at the later time

GetImages - all algorithms

This code segment is called immediately after the pair of images is read in. One possible use of this code is to pre-process the images to apply a mask, either a static one or one created dynamically based on the contents of the images. Typically, it will be more efficient to develop the processing algorithm outside the pattern matching process as debugging it in context is less straight forwards. (A mask is generally implemented by setting to zero the pixels that are not to be included in the pattern matching process.)

As an example, the following code was developed to remove the (relatively faint) images of fixed bubbles behind the illuminated PIV plane.

```
if (Time.iNow = 0) {
  # This setup is only executed once
  #
  # Recover the region selected for processing
  i0 := dlg.Experiment_Region.xMin;
  i1 := dlg.Experiment_Region.xMax;
  j0 := dlg.Experiment_Region.yMin;
  j1 := dlg.Experiment_Region.yMax;
  # Read in the 'background' image, created by determining the
     minimum
  # intensity over time for each pixel in the image
  back := read_image("MinIntensity.dfi");
  back := back[i0:i1,j0:j1];
  # Create a mask highlighting the bubbles
  thresh := 0.015;
  noBubbles := filter_median(back,11,11);
  mask := back - noBubbles > thresh;
  mask := filter_median(mask,3,3);
};

# Fix the two input images by removing the bubbles
A -= back;
B -= back;
A[] := where(mask and A < 2*thresh,0,A) + 1/255;
B[] := where(mask and B < 2*thresh,0,B) + 1/255;
```

Iin this case, the file MinIntensity.dfi was calculated in advance using Analyse: TimeAverage with Method set to Min.

PredictDisplacement - 2014a and later algorithms

At the start of the processing of each new pair of frames, DigiFlow requires an initial guess for the displacement field. By default, this is determined by a pixel-resolution pass for the first velocity field, whereas for subsequent fields it is determined either by the displacement from the previous calculation, or by another subpixel pass. The code specified to PredictDispalcement is called after any pixel resolution pass has been complete.

The current displacement field estimate is provided in `u[:,:]` and `v[:,:]`. Note that any assignment statements should be made to `u[:,:]` and `v[:,:]` rather than simply to `u` and `v` to ensure the same memory is used.

An example of where this facility is useful is given in the following example. Here, the file MeanVelocity.dfi contains the mean velocity that is to be used in place of a ‘zero’ prediction for the velocity field for the first iteration. In most cases, it is unlikely that the mean velocity is a less good prediction than the previous velocity, unless the successive velocity fields departure from this mean are poorly or negatively correlated.

```
# Use pre-existing velocity field to make prediction
if (Time.iNow = 0) {
  predFile := "MeanVelocity.dfi";
```

```
  predUV := read_image(predFile);
  predDet := read_image_details(predFile);
  view_variables();
  # Convert world velocities to pixel displacements
  u[] := predUV[:,:,0] * predDet.tStep/predDet.dx;
  v[] := predUV[:,:,1] * predDet.tStep/predDet.dy;
};
```

Note that the displacement is specified over the entirety of the image plane, not just at the centres of the interrogation windows.

InterpolateDisplacement – all algorithms

This code segment is called immediately after DigiFlow has interpolated a displacement field from the resolution of the interrogation grid (`uInterrog,vInterrog`) up to the resolution of the image being processed (`uField,vField`). The default interpolation scheme is set by the Interpolate Fields group on the Interrogation tab and acts to interpolate the two displacement components separately.

Displacement fields are interpolated at multiple different points in the pattern matching algorithm. These are distinguished by the `invokedBy` string. Both `"Forward"`, `"Reverse"` are used when doing reverse passes, `"Distorted"` indicates a distorted pass (with `iteration` giving the distorted pass number), `"FinalOutliers"` is during the removal of any remaining outliers, `"FinalField"` indicates generation of the final displacement field.

DistortionField – 2017a and later algorithms

During the 'distorted passes', DigiFlow uses the current estimate of the velocity field to distort both the images being used for PIV to its best estimate of the state at the mid-point in time between the two. The DistortionField code segment is called before DigiFlow produces the distorted pair of images, but after it has determined the displacement field that will be used to drive the distortion. The code segment can be used to modify this distortion field. Typical examples might include filtering the distortion field to try to ensure the image distortion is smooth.

The following code segment is motivated by a desire to use a fine spacing of the interrogation windows in cases where the particle seeding density and image quality may not be as high as desired. Here, a low pass filtering of the distortion field will help suppress high-wavenumber noise being introduced.

```
# Variables on entry
#  xDist,yDist The default distortion field
#  iteration   The iteration counter
#  Time        The current time information

xDist[] := filter_low_pass(xDist,3,3);
yDist[] := filter_low_pass(yDist,3,3);
```

DifferenceFilter – 2014a and later algorithms

With algorithms dated 2014a and later, a spatially tapered elliptical filter is applied when calculating the difference measure between the distorted images. The idea of this is to provide a greater emphasis for one set of directions compared with others.

By default, the major axis of the elliptical filter is aligned with the current estimate of the velocity, while the degree of ellipticity is controlled by $(\partial u/\partial y)^2 + (\partial v/\partial x)^2$. The sample code below simply displays the orientation and ellipticity fields.

```
if (Time.iNow = 0 and iteration = 0) {
```

```
  # Create view handles for diagnostics on first call
  hAngle := view(angle,-pi,pi);
  view_colour(hAngle,"circular");
  view_title(hAngle,"Difference Filter: Angle");
  hEllipticity := view(ellipt,0,5);
  view_title(hEllipticity,"Difference Filter: Ellipticity");
};

view(hAngle,angle,-pi,pi);
view(hEllipticity,ellipt,0,5);
```

DistortedImages - 2012a and later algorithms

During the ‘distorted passes’, DigiFlow uses the current estimate of the velocity field to distort both the images being used for PIV to its best estimate of the state at the mid-point in time between the two. The DistortedImages code segment is called immediately after DigiFlow produces the distorted pair of images and before it starts the pattern matching process on the distorted pair. The DistortedImages code can therefore be used to modify the distorted pair, or to provide diagnostic information about them.

The following code segment is motivated by the observation that for very small particles in sharp focus can change substantially in overall intensity from one frame to the next. This change can be due to the fill factor for the image sensor being less than 100%, or due to particles near the fringe of the light sheet moving in or out of the region of strong illumination. In either case, it can be desirable to decrease the impact of these extreme changes without completely eliminating the information provided by the particle. This process is one of fine-tuning, and should not be used until the velocity field is very close to being correct and so the distorted images are nearly perfectly matched. The modification made here to the images is to maintain the structure, but decrease the intensity difference in regions where this difference is very strong.

```
# Variables on entry
#  P,Q          The distorted images
#  iteration    The iteration counter
#  Time         The current time information

dPQ := P - Q; # Difference between distorted images

if (Time.iNow = 0 and iteration = 0) {
  # Create view handles for diagnostics on first call
  hScat := 0;
  hDist := 0;
  hDist := view(hDist,dPQ,-0.2,0.2); # Display unedited difference
  view_zoom(hDist,0.25);
  view_fit_to_zoom(hDist);
  view_title(hDist,"Difference between distorted images");
};


view(hDist,dPQ,-0.2,0.2); # Display unedited difference

# A scatter plot (optional) can provide good diagnostic information
scatter := scatter_to_array(make_array(0,256,256),255*P,255*Q,1,1);
scatter := log(scatter max 1e-2) + 2;
scatter /= max_value(scatter);
hScat := view(hScat,scatter);
view_title(hScat,"Scatter plot of intensities in distorted images");

if (iteration >=2) {
  # Only make adjustments to distorted images if sufficient
  # distorted passes have already been completed
  OK := P <> 0 and Q <> 0; # Mask out any zeros
  thresh := mean(abs(dPQ));
  sPQ := P + Q;
  delta := sign(dPQ)*(thresh + (abs(dPQ)-thresh)/4);
  P[] := where(abs(dPQ) > thresh and OK,(sPQ+delta)/2,P);
```

```
      Q[] := where(abs(dPQ) > thresh and OK,(sPQ-delta)/2,Q);

      # Update difference image as diagnostic
      # Could also update scatter plot
      dPQ := P - Q;
      view(hDist,dPQ,-0.2,0.2);
    };
```

Note that the code assigns new values to the distorted images `P[:,:]` and `Q[:,:]`, rather than simply to `P` and `Q`. This ensures that the same memory is used for the updated arrays as was used to pass them to the code. There are both computational reasons for wanting to do this and it ensures the returned arrays have the same dimensions as the input ones.

Validation page

The Interrogation tab is nearly identical to that for synthetic schlieren described in §5.6.4.3. Most users will not need to disable the automatic settings on any of the controls. One difference is that the Projection group contains an additional option, allowing projection of the velocity field onto either irrotational or incompressible (solenoidal) spaces. The latter is most likely to be of use when considering two-dimensional flows, as it only attempts to make the divergence in-plane measured velocity field vanish. In most other circumstances the projection should be turned off.

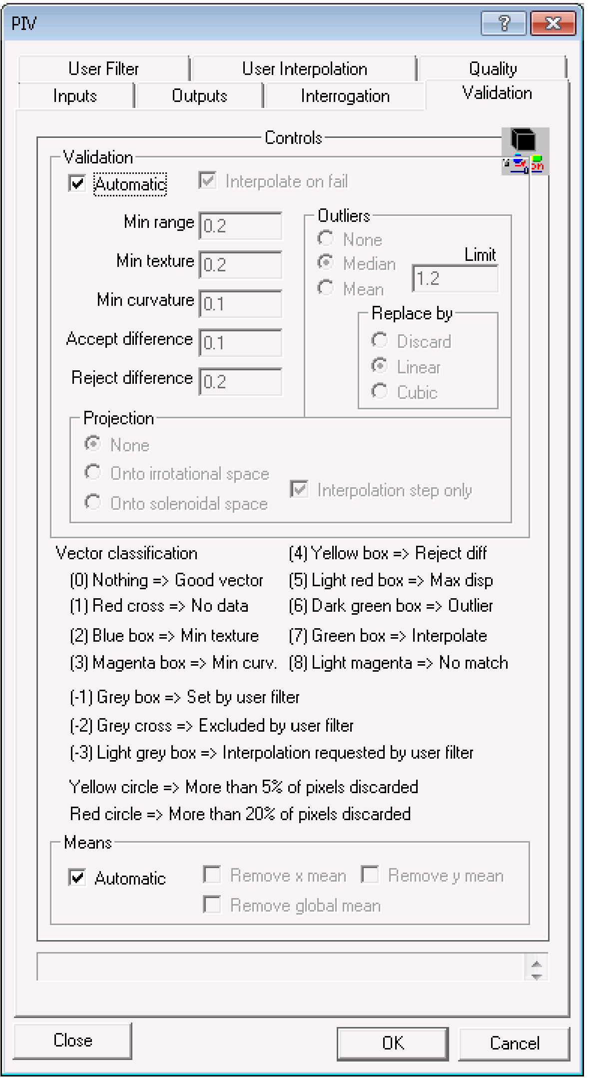


Figure 106: The Validation tab of the PIV dialog.

User filter

The User Filter tab provides the user with the ability to supplement or override DigiFlow's normal validation filters. When enabled DigiFlow provides the user's dfc code with the two images as the variables `Pa` and `Pb`, along with the pixel displacements as the two-dimensional arrays `u` and `v`. The locations of these vectors are supplied in `x` and `y`, while the current state of the vector is indicated by `state`. Finally, the string `calltype` indicates the point in the algorithm when the call to the filter is made. This may take one of the values `"Pixel"`, `"SubPixel"`, `"Reverse"`, `"Distorted"` or `"Final"`. The values taken by the `state` array reflect DigiFlow's default assessment of the individual displacement vectors. A list of the categories is given in the lower half of the User filter tab.

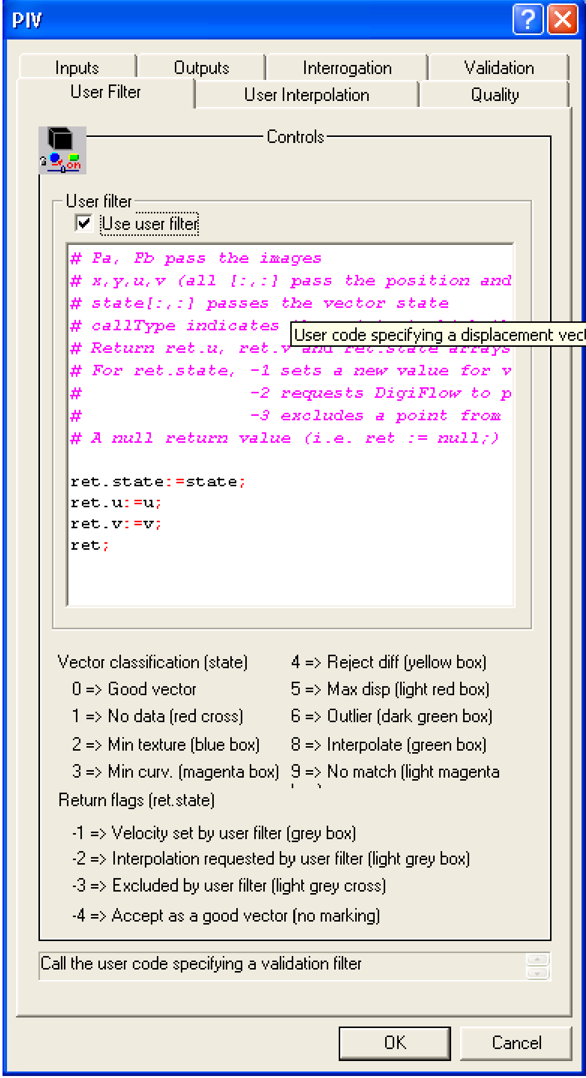


Figure 110: The User Filter tab.

The user code should return a compound variable containing the components `.state`, `.u` and `.v`, each of which are arrays of the same size as the corresponding arrays provided to the dfc code. The return values in `.state` request DigiFlow to treat the displacement vector in the manner specified in the lower half of the User filter tab. If `.state` for a given vector is set to -1 then the vector supplied `.u` and `.v` will be used in place of that calculated by DigiFlow.

User interpolation

The User interpolation tab provides a way of customising one of the key steps in the pattern matching algorithm, namely interpolating the velocity field from the location of the interrogation vectors to the entire image plane.

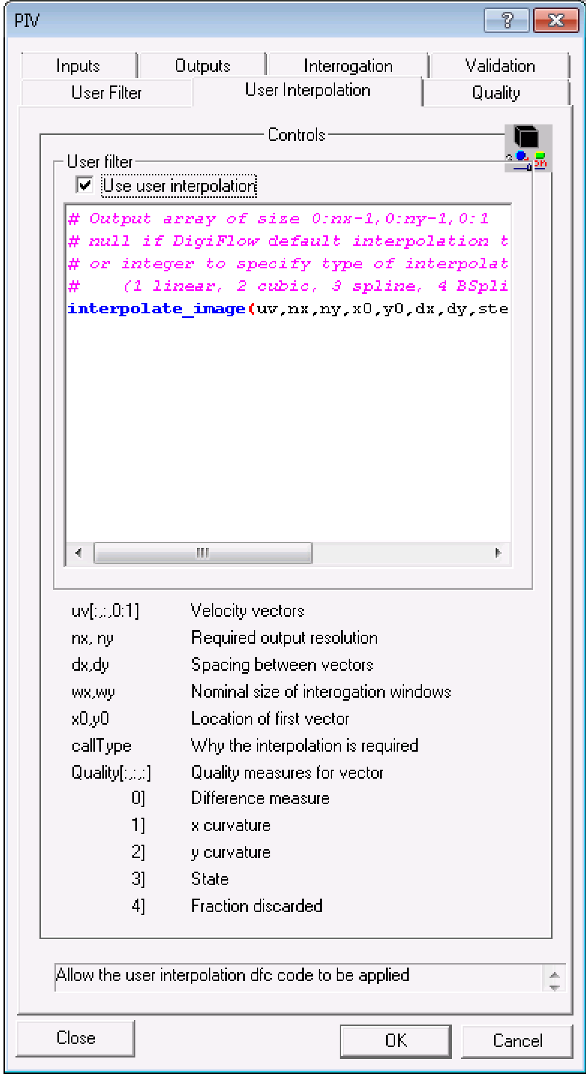


Figure 111: The User interpolation tab provides a way of customising the interpolation step whereby the velocities at the interrogation points are interpolated to the entire image plane.

The controls available on this tab are identical to those for the pattern matching in synthetic schlieren (see §5.6.4.3). The example given in figure 111 uses a biquadratic fitted using a least-squares routine for all parts of the PIV process where interpolation is required. Note that this process only gives approximate interpolation as the least squares solution will not generally pass through the corresponding mesh points. Substituting the following code will keep the default behaviour for all except the generation of the intermediate velocity field used to distort the images:

```
if (callType = "Distorted") {
  sx := x_size(uv);
  sy := y_size(uv);
  fit_image_b_spline(uv,nx,ny,x0,y0,dx,dy,nxParts:=sx/2,
     nyParts:=sy/2,xOrder:=3,yOrder:=3);
} else {
  null;
};
```

Here, we detect when the interpolated field is required for image distortion using the callType variable. If not, then returning a **null** indicates to DigiFlow to use the default interpolation. Here, when callType is "*Distorted*", we use a least squares fit of the velocity field using cubic b-splines to reconstruct a smoothed high-resolution version of the velocity field with which to distort the images prior to the next stage in the pattern matching algorithm.

Quality output

The Quality tab provides the option of outputting information that DigiFlow generates to assess the quality of the individual velocity vectors.

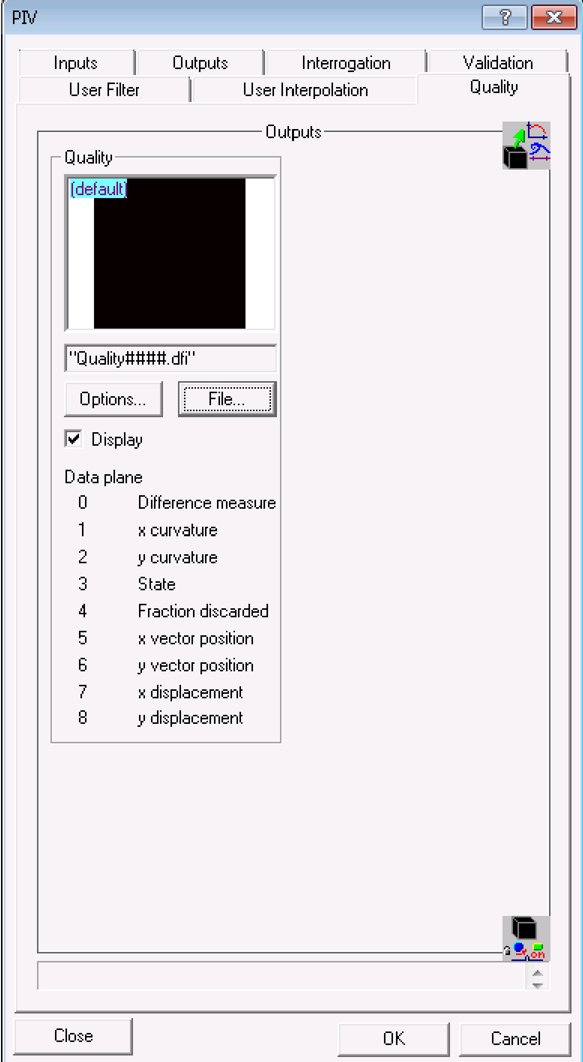


Figure 112: Optional output of information relating to the quality of the velocity vectors.

The optional output stream created by this feature must be saved as a .dfi file. The pseudo image created contains multiple planes of image data, as indicated in the dialog box. Note that unlike with a .dfi file containing velocity information, DigiFlow does not format the image in any particular way for display. Opening a quality .dfi file will simply display the first image plane. The contents of the quality image is identical to that for Synthetic Schliere, thus the reader is referred to §5.6.4.3 for further details.

Post processing

Selection of the most appropriate output file format (between .dfd and .dfi) depends on the type of post processing to be undertaken.

If the .dfi format is selected, then the PIV velocity files may be fed back into DigiFlow as multi-plane images containing the velocity field. These can be processed using most of the standard DigiFlow tools, preserving the nature of their contents. For example, the Analyse: Time Average facility can act upon a sequence of PIV velocity files to produce the time average velocity field. Similarly, the various other time series tools described in §5.6.1 can operate on these images, as can the general manipulation tools Recipe, Transform Intensity and Combine Images (see §§5.7.1, 5.7.2 and 5.7.3). There are standard recipes in the Recipe facility to aid with basic manipulations of this data. For example, the recipe Velocity.Background.Divergence recipe lets you change the background of the velocity field from the one saved during the PIV processing to display the in-plane divergence field. Similarly, there are recipes for vorticity, stream function, velocity potential, shear, *etc.* Note that for PIV data, velocity gradients are obtained by a finite difference operation of the velocity field.

Saving the output in .dfd format is appropriate if post processing is to be undertaken using a third party or user-written program as the .dfd file contains an ASCII representation of the velocity field. Note that you can always convert a .dfi file into a .dfd file using Edit Stream (§5.1.6) or one of the other related image manipulation tools by simply specifying a .dfd file for the output.

Stereo PIV

A single PIV calculation will provide two velocity components parallel to the image plane. However, if there are a pair of simultaneous recordings of the image plan from somewhat different angles, then these can be used as a stereo pair to recover three velocity components in the image plane. The documentation here is not intended to provide a full description of how to achieve this, but rather to act as a pointer to the separate document, DigiFlow_StereoPIV.pdf (and DigiFlow_StereoPIV.htm) that provides further information on this specialist procedure.

There are a number of key steps in obtaining the three-component two-dimensional (3C2D) velocity field:

1. Capture of a synchronous stereo pair of images of the flow
2. Processing, in pixel coordinates, of the two apparently two-dimensional velocity fields (one from each of the stereo pair).
3. Determination of the two-dimensional (in-plane) coordinate system for each of the images in the pair.
4. Determination of the stereo-pair to three-dimensional coordinate system and its derivative (used for transforming the velocities)
5. Utilisation of the coordinate systems to transform the stereo pair of velocity fields into the 3C2D velocity field in world coordinates in the image plane.

At present, steps 3, 4 and 5 are handled through macros utilising a tailor-made set of built-in dfc functions for improved efficiency.

5.6.6 Particle Tracking Velocimetry

5.6.6.1 Tracking particles

Toolbutton:
Shortcut:
Related commands: `process Analyse_PTVTrack(..)`

Background

Particle Tracking Velocimetry (PTV) differs from Particle Image Velocimetry (PIV) in a fundamental way. Whereas PIV (described in §5.6.5.2) relies on pattern matching in an essentially Eulerian way, PTV seeks to identify individual particles (or other equivalent features) and follow them in a Lagrangian sense. As PIV is the more widely used of these techniques, it is worth discussing the relative merits of the two approaches.

The strengths of PIV are that is fairly robust to noise and has excellent velocity resolution (the accuracy with which displacements may be obtained is a function of the cell size and the distribution of features within it rather than the pixel resolution). The spatial resolution is inversely proportional to the cell size: the overall data quality is thus a compromise between velocity and spatial resolution. The main disadvantages are the considerable time required to compute the optimal correlation and the inability to cope with any structure across the illuminated plane (*i.e.* velocity gradients parallel to the viewing direction). In general the method does not allow individual particles to be tracked, and hence has no immediate access to Lagrangian descriptions. However, it is a relatively simple matter to add some degree of particle tracing once the velocity field is known, and hence access the Lagrangian nature of the flow.

Particle tracking offers a more fundamental approach to PIV. There are two main approaches which are exactly equivalent to the manual methods of analysing streak (or multiple exposure) photographs and multiple (time series) photographs. In the streak photograph method, the effective camera shutter is opened for a long time during which the particles move many particle diameters. This long exposure may be produced directly with a suitably slow shutter speed, or synthesised by combining multiple exposures (*e.g.* ORing a sequence of video frames using a digital frame grabber with a shutter speed equal to the field rate - the DigiFlow facility described in §5.6.5.1 has this as an option). Once the streaks have been produced, image processing techniques may be applied to locate them and analyse their shape, orientation etc.

The alternative of utilising a time series of images offers a greater volume of information on the particle positions as a function of time, especially in the context of digital image processing where quantisation yields a relatively low spatial and intensity resolution. Knowing the approximate location of a particle at a relatively large number of times enables a much more accurate estimation of the position of a particle at a given time, and of its velocity, provided the sampling frequency is much higher than the highest frequency in the particle motion. To make use of this information some method must be developed for tracking particles from one image to the next. In the limit of particles moving only a small fraction of their diameter between each sample, the process of matching particles in one image with their position in the next image is straight forward - the particle images closest together in two adjacent samples will correspond to the same physical particle. However, if the particles may move many diameters between samples, more sophisticated algorithms must be employed.

The algorithm used in the matching process may utilise spatial and temporal information in addition to particle characteristics and prior knowledge of the flow. Generally, only some of these features will be needed to determine which particle image is which particle. For example, if spatial correlation is not utilised, then two-dimensional projections of three-

dimensional flows with significant velocity gradients parallel to the direction of viewing, may be analysed (recall that PIV techniques are unable to cope with such images). Moreover, the basic approach is not limited to a two-dimensional projection of a three-dimensional flow but is capable of full three-dimensional analysis. By applying the matching process repeatedly, time-series for individual particles may be obtained to describe some of the Lagrangian nature of the flow.

The accuracy with which the velocities may be measured is limited by the accuracy with which the individual particle images may be located and the time period over which the velocity may reasonably be evaluated (this must be shorter than the period corresponding to the maximum frequency in which you are interested). The accuracy of location depends in turn on the particle size, the bit depth and quality of the images, and the method used to determine their positions. In general, the velocity resolution will be less than that for the cross-correlation approach, but is nevertheless excellent in many situations. The spatial resolution is limited primarily by the number of particles in the flow: the more particles, the higher the resolution. In practice the resolution of video technology and the frame grabber imposes the most stringent limitation on the number of particles able to be tracked. Eulerian as well as Lagrangian descriptions may be obtained, utilising a suitable interpolation method, if the particle seeding density is sufficiently high.

The techniques and algorithms used by DigiFlow are based on those originally developed in 1988 and described by Dalziel (1992). These same techniques and algorithms were incorporated in the DigImage processing system. These algorithms have been refined and enhanced in DigiFlow to improve computational efficiency and, more crucially, to improve the overall performance of the particle tracking process. The interface with these algorithms has been greatly simplified when compared with DigImage, making the tracking process more generally accessible.

This section outlines and describes the two-dimensional particle tracking technique utilised by DigiFlow. This method represents an efficient, reliable approach to tracking particles from a two-dimensional projection of a flow. The computation required to analyse each frame pair increases only slightly faster than linearly with the number of particles, allowing very high processing rates.

Particle location

The basic strategy behind the particle location is to scan through the image for *blobs* that have an intensity satisfying some threshold requirement. If a blob is found, then its characteristics are determined and compared against a set of requirements for the blob to be considered a particle. If the blob satisfies these requirements, it is recorded as a particle, if it does not, it is discarded.

By scanning through the image with a range of different thresholds, it is possible to pick up particles with a broad range of intensities, allowing optimal performance. A blob that was rejected at one threshold may well be picked up as a particle at another threshold.

The particle location procedure ultimately records not only the location of the particle (as determined by its volume centroid, relative to the threshold, but also a broad range of other particle characteristics, some of which are used in the subsequent matching process.

Matching algorithm

Once all the particles in an image have been found (at $t = t_{n+1}$, say), they need to be related back to the previous image ($t = t_n$, say) to determine which particle image is which physical particle. In DigiFlow we use a modification of what is known in operations research as the *Transportation Algorithm*. This approach was that developed by Dalziel (1992). While the

problem solved by the transportation algorithm may be represented as a 0-1 totally unimodular integer linear program, it is more efficient and illuminating to take a graph theory approach.

The idea is to choose a set of associations between two sets of entities, such that the set of associations is optimal in the sense that it minimises some linear function of the associations it includes. For the particle tracking, one of the sets is the set of particles **P** at $t = t_n$ and the other the set of particles **Q** at $t = t_{n+1}$. We shall start by assigning a label to all the particles images in the two images. At $t = t_n$ the particle images are labelled p_i for i=1 to i=M, while at $t = t_{n+1}$ they are labelled q_j for j=1 to j=N. Each p_i or q_j contains not only the location of the particle, but other characteristics such as size, shape, intensity, and any other desired piece of information. We now define a set of association variables α_{ij}. If α_{ij} is equal to one, then we will say that p_i at $t = t_n$ is produced by the same particle as q_j at $t = t_{n+1}$. If α_{ij} is zero, then p_i and q_j represent different physical particles.

For the time being we shall assume that there is one and only one physical particle for each of the particle images. We shall consider groups of particles later in this discussion. For the present it is obvious that, for given p_i, at most only one value of j can give α_{ij} equal to one, otherwise the physical particle must be two places at once! Identical arguments apply for each p_j. If M is equal to N, it may be possible for there to be exactly $M = N$ values of α_{ij} equal to one. However, this will seldom happen in real experiments, where there will normally be fewer than $M = N$ values of α_{ij} equal to one. Moreover, the number of particles images at the two times will not always be equal.

There are many reasons why the number of particles in the image may be different at $t = t_n$ and $t = t_{n+1}$. The simplest is that the particle may have moved outside the region of the flow being tracked, either by moving outside the bounds of the tracking region, or by moving out of the illuminated region (*e.g.* moving out of a sheet of light). To overcome this problem we define α_{0j} and α_{i0} as dummy particles at times $t = t_n$ and $t = t_{n+1}$. Unlike ordinary particles, more than one value of j or i may give a nonzero value of α_{0j} or α_{i0} (respectively). In this case a nonzero value of α_{i0} indicates that particle p_i at $t = t_n$ has been lost from the image by $t = t_{n+1}$, either by moving out of the image or for some other reason. Similarly, $\alpha_{0j} = 1$ represents a particle q_j present at $t = t_{n+1}$ which was not there at $t = t_n$.

In order to determine the optimal set of nonzero α_{ij}, we must first define the functional to be optimised. The only restriction this method puts on the functional is that it is linear in the associations, α_{ij}, and so may be represented by Z, the sum over i and j of $\alpha_{ij}c_{ij}$. Elements of c_{ij} represent the *cost* of associating particle p_i at $t = t_n$ with particle q_j at $t = t_{n+1}$. The optimal solution will be chosen to minimise the *objective* function Z.

Typically the costs c_{ij} will be specified using some function of the particle positions, particle characteristics, temporal history and the physics of the flow. Conceptually the simplest model is to set c_{ij} equal to the separation between particle p_i and particle q_j (c_{0j} and c_{i0} may be set to the distance to the boundaries of the observed region, or the maximum allowable distance a particle may be allowed to travel between t_n and t_{n+1}). The optimal solution will then try to minimise the particle displacements, allowing only associations which do not exceed the cost limits placed by c_{0j} and c_{i0}. The costs c_{ij} could equally as easily be the squares of the displacements, yielding a type of least squares optimal solution.

If we are trying to measure the fluid velocity (rather than Brownian motion, say), then a more appropriate set of cost functions would include some fluid dynamics. This may be achieved at the most basic level by predicting the positions the particles at $t = t_n$ will have at $t = t_{n+1}$ using their velocity (and possibly acceleration) at $t = t_n$. The costs c_{ij} may then be some function of the separation between the predicted position of p_i and the position of q_j. If a particle at $t = t_n$ has only just entered the image, then we are unlikely to have more than a

rough estimate for its velocity and so are unable to predict accurately where it might be at $t = t_{n+1}$. To enable matchings to still occur to such particles, we must reduce the costs of associations with them and allow matchings over larger distances than for particles for which we have a velocity history (we may also, however, add some fixed cost for this *new member*). While the cost reduction – and associated increase in the allowable separations—when there is no velocity history may produce some mismatching, the requirement for a much more exact match would not then be satisfied at $t = t_{n+2}$, and so the mismatch would not continue. During subsequent analysis, if we accept only paths which passed through three or more samples during the tracking phase, then we will eliminate any mismatches due to the less stringent matching requirement for a particle with no velocity history.

Additional factors such as the particle size, intensity, shape or even colour may easily be brought into the costing function. Every added component in a well-chosen functional will increase the probability of a correct matching, but at the expense of increased computation. Fortunately, provided the particle seeding density is not too dense, the extra criteria are unlikely to add significantly to the quality of the results. Experience has shown that the tracking results are relatively insensitive to the exact function used for the costs c_{ij}. Any mismatches which arise due to a short coming in the costing procedure will be short lived (they will fail to match on the next step) and may be trapped during the subsequent analysis phase through acceleration checks.

The basic cost in DigiFlow is given by

$$c_{ij} = \Phi(p_i) + \sum_f \max\left(0, \omega_f(p_i)\zeta_f(p_i, q_j) - \tau_f\right), \tag{23}$$

where $\Phi(p_i)$ is a fee determined by previous history of p_i. The summation is over a list of properties f determined by the location process. These properties include location, threshold (intensity) and size, but in some cases a broader range can be used.

For each particle property there is a unit cost $\omega_f(p_i)$, a threshold τ_f and a cost function $\zeta_f(p_i,q_j)$. The cost function $\zeta_f(p_i,q_j)$ depends on the instantaneous properties of the particles p_i and q_j, whereas the unit cost $\omega_f(p_i)$ may depend on whether or not the history of p_i is known. A typical example of $\zeta_f(p_i,q_j)$ is that for the particle's location,

$$\zeta_x(p_i, p_j) = \left|\mathbf{x}_i + \mathbf{u}_i\delta t - \mathbf{x}_j\right|^2, \tag{24}$$

where $\mathbf{x}_i$ and $\mathbf{u}_i$ are the particle location and velocity at $t = t_n$, while $\mathbf{x}_j$ is the particle location at $t = t_{n+1}$. The corresponding unit cost is

$$\omega_x(p_i) = \begin{cases} \dfrac{1}{L_1^2} & \text{if no previous matches} \\ \dfrac{1}{L_2^2} & \text{if one previous match} \\ \dfrac{1}{L_3^2} & \text{if more than one previous match} \end{cases}, \tag{25}$$

where L_1, L_2 and L_3 are the maximum matching distances for the first, second and subsequent matches the particle p_i may make. The cost of a change in threshold is similar,

$$\zeta_T(p_i, p_j) = \left|T_i - T_j\right|^2, \tag{26}$$

where T_i and T_j is the threshold identifying the particle at t_n and t_{n+1}, respectively. Here the corresponding unit cost $\omega_T(p_i)$ is divided into only two costs depending on whether or not a particle has a history.

The fee $\Phi(p_i)$ is typically taken as zero if the particle has a valid velocity history, and positive if it does not (the 'joining fee'). The purpose of this fee is to promote the preferential matching of particles with a valid velocity history. In contrast, $\mu(p_i)$ is reduced when there is no velocity history to allow matches further a field.

This strategy to assigning costs has proven simple yet flexible and provides a framework that is relatively easy to understand. This model is more sophisticated than that used in DigImage in that particle properties such as intensity and size play a more prominent role in DigiFlow. Tests have shown that this provides a substantially improved matching performance when there are very high particle number densities.

Particle tracking streams

The DigiFlow PTV facility takes an input stream, showing the experiment, and produces an output stream that contains the particle locations, particle properties, and the inter-frame particle associations.

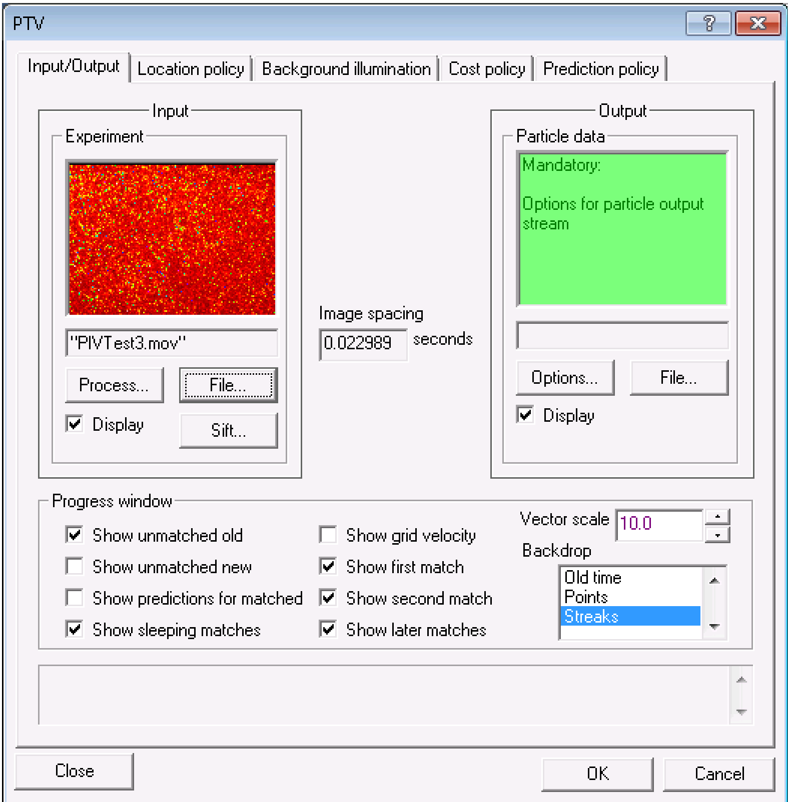


Figure 113: The PTV tab controlling the input and output streams.

The Input stream may be in any valid image format. This is specified in the normal way through clicking File to specify the source if the stream is to be taken from a movie or sequence of images. In such cases the stream may be trimmed for length, a subregion selected, *etc.*, using the Sift button (see §4.3). If the image source is from an upstream process, then this should be specified using the Process button.

The Output stream should be specified as a .dft file. This special file format contains all the particle data and its associations. These .dft files may be viewed using the normal DigiFlow tools; in such cases, the particle data is rendered back as an image. However, these files are really intended for use with the other PTV tools within DigiFlow which can access their contents directly.

The output stream is specified in the standard way through the File button. While Options may be set, there is not generally any benefit to be gained from doing so.

The Progress window group controls what is displayed as the particle tracking proceeds. The information selected here can help assess the performance of the particle tracking, and provide a guide to any adjustments to the Cost policy that may be required. In all cases the velocity of matched particles will be displayed, using white for particles that have been matched over three or more intervals in time, yellow for particles matched over two intervals, and cyan for particles matched only once. An example of the Progress window is shown in figure 114.

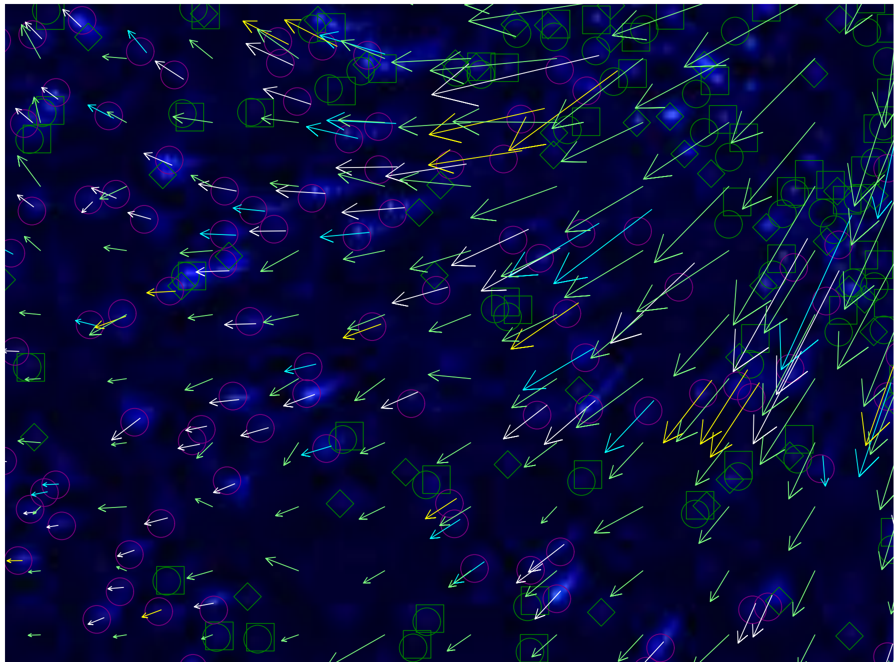


Figure 114: Enlarged example of the Progress window for particle tracking with all the optional output switched on and Streaks selected as the backdrop. Here Vector scale is set to 4.0, so the arrows are four times the length of the actual displacements. The white arrows are particles that have been matched more than two times, the yellow arrows particles that have been matched twice and cyan arrows particles that have been matched only once. Dark green squares are old particles that have not been matched, with dark green circles showing their predicted position. Dark green diamonds are new particles that have not been matched. Dark magenta circles are the predicted positions of particles that were matched, and light green arrows are the gridded velocity field.

If Show unmatched old is checked then particles at the earlier time step that are not matched to the later time step will be highlighted by a square box drawn in dark green around them, and by a circle (also in dark green) at their predicted location. Similarly, if Show unmatched new is checked, then any particles in the later time step that were not matched will be highlighted by a diamond drawn in dark green around them. (If the dark green diamond coincides with a dark green circle then the corresponding particle was not matched due to its change in intensity, area or one of the other image attributes.)

Checking Show predictions for matched will cause dark magenta circles to be drawn around the predicted position for particles that were matched. Any difference between these circles and where the particle is actually located may help diagnose why mismatches occur.

DigiFlow allows particles to go to ‘sleep’ for one frame but for them to still be matched across this period of sleep. The Show sleeping matches check box causes such matches to be shown in the Progress window..

At each time step DigiFlow calculates an approximate gridded version of the velocity field. The primary use of this is as an estimate for the velocity of particles with no prior history. By checking Show grid velocity this grid will be displayed in the progress window in light green.

The initial size of the arrows for plotting the velocity is set by Velocity scale. A unit value causes the arrows to be drawn at the same length as the displacements they represent.

The vector and particle information shown in the Progress window is displayed on top of an image of the experiment. The Backdrop list selects exactly how this image is constructed. Selecting Streaks will use a decaying series of images superimposed to give an impression of the particle motion, while Old time and New time will show one or the other of the two images being processed.

The Image spacing is shown here for information only. If the particle tracking is to be undertaken on a sequence of images that do not contain time information, then the default spacing will be 1.0 seconds. This spacing may be changed, however, in the Sift dialog (see §3.6).

Particle location policy

The location of particles is of central importance to the performance of PTV. In DigiFlow, this process is controlled by the Location policy tab.

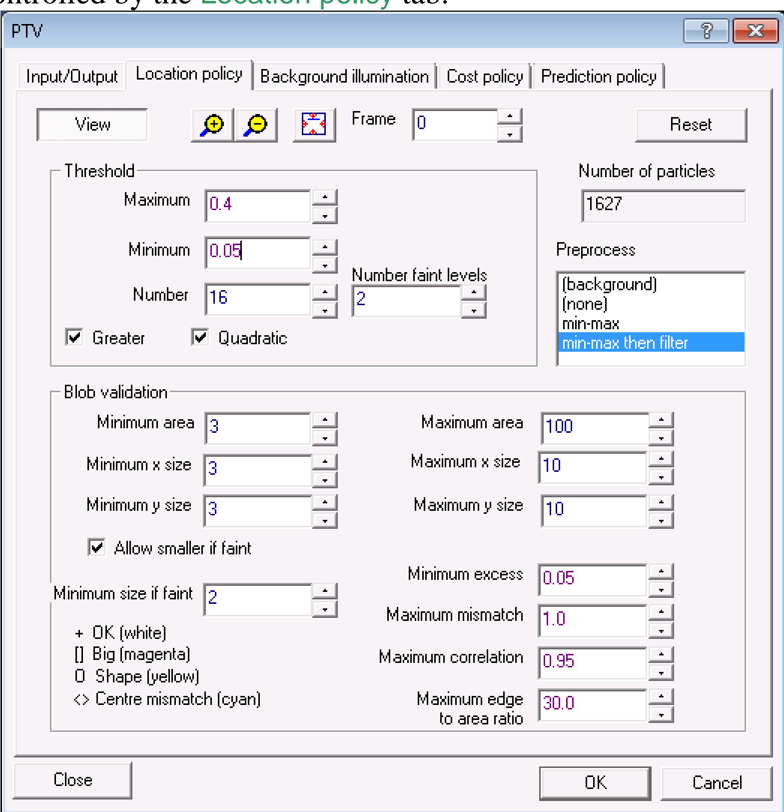


Figure 115: Parameters controlling the PTV particle location policy.

Best results can be achieved from high quality images that have bright, clear particles approximately two or three pixels in linear dimensions, on a uniformly black background. Such experiments, however, can be difficult to achieve in practice. The particle location strategy used in DigiFlow builds on the experience with DigImage to provide a robust, accurate and efficient method of getting the best possible results from the available images.

DigiFlow provides a preview of the located particles to aid the process of setting the various parameters. This preview is activated by clicking the View button. Placing the cursor over the preview will provide the normal feedback of the intensity at the location of the cursor, while the [zoom in], [zoom out] and [resize] buttons provide the ability to zoom in, zoom out and resize the preview window. The preview window contains white plus (+) marks indicating the particle locations superimposed on top of an image of the image (see figure 116). Additionally, a

subset of the rejected 'blobs' are indicated by magenta boxes (blobs too big), yellow circles (inappropriate particle shape, controlled by Maximum correlation and Maximum edge to area ratio) and cyan diamonds (mismatch between area and volume centroids, controlled by Maximum mismatch). The preview window is terminated by a second click of the View button. The location of the preview image within the time series is determined by the Frame control.

Even if the preview is not generated the Number of particles box will show the current estimate for the number of particles within the frame. This count is updated automatically whenever one of the location control parameters is changed. Note however, that if a control is changed while DigiFlow is still processing the last lot of changes, then the count (and preview) may not reflect the latest changes.

The best results can generally be obtained by directly probing a high quality raw image stream. However, for inexperienced users or less than ideal image streams, optimising the settings for this can be difficult. For this reason, DigiFlow provides the possibility of preprocessing the images to provide a more uniform and consistent structure to the images. This preprocessing necessarily destroys some of the information contained within the original images, but the algorithms are designed to keep this to a minimum.

The preprocessing is controlled through the Preprocess list box. As noted above, the greatest accuracy can be achieved by selecting (none) to suppress preprocessing, although for a given image stream this may not be appropriate. For inexperienced users the min-max filter option is recommended. This nonlinear filter attempts to remove background variations on scales larger than the particles, thus effectively resulting in the particles appearing on a uniform black background for subsequent location. A different form of preprocessing is available by selecting (background). This activates the controls on the Background Illumination tab (see below) which allows an image of the experimental setup *without particles* to be removed from the experimental images.

The starting point when changing the locations parameters is normally setting the range of intensities through which the threshold will be scanned. This is achieved using the Threshold group. The location process begins by looking for particles satisfying the threshold Maximum, gradually decreasing this in Number discrete steps down to Minimum. The Greater check box will cause DigiFlow to search for bright particles on a dark background, while clearing the check box sill invert the incoming image stream, thus allowing it to be treated in the same way. The Quadratic check box controls the distribution of thresholds between the two limits. For many experiments, having Quadratic checked works best.

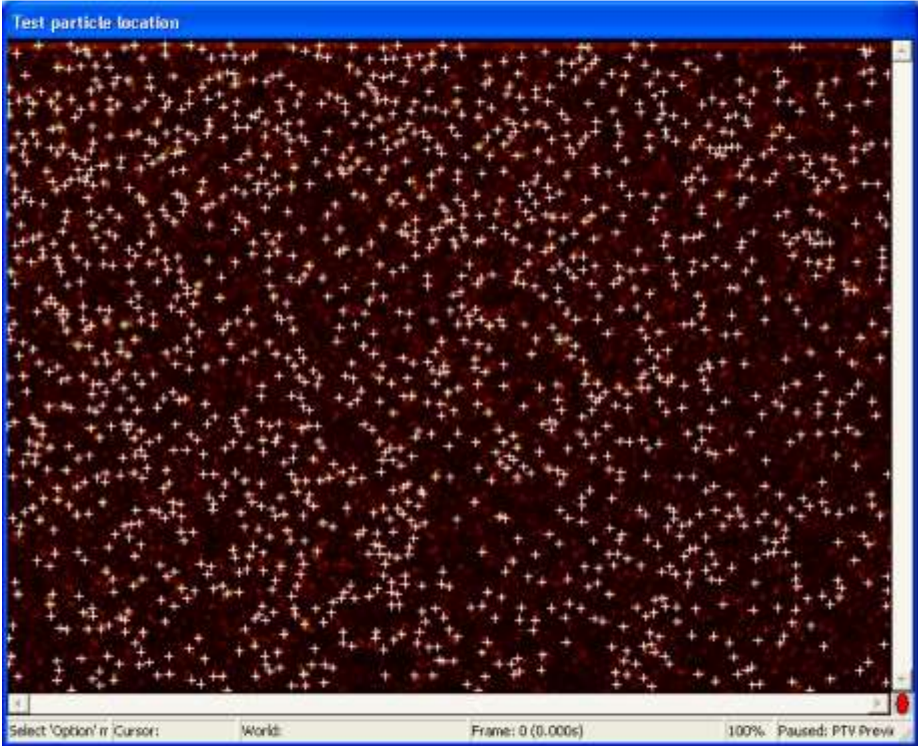


Figure 116: Preview window showing the particles that have been located.

The Blob validation group provides the information necessary to decide whether or not a given blob that has been located should be treated as a particle. The left-hand column is pretty much self-explanatory. Blobs smaller than Minimum area will be ignored at a given threshold, but they may well be picked up as particles at a later (lower) threshold. Blobs exceeding Maximum area will be discarded. The reason for having limits on both linear dimensions and particle area is to help ensure the particles are roughly circular and ensure that they may be located with subpixel accuracy. The upper limits are provided to prevent spurious features within the image from being picked up accidentally. The Maximum x size and Maximum y size not only set the upper size limits, but also provides the length scale for the filter that is used when the min-max filter is selected for Preprocessing.

The mean intensity of a blob relative to the threshold at which it is identified must exceed Minimum excess, which ensures the image is sufficiently well defined. The location assigned by DigiFlow to a particle satisfying all other criteria is the volume centroid, where the third dimension is the intensity relative to the threshold. However, DigiFlow also calculates the area centroid; the maximum difference between the locations of these two centroids is determined by Maximum mismatch.

Other aspects of the geometry are tested using Maximum correlation, which is the correlation coefficient of the pixels within the blob. In general a value close to 1 or -1 indicates that the blob is linear rather than circular in nature. Similarly, Maximum edge to area ratio compares the square of the number of pixels marking the boundary of the blob with the number within the blob. A large value for this ratio indicates either linear blobs or blobs with very convoluted boundaries. As an indication, a large, circular blob would have this ratio equal to $(2\pi r)^2/(\pi r^2) = 4\pi$, a square would have a ratio of 16, while a line of length L and a single pixel wide would have the ratio equal to $4L$. The default value is somewhat higher than this to allow a broader range of particles to be tracked.

The Reset button will restore all of these parameters to their default values.

In addition to providing a preview of the particles found, the View button also provides a plot of the size distribution of the particles identified.

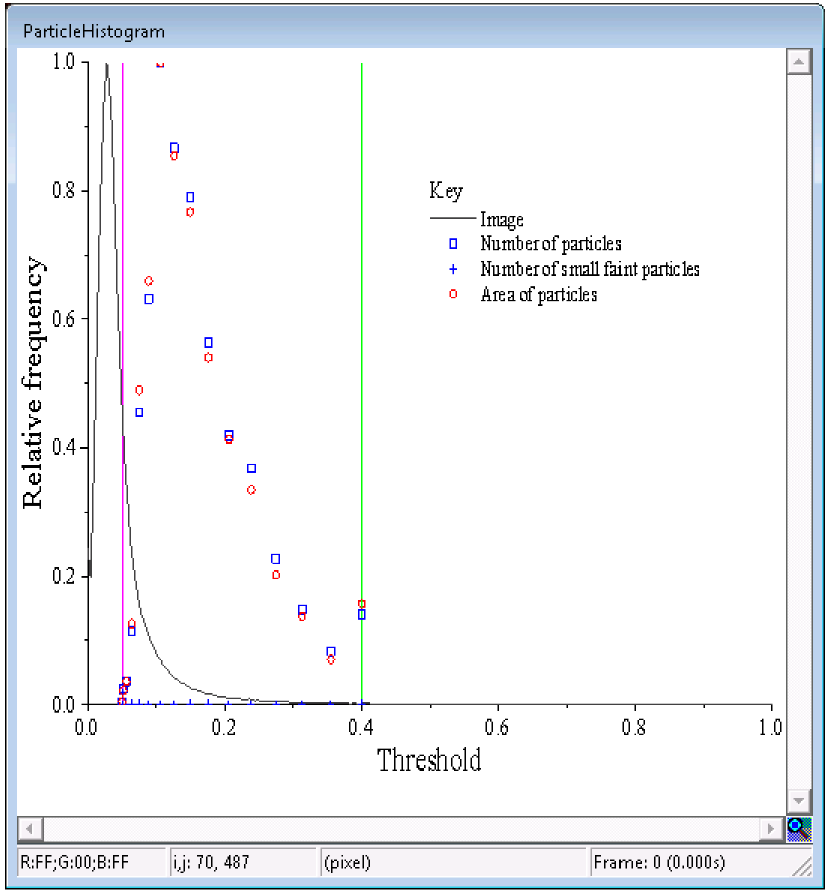


Figure 117: Histogram showing particle number and area for given categories of particles identified by the Locaion policy.

Background illumination

The controls on the Background illumination tab are enabled by selecting (background) in the Preprocess list on the Location policy tab. The Background illumination tab provides a convenient method of correcting your experimental images for a non-uniform, non-zero background illumination in the experimental images.

Typically all that is required is a single image. This can be an image of the experimental setup with no particles present, or may be constructed from the experimental setup itself. If the particles are brighter than the background then a typical strategy for the latter is to work out the minum intensity for each pixel using the min feature in Analyse: Time Average (see §5.6.1.1). The rationale behind this is that a given pixel will be at its darkest when no particle is present. (If the particles are darker than the background, then use max instead of min.)

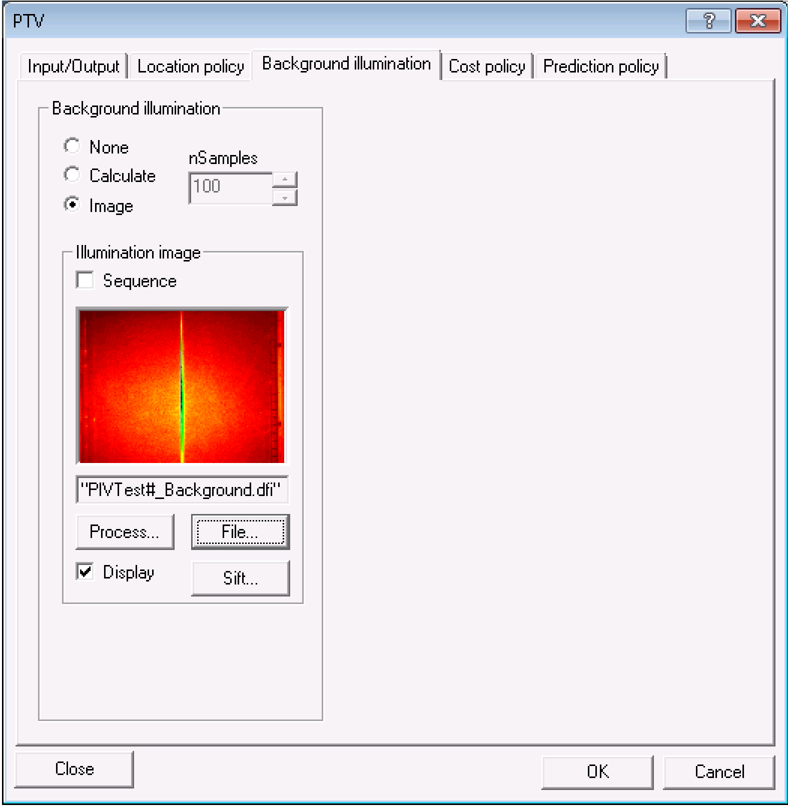


Figure 118: The Background intensity tab controls the removal of background variations in the intensity.

When Image is selected, the background image is specified as either a single image or as a sequence of images; which is determined by the Sequence check box. In most situations a sequence is unnecessary, but if there are moving parts, or significant predictable changes in illumination, then a sequence may be desirable.

If Calculate is specified, then a background illumination image is constructed from the experimental input in the manner described above. Rather than using every image in the input, it is frequently only necessary to use a subset of the images. The nSamples control specifies the maximum number of samples that should be used. These will be evenly distributed over the duration of the experimental image sequence. Note, this control shoul not be used when the experimental image sequence is obtained from a process rather than a file. The background image generated in this manner is not saved; moreover, it is not available until the particle tracking process starts, and so it can be more difficult to set the particle location parameters. For these reasons it will normally be more convenient to manually construct the background image using Analyse: Time Average (see §5.6.1.1), should you need one.

Costing policy

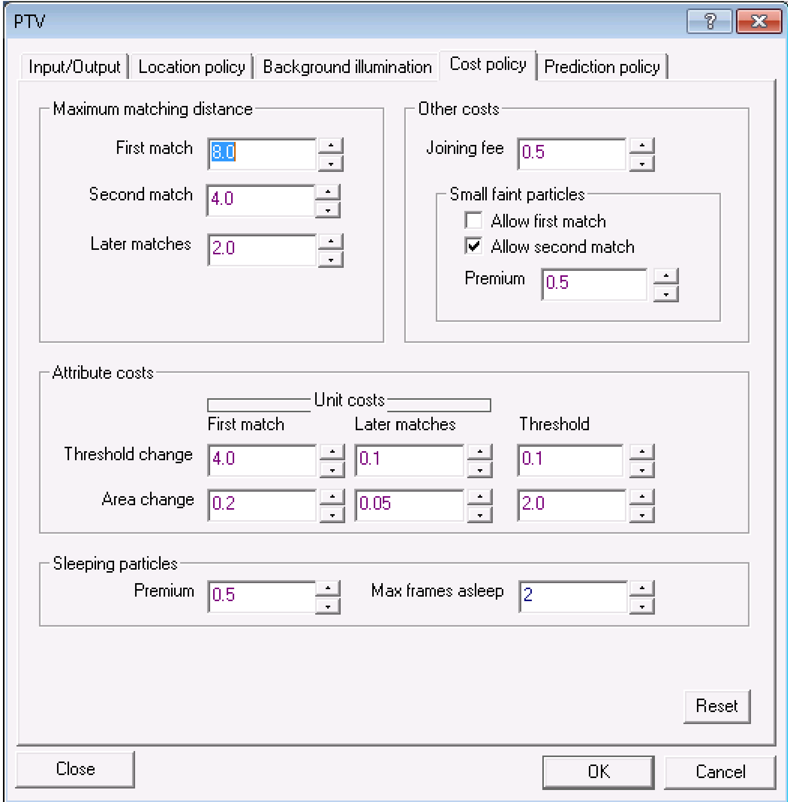


Figure 119: Parameters controlling the costing policy for particle tracking.

As noted earlier, the matching process is governed by the cost assigned to each of the possible associations between the sets of particles identified at different times. The Costing policy tab defines the various factors that go into determining the cost. Each of the parameters is described in turn below, followed by a brief guide on strategies for adjusting them, should this prove necessary.

The most important parameters in most cases are those in the Maximum matching distance group. The three distances given here determine the maximum distance (in pixel separation) between the predicted position of a particle and where one is actually found. As any prediction of a particle without a history the First match value should normally be larger than the other two. For flows with low accelerations the Second match and Later matches should be similar or even the same. These maximum separations will be realised only if the particles do not incur other costs in the Attribute costs group (see below). (For users familiar with DigImage, the Later matches is similar to `[;USPM Maximum matching distance]` and First match is similar to `[;USM Max new paths error]` when expressed in pixels.)

The Other costs group contains other costs that are used to modify the matching process. The Joining fee (range 0 to 1) is applied only to particles that do not have a history. Increasing the Joining fee does not affect the Maximum matching distance for the First match, but does decrease the probability that an association with the particle will be permitted.

The Attribute costs group is used to increase the cost of an association if the attributes of the particle images concerned differ. Two sets of values are specified: one for the First match, and a second for Later matches. In each case, no cost is incurred if the attributes differ by less than Threshold.

The Threshold change cost and Threshold, and the Area change cost and Threshold work in a similar way to the distance cost, although the measure of the area change is

$2|A_i - A_j|/(E_i + E_j)$, where A_i and A_j are the areas and E_i and E_j the number of edge points for the old and new particles, respectively.

In most circumstances the default values (which can be restored using the Reset button) will work well. However, in some flows it might be necessary to adjust things either to reduce the number of spurious matches, or to allow DigiFlow to lock on to particles that are moving very rapidly.

Prediction policy

The prediction policy (see figure 120) determines how velocity information is incorporated into the distance function (24). Velocity weighting determines how much of the velocity from the last match for a particle is used to predict its new position, and the Acceleration weighting does a similar thing with the particle Lagrangian acceleration (when there is sufficient history to evaluate this). This particle-based velocity is not the only potential source of velocity information. DigiFlow also calculates a grid velocity which is based on the average particle velocities within grid cells covering the domain. The Grid weighting determines how much of this is incorporated into the prediction. In particular, if V is the velocity weighting and G is the grid weighting, then for a particle with a velocity history the velocity the velocity used in (24) is

$$\boldsymbol{u}_i = V\boldsymbol{u}_i^n + (1-V)G\boldsymbol{u}_g, \quad (27)$$

where $\boldsymbol{u}_i^n$ is the particle velocity from the previous time step and $\boldsymbol{u}_g$ is the grid velocity. When there is no velocity history then

$$\boldsymbol{u}_i = G\boldsymbol{u}_g. \quad (28)$$

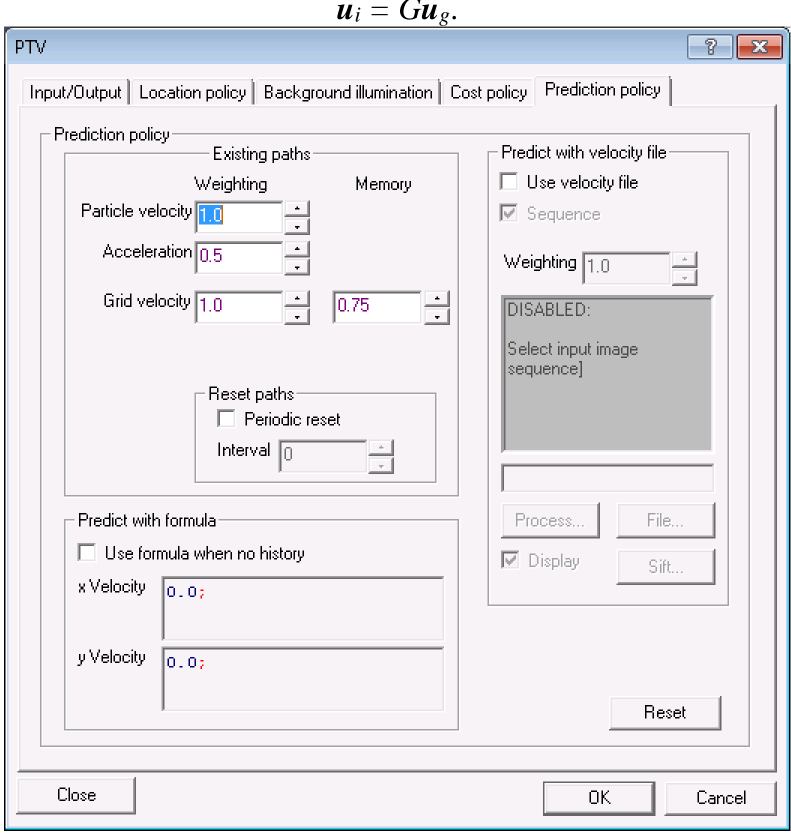


Figure 120: The controls for the prediction policy.

The above strategy for determining the velocity may not be appropriate if the mean velocity is significantly different from zero. In this case we may enable use formula and specify (in pixel units) a background velocity field using u Formula and v Formula. The resultant prediction is then given by

$$\boldsymbol{u}_i = V\boldsymbol{u}_i^n + (1-V)(G\boldsymbol{u}_g + (1-G)\boldsymbol{u}_f), \quad (29)$$

when there is a history, and

$$\boldsymbol{u}_i = G\boldsymbol{u}_g + (1-G)\boldsymbol{u}_f \quad (30)$$

when there is not. The u Formula and v Formula are specified in terms of its location x, y (in pixels) and the time t. A typical example of the use of this function would be for flow in a flume, where the two formulae would simply represent the mean flow.

When there is insufficient particle information to evaluate the grid velocity at a given point, DigiFlow will a memory of the last calculated grid at that point. This memory fades in a manner determined by Grid memory. If Use formula when no history is not set, then the grid velocity will decay towards zero by taking the product of Grid memory and the current grid at each time step.

In some cases it may be desirable to feed in the predicted velocity from a sequence of .dfi files. These may, for example, be the result of a previous attempt at PIV or PTV (using the PTV Grid velocity feature of §5.6.6.5). Such a two-stage process can help DigiFlow latch on to particles in problematic regions of high gradients or in high-speed flows. Note that once DigiFlow has latched on to the particles they will be treated in much the same way as normal. This feature is enabled via the Use velocity file checkbox. The supplied velocity information may be a single .dfi describing a steady mean flow (in which case clear the Sequence check box), or it may be a time-varying sequence. Note that in both cases it is essential that the velocity information is provided for the same region as the tracking and that the same time spacing is used. It is also normally best if the velocity information is provided in pixel coordinates. The Weighting control within the Predict with velocity file determines the relative importance of the supplied velocity file and the normal gridded velocity, described above.

Image sequences of high-speed flows sometimes consist of repeated short bursts of images where the image spacing within the burst is shorter than that between bursts. PIV often uses this technique with two closely spaced images in each burst. The Reset paths group is implemented to aid the tracking of sequences containing bursts of images with a different time interval between the bursts than between the images within the burst. For most cases, Periodic reset should be unchecked, meaning that the spacing between all images are the same and that matches should be made over each image pair in turn. Checking Periodic reset will force all paths to be broken (*i.e.* no matches allowed) at intervals specified by Interval (in frames). Not only will the paths be discarded, but also the gridded velocity field will be discarded. Thus, effectively, the particle tracking will start again from scratch. Note that utilising a Periodic reset on a flow that has a continuous record will degrade the results from the particle tracking. Moreover, the smaller the Interval, the poorer any velocity calculations will be. (It will also be necessary to ensure that the time used to calculate the velocity does not exceed the period of data between each reset.) For the case of PIV sequences with two images in each burst, then set Interval to two.

The default values may be restored using the Reset button.

Tracking

During the tracking process, DigiFlow will display three windows. The Experiment will display the raw experimental image being processed, while the Particles image will display each of the identified particles as a dot. The colour of each dot is related to the threshold at which the blob in the experimental image was considered to be a particle.

Perhaps the most useful window is the Progress window. This window displays a variety of information about both the velocity field and the performance of the tracking process. Details of the different arrows and symbols used was given earlier in this section, with an

example shown in figure 114. Statistics of the number of particles matched are also given in the title bar of the window.

Occasionally an obviously incorrect vector will be produced. If such a vector is yellow, then it is of little concern: the matching criteria for particles without a velocity history are necessarily less stringent, a feature that is likely to lead to the occasional mismatch. Such vectors are unlikely to persist, however, as the implied velocity history is much less likely to lead to a match on the next step.

There will be times, however, when spurious vectors persist. The table below lists potential problems and remedies.

Description	**Remedy**
Very few particles have vectors	Check that location policy is reliably picking up particles on successive frames.
	If the intensity of the particles is fluctuating a lot, try reducing the Cost of a Threshold change, or increasing the Threshold before a cost is incurred. This problem is most likely to occur when the particles are extremely small.
	If the particles are moving relatively far and fast between frames, try increasing the Maximum matching distance group.
Spurious white vectors persist.	DigiFlow may be identifying too many particles, some of which are really just noise. Check the Location policy.
	Check that matches are not being made too readily. Try reducing Maximum matching distance group.
	Try increasing the Cost or reducing the Threshold for Threshold changes.

Calculating particle velocity

Once the tracking has been completed, it is often desirable to calculate the particle velocities. The velocities may be calculated from a particle path in a number of ways. At the simplest level, the location of particle i on two consecutive frames, $\boldsymbol{x}_i^{(n-1)}$ and $\boldsymbol{x}_i^{(n)}$, can be used to estimate the velocity as

$$\boldsymbol{u}_i^{(n-½)} = (\boldsymbol{x}_i^{(n)} - \boldsymbol{x}_i^{(n-1)})/\Delta t,$$

where Δt is the spacing between two frames. Although this approach provides the highest possible frequency response, it is also the most subject to noise. If the error in the positions of the particle is $\sigma_{\boldsymbol{x}}$, then the error in the velocity is $\sigma_{\boldsymbol{u}} = 2\sigma_{\boldsymbol{x}}/\Delta t$. The simplest way of decreasing the error is to perform the calculation over a larger interval. If

$$\boldsymbol{u}_i^{(n-s/2)} = (\boldsymbol{x}_i^{(n)} - \boldsymbol{x}_i^{(n-s)})/(s\Delta t),$$

then the error is reduced to $\sigma_{\boldsymbol{u}} = 2\sigma_{\boldsymbol{x}}/s\Delta t$, provided the velocity is constant within the interval.

For most purposes it is better to decrease the interval between frames (decrease Δt) and then use a least squares fit to a sequence of s particle positions. The simplest alternative is to fit a line. Since the frame interval is constant, the estimate of the velocity is therefore

$$\mathbf{u}_i = \frac{1}{\Delta t}\left[\frac{\left(\sum_{j=0}^{s} j\right)\sum_{j=0}^{s}\mathbf{x}_i^{(n+j)} - (s+1)\sum_{j=0}^{s} j\mathbf{x}_i^{(n+j)}}{(s+1)\sum_{j=0}^{s} j^2 - \left(\sum_{j=0}^{s} j\right)^2}\right]. \quad (31)$$

This velocity is then assigned to the least squares estimate of the particle's position in the middle of the time interval. Key to the use of the least squares approach is its effect on the error in the velocity estimate. As shown by Dalziel (1992), the error estimate is reduced to

$$\sigma_{\mathbf{u}} = \left[\frac{12}{(s+2)(s+1)s}\right]^{1/2}\frac{\sigma_{\mathbf{x}}}{\Delta t}.$$

Increasing s leads to a reduction in the error estimate for the velocity, but only provided the velocity remains approximately constant over the interval $s\Delta t$. Increasing s and decreasing Δt can achieve this, although there will normally be limits imposed by the camera frame rate that limits s. However, the interval $s\Delta t$ may be increased further if the model for the particle path remains reasonable; fitting a quadratic rather than linear function can achieve this.

Post processing

In addition to the post-processing features described in the following sections, DigiFlow provides a dfc macro interface to access the .dft particle tracking data. The following segment of code illustrates some of the core functions. This code is intended for a scenario when there are only a small number of particles. It produces a scatter plot of the vertical velocity against the vertical position of the particles.

```
# Determine the tracking file
file := "PTV####.dft";
file := ask_string("Name of .dft files (including hashes)?",file);
# Get basic details of the file
det := read_image_details(file);

# Rather than utilising a coordinate system, use a known one-to-one
# relationship between the pixel size and the size of the imaged region
# Here a 1:1 magnification is being used
pMax := 5.0; # Maximum expected velocity in pixels/s
pixSize:= 7.4; # micrometres
wMax := pMax*pixSize;
zMax := pixSize*(det.ny-1);

# Create the drawing for the scatter plot
hD := draw_start();
draw_set_axes(hD,0,wMax,0,zMax);
draw_x_axis(hD,"$w  (\mu$m$s^{-1})$");
draw_y_axis(hD,"$z  (\mu$m)");
draw_colour_scheme(hD,"single cycle - half brightness");
draw_mark_size(hD,1);
draw_font(hD,0.5); # Make font smaller for the size labels


# Open the ptv data to reconstruct the paths
hP := ptv_open(file,coordSystem:="(pixel)"); # Returns handle of window
showing PTV input
for fNow:=det.fFirst to det.fLast {
  # Calculate the velocities using least squares over five frames
  ptv := ptv_velocity(fNow,5);
  # Also read the information for the particles at this time
  part := ptv_read_particles(file,fNow);

  # ptv and part will be null if there is no data
  if (is_array(ptv)) {
    # Scan through the list of particles
    for k:=0 to y_size(ptv)-1 {
      id := int(ptv[4,k]); # The unique track number for this particle
```

```
            # Select colour and mark style based on track number
            iCol := 0.9*(int(id/9) mod 16);
            iStyle := id mod 9;
            draw_line_colour(hD,iCol);
            draw_text_colour(hD,iCol);
            draw_mark_type(hD,2+iStyle);
            x := ptv[3,k]*pixSize;
            y := ptv[1,k]*pixSize;
            draw_mark(hD,x,y);
            # Find info for this particle
            this := ptv_particle_details(part,id);
            if (fNow = this.startFrame+3) {
              # Write details on third occurrence
              area := this.area*pixSize^2;
              dia := 2*sqrt(area/pi);
              draw_text(hD,x,y,"$    d ="+nice_number_string(dia)+"\mu$m");
            };
          };
        };
      };
      # Tidy up the access to the PTV data
      ptv_close();
      close_view(hP); # Remove window showing PTV input

      # Show the plot
      hV := view(hD); # Returns window handle
      view_title(hV,"Vertical velocity scatter plot");

      oFile := ask_string("Name to save plot to (blank to
                suppress)?","wScatter.dfd");
      if (is_null(oFile)) {
      } elseif (length(oFile) > 0) {
        write_image(oFile,hD);
      };

      # Tidy up
      draw_destroy(hD); # Free drawing memory
```

A key feature of this code is the use of the unique particle id assigned to each particle track to relate the velocity information provided by `ptv_velocity(..)` (the first index of the returned array set to 4) to extract further details from the .dft tracking file. In particular, `ptv_read_particles(..)` is used to determine all the particle information at a given time, and then `ptv_particle_details(..)` is used to extract the details for a specific track. Here we calculate the effective particle diameter from the area returned for the particle.

Further details of the individual functions can be found by accessing the dfc help system. See §5.2.10 and §5.9.2 for details.

5.6.6.2 PTV Basic statistics

Toolbutton:

Shortcut:

Related commands: `process PTVBasicStatistics(..)`, `ptv_open(..)`, `ptv_close(..)`, `ptv_tracks(..)`, `ptv_velocity(..)`

Basic velocity statistics for the particles are available through this feature. The statistics are weighted by the number of particles, rather than the region of space in which particles were found.

The controlling dialog takes the normal form with the .dft tracking data being specified in the PTV data input stream. The Basic Statistics output takes the form of a single .dfd (or .emf or .wmf) output plot.

The particle tracking process is undertaken in pixel space. However the results will generally be required in world coordinates. In DigiFlow the transformation between the two is made during the analysis stage by selecting the appropriate Coordinate system.

The method of calculating the velocity, and the number of time intervals across which the calculation is made, is determined by the Velocity group. Typically a value of `4` or `more` should be used for the Length entry in conjunction with the Linear or Quadratic methods. It is recommended that the Extremes option only be used for testing purposes as this provides the least accurate approach.

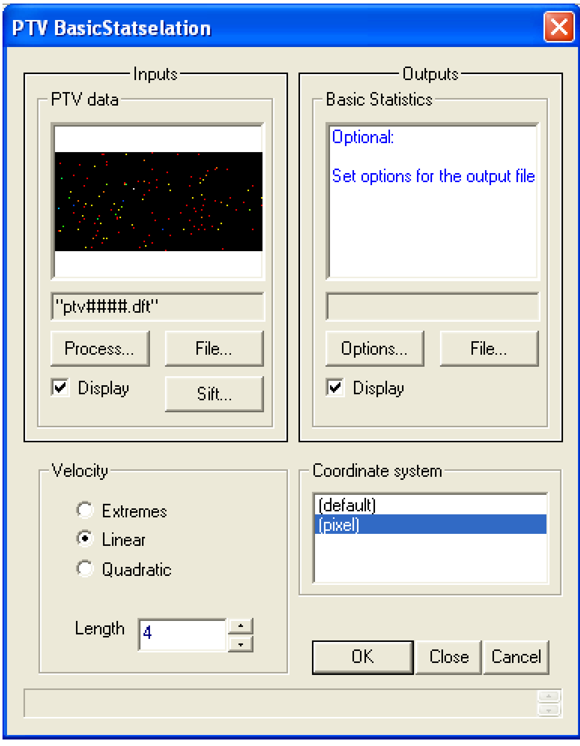


Figure 121: Dialog controlling the calculation of basic PTV statistics.

5.6.6.3 PTV autocorrelation

Toolbutton:

Shortcut:

Related commands: `process PTVAutocorrelation(..)`, `ptv_open(..)`, `ptv_close(..)`, `ptv_tracks(..)`, `ptv_velocity(..)`

Since particle tracking is an inherently Lagrangian process, it makes sense to analyse the particle tracks in a Lagrangian framework. The Lagrangian autocorrelation functions are one such way. Particle velocities are calculated for each point along a path using the methods outlined in §5.6.6.5 and then related to the velocity at another time along the same particle path to generate the autocorrelation coefficient

$$R_{ij}(\delta t) = \frac{N\sum u_i(t)u_j(t+\delta t) - \sum u_i(t)\sum u_j(t+\delta t)}{\left[\left(N\sum (u_i(t))^2 - \left(\sum u_i(t)\right)^2\right)\left(N\sum (u_j(t+\delta t))^2 - \left(\sum u_j(t+\delta t)\right)^2\right)\right]^{1/2}},$$

where the summation is over the N particles paths at least δt long occurring at any time t in a specified interval. Here the indices i and j refer to the velocity components u or v.

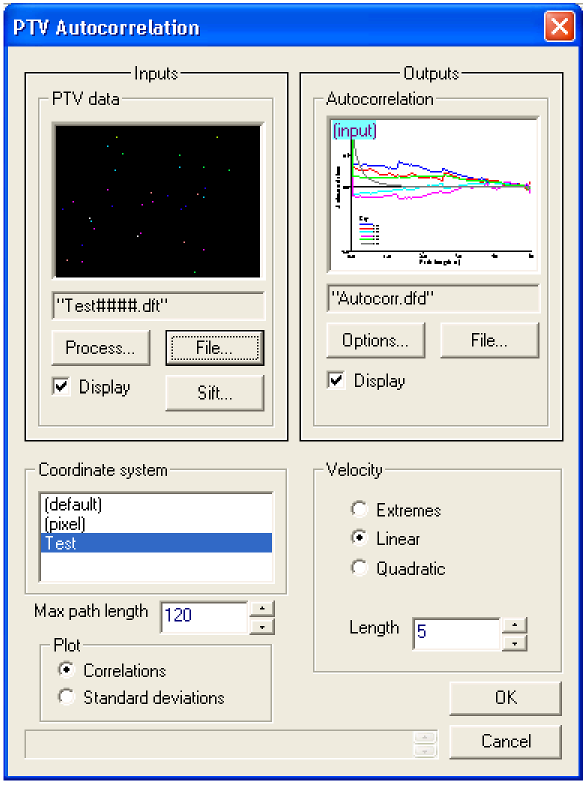


Figure 122: Control of the autocorrelation facility.

The PTV data input selector takes a series of .dft particle tracking files and extracts the data from them. The start and end points, and the spacing of output, can be set by the Sift button (see §4.3).

Output of the Autocorrelation is in the form of a .dfd drawing file, an .emf file, or a .wmf file. If you want access to the actual track data, then the .dfd option is preferred.

The particle tracking process is undertaken in pixel space. However the results will generally be required in world coordinates. In DigiFlow the transformation between the two is made during the analysis stage by selecting the appropriate Coordinate system.

The method of calculating the velocity, and the number of time intervals across which the calculation is made, is determined by the Velocity group. Typically a value of `4 or more` should be used for the Length entry in conjunction with the Linear or Quadratic methods. It is recommended that the Extremes option only be used for testing purposes as this provides the least accurate approach.

The autocorrelation function will be calculated for all separations δt up to the maximum specified by Max path length, although the calculation will proceed only as far as particle paths of that length are still found.

Normally the results of this calculation will be the autocorrelation functions, selected by Correlations in the Plot group. However, it can be valuable to determine the standard deviations (velocity fluctuations) of the data as the conditional sampling associated with the very long particle paths can lead to a bias in the statistics. Select Standard deviations to see this data.

5.6.6.4 PTV vectors

Toolbutton:

Shortcut:

Related commands: process Analyse_PTVVectors(..), ptv_open(..), ptv_close(..), ptv_tracks(..), ptv_velocity(..)

Particle tracking data begins as Lagrangian particle paths. Typically these are randomly distributed in space and variable length in time. The .dft file potentially contains particles that exist for only a single frame, and others that are part of paths spanning many frames. The PTV vectors facility provides the ability to review the contents of the .dft file, filtering out the paths that are too short.

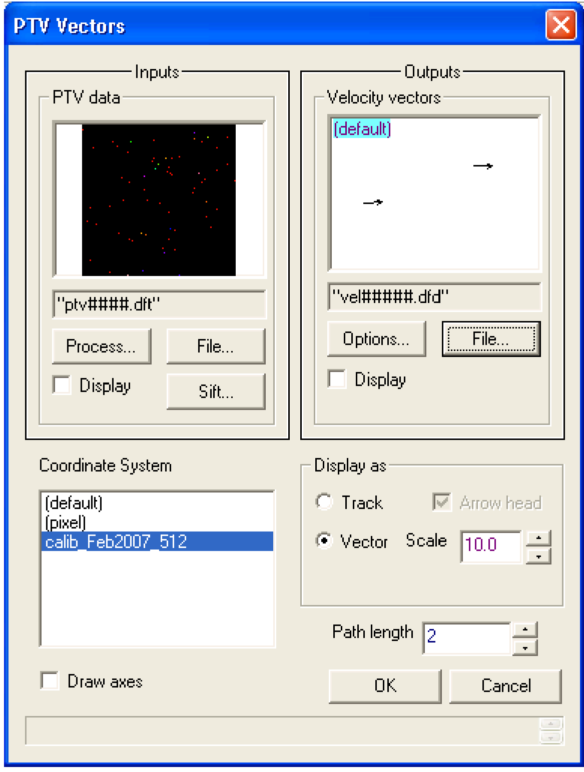


Figure 123: Dialog controlling the production of particle tracking vectors.

The PTV data input selector takes a series of .dft particle tracking files and extracts the data from them. The start and end points, and the spacing of output, can be set by the Sift button (see §4.3).

Output of the Velocity vectors or particle tracks is in the form of a .dfd drawing file, or a .wmf or .emf file. If you want access to the actual track data, then the .dfd option is preferred as this will contain all the paths individually listed. The Draw axes check box determines whether the drawing includes axes or only the vectors/tracks. Note that specification of an output stream is optional. If not specified, the output will be displayed on the screen while it is computed, but will be discarded at the end of the processing.

The particle tracking process is undertaken in pixel space. However the results will generally be required in world coordinates. In DigiFlow the transformation between the two is made during the analysis stage by selecting the appropriate Coordinate system.

The output can contain either velocity vectors (select Vector) or the particle tracks (select Track) at each time, where the vectors/tracks are determined only for particles that extend for at least Path length intervals in time (half before and half after the current time). When Track is selected, the Arrow head check box determines whether or not an arrow head is draw; arrow heads are always drawn when Vector is selected.

5.6.6.5 PTV grid velocity

Toolbutton:

Shortcut:

Related commands: `process PTVGridVelocity(..)`, `ptv_open(..)`, `ptv_close(..)`, `ptv_tracks(..)`, `ptv_velocity(..)`

Mapping particle velocities to grid

For some purposes, it is desirable to transfer the randomly distributed particle paths, and their associated Lagrangian velocities, onto a regular grid. The basic approach for doing this is using a weighting kernel to distribute the particle velocities onto the grid. Suppose $\boldsymbol{u}_i$, $i=0,1,\ldots n-1$ are the particle velocities known at locations $\boldsymbol{x}_i$, then we may estimate the velocity $\boldsymbol{U}$ at some location $\boldsymbol{X}$ by

$$\mathbf{U} = \frac{\sum_{i=0}^{n-1} \alpha\left(\frac{|\mathbf{x}_i - \mathbf{X}|}{L}\right)\mathbf{u}_i}{\sum_{i=0}^{n-1} \alpha\left(\frac{|\mathbf{x}_i - \mathbf{X}|}{L}\right)}, \tag{32}$$

where $\alpha(|\boldsymbol{x}_i - \boldsymbol{X}|)$ is the weighting function. We select $\alpha(r)$ to provide finite support over some length scale L. This approach was pioneered in DigImage.

By ensuring $\alpha(r)$ and its derivatives are continuous, then we may use the same form to provide velocity gradients by analytically differentiating the kernel. For example, $\partial \boldsymbol{U}/\partial x$ is given by

$$\frac{\partial \mathbf{U}}{\partial x} = \frac{\sum_{i=0}^{n-1} \frac{\partial \alpha}{\partial x}\mathbf{u}_i}{\sum_{i=0}^{n-1} \alpha} - \mathbf{U}\frac{\sum_{i=0}^{n-1} \frac{\partial \alpha}{\partial x}}{\sum_{i=0}^{n-1} \alpha}, \tag{33}$$

an approach that has had much use in the family of numerical techniques known as Smoothed Particle Hydrodynamics (SPH, *e.g.* Monaghan 1992) and offers substantially better performance than finite difference on the gridded velocities. Following the work with SPH, we use the axisymmetric cubic spline

$$\alpha\left(\frac{r}{L}\right) = \begin{cases} 1 - \frac{3}{2}\left(\frac{r}{L}\right)^2 + \frac{3}{4}\left(\frac{r}{L}\right)^3 & \frac{r}{L} \le 1 \\ \frac{1}{4}\left(2 - \frac{r}{L}\right)^3 & 1 < \frac{r}{L} \le 2 \\ 0 & \frac{r}{L} > 2 \end{cases}. \tag{34}$$

This is plotted in figure 124.

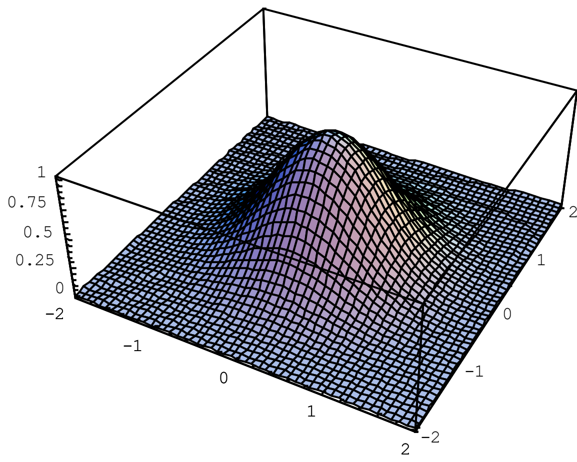


Figure 124: The axisymmetric cubic spline used to distributed particle data to the grid.

Grid velocity dialog

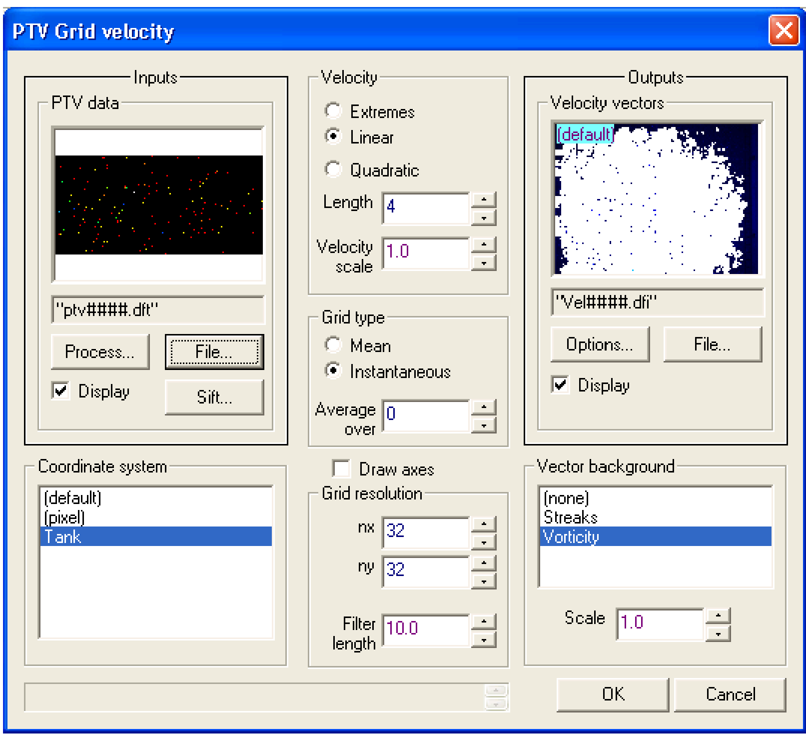


Figure 125: Dialog controlling the process of gridding the particle tracking data.

The PTV data input selector takes a series of .dft particle tracking files and extracts the data from them. The start and end points, and the spacing of output, can be set by the Sift button (see §4.3).

Output of the Velocity vectors, and optional background image, is in the form of a .dfd drawing file, an .emf file, a .wmf file, or a .dfi image file. If you want access to the actual track data, then the .dfd option is preferred as this will contain all the paths individually listed. The Draw axes check box determines whether the drawing includes axes. Using a .dfi file will allow the resulting velocity fields to be manipulated by the various image processing tools available within DigiFlow. Note that specification of an output stream is optional. If not specified, the output will be displayed on the screen while it is computed, but will be discarded at the end of the processing.

The particle tracking process is undertaken in pixel space. However the results will generally be required in world coordinates. In DigiFlow the transformation between the two is made during the analysis stage by selecting the appropriate Coordinate system.

The method of calculating the velocity, and the number of time intervals across which the calculation is made, is determined by the Velocity group. Typically a value of `4` or `more` should be used for the Length entry in conjunction with the Linear or Quadratic methods. It is recommended that the Extremes option only be used for testing purposes as this provides the least accurate approach. The length scale of arrows used for the velocity is determined by Velocity Scale. A unit value will cause the arrows to represent the actual distance moved between two consecutive frames.

The grid velocity can represent either the instantaneous velocity field, or a temporal mean of the selected interval. The choice of which is determined by the Grid type group. With an Instantaneous grid, it is possible to employ a moving average, either to filter out the higher frequency components, or (for a flow that is steady in the Eulerian frame) to increase the available data for the gridding process.

The resolution of the grid and length scale of the kernel function are fixed by the Grid resolution group. Decreasing the Filter length can lead to improved spatial resolution, provided it remains sufficiently large to include an adequate number of particles for each of the nx by ny grid points.

The velocity field may be rendered by itself, or superimposed upon a background image. This is controlled by the Vector background list box in conjunction with a Scale factor.

Post processing

Selection of the most appropriate output file format (between .dfd and .dfi) depends on the type of post processing to be undertaken.

If the .dfi format is selected, then the PTV gridded velocity files may be fed back into DigiFlow as multi-plane images containing the velocity field. These can be processed using most of the standard DigiFlow tools, preserving the nature of their contents. For example, the Analyse: Time Average facility can act upon a sequence of PTV gridded velocity files to produce the time average velocity field. Similarly, the various other time series tools described in §5.6.1 can operate on these images, as can the general manipulation tools Recipe, Transform Intensity and Combine Images (see §§5.7.1, 5.7.2 and 5.7.3). There are standard recipes in the Recipe facility to aid with basic manipulations of this data. For example, the recipe Velocity.Background.Divergence recipe lets you change the background of the velocity field from the one saved during the PTV gridding process to display the in-plane divergence field. Similarly, there are recipes for vorticity, stream function, velocity potential, shear, *etc.*, and for adding scales and other similar graphical manipulations. Note that for PTV data, velocity gradients are as part of the gridding process by analytical differentiation of the weighting kernel rather than by a finite difference operation of the velocity field. This retains more of the velocity information available from the randomly distributed particles.

Saving the output in .dfd format is appropriate if post processing is to be undertaken using a third party or user-written program as the .dfd file contains an ASCII representation of the velocity field. Note that you can always convert a .dfi file into a .dfd file using Edit Stream (§5.1.6) or one of the other related image manipulation tools by simply specifying a .dfd file for the output.

5.6.7 Optical flow

The idea behind Optical Flow is that illumination is a conserved quantity that is advected by some velocity field.

5.6.7.1 Follow

Toolbutton:
Shortcut:
Related commands: `process Analyse_FollowOpticalFlow;`
`follow_optical_flow(..).`

This menu item provides a basic algorithm for extracting the velocity field from an optical flow. The key idea is that if illumination is conserved then we can write an advection equation for it of the form

$$\frac{\partial P}{\partial t}+u\frac{\partial P}{\partial x}+v\frac{\partial P}{\partial y}=0. \tag{35}$$

If we then assume that the velocity field (u,v) is constant over some region S containing intensities P_0, P_1, … P_N, then we may write this as the over determined system

$$\begin{bmatrix} \frac{\partial P_0}{\partial x} & \frac{\partial P_0}{\partial y} \\ \frac{\partial P_1}{\partial x} & \frac{\partial P_1}{\partial y} \\ \vdots & \vdots \\ \frac{\partial P_N}{\partial x} & \frac{\partial P_N}{\partial y} \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = -\begin{pmatrix} \frac{\partial P_0}{\partial t} \\ \frac{\partial P_1}{\partial t} \\ \vdots \\ \frac{\partial P_N}{\partial t} \end{pmatrix}. \tag{36}$$

We can then estimate (u,v) as the least squares solution to (36). In reality, finite difference approximations to the spatial and temporal gradients of the illumination are used, and the noise in the signal, plus the need for the least squares problem to be well conditioned, places a practical lower limit on the size of the region S, while an upper limit is imposed by the velocity field not really being constant.

This process, which is sometimes referred to as ‘feature tracking’, can be used in a variety of contexts. With relatively slow flows containing particles, it provides a computationally cheap method of obtaining an estimate of the velocity field, although the resulting velocity field is less accurate than that obtained by other methods. The process can be particularly valuable for looking at the flow and distortion of dye fields, although the user must be aware that it is not necessarily the fluid velocities that will be returned.

Note, the Tools: Recipes: Slave Process (§5.7.5) facility provides a cut-down version of this facility for providing an estimated velocity in real time. This cut-down version is provided through the dfc function **`follow_optical_flow`**`(..)`.

Inputs

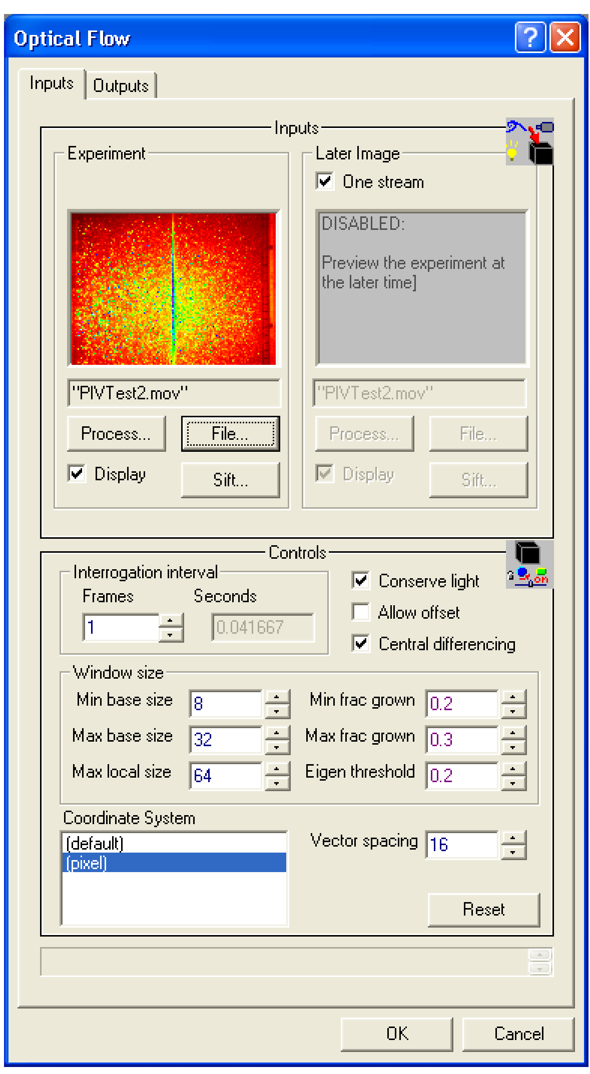


Figure 126: The inputs tab for Follow Optical Flow.

The Inputs tab bears some similarity to that for PIV (see §5.6.5.2) with either one or two input streams. If One stream is chosen, then the Interrogation interval group allows the number of frames between the first and second frame of each pair being processed. When the input is as two streams, the interval between each stream must be specified in seconds.

The Conserve light checkbox forces the mean intensity of each of the interrogation windows (the region given by S above) to be constant, Allow offset enables the temporal derivatives to be made in a semi-Lagrangian manner, and Central differencing makes the spatial derivatives a central second order approximation.

The Window size group controls the size of the interrogation window, which DigiFlow adjusts dynamically to give the best compromise between noise level, robust data, and the resolution of velocity gradients. This automatic adjustment is guided by minimum and maximum sizes for the base windows which DigiFlow applies to all interrogation points. DigiFlow will cause a window to increase above this base size, up to Max local size if the magnitude of the smallest (normalised) eigenvalue of the least squares problem falls below Eigen threshold (this indicates the least squares solution may be ilconditioned). If a bigger fraction of the windows are grown due to this criterion than Max frac grown, then DigiFlow will cause the base window size to increase (up to Max base size). If fewer windows are grown than the fraction Min frac grown, then DigiFlow will cause the base window size to decrease (down to Min base size).

The Coordinate System controls the conversion between pixel coordinates and any world coordinate system, while Vector spacing determines the spacing (in pixels) between the centres of the interrogation windows.

Outputs

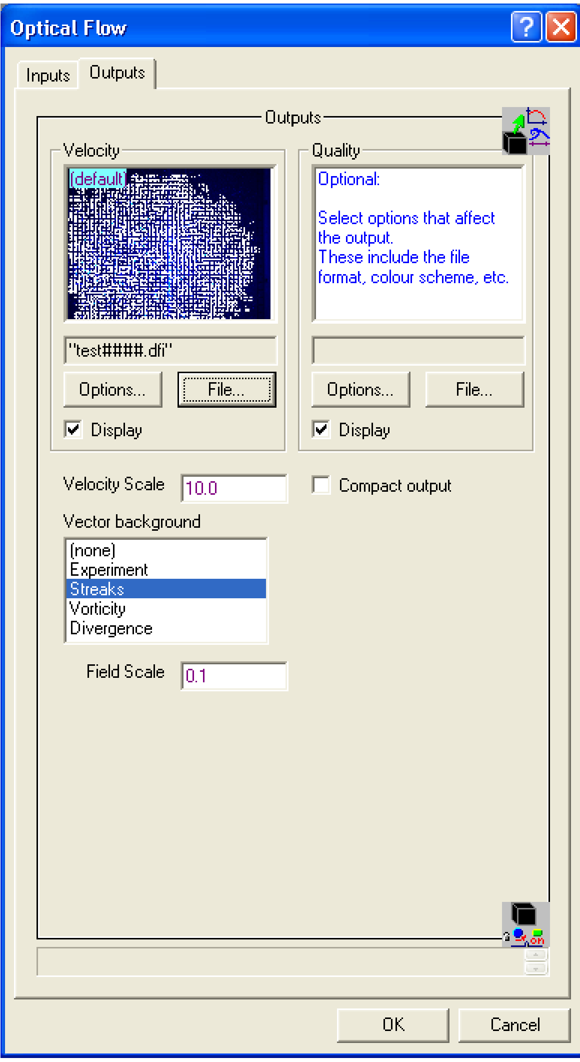


Figure 127: The outputs tab for Follow Optical Flow.

The principal output of Follow optical flow is the velocity field. This is specified by the Velocity selector. As with the PIV system, a scale for the arrows used to display this is specified by Velocity scale, and a background image may be placed behind the arrows with Vector background (the scale of which is controlled by Field scale, when appropriate). The Compact output check box forces the resolution of the output to be reduced so that data is saved only at the locations of the interrogation windows.

The Quality selector optionally stores information about the performance of the process. In particular, the first image plane gives the size of the minimum eigenvalue of the least squares process, while the second image plane gives the size of the window actually used.

5.7 Tools

5.7.1 Recipe

Toolbutton:

Shortcut:

Related commands: `process Tools_TransformRecipe(..)`

This process provides a simplified entry point to many commonly used image processing procedures. Internally, this facility uses the same mechanism as Transform intensity and Combine images described in §§5.7.2 and 5.7.3, but the interface here presents the user with a broad list of pre-written processes rather than requiring the user to enter their own code.

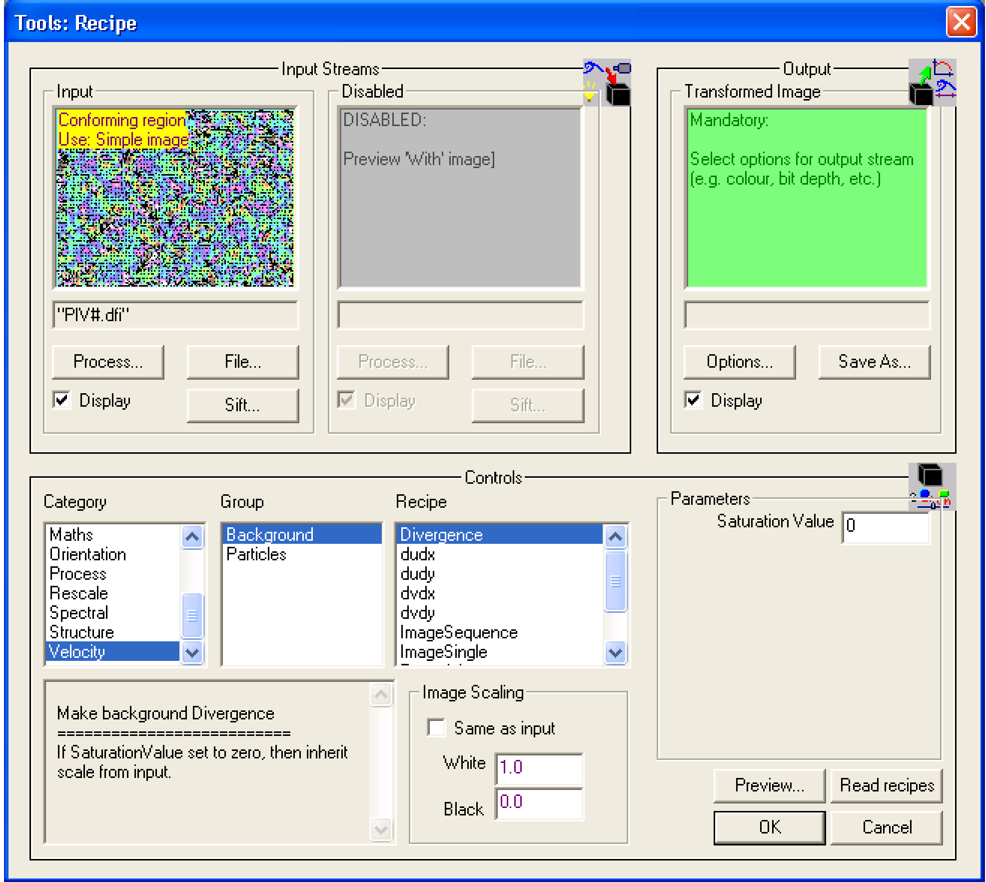


Figure 128 Transform an intensity using a recipe of transformations.

Depending on which recipe is selected, either one or two input image selectors are required. The Input selector is required for all functions. This selector determines the timing and other key features. The second stream, With, is required only for a subset of the functions. For some functions this will represent a single image, while for others it will be a sequence. The title of the group is changed to reflect these differences. The controls associated with the With selector are disabled when it is not required.

Both input selectors have the normal mechanism for their specification and the range of controls. This may be taken from a file using the File button, or from another Process. The input stream may be sifted (§4.3) to extract the desired subregion and times. This feature is activated using the Sift button (see §4.3) associated with the input streams.

The Output group specifies the destination of the transformed image using the Save As button. If this process is acting as the source for another process, the Save As button is suppressed (refer to §7 for further details). The colour scheme and other output options to be used for the output stream are set by clicking the Options button. Although the output image

will typically have a bit-map format, this is not always the case. Indeed, this tool can be used to transform a bitmap into a drawing, as will be illustrated below.

The Controls for this process centre on identifying the transformation. The predefined transformations are sorted by Category and Group. Each individual Recipe has a description that will be displayed beneath the selection lists. Some recipes will require one or more user-specified parameters. The required type for these parameters depends on the function selected. Some recipes produce images, and others produce drawings. The simplest way to determine which is by clicking the Preview button.

User-defined recipes

Users can add their own custom recipes to the list by creating a file named User_Recipes.dfc either in the current directory, or in the directory in which DigiFlow is installed. (A copy in the current directory will have precedence over one in the DigiFlow directory.) A typical entry for a single-stream recipe in this file would look like

```
Recipe.User.Stretch.Linear.Descr := "Stretch the intensity by a
     factor";
Recipe.User.Stretch.Linear.Code := {P*p0};
Recipe.User.Stretch.Linear.Prompt0 := "Factor";
Recipe.User.Stretch.Linear.Param0 := 2;
Recipe.User.Stretch.Linear.Check := {if (p0 = 1) {"No point
     multiplying by one"} else {null};
```

This would appear under Categgory User, Group Stretch, Recipe Linear. Here the recipe requires one parameter, producing the prompt Factor in the interface. The default value of this parameter is set by the `.Param0` variable, and the parameter is provided to the code as the variable `p0`. In this case, since a `.Check` variable is specified, the value of the parameter is checked. If the `.Check` code returns a string, then this is displayed as a warning message.

If two input streams are required, then the variable *xxx*`.With` should be defined, containing either `"sequence"` or `"single"`, depending on whether a sequence or only a single image is to be recovered from the With stream. The image recovered from the With stream is provided to the `.Code` in the variable `Pb` (or, for image planes in `Qb` – refer to §5.7.2 for further details).

The facilities available within the code segment `.Code` are exactly the same as those available in the Transform intensity and Combine images tools described in §§5.7.2 and 5.7.3. Up to 6 prompts may be requested, their types (integer, floating point or string) being determined by the type of the default value in the `.Param`*n* variable. Note that the description and code may be specified interchangeably as strings, code segments or memos.

The database of built-in recipes may be found in DigiFlow_Recipes.dfc in the DigiFlow installation directory.

5.7.2 Transform intensity

Related commands: `process Tools_TransformIntensity(..)`

This process allows the transformation of the intensities of an image stream using a sequence of user-specified operations. This produces a very versatile tool, but one which requires some experience to master. A simplified interface to the same underlying mechanism is provided in Transform recipe described in §5.7.1.

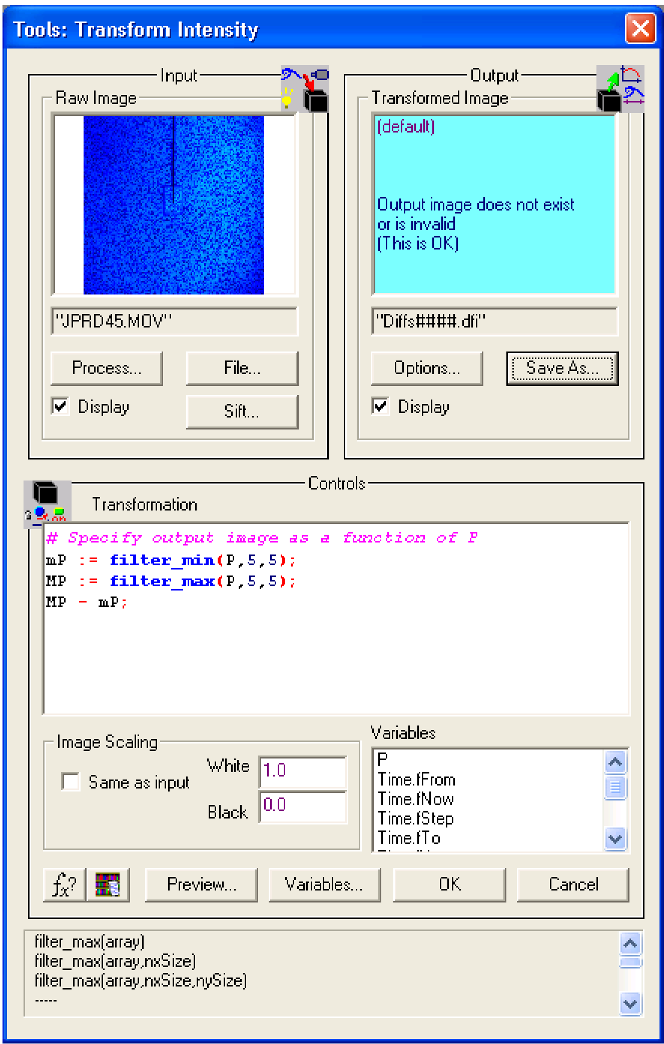


Figure 129: Transform an intensity using a mathematical expression.

A single image selector provides the input stream in the Input group. This may be taken from a file using the File button, or from another Process. The input stream may be sifted (§4.3) to extract the desired subregion and times. This feature is activated using the Sift button (see §4.3) associated with the input streams.

The Output group specifies the destination of the transformed image using the Save As button. If this process is acting as the source for another process, the Save As button is suppressed (refer to §7 for further details). The colour scheme and other output options to be used for the output stream are set by clicking the Options button. Although the output image will typically have a bit-map format, this is not always the case. Indeed, this tool can be used to transform a bitmap into a drawing, as will be illustrated below.

The Controls for this process centre on the transformation itself.

The Transformation edit box is used to specify the intensity mapping function using dfi code.

The basic image from the input stream is supplied in the array variable `P`. For simple images this will be a two-dimensional array. However, for more complex image formats (such as velocity fields stored in .dfi files), `P` will contain more than two dimensions. In such cases DigiFlow will also provide the same data split into its individual component two-dimensional arrays in the compound variable `Q`. For example, if the input stream contains a velocity field generated by the PIV facility (see §5.6.5.2) then `Q.u` and `Q.v` will contain the two components of the velocity field, and (depending on the options selected during the processing) `Q.Scalar`

may contain the vorticity field. Full colour images are supplied as their red, green and blue components with a three-dimensional P array: P[:,:,0] contains the red component, P[:,:,1] contains the green component, and P[:,:,2] contains the blue component. For convenience, these are also supplied as Q.Red, Q.Green and Q.Blue. The fx? button may be used to search for or provide information on specific DigiFlow functions.

DigiFlow also provides time information about the input stream through the Time compound variable. Typically this contains Time.fNow and Time.tNow giving the current frame number and time (in seconds) relative to the start of the entire input stream. An additional variable Time.iNow gives an iteration counter that is the frame number relative to the start of those that are actually being processed. Details of the entire input stream are provided through Time.fFirst, Time.fLast and Time.tFirst, Time.tLast that provide details of the first and last frame/time that exist in the input stream. Moreover, Time.fFrom, Time.fTo and Time.tFrom, Time.tTo provide information about which part of the stream is being processed.

The main variables available are listed in the Variables list box. This list does not, however, include any additional modifiers for the individual data plane variables beginning with Q. These modifiers include the description, scaling and (where appropriate) spacing of the data. A more comprehensive list may be viewed by clicking the Variables button. For further details, refer to the PIV data example below.

Note that regardless of the format of the input selectors, all processing is performed in floating point arithmetic and normally the images will be scaled between an intensity of 0.0 for the darkest parts and 1.0 for the brightest parts. By default, when the image is saved to an 8 bit format, intensities less than 0.0 will be mapped to 0 and those greater than 1.0 mapped to 255. Refer to §8 for further details on the interpreter within DigiFlow to evaluate expressions. The Preview button allows you to preview the result of the transformation before applying it to the whole image sequence.

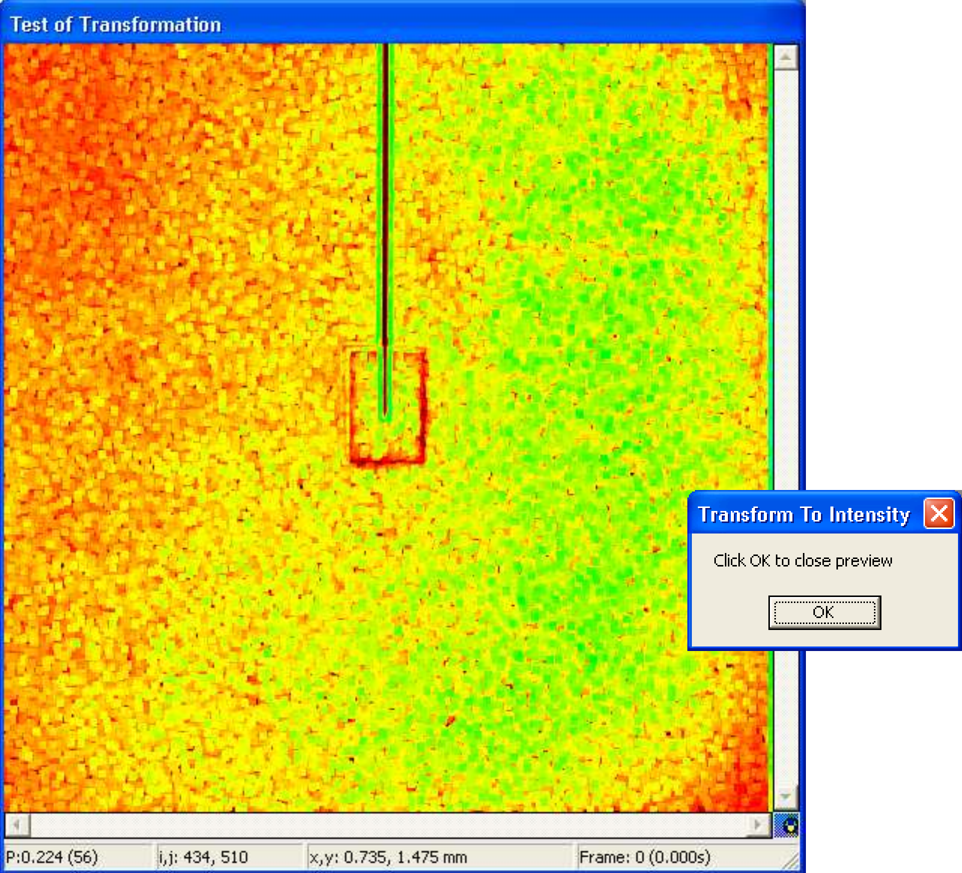


Figure 130: Example of previewing an image.

Note that the result of Transformation need not be an image, but can be a DigiFlow drawing. In this case the return value from code specified for the Transformation must be the handle to the drawing object (*i.e.* the handle returned by `draw_start(..)`). In this case, the output stream must have a format capable of containing a drawing.

If this feature is started from a dfc macro, then the code specified for the Transformation has access to functions and variables defined in that macro. In the case of variables, the `!` global access prefix must be specified.

A number of examples of transformation code are given below.

Rescaling an image

One of the simplest examples is rescaling an image so that its intensities always span the range 0 to 1, regardless of the input values.

```
# Find limits
vMin := min_value(P);
vMax := max_value(P);
# Rescale
(P-vMin)/(vMax-vMin);
```

This particular code segment would have problems if the image was all the same intensity as then `vMin` and `vMax` would be equal which would lead to division by zero. This problem could be overcome in a number of different ways. The most straightforward is illustrated below:

```
# Find limits
vMin := min_value(P);
vMax := max_value(P);
# Rescale
if (vMax = vMin) {
  0.5
} else {
```

```
    (P-vMin)/(vMax-vMin);
};
```

Filtering

One process used frequently in image processing is filtering. In DigiFlow a number of functions are provided specifically in support of this. In the context of the Transform intensity facility, all that is necessary is to specify the appropriate filter, if it exists. For example

```
filter_low_pass(P,5);
```

will implement a low-pass convolution filter with a 5×5 kernel. In particular, the intensity of each pixel will be replaced by the mean of it and its 24 nearest neighbours. The function `filter_convolution(..)` provides a more general alternative in which the convolution kernel is specified explicitly, allowing a broad range of filtering operations. (The `filter_low_pass(..)` function is effectively a call to `filter_convolution(..)` with all the kernel elements equal to $1/n^2$, where n is the specified size of the kernel.)

Built in nonlinear filters include erosion (`filter_min(..)`) and dilation (`filter_max(..)`).

Using DigiFlow's Fast Fourier Transform function allows the construction of filters in the wavenumber domain. For example, a low-pass filter could be constructed using the following Transformation code:

```
spec := fft_2d(P[0:128,0:128]);
k := sqrt(spec.kx^2 + spec.ky^2);
spec.re := where(k < 16,spec.re,0);
spec.im := where(k < 16,spec.im,0);
image := inverse_fft_2d(spec);
image.re;
```

Here, we make use of the wavenumber arrays returned by `fft_2d(..)` rather than having to work out the ordering in which the function returns the data. The `where(..)` function then simply sets all entries with wave numbers in excess of 16 to zero. Note that the `fft` functions can cope with arbitrary numbers of pixels, but are most efficient for powers of two and other small primes.

Contouring

Often we would like to contour an image for one reason or another. Suppose we just want to draw the contours in place of the image, then we could use the following code to step through the various contour levels, drawing only those contours that were at least 100 pixels long (thus discarding any high frequency 'noise'):

```
# Create image for output
out := make_like(P,0);
# Loop through thresholds
for thresh:=0.1 to 1 step 0.1 {
  # Find contours of this threshold and draw on blank image
  this := contour_image(P,thresh,fill:=thresh,minLength:=100);
  # Superimpose new contours on output image
  out := this max out;
};
# Return output image
out;
```

Obviously we could have superimposed the contours on the input image. Using other options in the `contour_image(..)` function would allow us to apply a low pass filter to the contours.

Similarly, we could fit a parametric curve to the contours, or perform a FFT and filter them to return a Fourier Descriptor of the enclosed region. This could be achieved by

```
this := pixel_contour(P,threshold:=0.9,minLength:=500);
if (this.found) {
  # Compute fourier descripter
  fft := fft_row(this.xy[0,:],this.xy[1,:]);
  fft.re := where(abs(fft.kx)<8,fft.re,0);
```

```
    fft.im := where(abs(fft.kx)<8,fft.im,0);
    cont := inverse_fft_row(fft.re,fft.im);
    out := scatter_to_array(P,cont.re,cont.im,fill:=0);
  } else {
    out := P;
  };
  out;
```

In this example we only find a single contour, the result of which is shown in figure131. The code could easily be modified to loop and so find all contours satisfying the length criterion. Note that the `scatter_to_array(..)` function may leave some gaps in the curve rendered as the curve is drawn using individual points rather than lines.

Figure 131: Eighth order Fourier Descriptor showing in black the region of the sheep's back with an intensity in excess of 0.9.

Fractal box count

Suppose we are interested in the fractal dimension of a contour from an LIF image (which may have been processed using the facility described in §5.6.3.2). In this case we would probably wish to have a log-log plot of the number of boxes verses the box size as the output. This may be achieved as follows:

```
# Extract fractal data
boxes := fractal_box_count(P,0.5);
# Fit least squares line
fit := fit_expression("1;ln(size);","size;",
     boxes[:,0],boxes[:,1],"ln(n);","n;");
curve := exp(evaluate_expression(fit,boxes[:,0]));
# Axes limits
minSize := min_value(boxes[:,0]);
maxSize := max_value(boxes[:,0]);
minNum := min_value(boxes[:,1]);
maxNum := max_value(boxes[:,1]);
# Create drawing
hDraw := draw_start(640,480);
draw_set_axes(hDraw,minSize,maxSize,minNum,maxNum,
     xLog:=true,yLog:=true);
draw_x_axis(hDraw,"Box size");
draw_y_axis(hDraw,"Number of boxes");
draw_create_key(hDraw,0.8*minSize+0.2*maxSize,
     0.6*minNum+0.4*maxNum,"Key");
# Draw data
draw_mark_type(hDraw,"plus");
draw_line_colour(hDraw,"red");
```

```
draw_mark(hDraw,boxes[:,0],boxes[:,1]);
draw_key_entry(hDraw,"Box counts");
# Draw fit
draw_line_colour(hDraw,"blue");
draw_lineto(hDraw,boxes[:,0],curve[:]);
draw_key_entry(hDraw,"Fit: slope="+(-fit.coeff[1]));
draw_end(hDraw);
hDraw;
```

Here we see the key element for producing a drawing: the code returns the drawing handle rather than an image.

Note that this code does more than the bare minimum. Not only does it plot the (hopefully power law) relationship between the number and the size of the boxes to cover the contour, but it also generates a least squares fit to that and plots it. Moreover, the key that is generated will inform the user of the slope (the fractal dimension) of that fit.

Changing background to velocity data

Suppose we have an image stream containing velocity and vorticity data, but we wish to change the background of the vectors to be speed rather than vorticity. In this case the following code could be used:

```
out.u  := Q.u;
out.v  := Q.v;
out.Scalar  := sqrt(Q.u^2+Q.v^2);
out;
```

In this example, we have extracted the velocity data without change. Note that we have used `Q.u` for the x velocity. The name ‘`u`’ comes from the description of the ‘`u`’ data plane stored in the input stream. We could equally have used the generic `Q.u` name instead. For the output *image*, we cannot use the ‘`u`’ name, but must resort to the generic ‘`u`’ name for the data plane, as we have not yet got a description for this plane. Similar arguments apply to the other two data planes. Indeed, we need to be a little bit careful as at present the output will inherit the ‘`Vorticity`’ description from the input, even though the output contains speed rather than vorticity.

This naming problem, along with an associated scaling one is handled as follows. If no other details are given, then the output will inherit the details from the corresponding input plane (*i.e.* the speed output plane will be called ‘`Vorticity`’ and have the same scaling as the vorticity). However, overrides can be specified. If we wish to do this for the above example we may specify a new description and scaling for the speed plane as follows:

```
out.u  := Q.u;
out.v  := Q.v;
out.Scalar  := sqrt(Q.u^2+Q.v^2);
out.Scalar_Descr  := "Speed";
out.Scalar_Black  := 0.0;
out.Scalar_White  := max_value(out.Scalar);
out;
```

Manipulations that you might want to apply to the velocity data include setting a different plot spacing and scale. This may be achieved in the above example by setting values for `out.u_xStep`, `out.u_yStep`, `out.u_Scale`, *etc.* Of course, you can also change the description for the velocity data, should you so wish.

Note that the input values of these additional controls are available through `Q.u_Scale`, `Q.u_xStep`, `Q.u_yStep`, … `Q.Scalar_Black`, and `Q.Scalar_White`. The input description is also available through `Q.u_Descr`, …, `Q.u_Descr`. Other features of the input image such as information about its coordinate system are available through `Q.dx`, `Q.dy`, `Q.xOrigin`, `Q.yOrigin`, `Q.xUnits`, `Q.yUnits` and `Q.CoordName`. These variables, however, are not listed in the Variables list box; a more comprehensive list may be viewed by clicking the Variables button.

Returning images

Images and drawings may be returned in a variety of ways from the code. In all cases, the final computed or referenced value represents the image returned, but this may be an array, a drawing handle, or a compound variable. The table below gives the possibilities and their interpretation by DigiFlow.

Return type	Components	Interpretation
2D array		Simple image. Colour scheme and scaling determined by dialog settings.
3D array		If the input is a 3D array, then the output will be interpreted in the same manner. For example, a 3D array output from full colour input will be taken as a full colour image, whereas a 3D array output from a velocity field input will be interpreted as a velocity field.
Compound	`.Image` (array) `.Black` (numeric) `.White` (numeric) `.ColourScheme` (string) `.LUT` (string or array)	Simple image, but with the code optionally specifying the intensity to set to black (`.Black` or `.black`) and white (`.White` or `.white`), and the colour scheme to be used (`.ColourScheme` or `.colourScheme` or `.LUT`). Only the specification of the image (`.Image`, `.image` or `.im`) is mandatory.
Compound	`.Red` (array) `.Green` (array) `.Blue` (array) `.Red_Black` (numeric) `.Red_White` (numeric) `.Green_Black` (numeric) `.Green_White` (numeric) `.Blue_Black` (numeric) `.Blue_White` (numeric)	Full colour image. The arrays specified for `.Red`, `.Green` and `.Blue` must all be two-dimensional and of the same size. The optional `.Red_Black`, `.Red_White`, *etc.*, set the intensities to be interpreted as black or white for each of the three colour components.
Compound	`.u` (array) `.v` (array) `.Scalar` (array) `.u_Scale` (numeric) `.u_xStep` (numeric) `.u_yStep` (numeric) `.u_ColourScheme` (string) `.Scalar_Black`	Vector field, specified in `.u` and `.v` arrays, with an optional background image specified in the `.Scalar` array. Both (all three) arrays must be the same size. The spacing between the vectors is determined by `.u_xStep` and `.u_yStep`, whereas the scale of the vectors is given by `.u_Scale`. The default colour for the arrows (black) may be changed by `.u_ColourScheme`. The black and white values of the optional background image, and the associated colour

	(numeric) .Scalar_White (numeric) .Scalar_ColourScheme (string)	scheme, may be changed by .Scalar_Black, .Scalar_White and .Scalar_ColourScheme, respectively.
Drawing handle		The drawing will be used. If the output is a raster image, then the drawing will be converted into a bitmap before saving.

5.7.3 Combine images

Related commands: process Tools_CombineImages(..)

This process allows multiple input image streams to be combined in arbitrary ways to produce an output image stream. This facility may be viewed as an expanded version of the Transform intensity described in §5.7.2. This produces a very versatile tool, but one which requires some experience to master. A simplified interface to the same underlying mechanism is provided in Transform recipe described in §5.7.1.

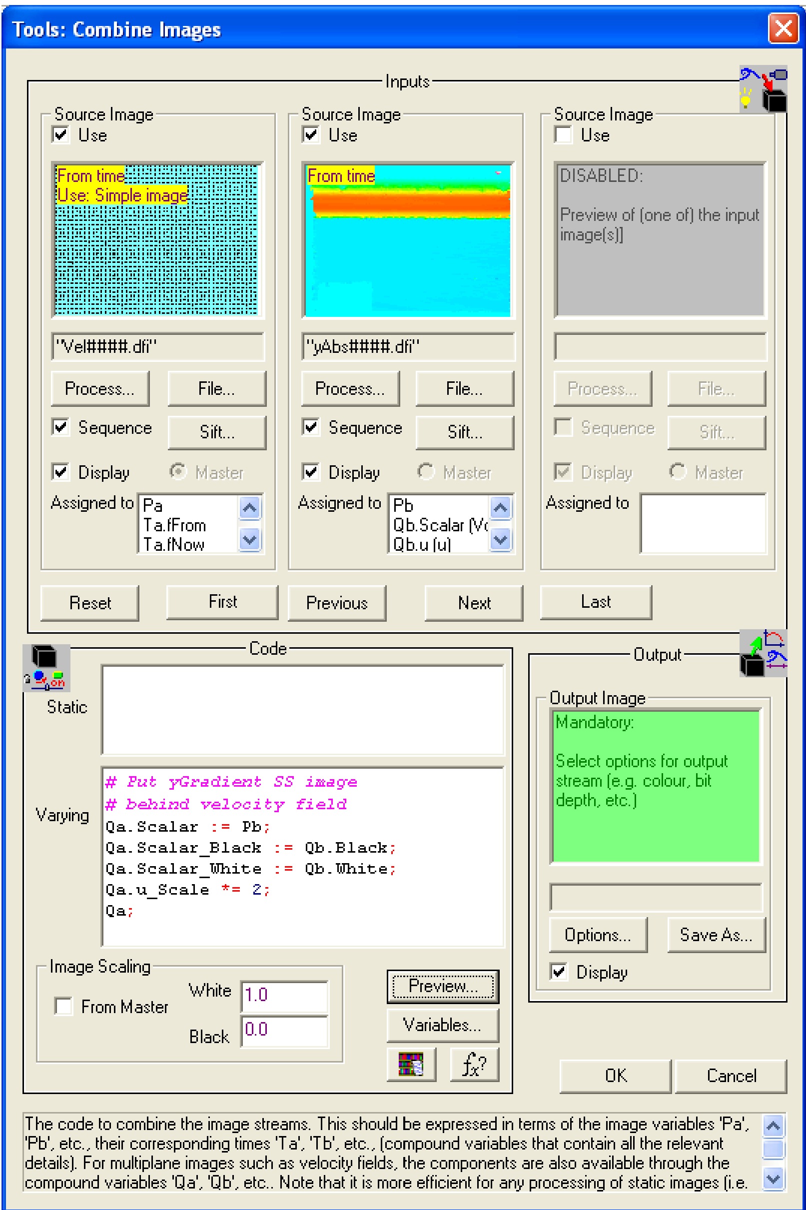


Figure 132: Combine image streams in an arbitrary way.

Up to 26 image selectors (fewer if operating on a free licence) provide multiple input streams in the Input group. These are visible three at a time, with the Next and Previous, First and Last buttons providing the ability to move along the list of selectors. Each image stream may be enabled or disabled through the Use check box, and each is assigned a two-letter name. For accessing the basic image the first letter is always `P`, while the second increases alphabetically from `a`, for the first stream, through to `z`, for the last possible stream. As we will see later, individual data planes for images with multiple planes of data may be accessed using `Qa`, `Qb`,… `Qz`, and drawings through `hDa`, `hDb`,… `hDz`. The Reset button will clear the inputs of all selectors, and clear the Use check box for all except the master stream (stream `a`).

The individual input streams may be taken as either dynamic or static. A dynamic stream, indicated by checking Sequence, will have one image read from it for each frame processed. In contrast, a static stream will read the input image only once at the start of the process.

Timing details may be set for both dynamic and static streams using the Sift buttons (see §4.3) to activate the standard Open Image dialog (§4.1). For a static stream, the effect of this is merely to select which image from a sequence is used as the static image.

Using the Save As button, the Output group specifies the destination of the combined image streams. If this process is acting as the source for another process, the Save As button is suppressed (refer to §7 for further details).

Timing details for the output stream are determined by the master input stream. This stream is selected using the Master radio button associated with each input stream. Note that while only one input stream can provide the master timing details, the process will be terminated when the first of the dynamic input streams runs out of images.

The Controls for this process comprise two code groups. The first code group, Static, may be used to define functions and manipulate the static input streams using their respective variables (*e.g.* `Pb`). This code is executed only once (except in so far as user defined functions – §8.9 – may be executed many times). Images from either static or dynamic streams may be referenced in the code, with those belonging to the dynamic stream corresponding to the first images in such streams. Any return value from this code will be discarded, but any variables created by the code will be available to the later Varying code.

The second code group, Varying, is executed once for each frame of the dynamic input stream in order to compute the output stream. This code may access any of the available images (*i.e.* whether they are from static or dynamic streams), as well as any variables or functions defined in the Static code. The final code statement provides the return value that is stored in the output stream.

The output from the Varying code may be an array, a drawing handle, or a compound value containing a number of different components. The various options here are identical to those for Tools Transform Intensity and are described at the end of §5.7.2.

As noted above, the basic image from the input streams is supplied in the array variables `Pa`, `Pb`, … For simple images these will be a two-dimensional arrays. However, for more complex image formats (such as velocity fields stored in .dfi files), `Pa`, `Pb`, … will contain more than two dimensions. In such cases DigiFlow will also provide the same data split into its individual component two-dimensional arrays in the compound variables `Qa`, `Qb`,… For example, if the first input stream contains a velocity field generated by the PIV facility (see §5.6.5.2) then `Qa.u` and `Qa.v` will contain the two components of the velocity field, and (depending on the options selected during the processing) `Qa.Scalar` may contain the vorticity field. Full colour images are supplied as their red, green and blue components with a three-dimensional `Pa` array: `Pa[:,:,0]` contains the red component, `Pa[:,:,1]` contains the green component, and `Pa[:,:,2]` contains the blue component. For convenience, these are also supplied as `Qa.Red`, `Qa.Green` and `Qa.Blue`. The fx? button may be used to search for or provide information on specific DigiFlow functions.. The fx? button may be used to search for or provide information on specific DigiFlow functions. If the input stream(s) contains a DigiFlow drawing (typically one or more .dfd files), then DigiFlow provides the drawing is available through its handle `hDa`, `hDb`, … `hDz` in addition to a bitmap version of it in the array variable `P`. Additional drawing commands may be added to the drawing handle, or it may be incorporated into a compound drawing using **`draw_embed_drawing`**`(..)`.

DigiFlow also provides time information about the input stream through the `Ta`, `Tb`,… compound variable. Typically this contains `Ta.fNow` and `Ta.tNow` giving the current frame number and time (in seconds) relative to the start of the entire input stream. An additional variable `Ta.iNow` gives an iteration counter that is the frame number relative to the start of those that are actually being processed. Details of the entire input stream are provided through `Ta.fFirst`, `Ta.fLast` and `Ta.tFirst`, `Ta.tLast` that provide details of the first and last frame/time that exist in the input stream. Moreover, `Ta.fFrom`, `Ta.fTo` and `Ta.tFrom`, `Ta.tTo` provide information about which part of the stream is being processed.

The main variables available are listed in the Variables list box. This list does not, however, include any additional modifiers for the individual data plane variables beginning with `Qa`, `Qb`,… Use the Variables button to generate a complete list of all the variables available and their contents. (This button loads all the image data then calls the **`view_variables(..)`** function.)

If this feature is started from a dfc macro, then the code specified for the Static and Varying code segments have access to functions and variables defined in that macro. In the case of variables, the `!` global access prefix must be specified. The Preview button allows you to preview the result of the transformation before applying it to the whole image sequence (see Figure 134).

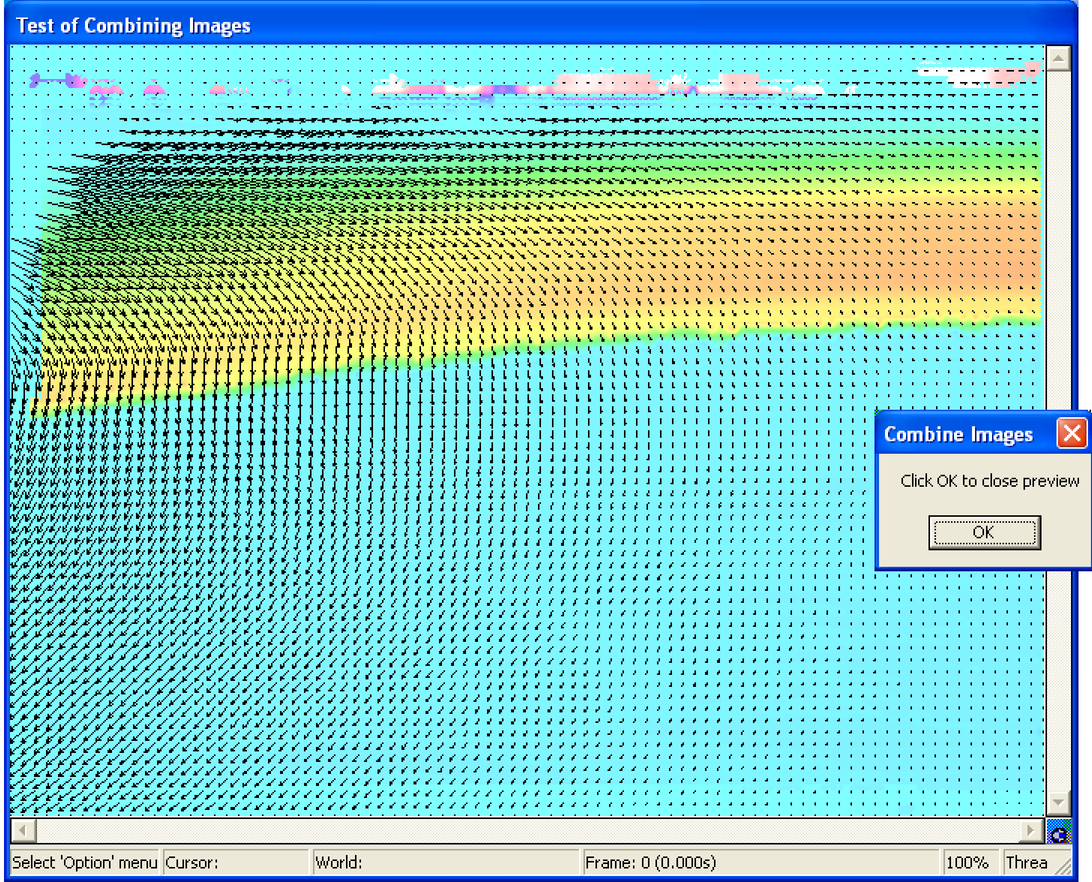


Figure 134: Preview for Tools: Combine Images.

As is standard with DigiFlow, all the input streams from integer format image files are interpreted as floating point values between 0.0 for the darkest parts and 1.0 for the brightest parts. By default, when the image is saved to an 8 bit format, intensities less than 0.0 will be mapped to 0 and those greater than 1.0 mapped to 255.

Note that the result of Varying need not be an image, but can be a DigiFlow drawing. In this case the return value from code specified for the Varying must be the handle to the drawing object (*i.e.* the handle returned by `draw_start(..)`). In this case, the output stream must have a format capable of containing a drawing. See the end of §5.7.2 for a complete list of the output options.

A number of non-trivial examples are given below.

Aligning images

In some circumstances it may be necessary to force alignment of images. This may be due to vibration of the camera, for example. Processing in this case would require two input

streams: the images to be aligned, and a reference image. If the misalignment is small, then the following code could achieve the desired effect. Suppose that we have some reference point located near the bottom left corner of the image. These reference points may be found by looking for some blobs with an intensity exceeding some predefined threshold. If the images to be processed are presented to input stream Pa and the reference image to Pb, then we can divide the task into two parts. The Static code would find the reference locations of the points, while the Varying code would not only find the current location of the points, but also shift the image accordingly. To save replication of code, we choose to define a user-defined function within the Static code that is responsible for finding the current location of the reference points:

```
function FindRef(Image,thresh) {
  # Find all the blobs
  blobs := find_blobs(Image,thresh);
  # Search for the largest blob.
  # Volume is stored in blobs[4,:]
  iBlob := max_index_x(blobs[4,:]);
  ret.x := blobs[0,iBlob];
  ret.y := blobs[1,iBlob];
  ret;
};
ref := FindRef(Pb[0:10,0:10],0.15);
```

For the Varying code we then use

```
now := FindRef(Pa[0:10,0:10],0.15);
dx := ref.x - now.x;
dy := ref.y - now.y;
shift(Pa,dx,dy);
```

Here the `shift(..)` function will only move the image to pixel resolution.

If you require subpixel resolution then use `shift_interpolated(..)` instead. Note, however, that the latter function must have arrays for the shift indices. This could be achieved using the following code segment for Varying code instead:

```
now := FindRef(Pa[0:10,0:10],0.15);
dx := make_like(Pa,ref.x - now.x);
dy := make_like(Pa,ref.y - now.y);
shift_interpolated(Pa,dx,dy);
```

Here we use the `make_like(..)` function to convert the shift increments into arrays for feeding into `shift_interpolated(..)`.

Velocity fluctuations

Suppose we are interested in examining the velocity fluctuations relative to some mean velocity field. In the simplest case we would look at the difference between the current velocity field and a time average field. Suppose the time varying velocity field is available through the Pa source stream, and the mean velocity field (perhaps computed by the Time Average facility described in §5.6.1.1) is specified as the Pb stream with the Sequence box cleared. In this case we need not specify any Static code. For the Varying code we could specify:

```
out.u := Qa.u - Qb.u;
out.v := Qa.v - Qb.v;
out.Scalar := Qa.Scalar - Qb.Scalar;
out;
```

In this particular case, since all three data planes are being treated in the same way and we are not changing the description of scaling of the data planes, we could treat all three planes simultaneously and simply use `Pa - Pb`. Validity of this, however, depends on the contents of the scalar field.

5.7.4 Accumulate

Toolbutton:

Shortcut:

Related commands: `process Tools_Accumulate(..)`

The operation of this menu item is similar to that of Tools: Recipe (see §5.7.1), but rather than taking a stream and returning one output image for each input image, here only a single output image is created. The simplest form of an accumulation is the mean of the image stream, but this tool is more flexible (but slightly slower) than the Analyse: Time Average (see §5.6.1.1). The dialog controlling Tools: Accumulate is almost identical to that of Tools: Recipe, with each accumulation recipe being assigned to a Group within a Category. Some of the accumulation recipies also require Parameters to control their action.

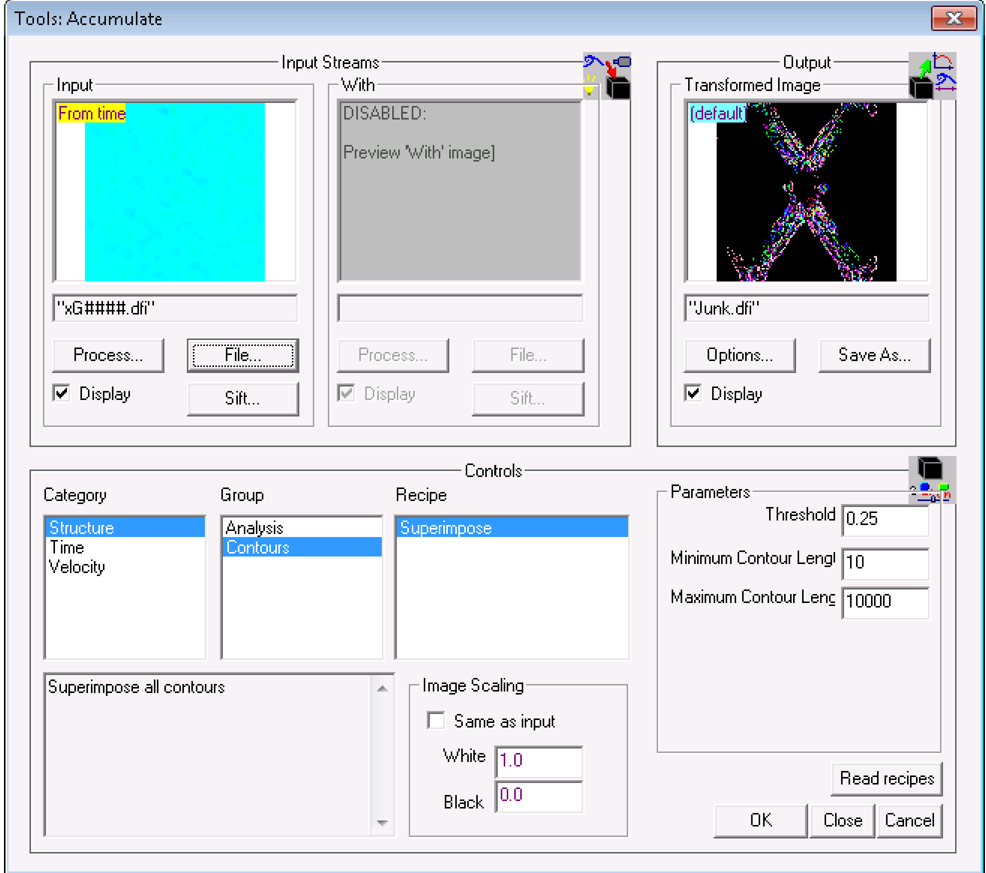


Figure 135: The Accumulate dialog.

User-defined recipes

Users can add their own custom recipes to the list by creating a file named User_Accumulate.dfc either in the current directory, or in the directory in which DigiFlow is installed. (A copy in the current directory will have precedence over one in the DigiFlow directory.) A typical entry for a single-stream recipe in this file would look like

```
Accumulate.User.Statistics.NumberOverThreshold.Descr := "Count the
     number of images exceeding a threshold";
Accumulate.User.Statistics.NumberOverThreshold.Accumulate := {
  if (Time.iNow = 0) {
    acc := make_like(P,0);
    thresh := p0;
  };
  acc += P > thresh;
```

```
};
Accumulate.User.Statistics.NumberOverThreshold.PostAccumulation :=
     {acc;};
Accumulate.User.Statistics.NumberOverThreshold.Prompt0 :=
     "Threshold";
Accumulate.User.Statistics.NumberOverThreshold.Param0 := 0.5;
Accumulate.User.Statistics.NumberOverThreshold.Check := {if (p0 <
    1) {"Must have positive threshold."} else {null};
```

This would appear under Categgory User, Group Statistics, Recipe NumberOverThreshold. Here the recipe requires one parameter, producing the prompt Threshold in the interface. The default value of this parameter is set by the `.Param0` variable, and the parameter is provided to the code as the variable `p0`. In this case, since a `.Check` variable is specified, the value of the parameter is checked. If the `.Check` code returns a string, then this is displayed as a warning message.

If two input streams are required, then the variable *xxx*`.With` should be defined, containing either `"sequence"` or `"single"`, depending on whether a sequence or only a single image is to be recovered from the With stream. The image recovered from the With stream is provided to the `.Code` in the variable `Pb` (or, for image planes in `Qb` – refer to §5.7.2 for further details).

The facilities available within the code segment `.Accumulate` are exactly the same as those available in the Transform intensity and Combine images tools described in §§5.7.2 and 5.7.3. Typically the code will start with an if statement, checking `Time.iNow`, to set things up on the first iteration. Once all the frames have been processed, the `.PostAccumulation` code is run to form the final output.

Up to 6 prompts may be requested, their types (integer, floating point or string) being determined by the type of the default value in the `.Param`*n* variable. Note that the description and code may be specified interchangeably as strings, code segments or memos.

The database of built-in recipes may be found in DigiFlow_Accumulate.dfc in the DigiFlow installation directory.

5.7.5 Slave process

Toolbutton:

Shortcut:

Related commands: `process Tools_SlaveProcess(..)`

Unlike most of the other features in DigiFlow, a Slave process is intended to provide a mechanism for extracting information directly from an input stream (including live video) for direct inspection by the user. The range of uses for this mechanism is continuously expanding and includes processes ranging from velocity calculation to aids in setting up and focusing a video camera.

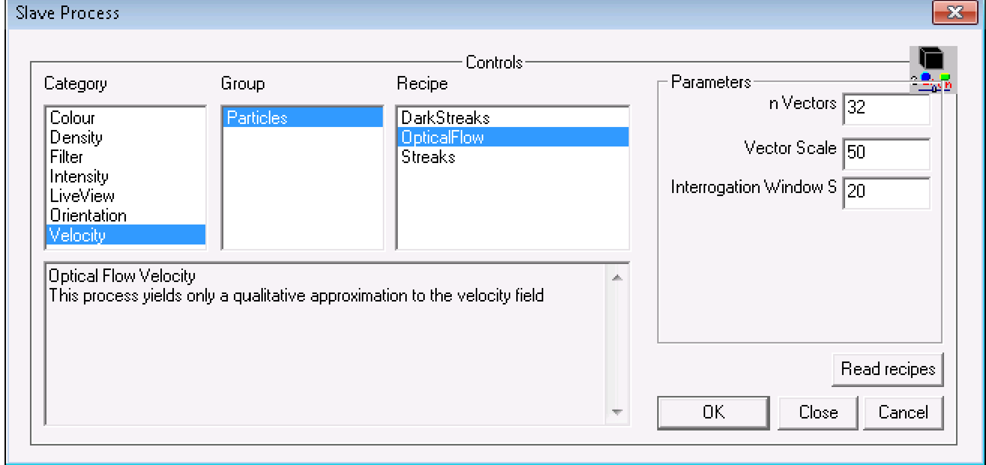


Figure 136: Dialog controlling a slave process.

As with the recipes described in §5.7.1, the interface provides a hierarchical interface to a series of macros providing the desired output. However, unlike a recipes feature, a slave process does not have an explicit image stream for the source of its data. Rather, it taps on to the image stream being displayed in the window that had the focus at the time the slave process was started. This master image may be a movie opened by File: Open Image, the output of another process, or live video. If the selected process is not too computationally expensive, then it will take the information from this stream and process it to produce output each time the master image is updated. When the master image is live video, then the output may be less frequent, but can be tailored to make use of adjacent frames, for example.

To specify a slave process, ensure the desired master image stream is the active window before selecting the slave process menu item. Slave processes in each Category are divided into one or more Group, each of which contains a selection of Recipes. Once the required recipe is selected, then the Parameters group may allow specification of optional parameters to provide some control over the process.

User-defined recipes

Users can add their own custom slave processes to the list by creating a file named User_SlaveProcess.dfc either in the current directory, or in the directory in which DigiFlow is installed. (A copy in the current directory will have precedence over one in the DigiFlow directory.) A typical entry for a single-stream recipe in this file would look like

```
Slave.User.Filter.LowPass.Descr := "Low pass filter";
Slave.User.Filter.LowPass.Code :=
 {hS := get_active_view();
  im := get_image(hS);
  if (not(is_null(im))) {
    hV := view(im.image);
    view_colour(hV,im.lut);
  };
  while (not(is_null(im))) {
    out := filter_low_pass(im.image,p0);
    view(hV,out);
    im := get_image(hS);
  };
  close_view(hV);
 };
Slave.User.Filter.LowPass.Prompt0 := "Length";
Slave.User.Filter.LowPass.Param0 := 3;
Slave.User.Filter.LowPass.Check := {if ((p0 <= 0) or ((p0 mod 2)
     <> 0)) {"Length must be positive odd integer"} else {null};};
```

This would appear under Categgory User, Group Filter, Recipe LowPass. Here the recipe requires one parameter, producing the prompt Length in the interface. The prompt is specified by the `.Prompt0` string and the default value of this parameter is set by the `.Param0` variable. The type for the returned parameter must be the same as that of its default value. The specified parameter is passed to the `.Code` as the variable `p0`. In this case, since a `.Check` variable is specified, the value of the parameter is checked. If the `.Check` code returns a string, then this is displayed as a warning message.

Unlike a recipe that needs to deal with only a single image (or single pair of images), a slave process needs to handle a continual stream of images, and look after both their extraction from the master image stream and their display in a suitable format. Typically the code for a slave process starts by determining the source for the master stream by a call to `get_active_view()`. The macro `get_image(..)` is then used to simplify the extraction of the images from the master stream, whether it be a standard image stream or live video. In both cases, `get_image(..)` waits efficiently for a new image to be available. In the above

example, a window (view) is created to contain the output, then further image are extracted repeatedly from the master stream until either the slave process or the master image stream is terminated.

5.7.6 To world coordinates

Toolbutton:

Shortcut:

Related commands: `process Tools_TransformToWorld(..)`

Transforms an image stream to make the associated world coordinate system orthogonal.

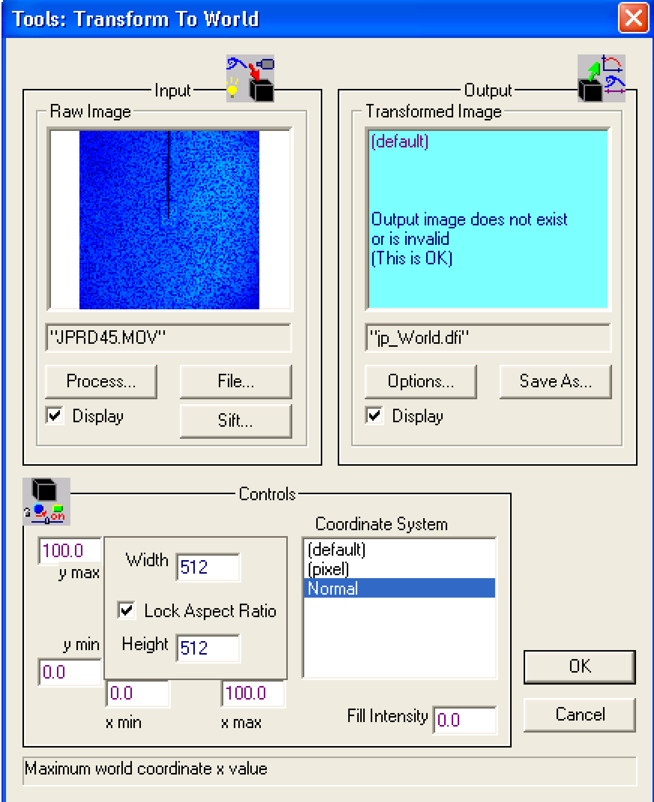


Figure 137: Transform an image stream to world coordinates.

A single image selector provides the input stream in the Input group. This may be taken from a file using the File button, or from another Process.

The Output group specifies the destination of the transformed image using the Save As button. If this process is acting as the source for another process, the Save As button is suppressed (refer to §7 for further details).

The Controls for this process include the specification of the Coordinate System to be used to map the image. The limits on the coordinates corresponding to the left (x min), right (x max), bottom (y min) and top (y max) of the output image, the size of which is specified by the Width and optional Height if the aspect ratio is not to be preserved.

Any pixels in the output image not corresponding to a point in the source image is filled with Fill Intensity, 0.0 representing the minimum value, and 1.0 the maximum.

For multi-plane images containing vector fields, the conversion process will also rescale the vector fields so that they are converted from pixel to world coordinate systems. Note that this only applies if the input stream is in pixel coordinates. This feature is of particular value if PIV velocity fields are computed in pixel coordinate yet are later required in world coordinates.

5.8 Window

The Window menu follows the standard Windows format and will not be given in detail here.

5.9 Help

Documentation for DigiFlow resides largely in this manual plus the dfc Help facility described in §4.5. This manual is distributed as both html format in DigiFlow.htm, and as an Acrobat file in DigiFlow.pdf.

5.9.1 Help (browser)

Toolbutton:
Shortcut:
Related commands:

Clicking on the Help entry in the Help menu will start up an instance of Internet Explorer and, where possible, take you to the table of contents in the html version of the DigiFlow User Guide. Selection the function key f1 at any point will have the same impact, but where possible will jump to the most relevant section of the guide.

5.9.2 dfc Help

Toolbutton:
Shortcut:
Related commands:

This will start up the dfc Help facility described in §4.5.

5.9.3 Auto help

Toolbutton:
Shortcut:
Related commands:

When checked, a browser window containing DigiFlow.htm is opened and automatically scrolled to the relevant section for each action undertaken within DigiFlow.

5.9.4 About DigiFlow

Toolbutton:
Shortcut:
Related commands:

The Help About DigiFlow option brings up a screen that gives you DigiFlow version and build date information (see figure).

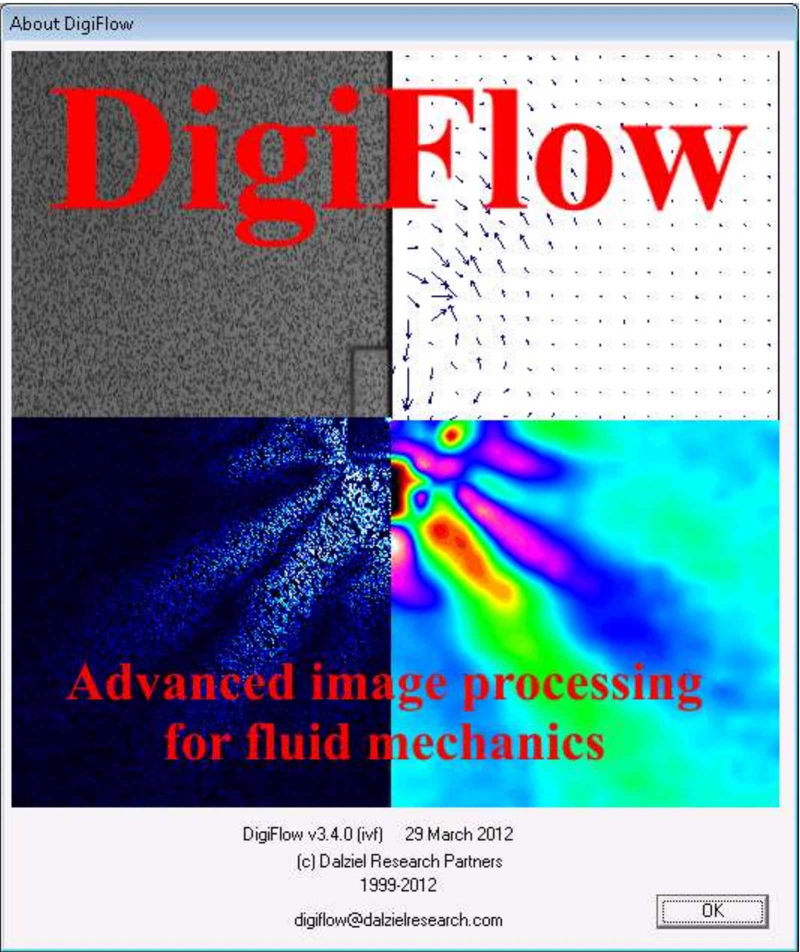


Figure 138: The About DigiFlow dialog.

6 Techniques

6.1 Determining black

There are a number of ways of determining the intensity to which black digitises. When working with two light sources for LIF the easiest way is to take three calibration images. The first image will have the left-hand light source only, the second the right-hand light source only, and the third with both light sources turned on. By then using Tools: Combine Images with these three images forming the Pa, Pb and Pc input streams, the following code will determine the black value and test the hypothesis of linear dye response simultaneously:

```
Image:=Pa+Pb-Pc;
black := mean(Image);
message("Black:"+black);
Image;
```

This code first evaluates the difference between the sum of the images due to the left and right hand light sources separately, and the image due to the two light sources working in tandem. If black were to digitise the some value *rblack*, then we would expect the resultant `Image` to be *rblack*. Inspection of the resultant image will highlight any defects in the images or assumptions, while the message box produced will give the black value.

An alternative method of determining black relies on the fact that it should have the same digitised value regardless of the camera aperture. Begin by acquiring two images of the same scene using different f-stops on the camera. The image with the wider aperture (smaller f/number) should not quite saturate; the second image should be with the lens stopped down by one f-stop. The scene should contain a broad range of intensities. Again using Tools: Combine Images, with the two images as Pa and Pb, use the following code to first generate a scatter plot, then fit a least squares regression to that line, and finally determine the intercept between this and a line of unit slope.

```
# Create scatter plot, scaling intensities from 0-1 to 0-255.
Image := make_array(0,256,256);
Image := scatter_to_array(Image,255*Pa,255*Pb,fill:=1,flags:=1);
# Find the centroid of the scatter plot
y := y_centroid(Image)/255;
x := x_index(y)/255;
# Fit line to plot, but only to central part of data
fit := fit_expression("1;x;","x;",x[50:250],y[50:250]);
# Look for root of x = a + bx => x = a/(1-b)
rblack := fit.coeff[0]/(1-fit.coeff[1]);
# Generate the fitted line
f := evaluate_expression(fit,x);
# Create a plot
hDraw := draw_start(640,480);
draw_set_axes(hDraw,0,1,0,1);
draw_x_axis(hDraw,"Bright image");
draw_y_axis(hDraw,"Dim image");
draw_create_key(hDraw,0.1,0.6,"Key");
draw_mark(hDraw,x,y);
draw_key_entry(hDraw,"Scatter plot",line:=false,mark:=true);
draw_line_colour(hDraw,"blue");
draw_lineto(hDraw,x,f);
draw_key_entry(hDraw,"Fitted curve",line:=true);
draw_text(hDraw,0.2,0.8,"Black:"+rblack);
draw_line_colour(hDraw,"green");
draw_line(hDraw,0,0,1,1);
draw_key_entry(hDraw,"Unit slope",line:=true);
draw_end(hDraw);
# Return the drawing handle as the "image"
hDraw;
```

The result of this code is shown in figure 139 below.

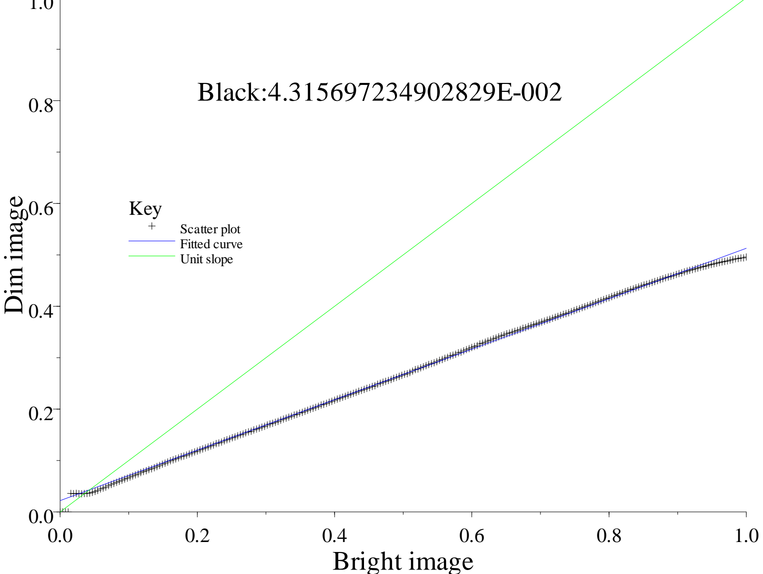


Figure: 139 Scatter plot used to determine ‘black’.

7 Chaining processes

A powerful feature of DigiFlow is the ability to chain multiple processes together, thus creating an efficient way of automating complex algorithms for processing image streams. In addition, piping images allows the full resolution of the image stream to be used, without the need to map the stream into some image format with a lower intensity resolution for each pixel.

The procedure for creating a process chain begins by identifying the process producing the output that is ultimately required, and work backwards from that point. For example, you may wish to determine the standard deviation of fluctuations in concentration from an image stream that contains corrected intensity images of a flow. The final process in this case is the Time Averaging found in §5.6.1.1.

In the Sequence group, click the Process button to indicate that the input image stream will be taken from another process. This starts the Image Source dialog.

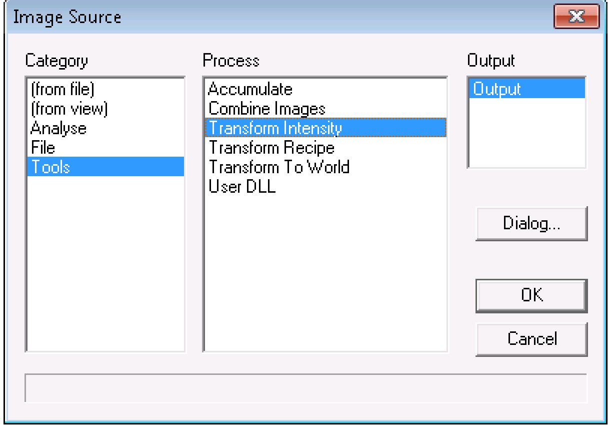


Figure 140: The image source dialog for connecting processes together.

This dialog contains three list boxes. The first, labelled Category, reflects the menu items controlling processes, with the addition of (from file) that allows the image stream to be taken from a standard file (enables the corresponding File button in the parent dialog). The Process list box then lists the various processes available within the Category list, and the Output list box indicates the one or more image streams produced. The source process is specified by the combination of items selected in these three list boxes.

Clicking Dialog (or OK if this is the first time the Image Source dialog has been started for this image stream and source process combination) will then start up the dialog box for the source process. In this case the Transform Intensity dialog described in §5.7.2.

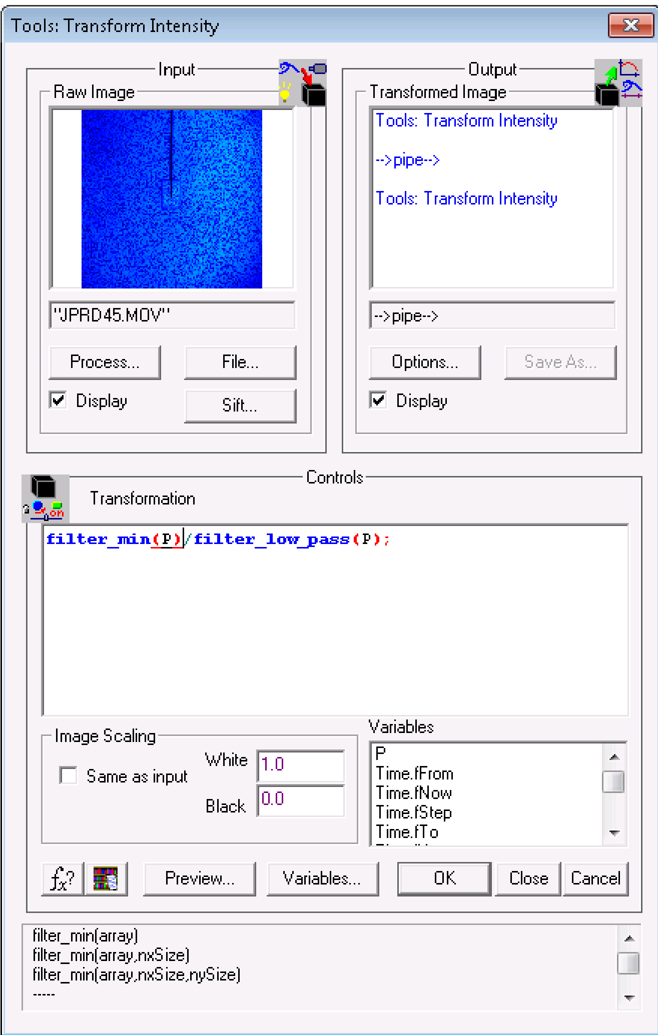


Figure 141: Transform intensity dialog as part of a process chain.

When a process is acting as a server for another process, the normal Save As button is disabled and the destination preview window indicates that the result will be piped into another process. The remainder of the dialog is unchanged. Once the image source has been specified using the File button, the timings and region may be set with the Sift button (§4.3). Alternatively, the chain may be extended by selecting another process with the Process button.

Exiting this dialog with the OK button returns to the parent dialog (here the Time Average dialog).

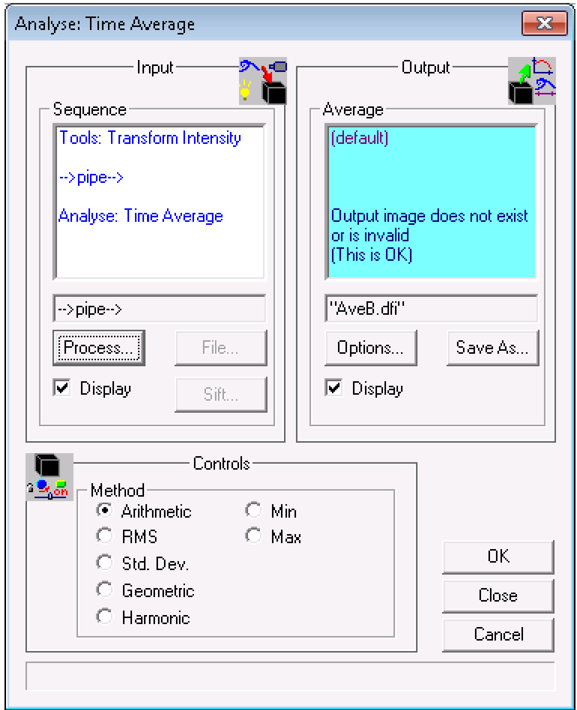


Figure 142: Time average dialog at the root of a chain.

This dialog, at the root of the processing chain, requires the destination for the Output image stream to be specified. The input image stream (the Sequence group) now has the File button disabled and indicates the source process in the preview pane. Once the specification of the dialog is complete, pressing OK will start the process.

8 Interpreter basics

DigiFlow contains a sophisticated mathematical interpreter capable of operating directly on numbers, arrays and/or entire images, and of controlling and automating complex processes. This interpreter is used widely within DigiFlow to provide the user with the maximum power and flexibility. The language utilised by this interpreter is often referred to as dfc code within this manual.

This section outlines the basic syntax, operators and execution control statements understood by the interpreter, and provides the key elements required to enter expressions and code segments in dialog boxes associated with the menu-driven processes provided by DigiFlow. Discussion of the broad range of basic and advanced functions and the use of the interpreter as a macro or command language are deferred until §9. Techniques used to access built in DigiFlow processes and write complete macros are described in §10. However, only brief details of the individual functions are given in this manual. More comprehensive documentation is to be found in the DigiFlow dfc Help facility described in §4.5.

Note that the interpreter is case sensitive with all pre-defined constants, functions, operators and variables specified by lower case names. Variables supplied by DigiFlow to represent a data stream or the result of a specific manipulation normally begin with an upper case character.

8.1 Syntax

The basic syntax of DigiFlow dfc code has some similarities with other high level languages such as C, Pascal or Matlab, but also has a number of significant differences.

Of key interest to experienced programmers is that the assignment statement is `:=`, similar to Pascal, with C-like `+=`, `-=`, `*=` and `/=` variants. The conditional assignment operator `?=` will only make an assignment if the target variable (on the right-hand side) does not already exist. Statements terminate with a semicolon (`;`), and blocks of code are delimited with braces (`{…}`).

Array and list indices utilise square brackets (`[…]`), with parentheses (`(…)`) being used for function arguments and mathematical brackets. Square brackets (`[…]`) can also be used to construct arrays, whereas double angle brackets (`<<…>>`) can be used to construct a compound variable.

Exponentation uses a caret (`^`), while `mod`, `div`, `max` and `min` are all binary operators. Logical **`true`** takes the numeric value of unity, while **`false`** is mathematically zero. Logical negation **`not`**`(..)` is a function, while the binary comparative operators are `=`, `<>`, `>`, `>=`, `<` and `<=`. Loops take the form **`for`** `i:=0` **`to`** `100` **`step`** `2 {…};` or **`while`** `(`*condition*`) {…};`. Conditional execution uses **`if`** `(`*condition1*`) {…}` **`elseif`** `(`*condition2*`) {…}` **`else`** `{…};`. Note that for both **`while`** and **`if`** that the condition must be delimited by brackets.

Variables may be integer, logical, real, array (of real values), string, compound or list (of arbitrary types). A compound variable is similar to a structure in other languages, but is more flexible, whereas a list is effectively an array that can contain a mix of variables of any or all types. Arrays can be constructed using **`make_array`**`(..)`, or directly using square brackets, *e.g.* `[1 2 3]` or `[1,2,3]`. Components within compound variables are separated with a dot and can be constructed using double angle brackets (*e.g.* `this := <<time:=3.0,frame:=6>>`).

Standard strings are specified using double quotes as `"This is a sample string"`. To include double quotes within the string, double the quote up, *i.e.* `"To use ""quotes"" within a string."` However, there is a maximum length to a standard string of 256 characters. Longer strings should be specified as a memo, delimited as `{/A memo can hold`

a string of arbitrary length. All DigiFlow functions where the 256 character limit on a standard string is likely to be limiting can accept memos as well as or instead of strings./}. Both strings and memos (and also blocks of code) may be concatenated using the + operator. Concatenating a numeric value with a string will cause the numeric value to be converted into a string before the concatenation. Similarly, concatenation of a string or a numeric value with a memo will result in a memo. For example, "File"+123 will yield the string "File123". More control over the format when converting a numeric value to a string is achieved using **make_string**(..). The function **replace_hashes**(..) provides a convenient method for constructing file names with a fixed number of digits including leading zeros.

A key difference compared with languages such as C or Pascal is that variable typing is all dynamic and determined by the assignment statement. A given symbol/name may change between any one of the basic types during the execution of code. Expressions involving mixed type are often permitted, with the result being, generally, what is expected. For example, if arr is an array, then the expression arr + 3 will add three to every element in arr.

By default, all user-defined functions are pure functions in that changes to any of the parameters are discarded with the return value being the way of returning all information to the calling code. The return value is the result of the last statement to be executed. It is not necessary for this statement to have a corresponding assignment. For example, the definition **function** Three() {ret := 3;}; and **function** Three() {3;}; will both return the integer three.

8.2 Variables

DigiFlow allows the use and creation of variables within all code segments. Variable names may use any alphanumeric character, plus the underscore. Names must not start with a numeric character. Variable names are case-sensitive.

8.2.1 Simple variables

There are four basic types of variable: integer, floating point, array and string. Additionally, there are a number of special purpose variable types, specifically memo, code and null. The last of these, **null**, is simply a place holder with no value, while memos provide a container for blocks of text that are too long to fit in standard strings. Code variable are a specialist form of memo that contain executable code.

Normally, declaration statements are not required to create a basic variable: it need simply occur on the left-hand side of an assignment statement (§8.3). However, in some cases an array of a particular size may be required, and in such cases the **make_array**(..) function, or one of the other more specialist array constructors (*e.g.* **make_like**(..), **random_array**(..), **gaussian_array**(..), **x_index**(..), **y_index**(..)) should be used on the right-hand side of an initial assignment statement. Similarly, a list can be created and initialised using **make_list**(..).

Both arrays and lists of strings can also be constructed in line using square brackets. If the contents between the square brackets are purely numeric, then an array is constructed. If the contents includes any strings, then a list is created. For example, [99, 98, 97, 96] will construct a one-dimensional array with four real elements (there are no integer arrays). In contrast, ["dog", "cat", "mouse"] produces a list with three string elements. The elements of a list need not all be of the same type, thus ["two", 3, "four"] is valid. Further information about arrays is given below in §8.4 and lists in §8.5.

A null value can be obtained either by assignment of **null**, or as the return value of certain functions (sometimes representing an error condition).

Memo variables can be constructed using the following syntax: `MyMemo := {/The memo is contained between an openning brace-slash pair, and a corresponding closing slash-brace pair./};`. Whereas standard strings in DigiFlow have a fixed maximum length, memos can be of arbitrary length. In many places, strings and memos may be used interchangeably.

Blocks of code are delimited by braces, whether as part of a control structure (*e.g.* a **for** loop or **if** block) or when being assigned to a variable. For example `MyCode := {a += 5;}; execute(MyCode);` creates the code variable `MyCode` then executes it.

For mathematical computations, type conversion will take place automatically as and where it is appropriate. For example, multiplying an array by a scalar will produce an array. Division of two integers will produce a real (floating point) value (integer division is achieved using the `div` operator).

8.2.2 Compound variables

Compound variables are similar to "structures" or user-defined "types" in other languages. Compound variables may be used to store more than one value of the same or different types. They are distinguished by having a dot (`.`) within their name. The part of the name to the left-hand side of the dot is the name of the compound variable, while the part of the name to the right-hand side is the name of the component: `name.component`. Each component may itself be any DigiFlow variable type, including arrays, lists and compound variables.

Whereas in most languages, the components contained within a compound variable need to be declared in advance, this is not true for DigiFlow. Here a compound variable is created by a standard assignment statement, and as many component variables as required may be added. Moreover, each of these component variables may themselves be compound variables.

The following example illustrates the use of simple and compound variables.

```
Start := 0;                   # Assignment to a simple variable
Using.Code := {1 - P};        # Create a compound variable
                              # and component variable
Using.File := "Test.dfm";     # Add a second component variable
Using.File_Time.FromStep := 0; # Component variable Time is a
                              # compound variable
Using.File_Time.ToStep := 1;
...
Result := MyProc(Start,Using); # Pass both simple and compound
                              # variables to a function.
```

If an existing simple variable appears on the left-hand side of a compound variable assignment, then the original contents of the simple variable will be discarded and a new compound variable of the same name created. In particular,

```
Var := "simple";              # Simple variable
Var.Handle := 1;              # The string "simple" is discarded
```

Both simple and compound variables may be passed to functions or returned from functions (see §8.9). Compound variables are of particular value dealing with the processes that can be started from menu items (see §10.1.1).

In some circumstances, it can be convenient to assign multiple parts of a compound variable in a single statement. This is particularly true when calling a function which takes a parameter as a compound variable, but where you do not have such a compound variable setup already. Consider the assignments

```
This.time := 1.0;
This.frame := 25;
This.file.name := "Test.dfi";
This.file.sequence := false;
```

These may be reduced to the single compound statement:

```
This := <<time:=1.0; frame:=25; file:=<<name:="Test.dfi";
     sequence:=false;>> >>;
```

The compound variable constructors, the << … >> pairs, are used to bracket the values to be combined into a single compound variable. As illustrated in the above example, the compound constructors may be nested.

8.2.3 Type query functions

The type of a given variable may be determined through one of the inquiry functions `is_array(..)`, `is_list(..)`, `is_code(..)`, `is_compound(..)`, `is_integer(..)`, `is_memo(..)`, `is_null(..)`, `is_numeric(..)`, `is_real is_string(..)`.

8.3 Assignment

Assignment takes place once all the operations and function evaluations are complete, if there is an assignment operator and variable at the start of the expression (*e.g.* `a := b+c;`). If there is no assignment, the result will be discarded, or, if it is the last result in a segment of code, it will be returned to the routine calling the interpreter.

The various assignment operators are listed below:

Assignment Operator	Description	Example
`:=`	Standard assignment. The result of the expression on the right-hand side is stored in the variable on the left-hand side.	`MyArray := (Pa + Pb)/2;`

`+=`	Increment assignment. The result of the expression on the right-hand side is added to the contents of the variable on the left-hand side and the result stored back on the left-hand side.	`Count += 1;`
		`# This is equivalent to:`
		`Count := Count + 1;`

`-=`	Decrement assignment. The result of the expression on the right-hand side is subtracted from the contents of the variable on the left-hand side and the result stored back on the left-hand side.	`Total -= a;`
		`# This is equivalent to:`
		`Total := Total - a;`

`*=`	Multiple assignment. The result of the expression on the right-hand side is multiplied by the contents of the variable on the left-hand side and the result stored back on the left-hand side.	`Value *= 2;`
		`# This is equivalent to;`
		`Value := Value*2;`

`/=`	Fraction assignment. The contents of the variable on the left-hand side is divided by the result of the right-hand side and the result stored back on the left-hand side.	`Test /= f;`
		`# This is equivalent to:`
		`Test := Test/f;`

`?=`	Conditional assignment. An assignment is made only if the target variable does not already exist. If the target variable does exist, then its contents remain unchanged.	`This ?= default;`
		`# The above is equivalent to`
		`Tmp := default;`
		`if (not(exists("This"))) {This := default;};`

8.4 Arrays

All array variables are inherently four-dimensional, although in most cases only the first one or two dimensions are used and some cases the dimensions may be collapsed to make a vector (scalar). Use of specific elements within an array, and assignment to specific elements of an array may be performed as shown below. Note that an assignment statement specifying

specific array elements requires the array to exist already. If the target array of an assignment does not already exist, then the assignment can only specify the entire array.

Arrays are generated as the result of expressions and as the return value of many dfc functions. DigiFlow also includes two functions specifically designed to construct arrays: `make_array(`*fill*`,`*nx*`,`*ny*`,…)` and `make_like(`*template*`,`*value*`)`. In the first case, between two and five parameters may be specified to the function, the first giving the value the array should be initialised with, and the remainder giving the dimensions of the array (up to four dimensions can be specified, although only the first dimension is mandatory). The *fill* parameter must be either an integer or floating point scalar value. The second constructor, `make_like(..)`, takes the specified template (which must be an existing array) as a guide to the dimensions of the array that is required, and initialises it with *value*. Unlike *fill* in `make_array(..)`, the *value* passed to `make_like(..)` may be an array, the values of which will be packed into the new array (*e.g.* the function may be used to convert a two-dimensional array into a one-dimensional array, or vice versa), filling any extra values in the new array with zeros, or discarding any surplus values from *value* if the total number of elements do not coincide. If *value* is a scalar (integer or floating point value), then it is simply replicated to each of the elements in the newly constructed array.

Array	Description	Example
a	For array variables in expressions, the entire array will be utilised. For arrays on the left-hand side of assignment statements, the old contents of the variable will be discarded and replaced by the result of the expression.	`# Average two images` `This := (First + Second);`

a[*i*,*j*]	Access to the *i*,*j*th element of the array *a*. Specification in an expression will return a floating point scalar. Specification on the left-hand side of an assignment will cause only the *i*,*j*th element to be updated. If the right-hand side returns an array, then the corresponding *i*,*j*th element from the right-hand side is used. If the right-hand side returns a scalar, then this value is used.	`# Location of centre`
		`i := x_size(Background)/2;`
		`j := y_size(Background)/2;`
		`# Intensity at centre`
		`iCentre := Background[i,j];`

a[i_0:i_1,j_0:j_1]	Access to the sub-array of a spanning from i_0 to i_1 and j_0 to j_1. Specification in an expression will return an array of size $(i_1 - i_0 + 1)\times(j_1 - j_0 + 1)$. Specification on the left-hand side of an assignment will cause only this sub-array to be updated. If the right-hand side returns an array, then the corresponding sub-array of elements from the right-hand side is used. If the right-hand side returns a scalar, then this value is used. If one or both limits are omitted, then the corresponding limit to the dimension will be used. Hence `a[:,:]` corresponds to the entire two-dimensional array.	`# Men intensity in window;`
		`Average := mean(`
		`Image[100:200,`
		`100:200]);`
		`# Increase gain`
		`Image[100:200,100:200] *=`
		`2/Average;`

a[i_0:i_1:s_i,j_0:j_1:s_j]	As with the above form, but access elements at intervals of s_i and s_j in the two dimensions. Note that s_i, s_j can be negative if the corresponding limits are in reverse order. In this case the order of elements will be reversed.	`reduced := this[::2,0:10:5]`
a[k]	Access to the kth element of a one-dimensional array. If a is specified in an expression, then this will return a floating point scalar. Specification on the left-hand side of an assignment will cause only the kth element to be updated. The right-hand side must evaluate to a scalar numeric value.	`Red := LUT[0:255,0:0];` `Red[0] := 0;`
a[k_0:k_1]	Access to the one-dimensional sub-array spanning from the k_0th to the k_1th element of a one-dimensional array. It does not matter if the array is a column or a row.	`Green := LUT[1:255,1:1];` `Green[0:128] := 0.5;`
a[k_0:k_1:s_k]	As with the above form, but access elements at intervals of s_k. Note that s_k can be negative if $k_1 < k_0$.	`reverse := this[nx-1:0:-1]`

a[]	The entire array with all its dimensions. This is equivalent to a[:,:,:,:]. If on the right-hand side of an expression, then it is simply equivalent to specifying a. However, if on the left-hand side, the array elements are replaced by the right-hand side, maintaining the size and shape of the array.	`# Assign to all elements of array`
		`this[] := 5;`
		`# Replace array with scalar`
		`this := 5;`

[*v0*,*v1*,…]	When at the end of a variable name, a [..] pair indicates array indices (or a range of indices) used to access an element (or range of elements) from an array. However, if not at the end of a variable name, then a [..] pair is used to construct an array from the list of numeric values it encloses. If the data is all on the same line, or there is only one data item per line, then a one-dimensional array is constructed. If there is more than one item per line and more than one line, then a two-dimensional array is constructed. For more general input, refer to **read_data**(..).	`Dash := [2,2,4,2];`

Additionally, it is possible to use an array containing integer values as an index into another array. For example,

```
x := x_index(10);
y := [1 4 5];
x[y] := 0; # Zero some elements
z := x[y+1]; # Extract some elements
```

returns one-dimensional arrays in x and z. These contain [0 0 2 3 0 0 6 7 8 9] and [2 0 6], respectively. If the index array is two-dimensional, then it may be used to access multi-dimensional source/target arrays. For example

```
x := x_index(10,10) + y_index(10,10)/10;
y := make_array(0,3,2);
y[0,:] := [5 2];
y[1,:] := [3 6];
y[2,:] := [9 4];
z := x[y];
```

gives z as [5.2 3.6 9.4]. An array index of this form always returns or absorbs a one-dimensional array. Similar functionality may also be obtained using **indirect**(..), **sample_values**(..) and **scatter_to_array**(..).

One-dimensional array indices can be used for each of the dimensions of the source/target array to extract or assign over an ordered two-dimensional space. Moreover, one-dimensional array indices may be used in conjunction with index spans or fixed index values. For example,

```
x := x_index(10,10) + y_index(10,10)/10;
i := [1 4 5];
z := x[i,2:3];
```

gives z as the two-dimensional array [[1.2 4.2 5.2][1.3 4.3 5.3]]. Although the above example is used with an array index on the right-hand side, a similar arrangement can be used for a combination of normal and array indices on the left-hand side of the assignment.

8.5 Lists

A list is similar to an array in that it contains multiple values which are accessed by specifying different indices or ranges of indices. However, unlike an array, a list can contain a mix of different data types. For example, *a*[0] might contain an integer, *a*[1] might contain a string and *a*[2] might contain an array, a compound value or indeed another list.

All list variables are inherently two-dimensional, although in most cases only the first dimension is used. Use of specific elements within a list, and assignment to specific elements of a list may be performed in the same way as for regular arrays. As with arrays, an

assignment statement specifying specific list elements requires the list to exist already. However, unlike arrays, computations cannot be performed simultaneously on the entire list, although lists can be passed as arguments to functions, *etc*.

Lists are generated as the result of the return value of some dfc functions (they cannot be the result of expressions other than a simple assignment). DigiFlow also includes a function specifically designed to construct arrays: **make_list**(*fill*,*nx*,*ny*). As with **make_array**(..), the list is initialised to the value specified in *fill*; this may be any data type, including a list. The second, and optionally the third, parameter then specifies the dimension(s) of the list.

Some restrictions apply to list elements containing arrays, compound values or lists. In particular, the list syntax does not allow direct access to components of such values, although the list may contain an array, compound value or list in its entirety. For example

```
List := make_list(null,3);
cValue.string := "Valid example";
cValue.version := 1;
List[1] := cValue;
…
this := List[1];
message(this.string);
List[1].string := "Replacement string";
A := x_index(100);
List[2] := A;
B := List[2][10:20];
```

is valid, while

```
List := make_list(null,2);
List.version[1] := 1; # List is a list, not a compound value
message(List.string[1]); # List is a list, not a compound value
```

is not.

The rules governing indices for lists is the same as those for arrays, with the one difference that lists are limited to two dimensions only. Thus, techniques such as array spans and index arrays can be applied to lists.

8.6 Operators

A complete list of the operators understood by DigiFlow is given below, grouped in order of the precedence (*i.e.* the order in which they are computed). For arrays, all operations are computed element by element. Hence, two arrays multiplied together produce and array where each element is the product of the two corresponding elements in the two source arrays (*i.e.* not matrix multiplication).

Group	Operator	Description	Examples
Association			
	(…)	Brackets. Terms within innermost brackets computed first.	
Unary			
	-	Negative. -*a* returns the negative of a.	
Power			
	^	Exponentation. *a*^*b* raises *a* to the power of *b*.	3^2 p^(1/2)
Term			
	*	Multiplication. *a***b* multiplies *a* by *b*.	3*2 2.1*sin(x*pi)
	/	Division. *a*/*b* divides *a* by *b*.	1/2 exp(r/p)

	`div`	Integer division. a `div` b returns the integer part of a/b.	p `div` 16 f `div` (1 + g)
	`mod`	Modulo division. a `mod` b returns $a - c$ where c is the largest integer multiple of b less than or equal to a.	q `mod` 10 (i+1) `mod` n
Sum			
	`+`	Addition. $a+b$ adds a and b. Also used to concatenate strings, memos or code variables.	3 + p/2 log(1+x)
	`–`	Subtraction. $a–b$ subtracts b from a.	1.9 – p pi*sin(x)–pi/2*cos(x)
	`min`	Minimum. a `min` b returns the lesser of a or b.	
	`max`	Maximum. a `max` b returns the greater of a or b.	
Group			
	`=`	Equality. a `=` b returns true (1) if a and b are equal, or false (0) if unequal.	
	`<>`	Inequality. a `<>` b returns false (0) if a and b are equal, or true (1) if unequal.	
	`>`	Greater than. a `>` b returns true (1) if a is greater than b, or false (0) if a is less than or equal to b.	
	`>=`	Greater than or equal to. a `>=` b returns true (1) if a is greater than or equal to b, or false (0) if a is less than b.	
	`<`	Less than. a `<` b returns true (1) if a is less than b, or false (0) if a is greater than or equal to b.	
	`<=`	Less than or equal to. a `<=` b returns true (1) if a is less than or equal to b, or false (0) if a is greater than b.	
Logical			
	`and`	Logical and. a `and` b returns true (1) if both a and b are true.	
	`Or`	Logical or. a `or` b returns true (1) if either a or b are true.	
	`eor`	Exclusive or. a `eor` b returns true (1) if only one of a and b is true.	
	`xor`	Identical to `eor`.	

8.7 Constants

Constant	Value	Description
true	1	Logical true. In arithmetic operations, true takes the value of unity.
false	0	Logical false. In arithmetic operations, false takes the value of zero.
pi	π	Approximately 3.141592653…

`null`	no value	Used to indicate that no value is specified. This may be tested by the `is_null(..)` function. Some functions (*e.g.* `read_image(..)`) return a `null` to indicate failure. Null values cannot take part in any expression except as the parameter to the `is_null(..)` function.
`wait_for_ever`	–1	This constant is intended for use as a *timeout* parameter in some of the thread and timing related functions (*e.g.* `kill_thread(..)`). Specifying this value will cause the corresponding function to wait for completion.
`do_not_wait`	0	This constant is intended for use as a *timeout* parameter in some of the thread and timing related functions (*e.g.* `kill_thread(..)`). Specifying this value will cause the corresponding function to return immediately and not wait for completion.

8.8 Execution control

Control	Description	Example
`# comment`	Comment. Ignore all text up to the end of the line.	`a := 3; # Initialse`

`if` (*condition*) {*code*};	If statement. The *code* is executed only if *condition* returns a nonzero scalar value. For array conditions, the `where(..)` function should be used.	`if (Failed) {`
		`  P := P^2;`
		`};`

`if` (*condition*) { $code_1$ } `else` { $code_2$ } ;	If statement with else clause. If *condition* is a nonzero scalar value, then $code_1$ will be executed, else if *condition* is a zero scalar value, then $code_2$ will be executed. For array conditions, the `where(..)` function should be used.	`if (is_array(Image)) {`
		`  view(hView,Image);`
		`} else {`
		`  close_view(hView);`
		`};`

`if` ($condition_1$)
{ $code_1$ }
`elseif` ($condition_2$)
{ $code_2$ }
`elseif` ($condition_3$)
{ $code_3$ }
…
`else` { $code_n$ } ;
	Compound if statement. The code associated with the first condition evaluating to a nonzero scalar will be executed. If all conditions produce zero values, then $code_n$ will be executed (if specified). Note that the else statement is optional.	`if (Result = 5) {`
		`  OK := false;`
		`} elseif (Result = 6) {`
		`  Test := 8;`
		`} else {`
		`  Test := 9;`
		`};`

`while` (*condition*) {*code*};	Execute the *code* repeatedly while *condition* evaluates to a nonzero value.	`i := 0;`
		`while (Image(i,10)< p) {`
		`  i += 1;`
		`};`

`for` *var* `:=` *start* `to` *end* `{`*code*`};`	Execute *code* repeatedly with *var* taking successive scalar values from *start* to *end*, incrementing by one on each successive iteration.	`for k:=0 to 255 {`
		`    LUT(k,0) := k/255;`
		`    LUT(k,1) := 1 - k/255;`
		`    LUT(k,2) := k/255;`
		`};`

`for` *var* `:=` *start* `to` *end* `step` *incr* `{`*code*`};`	Execute *code* repeatedly with *var* taking successive scalar values from *start* to *end*, incrementing by *incr* on each successive iteration. Note that *start*, *end* and *incr* may be either integer or floating point values.	`for i:=0 to 100 step 10 {`
		`    sum += v[i,0];`
		`};`

`for` *var* `:=` *array* `{`*code*`};`	Execute *code* repeatedly with *var* taking each element from *array* in turn..	`for this := val[:,3] {`
		`    m := f*log(this);`
		`};`

`for` *var* `:=` *list* `{`*code*`};`	Execute *code* repeatedly with *var* taking each element from *list* in turn..	`for this := files[:] {`
		`    det := read_image_details(this);`
		`};`

`execute(`*code*`)`	Executes the code, string or memo stored in a variable. This includes compiled code (see below).	`Code := {A := B + 1;};`
		`execute(Code);`
		`string := "cos(A)";`
		`q := execute(string);`

`try_execute(`*code*`)`	Similar to the `execute(..)` statement, except that if the *code* contains an error it does not prevent the dfc code from continuing to run. In particular, `try_execute(..)` returns a logical that indicates if the code rand without error (`true`) or not (`false`).	`Code := {A := B +/ 1};`
		`ret := try_execute(Code);`
		`# The return value (ret) will be false since Code contains an error`

`exit;`	Leave the current execution unit (*e.g.* a function or `for` loop).	
`quit;`	Terminate the current code.	
`exit_digiflow();`	Terminate DigiFlow with a zero exit code.	
`exit_digiflow(`*exitCode* `[,`*delay*`]);`	Terminate DigiFlow with the exit code *exitCode*. It is normally necessary for there to be a small delay between issuing this command and starting to terminate DigiFlow to allow the current code segment to complete. The default *delay* is 2 seconds, but may be changed with the optional *delay*.	

`compile(code[,run]);`	Compiles *code* to an intermediate level that provides approximately a factor of two improvement in performance for loops containing simple operations. The return value can be run multiple times, or the code may be executed directly by specifying the optional *run* parameter. Note that the use of array operations, rather than loops, will almost always execute very substantially faster.	`Code := {for i:=0 to x_size(P)-1 {P[i] /= I;};}; Comp := compile(Code); execute(Comp);`
`reverse_polish(code);`	Execute the specified code using dfcRP, the DigiFlow reverse polish interpreter. (The example here is directly equivalent to that used in the `compile(..)` example. Note that `compile(..)` can also be used to improve the performance of dfcRP code.	`reverse_polish({0 )P x_size( 1 - 1 \i {I ] I /= P[ });`
`in_parallel(parallel)`	On multiprocessor systems, determines whether DigiFlow should try to execute parts of its code in parallel (when possible) to improve performance.	
`is_parallel()`	Indicates whether parallel execution has been requested.	

8.9 User-defined functions

The DigiFlow interpreter accepts user-defined functions. The syntax of the definition is

`function` *func* `(a,b,...) { statements... };`

where *func* is the user-specified name of the function and *a*,*b*,… are the one or more formal arguments. The statements to be executed when the function is invoked are enclosed by the pair of braces.

By default, variables used within the function (including the formal arguments) are local to the function. If you wish to read (or write to) a variable that exists in the parent context, the name of the variable should be preceded by an exclamation mark (*e.g.* to access the variable *p* from the parent context, use `!`*p*). Note that `!` will provide access to variables in all ancestor contexts of the function (*i.e.* the variable need not be in the immediate parent). Global variables (*e.g.* `pi`) are always available for use in an expression and do not require the ancestor access prefix; any attempt to write to a global variable will throw an error.

The return value is the result of the last statement executed. To return a specific value, this need simply be the content of the last statement. Note that either simple variables (§8.2.1), compound variables (§8.2.2) or lists (§8.5) may be returned.

By default, the return value of a function (the last value of the function computed) is copied across before the local function variables are destroyed. This behaviour is not always optimal. For example, with the function

```
function UnitArrays(nx,ny) {
  a.x := x_index(nx,ny)/(nx-1);
  a.y := y_index(nx,ny)/(ny-1);
  a;
};
```

a copy of the compound variable `a` has to be made. This requires time and increases the peak memory requirements. As an alternative, the return value can be specified as `@`*name*, as in the modified example

```
function FastUnitArrays(nx,ny) {
  a.x := x_index(nx,ny)/(nx-1);
  a.y := y_index(nx,ny)/(ny-1);
  @a;
};
```

In this case the contents of the variable `a` is taken over by the return value of the function, improving performance. Note, however, that the syntax `@`*name* will not work if the return value is itself an expression, a number or a string.

The return value of a function need not be used by the calling code. Invoking a function without an assignment statement simply executes the function and discards any value returned.

In the function declaration it is possible to specify default values for the formal arguments, thus making their specification optional in the call to a function. For example, in

```
function RescaleImage(im,Scale:=1,Black:=0) {
  Scale*(im-Black);
};
```

The `im` parameter is mandatory, while both `Scale` and `Black` are optional. If `Scale` is not specified, then it takes the value `1`. Similarly, if `Black` is not specified, the value defaults to `0`. The function may then be called in one of the following ways:

```
Q := RescaleImage(P);
Q := RescaleImage(P,2);
Q := RescaleImage(P,Scale:=2);
Q := RescaleImage(P,2,0.03);
Q := RescaleImage(P,Scale:=2,Black:=0.03);
Q := RescaleImage(P,Black:=0.03);
```

When a parameter is not specified, the default value is that given by the corresponding assignment statement within the parameter list in the function declaration. Thus the above function calls are equivalent to

```
Q := RescaleImage(P,1,0);
Q := RescaleImage(P,2,0);
Q := RescaleImage(P,Scale:=2,0);
Q := RescaleImage(P,2,0.03);
Q := RescaleImage(P,Scale:=2,Black:=0.03);
Q := RescaleImage(P,1,Black:=0.03);
```

In most circumstances, the last of these in its original form, *i.e.*

```
Q := RescaleImage(P,Black:=0.03);
```

should be avoided: you should only exclude parameters from the end of the list. As an alternative, a **`null`** could be specified for `Scale`, with the declared function resolving the appropriate default in that case. In particular,

```
function RescaleImage(im,Scale:=1,Black:=0) {
  if (is_null(Scale)) then {Scale := 1};
  Scale*(im-Black);
};
```

then allowing a call of the form

```
Q := RescaleImage(P,null,Black:=0.03);
```

By default, variables, constants and expressions are passed to functions by value. This means that the formal parameter is treated as a local variable within the function and any changes you may make to it are not reflected in the value of the actual parameter in the code calling the function. In this model, the only way of returning information to the calling code is through the return value.

DigiFlow also supports a mechanism for passing arrays and compound variables by reference. Although problems can arise using this mechanism, it can greatly improve the efficiency of functions by reducing the amount of copying the interpreter must do.

To invoke passing a variable by reference, the names of the formal arguments must be prefaced by an @ character in the formal parameter list. For example, the code segment

```
function Try(@test) {
  test[1] := 2;
};

This := [0 0];
Try(This);
```

completes with `This` equal to `[0 2]`.

While this can speed up execution, especially if passing large arrays, the precise behaviour is complex. It is therefore recommended that you obey the following guidelines when using an @*name*:

◊ If @*name* refers to an array or list, it should only appear on the right-hand side of an assignment statement, although array elements or subarrays can be on the left-hand side of an assignment (as in the above example). Thus

```
function Assign(@array) {
array := 0;
};
```

is not acceptable whereas

```
function Assign(@array) {
array[:] := 0;
};
```

will behave predictably.

◊ If @*name* refers to a simple scalar (integer, floating point or string), it will always be passed by value.

◊ If @*name* refers to a compound variable, it may appear on either the left- or right-hand side of an assignment statement. If it appears on the left-hand side, then the assignment must be to only one of the component variables. In this case, the modified component will be passed back to the calling routine.

As the components of compound variables may be of any type without affecting the above guidelines, it is recommended that compound variables be used to improve execution speed where appropriate. In general, however, it is better to write pure functions that only return information via their return value. This return value, of course, may be any type of value, thus allowing full flexibility.

As with other DigiFlow functions, user-defined functions may be used with or without keywords.

8.10 User input and ouput

The interpreter supports a variety of functions for interacting with the user during execution. These include the input functions **ask_string**(..), **ask_list**(..), **ask_integer**(..), **ask_real**(..), **ask_yesno**(..), **ask_image**(..), **ask_file**(..),

ask_directory(..) and ask_printer(..). These functions are all modal (*i.e.*, they take the focus away from the rest of DigiFlow and you will not be able to do anything else until you have closed the associated dialog). However, most also have modeless equivalents that allow you to continue working on other things before dealing with the dialog. In particular, ask_yesno_modeless(..), ask_integer_modeless(..), ask_real_modeless(..), ask_string_modeless(..), ask_list_modeless(..), ask_image_modeless(..), ask_file_modeless(..), ask_directory_modeless(..), and message_modeless(..).

Alongside these are the mouse input functions get_mouse_click(..), get_mouse_line(..), get_mouse_rect(..), and get_mouse_box(..) with get_mouse_position(..) detailing the current location of the mouse. (The functions mouse_get_mode(..) and mouse_set_mode(..) can be used to determine or set whether the mouse is acting normally, scrolling/panning, measuring, *etc.*) At a more basic level, get_key(..) can be used to determine the state of a key on the keyboard.

User output is provided through message(..), beep(..) and status_bar_message(..).

A different approach to user input and output is through opening a console. This can be opened with open_console(..). Input and output via the console can then be achieved using read_console(..) and write_console(..). When finished, the console may be closed using close_console(..). Note that the functions open_file(..), write_file(..) and close_file(..) may also be used with a console.

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow. See also §9.7 on file handling.

8.11 Input of code from files

In many cases it is desirable to be able to store interpreter code in a file for later use. DigiFlow supports the use of such code through the include(*s*) command. Here, *s* represents a string variable or string constant specifying the file name. Upon execution, include(*s*) is replaced by the contents of the file named by *s*. This file may contain one or more statements, function definitions, *etc.*, and may be called either as a statement (with no return value), or as a function within an expression. Note that, unlike a normal user-defined function, variables used and declared within the file have the same scope as though they were included explicitly within the parent code. In this way the include(*s*) statement is similar to (but more flexible than) the 'include' directive used in many programming languages.

If the string *s* does not specify a file extension, then .dfc will be assumed. Moreover, if the file is not found in the current working directory, the dfc search path (see get_dfc_path(..) and set_dfc_path(..)) will be used. If the file is still not found, then the DigiFlow program directory will be tried.

To improve execution speed, the file specified by include(*s*) is read in only once within a given process, and stored for any subsequent use. This behaviour reduces the need to define functions in static code expressions to handle dynamic data streams.

The function get_file_variables(..) will execute the dfc code in a file in a similar way to include(..), but will return all the variables created in executing as a compound variable. The file specified in calling this function is executed as though embedded in a user defined function (*i.e.* it will need to make use of the ! global specifier to access variables from the calling context).

If you wish to include and execute code that might fail or might contain errors, then try_include(..) will include and execute the code, but return a logical variable that indicates if the code ran successfully (true) or failed (false).

8.12 Debugging

DigiFlow provides a number of useful debugging tools for identifying problems in user-supplied code. These tools include retrospective error handling, output messages, and interrogation of the variables defined at a given time.

8.12.1 Error handling

Inevitably there will be times when user dfc code encounters a problem. Although it may not always succeed, DigiFlow will attempt to identify this problem and terminate the process in a clean manner. In doing so, it will produce the diagnostic dialog shown in figure 143.

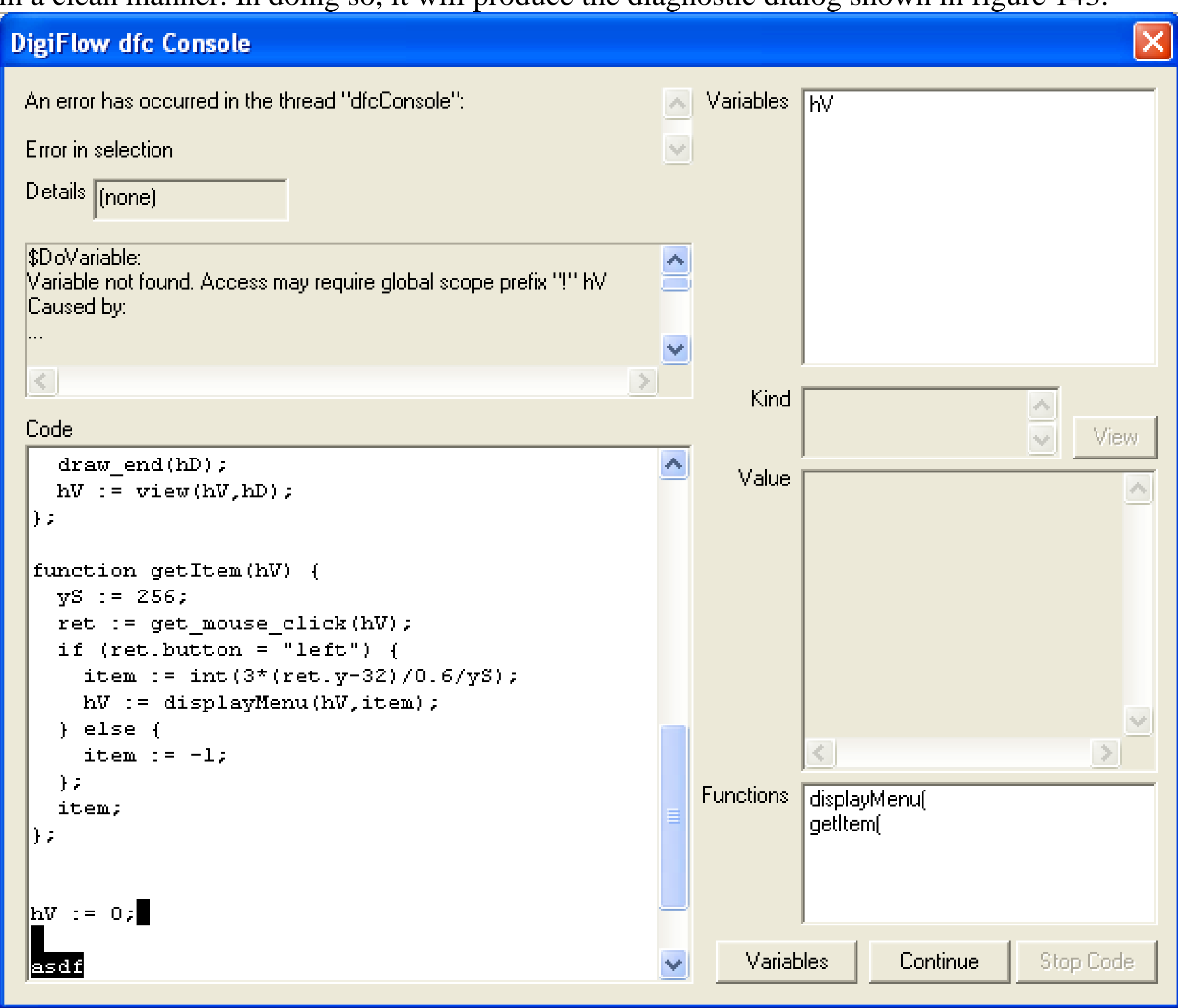


Figure 143: The *dfc* error diagnostics dialog.

The basic error message is displayed in the two top edit boxes of the dialog, while the code causing the problem is displayed in the Code box. The particular statement at fault (or one very close to it) is left highlighted in this box.

The Variables list gives information about all the variables defined at the time of the error occurring. The type contents of each variable are given in the Kind and Value boxes, respectively. For arrays and drawings, the View button may be used to give a graphical representation. If you require more detailed access to the variables, click the Variables button at the bottom of the dialog. This will start up the View Variables dialog (see §8.12.2). Finally, the Functions box lists any user-defined functions in the current context.

8.12.2 View variables

When debugging code it is often useful to interrogate the contents of all variables at a given point in the calculation. This may be achieved by a call to the `view_variables(..)` function. Calling this function from within a *dfc* code segment will produce the dialog shown in figure 144.

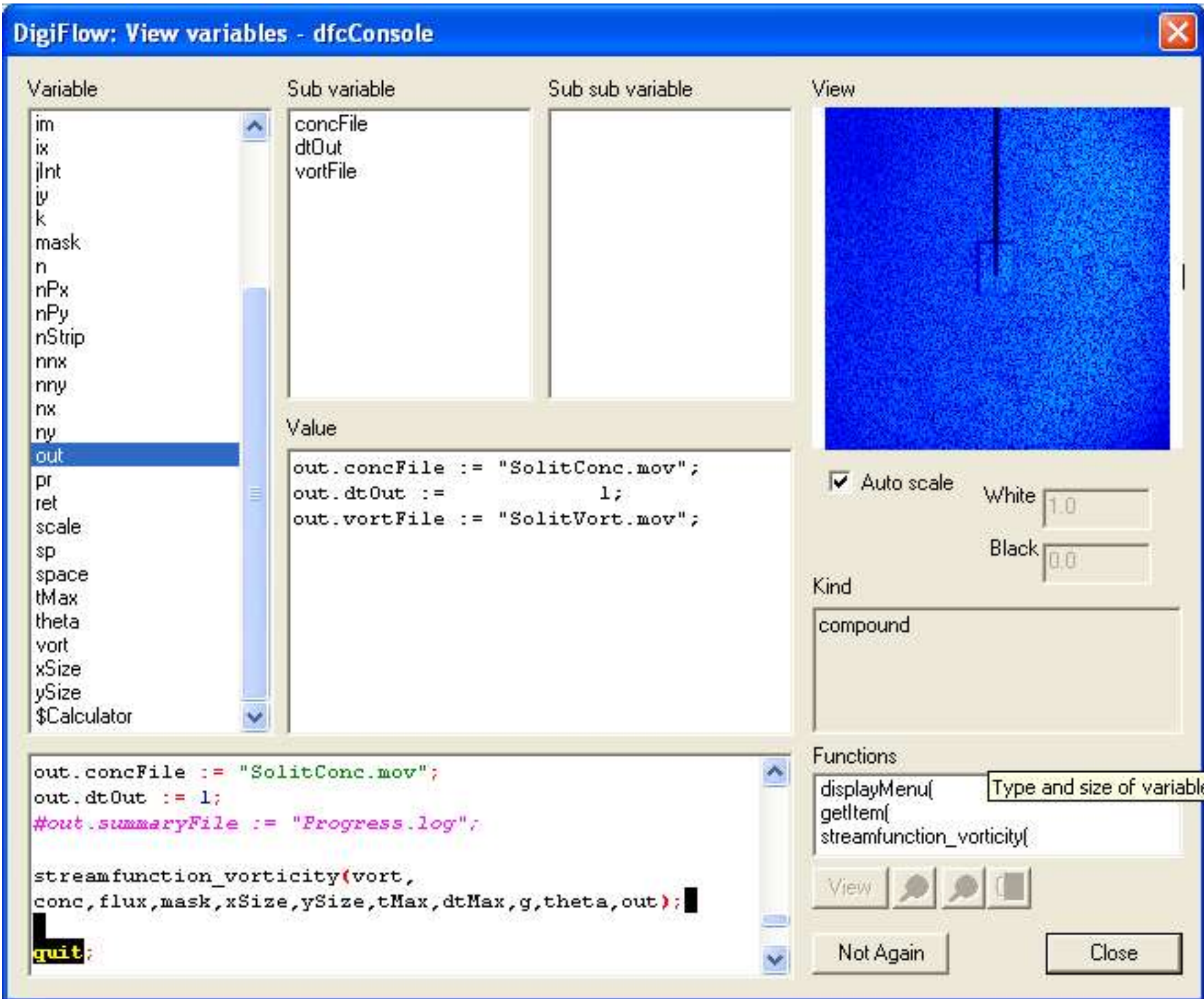


Figure 144: The `view_variables(..)` dialog.

Selecting a variable from the Variable list will display its contents in the Value box. If the variable is a compound variable, then the names of its components will be displayed in the Sub variable list. Selecting a name from the Sub variable list will refine the Value displayed. Similarly, if the selected Sub variable is itself a compound variable, the components will be displayed in the Sub sub variable list.

If the variable selected is an array or a drawing object, then a graphical representation of it will be displayed in the View box. To obtain a larger version, click on the View button. For an array, the scaling of the image is controlled using Auto scale in conjunction with Black and White. The View button will produce an enlarged version of the image in its own window, whilst , and zoom in, zoom out and change the colour scheme, respectively.

Note that variables beginning with a dollar (`$`) symbol are system variables and are not available for direct use by the user. In the example shown in figure 144 these system variables contain the root copy of the drawing identified by the user variable `hDraw`.

The `view_variables(..)` function has one optional logical parameter: if `true` (the default), then the dialog will be displayed. If `false`, then the dialog will not be displayed.

Related to this is the return value of the function: `true` if Close is clicked, or `false` if Not Again is pressed. This provides a convenient method of switching off the `view_variables(..)` output, as illustrated in the following code segment:

```
debug := true;
for i:=0 to 100 {
  ### Statements
  debug := view_variables(debug);
};
```

Here, the variable `debug` is initially set to `true`, thus enabling `view_variables(..)`. This state will continue until Not Again is clicked, effectively setting `debug` to `false`. Of course, it is possible for the code to subsequently set `debug` back to `true` and thus turn the viewing of variables on again.

8.12.3 Messages

The most basic approach to debugging is to write out information to the user/developer as execution of code proceeds. DigiFlow provides two main mechanisms for this: the `message(..)` function, and writing out to a console. While the `message(..)` function provides the simplest route, it can be annoying to the user to have to respond to each and every message produced from within a loop. In contrast, by using the `open_console(..)` and `write_console(..)` functions (or the equivalent `open_file(..)` and `write_file(..)` implementations) a continuous stream of data will be written to a console window. For example, the code

```
hFile := open_file();
x := make_array(0,128);
x := x_index(x);
for k:=0 to x_size(x)-1 {
  write_file(hFile,"Now at",k,x[k]);
};
```

produces the console window shown in figure 145.

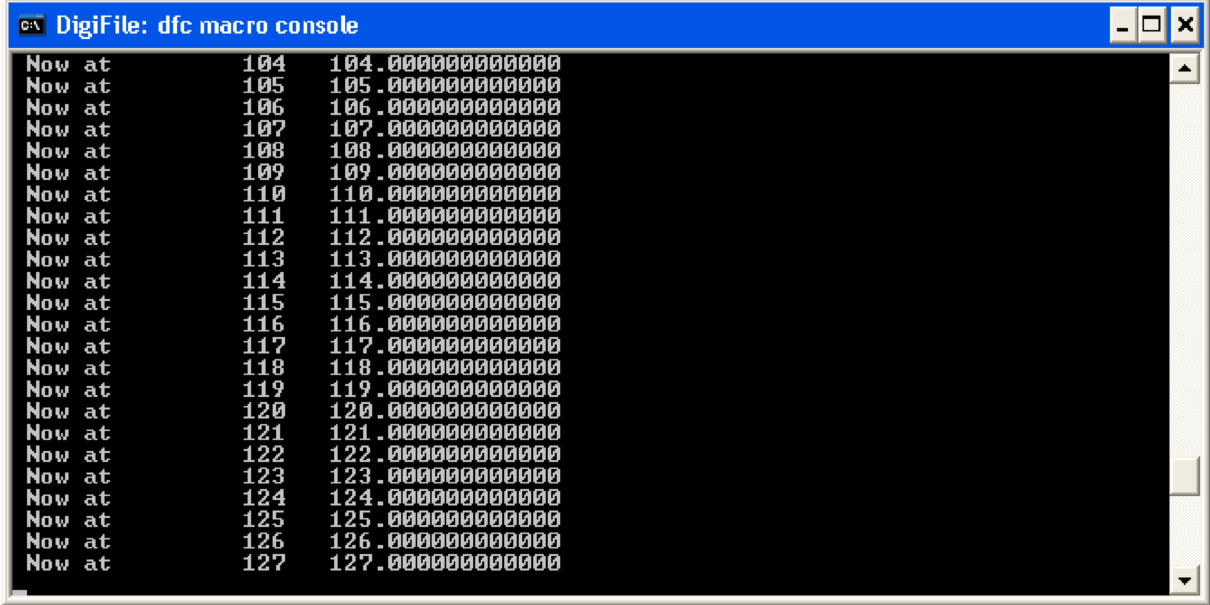


Figure 145: Example of output to console window.

For short messages, it may prove convenient to write them to the status bar with `status_bar_message(..)`. This technique can work well for something running in the background, but it can be confusing if more than one macro is trying to do the same thing!

Another option, which has some advantages but tends to be a bit more cumbersome, is to write text either to the window (view) title using view_title(..), or to write the text to its own view window. Both these possibilities are illustrated in the following code:

```
hV := new_view(256,64);
view_title(hV,"View to contain text messages");
view(hV,"This is an example of text to a view.");
```

The output produced by this is shown in figure 146. Note that unlike the console, a second view(..) statement specifying text replaces the original text rather than appending it to the view.

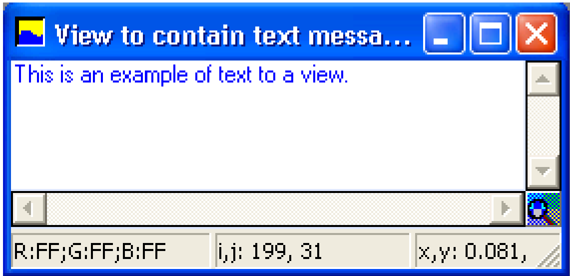


Figure 146: Example of text written to a view.

8.12.4 Queries

The ability to determine the intermediate results within a statement without affecting the result of that statement may sometimes be useful while debugging. DigiFlow supports this through the query operator ?. This operator has no effect on the sequence of execution, but simply causes the result of the statement or sub-statement immediately following it to be displayed in some manner. The following illustrates by way of example the effect of the query operator.

Example	**Result**
`?5 + 3`	5
`?(5 + 3)`	8
`?a := 5 + 3;`	8
`sqrt(?16)`	16
`?sqrt(16)`	4
`?img[0,0]`	The contents of the first element of the array.
`?img`	The whole of the img array.

By default, query operators are ignored in standard interpreter contexts. That is, they have no effect on the code and produce no output. To turn on the output, simply call allow_query(). In the standard context, this will then generate a message box for each query as it is processed. Query output may be turned off again by allow_query(false). When running in the dfcConsole (see §8.12.7), the output of queries is written to a dedicated window.

8.12.5 Break points

Another valuable debugging tool in DigiFlow is the provision of break points which allow monitoring of code execution without otherwise affecting that execution. A break point is specified by the ampersand character &, and may inserted into any point in the code. The interpreter's response to the break point depends on the environment in which DigiFlow is running.

In a standard interpreter context, breaks will have no effect unless first enabled by a call to allow_break(..). If called in a standard interpreter context, then will invoke the view_variables(..) function. Clicking Not Again to exit view_variables(..) will suppress the action of further break points unless another call to allow_break(..) is made.

If called in the dfcConsole, then execution will stop and the `view_variables(..)` functionality is again available. The execution or otherwise of breaks is controlled by a check box.

8.12.6 Tracing execution

Sometimes, the best way of locating a bug is to keep track of exactly where execution is taking place in the code. The function `trace(..)` turns on a facility to do this. Tracing can be provided either to a specified file, or to a console window (which is opened automatically). The trace will write to the file (or console window) each statement as it is executed. This logging continues for the period of time specified in the call to `trace(..)` (the default is 10 s). Note, however, that `trace(..)` is only available for dfc code executed through the dfcConsole.

8.12.7 dfcConsole

The dfcConsole, described in §5.2.10, provides a powerful interactive tool for both editing and debugging dfc code.

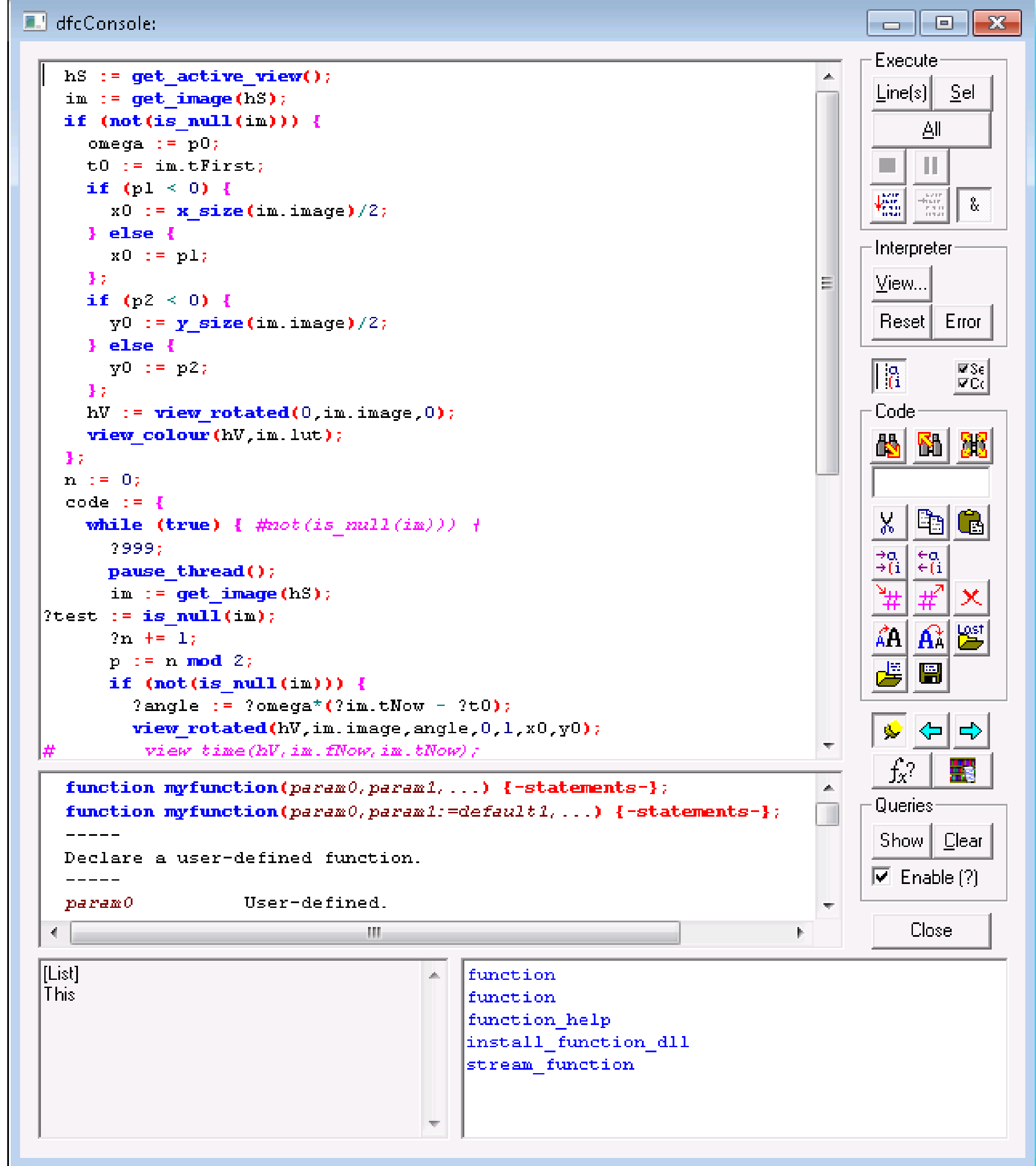


Figure 147: The dfcConsole for writing and debugging dfc code.

This resizable window contains an edit control allowing interactive editing of the dfc code to be run, alongside a series of controls allowing control over the execution environment and providing timely information.

The Execute group may be used to selectively execute code. If there is no text selected, then Line(s) will execute the current line. If there is an active selection, then Selection will execute the selected code, and Line(s) will execute not only the selected text, but all the lines on which some text is selected. Regardless of the selection, All will cause the entire code to be executed. Note that `<alt><enter>` is equivalent to clicking Line(s).

All the control buttons are disabled while the code is executing with the exception of Stop in the Execute group. Clicking the stop button ■ will abort the currently executing code, while the pause button ❚❚ will temporarily suspend execution. Checking Breaks (&) causes

break points, indicated by an ampersand in the code (see §8.12.5) to be executed as and when they aroe found by the interpreter. If cleared, then the break points are ignored. Note that the status of the Breaks (&) dialog may be changed by the user as the dfc program runs.

The Interpreter group controls the internal state of the DigiFlow interpreter. Reset will clear all variables and functions from the interpreter, while View displays the variables and objects defined within the interpreter using the **view_variables**(..) interface. If an error occurs, then Last Error will redisplay the last error message.

The Code group controls the action of the edit control containing the code.

For further details, refer to §5.2.10.

9 Functions

This section describes the more advanced functions available within DigiFlow. Like the functions described in §8.9, these functions can be called with or without key words. For example,

```
view(hPic,Image,0.0,1.0);
```

will display on the window identified by `hPic` the array `Image`, taking a value 0.0 to represent "black" and 1.0 to represent "white". The same command may be written more clearly as

```
view(hView:=hPic,array:=Image,black:=0.0,white:=1.0);
```

or with its arguments in a different order as

```
view(array:=Image, hView:=hPic,white:=1.0,black:=0.0);
```

Note, however, that the third of these options (*i.e.* the arguments not in their natural order) can incur a significant computational overhead, and is therefore discouraged except in circumstances where the reordering improves readability.

Similarly, many of the functions can accept arguments with a range of different types, and may have optional arguments. For example,

```
view(hView:=hPic,array:=Image);
```

will have the same effect as the earlier example, except that the black and white levels are not specified by the user (the default values are in fact 0.0 and 1.0, respectively). In contrast,

```
view(hView:=hPic,hDraw:=myDrawing);
```

will view a drawing previously created by the drawing routines described in §11. DigiFlow determines the action to be taken by the type of data it is provided with, hence

```
view(hPic,Image);
```

and

```
view(hPic,myDrawing);
```

would perform the same action as their counterparts with key words. Using the key words, however, improves the clarity of the resulting dfc file by underlining the role played by each of the arguments.

DigiFlow has a vast array of predefined functions. Full details of all of these are available via the interactive help system found under Help: dfc Functions… and at the fx? button of dialogs that make use of dfc code.

The following subsections give an overview of the functions available, but do not provide a complete list. In all cases the name of the function is self-explanatory, although of course the parameters and return value may need some explanation.

DigiFlow functions may be used with or without key words. If key words are given, then the order of the arguments does not matter. However, if keywords are not given, the arguments must be in the order stated here. For example,

```
This := where(Image>0.5,1.0,0.0);
```

will set `This` to an array of zeroes and ones, depending on whether the array `Image` is greater than or less than 0.5. The same command may be written as

```
This := where(mask:=(Image>0.5),vTrue:=1.0,vFalse:=0.0);
```

or with its arguments in a different order as

```
This := where(vFalse:=0.0,vTrue:=1.0,mask:=(Image>0.5));
```

Note, however, that the third of these options (*i.e.* the arguments not in their natural order) can incur a significant computational overhead, and is therefore discouraged except in circumstances where the reordering improves readability.

A complete list of all functions known to DigiFlow at time of writing is given in §9.32.

9.1 Basic mathematical functions

DigiFlow supports a full range of basic mathematical functions. Some of these have more than one variant. For example, `sin(..)` returns the sine of an angle specified in degrees, while `sin_rad(..)` expects the angle to be in radians. Similarly for `cos(..)`, `tan(..)`, and their inverses `asin(..)`, `acos(..)` and `atan(..)`. These are in turn supported by `degrees_from_radians(..)` and `radians_from_degrees(..)`.

Both natural and base ten logarithms are supported through `ln(..)` and `log(..)`, respectively, with the former's inverse `exp(..)`.

Other basic functions include `abs(..)`, `sign(..)`, `sqrt(..)`, `int(..)`, `real(..)` and `not(..)`.

Additional transcendental functions include `bessel(..)`, `erf(..)` and `erfc(..)`.

`round(..)`, `floor(..)`

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.2 String functions

The string functions bear some resemblance to those found in some dialects of Basic. The list of possible functions include `upper_case(..)`, `lower_case(..)`, `length(..)`, `search_string(..)`, `left_string(..)`, `right_string(..)` and `mid_string(..)`.The maximum length of a standard string is given by `max_string_length(..)`; the functions noted here can be applied to memos when longer strings are required.

Other string manipulation functions include `remove_spaces(..)` and `scrunch_string(..)` for creating shorter strings with undesirable characters removed, while `replace_hashes(..)` allows hash characters (`"#"`) in a string to be replaced by a number (useful when constructing file and directory names). Values complying to dfc syntax may be retrieved from a string using `read_this(..)`, or dfc code in a string may be executed with `execute(..)` or `try_execute(..)`.

The command line used to start DigiFlow is available through `command_line_arguments(..)`.

There are also a set of more specialist string functions for converting numeric data to strings. The simplest of these is `make_string(..)`, which provides Fortran-like control of the string formatting, while `remove_trailing_zeros(..)` can help clean up a string.. More specialist functions include `fit_as_text(..)`, which converts the result of a least squares fit into a LaTeX-compatible attractive text string, and `nice_number_string(..)` that creates a LaTeX-compatible attractive version of a number.

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.3 Array functions

To supplement standard array access using syntax of the form *var*`[`*i0*`:`*i1*`:`*is*`]`, and the constructor (`make_array(..)`, `make_like(..)`) and conditional (`where(..)`) functions, DigiFlow provides other methods of accessing arrays such as `extract(..)`, `indirect(..)`, `sample_values(..)` and `look_up_table(..)`. Creating an array containing a row or column repeated is achieved through `x_replicate(..)` and `y_replicate(..)`. Specialist array

constructors include x_index(..), y_index(..), z_index(..), t_index(..), identity_matrix(..), random_array(..) and gaussian_array(..).

Matrix support is provided through the likes of transpose(..), matrix_multiply(..), least_squares(..), eigen_system(..), eigen_values(..), solve_linear(..), solve_svd(..) and singular_value_decomposition(..). Other basic manipulation functions include roll(..), flip_horizontal(..), flip_vertical(..), rotate_clockwise(..) and rotate_anticlockwise(..).

In addition to the standard arithmetic operators (that function on and between arrays in the obvious way), there are a special set of functions that provide the possibility of combining arrays of different dimensions. The cropped_ set of functions (cropped_add(..), cropped_sub(..), cropped_sub_reversed(..), cropped_mul(..), cropped_div(..), cropped_div_reversed(..), cropped_power(..), cropped_power_reversed(..) and cropped_assign(..)) perform the indicated operation only on the portion of the arrays that overlap/intersect. Simlarly, the wrapped_ set of functions (wrapped_add(..), wrapped_sub(..), wrapped_sub_reversed(..), wrapped_mul(..), wrapped_div(..), wrapped_div_reversed(..), wrapped_power(..), wrapped_power_reversed(..), wrapped_assign(..) and wrapped_extract(..)) perform the indicated operation by wrapping the larger array around the smaller array.

Other array-specific functions include sort_array(..), which is an array-specific version of sort(..)

9.4 Type manipulation functions

This group of functions has the ability to manipulate the types and sizes of values.

The make_array(..) and make_like(..) functions are particularly valuable for constructing and reshaping arrays, with make_list(..) playing the same role for lists. The where(..) function provides a convenient method of conditionally accepting values (a little like an if statement for arrays), while make_string(..) provides a way of converting numeric data to a string using a particular format. At a more primitive level, char(..) and ascii(..) work on a single character basis. A read_data(..) statement in conjunction with an end_data statement provides a convenient method of entering arrays of data in-line in place of the normal [*value0, value1, ...*] syntax. (A related approach is used for the drawing functions draw_begin_lineto(..), draw_begin_mark(..) and draw_begin_vector(..)).

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.5 Information functions

This group of functions returns structural information about the value containing in a variable or expression. These include the type tests is_integer(..), is_real(..), is_numeric(..), is_string(..), is_array(..), is_list(..), is_compound(..), is_code(..),is_memo(..) and is_null(..). Other special information functions include is_drawing(..), is_view(..), is_live_view(..) and is_running(..) to determine information out specific objects. The function exists(..) determines whether a variable of a specified name exists, while the functions x_size(..), y_size(..), z_size(..) and n_size(..) return size information on an array. For compound variables, n_components(..) returns the number of subvariables contained.

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.6 Variable functions

In addition to being able to manipulate variables through the assignment statement, DigiFlow provides `set_variable(..)`, `get_variable(..)` and `get_component(..)` to create or retrieve variable values based on string arguments. While the first two of these work directly on the dfc interpreter's current context, the third recovers information from a compound variable. The function `component_names(..)` can be used to recover a list of names of components in a compound value, while `exists(..)` can be used to determine if a given variable exists. (Note: `exists(..)` should not be used in code to be compiled with `compile(..)`.) The functions `set_global(..)` and `get_global(..)` manipulate the global (root) interpreter context, and should be used only with great care, as should the functions `set_configuration(..)` and `get_configuration(..)`.

The functions `get_user_variables(..)`, `get_local_variables(..)` and `get_file_variables(..)` can provide a snapshot (as a compound value) of the current state of variables in the interpreter, while `list_local_variables(..)`, `list_user_variables(..)`, `list_global_variables(..)`, `list_system_variables(..)` and `list_components(..)` recover the names of all the relevant variables.

9.7 File handling

DigiFlow provides a variety of standard file functions for handling input and output from a dfc file. As is common with many languages, a file handle is provided by opening the file, and this handle must be used for all subsequent access to the file. The file will be closed either when the `close_file(..)` command is executed or the file becomes out of scope. A file becomes out of scope when the execution unit it was opened in is terminated. For example, a file opened within a function will be closed automatically when that function terminates.

Files are opened by `open_file(..)`. A console window may be opened either with `open_file(..)` or with `open_console(..)`. The handle returned by `open_file(..)` is then passed to the other file manipulation functions `write_file(..)`, `write_array(..)`, `read_file(..)`, `read_line(..)`, `read_array(..)`, `flush_file(..)` and `close_file(..)` (for a console, `write_console(..)`, `read_console(..)` and `close_console(..)` may be used instead).

Information about files may be obtained with `file_details(..)`, `is_file_local(..)` and `computer_for_file(..)`, while files may be copied using `copy_file(..)` or `copy_file_wait(..)`, moved with `move_file(..)`, and deleted using `delete_file(..)`. The current directory may be determined with `current_directory(..)`, and changed by `change_directory(..)`, while new directories are created with `create_directory(..)` or removed by `destroy_directory(..)`. Note that DigiFlow retains a current directory separately for each dfc code being run, as well as a separate current directory for the main DigiFlow process. The DigiFlow directory structure may be probed using `start_directory(..)` and `digiflow_directory(..)` with the variants `start_directory_url(..)` and `digiflow_directory_url(..)` providing an alternative view of the path for networked drives. The name of the DigiFlow executable itself can be determined from `digiflow_executable(..)`.

One of the simplest ways of reading data from a file is with `read_array(..)`, although the functions `read_into_array(..)` and `read_table(..)` provide an alternative when only partial data exist (`read_into_array(..)`) or the data is of mixed types (`read_table(..)`).

Support for binary files is provided through `open_binary_file(..)`, `read_binary(..)`, `write_binary(..)`, `set_file_pointer(..)` and `set_file_end(..)`. A binary file may be closed using `close_file(..)` as normal.

Support for files already in the file system includes `copy_file(..)`, `copy_file_wait(..)`, `move_file(..)` and `delete_file(..)`. Information about a file is obtained with `file_details(..)`, while `list_files(..)` provides access to the contents of directories, with greater detail available through `list_file_details(..)`. Unless an absolute path specification is given, the last two functions search relative to the current path for the dfc macro (see `change_directory(..)`). The same can be achieved relative to the global DigiFlow path using `list_files_global(..)` and `list_file_details_global(..)` instead.

The function `wait_for_file(..)` provides an efficient way to wait until a specific file exists.

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.8 Reading and writing images

In addition to accessing images through the built-in menu options, DigiFlow command files may read and write images directly. Note that this mechanism bypasses the normal file handling outlined in §9.2.

Functions that support the reading of images include `read_image(..)`, `read_image_when_ready(..)`, `read_image_from_view(..)` and `read_image_details(..)`. In some circumstances, asynchronous reading of images can substantially speed up the processing. This can be achieved by using `read_image_queue_create(..)` to set up the queue, `read_image_queue(..)` to recover the next image from the queue, and `read_image_queue_destroy(..)` to close the queue.

The function `get_process_details(..)` recovers what is known about how an image was created in a form compatible with the `process` and `dialog` commands.

Conversely, the saving of images is achieved through `write_image(..)` and `save_view(..)`. For a full colour image, `write_rgb_image(..)` provides a simplified interface. Rather than the direct write provided by `write_image(..)`, images can be queued for asynchronous writing using `write_image_queue(..)` and `write_rgb_image_queue(..)`. An indication of the number of images currently queued is obtained with `n_waiting_write_image(..)` and the wait function `wait_for_write_image_queue_empty(..)`.

Custom image readers may be constructed using dfc code and installed in DigiFlow using `add_image_reader_macro(..)`.

MetaFile support is provided through `draw_on_emf(..)` as well as the standard `write_image(..)`.

Printer support is provided through `ask_printer(..)`, `print_view(..)` and `print_view_dialog(..)` while Encapsulated PostScript generation support includes `export_to_eps(..)` and `export_to_simple_eps(..)`.

Turning on and off the reading and writing of DigiFlow image archives (.dfa files) is controlled by `read_image_archive(..)` and `write_image_archive(..)`.

Some image formats support additional functionality, such as `jpeg_get_comments(..)`. The function `add_movie_reader(..)` provides the ability the add additional movie formats to DigiFlow.

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.9 Windows and views

The image processing and plotting features of the DigiFlow interpreter are enhanced by its ability to handle windows. In the terminology used in DigiFlow, a window containing an image or other graphical object is referred to as a *view*.

Basic handling of views is achieved through new_view(..) and close_view(..). Once created by new_view(..), the handle returned by this function is used to identify the view to be operated on. The function view(..) has a large number of variants for displaying a diverse range of data in an existing view, while view_rotated(..) adds to it the possibility of rotating the display. Alternatively, images may be loaded directly into a new view using open_image(..) or open_image_when_ready(..). When a view contains a selector specifying multiple images, animate_view(..) may be used to control the replaying of the sequence.

Views with slightly different characteristics may also be created using new_view_clean(..) and new_view_floating(..), while the connection and disconnection of a window to an execution thread can be controlled through view_connect_thread(..) and view_disconnect_thread(..), respectively.

Icons may be superimposed on a view and manipulated with view_icon(..) and move_icon(..).

The currently active view may be identified using get_active_view(..), and its contents retrieved as an image using get_view_as_image(..). Information about the type of view can be recovered with get_view_class(..), while information about a view-specific coordinate system can be read or set using view_get_coord_system(..) and view_set_coord_system(..).

Associated data for a view may be set with view_title(..) and view_time(..), while the size and arrangement of views may be controlled using view_zoom(..), view_zoom_all(..), view_zoom_to_fit(..), view_zoom_all_to_fit(..), view_fit_to_zoom(..), view_fit_all_to_zoom(..), tile_views(..), cascade_views(..), maximise_view(..), minimise_view(..) and restore_view(..). The function close_all_views(..) can be used to close all views (or all views not currently involved in a process), and view_get_time(..) can recover timing information set either by view_time(..) or as part of a DigiFlow process.

The appearance of a view may be controlled through view_colour(..), with the associated colour schemes manipulated using colour_scheme(..), colour_scheme_from_image(..), add_colour_scheme(..) and delete_colour_scheme(..). A false colour scheme may be toggled to greyscale using view_toggle_colour(..). Other functions affecting the appearance in a view of a vector field include view_vector_colour(..), view_vector_scale(..), view_vector_spacing(..), view_scalar_colour(..) and view_scalar_range(..).

Specialised slave views are created and controlled using slave_view_3d(..), while plots may be rendered in 3d using view_3d(..), render_3d(..), view_points_3d(..) or render_points_3d(..).

Details from a view may be sent to a printer of PostScript file through print_view(..), print_view_dialog(..) and export_to_eps(..).

Arrangement of the main DigiFlow window is achieved through maximise_digiflow(..), minimise_digiflow(..) and restore_digiflow(..).

Icon-like images may be placed on, moved around and deleted from views using view_icon(..), move_icon(..) and remove_icon(..), while view_counter(..) returns a value that indicates if the contents of the view has changed.

For a more complete list, and further details on these functions, refer to the *.dfc* function help facility within DigiFlow.

9.10 Timing functions

This group of functions returns timing information. The group includes **time**(..), **date**(..) and **process_time**(..). Functions for generating delays include **start_timer**(..), **wait_for_timer**(..) and **sleep_for**(..). For high-precision timing, use **time_interval**(..).

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.11 Statistical functions

This group of functions returns statistical information. Most of these functions have three variants: one that returns scalar statistics for the entire array, one that returns an array of statistics collected along the x direction, and a third that returns an array of statistics collected in the y direction. Examples of entire-array statistics include **count**(..), **sum**(..), **mean**(..), **rms**(..), **max_value**(..), **min_value**(..). The corresponding x direction statistic functions are **x_count**(..), **x_sum**(..), **x_mean**(..), **x_rms**(..), **x_max_value**(..), **x_min_value**(..) and the y direction statistic functions are **y_count**(..), **y_sum**(..), **y_mean**(..), **y_rms**(..), **y_max_value**(..), **y_min_value**(..). A subset of the radial (**r_count**(..), **r_mean**(..), **r_sum**(..)) and azimuthal (**theta_mean**(..)) equivalents are also provided.

The moment functions **x_moment**(..), **y_moment**(..), **x_centroid**(..) and **y_centroid**(..) all return arrays of data. The location functions **max_index_x**(..), **max_index_y**(..), **min_index_x**(..) and **min_index_y** apply to the entire array, while **x_max_index**(..), **y_max_index**(..), **x_min_index**(..) and **y_min_index**(..) return arrays. In a similar vein, the function **where_is**(..) determines the indices where a particular condition is satisfied.

Random number support is available through both scalar **random_number**(..) and array **random_array**(..) entry points, while **randomise**(..) will reorder an array or list, and **gaussian_array**(..) provides normally distributed random numbers.

The **histogram**(..) function allows binning of information, while **sample_values**(..) provides an efficient mechanism for extracting data from predetermined locations in an array. The functions **x_accumulate**(..) and **y_accumulate**(..) accumulate the contents of arrays, effectively integrating them in one direction from one edge.

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.12 Image processing functions

Image processing functions allow basic manipulations of array values in a manner that is of use for image processing operations. The functions **transpose**(..), **flip_horizontal**(..), **flip_vertical**(..), **rotate_clockwise**(..) and **rotate_anticlockwise**(..) can be used to re-orient an image, while **rotate_image**(..) allows for more general rotations. Images may be translated using **shift**(..) or **shift_interpolated**(..), and their resolution changed using **rescale_image**(..) or the more sophisticated weighted **interpolate_image**(..).

Basic filter operations include **filter_low_pass**(..), **filter_convolution**(..), **filter_min**(..), **filter_max**(..), **filter_median**(..), **filter_centile**(..),

filter_std_dev(..) and filter_geometric(..). These may be extended further using the Fast Fourier Transform function fft_2d(..) and its inverse inverse_fft_2d(..).

The basic filters assume the edges of the image are independent. However, sometimes it is more appropriate to consider the image as one period of a periodic array. Support for this view is provided through filter_periodic_centile(..), filter_periodic_convolution(..), filter_periodic_geometric(..), filter_periodic_low_pass(..), filter_periodic_max(..), filter_periodic_median(..), filter_periodic_min(..) and filter_periodic_std_dev(..).

Variants on filter_min(..) and filter_max(..), that exclude the pixel itself, are available through filter_min_neighbours(..) and filter_max_neighbours(..). These can be particularly valuable for identifying turning points in an image.

Further information about the structure of an image is available via contouring with contour_image(..), find_contour_start(..), pixel_contour(..) and smooth_contour(..). A contour may be resampled with resample_curve(..). A related function, find_edge(..), uses gradient information to identify the edge of a region. Alongside these, find_blobs(..) can determine the properties of islands satisfying an intensity threshold, and fill_blobs(..) and fill_blob_list(..) can flood-fill such entities. The function fractal_box_count(..) provides access to the Kolmogorov capacity, while the related fractal_box_count_digimage(..) provides a similar function that makes the calculation in the same way as DigImage.

Transitions within an array may be found conveniently with x_transition_index(..) or y_transition_index(..).

Colour space manipulation is available through rgb_from_bayer(..), bayer_from_rgb(..), hsi_from_rgb(..), hue_from_rgb(..), saturation_from_rgb(..), intensity_from_rgb(..), grey_from_rgb(..), red_from_rgb(..), green_from_rgb(..), blue_from_rgb(..), cyan_from_rgb(..), magenta_from_rgb(..), yellow_from_rgb(..), cmy_from_rgb(..), cmyk_from_rgb(..) and rgb_from_hsi(..).

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.13 Flow functions

The purpose of these functions is to help with the post-processing of fluid flows. These include stream_line(..) and follow_optical_flow(..).

9.14 Coordinate functions

These functions provide access to the coordinate system mechanism within DigiFlow.

At the simplest level x_index(..) and y_index(..) provide a convenient method of generating arrays containing pixel indices, while x_index_world(..) and y_index_world(..) do the equivalent with world coordinates. The more general functions world_coordinate(..) and pixel_coordinate(..) may be used to convert between coordinate systems.

Coordinate systems may be created with coord_system_create(..) in conjunction with coord_system_add_point(..) and coord_system_mapping(..). The default coordinate system may be set using coord_system_set_default(..), and a coordinate sytem can be removed with coord_system_destroy(..). The available coordinate systems may be determined with coord_system_list(..) while information about an individual coordinate system can be obtained with coord_system_destroy(..),

coord_system_get_mapping(..), coord_system_get_points(..) and coord_system_units(..).

Once created, a coordinate system can be modified, thus creating a new coordinate system using coord_system_apply_region(..),coord_system_translate(..) and coord_system_translate_pixel(..).

The definition of a coordinate system can also be written to a dfc file with coord_system_save(..).

In some cases, it can be desirable to create a mapping from pixel to world coordinate systems. The functions coord_system_create_mapping_array(..) and coord_system_transform_array(..) support this. The specialist function stereo_velocity_to_3d(..) provides support for stereo PIV calculations.

View-specific coordinate information can be read or set through view_get_coord_system(..) and view_set_coord_system(..).

Alongside the coordinate system creation is the ability to create regions for use in the Sift facility using region_create(..) and match_intensity_create(..). The regions already defined can be determined with region_list(..) and the details of an individual region are retrieved using get_region(..). Regions may be removed from the list with region_destroy(..).

9.15 Bit-wise operations

It can be useful sometimes to operate on the individual bits of integers as binary numbers. This functionality is provided through bit_and(..), bit_or(..), bit_eor(..), bit_not(..), bit_rotate(..), bit_test(..), bit_set(..), bit_clear(..) and bit_shift(..).

9.16 Camera control

Camera control in DigiFlow dfc code is provided through two different, complementary mechanisms. The first is via the process command to invoke the corresponding items from the File menu. The second mechanism is provided through a direct dfc interface. To make use of this, first create a live view, either using the process interface, ore more simply by a call to camera_live_view(..). If you then wish to save the digitised images to a DigiFlow .dfm file, then first call camera_capture_file(..) to set up the file, and camera_start_capture(..) to start the capture process. Image saving capture may be terminated either when a specified number of images have been captured, or a call is made to camera_stop_capture(..). During the period while the video is being captured, the function camera_is_capturing(..) will return true. The live view may be terminated by simply closing the associated view (camera_live_view(..) and camera_get_view(..) return the necessary handle). A typical example of code to capture a sequence using this interface is given below.

```
hView := camera_live_view();
sleep_for(10); # Wait until we are sure the camera has started
camera_capture_file("MyMovie.dfm");
camera_start_capture();
sleep_for(20); # capture period
ret := camera_stop_capture();
message("Frame rate achieved:"+ret.fpsAchieved);
close_view(hView);
```

For cameras that support full asynchronous triggering, camera_set_mode(..) may be used to change from continuous acquisition to one-shot triggered mode. In the latter case an image will only be acquired (and correspondingly written to any output file) when an explicit trigger is sent. This trigger may be provided either by external hardware (via the frame grabber card),

or from dfc code through camera_trigger(..). The code example below illustrates how to capture images on demand by the user clicking a button.

```
hView := camera_live_view();
sleep_for(10); # Wait until we are sure the camera has started
camera_capture_file("MyMovie.dfm");
camera_wait_for_capture_ready(); # Wait before making async
camera_set_mode("oneshot");
camera_start_capture();
another := true;
while (another) {
  camera_trigger();
  another := ask_yesno("Capture another image?","Snap
    images",allowCancel:=false);
};
camera_stop_capture();
camera_set_mode("continuous");
sleep_for(5);
close_view(hView);
```

Of course the sleep_for(..) or wait_for_timer(..) functions may be used in place of the user clicking a button to provide more precise but flexible timings.

The camera_wait_for_capture_ready(..) call is necessary here to ensure that the capture file is set up properly before switching to asynchronous mode. This is necessary because camera_capture_file(..) does not itself initialise the capture file, but rather asks the video subsystem to do so asynchronously as frames are processed by the system. In the first example above the camera_start_capture(..) call implicitly issued a camera_wait_for_capture_ready(..) call before commencing the capture. However, in the second example, the code would stall if we relied on this since camera_set_mode("oneshot") prevents any more frames being processed except by calls to camera_trigger(..). We must therefore either do the wait with camera_wait_for_capture_ready() *before* making the camera asynchronous, or ensure the camera produces a few frames using camera_trigger(..) between camera_set_mode("oneshot") and camera_start_capture(). The functions camera_frame_number(..) and camera_frames_captured(..) provide additional functionality for monitoring the capture process.

Single frames may be grabbed directly from live video through camera_grab(..) or camera_grab_last(..), while individual lines or columns can be returned through camera_grab_line(..) and camera_grab_column(..), respectively.

In some circumstances it may be desirable to lock the acquisition to the display rate of the computer monitor. To aid in this the directdraw_trigger(..) and directdraw_trigger_period(..) functions not only handle the display, but also send a trigger to the camera. These can interact with functions such as camera_set_sync_line(..), camera_set_strobe(..), camera_trigger(..), camera_wait_for_frame(..), camera_wait_for_sync(..), wait_for_capture(..), wait_for_preprocess(..).

Communications with the camera are possible for many CameraLink cameras using camera_serial(..). The gain and shutter speed can be controlled using camera_set_gain(..) and camera_shutter_speed(..), respectively. The function camera_set_frame_rate(..) uses a variety of techniques, dependent on camera type, to adjust the frame rate. This should be issued prior to the corresponding camera_capture_file(..) if a different capture rate is required. Capture through the default capture file is aided by camera_save_cache(..) and camera_cache_file_name(..).

Where it can be adjusted, the black level of the camera is set by camera_set_black(..). For some cameras, the optimal black level depends on the shutter speed. In such cases, a call

to camera_optimal_black(..) may be used from within camera_shutter_speed(..) to set the black level.

The current settings for the camera can be established with camera_get_settings(..), and a summary of these settings displayed on the status bar using camera_show_status(..).

The display of images can be controlled with camera_display_now(..) and camera_set_display_rate(..).

Other low-level framegrabber specific controls include camera_low_level(..), camera_gpout(..) and camera_override_sync(..).

Details of the camera may be found using camera_capabilities(..), and some of these may be overridden through camera_override(..), while camera_get_settings(..) determines key parameters controlling the camera. Persistent wrapping problems may sometimes be solved through camera_set_frame_offset(..).

9.17 Array plotting functions

Array plotting functions allow data to be transferred into an array in a manner similar to plotting.

Some of these functions are much more restrictive than the drawing functions described in §11, but have their use in manipulating images. Examples include scatter_to_array(..).

A three-dimensional iso-surface can be created from a three-dimensional array using render_3d_isosurface(..).

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.18 Numerical functions

DigiFlow provides a variety of numerical functions that can be used to for purposes ranging from the manipulation of images to numerical solution of equations.

Linear algebra functions include identiy_matrix(..), matrix_multiply(..), solve_linear(..), solve_svd(..), singular_value_decomposition(..), least_squares(..), fit_expression(..), evaluate_expression(..), with fit_line(..) providing a faster route to the fitting of a straight line. (The function fit_as_text(..) provides a convenient method of producing LaTeX-like text from the output of fit_expression(..) or fit_line(..).)

More specialist fitting procedures are available for one-dimensional data through fit_ellipse(..), fit_periodic(..) and evaluate_periodic(..), while for two-dimensional data fit_spline_surface(..), b_spline_2d(..) and b_spline_2d_least_squares(..) are available.

Spectral functions such as fft_row(..), fft_column(..), fft_2d(..) and fft_3d(..), and their inverses inverse_fft_row(..), inverse_fft_column(..), inverse_fft_2d(..), and inverse_fft_3d(..), are complemented by power_spectrum_row(..), power_spectrum_column(..) and power_spectrum_2d(..). Additionally, power_spectrum_1d(..) provides the shell averaging of two-dimensional data to produce a one-dimensional spectrum. For one-dimensional data, the maximum entropy method equivalents mem_spectrum_row(..) and mem_spectrum_column(..) are available. Similarly, DigiFlow also provides spectral calculation of auto_correlation_2d(..), auto_correlation_row(..), auto_correlation_column(..) and cross_correlation_2d(..), cross_correlation_row(..), cross_correlation_column(..).

Root finding is supported through find_root_bisection(..) and find_root_secant(..).

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.19 Differential functions

Functions oriented at differential equations include the calculation of derivatives through `d_dx(..)`, `d_dy(..)`, `d2_dx2(..)`, `d2_dy2(..)`, `curl(..)`, `div(..)`, `grad(..)`, `laplacian(..)`, and solution of the Poisson equation with `solve_poisson(..)`. Functions aimed at supporting numerical solution of the equations include `advect_2d_psi(..)`, `upwind_value(..)` and `shallow_water(..)`. Direct inversion of a gradient field is provided through `inverse_gradient(..)`, while `multigrid(..)` provides a flexible route to template-based equation solution and is used in the computation of `density_from_gradient(..)`.

An example of the use of some of these functions can be found in StreamFunctionVorticity.dfc.

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.20 Handling threads

The DigiFlow interpreter has the ability to handle multithreaded code, thus allowing improvements in performance on multiprocessor machines, and apparent improvements in the time to first results.

Any piece of code may be started in its own thread using `as_thread(..)`, while a DigiFlow processing feature may be started in a separate thread with `process_as_thread(..)`. Both of these functions return a thread handle that may be used in `is_running(..)` to determine if the thread is still running, or `wait_for_end(..)` to suspend execution until the thread has finished running. The handle of a thread associated with a view may be determined using `thread_for_view(..)`. Execution of a thread may be paused with `pause_thread(..)`, restarted with `unpause_thread(..)`, or terminated prematurely with `kill_thread(..)` or `stop_view_thread(..)`. Delays and synchronisation within a thread is achieved through `sleep_for(..), start_timer(..) and wait_for_timer(..)`.

For complex processes (and experienced users), the priority of individual threads may be adjusted with `set_thread_priority(..)`, with the matching `get_thread_priority(..)` recovering the priority of a thread.

The function `thread_set_stopping_time(..)` can be used to specify how much time is expected to elapse between an attempt to stop a thread using `kill_thread(..)` and it actually stopping. Amongst other things, this sets the time delay between requesting a window be closed and it actually closing. If this time is too short, it is more likely that an error is thrown.

External processes may be started and controlled using `issue_command(..)`.

Files containing dfc code may be set to run automatically upon their creation by issuing `autorun_file(..)`.

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

9.21 Web browsing

Basic functionality for controlling the Internet Explorer web browser is available through `www_action(..)`, `www_browse(..)` and `www_exit(..)`. Note, however, that default security settings means that the level of functionality is greater when browsing to a http:// address rather than a file on your own computer.

9.22 ftp functions

Commercial versions of DigiFlow provide the user with a set of ftp functions for transferring data to and from an ftp server. In particular, **ftp_open**(..), **ftp_close**(..), **ftp_current_directory**(..), **ftp_change_directory**(..), **ftp_create_directory**(..), **ftp_remove_directory**(..), **ftp_list_files**(..), **ftp_get_file**(..), **ftp_put_file**(..), **ftp_rename_file**(..) and **ftp_delete_file**(..).

Related to this are the network functions **get_ip_address**(..) and **get_mac_address**(..).

Refer to the dfc help for further details.

9.23 DirectDraw functions

DigiFlow provides a variety of functions using the Microsoft DirectDraw interface in order to provide specialised synchronisation between computer display and camera control. These functions include **list_monitors**(..) to provide the necessary information in multi-monitor systems, **directdraw_create**(..) and **directdraw_destroy**(..) to initialise or destroy the DirectDraw interface, **directdraw_view**(..) to set an image to a flappable buffer, and **directdraw_animate**(..), **directdraw_animate_period**(..), **directdraw_trigger**(..) and **directdraw_trigger_period**(..) to switch between multiple buffers and (for the 'trigger' variants) trigger camera acquisition. The functions **get_monitor**(..) and **list_monitors**(..) are required when dealing with multiple monitors on your system.

The following code segment illustrates a simplified situation in a dual monitor system. The view identified by hView is located on the monitor where the DirectDraw mechanism is to be used, Im0 and Im1 are the two images to be displayed, nFlips are the number of image changes, and nWait is the number of vertical blanking periods to wait between each image change.

```
monitor := get_monitor(hView); # Find which monitor
list := list_monitors();
if (search_string(monitor.name,"1")) {
  # First monitor
  guid := list.GUID0;
} elseif (search_string(monitor.name,"2")) {
  guid := list.GUID2;
} else {
  message("Selected "+monitor.name+", using default");
  guid := "null";
};
# Size of display region
nx := monitor.rect.right - monitor.rect.left + 1;
ny := monitor.rect.bottom - monitor.rect.top + 1;
ddraw := directdraw_create(2,hView,guid); # Use two buffers
directdraw_view(ddraw,0,Im0,"greyscale");
directdraw_view(ddraw,1,Im1,"greyscale");
tFlips := directdraw_animate(ddraw,nFlips,nWait);
directdraw_destroy(ddraw);
```

9.24 Data acquisition functions

Some users find it convenient to control a data acquisition card from within DigiFlow. Support is provided here for controlling PD2-MFx cards from United Electronic Industries (http://www.uei.com). These cards provide high-speed analog input and output and up to sixteen digital input and output data lines. The design of the cards means that minimal cpu load is imposed while doing analog input/output operations, thus allowing the computer to focus on other data.

DigiFlow support for one of these cards is achieved first by the call `install_function_dll("DataAcquisition.dll","UEIDataAcquisition",true);` before using `uei_open(..)` to open the card. At the end of the session, `uei_close(..)` must be called to tidy up and release resources. Note that DigiFlow can only access only one such card at a time.

Analog output is provided by the single command `uei_analog_out(..)`, whereas three commands – `uei_analog_in_configure(..)`, `uei_analog_range(..)` and `uei_analog_in(..)` – are necessary to provide the analog in functionality.

TTL-level digital input can be read with `uei_digital_in(..)`, `uei_digital_in_clear_events(..)`, `uei_digital_in_configure(..)`, `uei_digital_in_status(..)` and `uei_digital_in_wait(..)`. Similarly, digital output is made available with `uei_digital_out(..)` and `uei_digital_out_array(..)`.

Some models of the PD2-MFx card are provided with a three-channel Intel Universal Counter Timer chip. This can be configured in many different ways using the command set `uei_counter_clear_events(..)`, `uei_counter_configure(..)`, `uei_counter_gate(..)`, `uei_counter_mode(..)`, `uei_counter_pulse(..)`, `uei_counter_read(..)`, `uei_counter_reset(..)`, `uei_counter_status(..)`, `uei_counter_wait(..)` and `uei_counter_write(..)`.

The low-level functions `uei_get_configuration(..)` and `uei_set_configuration(..)` provide some additional functionality.

Built on top of these data acquisition and control functions is a set of functions and macros specifically intended for the control and monitoring of a high-precision rotating table. These functions are identified through names beginning with `turntable_`. Further details may be available on request.

9.25 Serial communications

Some laboratory equipment can be controlled and/or data read through a protocol based on RS232 serial communications. To facilitate this, DigiFlow provides a basic set of functions for this. In particular, `serial_open(..)` opens and configures a communication port, while `serial_close(..)` releases this resource. Whilst open, `serial_write(..)` and `serial_read(..)` allow data to be transferred.

9.26 GhostScript functions

Although GhostScript does not form part of DigiFlow, if DigiFlow detects that it is installed on your system then DigiFlow can make use of GhostScript for converting PostScript files into raster images for processing (or simply displaying). The `ghost_script(..)` function provides the simplest and most direct support, giving access to the GhostScript interpreter and returning resulting image in a rgb formatted array. More detailed control over the interpreter is provided through `ghostscript_start(..)`, `ghostscript_execute(..)`, `ghostscript_end(..)`, `ghostscript_get_image(..)`, `ghostscript_show_image(..)` and `ghostscript_get_output(..)`.

9.27 Particle tracking functions

Files produced by the particle tracking system in DigiFlow necessarily have a different format from those of other features. While the particle tracking menu provides a number of post-processing options, there will be times when it is desirable to construct post-processing using dfc code. Support for this is provided through `ptv_open(..)` and `ptv_close(..)` to access the .dft particle tracking files. Once opened, `ptv_tracks(..)` provides a mechanism for enumerating the tracks of individual particles (*i.e.*, assembling paths from the coordinate

and linking information stored in the .dft files), and ptv_velocity(..) calculates the velocities associated with the tracks. The functions ptv_tracks_compound(..) and ptv_velocity_compound(..) provide some additional information in a different format. More specific information about individual particles can be determined using ptv_read_particles(..), and a specific particle within this located using ptv_particle_details(..).

9.28 Logging

DigiFlow includes code for helping track down problems, either with DigiFlow code, or with user dfc files. The following group of functions support this process by logging certain groups of activities. These functions are intended primarily for the developer to help track down the cause of any lingering bugs.

Key entry points are log_start(..), log_stop(..), log_message(..), log_allocated(..), log_memory(..), log_flush(..) and log_flush_every_time(..).

Normally these functions will only be used while seeking technical support for DigiFlow.

The logging functions produce output into a file named DigiFlow.log. (Note that when DigiFlow crashes, it will attempt to write as much diagnostic information as possible to DigiFlow.log.) For example, if log_start(..) is used for levels 8, 9 and then, then this file contains entries such as

```
543  136295776        92 Allocate   : $AddItem:New - nTotal
544  136295880       472 Allocate   : $AddItem:pItem - nTotal
545  158924832   5242880 Allocate   : CreateImage:RR
546  164233248   5242880 Allocate   : CreateValue$Kind:Value%AA
547  158924832   5242880 Deallocate: $DiscardImage:RR
547  147345992        92 Allocate   : $AddItem:New - lay0avg
548  147346096       472 Allocate   : $AddItem:pItem - lay0avg
549  147346584       128 Allocate   : CreateValue$Kind:Value%List
```

where the first column gives a sequence number, the second a memory address, the third the size of the associated structure, the fourth the action and the fifth an indication of where in DigiFlow the memory was allocated. Here, nTotal is a scalar variable created by a user dfc file, and lay0avg is an array read from an image. Note that the image itself is created as item 545, then destroyed as 547. The array is created as 546 and stored as lay0avg in 547 and 548.

If, subsequently, log_allocated(..) is called, then a block like that below will be added to the log:

```
################################################
 # Currently allocated memory -- START
 # current allocation =
 ################################################
  521  136310400        92 $AddItem:New - i_min
  522  136310504       472 $AddItem:pItem - i_min
  523  136310992        92 $AddItem:New - i_max
  524  136311096       472 $AddItem:pItem - i_max
  525  136311584        92 $AddItem:New - j_min
  526  136311688       472 $AddItem:pItem - j_min
...
 4285  148167192       472 $AddItem:pItem - k
 4286  148167680        92 $AddItem:New - iFile
 4287  148167784       472 $AddItem:pItem - iFile
 4325  224133152   3672360 CreateValue$Kind:Value%AA
 ################################################
 # Currently allocated memory -- END
 # current allocation =
 ################################################
```

This block lists all memory that was allocated following log_start(8); (and before log_stop(8);) that has been allocated but not deallocated. Note that the sequence number

(first column) is not necessarily in order, but refers back to the numbers in the previous list. In this case, most of the entries in the list are the direct result of the dfc file that was running when it was created.

9.29 Registry functions

DigiFlow itself makes minimal use of the registry, but does provide some access to it through the functions **registry_list_keys**(..), **registry_get_value**(..), **registry_set_value**(..) and **registry_create_key**(..).

9.30 Configuration and licence functions

During start-up, dfc code is responsible for some of the configuration of DigiFlow. Advanced users may enhance this using functions such as **add_menu_item**(..), **add_menu_separator**(..), **get_submenu**(..), **add_submenu**(..), **add_image_reader**(..), **add_image_writer**(..), **add_movie_reader**(..), **add_image_reader_macro**(..), **install_function_dll**(..).

Information about the state of DigiFlow licensing is obtained with **is_digiflow_licensed**(..) and **digiflow_licence_type**(..), while the DigiFlow splash screen may be manipulated with **show_splash**(..) and **hide_splash**(..).

Some releases of DigiFlow are available compiled with more than one compiler. Specific information about this is provided through **compiler_supplier**(..) and **compiler_version**(..). The date on which the main `digiflow.exe` was compiled is accessible through **digiflow_build_date**(..), with options within this determined using **is_debug**(..) and **is_openmp**(..).

Information about the folder in which it is installed can be determined using either **digiflow_directory**(..) or **digiflow_directory_url**(..), the latter providing details of the machine on which DigiFlow resides if you are using a mapped network drive. The folder in which DigiFlow was started can be determined from **start_directory**(..) and **start_directory_url**(..).

The version of Windows on which DigiFlow is running can be determined using **windows_version**(..).

DigiFlow's use of resources can be monitored through **system_load**(..), **memory_status**(..) and **memory_status_show**(..), while an attempt to free some resources is achieved with **memory_tidy**(..).

9.31 Miscellaneous functions

The equation of state for salt water is accessible through **seawater_density**(..).

The DigiFlow help system can be directed to a particular section in a web browser using **digiflow_help**(..).

9.32 All functions

The table below gives a list of the names all the operators, functions and constants understood by DigiFlow, including any alternate spellings.

```
!
"
#
&
()
*
*=
+
+=
-
-=
/
/=
0x
:=
<
<<>>
<=
<>
=
>
>=
?
?=
[]
{/ /}
{}
^
_!News - latest
_!News - old
_Input streams
_LaTeX
_Output streams
_Returning images
_Simple plot
abs
```

acos
acos_rad
add_color
add_color_scheme
add_colour
add_colour_scheme
add_image_reader
add_image_reader_macro
add_image_writer
add_menu_item
add_menu_separator
add_movie_reader
add_submenu
advect_2d_psi
allow_break
allow_query
and
animate_view
as_thread
asin
asin_rad
ask_directory
ask_directory_modeless
ask_file
ask_file_modeless
ask_image
ask_image_modeless
ask_integer
ask_integer_modeless
ask_list
ask_list_modeless
ask_printer
ask_real
ask_real_modeless
ask_string
ask_string_modeless
ask_yesno
ask_yesno_modeless
atan
atan_rad
auto_correlation_2d
auto_correlation_column
auto_correlation_row
autorun_file
b_spline_2d
b_spline_2d_least_squares
bayer_from_rgb
beep
bessel
bit_and
bit_clear
bit_eor
bit_not
bit_or
bit_rotate
bit_set
bit_shift
bit_test
blue_from_rgb
camera_cache_file_name
camera_capabilities
camera_capture_file
camera_display_now
camera_external_shutter_speed
camera_frame_number
camera_frames_captured
camera_get_settings
camera_get_view
camera_gpout
camera_grab
camera_grab_column
camera_grab_last
camera_grab_line
camera_is_capturing
camera_live_view
camera_low_level
camera_open_serial
camera_optimal_black
camera_override
camera_override_sync
camera_save_cache
camera_serial
camera_serial_online
camera_set_black
camera_set_display_rate
camera_set_frame_offset
camera_set_frame_rate
camera_set_frame_straddle
camera_set_gain
camera_set_mode
camera_set_strobe
camera_set_sync_line
camera_show_status
camera_shutter_speed
camera_start_capture
camera_start_frame_straddle
camera_stop_capture
camera_stop_frame_straddle
camera_switch_mode
camera_trigger
camera_wait_for_capture
camera_wait_for_capture_ready
camera_wait_for_frame
camera_wait_for_preprocess
camera_wait_for_sync
cascade_views
change_directory
char
close_all_views
close_console
close_file
close_view
cmy_from_rgb
cmyk_from_rgb
color_scheme
color_scheme_from_image
colour_scheme
colour_scheme_from_image
command_line_arguments
compile
compile_rp
compiler_supplier
compiler_version
component_names
computer_for_file
contour_image
convert_to_rp
coord_system_add_point
coord_system_apply_region
coord_system_create
coord_system_create_mapping_array
coord_system_destroy
coord_system_get_mapping
coord_system_get_points
coord_system_list
coord_system_mapping
coord_system_save
coord_system_set_default
coord_system_transform_array
coord_system_translate
coord_system_translate_pixel
coord_system_units
copy_file
copy_file_wait
cos
cos_rad
count
crash_digiflow
create_directory
cropped_add
cropped_assign
cropped_div
cropped_div_reversed
cropped_mul
cropped_power
cropped_power_reversed
cropped_sub
cropped_sub_reversed
cross_correlation_2d
cross_correlation_column
cross_correlation_row
curl
current_directory
cyan_from_rgb
d2_dx2
d2_dy2
d_dx
d_dy
date
degrees_from_radians
delete_color_scheme
delete_colour_scheme
delete_file
density_from_gradient
destroy_variable
dfc_as_latex
dialog
digiflow_build_date
digiflow_directory
digiflow_directory_url
digiflow_help
digiflow_licence
digiflow_licence_type
digiflow_site_licence
digiflow_version
directdraw_animate
directdraw_animate_period
directdraw_create
directdraw_destroy
directdraw_trigger
directdraw_trigger_period
directdraw_view
directdraw_wait
div
do_not_wait
draw_append_drawing
draw_arc
draw_begin_image
draw_begin_lineto
draw_begin_marks
draw_begin_vector
draw_bottom_axis
draw_circle
draw_clip_box
draw_colour_scheme
draw_create_key
draw_defaults
draw_destroy
draw_embed_drawing
draw_enable_latex
draw_end
draw_fill_color
draw_fill_colour
draw_font
draw_get_axes
draw_get_font_height
draw_get_status
draw_group_begin
draw_group_end
draw_image
draw_image_scale
draw_image_scale_vertical

draw_install_latex_macro
draw_key_entry
draw_left_axis
draw_line
draw_line_color
draw_line_colour
draw_line_style
draw_line_width
draw_lineto
draw_mark
draw_mark_size
draw_mark_type
draw_move
draw_no_fill
draw_on_array
draw_on_emf
draw_on_file
draw_on_view
draw_polygon
draw_rectangle
draw_restore_state
draw_right_axis
draw_save_state
draw_set_axes
draw_set_base_scale
draw_set_size
draw_skip_points
draw_start
draw_text
draw_text_color
draw_text_colour
draw_top_axis
draw_user_line_style
draw_vector
draw_vector_scale
draw_vector_scale_vertical
draw_x_axis
draw_y_axis
dye_ideal
dye_red_fiesta
eigen_system
eigen_values
else
elseif
enable_timing
end
end_data
eor
erf
erfc
evaluate_expression
evaluate_periodic
execute
exists
exit
exit_digiflow
exp
export_to_eps
export_to_simple_eps
extract
false
fft_2d
fft_3d
fft_column
fft_row
file_details
fill_blob_list
fill_blobs
filter_centile
filter_convolution
filter_geometric
filter_low_pass
filter_max
filter_max_neighbours
filter_median
filter_min
filter_min_neighbours
filter_periodic_centile
filter_periodic_convolutio
n
filter_periodic_geometric
filter_periodic_low_pass
filter_periodic_max
filter_periodic_median
filter_periodic_min
filter_periodic_std_dev
filter_std_dev
find_blobs
find_contour_start
find_edge
find_root_bisection
find_root_secant
fit_as_text
fit_ellipse
fit_expression
fit_image_b_spline
fit_line
fit_periodic
fit_spline_surface
flip_horizontal
flip_vertical
floor
flush_file
follow_optical_flow
for
fractal_box_count
fractal_box_count_digimage
from
ftp_change_directory
ftp_close
ftp_create_directory
ftp_current_directory
ftp_delete_file
ftp_get_file
ftp_list_files
ftp_open
ftp_put_file
ftp_remove_directory
ftp_rename_file
function
function_help
gaussian_noise
get_active_view
get_component
get_configuration
get_dfc_path
get_dialog_response
get_file_variables
get_global
get_image
get_ip_address
get_key
get_local_variables
get_mac_address
get_monitor
get_mouse_box
get_mouse_click
get_mouse_line
get_mouse_position
get_mouse_rect
get_process_details
get_region
get_submenu
get_user_variables
get_variable
get_view_as_image
get_view_class
ghostscript
ghostscript_end
ghostscript_execute
ghostscript_get_image
ghostscript_get_output
ghostscript_show_image
ghostscript_start
grad
green_from_rgb
grey_from_rgb
hide_splash
histogram
hsi_from_rgb
hue_from_rgb
identity_matrix
if
ifelse
in_parallel
include
indirect
install_function_dll
int
intensity_from_rgb
interpolate_image
inverse_fft_2d
inverse_fft_3d
inverse_fft_column
inverse_fft_row
inverse_gradient
is_array
is_code
is_compound
is_debug
is_digiflow_licenced
is_digiflow_licensed
is_drawing
is_file_local
is_integer
is_list
is_live_view
is_memo
is_null
is_numeric
is_openmp
is_parallel
is_real
is_running
is_string
is_view
issue_command
jpeg_get_comments
kill_thread
laplacian
least_squares
left_string
length
list_components
list_file_details
list_files
list_files_global
list_global_variables
list_local_variables
list_monitors
list_names
list_system_variables
list_user_variables
ln
log
log_allocated
log_flush
log_flush_every_time
log_memory
log_message
log_start
log_stop
look_up_table
lower_case
magenta_from_rgb
make_array
make_like
make_list

make_string
map_cartesian_to_polar
map_polar_to_cartesian
match_intensity_create
matrix_multiply
max
max_index_x
max_index_y
max_string_length
max_value
maximise_digiflow
maximise_view
maximize_digiflow
maximize_view
mean
mem_spectrum_column
mem_spectrum_row
memory_status
memory_tidy
message
message_modeless
mid_string
min
min_index_x
min_index_y
min_value
minimise_digiflow
minimise_view
minimize_digiflow
minimize_view
mod
mouse_get_mode
mouse_set_mode
move_file
move_icon
multigrid
n_components
n_size
n_waiting_write_image
new_line
new_view
new_view_clean
new_view_floating
nice_number
nice_number_string
not
null
open_binary_file
open_console
open_file
open_image
open_image_when_ready
or
pack_time
pause_thread
pi
pixel_contour
pixel_coordinate
plot
plot_axes
plot_axis_type
plot_destroy
plot_drawing
plot_fit
plot_get_state
plot_image
plot_line
plot_new
plot_titles
plot_update_view
plot_vectors
plot_view_handle
power_spectrum_1d
power_spectrum_2d
power_spectrum_column
power_spectrum_row
print_view
print_view_dialog
process
process_as_thread
process_time
ptv_close
ptv_open
ptv_particle_details
ptv_read_particles
ptv_tracks
ptv_tracks_compound
ptv_velocity
ptv_velocity_compound
quit
r_count
r_mean
r_sum
radians_from_degrees
random_array
random_number
randomise
read_array
read_binary
read_compound_image
read_console
read_data
read_file
read_image
read_image_archive
read_image_details
read_image_queue
read_image_queue_create
read_image_queue_destroy
read_image_from_view
read_image_when_ready
read_into_array
read_line
read_table
real
red_from_rgb
region_create
region_destroy
region_list
registry_create_key
registry_get_value
registry_list_keys
registry_set_value
remove_icon
remove_spaces
remove_trailing_zeros
render_3d
render_3d_isosurface
render_points_3d
replace_hashes
replace_values
resample_curve
rescale_image
restore_digiflow
restore_view
reverse_polish
rgb_from_bayer
rgb_from_hsi
rgb_from_image
right_string
rms
roll
rotate_anticlockwise
rotate_clockwise
rotate_image
round
sample_values
saturation_from_rgb
save_view
scatter_to_array
scrunch_string
search_string
seawater_density
serial_close
serial_open
serial_read
serial_write
set_configuration
set_dfc_path
set_dialog_response
set_file_end
set_file_pointer
set_global
set_variable
shallow_water
shift
shift_interpolated
show_memory_status
sign
sin
sin_rad
singular_value_decompositi
on
slave_view_3d
sleep_for
smooth_contour
solve_linear
solve_poisson
solve_svd
sort
sort_array
sort_index
sqrt
start_directory
start_directory_url
start_timer
status_bar_message
step
stereo_velocity_to_3d
stop_view_thread
stream_function
streamline
sum
system_load
t_index
t_size
tan
tan_rad
theta_mean
thread_for_view
thread_set_stopping_time
tile_views
time
time_interval
to
trace
transpose
true
try_execute
try_include
turntable_angle
turntable_average_output
turntable_ctrl
turntable_current_speed
turntable_enable
turntable_filter_low_pass
turntable_filter_median
turntable_fourier
turntable_get_variable
turntable_heart_beat
turntable_output
turntable_record
turntable_set_speed
turntable_set_variable
turntable_state
turntable_status_bar
turntable_stop
turntable_target_speed

turntable_update_base
turntable_update_calibration
turntable_update_waveform
turntable_varying_speed
turntable_wait
turntable_where
uei_analog_in
uei_analog_in_configure
uei_analog_in_range
uei_analog_out
uei_analog_start
uei_close
uei_counter_clear_events
uei_counter_configure
uei_counter_gate
uei_counter_mode
uei_counter_pulse
uei_counter_read
uei_counter_reset
uei_counter_status
uei_counter_wait
uei_counter_write
uei_digital_in
uei_digital_in_clear_events
uei_digital_in_configure
uei_digital_in_status
uei_digital_in_wait
uei_digital_out
uei_digital_out_array
uei_get_configuration
uei_open
uei_set_configuration
unpack_time
unpause_thread
upper_case
view
view_3d
view_color
view_colour
view_connect_thread
view_counter
view_disconnect_thread
view_fit_all_to_zoom
view_fit_to_zoom
view_get_coord_system
view_get_time
view_icon
view_live
view_points_3d
view_rgb
view_rotated
view_scalar_colour
view_scalar_range
view_set_coord_system
view_time
view_title
view_toggle_color
view_toggle_colour
view_variables
view_vector_colour
view_vector_remove_mean
view_vector_scale
view_vector_spacing
view_vectors
view_zoom
view_zoom_all
view_zoom_all_to_fit
view_zoom_to_fit
wait_for_end
wait_for_ever
wait_for_file
wait_for_timer
wait_for_write_image_queue_empty
where
where_is
while
window_from
windows_version
world_coordinate
wrapped_add
wrapped_assign
wrapped_div
wrapped_div_reversed
wrapped_extract
wrapped_mul
wrapped_power
wrapped_power_reversed
wrapped_sub
wrapped_sub_reversed
write_array
write_binary
write_console
write_file
write_image
write_image_archive
write_image_queue
write_integer_array
write_rgb_image
write_rgb_image_queue
write_thesis
www_action
www_browse
www_exit
x_accumulate
x_centroid
x_count
x_index
x_index_pixel
x_index_world
x_max_index
x_max_value
x_mean
x_min_index
x_min_value
x_moment
x_replicate
x_rms
x_size
x_sum
x_transition_index
xor
y_accumulate
y_centroid
y_count
y_index
y_index_pixel
y_index_world
y_max_index
y_max_value
y_mean
y_min_index
y_min_value
y_moment
y_replicate
y_rms
y_size
y_sum
y_transition_index
yellow_from_rgb
z_index
z_size

10 Macros

The full power of DigiFlow is released through its ability to run macros in the form of DigiFlow command files (more frequently referred to as dfc files or dfc code) to automate complex or repetitive tasks. This section supplements the discussion of the basic

10.1 DigiFlow command files

DigiFlow command files are simple text files that contain code that is interpreted and executed by the DigiFlow interpreter. The language used by this interpreter is a simple superset of that described already in this section, with the addition of commands to invoke specific DigiFlow processes by mimicking the functionality of the user-interface and process chaining ability (see §7).

It is recommended that the default dfc extension be used for all DigiFlow command files. This will not only ensure that the command files are visible to the Run Code dialog (see §5.1.2), but also that double-clicking on a dfc file in Windows Explorer, or dragging a dfc file to DigiFlow, will ensure that it is run correctly.

10.1.1 Running processes

The basic command for running a DigiFlow process from dfc code is

process *dlg*;

where *dlg* is a compound variable containing the responses for the dialog associated with that menu item and the name of the process to be run. This is best illustrated by example.

To compute the arithmetic time average of a movie test.dfm and store the result in ave.dfi, we may construct the following dfc code:

```
dlg.Input := "test.dfm";
dlg.Output := "ave.pic";
dlg.Kind := "Arithmetic";
dlg.process := "Analyse_TimeAverage"
process dlg;
```

or equivalently using the compound variable constructor ≪ .. ≫,

```
dlg := ≪
     Input := "test.dfm";
     Output := "ave.pic";
     Kind := "Arithmetic";
     process := "Analyse_TimeAverage";
≫;
process dlg;
```

In this example, the variable `dlg` is used to store the dialog responses that are required for the Analyse_TimeAverage process (see §5.6.1.1). The dialog responses have been stored in the form of a compound variable (see §8.2.2).

Typically, a process requires a number of mandatory responses, and may also accept a variety of optional responses. Here we have defined only the mandatory input, output and average type responses. If no errors are detected, the interface uses these values to initialise the control structures for the averaging process, and then starts the averaging.

Although not used in the above examples, the **process** command has a return value that contains information about the final state of the process. For most processes, the return value is a compound value that contains things like the final image computed and the handle(s) of any views left open by the process. In the above example, if

```
ret := process dlg;
```

then the variable `ret` contains components `.hOutput`, the handle of the view left open, `.imgOutput`, a copy of the image created by the time average, and `.process`, which is a copy

of the entry value (here "Analyse_TimeAverage"). The simplest way of determining the contents of the return value is to make a call to **view_variables(..)** after executing **process**. If no assignment is made of the return value then this information is simply discarded.

By default, processes are run in a separate thread from the interpreter handling the dfc code, but the execution of the dfc code is suspended until the execution of the process is complete. This behaviour may be modified by starting the process using

thread_id := **process_as_thread** *dlg*;

(*dlg* must always contains the .process declaration). When started in this way, control is returned to the dfc code as soon as the *dlg* variable has been executed to start the process. The dfc code is then free to start other processes or make other computations. However, any code that relies on an output created by the *command* started in this manner must execute **wait_for_end**(*thread_id*); or at least check **is_running**(*thread_id*) prior to making use of this output. See §9.20 for further information on threads. Note if **process_as_thread** is used, then the return value available with **process** is automatically discarded and is not available to the calling dfc code.

For each of the processes within DigiFlow that may be accessed by this method. The easiest way of obtaining this list is to run the process interactively, then enter the Dialog responses facility described in §5.2.9. This gives a list of the values used, including any relevant optional ones.

10.1.2 Control of input streams

Input streams, such as that represented by dlg.Input in the previous section, may be either a single file (which may contain either a single image or a movie of images), a series (with a varying numeric part represented by one or more hash (#) characters, or a collection of images. For a macro, a collection is typically specified using wild cards. The wild card for a single character is either a question mark (?) or percentage symbol (%), whereas an asterisk (*) or dollar symbol ($) represents an arbitrary number of characters. The reason why there are two symbols for each type of wild card stems from the way Windows interprets wild cards immediately when using an open or save file dialog. Note that it is more efficient to utilise the numeric substitution character (#) than wild cards, and that numeric substitution can cope with a much larger number of files.

The input streams can be modified in a number of ways, just as they could through interactive use of DigiFlow. For most streams (those with a Sift button), aspects of the stream such as the timing and region of interest can be changed. If the image source is a full colour image, then the Colour component to be processed may be selected. In each case, the additional control is optional and is achieved by appending further details to the name of the associated control. For example, to select the green component of the full colour image MyPic.bmp for dlg.Input, then the lines

```
dlg.Input := "MyPic.bmp";
dlg.Input_Component := "green";
```

should be included in the dfc code.

The various controls available for input streams are discussed below.

10.1.2.1 Folder for input stream

By default, if the file name specified for the input stream does not contain a path component, then the stream will be taken from the current folder (directory). If you wish to specify a different folder without including it in the file name, then _Folder may be appended to the input stream name and set to a string value specifying the folder required.

10.1.2.2 Archive file for input streams

If available, .dfa archive files can be useful as they both act as a collection point for sequences of files and store additional information for file formats that cannot store this internally. Appending `_Options.UseArchive` to the input stream specifier allows the logical **true** or **false** to specify whether or not DigiFlow searches for a .dfa file when processing an input image stream. If **true** and an archive is found, then the information contained in it is merged with that found in the image files themselves.

10.1.2.3 Displaying input

Normally it is desirable to show the input streams on the display as the process takes place. However, in some cases it may be desirable to suppress this. Control of whether or not input is displayed is achieved by appending `_Options.Display` to the corresponding input name. The resulting logical variable will then display the input if **true**, or suppress it if **false**. If not specified, then the input stream will normally be displayed.

10.1.2.4 Colour component

The colour component input is applicable only to source streams providing true colour images (*e.g.* 24 bit .bmp files). The control is accessed by appending `_Options.Component` to the corresponding input name. For example, if `This.Experiment` is set to a full colour image, then `This.Experiment_Options.Component` provides selection of the colour component, as detailed in the following table. Here, *stream* represents the base name of the input stream (*e.g.* `This.Experiment` in the above example).

*stream*_Options.Component :=	**Description**
"RGB"	Returns a three-plane full colour image. Note that some DigiFlow options will only process the first (red) plane when presented with a full colour image.
"mono"	Return the best monochrome version of the image.
"red"	Return the red component of the image in RGB space.
"green"	Return the green component of the image in RGB space.
"blue"	Return the blue component of the image in RGB space.
"hue"	Return the hue (colour) of the image, in Hue/Saturation/Intensity space.
"saturation"	Return the saturation (purity of the colour) of the image, in Hue/Saturation/Intensity space.
"intensity"	Return the intensity of the image, in Hue/Saturation/Intensity space.
"cyan"	Return the cyan component of the image in CMY space.
"magenta"	Return the magenta component of the image in CMY space.
"yellow"	Return the yellow component of the image in CMY space.
"black"	Return the black component of the image in CMYK space.
"grey"	Return the equivalent grey level.
"mean"	Return the mean of the three RGB components.
"max"	Return the maximum of the three RGB components.
"min"	Return the minimum of the three RGB components.

Consult §4.1 for details on the relationship between the returned value and the individual red, green and blue colour components.

10.1.2.5 Timing control

For input streams having a Timings button, the timing details may be set by appending _Time to the corresponding input name. This new variable is a compound variable that provides a number of ways of controlling the timings. For example, if dlg.Input controls an input stream, then dlg.Input_Time.ToStep controls the last frame to be processed.

The timings may be specified in terms of either frames or seconds. If both are specified, the frames version takes precedence. Details of both methods of control are given below. Note that you need specify only those controls you wish to change from their defaults: the default action is to process every frame of the input stream.

Variable	Description
*stream*_Time.FromStep	Select the first frame to be processed.
*stream*_Time.ToStep	Select the last frame to be processed.
*stream*_Time.StepCount	The number of frames to be processed. This has priority over _Time.ToStep.
*stream*_Time.StepBy	The spacing of the frames to be processed.
*stream*_Time.FromTime	Select the start time for the sequence. This is rounded to the nearest frame.
*stream*_Time.ToTime	Select the end time for the sequence. This is rounded to the nearest frame.
*stream*_Time.TimeStep	The time step for the sequence. This is rounded to the nearest frame.
*stream*_Time.TimeStepFile	The interval between the frames in the sequence. This is ignored for file formats that store time information, but is used for file formats (*e.g.* sequences of .bmp files) that do not store such information.

10.1.2.6 Selecting regions

It is often desirable to select only a part of an image for processing. This is achieved through the specification of a region by appending _Region to the corresponding input name. This new variable is a compound variable that provides a number of ways of controlling the region. For example, if dlg.Input controls an input stream, then dlg.Input_Region.xMin sets the left-hand side of the region. The table below summarises the available options.

Variable	Description
*stream*_Region.Kind	Select the type of region. This is a string variable that should be set to one of: "All" Indicates that all the input stream should be used. "Conform" Indicates that a region conforming to that of the *master* stream should be used. This option is not available for streams that are the master. Typically, a given process will have only one master stream, and this will be the first stream in the dialog box. "PixelRect" Indicates that a rectangle (specified in pixel coordinates) will be used. Values must be specified for the _Region.xMin, _Region.xMax, _Region.yMin and _Region.yMax variables. "Named" Indicates that a named region should be used. The name must be specified for the _Region.Name variable.
*stream*_Region.xMin	Must be specified when _Region.Kind is "PixelRect". Specifies

	the left-hand side of the pixel rectangle defining the region.
*stream*_Region.xMax	Must be specified when _Region.Kind is "PixelRect". Specifies the right-hand side of the pixel rectangle defining the region.
*stream*_Region.yMin	Must be specified when _Region.Kind is "PixelRect". Specifies the bottom of the pixel rectangle defining the region.
*stream*_Region.yMax	Must be specified when _Region.Kind is "PixelRect". Specifies the top of the pixel rectangle defining the region.
*stream*_Region.Name	Must be specified when _Region.Kind is "Named". Specifies the name of the previously saved region.

10.1.2.7 Matching intensities in input streams

As described in §4.3.3, it can be necessary to adjust the intensities of images on an image-by-image basis in order to match their intensities to a reference level. This is achieved through the specification of a Match Intensity by appending _MatchIntensity to the corresponding input name. This new variable is a compound variable that provides a number of ways of controlling the intensity matching. For example, if dlg.Input controls an input stream, then dlg.Input_MatchIntensity.Name sets the intensity matching to a previously named scheme. The table below summarises the available options.

Variable	**Description**
*stream*_MatchIntensity.Kind	Optional variable that select the type of intensity matching. This is a string variable that should be set to one of: "None" Indicates that no intensity matching will be used.. "Local" Indicates that the settings are defined locally. Values must be specified for _MatchIntensity.xMinA, _MatchIntensity.xMaxA, _MatchIntensity.yMinA, _MatchIntensity.yMaxA, _MatchIntensity.xMinB, _MatchIntensity.xMaxB, _MatchIntensity.yMinB, _MatchIntensity.yMaxB "Named" Indicates that a named setting should be used. The name must be specified for the _MatchIntensity.Name variable. If this variable is not specified, but _MatchIntensity.xMinA is, then "Local" will be assumed. Similarly, if this variable is not specified, but _MatchIntensity.Name is, then "Named" is assumed.
*stream*_MatchIntensity.xMinA	The left-hand edge of region A.
*stream*_MatchIntensity.xMaxA	The right-hand edge of region A.
*stream*_MatchIntensity.yMinA	The bottom edge of region A.
*stream*_MatchIntensity.yMaxA	The top edge of region A.
*stream*_MatchIntensity.xMinB	The left-hand edge of region B.
*stream*_MatchIntensity.xMaxB	The right-hand edge of region B.
*stream*_MatchIntensity.yMinB	The bottom edge of region B.

*stream*_MatchIntensity.yMaxB	The top edge of region B.
*stream*_MatchIntensity.IntensityA	If specified, then this gives the reference intensity for region A. If not specified, then the reference intensity is determined from the first image to be processed.
*stream*_MatchIntensity.IntensityB	If specified, then this gives the reference intensity for region B. If not specified, then the reference intensity is determined from the first image to be processed.
*stream*_MatchIntensity.Name	If specified, then this string gives the name of the match intensity scheme to use.

10.1.2.8 Waiting for input streams

In some cases, the input stream will not exist when a process is started. DigiFlow allows the possibility of the process waiting for the input stream to come into existence through some other mechanism (*e.g.*, being created or copied by an external program or the user) rather than simply throwing an error. The table below summarises the possible actions.

Variable	**Description**
*stream*_Options.WaitFor	Determines the time DigiFlow will wait for the input stream to be created, if it does not exist already. A numeric (floating point or integer) value should be assigned to this variable. A zero or negative value implies no waiting, while a positive value gives the timeout period for the stream.

10.1.3 Control of output streams

In a similar way to the ability to modify input stream timing, colour component, *etc.*, some aspects of the output streams may also be modified. The following subsections detail the available controls.

10.1.3.1 Folder for output streams

By default, if the file name specified for the output stream does not contain a path component, then the stream will be taken from the current folder (directory). If you wish to specify a different folder without including it in the file name, then _Folder may be appended to the output stream name and set to a string value specifying the folder required.

10.1.3.2 Archive file for output streams

For many users, it is desirable to generate .dfa archive files that both act as a collection point for sequences of files and store additional information for file formats that cannot store this internally. Appending _Options.UseArchive to the output stream specifier allows the logical **true** or **false** to specify whether or not the .dfa file is generated.

10.1.3.3 Displaying output

Normally it is desirable to show the output streams on the display as the process takes place. However, in some cases it may be desirable to suppress this. Control of whether or not output is displayed is achieved by appending _Options.Display to the corresponding output name. The resulting logical variable will then display the output if **true**, or suppress it if **false**. If not specified, then the output stream will normally be displayed.

10.1.3.4 Output stream colour

The colour scheme used for an output stream may be set by appending `_Options.Colour` to the name of the stream, and specifying a colour scheme, either as the name of the scheme, or as an array of RGB colour values.

Variable	Description
stream`_Options.Colour := "single cycle";`	Specify a named colour scheme for this output stream.
stream`_Options.Colour :=` *my_array* `;`	Specify the colour scheme as an array of colour values. The array should contain at least 256×3 elements, the first index corresponding to an 8-bit intensity, and the second to the colour component in the order Red, Green, Blue. Each element of the array should be scaled between 0.0 and 1.0.
stream`_Options.Color := "single cycle";`	Identical to the above description with the UK spelling of colour.
stream`_Options.Color :=` *my_array*`;`	Identical to the above description with the UK spelling of colour.

Colour scheme information may also be specified simultaneously for all output streams by omitting the *stream*`_Options` prefix.

Variable	Description
`Colour := "single cycle";`	Specify a named colour scheme for all output streams.
`Colour :=` *my_array*`;`	Specify the colour scheme as an array of colour values. The array should contain at least 256×3 elements, the first index corresponding to an 8-bit intensity, and the second to the colour component in the order Red, Green, Blue. Each element of the array should be scaled between 0.0 and 1.0.
`Color := "single cycle";`	Identical to the above description with the UK spelling of colour.
`Color :=` *my_array*`;`	Identical to the above description with the UK spelling of colour.

If both the *stream*`_Options.Colour` and `Colour` variants are used for a given output stream, then the *stream*`_Options.Colour` variant has priority. Similarly, if both the UK and US spellings of colour are used, then the UK spelling has priority.

For output streams that allow full (true) colour images, whether or not one is saved may be controlled by setting the logical `_Options.TrueColour`.

10.1.3.5 First index

The first index to be used in naming the files in an output sequence may be specified by appending `_Options.FirstIndex` to the name of the stream and specifying an integer value. The default value is zero.

10.1.3.6 Output stream bit depth

The bit depth used for an output stream may be set (where the file format allows) by appending `_Options.nBits` to the name of the stream, and specifying the bit depth as an integer.

10.1.3.7 Output stream compression

The compression setting used for an output stream may be set (where the file format allows) by appending `_Options.Compression` to the name of the stream, and specifying an integer value. A value of zero turns off compression, while positive values give compression (how many levels of compression are available depend on the image format).

10.1.3.8 Output stream quality

For output formats using a lossy compression scheme (*e.g.* .jpg files), it is possible to specify the quality of the resulting image. There will, of course, be a trade-off between the quality and the degree of compression. Access to the quality control is provided by appending `_Options.Quality` to the name of the stream. This control takes a string value which is normally one of `"Default"`, `"Fast"`, `"Accurate"`, `"Superb"`, `"Good"`, `"Normal"`, `"Average"` or `"Low"`.

10.1.3.9 Output stream resampling

When the .dfi image format is selected, it is possible to rescale the output stream before it is saved and then reverse this rescaling when the image is subsequently read in. Typically this option is used to reduce the resolution of the saved image, but maintain its size by interpolating back to the original size before using the image again. Overall control of this is provided by appending `_Options.Resample` to the name of the output stream and specifying one of `"none"`, `"source"` or `"local"`. The first of these turns off resampling (default), whereas the second causes any resampling parameters to be inherited from the source image stream or the process that is creating the images, as appropriate. The `"local"` option provides direct control over the resampling through the additional keys described below.

The resolution of the saved image is controlled by appending `_Options.ResampleFactor` to the name of the output stream and specifying a floating point value for the relative resolution of the saved image. For example, a value of 0.5 will cause the saved image to have only ¼ of the number of pixels of the original in the file, but through interpolation the missing pixels are reconstructed when the image is read in again. The method of interpolation may also be controlled using `_Options.ResampleInterpolation` with a value of `"none"` for no interpolation (replicating pixels), `"linear"` for bi-linear interpolation and `"cubic"` for bi-cubic interpolation.

10.1.3.10 Comments in output streams

For output file formats that support comments, a comment may be specified by appending `_Options.Comments` to the name of the output stream and specifying the comment as a character string.

10.1.3.11 Leaving output streams visible

When DigiFlow is run interactively, the principle output streams are opened and left visible at the end of a process. This behaviour, however, may not be desirable when running DigiFlow from a macro.

The macro can select whether or not to leave an output stream open by defining the symbol (within the controlling compound variable or code segment) `DisplayOnExit` and setting the value to **`true`** or **`false`**, as desired. If `DisplayOnExit` is not defined, it is assumed to be **`true`**.

10.1.3.12 Deleting existing streams

To automatically delete an existing output stream at the point when the first image is written to the new stream, the stream modifier `_Options.DeleteExisting` should be added to the output stream name and set to true, *viz*:

```
stream_Options.DeleteExisting := true;
```

10.1.4 Chaining responses

As with interactive use of DigiFlow, dfc code may be used to build complex processes by chaining together simpler processes. The mechanism beneath this is a simple extension of the basic interface between dfc code and the various menu-driven processes. Whereas the example given in the previous sections had a single input specified by a file name, here we shall have a single input specified by a compound variable.

For example, if we wish to use the facility for transforming intensity (Tools: Transform: Intensity, see §5.7.2) as the input to a time average, then we could construct a dfc code:

```
dlgTrans.Input := "Test.dfm";
dlgTrans.Code := "sqrt(P)";
dlgTrans.process := "Tools_TransformIntensity";


dlgAve.Input := dlgTrans;
dlgAve.Output := "ave.pic";
dlgAve.Kind := "Arithmetic";
dlgAve.process := "Analyse_TimeAverage";

process dlgAve;
```

Here, we first constructed a compound variable for Tools: Transform Intensity, but do not set a destination for its output. We then assign this compound variable to the input of the averaging process. Obviously this same code could be written more directly without constructing the intermediate `dlgTrans` variable:

```
dlgAve.Input.Input := "Test.dfm";
dlgAve.Input.Code := "sqrt(P)";
dlgAve.Input.process := "Tools_TransformIntensity";
dlgAve.Output := "ave.pic";
dlgAve.Kind := "Arithmetic";
dlgAve.process := "Analyse_TimeAverage";

process dlgAve;
```

10.1.5 Multiple output streams

If a process, which you wish to use to provide input to a second process, produces more than one output stream, then it is necessary to select which output stream you require. This is achieved by specifying `.pipe` in the compound variable used for the input. The example below selects the YGradient output from the synthetic schlieren process, and averages this over time.

```
dlgAve := ≪
     Input := ≪
         Experiment := "Expt.dfm";
         Background := "Reference.dfi";
         Difference := "Absolute";
         CameraToTexture :=  4.0;
         ExperimentToTexture := 0.3;
         ExperimentThickness := 0.2;
         Medium := "Water";
         CoordSystem := "internal waves";
         GradientScale :=  0.1;
         DisplacementScale :=  0.1;
         DensityScale :=  1.00000;
         AutomaticInterrogation := true;
         AutomaticValidation := true;
         AutomaticMeans := true;
         process := "Analyse_SyntheticSchlierenPatternMatch";
         pipe := "YGradient";
     ≫
     Output := "ave.pic";
     Kind := "Arithmetic";
     process := "Analyse_TimeAverage";
≫;
```

```
process Analyse_TimeAverage(dlgAve);
```

10.1.6 Accessing dialogs

Sometimes it is convenient to accept user input via one of the standard process dialogs, then modify this before executing the process. This may be achieved using the **dialog** statement either as:

dlg_value := **dialog** *dlg*;

or

dlg_value := **dialog** *command*;

In the first of these, *dlg* is a compound variable with one mandatory component, `.process`, to specify the process that is being invoked. As with the **process** command, the `.process` component must be a string value. In the second option, *command* is literal text for the name of the process. For example, the following two code segments have the same effect:

```
dlg.process := "Analyse_TimeAverage";
dlg := dialog dlg;
```

and

```
dlg := dialog Analyse_TimeAverage;
```

In both the above examples the return value (here `dlg`) is a compound variable that contains the responses, including details of any nested (chained) dialogs. Note that it does not contain details of settings you have not made and will not affect the process you have selected. For example, the time averaging process returns

```
dlg.process := "Analyse_TimeAverage";
dlg.Input := "randr.dfm";
dlg.Input_Time.FromStep := 0;
dlg.Input_Time.ToStep := 20;
dlg.Input_Time.StepBy := 1;
dlg.Input_Time.FromTime := 0.0;
dlg.Input_Time.ToTime := 8.0;
dlg.Input_Time.TimeStep := 0.20;
dlg.Kind := "Arithmetic";
dlg.Output := "Average.pic";
```

Obviously, the details on the right-hand side will vary, depending on the precise options selected by the user, and the base variable (here `dlg`) is determined by the left-hand side of the assignment expression. There is also some redundancy in this information in that the time period is specified in both steps and times. In such a case, the step specification has priority.

The following example takes the returned compound variable from the time averaging process and modifies the time period for the average:

```
# Retrieve the dialog responses from the user.
dlg := dialog Analyse_TimeAverage;

# Change time period to only 1 second. Since both step and time
# specifiers are present, and we only want time, we must either
# ensure they are compatible, or remove the unwanted specifier.
# Here we shall completely replace the time specifier.
# Could either remove the variable (as here) or assign it
# some dummy value, such as dlgInput_Time := null;
destroy_variable("dlg.Input_Time”);

# Specify the new times
dlg.Input_Time.FromTime := 0.0;
dlg.Input_Time.ToTime := 1.0;

# Execute the process.
ret := process dlg;
```

If you wish to specify the initial values of the controls within the dialog, then this may be achieved using the

dlg_value := **dialog** *dlg*;

form of the command. Any valid components in *dlg* will be used to initialise the corresponding controls within the dialog.

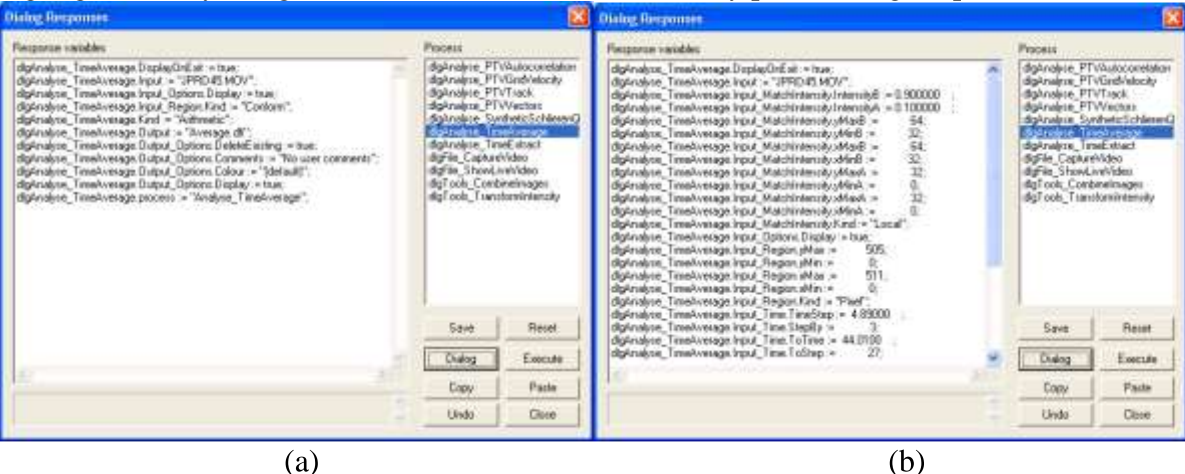
The functions `get_dialog_response(..)` and `set_dialog_response(..)` may be used to retrieve or set (respectively) the current default response for a given dialog.

10.2 Recording user input

Constructing dfc code to control a process from scratch can be time consuming and prone to error, especially when working with the more complex DigiFlow processes. To simplify matters, DigiFlow is able to record many aspects of interactive use, and convert these to dfc code.

Indeed, DigiFlow does this all the time, and records a log of user responses in the file DlgResponses.log in the directory the process was run from. This file will gradually grow with time as it accumulates more and more of the users' interactive activity. This file may be deleted without any harmful effects.

Moreover, the Edit Dialog Responses menu item (see §5.2.9) provides direct access to the latest responses for all process dialogs, and provides the ability to fire up the dialog to determine the dfc responses without initiating the process. Note, however, that the responses displayed in Edit Dialog Responses is only the minimum set required for the options selected in the dialog. For example, the various sifting options will not be included in the response unless they were selected in the dialog. Figure 148 illustrates this point. Note that the highlighted entry in figure 148b is included to remove any pre-existing output file.



(a) (b)

Figure 148: Example of Edit Dialog Responses for the same process: (a) no sifting options selected, and (b) sifting both time and space.

11 Plotting and drawing

Many of the features of DigiFlow produce graphical output. Similarly, it is often desirable for a dfc file to plot the results of its processing, or indeed for the user to plot data from a wide variety of external sources. This section describes the features within DigiFlow that support this process.

Before describing the commands controlling this process, consider the following example.

```
hD := draw_start(width:=512,height:=512,
     description:="Unspecified DigiFlow drawing");
draw_group_begin(hD,name:="Velocity");
draw_begin_vector(hD,vectorScale:=10.0,autoWidth:=false);
#Data: xFrom,yFrom,xTo,yTo
 100.0 100.0 2.0 2.0
 300.0 300.0 -1.0 1.0
end_data; #vector
draw_group_end(hD,name:="(group)" );
draw_end(hD);
hView := new_view(512,512);
view(hView,hD,erase:=true);
draw_destroy(hD);
```

Here, the drawing object, referred to by the handle hD, is created by the call to draw_start(..). The draw_group_begin(..) draw_group_end(..) pair are optional: they cause the graphics objects between to be grouped in any enhanced metafile created from this drawing. Data is plotted by specifying it should be drawn as vectors through the draw_begin_vector(..) command, then the data is enumerated. As many data lines as desired may be included, the end of the data being indicated by end_data; Completion of the drawing should be indicated by draw_end(..). The drawing object may then be rendered on the screen by first using new_view(..) to create the view window, then calling draw_on_view(..) (or equivalently a variant of the view(..) command) to do the rendering. Finally, the drawing object may be destroyed using draw_destroy(..) to free up the associated memory.

11.1 Drawing commands

There are two levels of drawing commands in DigiFlow. The main group, all of which start with draw_, provide the most flexible, versatile and powerful approach to creating graphs, plots, *etc*. However, in some situations, a simpler interface is required. To this end a second set of drawing functions is provided in DigiFlow, with names beginning plot_, which effectively act as a simplified interface to the draw_ family. This section deals with the draw_ family, whereas plot_ is dealt with in §11.3.

DigiFlow drawing commands may be subdivided into a number of groups.

Drawing initialisation is provided by draw_start(..), which returns a handle for use in all other drawing commands. Ultimately, after drawing all elements, draw_end(..) indicates the drawing is ready for rendering, after which time draw_destroy(..) may be used to tidy up the memory that was used. The rendering itself is achieved through one or more of draw_on_view(..), draw_on_file(..) and draw_on_emf(..). Note that there is a variant of the view(..) function that is directly equivalent to draw_on_view(..).

The axes for the drawing are set up using draw_set_axes(..), with the labels and tick spacing specified with draw_x_axis(..) and draw_y_axis(..). Alternate axes, independent of the coordinate system, may be specified with draw_bottom_axis(..), draw_top_axis(..), draw_left_axis(..) and draw_right_axis(..). All of these understand LaTeX-like text formatting with fully licensed copies of DigiFlow; see §11.4 for details.

Basic drawing primitives include draw_move(..), draw_line(..), draw_lineto(..), draw_mark(..) and draw_vector(..). The block data equivalents of these are draw_begin_line(..), draw_begin_lineto(..), draw_begin_mark(..) and draw_begin_vector(..), each block being terminated by end_data. In some cases it may be desirable to reduce the number of discrete points to be plotted. This can be controlled for subsequent draw commands using draw_skip_points(..).

Additional drawing primatives include draw_arc(..), draw_circle(..), draw_rectangle(..) and draw_polygon(..).

The attributes applied to drawing primitives are set by draw_line_colour(..), draw_fill_colour(..), draw_no_fill(..), draw_line_width(..), draw_line_stye(..), draw_mark_type(..) and draw_mark_size(..). Note that colours can be specified in a number of ways for draw_line_colour(..) and draw_fill_colour(..), including using colour names (*e.g.* "black" or "red"), specifying the red, green and blue components, or as an index into the current colour scheme (set by draw_colour_scheme(..)). Further colour names can also be added using add_colour(..).

Text output is provided through draw_text(..) in conjunction with draw_font(..) and draw_text_colour(..). In addition, draw_create_key(..) and draw_key_entry(..) provide a convenient method of producing a legend for a plot. All of these understand LaTeX-like formatting commands. See §11.4 for further details.

It is possible to place an image on the drawing with draw_image(..), setting the colour scheme through draw_colour_scheme(..). An intensity scale for the image (or other plotting feature) can be generated with draw_image_scale(..) for a horizontal scale, or draw_image_scale_vertical(..) for one oriented vertically. A scale can also be provided for vector elements using draw_vector_scale(..) and draw_vector_scale_vertical(..).

Grouping of drawing objects may be achieved with draw_group_begin(..) and draw_group_end(..), while plot attributes may be localised with draw_save_state(..) and draw_restore_state(..). One drawing object may be embedded within another using draw_embed_drawing(..), and the clipping area may be customised with draw_clip_box(..).

Information about a specified drawing handle can be returned to the calling dfc code using draw_get_axes(..), draw_get_font_height(..) and draw_get_status(..).

For a more complete list, and further details on these functions, refer to the dfc function help facility within DigiFlow.

11.2 The DigiFlow Drawing format

The DigiFlow Drawing format (.dfd) represents a subset of the dfc *dfc* format that contains a mixture of data representing DigiFlow results, and commands that allow DigiFlow to read the data back in to form a plot. Typically, when DigiFlow produces a .dfd file, it will also embed within it both time information, and documentation that records how the file was created. This latter information is then available through the Edit: Properties dialog described in §5.2.5.

A .dfd file contains four types of lines: comment lines, which start with a hash (#) character; drawing command lines (all valid commands start with draw_); data lines containing numeric values, and macro lines. A macro line is a line that contains other dfc code. Such lines should only be used to manipulate data for the drawing; they should have no external or permanent effect on DigiFlow. Finally, a .dfd file can contain only a single drawing object and must not attempt to display it (*e.g.* it must not call view(..)). A .dfd file may be

specified at the Open Image dialog (§4.1, 5.1.1) or at the Run Code dialog (§5.1.2) prompt. The converse, however, is not true.

DigiFlow can generate .dfd files in a number of ways. This process is automatic if the output file name is given the .dfd extension (particularly suitable for plots such as those from Analyse: Time Summarise, §5.6.1.6) or using the draw_on_file(..) function. To aid interpretation by the user or user-written programs, .dfd files created by DigiFlow always include keywords in the calls to the various drawing commands. Similarly, DigiFlow-created .dfd files do not contain macro lines.

The following illustrates a trivial .dfd file:

```
hDraw := draw_start(512,512);
draw_set_axes(hDraw,0,10,0,100);
draw_x_axis(hDraw,"$x$");
draw_y_axis(hDraw,"$y$");
draw_begin_marks(hDraw);
 0  0
 1  1
 2  4
 3  9
 5  25
 8  64
end_data;
draw_end(hDraw);
```

Note that if the .dfd file contains an error, then the part of the drawing preceding the error will still be rendered, however no error message will be generated. The .dfd code is run in its own (isolated) interpreter context and can not access variables in any dfc code that may be causing the .dfd to be processed.

The output from a .dfd file, as with any drawing command, is rendered as an Enhanced MetaFile which utilises vector graphics. It is therefore ideal for incorporation in manuscripts and may readily be converted to PostScript. Figure 149 illustrates the output from the above trivial .dfd file.

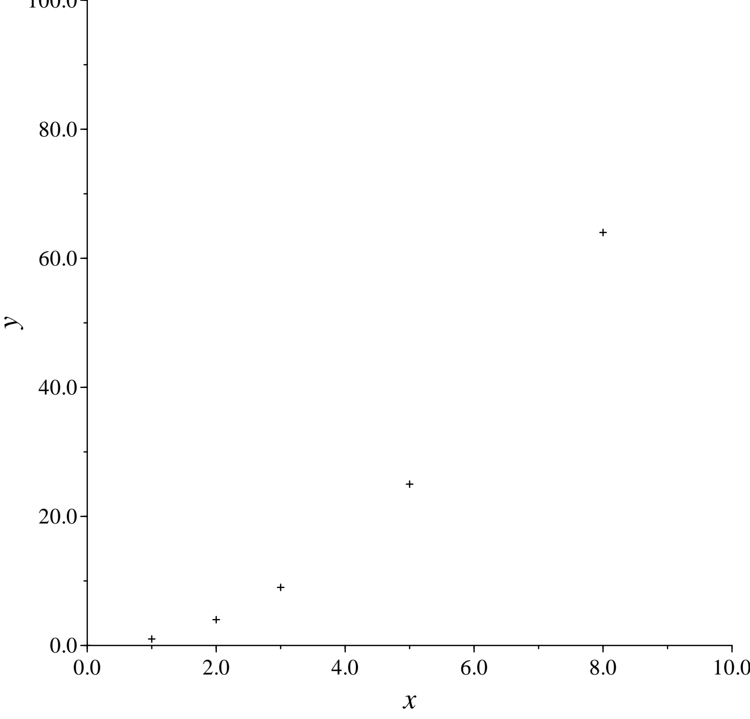


Figure 149: Example of the output from the trivial .dfd file given above.

11.3 Simple plot

The Simple Plot family of routines, which have names beginning with **plot_**, provides a simplified interface for producing basic graphs of data with a minimum of commands. In the simplest case, a single command is all that is necessary to produce a line plot or a plot showing individual data points. Consider the following example:

```
x := x_index(100)/10.0;
y := x*(x-2);
plot_line(x,y);
```

The first two lines simply define a quadratic to be plotted, and the final line takes the data in the `x` and `y` arrays and produces a line plot using the default colour and style. Further lines may be plotted, either by making the *x* and *y* arrays multidimensional, or by repeated calls to the **plot_line**(..) function. The following two code segments would produce the same results:

```
x := x_index(100,2)/10.0;
y[:,0] := x[:,0]*(x[:,0]-2);
y[:,1] := x[:,1];
plot_line(x,y);
```

or

```
x := x_index(100)/10.0;
y := x*(x-2);
plot_line(x,y);
y := x;
plot_line(x,y);
```

The limiting values for the axes are determined automatically as the extremes in the specified data. However, these may be overridden by calling **plot_axes**(..) giving the desired limiting values. Similarly, the default titles for the axes may be overridden by **plot_titles**(..).

If calling any of these plot functions directly from dfc code, then DigiFlow will automatically display the plot by creating an appropriate view. Subsequent calls to Simple Plot functions will cause that view to be updated. If, however, the call is made within one of the menu options, then DigiFlow will display the plot only when appropriate and not update it for each and every call.

Many of the Simple Plot functions (which are implemented dfc macros to the **draw_** series of functions) have optional parameters to provide greater flexibility. For example, the **plot_line**(..) function used above includes an optional style parameter that can be used to select between lines and points, and an optional colour parameter. Each of these may be supplied either as a single value, or (when multiple sets of data are being plotted simultaneously) as a list of values.

The Simple Plot family of functions includes:

`plot(..)`	Plots a series of points.
`plot_line(..)`	Plots a line.
`plot_vectors(..)`	Produce a vector map.
`plot_image(..)`	Add an image to a plot.
`plot_fit(..)`	Perform and plot a least squares fit.
`plot_axes(..)`	Explicitly specify the limits for the axes.
`plot_axis_types(..)`	Specify the types of axis (linear or logarithmic).
`plot_titles(..)`	Specify the titles for the axes.
`plot_new(..)`	Start a new plot.
`plot_view_handle(..)`	Return the handle for the view window displaying the plot.

plot_drawing(..)	Return a handle to the base plot. This does not include information about the axes.
plot_update_view(..)	Cause the view to be updated.

Most of the Simple Plot family of functions return a handle to the drawing that is being displayed. This handle may be passed to any of the draw_ family of functions to provide more advanced control over the appearance of the plot. The reason for this is related to the manner in which the limits on the axes are automatically determined.

If you wish to add details to a plot using the draw_ family and wish this to be retained after subsequent calls to the Simple Plot functions, then recover a handle to the base drawing using plot_drawing(..). (Alternatively, passing a null or the integer value zero as the plot handle to many of the draw_ family will have the same effect.) However, adding any further details to the plot using one of the simple plot functions will cause this additional information to be discarded. Note the base drawing does not have the limits for its axes set and so should not be viewed directly. After modifying the drawing using the handle obtained from plot_drawing(..) you should either call another of the Simple Plot family of functions or call plot_update_view(..).

11.4 Text

With fully licensed copies of DigiFlow, text added to drawings and plots, whether via the draw_... or plot_... set of functions, understands simple LaTeX-like formatting commands within the specified strings. For example, the string `"Dimensionless height $\big(\frac{h}/{\alpha^2H_0}\big)$"` if specified in draw_text(..) would produce the label

$$\text{Dimensionless height}\left(\frac{h}{\alpha^2 H_0}\right).$$

Although DigiFlow does not understand the full range of LaTeX commands and macros, it can interpret those most likely to be of use in figures and graphs. The file DigiFlow_Latex.dfc defines the majority of macros, using a set of more primitive macros built in to DigiFlow's LaTeX-like interpreter. Consult the dfc documentation on draw_install_latex_macro(..) for details of how to define macros. The LaTeX interpretation may be turned off using draw_enable_latex(..).

Text size may be changed within a given string using the LaTeX commands \tiny, \small, \subscriptsize, \footnotesize, \normalsize, \large, \Large, \LARGE, \huge and \HUGE. However, it is normally more convenient to change the size using draw_font(..) when outputting text through draw_text(..). On the other hand, draw_font(..) does not change the size of elements such as the labels and scale for the axes. The command draw_set_base_scales(..) changes the base scale for all text, and can thus be used to change the scale for axes, *etc.*

The LaTeX macros understood by DigiFlow are listed below.

11.5 LaTeX macros

```
_
^
\!
\#
\$
\%
\&
\,
\2dots
\:
\;
\Alpha
\BIG
\BIG
\Beta
\Big
\Big
\Chi
\Delta
\Downarrow
\Epsilon
\Eta
\Gamma
\HUGE
\HUGE
\Im
\Iota
```

```
\Kappa
\LARGE
\LARGE
\Lambda
\Large
\Large
\Leftarrow
\Leftrightarrow
\Mu
\Nu
\O
\Omega
\P
\Phi
\Pi
\Psi
\Re
\Rho
\Rightarrow
\S
\Sigma
\Tau
\Theta
\Uparrow
\Upsilon
\Varphi
\Varpi
\Xi
\Zeta
\\
\^
\_
\aleph
\alpha
\angle
\approx
\backslash
\bar
\bar
\beta
\bf
\bf
\big
\big
\bigsizes
\bullet
\calc
\cdot
\chi
\circ
\copyright
\cos
\cosh
\currentx
\currenty
\dagger
\ddagger
\dddot
\dddot
\ddot
\ddot
\delta
\div
\dot
\dot
\dots
\downarrow
\ell
\emdash
\endash
\epsilon
\equals
\equalsspace
\equiv
\eta
\euro
\exists
\exp
\footnotesize
\footnotesize
\forall
\frac
\frac
\gamma
\ge
\geq
\gg
\gotox
\gotox0
\gotoxy
\gotoxy0
\gotoy
\gotoy0
\hat
\hat
\huge
\huge
\hyphen
\in
\infty
\int
\iota
\it
\it
\kappa
\lambda
\langle
\large
\large
\lbrace
\lbrack
\le
\left
\left
\leftarrow
\leftrightarrow
\leftszbracket
\leq
\ll
\ln
\log
\mark
\mathbf
\mathbf
\mathit
\mathit
\mathrm
\mathrm
\mathsprime
\minus
\minusspace
\moveto
\mp
\mu
\nabla
\ne
\neq
\normalsize
\normalsize
\notin
\nu
\nudge
\o
\omega
\oplus
\oslash
\otimes
\overchar
\overcharoffset
\partial
\phantom
\phi
\pi
\plus
\plusspace
\pm
\pop
\popcalc
\pounds
\prime
\print
\prod
\propto
\psi
\push
\pushcalc
\qquad
\quad
\querybottom
\queryglyphbottom
\queryglyphheight
\queryglyphleft
\queryglyphright
\queryglyphtop
\queryglyphwidth
\queryheight
\queryheightline
\queryheighttotal
\queryleft
\queryright
\querytop
\querywidth
\querywidthline
\rangle
\rbrace
\rbrack
\rho
\right
\right
\rightarrow
\rightszbracket
\rm
\rm
\rmove
\rule
\scriptsize
\scriptsize
\sigma
```

```
\sim
\simeq
\sin
\sinh
\size
\sizeto
\small
\small
\sqrt
\sqrt
\subscript
\subscriptIt
\subset
\subseteq
\sum
\superscript
\superscriptIt
\supset
\supseteq
\surd
\tan
\tanh
\tau
\textbf
\textbf
\textit
\textit
\textnormal
\textnormal
\textrm
\textrm
\therefore
\theta
\tilde
\tilde
\times
\tiny
\tiny
\underline
\underline
\uparrow
\upsilon
\varphi
\varpi
\vcentre
\wedge
\widthheight
\wp
\xi
\yen
\zeta
\{
\}
\~
```

12 Image file formats

In this section, some of the key image file formats supported by DigiFlow are described.

Which image file is most appropriate depends in part on the intended use of the final images, and in part on the amount of disk space available. For all but the simplest processing operations, use of a ‘lossless’ integer image format (all industry standard formats are integer based, most using 8-bit representations of the intensity) will introduce losses through the quantisation of a floating point value into an integer domain. The .dfi format introduced in DigiFlow (see §12.7) overcomes this problem by storing the images in a floating point format (either 32 or 64 bit, although it can also store as 8 bit); the cost is a greatly increased storage requirement.

In environments where DigiFlow is being used alongside DigImage, use of the older DigImage formats (.pic and .mov) is recommended to facilitate exchange of information between these two applications. Indeed, the DigImage .mov format (now also known as .dfm) still plays a central role as the initial format when capturing video from a supported camera (see §5.1.5.2).

When DigiFlow is used in conjunction with other image processing packages, or with painting programmes, then use of standard formats such as .bmp and .tif is recommended. With vector drawing packages, then the enhanced metafile format (.emf) is normally the best option, although the older style Windows metafile format (.wmf) may also be used. Note that a Matlab macro is available for reading uncompressed .dfi and .dfd files into Matlab.

For incorporating images or graphics into documents, the best results may be achieved with encapsulated PostScript (.eps), if your printer supports this. If you do not use a PostScript printer, then use standard formats such as .bmp, .tif .jpg, .emf or .wmf.

12.1 Windows bitmap files (.bmp)

The .bmp format is central to the design of Windows, and offers a universal but inefficient standard for simple images. There are a number of variants of .bmp files, and DigiFlow can read the most common variants (including 24-bit colour files). DigiFlow will normally, however, only create 8-bit uncompressed files.

See standard Windows documentation for further details.

12.2 TIFF files (.tif)

The Tagged Image Format File (TIFF) is one of the oldest commonly used image formats. It offers great flexibility, but also great difficulty as there are so many variants.

DigiFlow can read a wide variety of TIFF files using the FreeImage library.

See standard TIFF documentation for further details.

12.3 GIF files (.gif)

For a long time, the .gif format was widely used, providing an effective lossless compression for a broad variety of images. However, in the late 1990s, Compuserve, who owned the intellectual property rights for the GIF format, decided to charge a royalty. Since then, the use of GIF has declined sharply, and many applications that once supported GIF no longer do so. More recently the original patent on the GIF format expired, and DigiFlow is again able to offer comprehensive support for GIF.

12.4 Enhanced metafiles (.emf)

Enhanced metafiles (.emf) are a standard Windows format, intended primarily for vector graphics, but also supporting bit mapped images. Most Windows-based packages support

embedding and/or linking with these files to provide graphical content. DigiFlow can both read and write .emf files, although they should not normally be used as an image source.

12.5 Windows metafiles (.wmf)

Windows metafiles (.wmf) are a standard Windows format, dating from the days when Windows was only 16 bits. This format is intended primarily for vector graphics, but also supports bit mapped images. Most Windows-based packages support embedding and/or linking with these files to provide graphical content. DigiFlow can both read and write .wmf files, although they should not normally be used as an image source. In general the newer Enhanced metafile format (.emf) should be used in preference (see §12.4).

12.6 Encapsulated PostScript (.eps)

Encapsulated PostScript is fundamentally an output format, intended for inclusion in documents that will be printed using a PostScript printer. DigiFlow does not provide the ability to read data from .eps files, although may be able to use GhostScript to convert .eps into a format it can read (see §2.2.2). Encapsulated PostScript typically provides the best quality output for a printed document and may be imported readily into standard word processors and text formatting languages such as LaTeX.

12.7 DigiFlow floating point image format (.dfi)

The purpose of this format is to store image and related data without significant loss of precision. Indeed for most elements of the format, there are both four-byte and eight-byte floating point representations as an option, in recognition that DigiFlow internally uses an eight byte floating point representation, but often a four byte representation is sufficient and is more compact. For compactness, a single-byte image format is also available.

A tagged format is used to distinguish the different data objects within the file, and the four and eight byte variants simply have different tags. However, for the convenience of the user, DigiFlow uses a single extension, .dfi, for all of these, with the Options button for the output stream allowing selection of the desired variant (32 or 64 bits). Additionally, the .dfi format can store the image data in a single byte integer (8 bit) format.

Overall, the structure of the .dfi files may represented as

header
tag
object data
tag
object data
…

Each of these elements is described in turn below.

12.7.1 Header

The file header has been kept as simple as possible while still conveying the essential data.

Field	Data type	Description
idFormat	`Character(32)`	Contains the text "Tagged `floating point image file`" (excluding quotes). This is used by DigiFlow to identify the file type.
Version	`Integer (4)`	Version number. Here must equal zero.

12.7.2 Tag

Each data object is preceded by a tag that indicates the type of object and the size of the object.

Field	Data type	Description
DataType	`Integer (4)`	The type of data contained in the ext object.
nBytes	`Integer (4)`	The number of bytes of data used to represent the object.

Valid tags and the associated data objects are described in the following subsections. Note that the quoted value is in hexadecimal (base 16), as indicated by the hash (#) in front of the *DataType* value.

12.7.3 8 bit image (DataType = #1001)

This data object contains an image using an eight-bit (single byte) integer representation. Note $nBytes = 8 + nx*ny$.

Field	Data type	Description
nx	`Integer (4)`	The width of the image, in pixels.
ny	`Integer (4)`	The height of the image, in pixels.
$c(0{:}nx-1, 0{:}ny-1)$	`Integer (1)`	The pixel intensities. The first index is across from left to right, and the second is up from bottom to top. Storage in the file is ordered $c(0,0)$, $c(1,0)$,…

12.7.4 8 bit multi-plane image (DataType = #11001)

This data object contains a multi-plane image using an eight-bit (single-byte) integer representation. Note $nBytes = 12 + nx*ny*nz$.

Field	Data type	Description
nx	`Integer (4)`	The width of the image, in pixels.
ny	`Integer (4)`	The height of the image, in pixels.
nz	`Integer (4)`	The number of pixel planes.
$c(0{:}nx-1, 0{:}ny-1,0{:}nz-1)$	`Integer (1)`	The pixel intensities. The first index is across from left to right, and the second is up from bottom to top. Storage in the file is ordered $c(0,0,0)$, $r(1,0,0)$,…

12.7.5 Compressed 8 bit image (DataType = #12001)

This data object contains an image using an eight-bit (single byte) integer representation compressed using ZLib. Note that *nBytes* varies depending on the efficiency of the compression, and that nine different levels of compression are available through the output options setting (see §4.4).

Field	Data type	Description
nx	`Integer (4)`	The width of the image, in pixels.
ny	`Integer (4)`	The height of the image, in pixels.
nz	`Integer (4)`	The number of image planes.
szCompressed	`Integer (4)`	The number of bytes of compressed data.
c(0:*szCompressed*–1)	`Integer (1)`	The compressed pixel data. When reading an image, this should be fed to the ZLib function `uncompress` to recover the original image. When writing an image, the ZLib function `compress(..)` or `compress2(..)` should be used.

12.7.6 32 bit image (DataType = #1004)

This data object contains an image using a four-byte floating point representation. Note $nBytes = 8 + 4*nx*ny$.

Field	Data type	Description
nx	`Integer (4)`	The width of the image, in pixels.
ny	`Integer (4)`	The height of the image, in pixels.
$r(0{:}nx-1, 0{:}ny-1)$	`Real (4)`	The pixel intensities. The first index is across from left to right, and the second is up from bottom to top. Storage in the file is ordered $r(0,0)$, $r(1,0)$,…

12.7.7 32 bit multi-plane image (DataType = #11004)

This data object contains a multi-plane image using a four-byte floating point representation. Note $nBytes = 12 + 4*nx*ny*nz$.

Field	Data type	Description
nx	`Integer (4)`	The width of the image, in pixels.
ny	`Integer (4)`	The height of the image, in pixels.
nz	`Integer (4)`	The number of pixel planes.
$r(0{:}nx-1, 0{:}ny-1,0{:}nz-1)$	`Real (4)`	The pixel intensities. The first index is across from left to right, and the second is up from bottom to top. Storage in the file is ordered $r(0,0,0)$, $r(1,0,0)$,…

12.7.8 Compressed 32 bit image (DataType = #12004)

This data object contains an image using an four-byte floating point representation, compressed using ZLib. Note that *nBytes* varies depending on the efficiency of the compression, and that nine different levels of compression are available through the output options setting (see §4.4).

Field	Data type	Description
nx	`Integer (4)`	The width of the image, in pixels.
ny	`Integer (4)`	The height of the image, in pixels.
nz	`Integer (4)`	The number of image planes.
szCompressed	`Integer (4)`	The number of bytes of compressed data.
c(0:*szCompressed*–1)	`Integer (1)`	The compressed pixel data. When reading an image, this should be fed to the ZLib function `uncompress(..)` to recover the original four-byte floating point image. When writing an image, the ZLib function `compress(..)` or `compress2(..)` should be used.

12.7.9 64 bit image (DataType = #1008)

This data object contains an image using an eight-byte floating point representation. Note $nBytes = 8 + 8*nx*ny$.

Field	Data type	Description
nx	`Integer (4)`	The width of the image, in pixels.
ny	`Integer (4)`	The height of the image, in pixels.
$r(0:nx-1, 0:ny-1)$	`Real (8)`	The pixel intensities. The first index is across from left to right, and the second is up from bottom to top. Storage in the file is ordered $r(0,0)$, $r(1,0)$,…

12.7.10 64 bit multi-plane image (DataType = #11008)

This data object contains a multi-plane image using a eight-byte floating point representation. Note $nBytes = 12 + 8*nx*ny*nz$.

Field	Data type	Description
nx	`Integer (4)`	The width of the image, in pixels.
ny	`Integer (4)`	The height of the image, in pixels.
nz	`Integer (4)`	The number of pixel planes.

r(0:nx–1, 0:ny–1,0:nz-1)	`Real (8)`	The pixel intensities. The first index is across from left to right, and the second is up from bottom to top. Storage in the file is ordered r(0,0,0), r(1,0,0),…

12.7.11 Compressed 64 bit image (DataType = #12008)

This data object contains an image using an eight-byte floating point representation, compressed using ZLib. Note that *nBytes* varies depending on the efficiency of the compression, and that nine different levels of compression are available through the output options setting (see §4.4).

Field	Data type	Description
nx	`Integer (4)`	The width of the image, in pixels.
ny	`Integer (4)`	The height of the image, in pixels.
nz	`Integer (4)`	The number of image planes.
szCompressed	`Integer (4)`	The number of bytes of compressed data.
c(0:*szCompressed*–1)	`Integer (1)`	The compressed pixel data. When reading an image, this should be fed to the ZLib function `uncompress` to recover the original eight-byte floating point image. When writing an image, the ZLib function `compress(..)` or `compress2(..)` should be used.

12.7.12 32 bit range (DataType = #1014)

This data object specifies the range of image values that can be displayed. Note *nBytes* = 8. This tag should be located after the image to which it applies.

Field	Data type	Description
rBlack	`Real (4)`	The intensity (value of *r*) that will be displayed as "black".
*rWhite*0	`Real (4)`	The intensity (value of *r*) that will be displayed as "white".

12.7.13 64 bit range (DataType = #1018)

This data object specifies the range of image values that can be displayed. Note *nBytes* = 16. This tag should be located after the image to which it applies.

Field	Data type	Description
rBlack	`Real (8)`	The intensity (value of *r*) that will be displayed as "black".
*rWhite*0	`Real (8)`	The intensity (value of *r*) that will be displayed as "white".

12.7.14 Rescale image (DataType = #1100)

This data object requests that the image is rescaled after being read in. Typically this is used to render a low resolution image to a given size.

Field	Data type	Description
nxWant	`Integer (4)`	The desired width after rescaling. Typically this is the width of the original image prior to resampling and saving.
nyWant	`Integer (4)`	The desired height after rescaling. Typically this is the height of the original image prior to resampling and saving.
method	`Integer (4)`	Method of rescaling image: 0 Constant 1 Bilinear 2 Bicubic 3 Natural spline 4 Cubic b-spline 5 Quintic b-spline.

12.7.15 Rescale image rectangle (DataType = #1101)

This data object requests that the image is rescaled after being read in. Typically this is used to render a low resolution image to a given size. In contrast with DataType #1100, this data object allows the input image to coincide with a particular rectangle within the resultant image. This feature is designed primarily to allow PIV velocity fields to be rescaled back to the original size and location if saved in a compact format with vectors only at the interrogation points.

Field	Data type	Description
nxWant	`Integer (4)`	The desired width after rescaling. Typically this is the width of the original image prior to resampling and saving.
nyWant	`Integer (4)`	The desired height after rescaling. Typically this is the height of the original image prior to resampling and saving.
method	`Integer (4)`	Method of rescaling image: 0 Constant 1 Bilinear 2 Bicubic 3 Natural spline 4 Cubic b-spline 5 Quintic b-spline.

useRectangle	`Integer (4)`	Indicates if *Rectangle* is to be used. If zero (false), then the net effect of this tag is the same as for #1100.
Rectangle.xMin	`Integer (4)`	The left side of the rescaled source image in the output image.
Rectangle.yMin	`Integer (4)`	The bottom of the rescaled source image in the output image.
Rectangle.xMax	`Integer (4)`	The right side of the rescaled source image in the output image.
Rectangle.yMax	`Integer (4)`	The top of the rescaled source image in the output image.

12.7.16 Colour scheme (DataType = #2000)

This data object contains the colour scheme that the image should be displayed with by default. Note that *nBytes* = 768. This tag should be located after the image to which it applies.

Field	**Data type**	**Description**
red(0:255)	`Integer (1)`	Defines the red component of the colour scheme to be used to display the image.
green(0:255)	`Integer (1)`	Defines the green component of the colour scheme to be used to display the image.
blue(0:255)	`Integer (1)`	Defines the blue component of the colour scheme to be used to display the image.

12.7.17 Colour scheme name (DataType = #2001)

This data object contains the name of a colour scheme. If the name is recognised by DigiFlow, then the corresponding colour scheme will be used. Note that *nBytes* is 64.

Field	**Data type**	**Description**
name	`Character (64)`	The name of the colour scheme to be used.

12.7.18 Colour scheme name variable (DataType = #2002)

This data object contains the name of a colour scheme. If the name is recognised by DigiFlow, then the corresponding colour scheme will be used. Note that *nBytes* is 4 plus the length of the name.

Field	**Data type**	**Description**
iLen	`Integer (4)`	The length of the name string.
name	`Character (iLen)`	The name of the colour scheme to be used.

12.7.19 Description (DataType = #3000)

This data object contains a 512 character description. Note *nBytes* = 512.

Field	Data type	Description
Descr	`Character (512)`	A description.

12.7.20 User comments (DataType = #3001)

This object contains user comments about the process that created the file.

Field	Data type	Description
nBytes	`Integer (4)`	The length of the description in bytes (characters).
Descr	`Character (nBytes)`	The description.

12.7.21 Creating process (DataType = #3002)

This object contains information about the process that created the file. Typically this is a copy of the dialog responses for the controlling process.

Field	Data type	Description
nBytes	`Integer (4)`	The length of the description in bytes (characters).
Descr	`Character (nBytes)`	The description.

12.7.22 Creator details (DataType = #3003)

Records details of the creator. Note, *nBytes* = 304.

Field	Data type	Description
DigiFlow	`Character (32)`	The DigiFlow version.
buildDate	`Character (16)`	The build date.
licenceType	`Character (16)`	The type of licence.
nameUser	`Character (32)`	The name of the user.
nameComputer	`Character (32)`	The name of the computer.
nameDomain	`Character (32)`	The name of the Windows domain.
guidUser	`Character (32)`	The user GUID.
macAddress(0:3)	`4 lots of Character (12)`	Mac address of creating computer.
ipAddress(0:3)	`4 lots of Character (16)`	IP address of creating computer.

12.7.23 Image time (DataType = #3018)

This data object contains time information. Note *nBytes* = 28.

Field	Data type	Description
iFrame	`Integer (4)`	Ordinal position of the frame.
Reserved	`Integer (4)`	Reserved.
Time	`Real (8)`	The time for the image.
tStep	`Real (8)`	The size of the time steps in the sequence the image belongs to.
tFirst	`Real (8)`	The time for the first frame in the sequence (*i.e. iFrame* = 0)

12.7.24 Image coordinates (DataType = #4008)

This data object gives information on the coordinate system if this is not a standard one.

Field	Data type	Description
Kind	`Integer (4)`	The type of coorindates stored here: 0 None 1 Approximation to world 2 Custom
xWorldPerPixel	`Real (8)`	Number of world units per pixel.
yWorldPerPixel	`Real (8)`	Number of world units per pixel.
xOriginWorld	`Real (8)`	The world origin of the image.
yOriginWorld	`Real (8)`	The world origin of the image.
xUnits	`Character (16)`	The name of the world units.
yUnits	`Character (16)`	The name of the world units.
OriginalName	`Character (64)`	The name of the coordinate system on which this was based. If none, then “`(none)`”.

12.7.25 Image plane details (DataType = #4108)

This data object gives details of the contents of individual image planes when there is more than one.

Field	Data type	Description
nPlanes	`Integer (4)`	The number of planes of data contained in this field.
Contains0	`Integer (4)`	Indicates the type of data in image plane 0: #000 None #001 Greyscale #002 Red #003 Green #004 Blue #101 xCoordinate #102 yCoordinate #103 zCoordinate #201 xVector (u) #202 yVector (v) #203 zVector (w)
Descr0	`Character (32)`	Text description or name for bit plane.
ParamA0	`Real (8)`	First parameter
ParamB0	`Real (8)`	Second parameter
ParamC0	`Real (8)`	Third parameter
ParamD0	`Real (8)`	Fourth parameter
Contains1	`Integer (4)`	As for *Contains0*, but for

		second image plane.
Descr1	`Character (32)`	As for *Descr0*, but for second image plane.
…		

12.8 DigiFlow Particle tracking format

The format used for DigiFlow .dft particle tracking files shares some elements in common with .dfi files. Both use the same general tagged structure. However, while .dfi files are aimed primarily at containing rasterised images, .dft files are designed to store information about particles located in an image, ant the relationship between these particles and the neighbouring images in a time sequence.

Further information on the format of these files is available on request.

12.9 DigiFlow pixel data format (.dfp)

The .dfp format is a simple plain text format intended for direct use by other programs, or to be imported into spreadsheets, *etc.* The first line contains the width and height of the image (in pixels). Optionally, the number of data planes can also be specified Subsequent lines each contain the pixel indices (*i*,*j*) and a floating point representation of the intensity. For a simple image, with only a single plane of data, the format is therefore

```
nx ny
i j pix_ij
i j pix_ij
…
```

With more than one plane of data, the format is

```
nx ny nz
i j pix_ij0 pix_ij1 … pix_ijk
i j pix_ij0 pix_ij1 … pix_ijk
…
```

This choice of format is motivated by providing a compact, readable output for velocity fields, rgb full-colour images, *etc.* For example, with an image containing both velocity and vorticity, the output format would be

```
nx ny 3
i j u v vort
i j u v vort
…
```

while for a full colour image it would be

```
nx ny 3
i j red green blue
i j red green blue
…
```

When DigiFlow creates a .dfp file, it contains all the image data, ordered from left to right, then from bottom to top. However, DigiFlow does not care on the order of the pixel data when reading a .dfp file, and the file need not contain all valid combinations of the pixel indices. Note that the first index gives the column number (from 0 at the left to *with*−1 at the right), while the second determines the row number (from 0 at the bottom, to *height*−1 at the top).

12.10 DigiFlow drawing format (.dfd)

The DigiFlow drawing format (.dfp files) can store image as well as vector graphic data. These files may be read as well as written, although they are not recommended for simple raster images. For further information, refer to the information on the format of these files in §11.2.

12.11 DigiFlow archive format (.dfa)

The DigiFlow archive file format (.dfa files) do not themselves store images. Rather, they provide a container for information describing the contents of a sequence of image files, and provide a way of specifying the sequence.

If a .dfa file is generated for a sequence of images image000.png, image001.png, … image732.png, then the name of the .dfa file will be image###.png.dfa. Note that this name comprises the manner in which you would specify the sequence to DigiFlow (image###.png) but with the .dfa appended.

The archive file itself contains dfc code specifying variables that describe the image sequence. For example, generating the above image###.png sequence using File: Edit Stream would produce

```
# image###.png.dfa
# ##### Begin
# DigiFlow Archive File for: image###.png
File := "image###.png";
SelectorKind := 3;
IndexPtr := 5;
nDigits := 2;
FileType := 21;
nx := 512;
ny := 512;
nz := 1;
Time.LocateWith := 1;
Time.fFirst := 0;
Time.fLast := 732;
Time.tFirst := 0.0;
Time.tLast := 361.00000000000;
Time.tStep := 0.50000000000000;
Comments.UserComments := "No user comments";
Comments.CreatingProcess := {dlgFile_EditStream.DirectCopy :=
     true;
dlgFile_EditStream.DisplayOnExit := true;
dlgFile_EditStream.Input := "JPRD45.MOV";
dlgFile_EditStream.Input_Options.Display := true;
dlgFile_EditStream.Input_Options.UseArchive := true;
dlgFile_EditStream.Input_Region.Kind := "All";
dlgFile_EditStream.Output := "junk##.png";
dlgFile_EditStream.Output_Options.DeleteExisting := true;
dlgFile_EditStream.Output_Options.Compression :=            0;
dlgFile_EditStream.Output_Options.TrueColour := false;
dlgFile_EditStream.Output_Options.Colour := "(default)";
dlgFile_EditStream.Output_Options.Display := true;
dlgFile_EditStream.Output_Options.UseArchive := true;
dlgFile_EditStream.ReviewCapture := false;
dlgFile_EditStream.process := "File_EditStream";

# process dlgFile_EditStream;
};
Creator.DigiFlow := "DigiFlow v3.4.0 (ivf)";
Creator.Licence := "commercial";
Creator.UserName := "sd103";
Creator.ComputerName := "WETA";
Creator.UserDomain := "WETA";
Creator.UserNumber := "{…}";
Creator.MacAddress0 := "…";
Creator.MacAddress1 := "…";
Creator.MacAddress2 := "";
Creator.MacAddress3 := "";
Creator.IPAddress0 := "127.0.0.1";
Creator.IPAddress1 := "192.168.1.2";
Creator.IPAddress2 := "192.168.56.1";
```

```
Creator.IPAddress3 := "";
Coord.Kind := 0;
Coord.xWorldPerPixel := 0.0;
Coord.yWorldPerPixel := 0.0;
Coord.xOriginWorld := 0.0;
Coord.yOriginWorld := 0.0;
Coord.xUnits := "";
Coord.yUnits := "";
Coord.OriginalName := "";
Appearance.rBlack := 0.0;
Appearance.rWhite := 1.00000000000000;
 Appearance.LUT.Red:=[ 0.0000 … 1.0000 1.0000];
 Appearance.LUT.Green:=[ 0.0000 … 0.9608 0.9804];
 Appearance.LUT.Blue:=[ 0.0000 … 0.9608 0.9804];
Appearance.ColourScheme.Name := "(default)";
Appearance.DisplayAs := 1;
Appearance.TopDown :=false;
Resample.Kind := 0;
# # Tue May 29 21:32:57 2012
# ##### End
# junk##.png.dfa
```

Note that some of the entries in the .dfa file above have been shortened or removed (and replaced by an elpsis) to aid clarity. The details stored here include what can be determined using **read_image_details**(..).

If you attempt to open a .dfa file using a function intended to read an image, then the corresponding image or image stream will be opened (as specified in the `File` variable), but key details will be taken from the .dfa file.

12.12 DigImage raw format (.pic)

This format, developed originally for DigImage, is the simplest supported by DigiFlow. It may be both read and written.

Field	Data type	Description
ni	`Integer (2)`	Number of rows in the image.
nj	`Integer (2)`	Number of columns in the image.
iPixel(0:*nj*–1,0:*ni*–1)	`Byte (1)`	Array of un-signed image intensities, ordered across (first index) then down (second index) from the top left.
iOLUT	`Integer (2), optional`	The DigImage output look up table giving the colour scheme.
Red(0:255)	`Integer (1), optional`	Defines the red component of the colour scheme to be used to display the image. This entry is optional if and only if *nChannels* = 1.
Green(0:255)	`Integer (1), optional`	Defines the green component of the colour scheme to be used to display the image. This entry is optional if and only if *nChannels* = 1.
Blue(0:255)	`Integer (1), optional`	Defines the blue component

		of the colour scheme to be used to display the image. This entry is optional if and only if *nChannels* = 1.
*iw*0	`Integer (2), optional`	The location of the top of the window saved in the file.
*iw*1	`Integer (2), optional`	The location of the bottom of the window saved in the file.
*jw*0	`Integer (2), optional`	The location of the left of the window saved in the file.
jw1	`Integer (2), optional`	The location of the right of the window saved in the file.

12.13 DigImage compressed format (.pic)

The compressed version of the DigImage file format was developed to allow efficient compression using a hybrid adaptive run-length encoding scheme based on individual bit planes. The degree of compression achieved depends on the structure of the image. Although DigiFlow is able to read these files, it does not provide a user interface to allow them to be created.

Note that the image is stored top-down.

Bytes	**Data type**	**Description**
0-1	`Integer (2)`	Always zero to distinguish from uncompressed format.
2-3	`Integer (2)`	–*nBitPlanes*, indicating the number of bit planes stored in the file.
4-5	`Integer (2)`	The height of the image in pixels.
6-7	`Integer (2)`	The width of the image in pixels.
5	`Integer (1)`	Indicates the type of encoding used in the following bytes: Bit7 If set, then run-length encode, otherwise bit-image encoding. Bit6 If bit7=1, then bit6 set indicates the length of the run is given by bits 0-4 in conjunction with the following byte. If bit6 is clear, then the run length is given only by bits 0-4. If bit7=0, then bit6 set indicates the number of BYTEs specified by a bit-image is given by bits 0-4 and the following byte; if clear and bit5 is set, then only bits 0-4 are used to give the number of BYTES in the bit-image. However, if bit5 is clear, then bits 0-4 are themselves a bit-image. bit5 If bit7=1, then this bit indicates whether the corresponding bit plane is set or clear in the run. If bit7=0, then this bit indicates whether bits 0-4 are used as (part of) the length of the bit-image, or the bit-image itself (clear). bits0-4 Used in giving the length of the run length or bit-image, or as part of the bit-image (bits 5,6&7 all clear).
6	`Integer (1)`	If bit6 of byte4 is set, then this byte is used in specifying the length of the run or bit-image. If bit 6 of byte 4 is clear, then this is the first byte of the encoding segment (if previous byte was run-length), or part of

		the bit-image.
7		This could be part of the bit-image specified by bytes 5 & 6, or a new key code similar to 5, *etc.*
…		Repeat run-length and/or bit-image encoded segments, bit plane by bit plane, until all image data has been processed.
	`Integer (2)`	*iOLUT* Optional specification of the logical output look up table number within DigImage (not used by DigiFlow).
Red(0:255)	`Integer (1)`	Defines the red component of the colour scheme to be used to display the image.
Green(0:255)	`Integer (1)`	Defines the green component of the colour scheme to be used to display the image.
Blue(0:255)	`Integer (1)`	Defines the blue component of the colour scheme to be used to display the image.
iw0	`Integer (2)`	The top of the source window saved in this file.
iw1	`Integer (2)`	The bottom of the source window saved in this file.
jw0	`Integer (2)`	The left of the source window saved in this file.
jw1	`Integer (2)`	The right of the source window saved in this file.

12.14 DigImage movie format(.mov or .dfm)

The DigImage movie format is of central importance for sharing image sequences between DigiFlow and the earlier DigImage. It also provides a computationally efficient medium for storing sequences of 8-bit images of any resolution. The images are stored top-down, and the file header contains an index of their location within the file.

```
C= DigImage Movie Genearal Header Information                          =
C=      Size      Name                Description                      =
C=        8       FileOwner           Contains the text "DigImage"     =
C=        8       Version             Contains the DigImage version    =
C=                                    string.                          =
C=        4       iPtrHistoryHeader   Points to the location of the    =
C=                                    history header block.            =
C=        16      FileType            The type of file. Terminated by  =
C=                                    <CR>.                            =
C=        220     Comments            Comments, terminated by <FF>.    =
C=                                    Not mapped on to iGeneralHeader. =
C= History Header Information                                          =
C=        4       iPtrPrivateHeader   Points to the location of the    =
C=                                    header for this file type        =
C=        4       iDummy              Unused.                          =
C=        8       CreatedBy           The program which created the    =
C=                                    file. Normally "DigImage".       =
C=        8       Version             The version of the program which =
C=                                    created the file.                =
C=        16      CreatedUser         The name of the user who created =
C=                                    the file                         =
C=        64      CreatedName         The original name of the file    =
C=        8       CreatedDate         The date the file was created    =
C=        8       CreatedTime         The time at which the file was   =
C=                                    created                          =
C=        16      ModifiedUser        The name of the user who         =
C=                                    modified the file                =
C=        64      ModifiedName        The name of the file when it was =
C=                                    last modified                    =
C=        8       ModifiedDate        The date the file was last       =
C=                                    modified                         =
C=        8       ModifiedTime        The time the file was last       =
C=                                    modified                         =
C=        40      UnUsed              Additonal information. Not       =
C=                                    currently assigned.              =
C= Movie Header Information                                            =
C=        2       iFormatType         Specifies the format of the      =
C=                                    movie:                           =
```

```
C=                                   0       Raw bit image             =
C=                                   1       Aligned raw bit image.    =
C=                                           The movie frames are      =
C=                                           aligned with nPixels/8    =
C=                                           byte boundaries, where    =
C=                                           nPixels is the total      =
C=                                           number of pixels in the   =
C=                                           movie window.             =
C=        2       iFrameRate         Number of frames per second in    =
C=                                   original input                    =
C=        4       iSampleSpacing     The nominal spacing (in frames)   =
C=                                   between images in the movie.      =
C=        4       iMovieDuration     The expected duration of the      =
C=                                   movie (in original frames)        =
C=        4       iPtrFrameTable     Points to the start of the table  =
C=                                   containing the frame data         =
C=        4       nMovieFrames       The number of movie frames in     =
C=                                   the frame table.                  =
C=        2       iw0                The first row stored for the      =
C=                                   image                             =
C=        2       iw1                The last row stored for the image=
C=        2       jw0                The first column stored for the   =
C=                                   image                             =
C=        2       jw1                The last column stored for the    =
C=                                   image                             =
C=        2       idi                The step between sampled rows     =
C=        2       jdj                The step between sampled columns  =
C=        4       nSize              The size of the image (iw1-       =
C=                                   iw0+1)*(jw1-jw0+1)                =
C=        4       AspectRatio        The aspect ratio of the pixels    =
C=                                   in the image.                     =
C=        2       nBits              The number of bit planes stored   =
C=        256     iOLUTRed           Red component of OLUT             =
C=        256     iOLUTGreen         Green component of OLUT           =
C=        256     iOLUTBlue          Blue component of OLUT            =
C=        4       nFrameTableLength  The number of bytes in the        =
C=                                   frame table.                      =
C=        2       RecordAtFieldSpacing Indicates if the recording      =
C=                                   sample spacing is determined by   =
C=                                   iSampleSpacing or                 =
C=                                   dtSampleSpacing (if different).   =
C=        4       dtSampleSpacing    Nominal sample spacing (in        =
C=                                   seconds). This is used in         =
C=                                   priority to iSampleSpacing if     =
C=                                   RecordAtFieldSpacing is .FALSE.   =
C=        204     UnUsed             Additional information, not       =
C=                                   currently assigned.               =
C= Frame Table Information                                             =
C=        4       iFrameNumber0      The first movie frame number      =
C=        2       iLength            The number of frames required     =
C=                                   to process the movie during       =
C=                                   acquisition.                      =
C=        2       iDummy             Unused                            =
C=        OBSOLETE 4       iPtrFrame0         Points to the first frame.        =
C=        OBSOLETE 4       iPtrData0          Points to the additional data     =
C=        OBSOLETE                            for the frame (0 if no            =
C=        OBSOLETE                            additional data).                 =
C=        8       iPtrFrame0         Points to the first frame.
C=        4       iFrameNumber1      The second movie frame number     =
C=        OBSOLETE 4       iPtrFrame1         Points to the second frame.       =
C=        OBSOLETE 4       iPtrData1          Points to the additional data     =
C=        OBSOLETE                            for the frame (0 if no            =
C=        OBSOLETE                            additional data).                 =
C=        8       iPtrFrame1         Points to the second frame.
C=        ....                                                         =
C=     Note: DigImage limits the size of the frametable to 2048        =
C=     entries. In DigiFlow, this is extended to 32768 entries.        =
C=     DigImage will only be able to access the first 2048 entries.    =
```

13 Configuration files

As noted in §2, DigiFlow access a number of start-up files in the program directory each time it is started. The list below illustrates their use and the order in which they are called. Note that there is information about which file is being executed that is displayed in the status bar at the bottom of DigiFlow as it is started.

File	Usage
DigiFlow_GlobalData.dfc	Controls setup of root (global) interpreter
DigiFlow_dfcInstall.dfc	Installs **dfc** functions located in DLLs rather than kernel
DigiFlow_Constants.dfc	Defines constants used by **dfc** code
DigiFlow_CheckLicence.dfc	Controls the checking of the licence for DigiFlow
DigiFlow_Licence.dfc	Contains the DigiFlow licence. This file differs for each installation
DigiFlow_LocalData.dfc	Local customised data
DigiFlow_Plotting.dfc	Various plotting macros
DigiFlow_SimplePlot.dfc	The simple plot (`plot_`) functions
DigiFlow_Utilities.dfc	Utility macros
DigiFlow_Cameras.dfc	Details of supported cameras
DigiFlow_Dialogs.dfs	Status file restoring default dialog responses from last time DigiFlow was run in directory
DigiFlow_Configuration.dfc	Controls configuration of DigiFlow
DigiFlow_Update.dfc	Controls the updating of DigiFlow
DigiFlow_7Z_Install.dfr	Responses for extracting DigiFlow from archive
DigiFlow_Registry.dfc	Handles DigiFlow registry settings
DigiFlow_dfcCommands.dfc	Contains main **dfc** documentation
DigiFlow_Changes.dfr	Contains information about recent changes
DigiFlow_General.dfr	Contains **dfc** information for processes
DigiFlow_Latex.dfc	Defines LaTeX-like macros
DigiFlow_Status.dfs	Status file restoring settings from last time DigiFlow was run in directory

Subsequently, the following files may also be used, depending on the processing selected.

DigiFlow_Phrases.dfr
DigiFlow_Recipes.dfc
DigiFlow_SlaveProcess.dfc

Most of these files do not require modification by or knowledge of the user. The following sections discuss those configuration files that might require user customisation, or at least those that migh require a user knowledge of their structure. Those files were highlighted in red in the table above.

13.1 DigiFlow_Licence.dfc

The DigiFlow_Licence.dfc file, located in the DigiFlow installation directory, contains the licence for DigiFlow on one or more machines. This file is read and processed when DigiFlow starts. If the file cannot be found, then DigiFlow will prompt the user to create a LicenceRequest.dat file. This file should be sent to Dalziel Research Partners in order that they can generated a licence for your machine.

13.2 DigiFlow_LocalData.dfc

The standard distribution of DigiFlow does not include nor create a DigiFlow_LocalData.dfc file. Rather, this file is intended to contain user customisations that are not overwritten by

updating DigiFlow. (Note that a Site Licence server installation of DigiFlow will create a DigiFlow_LocalData.dfc on the server.) If the file is not detected, then default values will be used. Similarly, default values will be used if a specific value is not specified in DigiFlow_LocalData.dfc. The file may contain some or all of the following settings:

Variable	Type	Default	Comments
VideoCapture.UseCache	Logical	true	Causes DigiFlow to use a fixed cache file and undergo a review process each time video sequences are captured.
VideoCapture.CacheFile	String	"V:\Cache\CaptureVideo.dfm"	The default file and path to be used for video capture.
VideoCapture.CameraBuffer	String	"(default)"	Selects the buffering mechanism for the cache file. One of "(default)", "buffer", "nobuffer", "writethrough". Changing this may improve the performance when writing the capture file on some systems.
DigiFlowServer.Server	String	The server DigiFlow was installed from.	The server DigiFlow was installed from.
DigiFlowServer.Path	String	\\Server\DigiFlow$	The path (network share) where the server may be found.
DigiFlowServer.InstallDate	String		The date DigiFlow was installed on the server.
DigiFlowServer.InstallTime	String		The time DigiFlow was installed on the server.
DigiFlowServer.InstalledBy	String		The user-id of the person installing DigiFlow on the server.
DigiFlowServer.UpdateDirectory	String	\\Server\DigiFlow$	The path DigiFlow is to check for updates.
DigiFlow.Update.Type	String	Dependent on licence type: "manual" Free "ftp" Commercial "site" Site	Determines the location from which updates will be sought. Must be "manual" for free licences.
DigiFlow.Options.Bayer.rGain	Real	1.0	The gain applied to the red channel when a Bayer filter is used to determine a colour image from a single plane.
DigiFlow.Options.Bayer.gGain	Real	1.5	The gain applied to the green channel when a Bayer filter is used to determine a colour image from a single plane.
DigiFlow.Options.Bayer	Real	1.0	The gain applied to the blue

.bGain			channel when a Bayer filter is used to determine a colour image from a single plane.
DigiFlow.Options.Bayer .Roll	Integer	1	Controls the phase of the colour information within the single plane of pixels.
DigiFlow.Options.dfi .useCompression	Logical	true	Indicates that compression should be used by default for dfi files.
DigiFlow.Options .Display.dpi	Integer	96	Specifies the assumed display resolution. This overrides the `/dpi:`*n* command line switch.
DigiFlow.Options .Display.Scaling	Real	1.0	Specifies the scaling applied to certain visual elements of the user interface. This overrides the scaling implicitly calculated from the `/dpi:`*n* command line switch.

13.3 DigiFlow_Cameras.dfc

DigiFlow requires details of the cameras it is going to use as in many cases there is no mechanism for determining key information via the camera interface. These details are supplied in the DigiFlow_Cameras.dfc file. This file sets the `CameraInfo` compound variable, that is stored in the global interpreter context, and specifies both hardware details of the cameras that may be connected, and preferences for their use. The table below summarises the entries, each of which has the form `CameraInfo`.*xxx*.*yyy*, where *xxx* identifies the camera, and *yyy* the specific configuration item. The camera identifier is related to the name of its BitFlow framegrabber configuration file. It need not be the whole of the file name, but it does need to be unique within DigiFlow_Cameras.dfc and follow normal *dfc* syntax (*e.g.* it can not contain a hyphen character as this would be interpreted as minus).

Variable	Type	Comments
CameraInfo.*xxx*.CameraFile	String	Specifies the full name of the BitFlow configuration file.
CameraInfo.*xxx*.CameraName	String	A descriptive name for the camera.
CameraInfo.*xxx*.nChannels	Integer	The number of taps or channels feeding data from the camera to framegrabber.
CameraInfo.*xxx*.fpsMin	Real	The lowest frame rate supported by the camera.
CameraInfo.*xxx*.fpsMax	Real	The highest frame rate supported by the camera.
CameraInfo.*xxx*.CanChangeExposure	Logical	Indicates that the exposure can be changed independently of the frame rate.
CameraInfo.*xxx*.fpsDisplay	Real	The highest frame rate that should be used for displaying the output on screen. Typically this

		should be less than or equal to the smaller of 25 and `CameraInfo.`*xxx*`.fpsMax`.
`CameraInfo.`*xxx*`.fpsKind`	Integer	The method by which the number of frames per second can be changed. Values are 0 for no change possible, 1 for change via a CameraLink interface, and 2 for changes in the BitFlow CTab entry (typically for Dalsa cameras).
`CameraInfo.`*xxx*`.Untangle`	Logical	Indicates processing is required to untangle the information from the camera to generate a valid display.
`CameraInfo.`*xxx*`.prefPreviewResolutionFactor`	Integer	The scale factor that should be applied to the image when previewing it on screen. For very high resolution cameras, a value greater than 1 will reduce the size of the preview image, allowing more rapid display and allowing the image to fit more comfortably on the screen (which may have a much lower resolution). Note that the user can override this setting in the dialog used to start the preview.
`CameraInfo.`*xxx*`.prefPreviewProcessing`	String	It is often desirable to have some form of processing on the preview image. The default processing (which may be overridden by the user in the dialog starting the preview) is specified by this string. Typical examples include `"particle streaks"` and `"synthetic schlieren"`.
`CameraInfo.`*xxx*`.prefFpsDisplay`	Real	The preferred display frames per second for the preview. This may be overridden by the user in the dialog used to start the preview.
`CameraInfo.`*xxx*`.nCaptureBuffers`	Integer	The number of buffers to which the video is initially captured. Typically 2 for an R3 framegrabber, or 8 for an R64-based card..
`CameraInfo.`*xxx*`.nTotalBuffers`	Integer	The number of buffers to be

		reserved for the camera. Typically 8 for an R3 framegrabber, or 16 for an R64-based card..
`CameraInfo.xxx.nProcessThreads`	Integer	The number of threads used to process the sequence being captured prior to saving.
`CameraInfo.xxx.nDisplayThreads`	Integer	The number of threads used to display the sequence being captured. Typically 1.
`CameraInfo.xxx.fpsToInteger`	Code	Converts the requested number of frames per second into an integer value for internal use. For many cameras, the permissible number of frames per second is restricted to be an integer fraction of the base frame rate.
`CameraInfo.xxx.fpsFromInteger`	Code	Converts an integer (produced by `fpsToInteger`) back into a frame rate.
`CameraInfo.xxx.ReserveSpaceInFile`	Logical	Causes extra space to be reserved in the Cache file.
`CameraInfo.xxx.minReserveFramesInFile`	Integer	If the cache file must be extended, then this will be the minimum additional space reserved for future growth.
`CameraInfo.xxx.SetGain`	Code	Code used for suitable CameraLink cameras to set the gain.
`CameraInfo.xxx.maxGain`	Integer	The maximum gain for the camera.
`CameraInfo.xxx.defaultGain`	Integer	The default gain value for the camera.
`CameraInfo.xxx.ShutterSpeed`	Code	Code used for suitable CamerLink cameras to set the shutter speed.
`CameraInfo.xxx.minShutter`	Integer	The minimum shutter speed for the camera.
`CameraInfo.xxx.maxShutter`	Integer	The maximum shutter speed for the camera.
`CameraInfo.xxx.defaultShutter`	Integer	The default shutter speed for the camera.
`CameraInfo.xxx.AllowSerial`	Logical	Enable serial communication over CamerLink connection.
`CameraInfo.xxx.SerialLineFeed`	Logical	Indicates that the CameraLink serial protocol requires line feeds as well as carriage returns.
`CameraInfo.xxx.SerialStatusQuery`	String	The string that should be sent to

		the CameraLink camera to query its status.
CameraInfo.*xxx*.SerialOnLineResponse	String	The first part of the response to the CameraLink status query that, if received, indicates the camera is on line.
CameraInfo.*xxx*.SerialError	String	The response from a CameraLink camera that indicates an error condition.
CameraInfo.*xxx*.SetupCode	String or Code	dfc code that is run before starting the capture process to set up the camera.
CameraInfo.*xxx*.PostSetupCode	String or Code	dfc code that is run after starting the capture process to set up the camera.
CameraInfo.*xxx*.SetFrameRate	String	dfc code to alter the frame rate.
CameraInfo.*xxx*.minFrameRate	Integer	The minimum frame rate.
CameraInfo.*xxx*.maxFrameRate	Integer	The maximum frame rate.
CameraInfo.*xxx*.defaultFrameRate	Integer	The default frame rate.
CameraInfo.*xxx*.delayFrameStart	Integer	The number of pixels to delay that separates the start of the image from the start of the frame sent by the camera.
CameraInfo.*xxx*.StrobeStart	Integer	This optional setting is used in conjunction with the BitFlow framegrabber's VSTROBE output signal (available via the 15 pin D connector). Typical uses of this include controlling a strobe light, or driving a liquid crystal shutter. Adjusting this value changes the phase of the start of the VSTROBE pulse relative to the camera acquisition cycle.
CameraInfo.*xxx*.StrobeStop	Integer	As with CameraInfo.*xxx*.StrobeStart, but sets the timing for the end of the VSTROBE pulse. Note that if StrobeStop is less than StrobeStart then VSTROBE is low only between the stop and the start.
CameraInfo.*xxx*.TriggerEventType	String	Indicates the type of event that is triggered.
CameraInfo.*xxx*.TriggerEventLine	Integer	The line within the image that the event is triggered at. Note for R2 and R3 framegrabbers, this will typically be 4096 plus the

		scan line number.
CameraInfo.*xxx*.TriggerEventMaxCount	Integer	Causes wrapping of the trigger event.

13.4 DigiFlow_Dialogs.dfs

This status file is stored in the directory from which DigiFlow is started. It contains and localises information about the responses used most recently in each of the main DigiFlow dialogs. The format of the entries is that of dfc calls to each of the DigiFlow facilities, with the name of the compound variable being descriptive.

The DigiFlow_Dialogs.dfs will only contain dialog entries for facilities that have been run for DigiFlow started in the directory on this or a previous invocation. The following is an example of this file. Note that the order of the entries is not fixed.

```
dlgFile_CaptureVideo.Output_Options.Colour := "(default)";
dlgFile_CaptureVideo.Output_Options.Display :=           1;
dlgFile_CaptureVideo.Output  := "CaptureVideo.dfm";
dlgFile_CaptureVideo.RegionName := "(all)";
dlgFile_CaptureVideo.Gain :=           0;
dlgFile_CaptureVideo.ShutterSpeed :=           0;
dlgFile_CaptureVideo.DisplayProcessDialog :=           1;
dlgFile_CaptureVideo.DisplayProcessing := "(none)";
dlgFile_CaptureVideo.PreProcessFrame := "Default";
dlgFile_CaptureVideo.Time :=  60.0000     ;
dlgFile_CaptureVideo.DisplayResolutionFactor :=           3;
dlgFile_CaptureVideo.TimeMode := "Time";
dlgFile_CaptureVideo.DisplayDuringCapture :=           1;
dlgFile_CaptureVideo.nSaveBits :=           8;
dlgFile_CaptureVideo.fpsDisplay :=  24.0000     ;
dlgFile_CaptureVideo.fpsShutter :=  30.0000     ;
dlgFile_CaptureVideo.fpsCapture :=  30.0000     ;
dlgFile_CaptureVideo.AcquireEqShutter :=           1;
dlgFile_CaptureVideo.DisplayOnExit :=           1;
dlgFile_CaptureVideo.process := "File_CaptureVideo";

LastResponse := "dlgFile_EditStream";

openImage.ChangeToHashes :=           1;
openImage.CompactList :=           1;

runCode.DefaultDir := "s:\users\stuart\img\digiflow\";

dlgFile_EditStream.Output_Options.DeleteExisting :=           1;
dlgFile_EditStream.Output_Options.Resample := "none";
dlgFile_EditStream.Output_Options.Comments := "No user comments";
dlgFile_EditStream.Output_Options.Colour := "(default)";
dlgFile_EditStream.Output_Options.Display :=           1;
dlgFile_EditStream.Output  := "junk####.dfi";
dlgFile_EditStream.DirectCopy :=           1;
dlgFile_EditStream.Input_Region.Kind := "All";
dlgFile_EditStream.Input_Options.Display :=           1;
dlgFile_EditStream.Input  := "JPRD45.dfm";
dlgFile_EditStream.DisplayOnExit :=           1;
dlgFile_EditStream.process := "File_EditStream";

dfcConsole.LastOpen :=
     "S:\Users\Stuart\Img\DigiFlow\StreamFunctionVorticity.dfc";
dfcConsole.LastSave :=
     "S:\Users\Stuart\Img\DigiFlow\FunctionNames.dfc";
```

In addition to the compound `dlg…` variables, DigiFlow also stores other user-interface related information in this file. In particular, `LastResponse` indicates the last dialog to be

activated. This value is used in the Edit Dialog Responses facility described in §5.2.9. The openImage compound variable records the current settings of the Open Image dialog (see §4.1), in particular how to handle numbered image sequences.

Handling of dfc files is controlled by runCode.DefaultDir, which stores the current directory for File: Run Code (see §5.1.2), while dfcConsole stores information about the last dfc files to be opened or saved.

13.5 DigiFlow_Status.dfs

As with DigiFlow_Dialogs.dfs, the DigiFlow_Status.dfs stores localised status information in the directory in which DigiFlow was started, restoring that saved previously each time it is started. However, whereas DigiFlow_Dialogs.dfs concentrates on data associated with the user interface, DigiFlow_Status.dfs stores information concerning the internal state of DigiFlow such as colour shemes, coordinate systems and region definitions.

The example below illustrates the structure of this file. Note that this is again a dfc file. However, for brevity, the contents have been shown truncated; truncations are indicated by an ellipsis (…).

```
#S:\Users\Stuart\Img\DigiFlow\DigiFlow_Status.dfs
###### Begin
## Mon Apr 23 17:57:06 2007
#
#Colour Schemes
add_colour_scheme(scheme:="bipolar",
 red:=[ 0.4980 0.4902 0.4824 0.4745 0.4667 0.4588 0.4510 …],
 green:=[ 1.0000 0.9843 0.9686 0.9529 0.9373 0.9216 0.9059 …],
 blue:=[ 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 …]);
add_colour_scheme(scheme:="schlieren",
 red:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …],
 green:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …],
 blue:=[ 0.0000 0.0118 0.0235 0.0353 0.0471 0.0588 0.0706 …]);
add_colour_scheme(scheme:="single cycle - double brightness",
 red:=[ 0.5020 0.5176 0.5333 0.5490 0.5647 0.5804 0.5961 …],
 green:=[ 0.5020 0.5020 0.5020 0.5020 0.5020 0.5020 0.5020 …],
 blue:=[ 0.5020 0.5020 0.5020 0.5020 0.5020 0.5020 0.5020 …]);
add_colour_scheme(scheme:="single cycle - aperture",
 red:=[ 0.0000 0.0157 0.0314 0.0471 0.0627 0.0784 0.0941 …],
 green:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …],
 blue:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …]);
add_colour_scheme(scheme:="single cycle - half brightness",
 red:=[ 0.0000 0.0157 0.0314 0.0471 0.0627 0.0784 0.0941 …],
 green:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …],
 blue:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …]);
add_colour_scheme(scheme:="(default)",
 red:=[ 0.0000 0.0314 0.0510 0.0902 0.1255 0.1569 0.1804 …],
 green:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …],
 blue:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …]);
add_colour_scheme(scheme:="single cycle",
 red:=[ 0.0000 0.0314 0.0627 0.0941 0.1255 0.1569 0.1882 …],
 green:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …],
 blue:=[ 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 …]);
add_colour_scheme(scheme:="negative",
 red:=[ 1.0000 0.9961 0.9922 0.9882 0.9843 0.9804 0.9765 …],
 green:=[ 1.0000 0.9961 0.9922 0.9882 0.9843 0.9804 0.9765 …],
 blue:=[ 1.0000 0.9961 0.9922 0.9882 0.9843 0.9804 0.9765 …]);
add_colour_scheme(scheme:="greyscale",
 red:=[ 0.0000 0.0039 0.0078 0.0118 0.0157 0.0196 0.0235 …],
 green:=[ 0.0000 0.0039 0.0078 0.0118 0.0157 0.0196 0.0235 …],
 blue:=[ 0.0000 0.0039 0.0078 0.0118 0.0157 0.0196 0.0235 …]);
#
#Coordinate Systems
```

```
#Coordinate System: Tank
coord_system_create(name:="Tank",units:="mm");
coord_system_mapping(name:="Tank",mapping:="Linear: 1;x;y;");
coord_system_add_point(name:="Tank",xPixel:=155.0,yPixel:=385.0,
     xWorld:=0.0,yWorld:=1.0);
coord_system_add_point(name:="Tank",xPixel:=551.0,yPixel:=104.0,
     xWorld:=1.0,yWorld:=0.0);
coord_system_add_point(name:="Tank",xPixel:=165.0,yPixel:=111.0,
     xWorld:=0.0,yWorld:=0.0);
#Set default coordinate system
coord_system_set_default(name:="Tank");
#
#Regions
#
#MatchIntensity
set_dfc_path(path:="s:\users\stuart\img\digiflow\;");
###### End
#S:\Users\Stuart\Img\DigiFlow\DigiFlow_Status.dfs
```

In this particular example, there are no regions and no Match Intensity specifications set. The function `set_dfc_path(..)` specifies the path along which DigiFlow will search for any dfc files.

14 Extending DigiFlow

DigiFlow is designed so as to allow users to extend its core functionality in a number of ways. This section gives a brief introduction to the facilities and techniques available.

14.1 Installing extensions

DigiFlow extensions are installed via a dfc interface.

add_image_reader(

add_image_reader(*dll*)

add_image_reader(*dll*,*routine*)

Adds a new image reader.

dll string

The name of the DLL file containing the image reader.

routine optional string (default "ReadImageDLL")

The name of the function within the DLL that provides the required functionality..

Return value None.

The following example (using Fortran 90 syntax) may be used as the basis of a user-written reader. In this example, the image is provided as ASCII values in a file, prefixed by the size of the image. If the function fails, it should return `Image_FileType_Unknown`; if it succeeds, it should return `Image_FileType_DLL`:

```
      function ReadImageDLL(Image,File,Options,Descriptor)
!DEC$ ATTRIBUTES DLLEXPORT, REFERENCE :: ReadImageDLL
!DEC$ ATTRIBUTES ALIAS:'ReadImageDLL' :: ReadImageDLL
!=====Modules
      use T_All
!=====Parameters
      integer (4) ReadImageDLL
      type (F_Image), intent(inout) :: Image
      character (*), intent(in) :: File
      type (F_ImageOptions), intent(inout), optional :: Options
      type (F_ImageDescriptor), intent(inout), optional :: Descriptor
!=====Local variables
      integer (4), automatic :: iFile,i,j,io,nx,ny
!=====Code
      ReadImageDLL = Image_FileType_Unknown
      call NewFileHandle(iFile)
      open(iFile,file=File,status='old',form='formatted',err=99)
      read(iFile,*,err=99)nx,ny    ! Read image size
      ! Create the image
      call CreateImage(Image,nx=nx,ny=ny,AccessAs=Image_AccessAs_Real)
      ! Read in the intensities
      do j=0,Image%Height-1
        read(iFile,*,err=99)(Image%R2(i,j),i=0,nx-1)
      enddo
      ! Automatic scaling of intensities
      Image%rBlack = minval(Image%R2)
      Image%rWhite = maxval(Image%R2)
      if (present(Descriptor)) then
        Descriptor%Time%iFrame = 999
        Descriptor%Time%tNow = 999.0
        Descriptor%Time%tStep = 0.1
        Descriptor%Time%tFirst = 0.0
        Descriptor%Comments%UserComments = 'This is a test'
        Descriptor%Comments%CreatingProcess = 'Sample'
      Endif
      ReadImageDLL = Image_FileType_DLL
99    continue
      close(iFile)
      call FreeFileHandle(iFile)
      return
      end Function
```

15 Miscellaneous publications

This section lists a small subset of the papers that have been published using DigiFlow (or DigImage). The subset has been selected as these papers showcases a new feature in or new application of the technology within DigiFlow. This is by no means a complete list of publications in which DigiFlow has been used, even by the authors of DigiFlow. For a complete list of the latter, refer to http://www.damtp.cam.ac.uk/lab/people/sd103/papers/

Dalziel, S.B. 1992 Decay of rotating turbulence: some particle tracking experiments; *Appl. Scien. Res.* **49**, 217-244. [[pdf]]

Dalziel, S.B. 1993 Rayleigh-Taylor instability: experiments with image analysis; *Dyn. Atmos. Oceans*, **20** 127-153. [[pdf]]

Boubnov, B.M., Dalziel, S.B. & Linden, P.F. 1994 Source-sink turbulence in a stratified fluid; *J. Fluid Mech.* **261**, 273-303. [[pdf]]

Linden, P.F., Boubnov, B.M. & Dalziel, S.B. 1995 Source-sink turbulence in a rotating, stratified fluid; *J. Fluid Mech* **298**, 81-112. [[pdf]]

Hacker[S], J., Linden, P.F. & Dalziel, S.B. 1996 Mixing in lock-release gravity currents; *Dyn. Atmos. Oceans* **24**, 183-195. [[pdf]]

Holford[S], J.M. & Dalziel, S.B. 1996 Measurements of layer depth in a two-layer flow; *Appl. Scien. Res.* **56**, 191-207. [[pdf]]

Cenedese[S], C. & Dalziel, S.B. 1998 Concentration and depth fields determined by the light transmitted through a dyed solution; *Proceedings of the 8th International Symposium on Flow Visualization*, ed. G.M. Carlomagno & I. Grant. ISBN 0 9533991 0 9, paper 061. [[pdf]]

Dalziel, S.B., Hughes[S], G.O. & Sutherland[P], B.R. 1998 Synthetic schlieren; *Proceedings of the 8th International Symposium on Flow Visualization*, ed. G.M. Carlomagno & I. Grant. ISBN 0 9533991 0 9, paper 062. [[pdf]]

Sutherland[P], B.R., Dalziel, S.B., Hughes[S], G.O. & Linden, P.F. 1999 Visualisation and Measurement of internal waves by "synthetic schlieren". Part 1: Vertically oscillating cylinder; *J. Fluid Mech.* **390**, 93-126. [[pdf]]

Dalziel, S.B., Linden, P.F. & Youngs, D.L. 1999 Self-similarity and internal structure of turbulence induced by Rayleigh-Taylor instability; *J. Fluid Mech.* **399**, 1-48. [[pdf]]

[[pdf]]*J:* **(21)** Sutherland[P], B.R., Hughes[S], G.O., Dalziel, S.B. & Linden, P.F. 2000 Internal waves revisited; *Dyn. Atmos. Oceans*. **31**, 209-232.

Dalziel, S.B., Hughes[S], G.O. & Sutherland[P], B.R. 2000 Whole field density measurements by 'synthetic schlieren'; *Experiments in Fluids* **28**, 322-335. [[pdf]]

Leppinen[P], D.M. & Dalziel, S.B. 2001 A light attenuation technique for void fraction measurement of microbubbles; *Experiments in Fluids* **30**, 214-220. [[pdf]]

Ross[S], A.N., Linden, P.F. & Dalziel, S.B. 2002 A study of three-dimensional gravity currents on a uniform slope; *J. Fluid Mech.* **453**, 239-261. [[pdf]]

Thomas, L.P., Marino, B.M. & Dalziel, S.B. 2003 Measurement of density distribution in a fluid layer by light induced fluorescence in non-rectangular cross section channels. *Int. J. Heat & Tech.* **21**, 143-148. [[pdf]]

Thomas, L.P., Dalziel, S.B. & Marino, B.M. 2003 The structure of the head of an internal gravity current determined by Particle Tracking Velocimetry. *Expt Fluids.* **34**, 708-716. [[pdf]]

Higginson[S], R.C., Dalziel, S.B. & Linden, P.F. 2003 The drag on a vertically moving grid of bars in a linearly stratified fluid. *Expt. Fluids.* **34**, 678-686. [[pdf]]

Munro[P], R.J., Dalziel S.B. & Jehan[S], H. 2004 A pattern matching technique for measuring sediment displacement levels. *Expt. Fluids*. **37**, 399-408. [[pdf]]

Shin[S], J.O., Dalziel, S.B. & Linden, P.F. 2004 Gravity currents produced by lock exchange. *J. Fluid Mech.* **521**, 1-34. [[pdf]]

Jacobs, J. & Dalziel, S.B. 2005 Rayleigh-Taylor instability in complex stratifications. *J.Fluid Mech.* **542**, 251-279. [[pdf]]

Munro[P], R.J. & Dalziel S.B. 2005 Attenuation technique for measuring sediment displacement levels. *Expt. Fluids* **39**, 602-613. [[pdf]]

Sveen[P], J.K. & Dalziel, S.B. 2005 A dynamic masking technique for combined measurements of PIV and Synthetic Schlieren applied to internal gravity waves. *Meas. Sci. Technol.* **16**, 1954-1960. [[pdf]]

Ross[S], A.N., Dalziel, S.B. & Linden, P.F. 2006 Gravity currents on a cone. *J. Fluid Mech.* **565**, 227-253. [[pdf]]

Scase[S], M.M. & Dalziel, S.B. 2006 An experimental study of the bulk properties of vortex rings translating through a stratified fluid. *Eur. J. Mech. B/Fluids* **25**, 302-320. [[pdf]]

Dalziel, S.B., Carr[P], M., Sveen[P], K.J. & Davies, P.A. 2007 Simultaneous Synthetic Schlieren and PIV measurements for internal solitary waves. *Meas. Sci. Tech.* **18**, 533-547. [[pdf]]

Hazewinkel[S], J., Breevoort[S], P. van, Maas, L.R.M., Doelman, A. & Dalziel, S.B. 2007 Equilibrium spectrum for internal wave attractor in a trapezoidal basin. In *Proceedings of the 5th International Symposium on Environmental Hydraulics.* [[pdf]]

Hazewinkel[S], J., Breevoort[S], P. van, Dalziel, S.B. & Maas 2008 L.R.M. Observations on the wave number spectrum and evolution of an internal wave attractor in a two-dimensional domain. *J. Fluid Mech.* **598**, 81-105. [[pdf]]

Dalziel, S.B., Patterson[P], M.D., Caulfield, C.P. & Coomaraswamy, I.A. 2008 Mixing efficiency in high aspect-ratio Rayleigh-Taylor experiments. *Phys. Fluids* **20**, 065106. DO*I:*10.1063/1.2936311. [[pdf]] [[web]]

Munro[P], R.J., Bethke[S], N. & Dalziel, S.B. 2009 Sediment resuspension and erosion by vortex rings. *Phys. Fluids.* **21**, 046601. [[pdf]]

Ihle[S], C.F., Dalziel, S.B. & Niño, Y. 2009 Simultaneous PIV and synthetic schlieren measurements of an erupting thermal plume. *Meas. Sci. Tech.* **20**, 125402. [[pdf]]

Hazewinkel[S], J., Tsimitri[S], C., Maas, L.R.M. & Dalziel, S.B. 2010 Observations on the robustness of internal wave attractors to perturbations. *Phys. Fluids* **22**, 107102, do*I:*10.1063/1.3489008. [[pdf]] [[web]]

Hazewinkel[S], J., Maas, L.R.M. & Dalziel, S.B. 2011 Tomographic reconstruction of internal wave patterns in a paraboloid. *Exp. Fluids* **50**, 247-258. DO*I:* 10.1007/s00348-010-0909-x. [[pdf]] [[web]]

Hazewinkel[S], J., Grisouard[S], N. & Dalziel, S.B. 2011 Comparison of laboratory and numerically observed scalar fields of an internal wave attractor. *Eur. J. Mech/B*. **30**, 51-56. DO*I:*10.1016/j.euromechflu.2010.06.007. [[web]]

Dalziel, S.B., Patterson[P], M.D., Caulfield, C.P. & Le Brun[S], S. 2011 The structure of low Froude number lee waves over an isolated obstacle. *J. Fluid Mech.* **689**, 3-31. doi:10.1017/jfm.2011.384. [[pdf]] [[web]]

Bethke[S], N. & Dalziel, S.B. 2011 Resuspension onset and crater erosion by a vortex ring interacting with a particle layer. To appear in *Phys. Fluids.*

See http://www.damtp.cam.ac.uk/lab/people/sd103/papers/ for a comprehensive list (including many with electronic copies) of papers related to the techniques used by DigiFlow.

References

Dalziel, S.B. 1992 Decay of rotating turbulence: some particle tracking experiments; *Appl. Scien. Res.* **49**, 217-244.

Dalziel, S.B. 1993 Decay of rotating turbulence: some particle tracking experiments; in *Flow visualization and image analysis*; Ed. Nieuwstadt, Kluwer, Dordrecht, 27-54.

Dalziel, S.B. 1993 Rayleigh-Taylor instability: experiments with image analysis; *Dyn. Atmos. Oceans*, **20** 127-153.

Dalziel, S.B., Hughes, G.O. & Sutherland, B.R. 2000 Whole field density measurements by 'synthetic schlieren'; *Experiments in Fluids* **28**, 322-335.

Monahan, J.J. 1992 Smoothed particle hydrodynamics. *Ann. Rev. Astrophys.* **30**, 543-574.

Sutherland, B.R., Dalziel, S.B., Hughes, G.O. & Linden, P.F. 1999 Visualisation and Measurement of internal waves by "synthetic schlieren". Part 1: Vertically oscillating cylinder; *J. Fluid Mech.* **390**, 93-126.

Index

3D view, 64

Accessing dialogs, 232
Accumulate images, 171
Aligning images, 169
Analyse
  Ensemble mean, 84
  Harmonic Analysis, 75
  Time average, 71
  Time extract, 79
  Time series, 77
Analyse_DyeAttenuation, 86
Analyse_EnsembleMean, 84
Analyse_FollowOpticalFlow, 154
Analyse_PIV, 121
Analyse_PTVTrack, 131
Analyse_PTVVectors, 150
Analyse_ShowAsStreaks, 118
Analyse_SyntheticSchlierenInterpolative, 100
Analyse_SyntheticSchlierenPatternMatch, 105
Analyse_SyntheticSchlierenQualitative, 98
Analyse_TimeAverage, 71
Analyse_TimeExtract, 79
Analyse_TimeSummarise, 81
Appearance, 61
Archive file
  input stream, 13, 225
  output stream, 228
Array functions, 204
Array plotting functions, 212
Arrays, 185
as_thread(..), 66
Assignment, 185
Autocorrelation, 148
AutoHelp, 12
Average
  Ensemble, 84
  Time, 71
AVI files, 42

Bit depth of output stream, 229
BitFlow, 7
Bit-wise operations, 210
bmp
  file format, 241
Break points, 200

Cache file, 8
Camera configuration, 7
Camera configuration - BitFlow, 7
Camera control, 210
camera_*, 33
Capture configuration, 8
Capture video, 36
Chaining responses, 230
Close, 46
Close all, 46
Code library, 31
Collection - definition, 13
Colour
  output stream, 27
  toggle, 64
Colour images, 20
Colour of output stream, 229
Colour scheme, 62
Combine images, 166
Command prompt, 9
Command prompt - commands, 9
Comments in output stream, 230
compile(), 193
Compound variables, 184
Compression of output stream, 229
Concentration power spectrum, 81
Configuration, 258
Configuration files, 257
Configuration functions, 217
Contouring images, 162
Coordinate functions, 210
Coordinate system wizard, 52
Coordinate systems, 50, 51, 52, 174
Coordinates
  copy world system, 52
  edit world system, 51
  new world system, 50
  transform to world, 174
  world, 49
Copy, 47
  as bitmap, 47
Cursor, 60

Data acquisition functions, 215

Data Translation, 1
Debugging, 196
Deleting existing output stream, 230
Depth of a gravity current, 80
dfa
  file format, 251
dfc
  run, 33
dfc code, 14
dfc Console, 201
dfc Help, 30
dfcConsole, 55
dfd
  file format, 235, 251
dfi
  file format, 242
dfm
  file format, 255
dfp
  file format, 251
dft
  file format, 251
Dialog responses, 55, 233
Differential functions, 213
DigiFlow command files, 223
DigiFlow configuration, 3
DigiFlow_Cameras.dfc, 258
DigiFlow_Dialogs.dfs, 3, 262
DigiFlow_Licence.dfc, 257
DigiFlow_LocalData.dfc, 258
DigiFlow_Status.dfs, 3, 263
DigImage, 1
DirectDraw functions, 214
Displaying output stream, 228
Distance
  measure, 60
Drawing, 234
Drawing commands, 234
Drawing file format, 235
Dye attenuation, 86
Dye concentration, 89, 90

Edit
  dialog responses, 55
  process again, 54
  properties, 48
  region, 54
  world coordinates, 49
Edit dfc code, 55
Edit stream, 40
emf
  file format, 241
Encapsulated PostScript, 4, 29, 43
Enlarge
  zoom in, 58
  zoom out, 58
Ensemble
  Average, 84
eps, 29, 43
  file format, 242
  printer setup, 4
Error handling, 196
Eulerian, 131
execute statement, 192
Exit, 46
exit statement, 192
exit_digiflow, 46
exit_digiflow(), 192
Export AVI, 42
Export to simple EPS, 46
Extending DigiFlow, 265
Extract time series, 79

File handling functions, 206
File menu, 33
File_CaptureVideo(..), 36
File_EditStream, 40
File_ExportAVI, 42
File_MergeStreams, 41
File_ShowLiveVideo, 33
Filtering images, 162
First index, 28
First index for output stream, 229
Flow functions, 210
Folder for output stream, 228
Follow optical flow, 154
for statement, 192
Format
  bmp, 241
  dfa, 251
  dfd, 235, 251
  dfi, 242
  dfm, 255
  dfp, 251
  dft, 251
  emf, 241
  eps, 242
  gif, 241

image files, 241
mov, 255
output stream, 28
pic, 253, 254
tif, 241
wmf, 242
Fourier descriptors, 162
Fractal
box count, 163
Fractal dimension, 83
Frame grabber, 7
ftp functions, 214
Full colour
output stream, 28
function, 193
Functions, 203
all, 218
array, 204
array plotting, 212
basic mathematical, 204
configuration, 217
coordinate, 210
differential, 213
file handling, 206
flow, 210
image processing, 209
information, 205
logging, 216
miscellaneous, 217
numerical, 212
reading and writing images, 207
statistics, 208
string, 204
threads, 213
timing, 208
type manipulation, 205
variables, 206
windows and views, 207

GhostScript functions, 215
gif
file format, 241
Grid PTV velocity, 151

Help, 12
History, 1

if statement, 191
Image
open, 17, 33
properties, 48
save, 21, 33
Image file formats, 241
Image processing functions, 209
Image selectors, 13
Images streams, 13
include(..), 196
Information functions, 205
Input
from file, 196
Input stream
archive file, 13, 225
colour component, 225
control in macro file, 224
displaying, 225
folder for, 224
match intensity, 25
matching intensity, 227
region, 23, 226
selecting times, 226
sifting, 22
timing, 22
waiting for, 228
Installation, 3
Intensity
transform, 158
transform recipe, 157

Key features, 2

Lagrangian, 131
LaTeX, 15, 30, 43
LaTeX macros, 238
Leaving output stream visible, 230
Library, 31
Lists, 188
Live video
particle streaks, 35
synthetic schlieren, 35
Live view, 33
Local data, 258
Logging functions, 216

Macro files, 223
Macros, 14, 223
accessing dialogs, 232
chaining responses, 230
control of input streams, 224

control of output stream, 228
multiple output streams, 231
make_array(..), 185
make_like(..), 185
make_list(..), 188
Match intensity, 25, 227
Matching algorithm, **133**
Mathematical functions, 204
MatLab, 2
Mean
Ensemble, 84
Time, 71
Measure distance, 60
Merge streams, 41
mf
file format, 242
mod, 190
mov
file format, 255
Move image, 61
Multiple output streams, 231

Numerical functions, 212

Objective function, 133
Open image, 17, 33, 46
Operators, 189
mod, 190
Optical flow, 153
follow, 154
Follow, 154
Options
output stream, 27
Output stream
archive file, 228
bit depth, 229
colour, 27, 229
comments, 230
compression, 229
controlling, 228
deleting existing stream, 230
displaying, 228
file format, 28
first index, 28, 229
folder for, 228
full colour, 28
leaving visible, 230
multiple, 231
options, 27
quality, 230
resampling, 28, 230
user comments, 29

Particle Image Velocimetry (PIV), 121
Particle streaks, 118
live video, 35
Particle tracking functions, 216
Particle Tracking Velocimetry (PTV), 131
Particles
streaks, 118
pic
file format - compressed, 254
file format - raw, 253
PIV data
example of post-processing, 164
Plot, 237
Plotting, 234
PostScript, 215
PostScript driver, 5
Printing, 43
process, 223
Process again, 54
psfrag, 30, 43
PTVAutocorrelation, 148
PTVBasic statistics, 147
PTVGridVelocity, 151

Quality of output stream, 230
Queries, 200
quit statement, 192

Reading and writing images, 207
Recording user input, 233
Redo process, 54
Region, 226
edit, 54
input stream, 23
naming, 24
Registry, 7
Registry functions, 217
Resampling, 28
Resampling output stream, 230
Rescaling an image, 161
Run code, 33
Running processes, 223

Save image, 21, 33
Security, 7

Selectors, 13
Sequence - definition, 13
Serial communications, 215
Setup video, 39
Show where, 60
Sifting, 14
  input stream, 22
Sifting data, 89
Simple plot, 237
Slave process, 34, 172
Slaves, 64
Starting DigiFlow, 9
Statements
  exit_digiflow, 46
Statistical functions, 208
Status files, 3
Streaks, 118
String functions, 204
Summarise time series, 81
Synthetic schlieren, 108
  live video, 35
  process, 100, 105
  qualitative, 98
Synthetic Schlieren, 95

Text output, 15
Thread functions, 213
Threads, 14
  run code as thread, 66
tif
  file format, 241
Time
  Average, 71
  Mean, 71
Time average, 71
Time extract, 79
Time series, 77
Time summarise, 81
Timing
  input stream, 22
Timing functions, 208
Toggle colour, 64
Tools_CombineImages, 166
Tools_TransformIntensity, 158
Tools_TransformRecipe, 157
Tools_TransformToWorld, 174
Tracing execution, 200
Transform
  intensity, 158
  recipe, 157
  to world coordinates, 174
Transportation algorithm, 133
try_execute statement, 192
Type manipulation functions, 205
Type query functions, 185

User comments, 29
User functions
  definition, 193
User input, 195
User output, 195

Variable functions, 206
Vector scale, 61
Velocities
  PIV, 121
  PTV, 131
  PTV - autocorrelation, 148
  PTV – basic statistics, 147
  PTV - grid, 151
  PTV - vectors, 150
  Show as streaks, 118
Velocity fluctuations, 170
Velocity statistics, 147
Velocity vectors – PTV, 150
Video
  capture, 36
  live, 33
  setup, 39
View
  3D, 64
  appearance, 61
  close, 46
  close all, 46
  colour scheme, 62
  slaves, 64
  toggle colour, 64
  vector scale, 61
  zoom, 58
View variables, 197
view_variables(..), 197

Waiting for input streams, 228
Web browsing, 214
while statement, 191
Wild cards, 20
Window and view functions, 207
World coordinates, 49

Z - objective function, 133
ZLib, 245
Zoom
  all full size, 59
  all half size, 59
  all one third size, 59
  all quarter size, 59
  custom, 58
  Fit window to, 60
  full size, 58
  in, 58
  to window, 59
Zoom out, 58

16 Licence Agreement

DigiFlow Licence Agreement

Dalziel Research Parnters,
142 Cottenham Road, Histon, Cambridge CB4 9ET, England

Licence:

1. Agreement. Installation of part or all of the software suite known as DigiFlow, or any system derived from DigiFlow, is deemed to indicate agreement with the terms and conditions of this licence.

2. Parties. This agreement is between Dalziel Research Partners, the software Developer and copyright holder, and the person, company or institution installing or using the software, the Customer.

3. Definition. DigiFlow comprises the DigiFlow executable files, support files, documentation files, configuration files, and other utilities supplied with the system, and the source code, object code, libraries and documentation associated with or supplied with the system.

4. Types of Licence. Two types of licence are available for DigiFlow. A Free Licence has restricted functionality and more limited support. A Commercial Licence provides access to all of DigiFlow's features and has a broader level of support.

5. Right of use. The Developer hereby grants the Customer, a non-transferable and nonexclusive licence to use DigiFlow on a single microcomputers within the premises of the Customer. The software may be transferred to a different microcomputer, with the Customer's premises, only after it has been removed completely from any machine on which it was installed previously.

6. Licence Key. The Licence Key is the numeric code that controls the use of DigiFlow on a given computer. The Licence Key is unique to the network adapter(s) on the licensed computer. The Customer must inform the Developer if they wish to transfer DigiFlow from one computer to another, or if the network card is changed or removed. The Developer will then issue a new Licence Key for the new computer.

7. Use off site. DigiFlow, or any software developed using the DigiFlow Development System, may not be used outside the premises of the Customer without the express prior written consent from the Developer. The exception to this rule is that use during brief field studies (lasting no more than four weeks in a twelve month period), for demonstration and presentation purposes is permitted.

8. Revocation. The Developer reserves the right to revoke the Licence if the Customer fails to meet any of the terms under which the software is supplied.

9. Copyright. The copyright of DigiFlow, and all its components and manuals, is owned by the Developer. Copyright of certain third party open source libraries used by DigiFlow remains with the developers of those libraries. Copies of the software and documentation may be made for backup purposes. Additionally, copies may be made of the manuals for the purposes of training and use of DigiFlow *provided* such copies retain their original copyright declaration.

10. Reverse engineering. The Customer may not decompile, disassemble or otherwise reverse engineer any component of DigiFlow. Further, the customer must not attempt to break, bypass or disable the licence key system controlling the use of DigiFlow.

11. Transfer of Ownership. The Customer may not sell DigiFlow or any code developed under the Development System without prior consent from the Distributor. The customer may,

however, distribute images processed by DigiFlow and DigiFlow dfc macro code without restriction.

12. Other Copies. All the terms of this licence apply equally to the original supplied version of DigiFlow, to any upgrades or updates to the *DigiFlow* or subsequent systems derived from Digi*Flow*, or documentation which may be supplied from time to time by the Distributor, and to any copies made of Digi*Flow* or its updates under the terms of this licence.

Warranty:

13. Limited Warranty. The Developer guarantees holders of Commercial Licences that Digi*Flow* will perform substantially in accordance with the accompanying documentation. No such guarantee exists for holders of Free Licences. The Developer disclaims all other warranties either express or implied.

14. Period. The period for the stated and any implied warranty is limited to 90 days from receipt of a Commercial Licence for DigiFlow. Updates or upgrades to DigiFlow outside this period carry no warranty.

15. Consequential Damages. Neither the Developer nor their suppliers shall not be liable for any damages whatsoever arising out of the use, misuse or inability to use DigiFlow or any updates or upgrades to this system.

Other Conditions:

16. Support. User support for holders of Commercial Licences will be provided by the Developer free of charge for the first year following receipt of the System. This support will normally consist of a combination of electronic, written and verbal communication. On-site training is not included. The Developer reserves the right to alter the precise nature and scope of this support. Support for users with Free Licences is at the discretion of the Developer.

17. Upgrades. The Developer undertakes to make available, free of charge, any upgrades or updates to DigiFlow released by the Developer during the first year after the receipt of a Commercial Licence, *provided* the appropriate licence fee has been paid. This undertaking does not imply that Developer is obligated to modify DigiFlow in any way, or release any modifications made to DigiFlow.

18. Expiration. Licence Keys are perpetual for the version of DigiFlow for which they were issued. The Developer, however, reserves the right to change the licence key mechanism in future versions of DigiFlow in a way that may render invalid Licence Keys from earlier versions. In such cases, Holders of Commercial Licences within their upgrade period will be entitled to a new Licence Key. The Developer also reserves the right to discontinue the issue of Free Licences.