ImageStream

Version 6.00

System Theory and Design

by Dr Roger Nokes

February 2007

ImageStream

Version 6.00

System Theory and Design

February 2007

Department of Civil Engineering
University of Canterbury
Christchurch, NZ
roger.nokes@canterbury.ac.nz

Design and layout: Melody Callahan

1 Introduction to Image Processing

1.1 Flow visualisation

Experimental fluid dynamics has always used flow visualisation as a way of understanding and interpreting fluid flows. The techniques that have been employed have varied from the simple, such as sprinkling leaves or twigs onto the surface of a stream and watching the individual objects wend their way downstream, to the sophisticated, an example of which is **laser-induced fluorescence** (LIF) where high intensity laser light is used to optically excite dye molecules within a flow. Each technique has its purpose, its limitations and its advantages. The books by van Dyke (1982) and Smits and Lim (2000) demonstrate the range of techniques that have been employed over the years, the beauty of the images that they produce, and the imagination possessed by their inventors.

Flow visualisation techniques fall into two broad categories – qualitative and quantitative. Qualitative techniques are designed to allow the researcher to "see" the flow – to gain a feeling for its structure and peculiarities. Quantitative techniques aim to measure flow fields. Qualitative techniques are frequently simpler to set-up than quantitative techniques, as calibration is rarely an issue.

Some qualitative techniques require the addition of some type of tracer to highlight the flow. For example, **streak photography** (see Stamp, Hughes, Nokes and Griffiths 1998) which, although it can be used for quantitative analysis, has often been used to simply visualise the flow. Here, particles, that are assumed to be dynamically unimportant, are added to the flow, illuminated in some fashion, perhaps with a light sheet, and tracked through long exposure photography. The resulting images reveal light streaks that are long or short depending on the speed of the fluid, and which track the fluid motion.

Other qualitative techniques use properties of the flow itself to highlight the motion. A number of good examples exist in the area of stratified fluids, where the density, and, hence, optical refractive index, vary throughout the flow. The simple technique, of shining

light through a tank containing fluid of varying density, and observing the images on a translucent screen placed on the front of the tank is known as **shadowgraph**. Changes in refractive index cause the light rays to deviate from their original path, leaving a complex pattern of light and dark lines and regions on the screen. A superior technique, known as **schlieren** (see Stamp et al 1998), uses concave mirrors to transmit the light from a point source, through a tank as a parallel beam, and then refocus the image onto a knife edge in front of a camera. This technique provides more informative images than the shadowgraph technique.

Quantitative techniques are designed to measure field values within the fluid flow. Due to advances in data capture – principally through digital video – and low cost computing power, quantitative techniques have developed rapidly over the last 20 or 30 years.

The type of information that the researcher seeks through these types of techniques varies from application to application. However the estimation of fluid velocities, and derived quantities such as vorticity or turbulent stresses, is common to all areas of fluid dynamics. A huge amount of effort has gone into the development of reliable and accurate velocity measurement techniques using flow visualisation tools. The dominant technique is known as **particle image velocimetry** (PIV) (see Raffel, Willert and Kompenhans 1998) and its use is now widespread in many areas of fluid dynamical research. Another technique, **particle tracking velocimetry** (PTV), is based on similar technology but has received less attention (see Cowen and Monismith 1997, Blackett 1994 and Nokes 2007). Both techniques seed the flow with tiny particles that are illuminated with a sheet of light. Digital video cameras are used to capture the motion of the particles and sophisticated algorithms are used to analyse the resulting images in order to deduce the fluid velocities. PIV relies on cross-correlation of light intensity between frames to estimate the fluid velocities, while PTV attempts to track particles from frame to frame for the same purpose. The *FluidStream* system, also developed at the University of Canterbury is an implementation of a PTV system.

Another common piece of information sought by researchers is the concentration of some scalar quantity in the flow. This scalar could be some contaminant added to, and mixed within, the flow, or it could be some intrinsic property of the flow, such as the fluid density. A modification to the schlieren system described above (see

Ivey and Nokes (1989)), which measures the degree of light deflection by the flow, allows quantitative deductions regarding fluid density to be made.

However, most modern techniques in this area use a dye of some sort to tag the scalar of interest and to attempt to measure the concentration of this dye. LIF, as described above, is perhaps the most advanced of these. It has been used successfully in the experimental analysis of jets and plumes (see Davidson and Pun 1999) but it is certainly not limited to this area. High quality data can be obtained that provides not only the mean dye concentrations, but also the concentration fluctuations. Other cheaper and simpler techniques are being used with more frequency. Included in these is the use of **light attenuation** by simple food dyes to measure dye concentrations (see Holford and Dalziel 1996). Such techniques seem to offer good quantitative data for two-dimensional flows or depth-averaged data for three-dimensional flows. Certainly more opportunities exist in this area.

ImageStream is an image processing application developed in the Department of Civil Engineering at the University of Canterbury. While somewhat general, *ImageStream*, is particularly focussed on the deduction of dye concentrations using LIF or dye attenuation techniques.

1.2 Operational Issues

Image processing in experimental fluid mechanics usually includes 2 steps

- Image capture
- Image processing

Each of these steps is discussed in more detail in sections 1.2.1 and 1.2.2, both in a generic way, and with a particular focus on the *ImageStream* system.

1.2.1 Image capture

Image capture involves a number of issues. These are

- Lighting
- Dyes
- Frame capture

1.2.1.1 Lighting

Appropriate lighting is one of the biggest challenges in experimental fluid mechanics. A range of different types of light sources can be needed. In some applications a point light source is required, in others a light sheet and in others a parallel beam of substantial lateral dimension.

A wide variety of point light sources can be used. None are strictly point sources, but depending on the application, most fulfill this role adequately. Examples of such sources are standard incandescent light bulbs, slide projectors, halogen bulbs, and lasers.

Light sheets are considerably most challenging. The cheapest, but least flexible, light sheets are generated from white light sources such as a slide projector or high-intensity halogen bulb. A slit arrangement is generally required to provide the sheet, and due to the nature of white light the generated light sheet will diverge somewhat over the flow domain. The degree of spread, and the implications for the experimental results, will depend on the geometric configuration of the lighting system and the nature of the flow itself. The sheet thickness is generally easily changed for such a system by altering the slit size. However, intensity attenuation, due to spreading, can be an issue over long distances. One of the biggest challenges with a white light system is achieving high enough light intensity to allow the video capture equipment to “freeze” the flow.

Laser light sheets are commonly used in LIF, PIV and PTV work. High power lasers, typically 1-2W, are used, although at the time of writing, such laser systems are still very expensive. Laser generated light sheets have a number of advantages over those obtained from white light sources. In particular, the laser sheet is generally very thin, the order of 1 or 2 mm, and because of the ability to collimate a laser beam, the spread angle of a laser sheet is very small, normally about 0.5-1 mrad. Pulsed lasers, with the ability to produce very

high intensity light for a very short time, can be used to “freeze” high speed flows.

Wide beams of light are also challenging. Slide projectors are a simple source of a wide beam of light, although issues of divergence and light intensity may make them impractical for some applications. For dye attenuation studies we have used a bank of 8-10 fluorescent tubes behind a thin (3mm) sheet of semi-opaque acrylic. The acrylic acts to produce a more uniform light beam, and intensities across the majority of the beam are similar. This configuration appears to be superior to other set-ups including slide projectors or halogen spotlights.

The *ImageStream* system can be used with any type of light sheet. However, the overall quality of the results obtained will be somewhat dependent on the choice of light source.

1.2.1.2 Dyes

A large range dyes can be used for this type of visualisation work. Permanganate crystals used to be a standard technique for marking fluid particles. Nowadays food dyes are commonly used for dye attenuation studies, while special fluorescent dyes that have an excitation frequency matching that of the available laser system are used for LIF work.

1.2.1.3 Video capture

A large range of frame capture equipment is currently available. Digital still and digital video cameras are most commonly used.

Modern digital still cameras generally provide a low resolution, digital video mode, as well as a slow automated sequential frame capture capability.

Digital video cameras produced for the mass market are satisfactory tools for image capture in the laboratory provided the lighting and resolution demands are not too great. The modern digital video standard (PAL) has 720 pixels per video line with 576 lines per frame. Unfortunately most cameras on the consumer market today only capture video in interlaced mode (ie two half frames, $1/50^{th}$ of a second apart, are combined to give a complete frame every $1/25^{th}$ of a second). Progressive scan, or non-interlaced recording, whereby

the whole frame is grabbed at once, used to be common in the more expensive cameras, but most, if not all, manufacturers seem to have discontinued this feature, apparently through lack of consumer demand. For image processing work progressive scan is highly desirable, particularly if the flow speeds are high.

High definition video (HDV), with resolutions better than double that of standard PAL, is also now becoming available at reasonable prices.

One of the most attractive reasons for using mass market video cameras for video capture is not only their value for money, but the fact that software for capturing images directly onto computer, and manipulating these images, is readily available. *Adobe's Premiere* application is used in the fluids laboratory at the University of Canterbury for much of the video data acquisition. A digital video camera is connected through a FireWire (IEEE 1394) cable to the computer, and *Premiere* is used to control the camera remotely.

Higher resolution digital video cameras are available for more specialised scientific work. These cameras can be colour or gray-scale (8, 10 or 12 bit) and have resolutions up to, and beyond, 2000x2000 pixels. The software for controlling these cameras is specialised. The majority of these more expensive cameras provide a progressive scan mode.

One of the concerns in choosing a camera for image processing work is the camera's ability to work with low light levels. In general, we have found that digital video cameras provide much better performance than digital still cameras. However, even within the digital video camera range, the low light performance varies significantly. Manual control over focus, frame speed and *f* number are also crucial.

A range of digital image formats is currently supported by cameras and associated software. For example, digital video formats include AVI, Quicktime, and MPEG. Digital still images are normally stored in TIF, JPEG, GIF, PNG or BMP formats. The *ImageStream* system provides support for sequences of JPEG, TIF, PNG, BMP or GIF images. It does not directly support any digital video format. *Premiere* allows the user to export a digital video as a sequence of GIF or TIF files, each frame stored in a separate file.

1.2.2 Image processing

Once captured, the image files generated by the video capture system described above are loaded into *ImageStream* (or any other image processing software) and the pixel colour intensities decoded. For a 32bit RGB colour image the intensities of the three colour guns at each pixel – red, green and blue – are represented by 3 8bit integers. Thus each colour has a maximum intensity (ie the image is saturated with the colour) of 255, and a minimum intensity (ie none of that colour appears at a pixel) of 0. Poynton (1996) provides a good introduction to the technical issues surrounding the representation of light in RGB format. For a 10bit gray-scale image, the intensity of light at each pixel is represented by a single 10bit integer. Pure white has a value of 1023, while black has a value of 0. Similarly 12-bit gray scale measures a single intensity between 0 and 2047.

Once pixel colour information is available the image processing algorithms manipulate the light intensities in order to deduce the flow quantity of interest. For example, the light intensity from a 10bit gray-scale camera may be linearly related to the concentration of fluorescent dye at each pixel.

1.3 Summary

This chapter has aimed to provide the reader with some essential background in the operation of a typical image processing system. A brief introduction to the capabilities of the *ImageStream* software has also been provided.

2 Overview of the ImageStream system

2.1 Introduction

The previous chapter has provided a brief introduction to digital image processing in the area of fluid dynamics research. The remainder of this manual will focus on the *ImageStream* system, an image processing package developed in the Department of Civil Engineering at the University of Canterbury. The reader is encouraged to read the accompanying manual, **ImageStream 6.00 – User's Guide.**

In this chapter an overview of *ImageStream* will be given. This will provide an introduction to key concepts in the software design as well as information about the system's operation.

2.2 User interface

ImageStream is a graphical user interface based software application written in the Java programming language. All of the usual interaction mechanisms are available, namely pull-down menus, popup menus, list boxes, radio and check boxes, buttons, combo boxes, dialog boxes, windows etc.

In addition, *ImageStream* has been developed with an object-oriented user interface (see Nokes (2002)). This type of interface should provide the user with a very natural means for interacting with the data stored within the system.

ImageStream deals with a number of entities that, in the software arena, would be referred to as **objects**. These software objects represent things associated with the image analysis process. For example, one of these objects is the **image sequence** (see chapter 3). It corresponds to a sequence of video images that can processed at some later stage. Even *ImageStream* itself is an object that is created when the application runs.

Objects are usually displayed for the user in a list box, where the object's name and its associated icon are shown.

Each object in the *ImageStream* system is manipulated in a consistent way. A popup menu can be displayed by right-clicking the mouse on the object when it is displayed in a list box. This popup menu allows the user to view information about the object, and to perform operations on it – for example calculating intensity fields from an image sequence.

Each object supports a number of window-based **views** of itself. The views are available to the user, either through a popup menu selection, or by double clicking on the object. For each type of object a default view is defined, and it is this view that is displayed when the object is double clicked. As an example the image sequence has three views, each of which provides different information to the user. The image view is the default view. These views are discussed in detail in the accompanying **User's Guide**.

The user can choose to simultaneously display as many views of an object as they wish – even displaying the same view more than once. Whenever changes occur to an object each view updates its information automatically. When an object is closed – removed from the *ImageStream* system – all of its views are closed with it.

Each view has its own menu bar that provides the user with settings associated with that view. For instance, regions can be deleted through the *Region* menu of the image sequence image view.

This model of object, popup menu and views is consistent across all objects in the *ImageStream* system.

The *ImageStream* system itself is an object with four views. These views are displayed on opening the application. These are described briefly below.

2.2.1 Application view

This window acts as the parent for all other windows in the application. In other words, it acts as a backdrop to all other windows and when other windows are minimised they appear at the bottom of the application view. When the application view is minimised all other windows are minimised with it.

This window contains nothing except the menu bar associated with the *ImageStream* application. There are seven menus, *Application*, *View*, *Sequence*, *Intensity*, *Field*, *Image*, and *Info*. These are discussed in detail in the **User's Guide**.

This view is special in that closing it causes the application to exit.

2.2.2 Image sequence list view

A number of image sequences (see chapter 3) can be open within *ImageStream* at the same time. Each sequence is listed in the *ImageStream* image sequence list view, and can be manipulated from this view. This view can be closed, and reopened by selecting the *Sequence list view* option in the *View* menu.

2.2.3 Intensity field list view

A number of intensity fields (see chapter 5) can be open within *ImageStream* at the same time. Each intensity field is listed in the *ImageStream* intensity field list view, and can be manipulated from this view. This view can be closed, and reopened by selecting the *Intensity field list view* option from the *View* menu.

2.2.4 Field list view

The user can create, and save to disk, a variety of fields derived from an intensity field. For example, the user can calculate the time averaged intensity field from an intensity field time series. The field list view displays all field objects that have been opened in *ImageStream*.

2.3 Subsystems

ImageStream comprises three subsystems. These correspond to

- Image sequence definition
- Image processing, and
- Intensity field calculation.

The system does not provide support for image capture.

The image sequence definition subsystem will be discussed in chapter 3. The image processing will be discussed in chapter 4, and the intensity field calculation in chapter 5. In each chapter the objects associated with each subsystem will be introduced and discussed. Detailed information about the views available for these objects, and what options are presented in their popup menus are given in the **User's Guide**.

2.4 The Java environment

ImageStream is written in the Java programming language. In theory this means that the program should be able to run on any computer platform provided a java runtime environment (ie an implementation of the Java Virtual Machine (JVM)) is installed on the system. If your computer does not have a JVM installed then an appropriate JVM (this is platform dependent) can be downloaded from Sun's Java website. The address is

http://java.sun.com

Version 1.4 of the virtual machine is required.

Many computer systems have a JVM installed with their web browser to allow java applets to be downloaded from the web and run on that computer. However some of the latest versions of *Windows* do not provide this.

ImageStream has been developed in a Windows environment, and tested extensively on computers running the *Windows 2000* and *Windows XP* operating systems. It has also been tested on *Apple Macintosh* systems running OS X.

2.4.1 Performance Issues

Java has not been traditionally used for computationally intensive software applications due to the fact that it was designed to be an interpreted language. In fact, early in its development, its performance in this area was known to lag well behind such languages as FORTRAN, C and C++. However, the introduction of JIT (Just In Time) compilers that produce native code, and thus avoid the speed limitations associated with interpreted code, has led to substantial improvements in computational performance.

The designers of *ImageStream* performed some simple tests in order to compare the computational performance of java with that of C++. A number of the core algorithms of the sister application, *FluidStream*, were rewritten in C++ and compiled, using *Microsoft Visual C++*, into *Windows* 32bit applications. The results of this benchmarking suggested that the computational performance of the two languages was comparable.

2.4.2 Memory Management

The processing of large numbers of image files, and the manipulation of their pixel colour maps places heavy demands on the memory and processing power of the computer. The systems currently used at the University of Canterbury have processors with clock speeds in excess of 2GHz, and a minimum of 1 Gb of RAM. Large, 120+ Gb hard, disks, used solely for storing images and processed data, are also installed, and CD/DVD writers for archiving data are seen as essential to a viable laboratory system. Even with these relatively impressive specifications, by current standards at least, the system can still take considerable time (of the order of 10s of minutes) processing large video records (eg *1000+* images). Users of *ImageStream* who wish to handle long video records should try to access the highest performance machine possible for their work.

Due to this heavy demand on computer memory the system's memory management is of vital importance. When the system is first installed the user specifies the amount of RAM that is available to *ImageStream* for its storage and processing. See Appendix A for how this memory limit can be set.

The Java programming language handles memory in a particular way. Memory is dynamically allocated within the program when it is required, but recovery of this memory once it is no longer needed is beyond the programmer's control. Instead, Java has an automatic garbage collection mechanism that recovers memory automatically. While this relieves the programmer of considerable effort in memory management, it also leaves the responsibility of recovering memory with the Java runtime environment. If the runtime environment does not perform this task in a timely fashion it is possible for the application to become memory starved and to fail.

ImageStream has been designed to ask the runtime environment to recover memory when the available memory drops below a certain

level. The user can set this level (see section 2.5.1). During memory, and processor, intensive activities, such as the processing of a large video record, the user may be informed that the garbage collector is running, due to the fact that the available memory has dropped below the preset level. This is normal behaviour, although, if it happens frequently during a particular process it may indicate that a RAM upgrade could be beneficial.

The user can ask the garbage collector to run manually by selecting *Run garbage collector* from the *ImageStream Application* menu.

The total amount of memory available to *ImageStream*, and the amount of memory currently unused, can be viewed by selecting *Memory* from the *Info* menu. This displays a dialog box that updates these statistics every second.

Occasionally the Java runtime environment fails to recover unused memory when it is required. This event generates an error that is displayed in the console window (see section 2.5.2) but it generally does not result in the application crashing. If this does occur all data should be saved immediately, and the program exited.

2.4.3 Versioning

As improvements to *ImageStream* are made new versions of the software are released. Data files created with older versions of *ImageStream* may not be compatible with the new version (this is due to the method by which Java stores data on disk). However *ImageStream* allows the user to save most data objects in a text format, that will be readable by future versions. You are encouraged to keep copies of each version of *ImageStream* after the release of each new version. A recommended strategy is to store the correct version of *ImageStream* with each set of files created by it. *ImageStream* data files can be substantial in size, depending on the length of the video record. The additional memory required to save the *ImageStream* files should be relatively unimportant.

2.5 Application basics

In this section a number of miscellaneous topics relating to the *ImageStream* application are discussed.

2.5.1 Configuration file

ImageStream creates a configuration file in the user's home directory when *ImageStream* is registered and run for the first time (see Appendix B). On subsequent occasions *ImageStream* will load this file on start-up. When the application exits the current configuration settings are saved to the configuration file.

The configuration file provides a number of user selectable parameters. These are

Current directory

Whenever the user tries to open a file, or save a new file, *ImageStream* will use the current directory as its starting point. Whenever the user changes the directory when opening or saving a file, the new directory is stored.

Minimum memory level

If the memory available to *ImageStream* drops below this level during certain memory intensive processes, then the garbage collector will be asked to run. This has a default value of *40Mb*.

Garbage collector wait time

When the garbage collector is requested to run the time allocated to it, before normal operations resume, is specified with this parameter. This has a default value of *1000ms*.

All of these parameters can be changed by selecting *Properties* from the *Application* menu. Also displayed in this dialog box is your **Home directory**. This parameter cannot be changed.

2.5.2 Console window

When the user performs an illegal operation a dialog box explaining the error is normally displayed. However, it is possible that some errors are not detected by *ImageStream* itself, but are caught by the runtime environment. This should be a rare occurrence, but when it happens an error message will be displayed in the black console window that is opened when *ImageStream* first starts. Examples of errors that will be displayed in the console window are out of

memory errors. In this case the user should save his/her data and close the application.

The console window will be open throughout a *ImageStream* session. Clicking the close button on this window will ungracefully exit the application. By this is meant that the user will not be prompted to save unsaved data. All such data will be lost. The console window should be minimised at start-up and not touched again unless the user realises that an error has occurred.

2.6 Summary

This section has provided a broad overview of some of the fundamental design concepts behind the *ImageStream* image processing system, as well as discussing issues associated with the Java programming environment.

3 Image Sequences

In this chapter we will describe the image sequence object in detail, including how it can be created.

3.1 Overview of image sequences

The basic object in the *ImageStream* system is the **image sequence**. It represents a sequence of digital images captured during an experiment. These images may have been obtained with a digital video camera and later converted into a sequence of individual images (in JPEG format for instance), or captured with a digital still camera that can store the individual images directly in an appropriate format. The images are assumed to correspond to a time series of images, all of the same size, captured at a uniform rate. The time interval between images is specified by the user when the image sequence is created.

Image sequences work with digital image files. The file formats that are supported are JPEG, GIF, PNG, BMP and TIF. *ImageStream* does not work directly with digital video file formats such as AVI or MPEG. Therefore all video records must be converted to individual image files before *ImageStream* can process them. *Adobe Premiere* is an example of a video editing program that can perform this conversion – at least to GIF and TIF file formats.

The choice of format is important. Some file formats are small and lossy, others are large and lossless. Others, such as GIF, have a limited colour palette and therefore will limit the amount of quantitative information that can be garnered from the images. We would recommend that PNG, BMP and TIF are the best choices as all three are lossless.

3.2 Image data types

The image file formats discussed above dictate the manner in which the pixel information is stored within a file. Strictly speaking the way in which that information is interpreted is at the discretion of the programmer. *ImageStream* needs to be able to work with a range of data types. Clearly one of these data types is a standard RGB image where the red, green and blue guns are each stored in an 8 bit

segment of the full 32 bit data value at each pixel. The remaining 8 bits are known as the alpha value and are often not used when interpreting the pixel colour information.

However, *ImageStream* also needs to work with gray-scale images and, at times, images containing real numbers. Therefore *ImageStream* defines a number of formats to accommodate this variety of data types. When 10-bit gray scale images are stored the most significant 8 bits are stored in the red gun and the least significant 2 bits in the green gun. For 12-bit gray scale images the most significant 8 bits are stored in the red gun and the least significant 4 bits are stored in the green gun. Real value images, which simply store a floating point value at each pixel location, use the full 32 bit colour code (red, green, blue and alpha values) to store a 32 bit floating point number. The conversion of real data to 32bit colour is achieved by copying the bit pattern of the 32bit IEEE floating print number directly into the pixel colour value. Thus all of these data types can be stored in a standard RGB image and stored on disk using any of the supported image formats (JPG, BMP etc). It should be noted however, that when storing real data in an image it is essential that a non-lossy format is used.

3.3 Image sequence creation

Image sequences are created by selecting *Create image sequence* from the *Sequence* menu in the *ImageStream* application view. Once created the image sequence is displayed in the *ImageStream* image sequence list view where the user can interact with it through its popup menu.

The user selects the image files that form this image sequence. The image sequence object does not load these images and store their pixel intensities internally. The image sequence simply refers to the images through their filenames, loading them when they are to be processed. For this reason it is important that the image files are not moved once the image sequence has been created.

All images in an image sequence must have the same width and height (in pixels). The first image defined in the sequence is used to determine this width and height, and when the image sequence is created all other images in the sequence are checked to ensure that they possess the same dimensions as the first image. Images that do

not conform to these dimensions are excluded from the image sequence.

When an image sequence is created the user must specify the type of data that is stored in the image.

3.4 Opening and saving image sequence files

Image sequences can be saved to disk through a popup menu choice. All information relating to the sequence is saved with it. Image sequences, that have previously been saved, can be opened from the *ImageStream Sequence* menu. All image sequences are stored with a *.isq6* extension.

In addition to this method of storing image sequences *ImageStream* allows the user to save and reload image sequences to and from raw text files. The advantage of using text files is that software version changes will not prevent data created using previous versions being loaded into *ImageStream*. Raw text files have the extension *.stx6*.

3.5 Image processing

All image processing is performed on image sequence objects. This processing is performed by filter objects that will be discussed in depth in the next chapter.

3.6 Image sequence views

The image sequence provides three views designed to give the user access to different aspects of the sequence.

3.6.1 Image view

The image view is the default view for an image sequence. It provides a simple video player interface that allows the user to review the images in the image sequence. The images can be viewed one frame at a time, forwards or backwards, through the sequence, or replayed as an animation, again either backwards or forwards. The chief limitation of this animation is that images must be loaded from

disk before they are displayed and therefore the sequence may not be able to be played in real time.

The image view also allows the user to explore the image pixel information. As the cursor moves over an image the physical coordinates at the location of the cursor, as well as the information stored at the pixel beneath the cursor, are displayed. The information is dependent on the type of data stored in the image. It may be RGB, 10-bit gray scale, 12-bit gray scale or a real number. It should be noted that during an animation the pixel information is not updated (in order to save time).

Ultimately the images in an image sequence will be processed through the use of one or more filters. These filters are defined in the filter view (see section 3.6.3) of the image sequence. The image view permits the user to view the images in their raw form or after they have been filtered using the defined filters.

The image view also allows the user to create regions. These regions are areas within an image that can be used as part of a filter (see section 4.3). The regions can be rectangular, circular or polygonal.

3.6.2 Details view

The details view, like details views for all objects in the *ImageStream* system, simply displays fundamental information about the sequence. Much of this information is entered by the user when the sequence is created. It includes such things as the time step between images (assumed constant), the size of each image in pixels and the scale factors used to convert from pixels to mm. A list of the image files, in numerical order, is also displayed, allowing the user to interact with the images within the sequence directly by double clicking their filename.

It is important to note that the actual digital images are not stored within the image sequence object itself, only the filenames that correspond to these images. If these image files are deleted or moved, the image sequence will no longer be able to access them. Care must be taken to ensure that image files remain in place during the lifetime of an image sequence object.

3.6.3 Filter view

The filter view provides an environment in which the user can define transformations that can be performed on the images within the image sequence. These transformations are termed filters and will be discussed in chapter 4. More extensive detail is provided in the **User's Guide**.

Within an image sequence the raw images themselves normally only provide qualitative information about a flow. It is through some sort of image processing that more quantitative information can be extracted from them. Filters provide this image processing. A good example is the sequence of images obtained using a laser induced fluorescence (LIF) facility. The images in this case capture the intensity of light emitted by the fluorescing dye, which, with careful calibration, can be converted into quantitative measurements of the dye concentration. *ImageStream* includes a filter than can be used to convert intensities to concentrations pixel by pixel in an image.

The filter view allows the user to create filters that can be applied to the image sequence in order to obtain this quantitative information. Facilities are also provided for editing, and deleting these filters. The user may choose to apply the filters currently defined for the image sequence, thus creating a sequence of filtered images and a new image sequence containing them.

3.7 Image sequence transformations

Six different transformations are available that can be applied directly to an image sequence through the image sequence's popup menu. These are

3.7.1 Format transformation

The format transformation allows the user to convert all of the images in the sequence into an alternative format. The user simply selects a file name for the converted images (to which a 5 digit number will be added) with the extension of the desired format.

3.7.2 Trimmed transformation

This transformation takes each image in the image sequence and trims it's borders according to a list of insets specified by the user. The resulting image is a rectangular sub-image of the original image. The user selects a filename for the trimmed images (to which a 5 digit number will be added) with the extension of the desired format. This format can be different to the original image format.

This trimmed transformation is a useful way to save memory when only a portion of the original image is of interest.

3.7.3 Average colour transformation

The average colour transformation takes all images in the image sequence and averages each of the colours at each pixel. The resulting single image is stored in a file specified by the user. When only time averaged information is required this transformation can save considerable time in later processing tasks.

The user can choose to apply the filters for this sequence during the averaging process.

3.7.4 Average colour constant velocity transformation

The average colour constant velocity transformation takes all images in the image sequence and translates them in space according to the chosen velocity vector. A positive velocity in the x direction shifts the images in the negative x direction and a positive velocity in the y direction shifts the image in the negative y direction. Thus a positive velocity corresponds to a "flow" in that direction and the transformation attempts to transform the images into a steady state reference frame.

Once the images have been shifted in space they are averaged in time. The result is a single image that is considerably larger than the original images. This image is stored in a file specified by the user.

The user can choose to apply the filters for this sequence during the averaging process.

3.7.5 Average value transformation

This transformation is equivalent to the average colour transformation but works with real value images. Thus the average value at each pixel for all images in the image sequence is calculated.

3.7.6 Average value constant velocity transformation

This transformation is equivalent to the average colour constant velocity transformation but works with real value images. Thus the average value at each pixel for all images in the image sequence is calculated subject to a constant velocity imposed on the images.

3.8 Summary

This chapter has provided an introduction to image sequence objects. Filter objects, described in chapter 4, work with image sequence objects to provide the image processing capability of *ImageStream*.

4 Image Filtering

4.1 Introduction

Filters are at the core of *ImageStream*'s image processing capabilities. A filter is defined to be an algorithm that can be applied to the colour intensities (we use the word intensity to mean any type of data that is stored in a pixel – see section 3.2) of a digital image in order to produce another digital colour image. Filters are defined for image sequences and are applied to the images stored in them.

Filters can be **piped** so that the output of one filter can be used as the input to the next. If an image sequence has multiple filters defined for it, the application of these filters will cause each image in the sequence to be fed into the filter pipeline, starting with the first in the list, and finishing with the last. Intermediate images, resulting from the application of some, but not all, of the filters are discarded.

The filter pipeline is checked for its integrity by the *ImageStream* software. Each filter in the pipeline must have an input data type that is consistent with the output of the filter that precedes it in the pipeline. For the first filter in the pipeline the input data type must match the data type of the images of the image sequence itself. If a filter pipeline contains inconsistent input/output data types the user will not be able to apply the filters to the image sequence until any conflicts have been rectified.

The application of filters to an image sequence can occur in two ways. Firstly the straight application of the filters will lead to the production of a set of image files, containing the filtered images. The user can select from a number of formats for these filtered image files. At the same time a new image sequence is created that contains these filtered files. This sequence appears in the image sequence list view.

Secondly the filters can be applied to the image sequence as part of the process of creating intensity fields. In this case the result of the filtering process is used in the construction of the intensity field but is not saved to disk. If the filtered images are not required for another purpose, creation of the intensity field with the filtering

process incorporated into it is rather quicker than producing the filtered images as an intermediate step.

4.2 Filter inputs and outputs

Each filter has a specified data type for its input and output images. As discussed in section 3.2 this data type does not refer to whether the image is stored in JPEG, PNG or any other image format, but rather how the data in that image is to be interpreted. Most filters accept 32bit RGB colour images as their input and produce 32bit RGB images after processing the pixel intensities. For example an extract filter takes a 32bit colour image and extracts one of the colour intensities, or an average of them, from the image. The result has the same data type as the original.

However, a number of filters do not produce 32bit RGB colour images. For example the polynomial field calibration filter takes pixel intensities and converts them to real scalar field values generally corresponding to the concentration of a dye of some sort. *ImageStream* uses the 32bit pixel values to store these real numbers, so that any image manipulation software would be unable to tell that the stored image did not store 32bit colour data. The advantage of this system is that all input and output formats can be handled in a consistent way, and even real data can be stored in file formats such as PNG. A word of caution here. Any lossy file format, such as JPEG, is inappropriate as a format for storing real data. The slight inaccuracy of such a format is of little consequence for colour images, but can have a dramatic effect on floating point data stored in the file.

4.3 Filters

In this section the full array of filters available in ***ImageStream*** will be described. Further information can be found in the **User's Guide**.

4.3.1 Amplify filter

An amplify filter can be used to scale the intensity at each pixel. The filter can be defined for any data type (ie RGB, 10-bit or 12-bit gray scale and real value) and its output data type matches its input data type.

For all data types the amplification may produce an intensity that is outside the allowable range. In this case the result of the amplification is replaced by the maximum allowable value for that data type. For example, if an RGB image is amplified and the red gun at some pixel is amplified to a value beyond 255, then the red value at that pixel will be set to 255.

For RGB images the red, green and blue guns can be amplified independently.

4.3.2 Absorption colour filter

The absorption colour filter has been designed to work with the light attenuation technique described in chapter 1.

When light passes through a dyed solution one or more of the colour intensities is attenuated due to light absorption. For instance, a red food dye, in low concentrations, absorbs green and blue light but leaves the red light unaffected. A simple linear theory of light attenuation (see Holford and Dalziel 1996) suggests that the intensity of light passing through an optically dense medium will decay exponentially, where the rate of decay is given by the optical thickness. The optical thickness will depend on a number of things, but most importantly the thickness and concentration of the attenuating material.

Thus

$$d(x,y) = -\ln\left(\frac{I(x,y)}{I_0(x,y)}\right) \qquad (4.1)$$

where $d(x,y)$ is the calculated optical thickness, $I(x,y)$ is the intensity at a pixel, while $I_0(x,y)$ is a reference intensity at the same pixel. Here, the intensity stands for the intensity of the red, green, blue gun, or the gray scale intensity.

Two different types of absorption filter are available, depending on the manner in which the reference intensity is calculated. The absorption colour filter is discussed in this section while the absorption image filter is discussed in the next section.

To understand the absorption colour filter consider an experimental set-up where white light passes through a tank that initially contains clean water. As the experiment proceeds the water, through which the light passes, changes so that it contains low concentrations of red dye. Changes in green light intensity can occur due to two factors. Firstly changes in the overall light intensity - the reference intensity referred to in equation 4.1 – and secondly, absorption of the green light by the red dye. Consideration of only the green light does not allow the experimentalist to differentiate between these two sources of attenuation and hence the concentration of red dye cannot be determined.

If the white balance on the camera is initially set so that all colour components have the same intensity, then any change in the red intensity (which is independent of the dye concentration) will be due to changes in the overall light intensity. Thus the red light can be used to indicate the reference intensity I_0. Comparing the changes in green light intensity relative to the red light intensity allows the optical thickness to be deduced.

Absorption colour filters work in the way described above. The attenuation of one colour intensity relative to another is calculated using equation (4.1). The user can select any combination of the colours.

The absorption colour filter takes an RGB input and produces a real value output.

4.3.3 Absorption image filter

The absorption image filter differs from the absorption colour filter in the way it calculates the reference intensity in equation (4.1). Instead of using the intensity of one of the other colour guns to normalise the measured intensity, the intensity of the same colour in a reference image is used. If the same experimental configuration described in section 4.3.2 is considered again, an image of the clear water, before any dye has appeared, would act as the reference image.

The absorption image filter takes an RGB input and produces a real value output.

4.3.4 Barrel pincushion distortion filter

Most camera lenses involve some level of optical distortion. While low quality lenses tend to be worse than those of higher quality, even high quality lenses suffer from distortion when used in a wide-angle mode.

One of the most common distortions is described as a barrel/pincushion distortion. An image suffering from this type of distortion has its edges bowed outwards or inwards, thus giving it the appearance of a barrel or a pincushion.

The barrel pincushion distortion filter provides a mechanism for removing this distortion from an image. It is based on the work of Ojanen (1999). Equation (4.2) describes the transformation used in *ImageStream*.

$$\mathbf{r}_s = \left(c_0 + c_1 r + c_2 r^2 + c_3 r^3\right)\mathbf{r}_d \qquad (4.2)$$

where $\mathbf{r}_s$ is the pixel location in the source (distorted image), $\mathbf{r}_d$ is the pixel location in the destination (undistorted image), r is the magnitude of $\mathbf{r}_d$, and the c_i are coefficients chosen by the user. It should be noted that this transformation depends on the location of the origin used. Generally the origin is taken as the physical centre of the image but the user is at liberty to change this.

This transformation described by equation (4.2) can restore images that suffer from both pincushion and barrel distortion.

The selection of the coefficients in equation (4.2) is difficult to automate. Instead *ImageStream* provides an analysis view for the barrel pincushion distortion filter that enables the user to explore the effect of a variety of parameter choices. To help in this regard the user can overlay a rectangular grid on the original image. The alignment of objects within the image can be readily compared to the grid.

The barrel pincushion distortion filter can take any type of image data. It's output data type matches its input.

4.3.5 Bayer colour filter

Many modern colour video cameras use only one CCD to capture a colour image. This is achieved by placing a colour filter over the CCD so that pixels measure the intensity of only one colour. The filter has a repeating pattern whereby each square of four pixels includes 2 green pixels (diagonally opposite one another), one red pixel and one blue pixel. The full colour image is constructed by interpolating the missing colours from neighbouring pixels. This system is known as a Bayer filter.

We have found that the orientation of the colour filter may vary from camera to camera. A standard Bayer filter camera has a red pixel in the top left corner of each square of four pixels with a blue pixel in the bottom right. The Bayer colour filter assumes this orientation by default. However, it is possible for the user to select an inverted coding whereby the top left pixel is interpreted as blue, and the bottom right as red.

ImageStream provides a Bayer colour filter that takes input from a camera using the Bayer filter technology and converts it into a full colour image. This filter requires gains for each colour to be specified. These gains effectively amplify the red, green and blue intensities. For cameras from the mass consumer market the amplification process is normally carried out within the camera itself, based on the choice of white balance selected by the user. For scientific cameras this balance must be performed in the post processing – hence the gains required in *ImageStream*. These gains can be found by recording images of a white background, and adjusting the gains until the intensities of all three guns are the same.

The Bayer colour filter takes an RGB input and produces an RGB output.

4.3.6 Convert type filter

The convert type filter converts between data types. Thus, suppose the output of one filter is a real number, but the user wishes to convert that to a 10 bit gray scale image with an intensity corresponding to the real number, the convert type filter can accomplish this conversion. Of course in some conversions data accuracy is lost, for example in the illustration just given the fractional part of the real data will be rounded. In addition, some

data types have limits on their ranges (eg RGB values must lie within the range 0-255) so the conversion does its best to map the input data type into the output.

4.3.7 Extract filter

The extract filter isolates one of the colours at each pixel. The colour may be red, green, blue or gray. In the first three cases the intensities of the other two guns are set to zero, while in the final case all three guns are set to the average of their original intensities.

The extract filter takes an RGB input and produces an RGB output.

4.3.8 Identity filter

The identity filter is a “do-nothing” filter. It takes any type of data input and leaves the intensity at each pixel unchanged and of the same type as the input.

4.3.9 Invert filter

An inversion filter inverts one or more colours in the original image. Inverting a colour means subtracting its intensity from its maximum value. Thus an invert filter that inverts the green gun will subtract the green intensity from 255 while the red and blue intensities will be left unaltered. The invert all filter performs the inversion operation on all guns.

The invert filter can also work with 10-bit and 12-bit gray scale images.

This filter is a useful tool for pre-processing particle images that are to be analysed by *FluidStream*. *FluidStream* works only with light particles on a dark background. If natural light is used to capture particle tracks it may be that the particles are dark images on a light background (see Brown 2002). Thus the invert filter can be used to convert the images into a form that can be processed by *FluidStream.*

The invert filter can take any type of input data except real values. Its output data type matches its input.

4.3.10 Overwrite regions filter

The overwrite regions filter provides a mechanism for selectively overwriting the pixel information in an image. The areas of the image that are to be overwritten are defined by a list of regions specified by the user. These regions can be created in the image view of the image sequence.

This filter can take any type of input data. The user must specify the input data type (the output data type is the same) as well the pixel intensity that is to be overwritten in the selected regions. For an RGB image the user will specify all three colour intensities. For a 10-bit gray-scale, 12-bit gray-scale or real value image the user need only specify one intensity value.

4.3.11 Polynomial field calibration filter

The polynomial field calibration filter (PFCF), like the absorption filters, is designed to work with light intensities that are in some way affected by the presence of dye within the fluid, although it is not limited to such applications. In fact the PFCF fits into a rather more general framework.

The PFCF assumes that the light intensity at each point in the flow is dependent on some scalar quantity (eg dye concentration). The user can specify which component of the light signal is related to the scalar quantity of interest. For instance it could be the red gun alone, an average of all three guns, or a 10bit gray scale intensity encoded into the RGB image. The filter calculates the magnitude of the scalar based on separate calibration curves for each pixel in the image. These calibration curves are generated from a set of calibration images. Each calibration image corresponds to a constant value of the scalar being measured. The calibration curves themselves are n^{th} order polynomials, where the user is free to select the order of polynomial employed.

Once the calibration images have been specified the filter is able to take each image in an image sequence and convert the light intensities at each pixel, via the calibration curves deduced from the calibration data, into scalar values. The result is an image containing real values.

From a practical perspective a calibration curve based on the intensities at a single pixel may not be reliable, and a calibration based on the average intensities over a small region in each calibration image may be more useful. The PFCF provides this feature.

The most important application of this filter is to the technique of laser-induced fluorescence (LIF). Here a fluid, laced with fluorescing dye (eg Rhodamine) is excited by a sheet of high intensity laser light whose wavelength corresponds to the excitation wavelength of the dye. The resulting light intensity reaching the camera is dependent on the concentration of dye in the fluid. A set of calibration images, based on the response of a solution of known, uniform, concentration to a certain light intensity, is used to produce the pixel-by-pixel calibration curves.

This filter provides an analysis view that enables the user to explore the calibration. The calibration curve for any pixel in the image can be viewed. Similarly the calibration coefficients can be viewed for the whole image.

4.3.12 Remove filter

The remove filter sets the intensity of one of the guns in an RGB image to zero, thus effectively removing that colour from the image. The user can select the gun that is to be eliminated.

4.3.13 Subtract colour filter

A subtract colour filter subtracts the intensity of one of the colour guns in an RGB image from all three guns. Thus if the user selects to subtract the blue gun, the intensity of the blue gun is subtracted from the red and green guns and the blue gun's intensity is set to 0. If any of these operations lead to a value less than 0 then 0 is used in its place.

The average intensity of two guns (red and green, green and blue, or red and blue) can be selected instead of a single colour in the subtraction process.

.3.14 Subtract image filter

The subtract image filter is somewhat more sophisticated than the simple subtract colour filter. Here the intensities of the pixels in a separate image are used in the subtraction process. Images storing data of any sort can be used – RGB, gray scale or real values – provided the images in the image sequence have the same data type. The output data type matches the input in each case.

The subtraction process can occur in two different ways. Either the intensities of the pixels in the chosen image are subtracted from the pixel intensities of each image in the image sequence, or alternatively, the pixel intensities of each image in the sequence are subtracted from those of the separate image. For RGB images each gun is subtracted independently – ie red from red etc. By definition an image subtracted from itself results in an image whose every pixel has an intensity of zero.

4.3.15 Threshold filter

The threshold filter is used to set pixel intensities to 0, based on some threshold criteria. All pixels whose intensities are less than, or greater than, a specified value, meet the threshold condition and their intensity, or some component of it, is set to 0.

The user can select the threshold value and whether the criteria is “greater than” or “less than” the threshold value. For RGB images the user can also select which colour intensity (this could be an average of all the colours) the threshold should be compared to, and which gun (this could be all guns) should be set to zero as a result. For example the user could select to set the all guns to 0 if the red intensity is greater than 127.

4.4 Creating and initialising filters

Filters are created and displayed in the image sequence filter view. Some filters, once created, are ready to use, as they have no user specified parameters. However the majority of them require filter parameters to be set before they are applied to the image sequence. These parameters are set through the popup menu of the filter, using the *Set parameters* option. These filter parameters can be changed at any time, thus allowing the filter to do a number of jobs. Note,

however, that in the case of the PFCF a change in parameters means a recalculation of the calibration curves – a somewhat time consuming process.

The image sequence filter view highlights any inconsistencies in the filter pipeline associated with incompatible input and output data types. A filter that is not compatible with its preceding filter will have its name and input/output types displayed in red. An attempt to apply such a pipeline will fail due to the incompatibility.

When a filter, which contains one or more image files in its definition, is created, or an previously saved one loaded from disk, a check is done to ensure that the images are consistent with the images stored in the image sequence.

4.5 Summary

This chapter has provided an overview of the filtering process, the types of filters that are available, and how some of them might be used. The addition of new filters is a simple matter and new releases of the software are likely to include new filters.

5 Intensity Field Calculation

5.1 Intensity fields

While the transformed images obtained by filtering an image sequence may be of value in their own right, it is more likely that the data stored within these images has some quantitative value that the user wishes to explore. This exploration is possible through the intensity fields that can be created from the image sequence, or its filtered equivalent.

An **intensity field** is simply a three dimensional field of scalar values deduced from the images within the sequence. The three dimensions of the field correspond to the two physical dimensions of each image (*x* horizontal and *y* vertical) and time.

Each intensity field is defined on a rectangular grid or mesh. The spacing of the mesh in the temporal dimension is determined by the time step between images in the sequence, but the user is at liberty to choose the spacing and size of the mesh in the spatial dimensions. For full detail, the user can choose a mesh so that every pixel corresponds to a node. Normally a somewhat less refined mesh is adequate in determining the essential features of the field, and is less computationally demanding and memory hungry, particularly for long image sequences.

When creating the intensity field from an RGB image sequence the user must specify which component of the colour image is to be used to create the scalar field. It may be one of the colour intensities or an average of all three.

The intensity at each point on the defined grid is calculated from the colour field at the pixel in which the grid point is located. In other words the pixels are assumed to cover the entire image and each grid point will lie inside one pixel.

To allow gradients of the intensity field to be viewed derivatives of the intensity field are also calculated at each of the grid points. These derivatives can be either pixel based used a central difference scheme – thus the intensity at the pixel on either side of the grid point is used – or grid based where the intensity at the grid points on

either side of the grid point of interest are used. On the edges of the image only first order derivative estimates are used.

5.2 Intensity field creation

Intensity fields are created through the *Create intensity fields…* option in the popup menu of the image sequence. Once created, an intensity field object is added to the intensity field list view of the *ImageStream* object and the user can interact directly with it through this view.

5.3 Field calculators

Accessing the information stored in an intensity field is done through a field calculator. A **field calculator** provides the user with a powerful interface through which different projections of this three-dimensional intensity field can be viewed.

At present seven calculators are defined. These are listed in table 5.1.

Symbol	Description
I	the instantaneous intensity field
I’	the fluctuating intensity field. The time average intensity, calculated from the entire time series, is subtracted from I to generate this field.
I’I’	the intensity of the fluctuating intensity.
grad(I)	the gradient of the intensity field
grad(I)|	the magnitude of the gradient of the intensity field
dI/dx	partial derivative of intensity with respect to x
dI/dy	partial derivative of intensity with respect to y

Table 5.1 The list of available intensity field calculators.

The calculators defined for an intensity field object are displayed in the intensity field calculator view by double clicking on the intensity field object. Each field calculator provides a **manager view** that allows the user to generate different projections, or subfields, of the calculated field. For example the user may wish to view the intensity at one particular grid point over time. Or perhaps view the time-averaged intensity in space. In each case the user may wish to only consider a subset of the independent variables. For instance when viewing the time-averaged intensity the user may choose to average over the first 10 seconds only, and may only wish to look at the intensities in the left half of the field. All of these options can be set in the intensity field manager view.

Details of the field calculators and the fields they generate can be found in the **User's Guide**.

5.4 Intensity field transformations

Intensity fields, in their raw form, may not provide the information required by the experimentalist. For example, consider a fluid flow, that is steady in a moving frame of reference, captured by a stationary camera. The experimentalist may wish to view the data in the moving frame. ***ImageStream*** provides a number of intensity field transformations that allow the user to transform their intensity field data in some way. Table 5.2 provides a brief summary of the available transformations. See the **User's Guide** for more detailed information.

Name	Description
Linear	Allows the user to scale and offset the intensity field and the x, y and t coordinates. Useful for converting data to dimensionless form.
Swap x	Reflects the data along a vertical centre line.
Swap y	Reflects the data along a horizontal centre line.
Translate fields using constant velocity	Shifts the 2D intensity fields by a constant velocity so that the spatial domain becomes stretched in the direction of motion.
Translate fields using file displacements	Shifts the 2D intensity fields by displacements stored in a text file. Similar to the transformation above but this allows for non-uniform displacements.
Average fields in time	Performs a partial time average. The user specifies the number of frames over which the time average is computed. The result is a time series with fewer frames and a larger time step – each frame now representing an average.

Table 5.2 The list of intensity field transformations available in *ImageStream*.

5.5 Summary

The production of intensity fields is generally the final step in the image processing activity. Through the field calculator mechanism the user has powerful tools with which to view his/her intensity data.

6 Conclusions

This manual has presented an overview of *ImageStream*, the implementation of a simple image processing system developed in the Department of Civil Engineering at the University of Canterbury.

The following areas have been covered

- an overview of image processing in fluid dynamical research,
- practical issues of lighting and image capture,
- operational and system issues, including computer system requirements, and the user interface model,
- image sequence creation,
- different types of filtering and filter pipelines,
- intensity field calculation.

Nearly all of the specifics relating to the operation of *ImageStream* are included in the **User's Guide**. This includes detailed information on parameter definitions and selection, as well as descriptions of *ImageStream* objects and their associated views.

7 References

Blackett S A 1994, "Particle tracking velocimetry", *ME Thesis*, University of Auckland, Auckland, New Zealand, pp 86.

Brown C 2002, "Scale modelling of waste stabilisation ponds", *Third professional project report*, Dept of Civil Engineering, University of Canterbury, pp 78.

Cowen E A and Monismith S G, 1997, "A hybrid digital particle tracking velocimetry technique", *Experiments in Fluids*, **22**, pp 199-211.

Davidson M J and Pun K L 1999 "Weakly advected jets in cross-flow", *Journal of Hydraulic Engineering*, **125**, 1, pp 47-58

Holford J M and Dalziel S B, 1996 "Measurements of layer depth during baroclinic instability in a two-layer flow" *Applied Scientific Research*, **56**, pp191-207.

Ivey G N, and Nokes R I, 1989 "Vertical mixing due to the breaking of critical internal waves on sloping boundaries" *Journal Fluid Mechanics*, **204**, pp 479-500.

Nokes R I, 2002, "Interface proxy classes in object-oriented user interface design – an extension to the model-view architecture", *New Zealand Journal of Applied Computing and Information Technology*, **6**, 1, pp 37-44.

Nokes R I, 2007 "FluidStream 7.00 – System Theory and Design", Dept of Civil Engineering, University of Canterbury, pp 77.

Ojanen H, 1999, "Automatic correction of lens distortion by using digital image processing", *www.math.rutgers.edu/ojanen/undistort/undistort.pdf*

Poynton C A 1996, "A technical introduction to digital video", *John Wiley and Sons*, New York, pp 320.

Raffel M, Willert C and Kompenhans J, 1998, "Particle Image Velocimetry", *Springer-Verlag*, Berlin, Germany, pp 253.

Smits A J and Lim T T, 2000, “Flow visualisation”, *Imperial College Press*, London, UK, pp 396.

Stamp A P, Hughes G O, Nokes R I and Griffiths R W, 1998 “The coupling of waves and convection”, *Journal of Fluid Mechanics*, **372**, pp 231-271.

Van Dyke M 1982 “An album of fluid motion”, *Parabolic Press*, Stanford California, pp 176.

Appendix A
Installation and Execution

The *ImageStream* application is deployed as a ZIP file, either on CD or downloadable from the web (contact the designer for information on this). Different ZIP files are provided for Macintosh and Windows systems and the installation process is slightly different on these systems.

A.1 Installation under Windows

The ZIP file required for installation on a Windows computer is named

IS600_Win.zip

To install *ImageStream* on a *Windows* computer perform the following steps:

1. Create a directory in the root directory of the computer C: drive called *StreamsLibrary* (if it does not already exist). Within this directory create a second directory called *ImageStream6.00* and extract the contents of *IS600_Win.zip* into this directory. The *ImageStream6.00* directory should contain the following 4 files:

- ImageStream.jar – the java archive file that contains the application
- ImageStream6.00 – the shortcut for running the application
- java.exe – the java interpreter called by the shortcut to run *ImageStream*.
- ImageStream32x32.ico – the icon used for the *ImageStream* shortcut.

and a directory containing the PDF versions of the software manuals.

The shortcut for running Image*Stream* can be copied to any location you wish.

2. Edit the target within the shortcut to allow for your system configuration (if required). The target is accessed through the shortcut's popup menu. Select Properties and go to the Shortcut tab. You may need to change the file attributes on the General tab so that Read-only is switched off. As delivered the target is

C:\StreamsLibrary\ImageStream6.00\java.exe –Xms800m –Xmx800m –Xss32m –classpath
C:\StreamsLibrary\ImageStream6.00\ImageStream.jar ImageStream.Application.ImageStream

Provided you leave the .jar and .exe files in the *C:\StreamsLibrary\ImageStream6.00* directory none of the paths need to be changed. If you wish to move them to another directory then the new path should precede the filenames in the target. For example, if you wish to put these files in a directory

C:\IP

Then the target would become

C:\IP\java.exe –Xms800m –Xmx800m –Xss32m –classpath
C:\IP\ImageStream.jar ImageStream.Application.ImageStream

The three parameters following the java.exe set the memory size for the application. The first is the initial heap space, the second the maximum heap space and the third the desired stack space. The figures 800m, 800m and 32m refer to *800Mb*, *800Mb* and *32Mb* respectively. Depending on the RAM installed on your system you may need to change these settings. Experiment with these so that the application runs with the maximum settings. A minimum of *32Mb* for the stack space, with the rest of the memory allocated to the heap, should be a good initial setting.

It is also possible that you already have a later version of java installed on your computer. If this is the case then there is no reason to remove this version and install version 1.4 if you replace the file java.exe in the *ImageStream7.00* directory with the same file provided with the installed version of java. This file can be located in a subdirectory of the Java directory in C:\Program Files. A search of the C:\ drive is probably the easiest way to locate this file.

3. *ImageStream* is executed by double-clicking the *ImageStream6.00* shortcut icon.

A.2 Installation on a Macintosh

The ZIP file required for installation on a Macintosh computer is named

IS600_Mac.zip

To install *ImageStream* on a *Macintosh* computer perform the following steps:

1. Create a folder in the root directory of the hard drive called *StreamsLibrary* (if it does not already exist). Within this folder create a second folder called *ImageStream6.00* and extract the contents of *IS600_Mac.zip* into this folder. The *ImageStream6.00* folder should contain the following 2 files:

- ImageStream.jar – the java archive file that contains the application
- ImageStream6.00.app – the Apple script for running the application

and a directory containing the PDF versions of the software manuals.

The Apple script file for running *ImageStream* can be copied to any location you wish.

2. Edit the command within the Apple script file (*ImageStream6.00.app*) to allow for your system configuration (if required). Use the Apple Script editor to do this. As delivered the script file should contain a single command

```
do shell script java -jar –Xms800m –Xmx800m
–Xss32m //StreamsLibrary/ImageStream6.00/ImageStream.jar
ImageStream.Application.ImageStream
```

Provided you leave the .jar file in the *//StreamsLibrary/ImageStream6.00* folder the path need not be changed. If you wish to move it to another folder then the new path should precede the jar filename in the script command. For example, if you wish to move the .jar file to a folder called

//IP

then the command would become

```
do shell script java -jar –Xms800m –Xmx800m
–Xss32m //IP/ImageStream.jar ImageStream.Application.ImageStream
```

The three parameters following the -jar setting specify the memory size for the application. The first is the initial heap space, the second the maximum heap space and the third the desired stack space. The figures 800m, 800m and 32m refer to *800Mb*, *800Mb* and *32Mb* respectively. Depending on the RAM installed on your system you may need to change these settings. Experiment with these so that the application runs with the maximum settings. A minimum of *32Mb* for the stack space, with the rest of the memory allocated to the heap, should be a good initial setting.

3. *ImageStream* is executed by double-clicking the *ImageStream6.00.app* icon.

Appendix B Registration

ImageStream must be registered before it can be run. A registration key can be requested from Dr Roger Nokes.

When *ImageStream* is run for the first time a registration dialog box is displayed that prompts the user to electronically agree to the conditions of software use and to enter a 20 character registration key. This key is checked by the software, and if valid, the application is launched. Subsequent launches of the software do not require the registration key to be reentered.

Part of the registration key is the registration expiry date. If the application is run after this date the user will be prompted to reregister the software. The expiry date can be viewed in the application parameters dialog box under the *Application* menu of the main window. The date on which the software was registered is also displayed in this dialog box.