



Streams

Version 3.03

System Theory and Design

by Dr. Roger Nokes

Streams

Version 3.03

System Theory and Design



© Dr Roger Nokes
February 2021

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

Design and layout: Melody Callahan

I	Introduction to Flow Visualisation	1
1.1	General principles	1
1.2	Particle tracking velocimetry	5
1.2.1	Image capture	5
1.2.2	Image processing – particle identification	10
1.2.3	Particle tracking	10
1.2.4	Velocity field calculation	11
1.2.5	Alternative applications	12
1.3	Scalar field techniques	13
1.3.1	Lighting	13
1.3.2	Dyes	13
1.3.3	Image processing	14
1.4	Summary	15
2	Overview of the Streams System	17
2.1	Introduction	17
2.2	User interface	17
2.2.1	Application view	19
2.2.2	Object list view	19
2.3	Entities and processes	20
2.3.1	Process pipelines	21
2.3.2	Pipeline-based processes	23
2.4	Core entities	24
2.4.1	Image sequences	24
2.4.2	Particle records	24
2.4.3	Particle 2D record bundles	24
2.4.4	Eulerian field time series	25
2.4.5	Lagrangian path fields	26
2.5	Fields	26
2.6	Tablets	27
2.7	Formulae	27
2.8	Manuals	28
2.9	Summary	28

3	Image Sequences.....	29
3.1	Overview of image sequences	29
3.2	Image data types	30
3.3	Image sequence creation	30
3.4	Opening and saving image sequence files	32
3.5	Processes.....	32
3.5.1	Image filter pipeline.....	32
3.5.2	Particle identification.....	32
3.5.3	Create intensity field.....	33
3.5.4	Transforms	33
3.5.5	Other processes	35
3.6	Views.....	35
3.6.1	Image view.....	36
3.6.2	Details view	36
3.6.3	Process view.....	37
3.7	Summary.....	38
4	Image Filter Pipelines.....	39
4.1	Overview.....	39
4.2	Filter inputs and outputs	40
4.3	Image filters	40
4.3.1	Absorption colour filter.....	40
4.3.2	Absorption image filter	42
4.3.3	Amplify filter	42
4.3.4	Barrel pincushion distortion filter.....	42
4.3.5	Bayer colour filter.....	43
4.3.6	Convert type filter	44
4.3.7	Convolution filter	44
4.3.8	Discrimination 1 filter	45
4.3.9	Discrimination 2 filter	46
4.3.10	Eliminate objects filter.....	46
4.3.11	Extract filter	47
4.3.12	FFT filter.....	47
4.3.13	Identity filter	48
4.3.14	Invert filter	48

4.3.15	Mathematical formula filter	48
4.3.16	Overwrite regions filter	49
4.3.17	Parallax factor filter.....	49
4.3.18	Polynomial field calibration filter	49
4.3.19	Remove filter.....	50
4.3.20	Remove background filter	51
4.3.21	Scale by maximum in regions filter	51
4.3.22	Spatial average filter	52
4.3.23	Subtract colour filter.....	52
4.3.24	Subtract image filter.....	52
4.3.25	Subtract image sequence filter.....	53
4.3.26	Threshold filter.....	53
4.3.27	White balance image filter.....	54
4.4	Creating and initialising filters.....	54
4.5	Summary.....	55
5	Particle Identification.....	57
5.1	Introduction.....	57
5.2	Particle identifiers	59
5.2.1	Single threshold particle identifier	59
5.2.2	Dual threshold particle identifier.....	60
5.2.3	Dual threshold Gaussian particle identifier	61
5.3	Summary.....	62
6	Particle Records	63
6.1	Overview.....	63
6.2	Particles and frames	63
6.3	Particle record creation from models.....	64
6.3.1	Cellular model (2D).....	66
6.3.2	Irregular cellular model (2D).....	66
6.3.3	Multi-cellular model (2D).....	66
6.3.4	Turbulent shear model (2D)	67
6.3.5	Sediment motion model (2D).....	68
6.3.6	Custom formula model (2D, 3D surface, & 3D)	68
6.4	Particle record creation from text file	68

6.5	Segmentation	68
6.6	Processes	69
6.6.1	PTV analysis pipeline (2D & 3D)	69
6.6.2	Create velocity field (2D & 3D)	70
6.6.3	Create density field (2D)	70
6.6.4	Create shape field (2D)	70
6.6.5	Create displacement field (2D, 3D surface, & 3D) ..	71
6.6.6	Create Lagrangian path field (2D & 3D)	71
6.6.7	Transform pipeline (2D & 3D)	71
6.6.8	Particle filter pipeline (2D)	74
6.6.9	Other processes	74
6.7	Views	74
6.7.1	Particle view (2D & 3D)	74
6.7.2	Details view (2D & 3D)	76
6.7.3	Process view (2D & 3D)	76
6.7.4	Match summary view (2D & 3D)	76
6.7.5	Diagnostic view (2D & 3D)	76
6.7.6	Comparison view (2D & 3D)	77
6.7.7	Ancillary views	77
6.8	Summary	77
7	PTV Analysis	79
7.1	Overview	79
7.2	Optimisation process	79
7.2.1	The auction algorithm	81
7.3	PTV analysis	82
7.4	Costings	86
7.4.1	Costing Types	88
7.5	Process types	89
7.5.1	Local optimisation	89
7.5.2	Residual optimisation	90
7.5.3	Clean up process	90
7.6	Summary	90
8	Costings	91

8.1	Overview	91
8.2	State-based costings	91
8.2.1	Adjacency costing (2D & 3D)	91
8.2.2	Centre of mass costing (2D & 3D)	92
8.2.3	Colour costing (2D & 3D)	93
8.2.4	Correlation costing (2D)	93
8.2.5	Distance costing (2D & 3D)	94
8.2.6	Intensity costing (2D & 3D)	95
8.2.7	Pseudo-correlation costing (2D & 3D)	96
8.2.8	Shape costing (2D)	97
8.3	Matching-based costings	97
8.3.1	Direction velocity costing (2D & 3D)	98
8.3.2	Least squares velocity costing (2D & 3D)	98
8.3.3	Local velocity costing (2D & 3D)	99
8.3.4	Magnitude velocity costing (2D & 3D)	100
8.3.5	Path length costing (2D & 3D)	100
8.3.6	Polynomial velocity costing (2D & 3D)	101
8.3.7	Recent acceleration costing (2D & 3D)	102
8.3.8	Recent velocity costing (2D & 3D)	102
8.3.9	Space average acceleration costing (2D & 3D)	102
8.3.10	Time average velocity costing (2D & 3D)	103
8.4	Guidelines	103
8.4.1	State based costings	103
8.4.2	Matching based costings	104
8.4.3	Summary	105
8.5	Summary	106
9	Construction of Particle 3D Records	107
9.1	Overview	107
9.2	Particle 2D record bundles	107
9.3	Particle construction	109
9.4	Bundle matching analysis	110
9.5	Bundle matching costings	113
9.5.1	Colour costing	113
9.5.2	Distance costing	113
9.5.3	Intensity costing	114

9.5.4	Mapping error costing.....	114
9.5.5	Regions costing.....	114
9.5.6	Size costing.....	114
9.6	Predict and verify	114
9.7	Summary.....	117
10	Intensity Field Time Series.....	119
10.1	Overview.....	119
10.2	Intensity field creation.....	119
10.3	Calculators.....	120
10.3.1	Custom calculators.....	121
10.4	Processes.....	122
10.4.1	Transform pipelines.....	122
10.4.2	Save intensity field.....	124
10.5	Views.....	124
10.5.1	Calculator view.....	124
10.5.2	Details view.....	124
10.5.3	Process view.....	124
10.6	Summary.....	124
11	Velocity Field Time Series	125
11.1	Overview.....	125
11.2	Velocity field creation	125
11.3	Calculators.....	131
11.3.1	Custom calculators.....	131
11.4	Processes.....	132
11.4.1	Transform pipelines.....	132
11.4.2	Save velocity field.....	132
11.5	Views.....	133
11.5.1	Calculator view.....	133
11.5.2	Details view.....	133
11.5.3	Process view.....	133
11.6	Creation of velocity 3D field time series from model	133

11.7 Summary.....	134
12 Velocity-intensity Field Time Series.....	135
12.1 Overview.....	135
12.2 Velocity-intensity field creation	135
12.3 Calculators.....	136
12.3.1 Custom calculators.....	136
12.4 Processes.....	136
12.4.1 Transform pipelines	136
12.4.2 Save velocity-intensity field.....	137
12.5 Views.....	137
12.5.1 Calculator view.....	137
12.5.2 Details view	137
12.5.3 Process view.....	137
12.6 Summary.....	137
13 Density Field Time Series	139
13.1 Overview.....	139
13.2 Density field creation	139
13.3 Calculators.....	140
13.4 Processes.....	140
13.4.1 Transform pipelines	140
13.4.2 Save density field.....	140
13.5 Views.....	140
13.5.1 Calculator view.....	141
13.5.2 Details view	141
13.5.3 Process view.....	141
13.6 Summary.....	141
14 Shape Field Time Series.....	143
14.1 Overview.....	143
14.2 Shape field creation.....	143
14.3 Calculators.....	143

- 14.4 Processes 144
 - 14.4.1 Transform pipelines 144
 - 14.4.2 Save shape field..... 144
- 14.5 Views 144
 - 14.5.1 Calculator view..... 144
 - 14.5.2 Details view 144
 - 14.5.3 Process view..... 145
- 14.6 Summary..... 145
- 15 Displacement Field Time Series 147**
 - 15.1 Overview..... 147
 - 15.2 Displacement field creation 149
 - 15.3 Calculators..... 151
 - 15.3.1 Custom calculators..... 152
 - 15.4 Processes..... 152
 - 15.4.1 Transform pipelines 152
 - 15.4.2 Save displacement field..... 153
 - 15.5 Views 153
 - 15.5.1 Calculator view..... 153
 - 15.5.2 Details view 153
 - 15.5.3 Process view..... 153
 - 15.6 Summary..... 153
- 16 Lagrangian path fields..... 155**
 - 16.1 Overview..... 155
 - 16.2 Lagrangian path field creation 156
 - 16.3 Calculators..... 158
 - 16.3.1 Custom calculators..... 158
 - 16.3.2 Two path calculators 159
 - 16.4 Processes..... 159
 - 16.4.1 Join Lagrangian paths (2D & 3D) 159
 - 16.4.2 Transform pipeline (2D) 160
 - 16.4.3 Save path field (2D & 3D) 160
 - 16.5 Views 160

16.5.1	Calculator view.....	161
16.5.2	Details view.....	161
16.5.3	Path view.....	161
16.5.4	Path list view.....	162
16.5.5	Process view.....	162
16.5.6	Selected path list view.....	162
16.6	Summary.....	162
17	Technical Issues.....	163
17.1	The Java environment	163
17.1.1	Performance Issues.....	163
17.1.2	Memory Management	164
17.2	Application basics.....	166
17.2.1	Configuration file	166
17.3	Installation, execution, and registration	167
18	Summary	169
19	References	171

I Introduction to Flow Visualisation

I.1 General principles

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. The recent *Handbook of Environmental Fluid Dynamics Volume 2*, Fernando (2013), also provides some broad perspectives on the use of these optical techniques.

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) that, 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 that 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. In the last 30 or so years two new experimental techniques have been developed for the quantitative measurement of two, and sometimes three, dimensional velocity fields using digital video technology and the computational power of personal computers. These techniques are generally known as **particle image velocimetry** (PIV) and **particle tracking velocimetry** (PTV) (see Smits and Lim (2000) and Raffel, Willet and Kompenhans (1998)). The scope of these techniques has broadened significantly in recent years due to advances in computer and digital video technology and their practical application has increased while their associated costs have decreased.

Both techniques are based on the principle of capturing video images of a specially illuminated particle seeded fluid flow, and from that video record extracting quantitative information about the flow field. The processing of the video frames is automated, using specially designed algorithms, and executed on a computer. By far the most popular of these techniques has been PIV, although there are good arguments why PTV is equally worthy of attention. The book by Raffel et al (1998) provides an excellent introduction to all aspects of PIV. PTV is only mentioned in passing, although Cowen and Monismith (1997) suggest that PTV offers significant advantages in accuracy over traditional PIV methods. They describe a hybrid system based on both PIV and PTV techniques.

In simple terms PIV is based on the cross-correlation of the intensity fields in two consecutive frames, with fluid velocities obtained from peaks in the cross-correlation field. For this process to work effectively the particle seeding must be sufficiently dense for the

cross-correlation to be meaningful. Frames taken from the video record can be processed directly to obtain the desired flow information.

On the other hand PTV is somewhat more complex. The video frames must be processed in order to identify the particles within each frame. Particles are then tracked from frame to frame in order to produce particle-centred velocity estimates. These velocities may be interpolated onto some regular grid to obtain Eulerian velocity information, although one of the advantages of PTV, the ability to generate Lagrangian flow statistics, does not require this final step. Particle density is less of an issue with PTV than PIV. In fact, sparsely seeded flows are often easier to analyse as the tracking of particles from frame to frame is relatively straightforward. However more densely seeded flows can also be processed using PTV. In this aspect, at least, PTV is more flexible than PIV.

Particle identification is a key part of a particle tracking velocimetry system. However, particle identification is a rather more general tool that can be used in different fluid mechanics applications and, in fact, in other totally unrelated areas. For example, the Environmental Fluid Mechanics Group at the University of Canterbury has explored the dynamics of oil plumes in water. These plumes break down into fields of oil droplets which, under certain flow conditions, exhibit clear size differentiation. Particle identification is a very useful tool for characterizing these oil droplet fields. The motion of sediments on the bottom of an open channel can be explored in a similar way by first identifying the sediment particles and then associating their movement with their physical characteristics such as shape and size (see for example Campagnol, Radice, Nokes, Bulankina, Lescova and Ballio 2013 and Heays, Friedrich, Melville and Nokes 2014).

A recent development in our laboratory has been the employment of PTV in the area of structural mechanics. The technique is well suited to the measurement of displacement and hence strain fields in structural specimens under loading. In this case the particles are typically dots applied to the surface of the specimen and tracked during the loading process. In many ways this is a simpler application of PTV than in fluid mechanics as particles move only short distances. But in other ways this application offers other challenges such as the destruction of particles as the specimen begins to fail and the fact that only surface displacement fields are

measured. Similar techniques have existed in the structural field for some years.

All discussion to this point has focused on techniques that aim to measure two-dimensional velocity or displacement fields. In both the fluid and structural mechanics domains the extension of PTV to three-dimensions offers interesting advantages over its two-dimensional counterpart. While these techniques have been discussed in the literature for more than 20 years (see for example Maas, Gruen and Papantoniou 1993 and Malik, Dracos and Papantoniou 1993) they are still not commonly encountered due to their significant additional complexity. In fluid mechanics three-dimensional techniques enable secondary currents to be identified and full turbulence statistics to be computed (for example the turbulent kinetic energy). In structural mechanics it enables out of plane motions such as buckling to be quantified. Three-dimensional PTV presents significant technical challenges over and above those encountered with two-dimensional PTV and the experimentalist needs to judge whether the additional information garnered from this technique is worth the extra complexity.

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, Kikkert, Davidson and Nokes 2010 and Cenedese, Nokes and Hyatt 2016). 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.

Streams is an image processing application developed in the Department of Civil and Natural Resources Engineering at the University of Canterbury that aims to provide a broad range of tools for analyzing fluid flow visualization experiments, including particle tracking velocimetry, laser induced fluorescence and light attenuation. In the remainder of this chapter more detail about the techniques that have been briefly introduced above will be provided.

1.2 Particle tracking velocimetry

The process of obtaining velocity fields for a fluid flow using the PTV technique can be broken down into four steps. These are

- Image capture
- Image processing – particle identification
- Particle tracking
- Velocity field generation.

Each of these steps is discussed in more detail in sections 1.2.1 – 1.2.4, both in a generic way, and with a particular focus on the *Streams* system.

1.2.1 Image capture

Image capture involves a number of issues. These are

- Lighting
- Particle selection
- Frame capture

1.2.1.1 Lighting

For two-dimensional PTV work a sheet of high intensity light is needed to illuminate the particles in the flow of interest. A range of light sources can be used, and the selection will depend somewhat on the particular requirements of the experiment.

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 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 source is achieving high enough light intensities to allow the video capture equipment to “freeze” the motion in the flow.

Laser light sheets are commonly used in PIV and PTV work. High power lasers, typically 1-2W or more, are used. At the time of writing such laser systems are becoming more affordable although the quality of some of the cheaper systems is questionable. 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 a few millimetres, 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. However lasers also come with significant health and safety requirements and for this reason some laboratories avoid using them altogether.

In recent years the advent of high intensity LEDs has provided the experimentalist with another alternative light source. LEDs generate little heat and therefore are safer to use than traditional white light sources that use high wattage bulbs enclosed in light sealed enclosures. They often come with lens or mirrors that already provide some degree of directional constraint on the light and their intensities are now comparable with high wattage halogen bulbs. It is likely that LEDs will become the preferred option for simple, non-laser based, systems in the future.

The *Streams* system can be employed to analyse 2D data obtained from the use of any type of light sheet. However, particle motions perpendicular to the sheet may cause particles to remain in the sheet only for very short times, and this is obviously undesirable from the software’s perspective, as it is attempting to track particles from

frame to frame. Therefore out of plane motions may be a consideration in light sheet selection.

The light sources for three-dimensional PTV are actually simpler than those for two-dimensional PTV. In this case a whole region of the flow needs to be illuminated and therefore the constraint of producing a narrow light sheet is avoided altogether. A probably more important requirement for 3D PTV is ensuring that experimental surfaces, that may be illuminated by the domain light source, possess minimal reflectance so that the particles are easily seen.

1.2.1.2 Particle selection

In water based flows pliolite VT resin, a white material used in the rubber manufacturing process, has become the standard material from which to produce the seed particles. The resin has a density roughly 3% greater than water. It can be ground into particles of diameters ranging from a few microns up to hundreds of microns. Very small particles have fall velocities that can be ignored in most laboratory flows. In some flows it is possible to add salt to the working fluid in order to ensure that the particles are neutrally buoyant, somewhat removing the restriction on the size of particles used.

Particle size selection is determined by a number of factors. First and foremost the particles must act as tracers that accurately replicate the velocity of the fluid particles surrounding them. Raffel et al (1998) provide a crude estimate of the response time of such particles to fluid accelerations. This is given by

$$\tau_s = \frac{d_p^2 \rho_p}{18\mu} \quad (1.1)$$

where d_p is the diameter of the particle, ρ_p is their density, μ is the viscosity of the fluid and τ_s is the relaxation time of the particle velocity in response to fluid accelerations. Clearly the shortest timescale that can be accurately measured using PTV will depend on the particle size.

Particle size also affects the intensity of the reflected light, as seen by the image capture system. The intensity of the light increases in importance as the particle size decreases, and in most experiments a

compromise between particle size, light intensity and exposure time on the camera system is required.

Introducing such fine particles into a fluid flow can be difficult due to surface tension effects. Normally the particles are mixed into a small amount of the working fluid together with a surfactant to reduce the surface tension, before being introduced to the experimental system.

The density of particles in a flow again depends on the flow being investigated and the type of information required. Issues of seeding density will arise again in later sections.

Another possible application of 2D PTV is in tracking surface drogues. Materials other than pliolite resin are commonly used. In particular drogues are generally larger objects that may have a submerged “vane”. In the Fluids Laboratory at the University of Canterbury golf tees have been successfully used as drogues as have nylon particles of a few millimetres in diameter (see Nikora, Nokes, Veale, Davidson and Jirka 2007). However, surface tension can affect the motion of the drogues when they come close to a solid boundary, or one another, and the addition of a surfactant may be useful.

Another variation in particle selection is some natural material that is of importance in the system under consideration. Perhaps the most obvious is sediment moving along the bed of channel flow, but another example from our laboratory has been kelp propagules (eggs) whose dynamic properties were to be investigated.

1.2.1.3 Frame 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 high resolution, digital video mode, as well as a slow automated sequential frame capture capability.

Digital video cameras produced for the mass market are perfectly good tools for image capture in the laboratory provided the lighting and resolution demands are not too great. The modern high definition (HD) digital video supports a range of pixel resolutions

with standard HD having a resolution of 1920 pixels per video line with 1080 lines per frame. Some cameras on the consumer market only capture video in interlaced mode (i.e. two half frames, 1/50th of a second apart, are combined to give a complete frame every 1/25th of a second). Progressive scan, or non-interlaced recording, instead captures the whole frame at once. For PTV work progressive scan is highly desirable, particularly if the flow speeds are high.

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.

Higher resolution digital video cameras are available for more specialised scientific work. These cameras are colour or grey-scale (8, 10 or 12 bit) and have a wide range of resolutions. The software for controlling these cameras is specialized and dedicated frame-grabber cards may be required. However, with some new technologies, such as the GigE standard whereby the camera is connected to the computer through a gigabit per second Ethernet connection, the need for specialized cards is eliminated.

One of the chief concerns in choosing a camera for PTV work is the camera's ability to work with low light levels. In general, we have found that digital video cameras provide 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 aperture are crucial for good PTV work.

A range of digital image formats is currently supported by cameras and associated software. For example digital video formats include AVI, MOV, and MPEG. Digital still images are normally stored in TIF, JPEG, GIF, PNG or BMP formats. The *Streams* system provides support for sequences of JPEG, TIF, PNG, BMP and GIF files. It does not directly support any digital video format. Some video processing software may be required to convert digital video into a sequence of image files, although most scientific cameras store the captured images as individual image files as a matter of course.

I.2.2 Image processing – particle identification

For two-dimensional PTV once a sequence of video frames has been captured these images must be processed in order to identify the particles within each frame. For each particle we wish to obtain its location to sub-pixel accuracy as well as, perhaps, its intensity, size and shape. This analysis can be quite challenging due to issues of variable intensity across and along the light sheet, varying particle size (and hence brightness) and a range of other experimental issues such as camera performance.

Often some type of preprocessing of the images is required before the particle identification algorithms can be successfully employed.

Particle identification in three-dimensional PTV is substantially more challenging than that for two-dimensional PTV. In this case a number of cameras, at different orientations to the flow, are used to record the motion of the particles. Thus a set of two-dimensional recordings are obtained that can be analysed in the same way as two-dimensional PTV to obtain two-dimensional particle locations. These two-dimensional particle coordinates must then be combined to produce the three-dimensional coordinates of the recorded particles. This involves two steps. Firstly, each camera must be calibrated so that its location, orientation etc. is accurately known, and secondly the two-dimensional particle records from the various cameras must be integrated together to produce the full three-dimensional field of particles. Neither of these two processes is straightforward.

I.2.3 Particle tracking

The core of the PTV analysis process is the tracking of particles from frame to frame, and it is this aspect that is the most challenging. There is a wide range of techniques that can be used, but each essentially reduces to a decision-making algorithm that must select the “best match” between particles in two sequential frames. An example of a best match algorithm is simply to select the particle in frame 2 that is closest to the particle in frame 1.

It should be noted that a match is not always possible for a number of reasons. These include

- the particle in frame 1 left the domain before frame 2 was captured
- the particle in frame 1 is obscured by another particle in frame 2
- the particle identification process failed to identify the particle in frame 2.

Streams provides a relatively sophisticated analysis environment for matching particles between frames that is effectively the same for both two-dimensional and three-dimensional PTV. A range of criteria, or algorithms, can be used to guide the optimisation process, and these algorithms can be fine-tuned by the user for their particular needs. Both sparsely and densely seeded flows can be analysed.

It should be noted that the particle matching process is imperfect. Particles can be incorrectly matched, or not matched at all. It is important that the impact of such imperfections on the predicted velocity fields is understood.

1.2.4 Velocity field calculation

The ultimate goal of a PTV system is to generate velocity fields for the two-dimensional (or three dimensional if 3D PTV is being used) flow domain being observed. The velocity can be estimated at each particle location (provided that particle has a match in the previous and/or subsequent frame) by analysing the displacement of the particle between frames that are a known time step apart.

Unlike PIV, where velocities can be calculated at any point in the flow, velocities in PTV are particle-centred. This means that the velocities are known at a set of random locations in each frame. In order to deduce Eulerian information from these velocities, for example the development of the velocity at a particular position in the flow with time, velocities must be interpolated onto a regular grid. Cowen and Monismith (1997) suggest an additional technique of “binning” the velocity measurements from all frames in order to improve this interpolation process. On the other hand PTV systems readily produce Lagrangian flow data.

The accuracy of the velocities generated by a PTV system will be limited by the following factors:

- inaccurate mapping from pixel space to physical space,
- optical distortion of the video images,
- incorrect particle identification,

- incorrect particle location determination,
- incorrect particle matches,
- missed particle matches,
- insufficient particle density to resolve the velocity field,
- inaccurate velocity estimate from matching due to streamline curvature,
- inaccurate interpolation scheme,
- inaccurate extrapolation scheme at the edge of a frame,
- particles not faithfully following the flow, and
- camera resolution.

Many of these sources of error can be minimised through careful experimental design and analysis.

Once the Eulerian velocity fields have been generated, derived information, such as turbulence statistics, vorticity etc., can be calculated. However, it is important to recognise what information can and cannot be deduced accurately from the interpolated velocity fields. Turbulent properties of the flow in particular must be sought with care. The duration of sampling and the sampling frequency must be sufficient for meaningful turbulent averages to be obtained. The seeding density and flow resolution will determine the range of scales that the PTV system can resolve. In addition, the sources of error discussed above can lead to “pseudo-turbulence” that may be hard to identify and eliminate. If detailed turbulence data is required it is recommended that an LDA or ADV is used in conjunction with the PTV system in order to check the tracking system’s outputs.

1.2.5 Alternative applications

While the discussion in this book, and its associated manuals, is primarily focused on the application of PTV to fluid flows the technique is, in fact, quite general. There are many fields in which the identification of distinct objects and the tracking of their motion are helpful in understanding the dynamics of the system under observation. Examples include tracking vehicles in a road network or the biomechanical actions of the human body in sporting activities. We have already discussed the application to structural mechanics and the measurement of displacement and strain fields.

I.3 Scalar field techniques

Many of the issues already discussed in connection with particle tracking velocimetry are relevant to optical techniques for measuring scalar fields, such as light attenuation or laser-induced fluorescence. Here we will provide extra information that is important in these applications.

I.3.1 Lighting

The light sources required for scalar field techniques may differ from those required for PTV. Laser light sheets are the standard light source for laser-induced fluorescence although the use of white light sources has also been reported, and LEDs with a narrow emission band can be found that will work with a particular fluorescent dye. Light attenuation (LA) on the other hand commonly requires back lighting of the fluid flow. Numerous options are available. In our laboratory back lighting has been provided by banks of fluorescent bulbs mounted on flat racks, or the use of ranks of LED lights. These systems provide strong light sources and through the use of thin (3mm) sheets of semi-opaque acrylic sheeting the variability in light intensity across the light plane can be reduced. This configuration appears to be superior to other set-ups including slide projectors or halogen spotlights.

A more recent technique involved the use of LCD TV screens. While more limited in physical dimension than the fluorescent bulb racks the LCD screen has the significant advantage that the light intensity across the screen can be set in software when the screen is attached to a computer. This can increase light intensity uniformity substantially although overall light intensity levels are typically much lower than those produced by the fluorescent bulbs. Provided sensitive video cameras are used this may not be a major problem.

I.3.2 Dyes

A large range of 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.

1.3.3 Image processing

In scalar field studies the captured images provide quite different information to those designed for PTV. Now the intensity at each pixel in the image provides field information to the experimentalist. The image processing that follows extracts the density or concentration field from the light intensity recorded in the images.

It is worth spending a moment summarising some basic information about video image data. A number of different types of information can be captured by digital video cameras. The most common, found in consumer cameras, is colour data stored in RGB format. These RGB images – red, green and blue – are represented by three 8bit integers. Thus each colour has a maximum intensity (i.e. the image is saturated with the colour) of 255, and a minimum intensity (i.e. 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. It is important to understand the Bayer colour coding used in many cameras today. In these cameras only a single CCD or CMOS sensor is used to capture all three colour signals. This is done using an optical filter placed over the sensor so that each pixel only records one colour. The intensities of the other two colours are then interpolated at the pixel. This system is discussed further in chapter 4 of this manual.

Some more expensive cameras actually use 3 sensors to record images – one for each of the colour guns in the RGB format. The resulting images require no interpolation and generally are of higher quality.

Scientific video cameras are often monochromatic, but with a greater light sensitivity range. For example there are 10bit and 12bit gray-scale cameras. Here the intensity of light at each pixel is recorded as a single number with 10 or 12 bit resolution. Therefore the maximum intensity values that can be recorded are 1023 and 4095 respectively, and these ranges are clearly larger than the 8 bit intensities provided by RGB cameras. These monochromatic cameras are very well suited to laser induced fluorescence experiments.

I.4 Summary

This chapter has aimed to provide the reader with an overview of flow visualization techniques, and in particular the operation of a typical PTV system. This background will be particularly useful to the reader as they learn about the features and tools provided within the *Streams* system. A detailed guide on the design and analysis of particle tracking systems, Nokes (2021), is soon to be published and the reader seeking a more thorough introduction to such systems is advised to access this book once it is available.

2 Overview of the Streams System

2.1 Introduction

A general discussion of flow visualization techniques has been provided in the previous chapter. The remainder of this manual will focus on the *Streams* system, a flow visualization and analysis tool developed in the Department of Civil and Natural Resources Engineering at the University of Canterbury. Three supplementary manuals, **Streams 3.03 – User’s Guide: Core Objects**, **Streams 3.03 – User’s Guide: Field Objects** and **Streams 3.03 – User’s Guide: Tablets**, provide detailed descriptions of the application interface and all elements in the *Streams* environment.

In this chapter an overview of *Streams* 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

Streams 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 and so on.

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

Streams 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, an **image sequence** represents a continuous sequence of images captured by some imaging device. Image sequences are at the core of the *Streams* system and are discussed in more detail in chapter 3. A **particle 2D record** is another type of object. It corresponds to an idealization of a sequence of frames containing particles that have

been identified in an image sequence. Ultimately a particle 2D record will be analysed by the particle tracking subsystem in order to generate two-dimensional velocity fields.

Objects are usually displayed for the user in a list box, where the object's name and its associated icon are shown. For some types of objects *Streams* supports drag and drop, typically between list boxes. For example, the user might create a process to operate on a particular entity. That process would be displayed in a list box in the entity's process view. The user can drag that process to the process view of another entity and this will cause a copy of that process to be added to that entity. The entities that support drag and drop are described in the three associated *User's Guide* manuals.

Each object in the *Streams* 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 saving the object to disk.

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 an image sequence has three principal 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: Core Objects* manual.

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 generally updates its information automatically. When an object is closed – removed from the *Streams* system – all of its views are closed with it.

Each view may have its own menu bar that provides the user with settings associated with that view. For instance, the current drawing tool can be changed 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 *Streams* system.

The *Streams* system itself is an object with two primary, and ten subsidiary views. The two primary views are displayed on opening the application and 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 the menu bar associated with the *Streams* application. There are twelve menus, *Application*, *View*, *Image*, *Particle*, *Eulerian*, *Lagrangian*, *Process*, *Tablet*, *Field*, *Manual*, *Tools* and *Info*. These are discussed in detail in the *User's Guide: Core Objects* manual. Many of the menu items in these menus have global keyboard shortcuts.

At the top of the window is a toolbar that contains a number of buttons that provide shortcuts to operations available in some of the menus on the menu bar.

At the bottom of the window is a toolbar that displays a memory meter – indicating the current amount of memory allocated to the application and how much of that memory is still available. The memory requirements of image processing software such as *Streams* can be severe and it may be necessary to carefully monitor the system memory during intensive processing operations.

The application view is special in that closing it causes the application to quit.

2.2.2 Object list view

All objects currently open in the *Streams* system are displayed in the Object list view that appears on the left hand side of the application view window. In all, there are nine types of object, and objects of the same type are grouped together in the object list view, each group being separated by a dashed dividing line. The nine object types are:

- Image sequences (chapter 3)
- Particle records (chapter 6)

- Particle 2D record bundles (chapter 6)
- Eulerian Field time series (chapters 9, 10, 11, 12, 13, 14)
- Lagrangian path fields (chapter 15)
- Tablets
- Free processes
- Fields
- Manuals

2.3 Entities and processes

The objects encountered in *Streams* can be divided into two categories – **entities** and **processes**.

Entities are objects that represent inputs to, or outputs from, the image processing system. For example, image sequences, particle records, the various Eulerian field time series – velocity, displacement, intensity, velocity-intensity, density and shape field time series – and Lagrangian path fields are all entities. Processes on the other hand represent actions that can create or manipulate entities. For example, an image filter pipeline process takes the images in an image sequence, processes the pixel information in each image, and produces another image sequence that contains the processed images. Considered simplistically entities can be viewed as data objects and processes can be viewed as algorithmic, or functional, objects.

Processes fall into two further categories – **free** and **bound**. A free process can create an entity, or perform some action, without requiring a specific entity as an input. An example of such a process is a process that can create a particle record based on some theoretical model (these processes are discussed in chapter 6). Free processes are created through the process menu of the *Streams* application view, and once created are displayed in the *Streams* object list view.

Bound processes, as their name suggests, are bound to an entity in order to perform their action. The image filter pipeline mentioned above is an example of a bound process. It requires an image sequence in order to execute. Another example is a create velocity 2D field process. This process operates on a particle 2D record to produce a time series of two-dimensional velocity fields, based on the particle matches in the particle record.

Bound processes can be separated into two further groups – primary and secondary processes. A **primary** bound process is one that can operate directly on the entity to which it is bound, whereas a **secondary** process is one that cannot operate directly on that entity. The definition of primary and secondary processes is clearly dependent on the entity under consideration. For example an image filter pipeline is a primary process for an image sequence but a secondary process for a particle record. This distinction is discussed further below.

All entities, that support bound processes, possess a process view that displays all bound processes currently defined for that entity. Bound processes are created and manipulated in this view.

Bound processes can be locked by the user to ensure that none of their settings are changed. Once the parameters for a particular process have been refined this option guarantees that they cannot be accidentally adjusted.

2.3.1 Process pipelines

Bound processes provide much of the core processing power of the *Streams* system. Individually they enable the user to process one entity to either produce a new entity, or to change the state of the original entity. An image filter pipeline is an example of a bound process that produces a new entity, another image sequence, while a PTV 2D analysis pipeline, that is designed to match particles between frames in a particle 2D record, simply manipulates the original entity without producing a new one. Thus all processes can be viewed as having an input entity and an output entity, even if the output entity is, in fact, the same as the input. This idea leads naturally to the concept of a **process pipeline**, whereby multiple processes are “connected” together so that the output of the first process provides the input to the second process, the output of the second process in the pipeline becomes the input to the third process and so on. Process pipelines enable the user to automate their image analyses to a very high degree. As an example of a typical particle tracking application the following process pipeline might be created to operate on an image sequence containing the raw images captured during a laboratory experiment:

- A filter pipeline preprocesses the images in the image sequence

- A particle identification process identifies the particles in the preprocessed images to create a particle 2D record
- A particle 2D record transform maps the pixel coordinates of the particles to physical coordinate and produces a new particle 2D record
- A PTV 2D analysis pipeline matches the particles from frame to frame in the particle record
- A create velocity 2D field process produces a time series of two-dimensional velocity fields by interpolating the particle based velocities in the particle record onto a uniform rectangular grid
- A velocity 2D field transform scales the velocity field and the spatial and temporal coordinates so that the resulting velocity field time series is in dimensionless form
- A save velocity 2D field process saves the final dimensionless velocity field to a file on disk.

Each entity, that supports bound processes, defines a **process pipeline** – a linked sequence of bound processes that interact in the way described above. This pipeline is created and managed within the entity's process view where bound processes can be added, inserted and removed from the pipeline. As a process pipeline is likely to contain both primary and secondary processes all types of bound processes can be created in an entity's process view. For example a PTV 2D analysis pipeline process that takes a particle 2D record as its input, can still be created for an image sequence. By itself this process cannot execute on an image sequence, but it may take its logical place as part of a pipeline that generates a particle 2D record at some point. This mixture of processes must be carefully managed by *Streams*. Different colours are used in the process view to indicate which processes are compatible with the entity itself, those which are compatible but are currently inadequately defined, and those which are incompatible with the entity. Similarly the processes that form the entity's process pipeline are checked for compatibility and the user's attention is drawn to incompatibilities through the use of a similar colour scheme. A process, or process pipeline, that is malformed will not execute.

It is important to understand that when a process pipeline executes only the entity that is output by the final process in the pipeline is created and stored in the system. All intermediate entities are discarded once they are no longer needed, unless they are explicitly stored on disk by the use of a save entity process inside the pipeline. These save entity processes will be discussed in later sections.

One aspect of process pipelines that requires some care from the user is the specification of parameters in processes that are somewhat removed from the entity that is to be processed. For example the PTV 2D analysis pipeline process in the example given above has no information about the particle 2D record upon which it is to operate until that record is actually produced by the preceding process. The PTV 2D analysis pipeline can specify the frames within the particle record that are to be analysed and the frames chosen must lie within the valid range for the particle record. For some parameters, such as the range of frames, the user can choose to use some default setting such as all the frames. In this way the process decides on the range of frames once that information becomes available as the process executes.

Image processing applications can be processor intensive. One of the aims of process pipelines is to automate the processing so that the user is not required to intervene at each step. This automation is taken a step further with a free process known as a **batch process**. A batch process contains a list of entities that are currently stored on disk. When the batch process is executed each of the entities is loaded from disk and their process pipeline executed. On completion of the pipeline the entity is closed and the next entity in the list is loaded and its pipeline executed. Provided care is taken in defining the processes within the pipeline of each entity the batch process can relieve the user of any intervention in the processing of large amounts of data.

Batch processes are created via the process menu of the application view.

2.3.2 Pipeline-based processes

In the discussion to date a number of processes also have the word pipeline in their names. For example the PTV 2D analysis pipeline or image filter pipeline. These processes are also based on the concept of a pipeline whereby the output from one sub-process provides the input for the next sub-process in the pipeline. The filter pipeline is a good example of a process that possesses sub-processes. An image filter is simply an algorithm that manipulates the pixel values in an image. The input to an image filter is the same as its output, a pixel array. Thus filters can be linked into a pipeline so that the pixel data from an image fed into the pipeline will pass through a number of processing algorithms. These pipeline-based processes

behave in much the same way as the higher-level process pipeline. Compatibility between elements in the pipeline is checked and errors are clearly highlighted for the user.

2.4 Core entities

In this section we will very briefly introduce the entities defined within the *Streams* system. In general, these entities equate to datasets that arise as part of the image processing system and which will provide the user with their key experimental results.

2.4.1 Image sequences

Image sequences stand at the core of the *Streams* system. These objects represent a sequence of images captured by some type of imaging device. The creation of an image sequence is generally the first step in any image processing activity. These are discussed in the next chapter.

2.4.2 Particle records

Particle records, both two-dimensional and three-dimensional, represent a time series of frames each of which contains a set of particles in two or three dimensions. Particle records can be analysed in a variety of ways, the most common being with a PTV analysis that will ultimately lead to the calculation of Eulerian velocity, or Lagrangian path, fields. Chapter 6 is dedicated to particle records. Two-dimensional particle records, particle 2D records as they will be called throughout these manuals, are created from image sequences when a particle identification algorithm is applied to the images. On the other hand three-dimensional particle records, termed particle 3D records, require multiple particle 2D records together with associated camera information to be created. Particle 2D record bundles are designed for this purpose.

2.4.3 Particle 2D record bundles

Particle 2D record bundles provide the mechanism by which multiple two-dimensional views of a three-dimensional particle field can be combined to produce the three-dimensional field itself. A bundle comprises a number of what are termed **perspectives**. A perspective includes both the two-dimensional particle field recorded

by a camera and information about the camera itself. Bundles are discussed in more detail in chapter 9.

2.4.4 Eulerian field time series

Eulerian fields are often the final objective sought by an experimentalist. Eulerian fields define some flow, or field, variables on a three or four dimensional rectangular grid – the dimensions of which are x, y, possibly z, and t. *Streams* provides the user with access to a number of these fields.

- An intensity field time series is a three-dimensional field of pixel intensities. These intensities, if computed appropriately, will coincide with some scalar flow field such as density or tracer concentration. Spatial derivatives of the scalar field are also stored.
- A velocity field time series is a three-dimensional (for 2D PTV) or four-dimensional (for 3D PTV) velocity field. The velocity vector field, together with its spatial derivatives, is computed at each node in the grid.
- A velocity-intensity field time series is a three-dimensional field with the velocity and a scalar defined at each point. This combined field is most likely to arise from a PTV-LIF experiment and it enables the user to compute combined field quantities such as turbulent fluxes.
- A displacement field time series is a three-dimensional (for 2D PTV) or four-dimensional (for 3D PTV) displacement field. The displacement vector, together with its gradients, are stored at nodes in a grid. Strain fields can be computed from this field information. A 3D surface displacement field represents 3D displacements on a 2D surface. Gradients are stored and strain fields can be computed.
- A density field time series is a three-dimensional field that stores the density of particles in the original images. It is not the density of the fluid that is stored.
- A shape field time series is a three dimensional field that stores the distribution of some particle shape property, for example its diameter or aspect ratio.

The first of these is created directly from an image sequence. The second, fourth, fifth and sixth are derived from a particle record, or potentially a Lagrangian path field in the case of displacement fields, through a corresponding process. The third is created by combining two previously created velocity and intensity fields.

Eulerian fields have a number of things in common. The most important of which is the concept of a calculator. Three- and four-dimensional fields are difficult to visualize and understand. Each Eulerian field possesses a number of predefined calculators that give

the user the opportunity to extract certain information from the field and to present it in digestible chunks. Take a velocity field time series for example. This field provides calculators that can extract the velocity vector field, individual scalar fields corresponding to the x, y and possibly z velocity components, scalars fields corresponding to vorticity, turbulent stress and so forth. Thus each calculator extracts some physically interesting component of the flow field. In addition, the user can select the domain in which they wish to view the calculated field. For example the user could choose to look at the time-average of the vorticity field, or perhaps the velocity along the centreline of the flow. Each of these projections can be displayed in a number of ways and will be described further in section 2.5.

For most Eulerian fields the user is at liberty to create their own customised calculators. These will be explained in later sections.

2.4.5 Lagrangian path fields

Particle tracking velocimetry computes the velocities of particles as they travel. Unlike PIV systems, which are inherently Eulerian in nature, PTV systems have the capability of producing Lagrangian field information – in other words, displacement and velocity data following a particle. A Lagrangian path field, in 2D or 3D, is an entity, derived from a particle 2D record, or particle 3D record, that provides the user with access to the displacement and velocity data for each particle as a function of time.

A number of calculators are defined for Lagrangian path fields and these provide functionality similar to that associated with Eulerian fields.

2.5 Fields

The data created by the Eulerian and Lagrangian field calculators are scalar and vector fields with dimensions of 0, 1, 2, 3 or 4. In the *Streams* system these are simply referred to as **fields**. They can be saved to disk as separate entities and reloaded into the system for later analysis. They differ from other entities in that they cannot have processes defined for them and thus do not possess a process view. While the Eulerian and Lagrangian fields already described are the end point of most image processing tasks, it is through the fields generated by their calculators that the user actually gets to view and understand their data. The *Streams* environment provides the user

with a range of powerful visualization tools for each of these fields and a separate manual is dedicated to them - *User's Guide: Field Objects*. For example a one-dimensional scalar field can be viewed as a traditional x-y plot, as a table of numbers, as a plot of probability density function (PDF), or in frequency space via a Fast Fourier Transform (FFT).

The graphical views associated with these fields can be saved and reconstructed at a later time. A view saved in this way is termed **persistent**.

These graphical views also can be saved as images for later inclusion in reports or presentations and the user has considerable control over the colours, font size, line width and other formatting details.

For higher dimensional fields that provide animation views, the animations can be saved as sequences of images for later movie production.

2.6 Tablets

An important addition to *Streams 3.00* was the **tablet**. Tablets provide the user with a further mechanism for manipulating and visualizing field data. A tablet is a container of fields and related entities known as grids, matrices, scalars, strings and custom functions. New fields are created through a command-line interface that provides powerful functionality. For example, it is relatively easy to compute auto- or cross-correlations of velocity field components within a tablet. While users may frequently extract field data from *Streams* for further analysis in an external tool, such as a spreadsheet or Matlab[®], tablets provide functionality that may reduce the need for this.

Tablets are also useful as a simple management tool whereby related fields can be located and stored in one place.

Tablets are described in detail in the *User's Guide: Tablets* manual.

2.7 Formulae

A useful feature of *Streams* is the provision of user-defined **formulae**. The Eulerian and Lagrangian fields possess a number of predefined calculators based on the flow variables stored in each.

For intensity, velocity, displacement and velocity-intensity fields the user is at liberty to create their own custom calculators from the field variables at hand. For example, the user can create custom scalar and vector fields using formulae based on the field variables available in a velocity 2D field time series. These formulae can include flow variables, the independent variables, x , y , and t , as well as mathematical functions, the standard unary and binary operators and brackets. A number of special functions are provided that enable the user to average variables over any combination of x , y and t .

In reality almost none of the predefined calculators are necessary as all could be constructed using a custom calculator. However, the reason that predefined calculators are retained is that they are computationally significantly faster, and for many applications the predefined calculators are more than sufficient.

Formulae are used in other areas as well including the definition of custom image filters and customizable costings as described in chapters 4 and 8, as well as in the command-line of tablets.

2.8 Manuals

Streams does not provide an on-line help system. Instead four comprehensive manuals, including this one, are available in PDF format. While these manuals may be viewed using a standard PDF reader *Streams* provides a simple interface for loading and viewing the manuals within the application itself. Bookmarks, that facilitate fast navigation of the manuals, are supported.

2.9 Summary

This section has provided a broad overview of some of the fundamental design concepts behind the *Streams* system. The key concepts of entities and processes have been described in preparation for more detailed explanations of the core entities in later chapters.

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 core object in the *Streams* 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. Frames within an image sequence are assumed to have been captured at a uniform rate, or obtained in a pulsed mode, whereby frames are captured in pairs a very short time interval apart while pairs are captured at a slower rate. This pulsed mode is supported by certain scientific digital video cameras that are often used for PIV work.

Image sequences work with digital image files. The file formats that are supported are JPEG, GIF, PNG, BMP and TIF. *Streams* does not work directly with digital video file formats such as AVI, MOV or MPEG. Therefore all video records must be converted to individual image files before *Streams* can process them. Adobe Premiere Pro[®] or Quicktime Pro[®] are examples of video editing programs that can perform this conversion – at least to some image formats. In addition, it is worth noting that *Streams* does not provide any direct support for image capture.

The choice of format is important. Some file formats are small and lossy, others are large and lossless. Others, such as GIF, have a very 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. However, the last two of these are uncompressed and therefore the size of an image file stored in these formats is likely to be considerably larger than one stored in the PNG format.

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. *Streams* 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 provide information about transparency. The alpha value is not used by *Streams* when interpreting the pixel colour information.

However, *Streams* also needs to work with gray-scale images and, at times, images containing real numbers. Therefore *Streams* defines a number of formats to accommodate this variety of data types. Up until, and including, version 3.1 of the software *Streams* specified its own standard for the storage of gray scale data – namely, each pixel was assumed to store a 32 bit RGB value with the most significant 8 bits in the gray-scale signal being stored in the red channel, with the remaining bits stored in the lowest 2 or 4 bits of the green channel, depending on whether 10 or 12 bit gray scale images were involved. The blue and alpha channels were ignored in this case. In *Streams* 3.2 and beyond the software has adopted the GigE Vision Streaming Protocol[®] (GVSP) standard whereby the most significant 8 bits are stored in the red channel while the least significant bits are stored in the most significant bits of the green channel.

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 point number directly into the pixel colour value. Thus all of these data types can be stored in a standard RGB image and saved 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 through the top-level image menu in the *Streams* application view. Once created the image sequence is

displayed in the *Streams* object list view where the user can interact with it through its popup menu.

During the creation process 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. The image sequence can reference the images in an absolute or relative way depending on the needs of the user. When absolute referencing is used the image files are assumed to reside in a fixed directory in the file system. The image files must not be moved or the image sequence will not be able to access them. With relative referencing the images are stored in a sub-directory of the directory in which the image sequence is stored. When the image files are accessed they are always accessed relative to the directory in which the image sequence is stored. Thus both image sequence and image files can be move to another location in the file system while still allowing the image sequence to access the images.

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 the user can choose to check all other images in the sequence 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, together with such information as the time step between frames (and the time step between images in a pulsed pair if the images are from a pulsed source). In simple imaging systems, where camera distortion is unimportant and the camera's line of sight is normal to the flat surface being viewed, the conversion from pixels to physical coordinates can be achieved by specifying scale factors in the x and y directions. To cater for more complex particle tracking applications non-linear mapping functions can be employed for the same purpose. The user can also provide text strings that indicate the temporal and spatial units being used.

Some limited support is also provided for images captured by JAI[®] cameras and saved on disk using the JAI proprietary format (.JAI). In this case the image sequence creation follows a similar process to

that described above except with an additional step. This step involves each JAI image being loaded, converted to one of the standard image formats, and saved to disk. The resulting image sequence then references these converted files instead of the original JAI files.

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 Image menu. All image sequences are stored with a *.seq* extension.

3.5 Processes

In this section we provide a brief introduction to the various primary processes associated with image sequences. Some of these will be described in further detail in later chapters and information about specific parameter choices can be garnered from the *User's Guide: Core Objects* manual. Processes that deal directly with other entities will be discussed in the appropriate chapter.

3.5.1 Image filter pipeline

We have already talked briefly about image filter pipelines. These processes provide the image processing subsystem of the *Streams* environment. A range of flexible and powerful image filters enable the user to convert raw images into images that are suitable for such processing as particle identification, or the extraction of scalar field data such as concentration or density. These will be discussed in detail in the next chapter.

The input to an image filter pipeline is an image sequence, and the output is also an image sequence.

3.5.2 Particle identification

Particle identification processes interrogate the pixel intensity information stored in an image in order to identify two-dimensional particles within the image. These particles are at the heart of the particle tracking velocimetry (PTV) system, whether in two-

dimensions or three, but they can be useful in a variety of other applications as well. For example, simply the physical characteristics of the particles, such as shape and size, could be useful to the user and *Streams* provides functionality to record and analyse these characteristics.

Particle identification algorithms are discussed in some detail in Chapter 5.

The input to a particle identification process is an image sequence, and the output is a particle 2D record.

3.5.3 Create intensity field

The ultimate aim of the *Streams* system is to provide the user with physically important information about a physical system. In many cases this information will be two- or three- dimensional displacement or velocity fields obtained by tracking particles through a sequence of frames. However, another very important application of image processing is the measurement of concentration, or density, fields through the analysis of images of dyed fluid systems. A number of these techniques have been discussed in Chapter 1. In all cases the physical quantities of interest are related to the intensity of light captured by a camera. Some of the filters provided by *Streams* are specifically designed to convert raw images into a state where the pixel intensity is related to the concentration, density or some other scalar property of the flow. Once images have been preprocessed the scalar field (generically called the intensity field) can be computed on some user defined rectangular grid. This is achieved with a create intensity field process which will be discussed, along with the intensity field time series itself, in chapter 9.

The input to a create intensity field process is an image sequence, and the output is an intensity field time series.

3.5.4 Transforms

Streams provides a number of image sequence transforms. These processes operate on an image sequence and output another image sequence. However, unlike a filter pipeline that has similar inputs and outputs, these transforms will produce an image sequence that has fewer images, images of a different size, or that requires a

number of images in the sequence to be processed to produce each image in the output sequence. Thus they do not naturally fit within a filter pipeline framework where all images passing through the pipeline are assumed to have the same size, and the image sequence produced by the pipeline will possess the same number of images as the original.

Six different transform processes are provided. Here we will provide a brief description of each of these, leaving a detailed discussion to the *User's Guide: Core Objects* manual.

3.5.4.1 *Average images process*

This process collapses the images in an image sequence into a single image in which the pixel intensities are the average of all of the images in the original sequence. An example of the use of this process is when a sequence of images represents a single calibration image that is obtained after averaging.

In some applications a flow is steady in a moving frame of reference. This process allows the user to specify a constant velocity of translation which effectively transforms the images into the frame of reference where the flow is steady.

3.5.4.2 *Trim images process*

This process trims all images in the image sequence so the resulting images are some rectangular subset of the originals. The output images will have a smaller pixel dimension than the originals, and hence will require less storage if the same file format is retained.

3.5.4.3 *Scale images process*

The scale image process also decreases the size of the images but in this case the images are scaled down. Pixel information is lost but the entire scene is retained.

3.5.4.4 *Subtract images process*

This process produces a new image sequence that contains one less image than the original. Images are constructed by subtracting the pixel information of adjacent images – either the first from the

second or vice versa. The resulting difference images will clearly highlight changes that have occurred between frames. When two frames are identical the difference image will have an intensity of zero at every pixel.

3.5.4.5 *Rotate images process*

This process rotates the images in the sequence. The resulting image sequence will have the same number of images but in the vast majority of cases the images will be larger than the originals. The additional pixels created through the rotation are filled with a user-defined colour.

3.5.4.6 *Long exposure images process*

This process combines a number of sequential images into a single image in order to mimic the type of image that would be obtained if a still camera with a slow shutter speed had been employed to capture the images. The user can choose how many images to combine and the algorithm used to combine the intensities at each pixel.

The input to these transform processes is an image sequence, and the output is also an image sequence.

3.5.5 Other processes

One other process is defined for image sequences. This save image sequence process, like the save process for other entities, is useful within a process pipeline when an intermediate result is to be retained. Images that pass through a process pipeline will be transformed internally but no record of them remain at the end of the pipeline unless a save image sequence process is inserted into the pipeline.

The input to a save image sequence process is an image sequence, and the output is the same image sequence.

3.6 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.

Often the images recorded from digital cameras are unable to be displayed in full on a computer screen due to its limited resolution. It is possible to shrink the image so that the entire image can be viewed at one time without resorting to the use of the window's scroll bars.

The two most common processes applied to image sequences are filter pipelines and particle identifiers. Both of these processes possess a range of parameters that affect their behaviour. In order to be able to explore this behaviour interactively filter pipeline and particle identification processes defined for the image sequence can be selected in the image view and their output displayed. In the case of filter pipelines the filtered images are displayed, while for particle identifiers the particles identified by the algorithm are overlaid on the original image.

The image view also allows the user to create regions. These regions, which can be rectangular, circular or polygonal, are areas within an image that can be used for a range of purposes, including defining certain filters.

3.6.2 Details view

The details view, like details views for all objects in the *Streams* 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, the size of each image in pixels and possibly the scale factors used to convert from pixels to physical units. 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.

3.6.3 Process view

Nearly all entities, including image sequences, possess a process view. This view provides two list boxes that display the currently defined processes for the entity and the processes that currently comprise the entity's process pipeline. New processes are created, or loaded, into the process list and then added to, or inserted into, the pipeline. Individual processes can be executed through their popup menus, and the view also allows the pipeline to be initiated.

The input and output entity types for all processes listed in the view are displayed next to the process. In this way it is easy to see which processes could be selected to produce a valid process pipeline.

This view keeps track of the state of the various processes in both the process list and the pipeline list. Processes in the process list can have one of three states. The first is a valid state which indicates that the process can execute on the image sequence directly and is ready to do so. Processes in this state are displayed in black. The second is a state of valid but not ready, indicating that one or more parameters in the process are required but currently are not set. An example of such a situation would be a save image sequence process where the name of the image sequence file is not defined. Processes in this state are displayed in blue. Processes in these first two states will be primary processes for the image sequence. The third state is one of incompatibility. Processes that do not take an image sequence as input, namely secondary processes, will be displayed in red. Only those processes that are in the first of these states can be executed as they stand. The second can be executed once the missing parameters are specified and the third group can never be executed individually.

The states of the processes in the pipeline are also colour-coded. If all processes in a pipeline are displayed in black the pipeline is valid and can be executed. When the input of a process is incompatible with the output of the previous process in the pipeline the second of

the processes will be displayed in red, as will all subsequent processes. A process in the pipeline that has valid inputs and outputs, but is not completely defined will again be displayed in blue.

It is important to note that a process that is not ready to execute individually may be perfectly valid as part of a pipeline. For example a filter pipeline, if executed singly, will need to store the filtered images it produces on disk, and a base name for these files must be defined before it can execute. However, if the filter pipeline is a component of a process pipeline, but it is not the final process, the filtered images are not saved to disk, they are simply passed to the next process in the filter pipeline. Thus the file name for the images need not be specified if the filter pipeline is simply an element within a process pipeline.

3.7 Summary

Image sequences are the starting point for nearly all analyses performed within *Streams*. This chapter has provided a brief overview of their creation and processing and has introduced the three views that the user can use to interact with sequences and the images within them.

4 Image Filter Pipelines

4.1 Overview

Image filters provide *Streams*' image processing capabilities. An image 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 image.

Filters can be **pipelined** so that the output of one filter can be used as the input to the next. Thus filters are organized into filter pipelines. If an image sequence has a filter pipeline 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 filter, 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 *Streams* 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 all conflicts have been resolved.

The application of a filter pipeline to an image as a standalone process 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 storing these filtered image files. At the same time a new image sequence is created that contains these filtered files. This sequence appears in the object list view.

The user is quite at liberty to create a number of different filter pipelines for the same image sequence. These could be processed sequentially as part of a process pipeline, in which case the net effect would be the same as if all the filters had been part of the same filter pipeline.

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 32 bit RGB colour images as their input and produce 32 bit RGB images after processing the pixel intensities. For example an extract filter takes a 32 bit 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 32 bit 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. *Streams* uses the 32 bit 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 32 bit 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 Image filters

In this section the full array of filters available in *Streams* will be described. Further information can be found in the *User's Guide: Core objects* manual.

4.3.1 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 are 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 physical thickness and concentration of the attenuating material.

Thus

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

where $d(x,y)$ is the calculated optical thickness, $I(x,y)$ is the intensity at a pixel, and $I_0(x,y)$ is a reference intensity at the same pixel. Here, the intensity stands for the intensity of the red, green, or 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 (or through software – see section 4.3.22) 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 for low concentrations) 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.

4.3.2 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 4.3.1 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.

4.3.3 Amplify filter

An amplify filter can be used to scale the intensity at each pixel. The filter can be defined for any data type (i.e. 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.

4.3.4 Barrel pincushion distortion filter

Most camera lenses cause some level of optical distortion in the recorded images. 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). The transformation used in *Streams* is

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

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 upon the location of the origin. Generally the origin is taken as the physical centre of the image but the user is at liberty to change this.

The 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 *Streams* 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. Its 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 optical 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.

Streams 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 *Streams*. 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. 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 (e.g. 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 Convolution filter

The convolution filter executes a mathematical convolution on the pixel matrix. Effectively this corresponds to replacing each pixel value with the sum of the product of a user specified matrix – the

convolution matrix – with a region of pixels surrounding the pixel of interest. The region of interest and the matrix have the same number of elements and are centred on the pixel of interest. The size of the matrix/region is at the user's discretion and the user is responsible for defining the elements of the matrix used in the convolution.

For any matrix containing more than one row and one column the matrix will “hang over the edge” of the image when applied to pixels on or near the edge of the image. The user can define the strategy that should be adopted for the computation of the convolution at these pixel locations. For example the matrix could be simply truncated at the edge, or the image could be “extended” in some way beyond the edge.

The user must also define a strategy for handling the possibility of the resulting convolution lying outside the allowable pixel values for the image type being processed. For example if the convolution produces a colour intensity greater than 255 or less than 0 for an RGB image then the resulting pixel value will be defined by this strategy.

4.3.8 Discrimination I filter

The discrimination I filter is intended to help the user in pre-processing images from which particles are to be identified. For small, sparsely located particles, typical of PTV experiments where velocities are measured internally to the flow the identification of particles is generally straightforward. However for applications where the particles are large and may be touching one another, as is often the case in the tracking of sediment particles on a channel bed, the discrimination of particles is very difficult.

The discrimination filter analyses a row or column of pixels by identifying the maxima and minima in pixel intensity along the line of pixels. Even when particles touch the intensity is generally less (or greater depending on whether the fluid or particles are illuminated) in the region of contact than the intensity within the particles themselves. Unfortunately this reduction can vary significantly within an image and this presents challenges to the particle identification algorithms. Therefore the discrimination filter takes the maxima and minima and scales all intensities along the line of pixels relative to these. Thus all maxima are given the highest pixel intensity available for the particle image type (note for a real

value image the maximum and minimum values are chosen to be 1 and 0 respectively), while minima are set to the lowest intensity. This normalization of the intensities greatly helps the particle identification algorithms. The intensity of pixels between a maximum/minimum pair can either be scaled (graded) between the highest and lowest values in a linear fashion based on their own intensity or be allocated in a binary way. In that latter case a pixel intensity that is less than the average of the original maximum and minimum intensities will be set to the same intensity as the minimum, while those above will be set to the maximum intensity.

All images contain some level of noise. Thus a large particle may appear bright in an image but the intensity across it may fluctuate slightly due to non-uniformities in the light source perhaps, or due to unevenness of the particle surface reflectance. In either case these minor maxima and minima are not of interest to the particle identification process and the user can specify a fluctuation threshold that will cause these minor variations in intensity to be ignored.

4.3.9 Discrimination 2 filter

The discrimination 2 filter is also intended to help the user in pre-processing images from which particles are to be identified. In a similar way rows or columns of pixels are analysed in turn. For each row or column the average pixel intensity is computed. The pixels in that row or column are then compared to the average and those whose intensity differs from the average by more than a certain amount or percentage have their intensities set to the maximum intensity possible (for real images this is taken as 1). Pixels whose intensities differ from the average by less than this amount have their intensities set to 0. Thus the intensities of the resulting image are of a binary nature.

In addition the filter can be instructed to treat gaps between maximum intensity pixels as noise such that pixels of 0 intensity that comprise these gaps have their intensities increased to the maximum. The maximum gap size is set by the user.

4.3.10 Eliminate objects filter

The eliminate objects filter is designed to pre-process images for PTV analysis, although it is not limited to this use. Frequently PTV images contain spurious objects caused by the lighting system. As

well as the illuminated particles light reflections from boundaries or other objects may be present. Typically these spurious objects are brightly coloured on a black background and when the particle identification process is undertaken these objects lead to the identification of incorrect particles. The filter operates in much the same way as the particle identification algorithms themselves. A threshold light intensity is specified by the user, and regions of connected pixels whose intensities exceeds the threshold value are identified. The factor that differentiates these invalid objects from the particles themselves is typically their size, not their intensity. Thus the user can specify a minimum and maximum number of pixels that must be connected in this fashion for the region to be identified as an undesirable object. All objects thus identified have all of their pixel intensities set to zero.

The eliminate object filter takes any type of input and produces an output of the same type.

4.3.11 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.

4.3.12 FFT filter

The FFT filter provides a simple 2D bandpass filter of an image. The user may specify the lower and upper bounds on the wavelengths that are passed by the filter. It can operate on the red, green or blue guns of an RGB image, the average of the three, or a 10 bit or 12 bit gray scale image or a real value image. The output of the filter can be of two types. The first is the filtered image in which case the output data type matches that of the input image. Alternatively the user can choose to retain the PSD from the FFT (using a linear or logarithmic scale), in which case the output data type is real.

4.3.13 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.

4.3.14 Invert filter

An invert 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 a particle identification process. This process 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. Thus the invert filter can be used to convert the images into a form that can be searched for particles.

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

4.3.15 Mathematical formula filter

The mathematical formula filter is a highly flexible filter that enables the user to describe the transformation to be applied to the intensity at each pixel using a mathematical formula. It can take any type of image as an input and produces a real image output. The output value at each pixel is computed from the formula specified by the user. Take for example an RGB image. The formula can include the red, green and blue intensities, the x, y and t variables, along with the standard set of unary and binary operators, brackets and a range of mathematical functions. For a gray scale image the gray intensity can be incorporated into the formula instead of the RGB intensities.

The output data type of a mathematical formula filter is always real values.

4.3.16 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.

4.3.17 Parallax factor filter

The parallax factor filter is designed to compensate in some way for the effects of parallax. For example in a light attenuation experiment the reduction in light intensity is related to the thickness of dyed fluid through which a light ray passes. If a wide angle camera lens is used in such an experiment then the light detected at the edge of an image will have travelled through a greater amount of dyed fluid than the light entering along the lens axis simply by virtue of the angled light path. The parallax factor filter computes the relative increase in light path and this can be used to recalculate quantities, such as the concentration of a tracer in the flow, that are being detected by the light attenuation technique.

4.3.18 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 (e.g. 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 (e.g. 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.

4.3.19 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.

4.3.20 Remove background filter

The remove background filter is designed for the preprocessing of images used in particle identification applications. The particles are typically bright regions on a dark background, but due to lighting non-uniformity the background intensity may vary across the image. The remove background filter attempts to remove the background intensity from the image leaving only the signature of the particles in the filtered image.

The filter uses a range of algorithms for this background removal. Essentially a region of pixels whose average intensity is representative of the local background intensity is used to obtain an estimate of the background intensity. This estimate is then subtracted from the local pixel intensity. The background intensity estimate can be calculated in two ways, using local or global information. When local information is used a region of pixels centred on each individual pixel is used to calculate the background intensity for that specific pixel. This region can be a solid square of pixels centred on the pixel of interest, or alternatively a hollow square or cross of pixels centred on the pixel. When global information is used the image is divided into a checkerboard of rectangular regions, the sizes of which can be selected by the user. The average intensity in each of these regions is used as the estimate of the background intensity and this is subtracted from the intensity of each pixel within each rectangle of the checkerboard.

No matter which of these algorithms is used any filtered pixel intensity that is less than zero is set to zero.

4.3.21 Scale by maximum in regions filter

The scale by maximum in regions filter is another filter designed to preprocess images to facilitate particle identification by managing variations in pixel intensity both in time and space.

The user specifies a number of regions within an image. Within each of these regions a maximum pixel intensity is computed and all intensities within these regions are then scaled so that the maximum pixel intensity becomes the maximum allowable value for the images type. Thus, for example, for an RGB image the maximum intensity pixel will be scaled to 255 and all other pixels will be

scaled by the same factor. This process can help to ensure uniform intensities across all regions. As the maximum intensities in each region are computed for each image variations in light intensity with time can be removed using this filter.

The user can choose to leave the pixels outside the specified regions unaffected by the filter or they can set these pixel intensities to zero.

4.3.22 Spatial average filter

The spatial average filter generates a pixel intensity that is computed from a spatial average of the pixels surrounding it. This average can be computed in the same three ways that the local background intensity is computed by the remove background filter – namely using a solid square, a hollow square or a cross. The effect of this filter is to smear or defocus the image.

The output data type matches the input data type.

4.3.23 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.

4.3.24 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 – i.e. red from red etc. By definition an image subtracted from itself results in an image whose every pixel has an intensity of zero.

4.3.25 Subtract image sequence filter

The subtract image sequence filter extends the subtract image filter. Instead of subtracting a single image from the images in an image sequence the images from another image sequence are subtracted. The two image sequences must include images of the same dimensions, but need not include the same number of images. If insufficient images are available in the image sequence being subtracted then the images in the image sequence without a matching image are left untouched by the filter.

The user may also specify the order of subtraction – i.e. which sequence is subtracted from which.

Time is used as the image selection mechanism in this process. For example, consider two image sequences captured at two different rates, one capturing frames at half the speed of the other. Suppose a subtract image sequence filter is defined for the first of these sequences with the second sequence specified as the image sequence to subtract from it. In that case only every second image would be utilized from the second sequence as the filter attempts to match times of the images in the two sequences.

4.3.26 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 all guns to 0 if the red intensity is greater than 127.

4.3.27 White balance image filter

For some applications an accurate white balance is required. An example is the use of the absorption filter with the light attenuation technique. While the Bayer colour filter has gain settings for each of the three colour guns, these gains are global gains applied to every pixel in the image. The white balance image filter enables the user to apply different gains to each pixel in the image. The gains are computed from a reference image (which must be a true RGB image – i.e. an image taken by a Bayer filter camera must have been processed using the Bayer colour filter) that is nominally a monochrome image. At each pixel of the reference image a gain for each of the colour guns is computed so that if these gains were applied to the reference image the red, green and blue guns at each pixel would be the same. These gains are then applied pixel by pixel to the images in the image sequence to which the filter is applied.

4.4 Creating and initialising filters

Filter pipelines are created and displayed in the image sequence process view. The actual filters are created in the pipeline's 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 by opening the details view of the filter through its popup menu. 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 filter pipeline filter view highlights any inconsistencies in the filter pipeline associated with incompatible input and output data types as described above.

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 may include new filters.

5 Particle Identification

5.1 Introduction

A **particle identifier** is defined as an algorithm that processes a digital image by locating particles within the image. The term particle is used here in a generic way. In reality particles are simply regions within an image that have certain characteristics that differentiate them from the rest of the image. For example an image of cars on a motorway may be analysed in order to identify the number of cars and their locations. In this case the cars are the particles.

In fluid dynamical research particles are rather more likely to correspond to small physical objects suspended in a fluid flow, and it is this application that has motivated the development of the *Streams* system. Thus all particle identifiers have been developed with this application in mind and they may not work as effectively in applications where the particles take on quite different characteristics.

Common to all particle identifiers is the assumption that particles are identified as regions within an image where the intensities of the pixels contrast with those in the rest of the image. In other words it is the intensity, and sometimes the colour, of the light recorded in the image that is used to isolate particles from the background. Also common to all particle identifiers is the assumption that particle intensities are higher than the background. Thus a particle can be found if it is brighter, or redder, or greener than the background, but it cannot be found if it is dimmer, or less red and so forth.

Some image capture systems may not produce images where the contrast between background and particle matches that required by the particle identifiers. Such images will need to be preprocessed using a filter pipeline before being analysed with a particle identifier. For example images that contain dark particles on a light background can be readily converted into light particles on a dark background using an invert filter. Non-uniformity of background lighting can also cause problems for a particle identifier which uses an absolute intensity threshold to locate a particle in an image. The remove

background filter, the mathematical formula filter or the subtract image filter might be useful in this regard.

Each particle identifier calculates certain quantities for the particles it discovers – the x,y coordinates of the centre of the particle, its area and effective radius, and colour are all stored with the particle. The area is calculated from the number of pixels comprising the particle, effective radius or diameter is calculated from the area assuming that the particle is circular. The user may choose to store further shape information about the particles and even the intensity matrix that comprises the particle. If shape information is required *Streams* computes the principal axes of the identified particle, and the corresponding moments of inertia about these axes. The angle of the major axis to the horizontal is recorded, as is the aspect ratio of the particle deduced by assuming that the particle has an elliptic shape. One further quantity is calculated, called the eccentricity. It is a dimensionless measure of the distance between the centre of the particle and the centre of the rectangular region that contains the particle. For a symmetric particle these two points coincide, but for irregular shapes they do not and the eccentricity is then a gross measure of irregularity of the particle shape.

If the particles themselves are of particular interest to the user these bulk measures of the properties may be insufficient and the actual image of the particle may be required. This image is termed the particle's **intensity matrix**. Effectively the pixel intensities within the smallest rectangular region that contains all of the pixels that comprise the particle are stored with the particle. While this information might sound interesting it is important to realize that it is storage intensive and for large particle records it is likely to place unmanageable demands on the system memory. Particle intensity matrices should only be stored if absolutely necessary and then only for particle records of modest size. The additional particle statistics require considerably less space than the intensity matrices but even so they too should only be recorded if there is good reason.

The colour (red, green and blue) intensities are stored for each particle. In the case of grayscale images all three colour components will be the same but for RGB images they can be unique. The various particle identification algorithms described below use the colour information in different ways, but in each case three colour components are stored.

Each of these colour components is stored as an 8 bit value even if the original image was 10 or 12 bit monochrome. It is believed that the lost information is not important in the vast majority of PTV applications.

It often happens that images within an image sequence contain regions that do not contain genuine particles. For example stray light reflections off a flow boundary will lead to the identification of spurious particles. The user can specify regions within the images that are excluded from the analysis process. These regions are most easily defined in the image view of an image sequence (see section 3.6.1).

Each particle identifier has its own strengths and weaknesses and the selection of a particle identifier needs to be made with these attributes in mind. In the discussion below some of these strengths and weaknesses are discussed.

It should be noted that both two- and three-dimensional particle fields are created from two-dimensional images of the particles using the particle identifiers described in this chapter. In the case of two-dimensional particle fields this identification process leads directly to the creation of a particle 2D record. In the case of three-dimensional particle fields multiple cameras will be in use and a particle 2D record will be created for each camera. Then, once the cameras have been correctly calibrated a full three-dimensional particle field will be reconstructed from the set of particle 2D records via a particle 2D record bundle (see chapter 8).

5.2 Particle identifiers

In this section the set of particle identifiers available in *Streams* will be described. Further information can be found in the User's Guide: Core Objects.

5.2.1 Single threshold particle identifier

The single threshold particle identifier is the simplest of the particle identifiers (PIDs) available and comes in two forms. In the monochrome version the intensities of the pixels in the image are used in the particle identification process. Particles are found by searching the image for pixels whose intensity is greater than a threshold value specified by the user. The user can also select which

colour gun, in an RGB image, forms the basis for this intensity calculation, intensity here is used to refer to any of these intensities, not necessarily the average intensity.

In the colour version all three colour components are utilized. The user can specify separate threshold values for each colour and whether a pixel must exceed all of these thresholds or just one of them in order to be counted as part of a particle.

Once a pixel is found whose intensity/colour meets the specified criteria, all pixels connected to that pixel, whose intensities/colours meet the criteria, are treated as part of the same particle. This process is repeated until the entire image has been processed.

The user is provided with some additional control over the particles that are generated with this identifier. A lower limit on the particle diameter can be set, which ensures that all particles whose effective diameter is less than this value are discarded. Similarly a maximum diameter and maximum aspect ratio can be specified. If the user specifies a value of -1 for any of these parameters then no limit is imposed.

The location of the particle is computed to sub-pixel accuracy by computing the centre of mass of the pixels identified as comprising the particle. The user can choose to weight each pixel location by its intensity when the centre of mass is computed. Thus if the particle is thought of as a flat plate the intensity becomes a proxy for the plate density or plate thickness.

5.2.2 Dual threshold particle identifier

The dual threshold particle identifier is similar to the single threshold monochrome PID but provides more control over the determination of the particle size. A peak intensity threshold is used to identify pixels that signify the presence of a particle in the same way as the threshold setting in the single threshold PID. However, before the pixels that comprise the particle are found, the PID searches for the maximum pixel intensity in the region comprising the particle. Once found this maximum intensity is multiplied by a user defined edge threshold factor to determine the pixel intensity that is used to specify whether a pixel is part of the particle or not. For example, suppose the peak intensity threshold is set to 50 and the edge threshold factor is 0.5. Now imagine that a pixel of intensity 65 is

found, and the maximum intensity in the neighbourhood is 80. For this particle the edge threshold is then $80 \times 0.5 = 40$. Thus all pixels connected to the pixel with maximum intensity, and whose intensity exceeds 40, will be included in the particle.

Clearly the threshold used to determine whether a pixel is part of a particle or not will depend on the maximum intensity within each particle and thus will vary from particle to particle. This algorithm has the advantage that particles, that may lie towards the edge of a light sheet, and therefore have a lower intensity than a corresponding particle in the centre of the light sheet, will have a more realistic size than that computed using the single threshold PID.

As for the single threshold PID the user can specify a lower limit on the particle diameter, a maximum diameter and a maximum aspect ratio.

This PID also calculates the position of each particle using the centre of mass calculation but does not include weighting by the intensity.

The user can choose whether to store the particle's colour or just its intensity (i.e. the average colour leading to a monochrome particle). In either case the particle colour or intensity is obtained by averaging over all pixels comprising the particle.

5.2.3 Dual threshold Gaussian particle identifier

The dual threshold Gaussian particle identifier has been designed specifically to process images used in particle tracking velocimetry experiments. The primary goal of this PID is the accurate determination of the position of the small particles typically used in PTV experiments.

A peak threshold intensity is used to locate a pixel that signifies the presence of a particle. As for the dual threshold PID the pixel with maximum intensity connected to this pixel is sought. An edge threshold intensity is used to determine the extent of the particle in the pixel map.

The chief difference between this PID and the previous two is the manner in which the particle position is computed. Instead of using the centre of mass of the pixels comprising the particle, a Gaussian

fit is made to the pixel intensities in the x and y directions around the pixel of maximum intensity, and the maximum of this two-dimensional Gaussian is taken as the particle centre. Provided the particle comprises at least 3 pixels in each of the coordinate directions this Gaussian fit is known to give sub-pixel accuracy for the particle position. If less than 3 pixels are available in either direction a Gaussian fit in that direction is not possible. If the particle extends only a single pixel in one direction the particle is centred in the pixel. If it extends two pixels the centre is located somewhere between the centre of the pixel with maximum intensity and the boundary between the two pixels depending on the relative intensities of the pixels. The closer the intensities of the two pixels the closer the centre will be to the boundary between them.

This PID also differs from the other two PIDs in how it handles multiple intensity peaks. If more than one intensity maxima is found within the same particle, this algorithm splits the pixels into multiple particles – the number of particles matching the number of intensity peaks.

As for the simple threshold PID the user can specify a lower limit on the particle diameter, a maximum diameter and a maximum aspect ratio.

The user can choose whether to store the particle's colour or just its intensity (ie the average colour leading to a monochrome particle). In this case the particle colour is that of the pixel of maximum intensity while the intensity is the maximum intensity predicted by the average of the two Gaussian fits to the intensity profiles. If this exceeds the nominal maximum of 255 the intensity defaults to this value.

This PID works very well for small particles that are approximately spherical in shape and that extend at least 3 pixels in each coordinate direction. For these particles, provided they are uniformly lit, highly accurate particle locations should be obtained.

5.3 Summary

This chapter has provided an overview of the particle identification algorithms.

6 Particle Records

6.1 Overview

Particle records are the second core set of objects in the *Streams* system. They come in two varieties, particle 2D records and particle 3D records. The difference between them has been explained previously. While these two objects differ in dimensionality much about them remains common. This chapter will talk about particle records in general and will identify specific differences between two- and three-dimensional particle records as they arise.

In practical applications particle 2D records are created from image sequences using a particle identification process as described in the previous chapter. Particle 3D records on the other hand are created from groups of particle 2D records stored in particle 2D record bundles. These bundles will be discussed in more detail in chapter 9.

Particle records of both types can also be created from theoretical models that enable the user to explore and understand the *Streams* system without having to resort to the acquisition of experimental data. These models are described in section 6.3.

In addition, particle records can be constructed from text files generated by the user. These text files are discussed in section 6.4.

Both types of particle record can produce an appropriately dimensioned velocity field time series or Lagrangian path field. Particle 2D records can also spawn a number of other Eulerian field time series (for example a shape field time series).

Particle 2D records are saved to disk with a *.par2* extension (previously *.par*) while particle 3D records are stored in files with a *.par3* extension.

6.2 Particles and frames

A **particle record** comprises a time ordered series of **frames** each of which contains a list of **particles**. The particles are stored in no particular order although they tend to be identified starting from the top of the image in the case of particle 2D records. Particle 2D

records, like image sequences, can possess frames separated by a constant time step, or have frames that come in pulsed pairs.

A range of information is stored with all particles. This information is

- A particle number unique within the frame (the numbers in a frame start at 0).
- The x, y and possibly z coordinates of the particle.
- The radius of the particle assuming that it is circular or spherical.
- The colour of the particle.

If the user wishes they can choose to store additional statistical shape information about each particle in a particle 2D record during the identification process. This information is

- The angle the major principal axis of the particle makes with the x axis. The particle, defined by the pixels that comprise it, is assumed to be a flat plate in order to compute the principal axes.
- The moments of inertia relative to the two principal axes. In reality these values are of little interest to the user and if the particle is assumed to be elliptical in shape they can be used to compute the aspect ratio of the particle. This aspect ratio is the ratio of the length of the major axis to that of the minor axis, and hence always has a value greater than or equal to 1.
- The eccentricity of the particle as described in the previous chapter.

In addition, the user may choose to store the intensity matrix for each 2D particle. Effectively this is the pixel map that defines the particle.

For most PTV applications the core particle information is probably all that is required in the subsequent data analysis. Therefore the more refined shape information and particle intensity matrix should not be stored with the particle unless absolutely necessary.

6.3 Particle record creation from models

As noted above particle records, both two-dimensional and three-dimensional, can be created using a theoretical model. These theoretical models play a useful role in the understanding of the particle tracking performance of the *Streams* system. The theoretical models, based on analytical expressions for the velocity field,

simulate the motion of randomly distributed particles within the flow. Because the particle motion is artificially constructed the actual particle paths are known. Thus the results of any PTV analysis performed on these model-generated particle records can be compared with the correct results. We have found this to be a useful way of exploring the performance of the different matching algorithms discussed in chapter 8.

For each model a set of randomly located particles is generated in the first frame in the particle record. Using the known velocity fields, the displacement of each particle is calculated from frame to frame using a 4th order Runge-Kutta time-integration of the ordinary differential equations governing the particle motion. A particle may leave a frame by flowing through one of its boundaries, and the user has the choice of reseeding such particles. If the user chooses to “wrap the axes” these particles are reintroduced through the opposite boundary, otherwise they are randomly distributed across the frame. The loss of particles from the domain, perhaps due to exiting the light-sheet in the two dimensional case or being obscured by other particles in the three-dimensional case, can be simulated by randomly removing, and reseeding, particles in each frame.

While all of the theoretical flows are laminar in nature (although some contain features that attempt to model aspects of a turbulent flow), many of the practical issues affecting the performance of a PTV system, such as velocity shear and particle disappearance, are simulated, and allow realistic exploration of system performance for experimentally measured flows.

The user is able to adjust a number of parameters associated with the particles created through these models. Firstly a range of particle intensities or colours can be designated. Similarly a range of particle sizes are possible. Once a particle is allocated a colour/intensity and a size it retains these values throughout its lifetime in the flow.

These theoretical models are created using specific free processes that can be accessed in the process menu of the application view.

Eight theoretical models are available and these are discussed in the following sections. Full details for creating records from theoretical flow models can be found in the *User's Guide: Core Objects* manual.

6.3.1 Cellular model (2D)

This model creates a laminar cellular flow specified by a streamfunction of the following form

$$\psi = A \sin\left(\frac{n\pi x}{w}\right) \sin\left(\frac{m\pi y}{h}\right) \quad (6.1)$$

Here w and h are the width and height of the flow in mm. A , n and m are constants set by the user – note that m and n are integer values, and A is in mm²/s. The frame edge is a streamline so no particles flow out of the frame.

6.3.2 Irregular cellular model (2D)

This second model is a somewhat more complex version of the cellular model. Two different cells are superimposed on a uniform flow in the x direction. In addition the cells have an oscillatory time dependence.

$$\begin{aligned} \psi = & A_1 \sin\left(\frac{n_1\pi x}{w}\right) \sin\left(\frac{m_1\pi y}{h}\right) \cos(\omega_1 t) \\ & + A_2 \sin\left(\frac{n_2\pi x}{w}\right) \sin\left(\frac{m_2\pi y}{h}\right) \cos(\omega_2 t) + Uy \end{aligned} \quad (6.2)$$

Here w and h are the width and height of the flow in mm. A_1 , n_1 , m_1 , ω_1 , A_2 , n_2 , m_2 , ω_2 and U are constants set by the user – note that n_1 , m_1 , n_2 and m_2 are not necessarily integer values.

6.3.3 Multi-cellular model (2D)

This flow attempts to create an artificial flow including a range of length scales, each with an associated velocity scale. It is a crude attempt to produce a flow with homogeneous, mean shear free turbulent-like characteristics.

The flow is composed of a number of cell flows generated using the streamfunction

$$\psi = A \sin\left(\frac{n\pi x}{w}\right) \sin\left(\frac{m\pi y}{h}\right) \cos(\omega t) \quad (6.3)$$

The user specifies the constants A , n , m and ω for the largest cells, the number of cell sizes in the flow, N , and the ratio of the smallest to largest cell sizes, L_r . The model automatically generates the constants for the other cell sizes based on the length ratio and those for the large cells. The formulae used for the i^{th} cell size (1 being the smallest) are

$$\begin{aligned} z_i &= (L_r)^{N-i/N-1} \\ A_i &= A (z_i)^{4/3} \\ n_i &= \frac{n}{z_i} \\ m_i &= \frac{m}{z_i} \\ \omega_i &= \omega (z_i)^{-2/3} \end{aligned} \quad (6.4)$$

The streamfunctions for the N cells are superimposed to produce the flow.

6.3.4 Turbulent shear model (2D)

This theoretical model creates a flow with mean shear together with superimposed velocity fluctuations. The velocity field is chosen to ensure that continuity is satisfied.

$$u = U_{mean} + U_{top} \left(\frac{y}{h}\right)^{1/n} + u' \sin(\omega t) \sin\left(\frac{ky}{h}\right) \exp\left(\frac{-ry}{h}\right) \quad (6.5)$$

$$v = -v' \sin(\omega t) \quad (6.6)$$

Here h is the frame height in mm. U_{mean} , U_{top} , n , u' , k , r , v' and ω are constants set by the user.

6.3.5 Sediment motion model (2D)

This model attempts to provide a simulated environment where the moving particles are sediment grains on the bed of some flow. In this scenario there is no coherent velocity field in either space or time. Instead the user specifies the percentage of the particles that move during each time step together with information about how far these particles move on average in each direction. The actual particle displacements are randomly distributed around these mean values within a confined range specified by a spread parameter.

6.3.6 Custom formula model (2D, 3D surface, & 3D)

The final model is the most flexible, and is available for creating both two-dimensional, three-dimensional surface, and three-dimensional velocity fields. The u, v and possibly z velocity components are specified by the user through mathematical formulae that can include a range of functions of x, y, possibly z and t. Thus a simulated jet flow could be constructed with a mean Gaussian profile.

6.4 Particle record creation from text file

Particle records can be created manually by the user. The data required to construct these records are provided in space delimited text files and the commands to effect this construction are accessible in the Particle menu of the main application window. For particle 2D records a number of features are not provided. Pulsed records are not supported. Nor are particle statistics such as aspect ratio, eccentricity etc. The format of the text files can be found in section 2.2.1 of the User's Guide: Core Objects manual.

6.5 Segmentation

Matching particles in a particle record can be a computationally very expensive operation. Often much of that computational expense is due to searching particle frames for particles that meet certain criteria. For example, in the particle matching process normally only a small subset of the particles in a frame are considered as candidate

matches for a particle in a previous frame – typically those that are close to the location of that particle. However, this will typically require all particles in a particle frame to be processed in order to find those meeting the closeness criterion.

Segmentation of a particle record (and all of its frames) is designed to provide significant computational efficiencies at the cost of a small increase, typically 20%, in memory usage. Segmentation partitions each particle frame into a rectangular patchwork of segments, the number of rows, columns and perhaps planes (in the 3D case) of which are specified by the user. Each segment records the particles that lie within its bounds. It is not unusual that partitioning leads to speed improvements in a number of particle tracking algorithms by more than an order of magnitude.

The user can choose whether to use partitioning, but unless memory is precious it is highly recommended that they do so, and it is the default option for all new particle records.

6.6 Processes

In this section we provide an introduction to the various processes that manipulate particle records, some of which are unique to two-dimensional particle records. A number of these will be described in further detail in later chapters and information about specific parameter choices can be found in the *User's Guide: Core Objects* manual.

6.6.1 PTV analysis pipeline (2D & 3D)

A PTV analysis pipeline is a sequence of PTV analyses that attempt to match the particles stored in the particle record. Each analysis is executed completely before the next in the sequence is begun. Each analysis object has a range of parameters that can be set by the user and these are discussed in more detail in chapter 7.

The PTV analysis pipeline does not create a new particle record. Instead it simply manipulates the particle record that it is processing, allocating matches between particles. If the user wishes to retain a copy of the particle record before the matches are computed then they should do so manually.

6.6.2 Create velocity field (2D & 3D)

The create velocity field process interpolates an Eulerian velocity field onto a rectangular grid in three or four dimensions (space and time) using the particle-based velocities available from the particle record. This process, together with the entity that is created, a **velocity field time series**, will be discussed in chapter 11.

6.6.3 Create density field (2D)

A rather less common property of particle 2D records that might be of interest to the user is the density of particles within each frame. In this case the particle data is providing information about the distribution of particles in the flow and the particles are acting in some sense as a tracer. In oil plume experiments carried out at the University of Canterbury the distribution of oil droplets were of interest to the experimentalist. A **density field time series** records this particle density information on a three dimensional rectangular grid, and is generated by a create density field process.

Density field time series and the process that creates them will be discussed in chapter 13.

6.6.4 Create shape field (2D)

Shape field time series are very similar to density field time series in that particle information is interpolated onto a rectangular grid. A shape field represents some aspect of particle geometry, and the user can choose between diameter, area and aspect ratio, provided the aspect ratio information is available for the particles in the record (i.e. shape information was stored during the identification process). Shape fields are likely to be useful when the particles have some intrinsic property of interest. For example at low Reynolds numbers oil droplets discharging from an orifice tend to have different dimensions and shape in different regions of the plume.

Shape field time series and the process that creates them will be discussed in chapter 15.

6.6.5 Create displacement field (2D, 3D surface, & 3D)

Displacement field time series are actually very similar to velocity fields. Particle displacements, not velocities, are interpolated onto a rectangular grid. Both the displacement field and its gradients are produced and these gradient fields enable a number of strain fields to be computed through the calculators provided. One important difference between the displacement field and the other fields described is the ability for the user to choose to associate the displacement data with the material and not the physical location in space. Thus the strains associated with a particular part of the material being observed is stored in the location where that part of the material resided in the first frame of the video record. Thus, even though the material deforms during a material test all information is stored at the grid points of the original image.

Displacement field time series and the processes that create them will be discussed in chapter 15.

6.6.6 Create Lagrangian path field (2D & 3D)

The field time series that have been discussed so far are all Eulerian in nature. In other words the fields are defined by their space and time coordinates. One of the advantages of PTV over PIV is that particle specific information is available. Thus Lagrangian flow data can be extracted from a particle record that has particle matches. A Lagrangian 2D or 3D path field is an object that stores particle path data instead of Eulerian field data.

Lagrangian 2D and 3D path fields and the processes that create them will be discussed in chapter 16.

6.6.7 Transform pipeline (2D & 3D)

A transform pipeline comprises a sequence of transforms that operate on a particle record. The user can choose to operate directly on the particle record itself using the transform, or to create a copy to transform, thus leading to the creation of a new particle record. Currently ten types of transforms are available. These are:

Linear (3D)

This transform allows the user to scale and translate the spatial coordinates and scale the temporal coordinate.

Rotate (3D)

This transform rotates the coordinate axes through 3 angles, specified by the user, in sequential fashion. First the rotation about the x axis, then that about the y axis and finally that about the z axis.

Camera adjustment (2D)

The camera employed to capture the original particle images may not have been perfectly aligned with the flow, or the ideal coordinate directions. This transform allows the camera location to be translated and rotated about a user selectable centre.

Barrel pincushion distortion (2D)

Wide angle lenses invariably suffer some form of barrel or pincushion distortion. This distortion can be removed from the original images using a barrel pincushion filter, or it can be removed once a particle record has been created using this transform. The parameters used in this transform match those used in the corresponding filter.

Swap x (2D and 3D)

This transform inverts the frames in the record in the x direction. Thus the left hand side of each frame becomes the right hand side and vice versa. All particle coordinates are adjusted accordingly.

Swap y (2D and 3D)

This transform inverts the frames in the record in the y direction so that the top of the frame becomes the bottom and vice versa. All particle coordinates are adjusted accordingly.

Swap z (3D)

This transform inverts the frames in the record in the z direction so that the back of the frame becomes the front and vice versa. All particle coordinates are adjusted accordingly.

Constant velocity (2D and 3D)

This transform applies a constant velocity to the frames within the particle record. The velocity specified by the user is interpreted as the velocity that is to be applied to the camera capturing the images and thus the frames themselves are moved with the negative of the specified velocity. The result of the transform is to produce elongated frames in the record.

Remove camera movement (2D and 3D)

It is possible in some experimental configurations that the image capture device suffers from unwanted movement. For example a camera mounted on a moving gantry may suffer unwanted vibration. This transform attempts to remove this unwanted camera motion from the particle record by translating the particle locations in each frame based on an estimate of the camera motion. More details can be found in the *User's Guide: Core objects* manual.

Map coordinates (2D)

This transform can be thought of as a generalized coordinate transformation of which the removal of lens distortion is a particular special case. We have found it of particular value for situations where the camera is not aligned normal to the plane that contains the particle motion. More details can be found in the *User's Guide: Core objects* manual.

Transform coordinates (2D)

This transform has much in common with the remove camera movement transform. It uses a two dimensional transformation to adjust particle coordinates. The prime difference between the two transforms is that this transform includes both time-dependent translations and rotations of the particles, whereas the remove camera movement transform includes only the former. Specific free processes are available to generate the rotations and translations used in this transform. More details can be found in the *User's Guide: Core objects* manual.

6.6.8 Particle filter pipeline (2D)

A particle filter processes frames in a particle record in much the same way an image filter filters frames in an image sequence. Specifically the particle filter deletes particles within each frame based on its particular algorithm. Such a tool can be useful in eliminating spurious particles from a particle record before it is used to generate an Eulerian or Lagrangian field.

A particle filter pipeline comprises a sequence of filters that operate on a particle record, and it behaves in the same way as all pipeline-based processes. Currently only one type of filter is available. It is:

Delete tracked regions

This filter deletes particles that lie within a set of user-defined regions that move through time.

6.6.9 Other processes

One other process is defined for particle records. The save particle record process, like the save process for other entities, is useful within a process pipeline when an intermediate result is to be retained.

The input to a save particle record process is a particle record, and the output is the same particle record.

6.7 Views

A particle record provides a number of core views as well as a number of ancillary views. These views are designed to give the user access to different aspects of the record. The primary views will be discussed in some detail while the ancillary views will be mentioned briefly.

6.7.1 Particle view (2D & 3D)

The particle view is by far the most useful particle record view, and because of this it is the default view for a particle record.

The particle view provides a video simulation of the record. Frames within the record can be viewed individually or together. The record can be traversed forward or backward one frame at a time, or reviewed as an animated sequence in real time, slow motion or fast-forward. Viewing several frames at the same time can provide the user with significant insight into the flow structure. In many ways this feature simulates a long exposure photograph that clearly displays fluid path-lines in the flow.

Note that the particle view for a particle 2D record provides a two-dimensional perspective of the flow while the particle view for a particle 3D record provides a pseudo three-dimensional perspective with the ability to move the viewing location to explore the flow.

Information from the matching process can be superimposed on the particles themselves. In the case of particle records created from theoretical models this information also includes data about the correct matches, incorrect matches and so on.

For particle 2D records particles within the view can be displayed as either point particles or as particles representative of their actual shape. If detailed shape information is not available the particles will be drawn as circles of an equivalent diameter to their counterparts in the original images. However if this information is available the particles are drawn as ellipses with the principal axes angled correctly. For PTV data where particle shape information is important this feature can provide the user with a strong sense of the development of these shape characteristics as a particle moves. If the particle's size, for example, changes dramatically along a calculated path then it is likely that the matching algorithm is not performing correctly.

This view also provides the user with a way to remove extraneous particles from a flow and to delete undesirable matches or unmatched particles. The processing of real images may sometimes identify particles that don't actually exist within the flow. These normally arise due to light reflections off solid boundaries or a free surface, and their presence can adversely affect the matching process.

In PTV work the particle view is a critical tool for determining the quality of the particle sets obtained from the particle identification process, and also the quality of the matches produced by the application of a PTV analysis pipeline to the particle record.

6.7.2 Details view (2D & 3D)

The details view gives the user access to basic parameters associated with the particle record, such as the extent of its physical domain, the time step between frames and so on. Some of these parameters are fixed and cannot be changed but others are editable. The details view also provides the user with a way of interrogating the frames within the record. The user can inspect the particles within the frame together with all of their specific details, and look at the matches currently made to particles in that frame.

6.7.3 Process view (2D & 3D)

The process view for all entities has the same characteristics as the process view for an image sequence. The only difference between these views is that when new processes are created the list of primary and secondary processes will be entity specific. The primary processes for a particle record have been discussed in section 6.6.

6.7.4 Match summary view (2D & 3D)

The fourth primary view of a particle record is the match summary view. This view presents a table listing the frames in the record together with the number of particles in the frame and the number of matches that have been made. For a theoretically derived record this will also include information about the correct matches, and the number of errors made by the PTV analysis. In a separate table details of the particle paths identified by the PTV analysis are summarized. The number of paths of each length, up to the total number of frames in the record, is documented and the average path length computed.

This view also plays an important role in determining the quality of any PTV data. High matching percentages and long paths are most desirable.

6.7.5 Diagnostic view (2D & 3D)

The fifth primary view of a particle record is the diagnostic view. It is common that a user will wish explore a number of different PTV strategies when creating the velocities or displacements for a particle record. As part of that process it is important to be able to understand

the impact different choices in the PTV analyses have on the particle matches and hence on the particle velocities or displacements. This view provides the user with a platform to gain this understanding. Particle matching and particle velocity statistics can be computed and viewed for two particle records with identical particle sets. These are presented in tabular and graphical form (2D only) and give insights for individual frames in the particle record as well as for particular regions in the flow (2D only).

6.7.6 Comparison view (2D & 3D)

The sixth primary view of a particle record is the comparison view. The particle identification process for three-dimensional flows in particular is complex and achieved, as discussed above, through processes operating on a particle 2D record bundle. The user would like a way in which to reassure himself/herself that the particle identification is not sensitive to minor changes in the parameters used in these processes. The comparison view provides them with such a tool. The user can select another particle record, the comparison record, and analyse the commonalities and differences between the two particles fields in the two records. This view is available for both particle 2D records and particle 3D records but it is most useful for the latter.

6.7.7 Ancillary views

Both two-dimensional and three-dimensional particle records have a number of ancillary views. These views are accessed via the particle view and provide the user with more detailed information and particle matching and particle information for some sub-region of the flow domain. Details regarding these views can be found in the *User's Guide: Core objects* manual.

6.8 Summary

This chapter has provided an overview of the particle 2D record and particle 3D record entities. The processes defined for particle records have been discussed and the views associated with them have been introduced.

7 PTV Analysis

7.1 Overview

The core of a PTV system is its ability to match particles from frame to frame in a particle record. Without this ability the information obtained from the video record is qualitative in nature, providing only a broad-brush picture of the flow structures and their time dependence.

In this section we will look at the particle matching process in some detail. This process will be called by a number of synonymous names, such as PTV analysis, or particle matching process or procedure. It is common to both two- and three-dimensional particle records.

In addition, a process by which particles identified in multiple perspectives can be combined to produce three-dimensional particle fields can be couched in similar terms to that associated with particle tracking and this process will be discussed in chapter 9.

7.2 Optimisation process

To begin we will consider the theoretical problem of matching particles in one frame to those in another. Later we will talk specifically about *Streams*' implementation of this process.

The particle matching problem, whether between frames or between perspectives in a particle 2D record bundle, can be considered as a general optimisation problem, known as the **assignment problem**. In the assignment problem two groups of objects are defined. The aim of the problem is to assign objects in the first of these groups to those in the second based on costs associated with each assignment. The optimal solution is obtained when the total cost of the assignments is minimised.

Figure 7.1 illustrates a simple example of the assignment problem and its optimal solution.

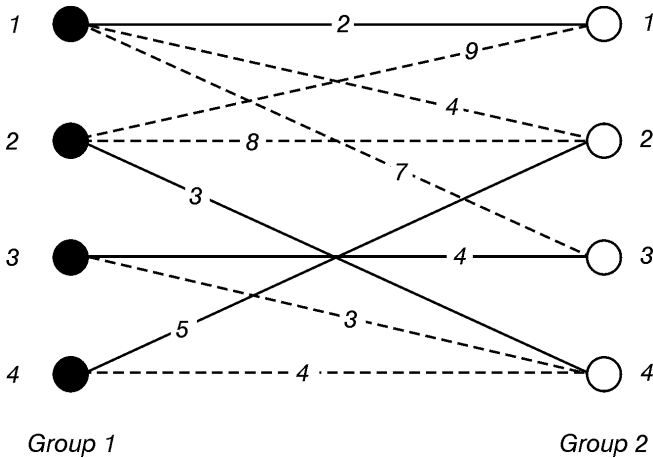


Figure 7.1 A simple example of an assignment problem. There are four objects in each of the groups. The lines indicate the possible assignments and the associated costs, c_{ij} , associated with assigning object i from group 1 to object j from group 2. The optimal solution is obtained when the total allocation cost is minimal. For this problem the optimal assignments are illustrated with bold lines.

The particle-matching problem is clearly of the assignment type. Particles in the first frame comprise the first group of objects, while those in the second frame comprise the second. Costs need to be assigned to each of the prospective matches between a frame one particle and a frame two particle. This can be done in a multitude of ways, and we will discuss a number of such costing strategies in chapter 8.

Even so, the particle-matching problem does not fit the assignment model exactly. The standard assignment problem requires the same number of objects in each of the two groups, so that each object in the first group is assigned to an object in the second group, and that the cost associated with each assignment is integral.

The number of particles present in each of two subsequent frames is nearly always different, and integral costs are unlikely to occur. Therefore the problem is a somewhat modified assignment problem.

A number of solution techniques are available for solving the assignment problem. The most general of these is a linear

programming technique known as the **simplex method**, although it is known to be computationally inefficient. However, because of its special structure (the assignment problem is a very specific linear programming problem) more specialised techniques have been developed. The currently favoured technique is known as the **auction algorithm**. This algorithm can be adapted to the modified assignment problem that corresponds to the particle-matching problem, and this be discussed in broad terms in the next section.

7.2.1 The auction algorithm

The auction algorithm, in the context of the particle-matching process, can be understood in the following simple terms (see Bertsekas 1992). The particles in the first frame are referred to as bidders in an auction, while the particles in the second frame are the objects for which the bidders are bidding. Each object carries with it a benefit for some of the bidders in the auction. A high benefit, corresponding to a low cost, means that a bidder desires the object highly. This benefit is defined to be the maximum matching cost (MMC) less the cost, where the MMC is defined in the next section.

The objective of the auction algorithm is to allocate objects to bidders so that the overall benefit accrued to the bidders is maximised. This is equivalent to allocating particles in frame 2 to particles in frame 1 such that the overall cost is minimised – ultimately the aim of the particle-matching algorithm.

Initially, all bidders are placed in a pool of unallocated bidders, and the auction is ready to start. All objects start with a price of 0, but as bidders bid for them, their prices rise. Unallocated bidders bid for objects if they can find one for which the net benefit for them is greater than 0. The net benefit is defined as the object's benefit less the object's current price. In *Streams* each unallocated bidder is considered in turn. If an object with a positive net benefit exists for that bidder then the bidder bids for the object with maximum net benefit. The bid increment is based on the difference between the net benefit of the most desirable object and the net benefit of the next most desirable object. This bid increment is added to the price of the most desirable object and this object is allocated to the current bidder. If this object was previously allocated to another bidder this bidder returns to the unallocated pool.

If a bidder in the unallocated pool cannot find an object that has a positive net benefit then they drop out of the auction and remain without an object. This corresponds to a particle in frame 1 being left unmatched.

The algorithm sweeps through the list of unallocated bidders repeatedly until all unallocated bidders have dropped out of the auction, or they have been allocated an object.

The algorithm just described is called the **naïve auction algorithm** because it may fail to complete. The conditions for this happening are easily seen. If a number of bidders have two or more objects with the same positive net benefit, then the naïve auction algorithm requires that the bid increment must be 0. Therefore two or more bidders can repeatedly swap objects without the prices for these objects increasing. A modified auction algorithm can be developed to overcome this circumstance although it is not clear how it can be implemented in the particle tracking problem where the cost (and hence the benefit) associated with an object can take on any real value.

In *Streams* a modified naïve auction algorithm is implemented. To avoid non-convergence the user specifies the maximum number of sweeps that are made through the list of unallocated bidders (the default is 1000). It has been found that when meaningful costings are used, convergence is not an issue. In other cases, only a few particles are affected, and the final allocations of these are generally acceptable.

As the optimisation process proceeds the user is informed of the number of bids made during the auction as well the number of sweeps taken, and the number of bidders that were left unallocated.

7.3 PTV analysis

Before discussing the specific costing strategies implemented in *Streams* we will introduce an overview of the particle matching process.

All of the information needed to match particles in a particle record is defined in a **PTV analysis** object. The user can create a PTV analysis object as part of a PTV analysis pipeline, which is simply a sequence of these analysis objects. Note that PTV analysis objects

and pipelines are different for two- three-dimensional particle records.

The pipeline analysis view provides mechanisms for the creation, deletion, editing, resetting, clearance and execution of PTV analysis objects. The user is free to define as many of these analysis objects in a pipeline as he or she wishes.

A PTV analysis object works through the sequence of frames in the particle record in pairs, attempting to calculate optimal matches for the particles in the first of the frames based on the specified costings. Note that for a pulsed clip *Streams* only works with the pairs of frames that are temporally close together. No attempt is made to match particles between frames in different pairs.

The user can choose to limit the analysis to a subset of the frames if they wish. In addition, the user can instruct the analysis object to start the analysis at the first frame and continue forward to the last frame, or alternatively to start the analysis at the last frame and work backwards to the first. The analysis in each direction is the same, except of course, for the fact that in the second circumstance time has been reversed.

While the particle matching process between two frames is a linear programming problem, the overall problem, matching particles in all frames in a record, will be non-linear. Some of the costing strategies discussed in chapter 8 rely on matches in subsequent or previous frames, and therefore, as the analysis continues, and matches change due to the optimisation process, previously allocated matches may become non-optimal. To alleviate this problem somewhat the user can instruct the optimisation process to iterate through the frames a number of times. In doing this it is assumed that the solution will converge towards a global optimal solution. We provide no proof for whether this is the case or not, and it is quite possible that for some problems such an iterative technique does not converge to the desired solution. However, handling the full non-linear problem is computationally too expensive to be practical.

When more than one iteration through the frames is selected the user may choose to perform each iteration in the opposite direction to the one before.

Now let us consider the details of the optimisation process at the level of an individual frame. In the most general case, every particle

in the first frame could be potentially matched to any particle in the second. This level of generality is computationally prohibitive. Consider a record where each frame contains around 2000 particles (this is not unusual in practice). The number of possible matches is the product of the number of particles in each frame, giving rise, in this case, to potentially 4 million matches. Each of these matches must be costed. For some costing strategies, each cost may require tens of thousands of calculations. The end result is that simply calculating the costs for the particles in one frame may require tens of billions of calculations. This requirement is crippling for typical desktop computer systems.

To circumvent this problem a **search window** is defined. The search window is a rectangular region defined relative to a particle in the first frame of the two being considered. Only particles lying within this physical rectangle in the second frame are considered as possible matches to the particle in the first frame. By a judicious use of the search rectangle the number of candidate matches for each frame 1 particle may be reduced to a fraction of the total number of particles in the second frame. The search window can be defined so that the particle in the first frame does not even lie within it. This may be useful for flows that are essentially unidirectional.

To address problems that sometimes occur in experimental systems where particles do not move between frames, perhaps because they are attached to a flow domain boundary or the particle is in fact some other optical signature unassociated with the flow, a circular or spherical region, centred on the particle, can be excluded from the search window. Thus only particles that move a finite distance between frames are considered as potential matches.

Even the use of a search window does not completely reduce the computational burden as the system still must search all of the particles in the second frame to find those that lie within the search window. *Streams* provides an additional mechanism, known as **segmentation**, that significantly reduces this search time. Segmentation is discussed in section 6.6.

Once the set of possible candidate matches has been deduced for each particle in the first frame the analysis process calculates the costs of these matches using the costing strategies selected by the user. Using these costs the set of possible matches for each particle is pruned further. Each PTV analysis object defines a **maximum**

matching cost (MMC), set by the user. This cost serves as a cut-off between costs that are viewed as reasonable, and costs that are seen as unreasonable. Any match that has a cost exceeding the MMC is eliminated as being undesirable. While the search window is a crude method for reducing the number of possible candidate matches for a particle, the use of the MMC is a rather more refined method of achieving the same objective. Its setting is crucial to a successful analysis. If the MMC is set too low, then correct matches will be excluded, and at best these matches will be missed. If the MMC is set too large too many spurious candidate matches will be considered, costing computational time, but also allowing incorrect matches to be made for particles that perhaps have no correct match in the next frame. The selection of an appropriate MMC is a skill that comes with experience, although experimentation with the theoretical model flows is an effective way of evaluating various choices of MMC for different costing strategies.

Streams provides another technique for controlling the matching process. The user can define **regions** for both the first and second frame that can be used to exclude particles, contained within those regions, from the matching process.

Once the candidate matches, whose costs exceed the MMC, have been eliminated a final set of candidate matches for each particle in the first frame is available, and the auction algorithm, as discussed in section 7.2.1, can be executed for the pair of frames under consideration. Once this optimisation process is complete, the analysis moves to the next pair of frames, the first of which is the second frame from the previous pair. And so the process continues until all selected frames have been analysed.

As the matching process executes a **process monitor dialog box** is displayed, keeping the user informed of progress. The initiation and completion of each step in the process is logged, together with the time taken for it to execute. For the optimisation process details of the auction algorithm are recorded.

As has been alluded to earlier the computational requirement of this process can be prohibitive. Therefore it is essential that the user make sensible parameter selections in order to reduce the computational demand. At this stage the most important parameter in this regard is the search window. The smaller this window is the better.

7.4 Costings

Each costing employs a particular algorithm for calculating the cost of a match between two candidate particles. This algorithm usually involves a number of parameters that can be set by the user, and the performance of the costing can be strongly dependent on the parameter choices.

All costs in *Streams* are non-dimensional. This eliminates the problem of combining costs with different dimensions. In addition, each cost is non-dimensionalised in such a way that, if possible, it lies between 0 and 1. A cost of 0 corresponds to a highly desirable match, while a cost of 1 corresponds to the least desirable match. For some costings the cost can be guaranteed to lie within this range. For others this is not possible. The MMC will normally be selected to lie between 0 and 1 when a single costing strategy is used.

The key to generating the correct particle matches between frames is the generation of accurate costs. Clearly, if the correct match for every particle has the smallest cost of all the possible matches, and particles, that have no correct match in the next frame, have no candidate matches in the next frame, then the optimisation process will generate a perfect set of matches for every frame.

These fortuitous circumstances are, unfortunately, unlikely to occur in practice, and it is through the skill of the user that satisfactory sets of matches are achieved.

Here we will consider the components that affect the success of the PTV analysis and discuss the roles they play in the process.

Costing strategy

The role of a costing strategy is to generate costs in such a way that the correct match carries a cost substantially less than all incorrect matches. In general, this cannot be achieved for all particles in a frame, in which case the costing must at least try to avoid an incorrect match having a substantially lower cost than the correct one (in this way the optimisation process may still select the correct match as the other desirable incorrect matches may be correctly matched to other particles). If this cannot be achieved then the costing may be a poor choice for the particular flow. Note, most costings have a range of parameters that affect their performance,

and therefore a costing with one set of parameters may perform poorly, while the same costing with another set of parameters may perform well.

The costing also has the duty of ensuring that when a particle has no correct match in the next frame, no incorrect match is given a low cost. In fact it is most desirous that all incorrect matches have costs exceeding the MMC.

MMC

The role of the MMC is to eliminate all incorrect matches from consideration, leaving only correct matches as candidate matches. If the costing strategy is able to clearly delineate the correct and incorrect matches then such a goal may be achieved. In practice the MMC is likely to be used to eliminate most of the incorrect matches, and in the case where the correct match has a similar cost to some of the incorrect matches, it needs to be carefully chosen so as to not exclude the correct match altogether. When no correct match exists the role of the MMC is to eliminate all candidate matches.

Default Cost

Some costing strategies cannot calculate a cost in certain circumstances. A simple example is the recent velocity costing. This costing needs matching information from the previous, or next, frame in order to calculate its cost. If such information is not available then no cost can be evaluated. In such circumstances the costing must specify a **default cost** that will be used when an actual cost is unavailable. This default cost should be greater than the MMC so that a match with the default cost is never considered. The philosophy here is that no news is bad news. A particle that can't be identified as the correct match, or an incorrect match, must be treated as an incorrect one. The reason for this is that incorrect matches tend to degrade the computed velocity field considerably more than missed matches.

Our discussion up until this point has focussed on the idea that only one costing strategy is used at a time. A more general approach is to use several costings simultaneously. This adds to the complexity of the analysis considerably.

The first consideration is how the costs generated by each of the costings contribute to the overall cost. This is handled by allocating

each costing a **weight**. The calculated cost for each costing is multiplied by its weight, and the resulting costs are added together. Thus the user is at liberty to impose an order of importance on the chosen costing strategies.

If all costings chosen specify a default cost then it is important that the weighted sum of the default costs should exceed the MMC. However the choice of MMC, default costs and weights becomes more challenging when some of the costings have defaults and others do not.

Costings can also be combined in another way by specifying one or more costings as **default costings**. This is most clearly illustrated through an example. Suppose a PTV analysis object includes two costings, say A and B, and costing B is designated as a default costing. When the cost of a particular match is initially computed only costing A is used. However, if costing A is unable to compute a cost, instead of using its default cost, costing B is used to compute the cost. In this way costing B is used only as a fallback position, with costing A playing the dominant role in the matching process.

The experience of the *Streams*' designer is that a number of PTV analyses, each with a single, but different, costing is rather more manageable than a single PTV analysis with multiple costings. However, this conclusion needs further testing.

7.4.1 Costing Types

A number of different costings are available in *Streams*, some of which are common to both two- and three-dimensional particle records, and some of which are available only for two-dimensional systems. These costings can be classified into two groups based on whether they require matches between particles in order to calculate their cost.

State based costings only require information regarding the state of each particle in a frame. Such information includes the particle's location, size and intensity. **Matching based costings** use matches between particles to determine their costs. Therefore only state based costings can be used to initiate the particle tracking process, or at least, if multiple costings are to be used, then at least one of these must be state based. Once an initial set of matches has been calculated, matching based costings may be used.

Another important difference between the two costing types is that the cost associated with a particular match is invariant for a state-based strategy, while for a matching-based strategy the cost will depend on the current set of matches. For this reason the performance of a matching-based strategy may improve if it is iteratively applied to the sequence of frames.

In general, the matching based costings are more flexible than their state based cousins. This is due to the fact that they can use dynamic information such as particle velocities to help the matching process. However in some applications matching based costings may be of little value. Take for example the problem of measuring sediment motion on the bed of a channel. This motion is quite unlike a coherent velocity field where assumptions of continuity and smoothness of the velocity field can be made. For the sediment problem the motion of one particle has no bearing on the motion of one of its neighbours. In fact the motion of a particular particle in one frame implies nothing about its possible motion in the next. Therefore matching-based costings are worthless in such problems and resort must be made to the state based costings.

7.5 Process types

The focus of discussion up to this point has been the matching of particles between frames using the full optimization process. *Streams* provides the user with three alternative processes to the optimization process already described. It is important to note that the optimization discussed so far is referred to as global optimization in *Streams*.

7.5.1 Local optimisation

Unlike the global optimisation the local optimisation attempts to optimize at the particle level, instead of the frame level. To do this a particle in frame 1 is matched to a particle in frame 2 only if two criteria are met. The first is that the cost associated with the frame 2 particle is the smallest for the frame 1 particle (hence a particle will never be matched to a particle which is less than optimal from its perspective). The second is that there is no other frame 1 particle for which the frame 2 particle has a lower cost (hence the frame 2 particle must also have its optimal match). The local optimisation process will typically generate fewer matches than its global

counterpart but, provided an appropriate costing strategy is adopted, it is likely that these matches will be of higher quality. The user may wish to compare the results of these two processes to see how they compare.

7.5.2 Residual optimisation

Residual optimisation is a limited optimisation process that works only with particles that are currently unmatched. Effectively all matched particles are removed from the optimisation process and the defined analysis applied only to those that remain. This process can be useful in that it allows the user to avoid compromising a set of high quality matches while still attempting to improve the overall result of the matching process.

7.5.3 Clean up process

The purpose of a cleanup is to remove incorrect matches from the record by selecting one or more costing strategies that can be used to evaluate the validity of a match. If a match incurs a cost in excess of the MMC, then the cleanup removes that match. New matches are not created. Thus a cleanup performs a post-processing role, as is commonly found in most PIV systems.

Optimisation parameters, such as the search window, are not relevant to the cleanup process, as new candidate matches are not considered.

7.6 Summary

This chapter has provided an overview of a PTV analysis. Details about each step in the process can be obtained from the relevant chapters in the *User's Guide: Core objects* manual.

8 Costings

8.1 Overview

This chapter provides a detailed introduction to the costings available in *Streams*. Sections 8.2 and 8.3 will discuss the costings available for matching particles in 2D and 3D particle records. Many of these costings are common for the two record types with very minor modification due to the different dimensionalities. In section 8.4 a practical guide to the selection of costings is provided. Section 8.5 introduces the costings that are used in bundle matching analyses.

In the discussion in sections 8.2 and 8.3 we will refer to the two frames being analysed as frames 1 and 2, frame 1 being the frame whose particles are being matched. We will use p1 to refer to an arbitrary particle in frame 1, and p2 to refer to an arbitrary particle in frame 2.

8.2 State-based costings

There are eight state based costings. These are

- adjacency costing (see section 8.2.1)
- centre of mass costing (see section 8.2.2)
- colour costing (see section 8.2.3)
- correlation costing (see section 8.2.4)
- distance costing (see section 8.2.5)
- intensity costing (see section 8.2.6)
- pseudo-correlation costing (see section 8.2.7) and
- shape costing (see section 8.2.8)

8.2.1 Adjacency costing (2D & 3D)

The **adjacency costing** is designed to measure the degree to which the pattern of particles surrounding particle p1 corresponds to the pattern surrounding p2. It has much in common with the correlation costing but is computationally considerably less expensive.

The user specifies a rectangular region (2D or 3D) surrounding particle p1 and all particles lying within that region are found. The same is done for particle p2. All particles in the p1 rectangle are “displaced” by the vector $\mathbf{r}_{p2} - \mathbf{r}_{p1}$, where $\mathbf{r}_{p2}$ is the position of particle p2 and $\mathbf{r}_{p1}$ is the position of p1. The list of particles in the p2 rectangle is then searched for the particle that is closest (adjacent) to each of the displaced frame 1 particles. The costing is the sum of the distances between these closest matches normalised by the distance between p1 and p2. Thus

$$C_{12} = \frac{\sum_{i=1}^{n_1} |\mathbf{r}_i + \mathbf{r}_{p2} - \mathbf{r}_{p1} - \mathbf{r}_{a_i}|}{n_1 |\mathbf{r}_{p2} - \mathbf{r}_{p1}|} \quad (8.1)$$

- n_1 – the number of particles in the frame 1 rectangle that have closest neighbours in the frame 2 rectangle.
- $\mathbf{r}_i$ – the position of particle i in the frame 1 rectangle
- $\mathbf{r}_{a_i}$ – the position of the particle in the frame 2 rectangle that is closest to particle i in frame 1.

The user has a number of ways to customise this costing. The contribution of each frame 1 particle to the total cost may be weighted by the relative distance of the particle from p1. Closer particles have a stronger influence on the total cost. A limit can be placed on the maximum distance used in the cost calculation, and the user can choose whether each particle in frame 1 must have a unique partner in frame 2. The normalization distance can be the sum of the radii of the two particles instead of the distance between p1 and p2.

8.2.2 Centre of mass costing (2D & 3D)

The **centre of mass costing**, like the adjacency costing, attempts to compare particle patterns. It does this in a gross way by calculating the positions of the centres of mass of the particles in the rectangles (2D and 3D) centred on p1 and p2. In calculating the centre of mass the actual size of the particles is not taken into account – in other words all particles are assumed to be the same size. The distance between the centres of mass, normalised by the distance between p1 and p2 provides the cost. Thus

$$C_{12} = \frac{|\mathbf{r}_{cm1} - \mathbf{r}_{cm2}|}{|\mathbf{r}_{p2} - \mathbf{r}_{p1}|} \quad (8.2)$$

- $\mathbf{r}_{cm1}$ – the position of the centre of mass of the particles surrounding p1
- $\mathbf{r}_{cm2}$ – the position of the centre of mass of the particles surrounding p2.

8.2.3 Colour costing (2D & 3D)

The **colour costing** measures the difference in colour between two particles. The absolute difference in the intensity of each of three guns is computed and each of these is multiplied by a user-defined weight (the default being 1, and the weight being limited to the range 0-1). The cost is computed as the root-mean-square of these three differences, normalized by the maximum intensity of 255. Thus two particles with the same colour will produce a cost of 0, while a black particle and a white particle will produce the maximum cost of 1.0 if all colour weights are set to their default values.

8.2.4 Correlation costing (2D)

The **correlation costing** is based on the cross-correlation of the intensity field of sub-windows in the two frames. In this sense it is similar to the correlation used in a PIV system.

The user chooses a rectangular sub-window centred on p1 and overlays this sub-window on frame 2, centring it on p2. The cross-correlation of the intensities in the two sub-windows is calculated, and normalised by the intensity auto-correlations of the two sub-windows. This calculation provides a measure of the similarity of the particle patterns in the two frames, and hence how likely p1 and p2 are the same particle.

Thus the cost can be expressed as

$$C_{12} = 1 - \frac{\text{cross correlation}_{12}}{\sqrt{\text{auto}_1 \text{auto}_2}} \quad (8.3)$$

- $\text{cross correlation}_{12}$ – the cross correlation of intensities in the two rectangles excluding particles p1 and p2
- auto_1 – the auto-correlation of the frame 1 rectangle excluding p1

- $auto_2$ – the auto-correlation of the frame 2 rectangle excluding p2

This process differs from the PIV calculation in a number of ways. Firstly the calculation is performed to determine the “goodness” of the match between p1 and p2. It is not used to find the displacement of the sub-window that leads to the highest correlation. Secondly, particles p1 and p2 are excluded from the correlation calculation. A near perfect correlation between these two particles is guaranteed by the choice of sub-window displacement, and it is the correlation of the particles around them that determines whether they are the same particle. Thirdly, as the correlation is calculated using artificially created frames of pixels (unlike the PIV system that uses the original digital images) a variety of manipulations can be performed on the particles in the artificial frame. The most important of these is the enlargement or compression of the particles in the frame. This capability is found to provide a powerful enhancement to the performance of the correlation costing.

8.2.5 Distance costing (2D & 3D)

The **distance costing** is a conceptually simple costing and one that has been used since the earliest days of PTV. It is based on the distance separating the two particles for which the cost is being calculated. The closer the particles are to one another the lower the cost.

However the implementation of the distance costing in *Streams* is somewhat more sophisticated and powerful than this simple costing. The distance between the particles is now computed as the distance between p2 and the position that p1 would be at if it had travelled at a user-defined velocity. The x and y components of this velocity are specified by two mathematical formulae entered by the user. As a simple example consider particles in a uniform fluid flow. By specifying this flow field in the mathematical formulae the predicted location of p1 should coincide with the location of p2 in frame 2 if p1 and p2 are the same particle. More complex flows are easily accommodated, for example a Gaussian jet profile or a logarithmic boundary layer velocity distribution. While this specification of a velocity field a-priori might seem to present a circular argument – specify a velocity field in order to match particles that will enable you to calculate the velocity field – even a rough approximation to the velocity field of the actual flow in the distance costing will greatly enhance its performance.

In the 2D case the costing also allows the user to weight the errors in the x and y directions separately. The resulting distance is normalized by a reference length set by the user and then raised to a user specified power to compute the cost.

Thus the costing is

$$C_{12} = \left(\frac{d}{d_{ref}} \right)^{pow} \quad (8.4)$$

where

$$d = \sqrt{(x_2 - x_1^p)^2 + W_y (y_2 - y_1^p)^2} \quad 2D \quad (8.5)$$

or

$$d = \sqrt{(x_2 - x_1^p)^2 + (y_2 - y_1^p)^2 + (z_2 - z_1^p)^2} \quad 3D$$

- (x_2, y_2) or (x_2, y_2, z_2) – the position of p2
- (x_1^p, y_1^p) or (x_1^p, y_1^p, z_1^p) – the predicted position of p1 using the specified velocity field.
- d_{ref} – the reference length.
- pow – a user specified power that must be positive.
- W_y – a weighting for the y distance specified by the user (2D only).

8.2.6 Intensity costing (2D & 3D)

The **intensity costing** measures the change in intensity as a particle moves. It assumes that the intensity of a particle should change little between frames.

The cost is calculated by

$$C_{12} = \frac{|I_1 - I_2|}{255} \quad (8.6)$$

- I_1 – the intensity of p1 (on an 8-bit scale of 0-255)
- I_2 – the intensity of p2 (on an 8-bit scale of 0-255)

8.2.7 Pseudo-correlation costing (2D & 3D)

The **pseudo-correlation costing** is another costing that, like the adjacency costing, attempts to imitate the correlation costing in a computationally more efficient manner. The pseudo-correlation costing is based on the adjacency costing and differs from it only in the way in which the cost is calculated. Consider two particles that have been determined to be adjacent to one another in the sense used in the adjacency costing. If the correlation between these two particles was to be computed it would be equal to 1 if the centres of the two particles (assuming that they are circular or spherical) are coincident. If the two particles are positioned so that they do not overlap at all their correlation would be 0. Thus the pseudo-correlation attempts to mimic this behavior. The cost is given in equation (8.7).

$$C_{12} = \frac{1}{n} \sum_{i=1}^n \frac{\min \{d_i, r_i + r_i'\}}{r_i + r_i'} \quad (8.7)$$

- d_i – the distance between the centre of particle i in frame 1 and its adjacent particle in frame 2. There are n particles in the rectangle in frame 1 surrounding $p1$. If particle i has no adjacent particle in frame 2 then d_i is set to infinity.
- r_i – the radius of particle i in frame 1.
- r_i' – the radius of the particle in frame 2 adjacent to particle i in frame 1.

If all adjacent particle pairs have coincident centres the cost in equation 8.7 will be 0. If none of the particle pairs actually touch each other the cost will be 1. Thus the parallels between the correlation and pseudo-correlation costings are clear. Like the correlation costing the user can choose to amplify the radius of the particles (which tends to decrease the cost) and this tool is found to be very effective at improving the performance of the costing. The primary difference between the two costings is the computational times – the pseudo-correlation costing typically being somewhat faster. The pseudo-correlation costing, like the adjacency costing, can also weight the contributions different particles make to the calculation of the cost based on their distance from $p1$.

8.2.8 Shape costing (2D)

The **shape costing** is based on changes to the particle's shape parameters. This is only available for two-dimensional particles. Four shape parameters are available – the effective diameter, the aspect ratio, the angle of the major principal axis to the horizontal, and the eccentricity. Each of these parameters is converted into a normalized difference as defined in equation 8.8.

$$\Delta s = \frac{|s_1 - s_2|}{\max\{s_1, s_2\}} \quad (8.8)$$

- s_1 – the shape parameter of p1
- s_2 – the shape parameter of p2

These four normalized shape differences can be combined in two ways to produce the cost. The first is to take a weighted sum, where the weights are specified by the user. The second is to utilise a mathematical formula that uses the four normalized differences as variables. The user can choose which of these methods is used.

8.3 Matching-based costings

There are ten matching based costings. These are

- direction velocity costing (see section 8.3.1)
- least squares velocity costing (see section 8.3.2)
- local velocity costing (see section 8.3.3)
- magnitude velocity costing (see section 8.3.4)
- path length costing (see section 8.3.5)
- polynomial velocity costing (see section 8.3.6)
- recent acceleration costing (see section 8.3.7)
- recent velocity costing (see section 8.3.8)
- space average acceleration costing (see section 8.3.9) and
- time average velocity costing (see section 8.3.10)

All of these costings, except the path length costing, require a default cost.

8.3.1 Direction velocity costing (2D & 3D)

The **distance velocity costing** uses only the change in direction of a particle path to compute the cost. The change in velocity magnitude is not taken into account. In this way it complements the magnitude velocity costing. The costing uses the match to p1 in the previous frame to frame 1, and the match to p2 in the frame subsequent to frame 2 to determine the particle displacements. The angles between these displacements and the displacement from p1 to p2 are computed, averaged and normalized by a reference angle specified by the user. This normalized angle is the cost. Thus if the particle trajectory, assuming p1 is matched to p2, is a straight line then the cost will be zero.

The user may choose not to use both estimates of the particle displacement but instead only to use the displacement computed from the match to p1 in the previous frame.

This costing has the potential to be employed productively in applications where particle displacements are not smooth in terms of magnitude, but are primarily unidirectional. A good example might be in the monotonic loading regime of a structural element close to failure.

8.3.2 Least squares velocity costing (2D & 3D)

The **least squares velocity costing** is a powerful costing strategy that uses estimates of particle velocities in the neighbourhood of p1 to predict its likely location in the next frame. A two-dimensional polynomial, fitted in a least squares sense, is computed for each velocity component, based on the velocity estimates of a number of particles that are p1's nearest neighbours in frame 1. The user selects the order of the polynomial to be used - up to a maximum of 4 - and the number of neighbours used to compute the least squares estimates. If the user selects an insufficient number of nearest neighbours, or the number of velocity estimates available (due to neighbouring particles being unmatched) are insufficient, for the order selected, *Streams* automatically reduces the polynomial order until the polynomials can be computed. The polynomial for each velocity component is evaluated at the position of p1 and this

estimated velocity vector is used to predict the location of p1 in frame 2.

The cost associated with a particular match is calculated from

$$C_{12} = \frac{|\mathbf{r}_{pred} - \mathbf{r}_{p2}|}{|\mathbf{r}_{pred} - \mathbf{r}_{p1}|} \quad (8.9)$$

- $\mathbf{r}_{pred}$ – is the predicted location of p1 in frame 2 using the velocity estimate described above
- $\mathbf{r}_{p2}$ – the location of p2 in frame 2.
- $\mathbf{r}_{p1}$ – the location of p1 in frame 1.

The user has a number of parameters that can modify the performance of this costing. To ensure that only particles that are close to the particle of interest are used in the least squares velocity estimate a maximum distance can be specified. Any of the nearest neighbours lying further than this distance from p1 are excluded from the least squares calculations. Particles lying far from p1 are also likely to have velocities that are less representative of the velocity of p1 than those close to it. The contribution of the velocity estimate of each surrounding particle is weighted in the least squares computations, based on its distance from p1. The weight factor decreases linearly from 1 for a particle at the same location as p1, to a value, specified by the user, for the furthest particle from p1.

The magnitude of the predicted displacement of p1 from frame 1 to frame 2 is used to normalise the cost. If this displacement falls below a default length, specified by the user, then the default length is used in the normalisation instead.

8.3.3 Local velocity costing (2D & 3D)

The **local velocity costing** is similar to the least squares velocity costing in that estimates of particle velocities in the neighbourhood of p1 are used to predict its likely location in the next frame. The neighbourhood can be defined in two ways. In the first all particles, in a user specified rectangle (2D or 3D) centred on p1, that have a velocity estimate available due to a particle match in the previous or next frame, are used to calculate a representative velocity for p1. In the second a specified number of nearest neighbor particles to p1, that have a velocity estimate available, are used to estimate the

velocity of p_1 . In this case a rectangle centred on p_1 is used to limit the distance the nearest neighbours can be from p_1 .

In both cases the representative velocity can be either the average of these particle velocities or their median. When an average of the velocities of the surrounding particles is used the contribution of each velocity to the average is determined by a weight factor. The weight factor decreases linearly from 1 for a particle at the same location as p_1 , to 0 for a particle located in one of the corners of the rectangle (whether nearest neighbours are used or not).

Equation 8.9 is again used to calculate the cost of a match, and a default normalization length, as described in section 8.3.1, is defined.

8.3.4 Magnitude velocity costing (2D & 3D)

The **magnitude velocity costing** attempts to minimise the change in velocity magnitude between frames. The difference between the velocity magnitude of p_1 , based on its match to p_2 , is compared to a second velocity magnitude estimate. This velocity magnitude may be calculated using a match to p_1 in the previous frame, a match to p_2 in the next frame, or both. The difference between these two velocity magnitude estimates is normalised by their average to yield the cost.

Thus

$$C_{12} = \frac{\left| \left| \mathbf{r}_{p_2} - \mathbf{r}_{p_1} \right| - |\Delta \mathbf{r}| \right|}{\frac{1}{2} \left(\left| \mathbf{r}_{p_2} - \mathbf{r}_{p_1} \right| + |\Delta \mathbf{r}| \right)} \quad (8.10)$$

- $\Delta \mathbf{r}$ – the estimate of the displacement based on previous or past matches.
- $\mathbf{r}_{p_2}$ – the location of p_2 in frame 2.
- $\mathbf{r}_{p_1}$ – the location of p_1 in frame 1.

8.3.5 Path length costing (2D & 3D)

The **path length costing** is designed to penalise short paths as these are generally seen as unreliable. The cost is given by

$$C_{12} = \frac{1}{\text{particles in path}} \quad (8.11)$$

- *particles in path* – is the number of particles in the path created by matching p1 to p2.

8.3.6 Polynomial velocity costing (2D & 3D)

The **polynomial velocity costing** is similar to the other velocity based costings. A particle path is constructed using p1, p2 and all of the particles linked to p1 in previous frames, and p2 in subsequent frames. A polynomial is fitted, in a least squares sense, to the x, y and possibly z coordinates of particles in the path, as functions of the frame number. The cost is calculated from the difference between the actual particle locations and their locations as predicted by the fitted polynomial. It is assumed that the correct particle path should be relatively smooth and therefore even a relatively low order polynomial should be able to accurately represent the path.

The cost is given by

$$C_{12} = \frac{\text{fit error}}{\text{average displacement}} \quad (8.12)$$

- *fit error* – the error between the positions of the particles in the path, and the positions of the particles predicted by the polynomial path fit. It can be defined in two ways. Either it is the total fit error based on all particles in the path (this is the root mean square error of the x and y coordinate fits for each particle), or it is a local error based solely on the fit error for particles p1 and p2. The user chooses which of these definitions to use.
- *average displacement* – the average displacement of the particle from frame to frame is used to normalise the cost.

The order of the polynomial used to fit the particle path is given by

$$\text{order} = \frac{\text{pathlength}}{2} \quad (8.13)$$

8.3.7 Recent acceleration costing (2D & 3D)

The **recent acceleration costing** is similar to the recent velocity costing described in the next section. It assumes that accelerations in the flow are roughly constant. The acceleration of the particle, calculated on the basis of a match between p1 and p2 is compared with an acceleration based on particles linked to p1 in previous frames, particles linked to p2 in subsequent frames, or an average of both if these are available.

Thus the cost is given by

$$C_{12} = \frac{|a_{pred} - a_{est}|}{\frac{1}{2}|a_{pred} + a_{est}|} \quad (8.14)$$

- a_{pred} – the predicted acceleration based on the match between p1 and p2.
- a_{est} – the estimated acceleration as described above.

8.3.8 Recent velocity costing (2D & 3D)

The **recent velocity costing** strategy was one of the earliest ideas in PTV analysis. It assumes that the accelerations in the flow are small and therefore if the velocity of the particle is known for the current frame, then the location of the particle in the next frame can be estimated from this velocity.

The estimate for the velocity can be based on a match to p1 in the previous frame, a match to p2 in the next frame, or both if they are available.

The cost is calculated using equation 8.9 where the predicted position is based on the velocity estimate just described.

8.3.9 Space average acceleration costing (2D & 3D)

The **space average acceleration costing** is similar to the local velocity costing. Equation 8.14 is again used to calculate the cost but now the estimated acceleration is based on the average acceleration of the particles lying within a rectangle (2D or 3D) centred on p1.

The accelerations are weighted in the average calculation by the inverse of their distance from p1.

8.3.10 Time average velocity costing (2D & 3D)

The **time average velocity costing** is similar to the recent velocity costing. The difference between the two is the way in which the velocity of p1 is estimated. For the time average velocity costing the velocity is a weighted average of the velocity throughout p1's time history. A particle path is constructed using all of the particles linked to p1 in previous frames, and p2 in subsequent frames, and the velocities of the particles comprising this path are used in the weighted average.

8.4 Guidelines

The selection of the best costing strategy to use for a particular flow tends to be guided by experience. However, the applicability of most of the costings is relatively general so this section will provide some general advice on the strengths and weaknesses of the various costings. [Note: The author has attempted to develop a clear dimensionless framework that can be used to classify the performance of the various costing strategies. This has been done for a number, but not all, of the strategies available in *Streams* and more work is required to make this framework of practical value.]

8.4.1 State based costings

For sparsely seeded flows the distance costing performs well without the use of a user specified velocity. It is important to note that a sparsely seeded flow is defined to be one where the typical inter-particle spacing is large compared to the typical particle displacement between frames. This definition is most easily stated in non-dimensional form as follows

$$\rho_d = U \Delta t \sqrt{\rho} \quad (8.15)$$

- U – an estimate of the velocity magnitude for the flow.
- Δt – the time step between frames.
- ρ – the density of particles in the frame.

This type of flow is difficult to handle with the PIV method, but can arise easily in practice. An example is the tracking of a small number of surface drogues.

As the particle seeding density increases the performance of the distance costing begins to degrade unless the velocity formulae are invoked. For these flows the pseudo-correlation, and correlation (2D only) costings are excellent performers. These two costings are closely related and their performance is likely to be similar. However, the computational speed of the pseudo-correlation costing may make it the costing of choice. The performance of both of these costings is affected by the presence of strong shear in the flow. However this is also the case with a typical PIV system. This difficulty can be mitigated to some extent using the radius amplification that has been mentioned in the descriptions of the two costings.

The shape costing offers significant potential to complement other state-based costings in typical fluid flow analysis, and to play a primary role in problems where no coherent flow field exists – sediment transport for example. In most flow problems the size of a particle does not change significantly from frame to frame, and the aspect ratio, if this information is available, is also relatively invariant provided robust particle identification algorithms are used.

The centre of mass costing, while inferior to its cousin the adjacency costing, has been used with some success in our laboratory.

8.4.2 Matching based costings

Of the matching based costings the local velocity costing is the most flexible and useful. It is considerably less sensitive to incorrect matches than the recent velocity, polynomial velocity and time averaged velocity costings due to the fact that it is using velocities from a number of particles in its estimate of the velocity of p1. The least squares velocity costing is comparable in performance to the local velocity costing and may, in fact, perform better when velocity gradients are strong and the current matches are of high quality.

State based costings, even when performing well, will frequently generate some incorrect matches. The local velocity costing is

effective at removing these erroneous matches and replacing them with correct ones if they exist.

The other velocity-based costings can perform well when the current matches are almost error free. In this case these costings can help to extend particle paths and improve the overall percentage of correct matches.

Theoretical analyses suggest that the equivalent acceleration-based costings have the potential to perform as well if not better than the velocity-based costings. However, the practicalities of real experimental data ensure that the particle accelerations are considerably less accurate than the velocities due to the fact that they are obtained from higher derivatives of the displacement with respect to time. Therefore for most practical flows the acceleration-based costings are of limited use.

The path length velocity is of limited use by itself. However, it can be a simple way of ensuring that only long particle paths are used in the velocity field estimates.

It should be noted that in flows where the velocity field varies considerably between different regions, for example the flow inside and external to a turbulent jet, a strategy where the different regions are analysed separately using different costings is more likely to generate robust matches than analyzing the entire flow with a single costing.

8.4.3 Summary

The most useful costings are the distance, adjacency, pseudo-correlation, correlation, shape, local velocity and least square velocity costings, with the recent and polynomial velocity costings also having some degree of applicability. However each flow needs to be considered separately and some creative ways of using some of the other costings may be discovered.

We have not attempted here to tackle the more difficult problem of considering the performance of multiple costings employed simultaneously. Our practice has generally involved the sequential application of a number of costings - each subsequent costing building on the performance of those before.

8.5 Summary

This chapter has introduced the costings associated with the particle matching processes. An intimate knowledge of these costings together with some experience of their performance, either for real systems or artificially created ones, is crucial to the successful application of particle tracking technology.

9 Construction of Particle 3D Records

9.1 Overview

Particle field construction for three-dimensional particle fields is rather more complex than for two-dimensional fields. Simple particle identification processes are not available when two-dimensional imaging devices are used. In the future, when three-dimensional imaging becomes commonplace, this is likely to change.

Using current technology three dimensional particle motion is recorded by multiple cameras in order to construct the three dimensional particle field. Each camera records a two-dimensional perspective of the particle field, and through the combination of these perspectives the full three-dimensional field can be produced. Particle 2D record bundles are designed to support this construction process.

9.2 Particle 2D record bundles

Particle 2D record bundles comprise a number of **perspectives**, where each perspective includes a particle 2D record, created in the same way as a particle record captures two-dimensional particle motion and a **perspective camera**. In order to enable accurate construction of particles in 3D space the perspective camera must be carefully calibrated. The user is at liberty to undertake this calibration in any way that they wish, but *Streams* does include a default calibration process.

The default process uses the recorded image of a **calibration target** – a distribution of particles in three-dimensional space for which the user knows the particle locations to a high degree of accuracy. The image of this target is analysed by *Streams* to produce a field of two-dimensional particles using the particle identification processes described in chapter 5. The camera is then calibrated by determining the camera parameters (see below) that will most accurately reproduce the measured set of two-dimensional particles from the known target particles locations.

As described in above the calibration process determines a set of camera parameters that uniquely encapsulate the camera's physical and optical characteristics. In versions of *Streams* preceding version 3.02 a perspective camera was defined by parameters that corresponded to the actual physical and optical attributes of the camera. These parameters included the location of the camera in 3D space, its orientation, its zoom angle and its lens characteristics (i.e. parameters associated with lens distortion). Such a definition has a number of limitations. Firstly, the calibration of such a camera is problematic. The camera is almost overly specified due to the fact that the zoom angle of the camera, and the camera's distance from its subject, are almost degenerate. By this it is meant that if the camera is moved further from, or towards, the target, while at the same time the zoom angle is adjusted appropriately, the two dimensional image of the calibration target is almost unchanged. Secondly, the calibration is determined by a non-linear optimization process that does not possess a single minimum, and therefore the choice of the initial camera parameters to initiate the optimization is critical to its ability to locate the global minimum. And finally, such a parameterization of a camera's physical and optical characteristics may work adequately for a camera viewing particles in a single optical medium, or at least through multiple media under very strict geometric constraints – for example the camera may be located in air looking through a single, planar, transparent barrier at another fluid, and all three media have known optical characteristics – but for general configurations this characterization of the camera becomes extremely cumbersome and ultimately inadequate.

In *Streams* 3.02 a totally different approach to camera characterization was adopted. The camera is treated simply as a mapping device that can map three-dimensional coordinates onto two-dimensional space. The mapping is no longer specified through a set of physical and optical parameters but simply through two mathematical mapping functions, one for the x coordinate (in 2D space) and one for the y coordinate. Currently these mapping functions are represented by three-dimensional n th order polynomials, where n can take any value the user chooses, and these mapping functions are valid over a physical domain delineated by the physical space occupied by the calibration target. Outside this domain there is no guarantee that the mapping functions are valid.

This characterization of a camera has a number of advantages. Firstly, the calibration process now corresponds to a simple least

squares minimization problem whereby the polynomial coefficients are computed to be those that minimize the error between the locations of the actual particles and the mapped particles in two-dimensional space. The only constraint is that the number of target particles exceeds the number of polynomial coefficients. Thus the calibration is unique, and very fast to compute. Secondly, complex optical arrangements, for example viewing a liquid flow through a curved transparent wall, involve no more complexity in the calibration process than for a single optical media. The variations in refractive index are captured by the mapping functions and additional geometric and optical parameters need not be included in either the calibration process or the camera description. Our limited experience suggests that this mapping function approach is robust and generally outperforms the more traditional physical characterization of a perspective camera.

9.3 Particle construction

Streams provides two different methods for generating a three-dimensional particle field from a particle 2D record bundle. The first of these, the **bundle matching analysis** – see section 9.4 – is closely related to the algorithms used for matching particles between frames seen in chapters 7 and 8. The second, and more robust process, termed **predict and verify**, takes a somewhat different approach whereby two perspectives are used to estimate a set of potential 3D particles and the remaining perspectives are employed to verify the validity of these potential particles. The predict and verify technique is discussed in section 9.6.

In both approaches it is useful to be able to deduce the geometry of a light ray that passes through the particle domain (defined by the three-dimensional space covered by the particles in the calibration target) and that emanates from a particular particle in a perspective's particle 2D record. By assuming that there are no variations in optical properties within the particle domain this light ray is a straight-line, and using an optimisation process based on the mapping functions for the camera the points at which the light ray enters and exits the particle domain can be computed. It should be noted that the computation of these entry and exit points can be problematic when the light ray enters or exits the domain very close to one of the domain's boundaries, and in these very rare cases the construction of this light ray may fail.

9.4 Bundle matching analysis

The bundle matching analysis approach closely resembles the optimization problems discussed in chapter 7. It is an assignment problem where the assignment is between two particles in two different perspectives instead of between two particles in subsequent frames. In this section we will describe this process in some detail.

A bundle matching analysis executes in the following way. Perspectives are taken in pairs and analysed in the same way as a PTV analysis, seeking to match particles using an auction algorithm based on a set of costings (see the next section). Remember that the matching occurs between the same frames (times) in different perspectives, instead of between different frames (times) in the same record, as is the case for PTV analysis. Pairs are analysed in the following way: perspective 1 is analysed with each of the subsequent perspectives in turn. Each of these pairs will generate a set of three-dimensional particles based on the matching process. These sets of particles are then consolidated by identifying corresponding particles in each of the sets. The consolidation is achieved through two key parameters. The first is a tolerance that indicates how far from one another two estimates of the same particle location can be. The second is the number of perspectives that must “agree” on the particle location.

The performance of this algorithm, like all PTV algorithms, is determined largely by the choice of costings. The range of costings available for this problem is substantially smaller than for the traditional PTV analysis. By far the most useful costing, termed the **distance costing**, is one based on the light rays described in the previous section, but other costings based on particle size or colour, for example, might also be useful.

Consider an idealized situation with two different perspectives, with perfectly calibrated cameras, and imagine that only a single particle is viewed by each camera. For such a situation each perspective possesses a single light path corresponding to the particle in view. Provided the two cameras are not degenerate (i.e. viewing along the same direction) the intersection of these two light paths determines the location of the particle in three-dimensional space. If more than two perspectives exist then the light paths from all perspectives must intersect at the same point, and the particle location is unique.

However such idealized conditions are unlikely to exist. The cameras will not be perfectly calibrated due to a range of factors (e.g. uncertainty in the target particle 3D coordinates), and nor will the particle locations in the particle 2D record be exact. Thus in any real application the light rays from the perspectives are unlikely to intersect. Instead they will pass very close to one another, and one might deduce the particle location as the point halfway between the rays at their closest point. For multiple perspectives these particle location estimates will all be slightly different.

In practice, of course, each perspective will be viewing hundreds or thousands of particles. These particles may not be present in all perspectives for a variety of reasons, such as some particles being obscured by others. In addition, we will not have the idealized conditions considered above. Therefore the problem of determining three-dimensional particle locations becomes an optimization problem. Consider again two perspectives. The particles in each perspective will give rise to a set of light rays passing through the particle domain. The optimization process seeks to match these rays (particles) and hence deduce the locations of particles in 3D space corresponding to these pairs of matched particles. It is perhaps clear that a costing that tries to minimize the closest distance between pairs of rays would lead to the most plausible set of matched particles and hence a robust set of 3D particles. This costing is the distance costing referred to above.

Construction of three-dimensional particle fields is much more robust when multiple perspectives are present. It is too early in our experience with these techniques to be able to recommend an optimal number of cameras, but from our observation of other systems more than four and less than eight seems to be appropriate. It is likely that this number will depend on the application.

Construction of three-dimensional particle fields is performed using a bundle PTV analysis pipeline that comprises a number of bundle matching analyses. The concepts behind these pipeline and analysis objects closely parallel the PTV analysis pipeline and PTV analysis objects discussed in chapter 7, although a number of the parameters associated with PTV analysis are not present in the bundle analysis.

The entire process is most easily illustrated with an example. Let us return to the situation where a single particle is present and perhaps four perspectives of the particle exist. Perspectives 1 and 2,

perspectives 1 and 3 and perspectives 1 and 4 will be analysed in turn. In each case an estimate of the particle location will be computed from the closest point between the two rays. Thus three particle location estimates will be available. If the user requires agreement between all four perspectives then all three of these estimates must lie within the specified tolerance of the average of the location. If only three perspectives must be agreement then if any pair of those estimates lie within the tolerance of their average location then a particle is identified.

It is clear that the smaller the tolerance and the larger the number of agreements required the more robust the particle identification. However, while it is tempting to set the number of agreements to its maximum value, in practice, this is likely to lead to a significant reduction in the number of particles identified. The most obvious reason is that whenever a particle is obscured by another or is “lost” from one perspective in some other way, this particle will never be identified. But a more subtle reason is when the particle domain includes obstacles of some sort, and particles in some locations simply cannot be seen by one or more of the cameras. For example the user may be measuring the flow around an opaque circular cylinder.

Once the consolidation process is complete all particles in the various perspectives are removed from the matching process. Perspective 2 is now considered and analysed in pairs with all subsequent perspectives. Again the identified particles are consolidated and added to the previous set of identified particles. And so the process continues until all perspectives are analysed in this way. It is worth noting that the number of perspectives that need to be analysed can be reduced based on the user requirement for the number of perspectives that must agree. Consider, for example, a system where eight cameras are present and the user has specified that at least five perspectives must agree on a particle location. In this case once perspectives 1, 2, 3, and 4 have been analysed there is no point in continuing the analysis due to the fact that only four perspectives remain and therefore any particles identified with these four perspectives will not satisfy the user requirement.

As discussed in chapter 6 *Streams* does provide a view where different particle 3D records can be compared. This provides a mechanism for exploring the particle records generated using different costings, tolerances and/or number of agreements.

The output of a bundle PTV analysis pipeline is a particle 3D record comprising a set of frames, each containing the 3D particles identified by the pipeline analysis.

9.5 Bundle matching costings

In this section we will consider the costings used to match particles between two different perspectives in a particle 2D record bundle.

There are six costings currently implemented. These are

- colour costing (see section 9.5.1)
- distance costing (see section 9.5.2)
- intensity costing (see section 9.5.3)
- mapping error costing (see section 9.5.4)
- regions costing (see section 9.5.5) and
- size costing (see section 9.5.6)

In the discussion in this section we will refer to the two frames being analysed as frames 1 and 2, frame 1 being the frame whose particles are being matched in perspective 1 and frame 2 being the frame in perspective 2. We will use p1 to refer to an arbitrary particle, or light path, in frame 1, and p2 to refer to an arbitrary particle and its associated light path in frame 2.

9.5.1 Colour costing

The **colour costing** measures the difference in colour between the two particles. The absolute difference in the intensity of each of three guns is computed and each of these is multiplied by a user-defined weight (the default being 1, and the weight being limited to the range 0-1). The cost is computed as the root-mean-square of these three differences, normalized by the maximum intensity of 255. Thus two particles with the same colour will produce a cost of 0, while a black particle and a white particle will produce the maximum cost of 1.0 if all colour weights are set to their default values.

9.5.2 Distance costing

The **distance costing** is probably the most useful costing as discussed in the preceding section. It computes the shortest distance

between the two light paths corresponding to particles p_1 and p_2 . The distance is normalized by a reference length set by the user to compute the cost.

9.5.3 Intensity costing

The **intensity costing** measures the difference in intensity between the two perspectives. This costing is only likely to be useful if there is some assurance that the intensity of particles is independent of viewing location.

9.5.4 Mapping error costing

The **mapping error costing** takes advantage of the mapping function characterization of the perspective cameras. This costing computes the location in 3D space of a particle that minimizes the mapping error for p_1 and p_2 . This process is computationally very expensive because the determination of the 3D particle location is computed by a non-linear optimization process. To make the costing more computationally efficient it is important to drastically reduce the particles in frame 2 that are considered as matches for p_1 . This can be done by using the distance costing as a filter on the particles to be considered in frame 2. The user has the choice of whether to do so or not but in the vast majority of cases this option should be adopted and for this reason it is the default.

9.5.5 Regions costing

The **regions costing** computes the point on the light path of p_1 that is closest to the light path of p_2 and determines whether this point lies within a user-defined set of regions. If it does the cost is that specified by the user. If not the cost is zero.

9.5.6 Size costing

The **size costing** computes the absolute difference in particle radius normalized by the larger of the two radii.

9.6 Predict and verify

The predict and verify process has similarities to the bundle matching analysis, in that it makes use to the concepts underlying the

distance and mapping error costings, but the overall approach is somewhat different.

The particle construction process involves two steps. In the first step, the **prediction step**, two of the bundle's perspectives are selected and, from their sets of 2D particles, a set of possible three dimensional particles is constructed. This process proceeds as follows. One particle from each perspective is chosen. Using their two-dimensional coordinates to guide it a non-linear optimisation process seeks a location (particle) in 3D space that will minimize the mapping error when that location is mapped onto each of the perspectives. This mapping error, like that discussed in section 9.5.4, is computed from the difference between the actual locations of the two particles in the two perspectives and the mapped location of the 3D point. If this mapping error is less than a user specified tolerance the 3D point is stored as a potential 3D particle. If not, the location is discarded. This process is repeated for all pairs of particles in the two perspectives until all possible potential particles have been identified.

Once the set of potential 3D particles have been found for the two perspectives the **verification step** uses the remaining perspectives to confirm the validity of these particles. For each perspective the 3D particle location is mapped onto the perspective's 2D space using the camera's mapping functions. The particle in that perspective, closest to the mapped point, is identified, and if the distance between its location and the mapped 3D particle location is less than a user specified tolerance (not necessarily the same as that referred to in the prediction step) the 3D particle is verified. Once all perspectives have been analysed in this way the number of perspectives that agree on a particle location can be computed, and if greater than or equal to the number of agreements required by the user, the particle is confirmed as a valid 3D particle.

At the conclusion of the verification step all particles that have contributed to the construction of one of the 3D particles are removed from their perspectives and the process starts again with two other perspectives. This process repeats until all perspectives have been analysed in this way and no further 3D particles can be found.

As indicated in section 9.5.4 the optimization process that determines the best 3D point, based on points from two perspectives,

is computationally demanding. The impact of this is that the predict and verify process, as described above, is unreasonably slow when the number of particles involved is typical of practical applications. In order to resolve this computational bottleneck the predict and verify process can use the same calculation as the distance costing described in section 9.5.2 to screen particle pairs before the full optimisation process is executed. If the distance between the light rays emanating from two particles is greater than a user specified value the full optimization process is not conducted and the two particles are judged to not correspond to the same 3D particle. For the vast majority of particle pairs in the two perspectives this screening process will fail, and due to the vastly lower computational demands of the distance costing calculation the overall process is speeded up considerably. It might also be noted that for any particle pair for which the determination of the light path fails for at least one of the particles (see the discussion in section 9.3) the full optimization process can proceed. As this failure is extremely rare the computational cost is negligible.

The predict and verify process proceeds in a similar way to the bundle matching process by working through pairs of perspectives. Thus, perspectives 1 and 2 are analysed and the potential 3D particles generated are then verified by perspectives 3, 4, 5 and so on. On completion significant numbers of particles are deleted from each of these perspectives and the process is repeated, with perspectives 2 and 3 being used in the predication step. Perspectives 4, 5 and so are then employed to verify the new set of 3D particles. These confirmed particles are added to the list of resolved 3D particles, more particles are removed from the contributing perspectives, and the process repeats again. It is perhaps clear that in normal circumstances each predict and verify cycle will require less computation than the last as the number of eligible particles in the perspectives reduces, sometimes quite dramatically after the first predict and verify cycle.

Our experience is that the predict and verify process is more reliable than the bundle matching analysis, and the resulting particle sets are more robust. However, our experience in 3D particle tracking is limited at this point and a final judgment on the preferred approach would be premature.

9.7 Summary

In this chapter we have considered the two analysis tools available to construct three-dimensional particle fields from particle 2D record bundles. The predict and verify process is viewed as the most reliable based on evidence to date, and is recommended as the preferred option.

I 0 Intensity Field Time Series

I 0.1 Overview

Intensity field time series (or simply intensity fields) are one of the Eulerian fields that can be generated within *Streams*. Its name is slightly misleading because the field produced typically corresponds to a tracer concentration. However, because these fields are based on the intensity of light recorded in the experimental images, the generic term intensity field has been chosen.

An **intensity field time series** is a scalar field defined on a rectangular three-dimensional grid. The three grid dimensions correspond to x , y and t – the horizontal, vertical and time coordinates. In addition to the scalar field itself spatial derivative information is also stored in the intensity field.

Intensity fields are stored on disk with a .int file extension and previously stored fields can be retrieved via the Eulerian menu in the *Streams* application view. Once loaded the field appears in the *Streams* object list view where its name and characteristic icon are displayed.

I 0.2 Intensity field creation

Intensity field time series are created by a create intensity field process operating on an image sequence. When the time series is created the user supplies a name for the field, information about the spatial grid and the number of frames in the image sequence that are to be used, details of how intensity gradients are to be computed, and the manner in which the intensity is defined if the images are RGB.

Each two-dimensional intensity field within the series 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 mesh 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. The user does have the option to use a spatial average of pixels surrounding each grid point in computing the intensity field. This can be useful in averaging noise in the image although it is computationally more expensive,

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 using 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 accurate derivative estimates are used.

More information on these parameters can be found in the *User's Guide: Core Objects* manual.

10.3 Calculators

A full three-dimensional intensity field is difficult to digest. For this reason all Eulerian field time series provide a number of specialized calculators that enable the user to extract specific information from the field. The calculators that are available for an intensity field can be found in the *User's Guide: Core Objects* manual. They include the intensity itself, and its gradient and derivatives.

The calculators defined for an intensity field object are displayed in the intensity field calculator view (see below) – the default view for 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 calculator manager view.

The result of any of these computations is a field object (discussed in section 2.5). These field objects are stored with the calculator.

10.3.1 Custom calculators

In addition to the standard predefined calculators, the user can create customized calculators. These calculators are based on a mathematical formula specified by the user. The formula instructs the calculator on how to compute the resulting field at each grid point. Included in this formula can be the 3 independent variables, x , y , and t , as well as the field variables such as the intensity and its derivatives. A full description of these field-based formulae can be found in the *User's Guide: Core Objects* manual.

A number of specialized functions available for inclusion in these custom calculator formulae are worth mentioning here. These are $\text{aveX}()$, $\text{aveY}()$, $\text{aveT}()$, $\text{aveXY}()$, and $\text{aveXYT}()$. The meaning of each is probably clear from its name. These functions provide a range of tools for averaging their arguments over one, two or three dimensions. For example, a custom calculator with the formula

$$I - \text{aveT}(I) \quad (10.1)$$

where, I is the intensity field, would generate the fluctuating intensity field, I' .

Four different brackets are provided for use in custom calculators. These are $()$, $[]$, $\{\}$ and $\langle \rangle$. The first three of these act as simple brackets, and can be used to group expressions within the formula. The first, of course, also plays the role of identifying the arguments of function calls. The last is commonly used as a short hand notation for time averaging and therefore in *Streams* it is equivalent to the function $\text{aveT}()$. Thus equation 10.1 could be written as

$$I - \langle I \rangle \quad (10.2)$$

Another slightly unusual function that can be very useful is the **Heaviside function** $h()$. This function is defined to have a value of 1 when its argument is zero or positive, and a value of 0 if its argument is negative. This function provides a mechanism for specifying functions in a piecewise fashion.

Custom calculators are stored with the intensity field when it is saved.

10.4 Processes

Only two processes are defined to operate directly on an intensity field. These are discussed below.

10.4.1 Transform pipelines

A **transform pipeline process** includes a number of individual transforms that can transform an intensity field in some way. When applied to an intensity field a new intensity field is created that corresponds to the output of the transform pipeline. Seven different individual transforms are available.

Linear

The linear transform enables the user to scale and offset the three coordinates, x , y and t , and to scale and offset the field data itself. This transform is primarily used for non-dimensionalising field data but it can also be useful in relocating the spatial and temporal origins.

Swap x

The swap x transform inverts the field in the horizontal direction so that the left hand edge of the field becomes the right hand edge and vice versa. The field data is unaffected.

Swap y

The swap y transform inverts the field in the vertical direction so that the top of the field becomes the bottom and vice versa. The field data is unaffected.

Subset

The subset transform extracts a subset of the original field data. The user can specify limits on the range of x , y and t variables present in the transformed field.

Average in time

The average in time transform compresses the field in time. The spatial fields in the transformed field are obtained by averaging a number of sequential frames in the original field. This is best explained by an example. Suppose an intensity field includes spatial fields at N different times, with a time step of Δt . If the user chooses to average over every 5 frames, the resulting intensity field will have only $N/5$ spatial fields and the time step between them will be $5\Delta t$. Each field in the new intensity field will be the average of 5 sequential frames in the original.

Constant velocity

It is a common experimental circumstance to use a video camera, fixed in the laboratory frame, to observe a fluid phenomenon that is steady in a moving frame of reference. Consider the example of a jet in a co-flow where the co-flow is artificially constructed by drawing the jet source through a stationary fluid, while the camera remains fixed. If the camera had moved with the jet source more useful flow data would be available. This can be achieved artificially by shifting the fields within the intensity field with a constant velocity in the opposite direction to the apparent camera motion. This is the effect of the constant velocity transform. The intensity fields within the time series are now spread out in space so the resulting field will have much larger spatial dimensions than the original.

File displacements

The file displacements transform effectively achieves the same result as the constant velocity transform except now the displacement of each frame is given by offsets stored in a text file. Displacements for all frames are not necessarily the same.

10.4.2 Save intensity field

The **save intensity field process** is a useful component of a pipeline where the result of a previous process, the creation of an intensity field, needs to be stored to disk.

10.5 Views

The intensity field time series has three views.

10.5.1 Calculator view

The calculator is the default view for the intensity field time series. It lists the currently defined calculators. This list will include all of the predefined calculators as well as any custom calculators that have been defined by the user. Calculators can be created and removed in this view, including the predefined calculators. The predefined calculators can be restored via a menu choice although any fields that were previously stored with the calculators will be lost. Custom calculators cannot be restored.

Double clicking a calculator in this view will bring up the calculator manager view where the fields of interest can be computed.

10.5.2 Details view

The details view provides information about the spatial and temporal extent of the field. None of this information can be changed by the user. The view also displays the field name and its description. The latter can be edited in this view.

10.5.3 Process view

The process view matches that for all other entities. It differs only in the lists of primary and secondary processes that can be created.

10.6 Summary

This chapter has provided a very brief overview of intensity field time series, Eulerian fields that provide access to scalar field data.

II Velocity Field Time Series

II.1 Overview

Velocity data is often the most informative when trying to understand fluid flows, so the generation of velocity fields from PTV data is common practice. In general fluid dynamicists prefer Eulerian velocity fields to Lagrangian fields, so the particle-based velocities obtained from a PTV system need to be interpolated onto a rectangular grid in order to convert them to an Eulerian form.

A **velocity field time series** is an Eulerian vector field defined on a rectangular three-dimensional or four-dimensional grid depending on whether the time series has been created from a two-dimensional or three-dimensional particle record. The grid dimensions correspond to x , y and t , or x , y , z and t . In order to provide access to such quantities as vorticity, velocity derivative information is also stored at each node in the grid, although for the velocity field derived from a particle 3D record the user has the option not to store this information in order to reduce memory requirements. Access to the derivatives is still available in the field time series calculators through finite difference based calculations using the field values on the nodes of the grid.

Velocity fields are stored on disk with a `.vel2d` or `.vel3d` file extension and previously stored fields can be retrieved via the Eulerian menu in the *Streams* application view. Once loaded the field appears in the *Streams* object list view where its name and characteristic icon are displayed.

II.2 Velocity field creation

Velocity fields are created by a create velocity field process operating on a particle record that has previously been processed by a PTV pipeline in order to generate particle matches. The processes for particle 2D records and particle 3D records have much in common and the following discussion is applicable to both. Where differences occur these will be noted.

The create velocity field process interpolates an Eulerian velocity field onto a rectangular grid using the particle-based velocities

available from the particle record. This interpolation process has the following steps:

Velocity calculation

The velocity of each particle, in each frame, must be calculated. Particles that are matched to other particles in the previous or subsequent frame are able to have velocities calculated for them, while particles that are unmatched cannot. The velocity calculation can be performed using a **forward difference** approximation (if the particle is matched in the next frame), a **backward difference** approximation (if the particle is matched in the previous frame) or a **central difference** approximation (if the particle is matched in both frames). By basing the interpolation on only those particles that possess two matches the overall accuracy of the velocity estimates will improve. However, this advantage may be counterbalanced somewhat by the reduced number of particles that can contribute to the interpolation.

If a match exists in the previous frame only, then the velocity is calculated using a simple backward difference scheme

$$\mathbf{u}_{ij} = \frac{\mathbf{r}_{ij} - \mathbf{r}_{kj-1}}{\Delta t} \quad (11.1)$$

$\mathbf{u}_{ij}$ – the velocity of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{ij}$ – the position vector of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{kj-1}$ – the position vector of the k^{th} particle in the $j-1^{\text{th}}$ frame, where it is assumed that this particle is matched to particle i in the j^{th} frame.

Δt – the time step between frames.

If a match exists in the next frame only, then the velocity is calculated using a simple forward difference scheme

$$\mathbf{u}_{ij} = \frac{\mathbf{r}_{kj+1} - \mathbf{r}_{ij}}{\Delta t} \quad (11.2)$$

$\mathbf{u}_{ij}$ – the velocity of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{ij}$ – the position vector of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{kj+1}$ – the position vector of the k^{th} particle in the $j+1^{\text{th}}$ frame, where it is assumed that this particle is matched to particle i in the j^{th} frame.

Δt – the time step between frames.

If matches exist in both frames then a central difference scheme is used.

$$\mathbf{u}_{ij} = \frac{\mathbf{r}_{mj+1} - \mathbf{r}_{kj-1}}{2\Delta t} \quad (11.3)$$

$\mathbf{u}_{ij}$ – the velocity of the i^{th} particle in the j^{th} frame.

$\mathbf{r}_{mj+1}$ – the position vector of the m^{th} particle in the $j+1^{\text{th}}$ frame, where it is assumed that this particle is matched to particle i in the j^{th} frame.

$\mathbf{r}_{kj-1}$ – the position vector of the k^{th} particle in the $j-1^{\text{th}}$ frame, where it is assumed that this particle is matched to particle i in the j^{th} frame.

Δt – the time step between frames.

Another possibility is to use more particles than just those in the previous and subsequent frames. A particle path can be constructed from the particles, linked to the particle of interest, through matches in past and future frames. A polynomial least squares fit to the positions of these particles can be differentiated to obtain the velocity. The order of the polynomial is half the total path length (the total number of particles linked by matches) less one up to a maximum value set by the user.

In this case the velocity is given by

$$\mathbf{u}_{ij} = \left. \frac{d\mathbf{r}}{dt} \right|_{t_j} \quad (11.4)$$

$\mathbf{u}_{ij}$ – the velocity of the i^{th} particle in the j^{th} frame.

$\mathbf{r}(t)$ – is the polynomial fit to the particle path

t_j – is the time of the j^{th} frame.

No matter which method is used to calculate the velocity of a particle, erroneous matches will adversely affect the calculated velocity field. The elimination of incorrect matches must remain the principal goal of the analyst. Sometimes this will come at the expense of the total number of matches made. In general, missed matches, compromise the velocity field to a far lesser degree than incorrect matches.

Grid specification

For the interpolation process to proceed, a regular, rectangular grid must be specified. Velocities will be interpolated onto the nodes of this grid. The selection of the grid must be an informed decision. Very fine grids, to resolve small-scale structures in the flow, may seem highly desirable. However, if the density of particles in a frame does not provide sufficient resolution to identify these structures a fine grid will not help.

Another consideration in grid selection is the issue of accuracy near the frame edges. To be more specific, accuracy near the edge of the particle field in each frame. A grid point that lies outside the region covered by the particles (actually, outside the triangulation of the particles – see below) will require extrapolation to determine its velocity. In addition, particle matching near frame boundaries is also error prone, due to the exit and entry of particles over time. Therefore, it is recommended that velocity information is not sought near the boundaries of the region covered by particles.

Interpolation

A number of schemes exist for interpolating a function, defined at randomly located points in the plane, and three different options are provided. These are denoted

- Triangle or tetrahedron based
- Least squares
- Binned

The triangle-based (where we will use the term triangle in a general sense to represent triangles in two dimensions and tetrahedra in three dimensions) approach adopts the methods of Cline and Renka (1984) and Wilson (1981) that produce a triangulation of the particles in a frame, and uses this triangulation as the basis for the interpolation.

This triangulation, known as a **Delaunay triangulation**, has a number of special properties, the most important of which is that the minimum interior angle in two adjacent triangles is maximised. This has a number of corollaries, but from a practical point of view it simply means that the triangles tend not to be long and thin. Thus interpolations within the triangle will be based on three or four particles that are “close together”.

To estimate the velocity at a grid point its containing triangle is found, and the velocity is interpolated linearly from the velocities of the particles at the triangle’s corners. This interpolation is based on that used for a triangular/tetrahedral element in the finite element method (FEM). If a grid point is located outside the triangulation the velocity at that point is left undefined. In addition the user is at liberty to specify the maximum size of triangle that may be used to interpolate the velocity data. If a grid point is located in a triangle that has dimensions larger than those set by the user then the velocity will be left undefined at that grid point.

The least squares approach fits a first, second or third order, two- or three-dimensional, polynomial function to the velocities of particles neighbouring the point of interest. This function is fitted in a least squares sense, and the user is at liberty to choose the number of particles that participate in this fitting process. If the user chooses to extrapolate their velocity data beyond the triangulation described above the least squares method must be used.

The binned approach follows the method suggested by Cowen and Monismith (1997). A rectangular bin (two- or three-dimensional), centred on each grid point, is defined by the user. The velocities of particles that fall within each bin are averaged in order to estimate the velocity at the associated grid point. This is a zeroth order interpolation scheme, but if particle densities are high, the result should be robust. If no particles lie within a particular bin in one frame, the velocity at that grid point is left undefined.

Partial derivatives of the velocities are also required if derived flow quantities such as vorticity are to be evaluated. The user has, again, a range of options for the calculations of the partial derivatives. These are

- Grid finite difference
- Triangle least squares

- Grid least squares
- Triangle constant interpolation

The grid finite difference approach uses a finite difference approximation to the velocity derivatives at each grid point using the velocities interpolated onto the rectangular grid. The user can select the order of the finite difference scheme used. If a second order scheme is selected then derivatives are only available at grid points where velocities are defined on either side.

The triangle least squares approach uses the least squares approach described above to fit a two- or three-dimensional polynomial to the particle velocities surrounding each particle in the frame. This function then provides first order derivative estimates at each particle and the partial derivatives at each grid point are calculated using triangulation-based interpolation of the derivatives.

The grid least squares method uses the same least squares approach but fits the function at the grid points themselves, thus avoiding the use of the triangulation altogether.

The triangle constant interpolation method uses the triangulation technique and fits a first order two- or three-dimensional function to the velocities at the three vertices of the triangle containing the grid point. The first derivatives of this function are used as the estimates for the partial derivatives at the grid point.

It should be noted that while velocity derivatives are always computed for two-dimensional velocity fields the user has the option not to do so for three-dimensional fields. The reason for this is to reduce memory requirements for storing the velocity field. If derivative information is not stored with the velocity field, derivatives can still be computed by the field calculators by using finite difference approximations based on the nodal values of the velocity. There is a computational cost in doing these calculations on the fly but the savings in memory are substantial and will generally outweigh this extra computational effort.

The user is at liberty to choose any combination of these various interpolation and partial derivative estimation methods.

The accuracy of the interpolation process is critical to the PTV analysis, and the limitations of the algorithms used here should be borne in mind. For smoothly varying flows, with high particle

density, none of the interpolation processes are likely to introduce significant error compared to other areas of uncertainty, such as particle identification and particle matching. However, for flows with significant gradients the legitimacy of the interpolated fields should always be critically reviewed.

Further information on the parameters involved in the interpolation refer to the *User's Guide: Core Objects* manual.

11.3 Calculators

As for all Eulerian field time series specific field information is extracted from the velocity field time series using a suite of predefined calculators. These calculators are described in the *User's Guide: Core Objects* manual and include quantities such as the velocity field, its individual components, vorticity, turbulent kinetic energy, turbulent stress and so on.

The calculators fall into two broad categories, those that produce scalar fields, such as the components of the velocity, u and v , the turbulent kinetic energy, vorticity etc., and those that produce vector fields such as the velocity field (u,v) . The range of predefined calculators is extensive and provides the user with access to a range of turbulent statistics as well as bulk flow quantities.

The calculators defined are displayed in the velocity field calculator view (see below) – the default view for the velocity field time series 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 u' velocity component at one grid point in order to obtain the spectra of the turbulence at this point. The desired projection can be set in the calculator manager view.

The result of any of these computations is a field object as discussed in section 2.5. Details of these fields can be found in the *User's Guide: Field Objects* manual.

11.3.1 Custom calculators

In addition to the standard predefined calculators, velocity fields enable the user to define custom calculators that produce scalar field or vector fields. These calculators are based on mathematical

formulae specified by the user. In the case of a customized scalar field calculator the user supplies a single formula, while for the customized vector field calculator the user provides two or three formulae, one for each component of the vector field produced.

The formula instructs the calculator on how to compute the resulting field at each grid point. This formula can include the independent variables, x , y , possibly z , and t , as well as a number of field variables. These include u , v , w , u' , v' , w' , ke , tke , $u'v'$, ω , $\partial u/\partial x$, $\partial u/\partial y$ etc. The formula can include the standard unary and binary operators, brackets, and mathematical functions such as the trig and exponential functions. A full description of these field-based formulae can be found in the *User's Guide: Core Objects* manual.

The averaging functions described in section 10.3.1 are available for velocity fields as well.

Custom calculators are stored with the velocity field when it is saved.

11.4 Processes

Only two processes are defined to operate directly on a velocity field time series. These are discussed below.

11.4.1 Transform pipelines

A **transform pipeline process** includes a number of individual transforms that can transform a velocity field. When applied to a velocity field a new velocity field is created that corresponds to the output of the transform pipeline. A range of different transforms are available for two- and three-dimensional velocity fields and these include linear transformations, swapping of the spatial axes and translating the field into a moving frame of reference.

11.4.2 Save velocity field

The **save velocity field process** is a useful component of a pipeline where the result of a previous process, the creation of a velocity field, needs to be stored to disk.

11.5 Views

The velocity field time series has three views.

11.5.1 Calculator view

The calculator is the default view for the velocity field time series. It lists the currently defined calculators. This list will typically include all of the predefined calculators as well as any custom calculators that have been defined by the user. Calculators can be created and removed in this view, including the predefined calculators. The predefined calculators can be restored via a menu choice although any fields that were previously stored with the calculators will be lost. Custom calculators cannot be restored.

Double clicking a calculator in this view will bring up the calculator manager view where the fields of interest can be computed.

11.5.2 Details view

The details view provides information about the spatial and temporal extent of the field. None of this information can be changed by the user. The view also displays the field name and its description. The latter can be edited in this view.

11.5.3 Process view

The process view matches that for all other entities. It differs only in the list the primary and secondary processes that can be created.

11.6 Creation of velocity 3D field time series from model

The user can create a velocity 3D field time series directly from a simple model. This model mimics that of the custom formula model for creating particle records discussed in section 6.3.6. The user defines the components of the velocity field through three mathematical formulae that are functions of time and space. This facility, accessed through the main application view process menu, is provided to enable the user to familiarize themselves with some of *Streams* capability with respect to manipulating and visualizing these

three-dimensional fields. See the *User's Guide: Core Objects* manual for more details.

11.7 Summary

This chapter has provided a very brief overview of velocity field time series, Eulerian fields that provide access to velocity field data.

12 Velocity-intensity Field Time Series

12.1 Overview

Modern experimental techniques, such as PIV-LIF and PTV-LIF, can provide simultaneous measurement of velocity and scalar fields. The advantage of obtaining these fields at the same time is that it enables scalar fluxes, both mean and turbulent, to be computed.

Streams does not directly support the analysis of images containing both velocity and scalar information but it does allow separately computed two-dimensional velocity and intensity fields to be combined. To perform this combination the two field time series must have identical “grids”. Thus the spatial grid spacings and the time steps must match exactly and the number of nodes in the x and y directions and the number of frames must also match. However the origins of the spatial grids and the times of the first frame need not match. The origin and initial time of the velocity field is taken as the origin and initial time of the combined **velocity-intensity field time series**.

Velocity-intensity fields are stored on disk with a .vin file extension and previously stored fields can be retrieved via the Eulerian menu in the *Streams* application view. Once loaded the field appears in the *Streams* object list view where its name and characteristic icon are displayed.

12.2 Velocity-intensity field creation

Velocity-intensity fields are created by combining two previously created velocity and intensity field time series that must satisfy the requirements referred to above. The velocity and intensity fields are loaded from disk and the user simply opens a dialog box that enables them to choose the two fields to combine. In order to save memory the original fields are closed once the combined field is created. The system automatically checks the correctness of the combination.

12.3 Calculators

Velocity-intensity fields share all of the predefined calculators for the separate velocity and intensity but with the addition of some cross-calculators that enable scalar fluxes to be computed. The full list of calculators is provided in the *User's Guide: Core Objects* manual.

The calculators defined are displayed in the velocity-intensity field calculator view (see below) – the default view for the velocity-intensity field object. Each field calculator provides a **manager view** that allows the user to generate different projections, or subfields, of the calculated field.

The result of any of these computations is a field object as discussed in section 2.5. Details of these fields can be found in the *User's Guide: Field Objects* manual.

12.3.1 Custom calculators

In addition to the standard predefined calculators, velocity-intensity fields enable the user to create custom calculators in the same way as for velocity and intensity fields. These custom calculators, based on user-defined formulae can access both the intensity and velocity field variables.

Custom calculators are stored with the velocity-intensity field when it is saved.

12.4 Processes

Only two processes are defined to operate directly on a velocity-intensity field time series. These are discussed below.

12.4.1 Transform pipelines

A **transform pipeline process** includes a number of individual transforms that can transform a velocity-intensity field. When applied to a velocity-intensity field a new velocity-intensity field is created that corresponds to the output of the transform pipeline. The seven different individual transforms available are the same as those for the intensity field described in section 10.4.1.

12.4.2 Save velocity-intensity field

The **save velocity-intensity field process** stores a velocity-intensity field, to disk.

12.5 Views

The velocity-intensity field time series has three views.

12.5.1 Calculator view

The calculator view is the default view for the velocity-intensity field time series and provides the same functionality as the equivalent view for intensity or velocity field time series.

12.5.2 Details view

The details view provides information about the spatial and temporal extent of the field. None of this information can be changed by the user. The view also displays the field name and its description. The latter can be edited in this view.

12.5.3 Process view

The process view matches that for all other entities. It differs only in the list the primary and secondary processes that can be created.

12.6 Summary

This chapter has provided a very brief overview of velocity-intensity field time series, Eulerian fields that provide access to velocity and intensity field data.

I3 Density Field Time Series

I3.1 Overview

Density field time series (or simply density fields) are one of the Eulerian fields that can be generated within *Streams*. A **density field time series** is a scalar field defined on a rectangular three-dimensional grid that stores information about the density of particles in a particle 2D record. The density does not refer to fluid density which is likely to be represented by an intensity field time series, as discussed in chapter 10.

Density fields are stored on disk with a .den file extension and previously stored fields can be retrieved via the Eulerian menu in the *Streams* application view. Once loaded the field appears in the *Streams* object list view where its name and characteristic icon are displayed.

I3.2 Density field creation

Density fields are created by a create density field process operating on a particle 2D record. When the field is created the user supplies a name for the field, information about the spatial grid, the number of frames in the image sequence that are to be used, and details of the interpolation process.

The user defines the grid on which the density is to be evaluated. The number of particles lying in the rectangle surrounding each grid point is evaluated and divided by the area of the rectangle. Derivatives of the density at each grid point are calculated using the finite difference method described for the interpolation of the velocity field.

More information on these parameters can be found in the *User's Guide: Core Objects* manual.

I3.3 Calculators

As for all Eulerian fields the density field provides a number of pre-defined calculators that allow the user to extract specific information about the field. See the *User's Guide: Core Objects* manual for details.

The calculators defined for density field object are displayed in the density field calculator view (see below) which is the default view for the density field object. Each field calculator provides a **manager view** that allows the user to generate different projections, or subfields, of the calculated field as discussed in section 10.3. Density fields do not provide custom calculator capability.

I3.4 Processes

Only two processes are defined to operate directly on a density field.

I3.4.1 Transform pipelines

A **transform pipeline process** includes a number of individual transforms that can transform a density field in some way. When applied to a density field a new density field is created that corresponds to the output of the transform pipeline. The available transforms match those provided for an intensity field time series – see section 9.4.1 for details.

I3.4.2 Save density field

The **save density field process** is a useful component of a pipeline where the result of a previous process, the creation of a density field, needs to be stored to disk.

I3.5 Views

The intensity field time series has three views.

13.5.1 Calculator view

The calculator is the default view for the density field time series. It provides the same functionality as the same view for other Eulerian field time series.

13.5.2 Details view

The details view provides information about the spatial and temporal extent of the field. None of this information can be changed by the user. The view also displays the field name and its description. The latter can be edited in this view.

13.5.3 Process view

The process view matches that for all other entities. It differs only in the list the primary and secondary processes that can be created.

13.6 Summary

This chapter has provided a very brief overview of density field time series, Eulerian fields that provide access to particle density data.

I4 Shape Field Time Series

I4.1 Overview

A **shape field time series** is an Eulerian scalar field defined on a rectangular three-dimensional grid that stores information about one of the shape parameters defined for particles in a particle 2D record. Currently three different shape parameters can form the basis of a shape field – the equivalent particle diameter, the particle area, or the aspect ratio of the particle. The third of these is only available if the particles within the particle record have the detailed shape information stored with them. In addition to the scalar field itself spatial derivative information is also stored in the shape field, although this data tends to be fairly unreliable unless the shape field is smooth in space and the particle density is high.

Shape fields are stored on disk with a .shp file extension and previously stored fields can be retrieved via the Eulerian menu in the *Streams* application view. Once loaded the field appears in the *Streams* object list view where its name and characteristic icon are displayed.

I4.2 Shape field creation

Shape fields are created by a create shape field process operating on a particle 2D record. The interpolation process follows that of the create velocity field time series process except in this case the user must select the shape field upon which the time series is to be based.

More information on these parameters can be found in the *User's Guide: Core Objects* manual.

I4.3 Calculators

The shape field provides a number of pre-defined calculators that allow the user to extract specific information about the field. See the *User's Guide: Core Objects* manual for details.

The calculators defined for a shape field object are displayed in the shape field calculator view (see below) which is the default view for the shape field object. Shape fields do not provide custom calculator capability.

14.4 Processes

Only two processes are defined to operate directly on a shape field. These are discussed below.

14.4.1 Transform pipelines

A **transform pipeline process** includes a number of individual transforms that can transform a shape field in some way. When applied to a shape field a new shape field is created that corresponds to the output of the transform pipeline. The available transforms are the same as those provided for an intensity field time series – see section 10.4.1 for details.

14.4.2 Save shape field

The **save shape field process** is a useful component of a pipeline where the result of a previous process, the creation of a shape field, needs to be stored to disk.

14.5 Views

The shape field time series has three views.

14.5.1 Calculator view

The calculator view is the default view for the shape field time series and provides the same functionality as the calculator view for other Eulerian field time series.

14.5.2 Details view

The details view provides information about the spatial and temporal extent of the field. None of this information can be changed by the user. The view also displays the field name and its description. The latter can be edited in this view.

I4.5.3 Process view

The process view matches that for all other entities. It differs only in the list the primary and secondary processes that can be created.

I4.6 Summary

This chapter has provided a very brief overview of shape field time series, Eulerian fields that provide access to particle shape data.

I5 Displacement Field Time Series

I5.1 Overview

As discussed in the first chapter there is considerable potential to apply PTV technology to other scientific disciplines. Structural mechanics is one area where these techniques, or similar ones, are starting to be applied and we have begun using *Streams* in both our material testing and structural testing laboratories. For these applications it is not velocity field information that is sought but displacement and strain fields. Often in the structural testing arena the time associated with the image recording is meaningless. The tests may be quasi-dynamic in that loads are incrementally applied to a structural specimen but these do not occur in a continuous fashion or at regular intervals. Thus the image recording may not occur at a fixed rate - the cameras capturing the material behavior being triggered based on the loading protocol, not by time triggers. For this reason the time variable typically plays no role in the displacement fields produced other than to provide an ordering of the images. In reality the displacement and strain data produced by *Streams* is not so much time dependent as load dependent and therefore time is just a surrogate for some other physical quantity in the system.

Of course, if structural testing is performed dynamically with appropriate rams, or a shake table, then time takes on its true meaning.

A **displacement field time series** is an Eulerian vector field defined on a rectangular grid. There are three different types of displacement field time series, two-dimensional, three-dimensional and three-dimensional surface. The first two are analogous to the two-dimensional and three-dimensional velocity field time series. In the first case the rectangular grid is three-dimensional with coordinates to x , y and t – the horizontal, vertical and time coordinates. In the second case the grid is four-dimensional with coordinates x , y , z and t .

Three-dimensional surface fields lie somewhere between the two-dimensional and three-dimensional fields. In common with the three-dimensional fields all displacements occur in three-

dimensional space. However these displacements are associated with a material that is described by a two-dimensional surface. A simple example should suffice to clarify the concept. Consider a road pavement surface. This pavement is covered in dots and these dots are tracked using a 3D PTV system. While these particles move in three-dimensional space, for much of the deformation of the pavement (perhaps due to cyclic loading by a wheel), the particles stay on a well-defined two-dimensional surface. The point is that there are no particles within the material that can be observed and tracked. Such an experiment would give rise to a three-dimensional surface field. All displacements occur in three-dimensional space but the grid required to specify these displacements is only two-dimensional (taken to be the x-y plane). At each grid point the displacement is recorded as well as the height of the surface above the plane.

These displacement fields are unlike other Eulerian fields in that the user has some control over how the displacement data maps onto the grid. The most obvious approach, referred to as **space-based**, mimics that for the velocity field whereby the displacement at a grid point is obtained by interpolating the displacements of the neighbouring particles. This is indeed one option. Thus if the test involved the bending of a beam the resulting displacement field would clearly show the physical displacements of the beam in that grid points at the top of the domain, for example, which initially corresponded to the upper region of the beam, would disappear from the field as that region of the beam moved downwards under the loading. A second approach, known as **material-based**, is to map all displacement, and hence strain, information back to the original location of the particles. In this way, taking again the beam example, the grid is fixed to the material in its original location. Thus the user can then easily track the strain, say, in the centre of the beam during the loading cycle.

Displacement fields are stored on disk with a .dis2d (previously .dis), dis3d or dis3dsur file extension and previously stored fields can be retrieved via the Eulerian menu in the *Streams* application view. Once loaded the field appears in the *Streams* object list view where its name and characteristic icon are displayed.

15.2 Displacement field creation

Displacement fields can be created by a create displacement field process operating on a particle 2D record or particle 3D record that has previously been processed by a PTV pipeline in order to generate particle matches. There are three different create displacement field processes that generate two-dimensional, three-dimensional and three-dimensional surface displacement fields. In addition, it is possible, and generally preferable, to create a two-dimensional displacement field from a two-dimensional Lagrangian path field. Whether the field is created from a particle 2D record or a Lagrangian 2D path field the process looks the same. However, because displacement fields are generally material-based, the latter generally results in a less noisy dataset.

The create displacement field process interpolates an Eulerian displacement field onto a rectangular grid using the particle-based displacements available from the particle record, or Lagrangian path field. This interpolation process has the following steps:

Displacement calculation

The displacement of each particle, in each frame, is calculated in the same way as for the velocity field described in chapter 11 with the difference that a temporal derivative is not taken. Particles that are matched to other particles in the previous or subsequent frame are able to have displacements calculated for them, while particles that are unmatched cannot be used. The displacement calculation is always performed using a **central difference** approximation, if it is available (i.e. a particle has matches in both previous and next frames). A **backward difference** approximation (if the particle is only matched in the previous frame) or a **forward difference** approximation (if the particle is only matched in the next frame) is used if necessary. Equations 11.1 – 11.3 are used but as simple differences in position of the particle.

In general the video records of structural or material tests are easier to analyse than those of most fluid flows as particles do not tend to enter or leave the domain in random fashion. Therefore it is likely that the vast majority of particles will include matches in both previous and next frames and there should be a high confidence in their accuracy. Even so, as discussed in previous sections, it is of

primary importance that erroneous matches are eliminated before the displacement field creation is undertaken.

Grid specification

For the interpolation process to proceed, a regular, rectangular grid must be specified. Displacement will be interpolated onto the nodes of this grid. The grid should cover the entire specimen during the testing regime. It is likely that portions of the grid will have no valid field values at some times, particularly if a space-based field and not a material based field is used (see below).

It should also be remembered, as discussed in chapter 11, that the resolution of the grid should be guided by the density of the particle field. There is little point in having a grid spacing much smaller than the typical inter-particle separation.

The grids for two-dimensional and three-dimensional displacement fields are the same as for the corresponding velocity field. However the grid for a three-dimensional surface displacement field will be only a two-dimensional grid. As explained previously it is assumed that all particles are located on a two-dimensional surface, that is a single-valued function of (x,y) , in three-dimensional space. Thus the grid in this instance is a grid in the x - y plane.

Field specification

As discussed above, the user can choose to compute the displacement field in space, or track the behaviour of the material itself. In the first case the field interpolation is done in exactly the same manner as for the velocity field. Here the displacement at a grid point is obtained by interpolating the displacements of the neighbouring particles according to the interpolation scheme used. In the second case the grid points in the first frame are taken as representing points on the material being tested. Using the particle motions the motion of these grid points are computed from frame to frame. The result is a trajectory for each grid point through time. At each subsequent location of a grid point the displacement is computed from interpolating the displacements of the neighbouring particles and this displacement is then mapped back to the grid point's original location. Thus the displacements, and gradients, at one grid point correspond to those of that part of the material through time.

Interpolation

The interpolation schemes available for the displacement field are identical to those used to interpolate the velocity field. These are discussed in detail in chapter 10. Note however that when material based displacement fields are produced the only option for derivative calculation is grid based finite difference.

Further information on the parameters involved in the interpolation refer to the *User's Guide: Core Objects* manual.

15.3 Calculators

Specific field information is extracted from the displacement field time series using a suite of predefined calculators. For the displacement field these calculators include the incremental displacement between frames, the cumulative displacement from the first frame, a variety of strain fields and rotation. These calculators are listed in the *User's Guide: Core Objects* manual.

There is ambiguity in how the various strain fields can be defined. Equations 15.1 – 15.5 provide definitions for the strain fields and rotation field computed within *Streams*. In these formulae X_c and Y_c are the cumulative displacements in the x and y directions.

$$\text{eps } x = \varepsilon_x = \frac{\partial X_c}{\partial x} \quad (15.1)$$

$$\text{eps } y = \varepsilon_y = \frac{\partial Y_c}{\partial y} \quad (15.2)$$

$$\text{eps } xy = \varepsilon_{xy} = \frac{1}{2} \left(\frac{\partial X_c}{\partial y} + \frac{\partial Y_c}{\partial x} \right) \quad (15.3)$$

$$\text{eps } vol = \frac{\partial X_c}{\partial x} + \frac{\partial Y_c}{\partial y} \quad (15.4)$$

$$\text{rot} = \omega = \frac{1}{2} \left(\frac{\partial Y_c}{\partial x} - \frac{\partial X_c}{\partial y} \right) \quad (15.5)$$

Three-dimensional surface displacement fields include a number of unique calculators associated with the surface itself. These calculators are the surface height above the x-y plane, designated as z , and the derivatives of this height in the x and y directions.

The calculators defined are displayed in the displacement field calculator view (see below) – the default view for the displacement 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 X_c at one grid point as a function of time/loading. The desired projection can be set in the calculator manager view.

The result of any of these computations is a field object as discussed in section 2.5. Details of these fields can be found in the *User's Guide: Field Objects* manual.

15.3.1 Custom calculators

In addition to the standard predefined calculators, displacement fields enable the user to create custom calculators in the same way as velocity field time series. These custom calculators can utilize the spatial and temporal variables as well as the field variables such as X_c , and Y_c . A full description of these field-based formulae can be found in the User's Guide: Core Objects manual.

Custom calculators are stored with the displacement field when it is saved.

15.4 Processes

Only two processes are defined to operate directly on a displacement field time series. These are discussed below.

15.4.1 Transform pipelines

A **transform pipeline process** includes a number of individual transforms that can transform a displacement field. When applied to a displacement field a new displacement field is created that corresponds to the output of the transform pipeline. The transforms available include linear transformations, swapping of the independent variable axes (x and y or x , y and z), extracting a subset

of the field and averaging over short time intervals. These correspond to the associated transformations described in section 10.4.

15.4.2 Save displacement field

The **save displacement field process** is a useful component of a pipeline where the result of a previous process, the creation of a displacement field, needs to be stored to disk.

15.5 Views

The displacement field time series has three views.

15.5.1 Calculator view

The calculator view default view for the displacement field time series and is similar to that provided for all Eulerian field time series.

15.5.2 Details view

The details view provides information about the spatial and temporal extent of the field. None of this information can be changed by the user. The view also displays the field name and its description. The latter can be edited in this view.

15.5.3 Process view

The process view matches that for all other entities. It differs only in the list the primary and secondary processes that can be created.

15.6 Summary

This chapter has provided a very brief overview of displacement field time series, Eulerian fields that provide access to displacement field data.

16 Lagrangian path fields

16.1 Overview

Lagrangian 2D and 3D path fields provide the user with Lagrangian information about two-dimensional and three-dimensional particle fields. A **Lagrangian 2D path field** is a set of particle paths generated from a pre-matched particle 2D record. A **Lagrangian 3D path field** is a set of particle paths generated from a pre-matched particle 3D record. The extraction of Lagrangian data from a particle record is considerably easier than its Eulerian counterpart because no interpolation process is required.

Data within a Lagrangian path field are not based on a three-dimensional grid, as it is for the Eulerian fields. Instead the data can be viewed as a set of particle paths, each of which is comprised of a sequence of particle locations over time. Thus, in a general sense, the field can be seen as a list of times series. Each time series is referred to as a **path**, and while a path is effectively the trajectory of a single particle, there is not a unique one to one correspondence between paths and particles. A particle path may be broken during an experiment either because the particle was not identified in one or more frames, or because it was identified but not matched. Therefore a single particle may, in fact, contribute a number of paths, starting and ending at different times.

The data within the Lagrangian field can also be viewed as a two dimensional matrix of data. The path number can be considered to label the rows of the matrix and time the matrix columns. However, because each path starts and ends at a different time this matrix is not fully populated. Matrix elements for a particular row are undefined at times when that path does not exist.

The data stored at each time along a path include the particle location, the particle velocity, computed in the same fashion as for the Eulerian fields, the particle radius and intensity, and the additional shape information if that is requested by the user and is available (for 2D fields only). Thus the path provides a time history of the state of each particle as it traverses its path. For pure fluid flows only the location and velocity data might be of concern to the

experimentalist, but for experiments where the particle tracking is associated with sediment transport for example, the extra shape information may well be of significant importance.

Lagrangian path fields are particularly meaningful for materials testing where particle paths correspond to the motion of different parts of the specimen under investigation. It is for this reason that the generation of displacement fields from Lagrangian path fields, referred to in the previous chapter, is the most natural.

Lagrangian 2D path fields are saved to disk with a .lgr2d (previously .lgr) file extension while the Lagrangian 3D path field has a .lgr3d file extension. Previously saved files can be loaded from disk using the Lagrangian menu in the *Streams* application view.

16.2 Lagrangian path field creation

Lagrangian 2D path fields are created by a create Lagrangian 2D path field process operating on a particle 2D record in which particles have been matched. Similarly Lagrangian 3D path fields are created by a create Lagrangian 3D path field process operating on a particle 3D record in which particles have been matched.

The create Lagrangian path field processes extract the particle information from the particle record and store the data as a sequence of particle paths. The paths have no particular order, although the processing of the record starts at the first frame and proceeds to the end. Thus paths that begin early in the record appear before those that start later. At each location in the path particle data are stored. This includes the particle position, velocity, and particle characteristics such as shape and intensity information if it is available in the case of a 2D field.

This extraction process involves the following decisions.

Particle

The user must specify how the velocity of the particle is to be calculated. The options match those that are available when an Eulerian field is computed – single estimate, double estimate and polynomial estimate. For a Lagrangian 2D path field the user also specifies whether they wish to store detailed particle shape

information such as aspect ratio, angle of the principal axes and eccentricity if it is available. The availability of this information is determined by the choice made when the particles were originally identified.

Length

Not all particle paths may be of interest. For example very short paths may provide little in the way of flow statistics and the user may wish to exclude them from the Lagrangian path field. Lower and upper bounds on the acceptable path lengths can be imposed.

Time

Similarly, paths throughout the entire time period of the particle record may not be required. For example, during an unsteady flow different time periods may be handled separately. The earliest and latest times for a path can be specified. This does not mean that a particle path starting before the earliest time, or one ending after the latest time, is ignored. These paths are simply truncated so that the stored path lies within the prescribed limits.

Space

Of more practical consideration might be the limiting of paths to certain regions of the flow. Paths within a wake region, but not external to it, might be of primary interest to the researcher for example. Therefore physical constraints may be placed on the acceptable paths. The user can choose to provide no spatial constraints, demand that paths start and end in particular regions, or require that particle paths lie entirely within certain regions.

Frames

The frames processed by this process can be limited.

Additional information about the constraints available, and the specific parameters set by the user is provided in the *User's Guide: Core Objects* manual.

16.3 Calculators

The Lagrangian 2D and 3D path fields possesses a number of predefined calculators, although it needs to be remembered that these calculators are operating on quite different data to their Eulerian field counterparts. Each field variable depends on two independent variables, path number and time, instead of the coordinates x , y , possibly z and t . These two variables are given the symbols p and t .

The available calculators, listed in the *User's Guide: Core Objects* manual compute quantities such as the particle velocity and its components, the particle position and the particle displacement between frames.

The calculators are displayed in the Lagrangian 2D and 3D path field calculator views (see below). Each field calculator provides a **manager view** that allows the user to generate different projections, or subfields, of the calculated field. For Lagrangian fields the user can manipulate only two variables, the path and the time. Averaging over time corresponds to averaging over each path, while averaging over paths corresponds to averaging over all paths at a particular time.

16.3.1 Custom calculators

Lagrangian 2D and 3D path fields provide custom calculators based on user define formulae. These formulae can include the path and time variables as well as the field variables such as velocity.

The formulae can include a number of special functions that are worthy of mention here. The functions `aveP()`, `aveT()` and `avePT()` provide averages of their arguments over all paths, over time and over both paths and time respectively. The function `corr()` computes a normalised auto-correlation of its argument on a path. This is given in equation 16.1.

$$\text{corr}(u(p, \tau)) = \frac{\overline{u(t)u(t + \tau)}}{\overline{u(t)}^2} \quad (16.1)$$

where an overbar indicates a time average on path p , and τ is the time offset used in the correlation calculation. For this function the time variable becomes the time offset. Thus this function will always evaluate to 1 for $t = 0$ (i.e. for no time offset). All other values will

be less than or equal to 1. As the t variable is now the time offset, even though different paths start at different times, the correlation function always starts at $t = 0$.

Custom calculators are stored with the Lagrangian field when it is saved.

16.3.2 Two path calculators

Two- and three-dimensional Lagrangian path fields also include special calculators referred to as **two path calculators**. These calculators attempt to mimic the type of data that would be obtained through traditional measurement devices such as potentiometers. A potentiometer is a physical device whose two ends are fixed to a material specimen. As the specimen deforms the relative displacement of the two ends of the device is measured electrically and hence a space-averaged strain in the region of the potentiometer can be deduced. The two ends of a potentiometer are exactly equivalent to particles located to the surface of a material specimen. Two path calculators mimic the potentiometer by the simple selection of two paths to represent the two ends of the device. The calculator can provide the same information as the potentiometer and more, because while the potentiometer cannot detect rotation of the device orientation, the two path calculator can.

It is perhaps worth dwelling briefly on the wealth of information such calculators can provide. Consider a specimen tracked using 1000 particles. Each pair of particles can be treated as a potentiometer. So, excluding duplicate path pairs, this generates 499,500 unique pseudo potentiometers.

16.4 Processes

A number of processes are defined to operate directly on a Lagrangian path field. Some are only available for two-dimensional path fields.

16.4.1 Join Lagrangian paths (2D & 3D)

A **join Lagrangian paths process** allows the user to try to connect discrete Lagrangian paths that are perceived as corresponding to the same particle. Typically one path will finish at a particular time and

location and the second path will start nearby, physically, within a few frames of the end of the first path. The size of the particle might help confirm that the two particles are indeed the same one.

16.4.2 Transform pipeline (2D)

A transform pipeline comprises a sequence of transforms that operate on a Lagrangian 2D path field. Currently three types of transforms are available. These are:

Swap x (2D)

This transform changes the sign of the x coordinate of all particles in the path field.

Swap y (2D)

This transform changes the sign of the y coordinate of all particles in the path field.

Transform coordinates (2D)

This transform is identical to the transform coordinates transform for a particle 2D record. It uses a two dimensional transformation, including both time-dependent translations and rotations, to adjust the particle coordinates. Specific free processes are available to generate the rotations and translations used in this transform. More details can be found in the *User's Guide: Core objects* manual.

16.4.3 Save path field (2D & 3D)

The **save Lagrangian path field process** is a useful component of a pipeline where the result of a previous process, the creation of a Lagrangian path field, needs to be stored to disk.

16.5 Views

The Lagrangian path field has four views.

16.5.1 Calculator view

The calculator view provides the user with access to the predefined and custom calculators for the path field.

16.5.2 Details view

The details view provides information about the spatial and temporal extent of the field. None of this information can be changed by the user. The view also displays the field name and its description. The latter can be edited in this view.

16.5.3 Path view

The path view is the default view. It has much in common with the particle 2D and 3D record particle views as it enables the user to visualize the particle paths and explore them in a little detail. The view provides a video player interface that allows the particle paths to be displayed as a function of time. However the actual data displayed differ from that in the particle record particle view. Instead of displaying the particles that are present in the range of frames selected by the user, here all paths that exist during one of these frames are displayed. Thus as the view is played some paths will simply remain static until the path no longer exists for the range of frames displayed.

Particle paths are displayed using a particular colour coding. The portion of the path that exists within the range of frames selected is shown in red. The earlier portion of the path is displayed in yellow and the later portions of the path are shown in blue. As an example consider a particle path that starts in frame 10 and ends in frame 20. Suppose that the user is currently displaying frames 12-15. The portion of the path from frames 10-12 will be rendered in yellow, that from frames 12-15 will be in red and the remainder will be in blue.

The particles displayed in the view can be shown as points or as ellipses representing their true shape and orientation. In addition the user can choose to colour the particles according to their intensity. Velocity vectors can be drawn at each particle location.

A popup menu allows the user to display path information for paths that cross a particular region of the physical domain, as well as to eliminate paths within these regions.

The view is intended for exploration of the field and to provide as many tools as possible to display useful information about the particle paths.

16.5.4 Path list view

The path list view displays details of the paths in tabular form. These details include the time at which the path started, the length of the path in number of frames, the physical start and end point of the path, the distance travelled by the particle along the path and the average velocity of the particle during its motion.

16.5.5 Process view

The process view matches that for all other entities. It differs only in the list the primary and secondary processes that can be created.

16.5.6 Selected path list view

The selected path list view displays, in tabular form, details of the paths that have particle locations in a specified region in the path view. These details include the time at which the path started, the length of the path in number of frames, the physical start and end point of the path, the distance travelled by the particle along the path and the average velocity of the particle during its motion.

16.6 Summary

This chapter has provided a very brief overview of Lagrangian 2D and 3D path fields. These objects store particle data along particle paths.

17 Technical Issues

The previous chapters have described the various *Streams* subsystems. In this final chapter some of the technical issues involved in the *Streams* system are discussed.

17.1 The Java environment

Streams 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 Development Kit (JDK) is installed on the system. If your computer does not have a JDK installed then an appropriate JDK (this is platform dependent) can be downloaded from Oracle's Java website. The address is

<http://www.oracle.com/technetwork/java/index.html>

Note that you will need to register as a java user to be able to download the JDK. Also, note that the Java Runtime Environment (JRE) is not sufficient to run *Streams* as some of the required tools are only available in the JDK.

Version 1.8.0 or later of the JDK is required. Unlike previous versions of *Streams* this is the case for both PC and Mac versions. Therefore a JDK for Mac OS X must be downloaded from the Oracle[®] website as Apple does not provide support for the more recent versions of Java.

Most computer systems will not have the JDK installed so installation of the JDK will generally be a requirement to run *Streams*.

Streams has been developed in both a Windows[®] and a Mac[®] environment, and tested extensively on computers running the Windows 7[®] and OS X[®] operating systems. It has also been tested on computers running Linux[®].

17.1.1 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 *Streams* have performed some simple tests in order to compare the computational performance of Java with that of C++. A number of *Streams*' core algorithms 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.

An issue encountered since the advent of Retina® displays on Macintosh computers is sporadic poor video performance on Mac computers with such displays. This can be particularly noticeable when multiple animated views are open at the same time (not necessarily running) but occurs at other times as well. It is hoped that this is a temporary issue that will be resolved by Apple and Oracle in due course.

17.1.2 Memory Management

The processing of large numbers of image files, and the manipulation of millions of particles, places heavy demands on the memory, and processing power, of the computer. Certainly computer systems with large storage capacity and high performance processors are most desirable.

As of writing 64bit operating systems are the norm and 64bit Java is now standard. Tests on Mac OS X running 64bit Java show that memory should become much less of an issue in future. These systems nominally have the ability to access 16Tb of RAM – a limit that will not hamper image processing applications for some time one would hope. In reality the actual physical memory on these systems is likely to be in the 8-32Gb range in the near future.

Because of the large datasets typical of particle tracking analyses the management of memory within *Streams* is of paramount importance. When the application is first launched the user specifies the amount of memory (RAM) they wish to allocate to the system. This memory is the total available to the user – access to virtual memory on disk is

not possible. Thus if the user has memory demands in excess of that allocated the program becomes starved of memory and fails. All unsaved data will be lost.

The primary ramification of this is that the user must be aware of the memory usage within the system and manage it carefully. A number of tools are provided within *Streams* to help the user in this respect. These are described at the end of this subsection.

First it is useful to briefly review how the Java programming language handles memory. Memory is dynamically allocated from within the total pool available when it is required, but recovery of this memory, once it is no longer needed, is somewhat beyond the programmer's control. Instead, Java has an automatic garbage collection mechanism that recovers memory automatically and returns it to the pool. While this relieves the programmer of considerable effort in memory management, it also leaves the responsibility of recovering memory with the Java Virtual Machine. If the virtual machine does not perform this task in a timely fashion it is also possible for the application to become memory starved and to fail.

The user can ask the garbage collector to run by selecting reclaim memory from the *Streams* application menu.

To help the user manage the memory pool a toolbar at the bottom of the main window includes a memory monitor that displays the current amount of memory available as a numeric value (in megabytes) and a slider that indicates the percentage of the total memory that is unused. By clicking on the slider the user can invoke the garbage collector. This has the same effect as selecting reclaim memory from the application menu.

In addition, when a memory intensive activity is to be undertaken *Streams* will attempt to determine whether sufficient memory is available for the requested operation. This most commonly occurs when the user attempts to load one or more files from disk. If the system estimates that the memory demand is excessive a dialog box will appear warning the user of the likely difficulty. The user may choose to override this warning but if they do so they should ensure that all data currently within the system is saved.

While the user can take steps to ensure that *Streams* never exhausts its memory supply it is still possible for this to happen. For this

reason it is important that the user regularly saves current data to disk.

Our recent experiences with memory intensive processes indicate that the latest versions of java do not exhibit memory management issues within the virtual machine and hence we have enhanced confidence in its performance in this area.

17.2 Application basics

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

17.2.1 Configuration file

Streams creates a configuration file in the user's home directory when *Streams* is registered and run for the first time (see below). On subsequent occasions *Streams* 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 as well as others that cannot be changed. These can be accessed through the Application/Properties... menu of the main application view. This displays a dialog box with four tabs. These are:

Registration

- *Registered on* – the date the user registered the *Streams* application. This cannot be changed by the user.
- *Registration expires on* – the date the registration expires at which point a new registration code is required. This cannot be changed by the user.

Directories

- *Home directory* – the home directory of the user. This cannot be changed by the user.
- *Manual directory* – the user can specify a directory in which the *Streams* manuals are located. This directory is set through the Manual menu of the application view. It is visible here.
- *Current directory* – when the user accesses the file system, for example to open or save a file, the current directory will be the initial directory

accessed. This is updated each time the user uses the file system and is saved when the user exits.

- *Browse current directory* – the user can change the current directory manually by clicking this button.

Interface

- *Look and feel* – this combo box provides the user with a number of “look and feels” which change the way the interface looks. The default look and feel will always be that of the operating system on which *Streams* is running but the user is at liberty to change this if they wish.
- *Use normal frames* – this check box is only relevant if *Streams* is running on a Mac. Windows on a Mac are normally displayed with a small vacant border around them. If this check box is ticked this border is removed and windows are displayed with no gap between them.

System

- *Minimum memory level (bytes)* – when the memory available in *Streams* falls below this level during certain operations the garbage collector will be invoked.
- *Garbage collector wait time (ms)* – when the garbage collector is involved *Streams* may freeze the system for the given period of time so the garbage collector (which is a low priority task) can execute.

17.3 Installation, execution, and registration

Installation, execution, and registration instructions for *Streams* are available in a separate document, ***Streams 3.03 – Installation, Execution and Registration***, downloadable from the *Streams* website

<https://www.canterbury.ac.nz/engineering/schools/cnre/software/streams/>

18 Summary

This manual has presented an overview of the theory and design of *Streams*, an image processing and PTV application developed in the Department of Civil and Natural Resources Engineering at the University of Canterbury.

The following areas have been covered

- an overview of image processing and PTV,
- practical issues of lighting and image capture,
- operational and system issues and computer system requirements,
- the user interface model,
- entities and processes,
- image sequences, image processing and particle identification,
- particle records, including creation by theoretical models,
- particle matching and its optimisation,
- the creation of Eulerian velocity, intensity, velocity-intensity, density, displacement and shape field time series, and
- the creation of Lagrangian path fields.

Nearly all of the specifics relating to the operation of *Streams* can be found in the two *User's Guides*. They include detailed information on parameter definitions and selection, as well as descriptions of *Streams* objects and their associated views.

The user will find that effective use of *Streams* requires experience in its operation, as well as a firm understanding of its various components. We would encourage the user to spend some time exploring the *Streams* environment before using it in earnest.

19 References

- Bertsekas D P 1992, “Auction algorithms for network flow problems: a tutorial introduction”, *Report LIDS – P – 2108*, Laboratory for Information and Decision systems, MIT, Cambridge, Mass pp 54.
- Campagnol J, Radice A, Nokes R, Bulankina V, Lescova A and Ballio F, 2013, “Lagrangian analysis of bed-load sediment motion: database contribution”, *Journal of Hydraulic Research*, **51**, 5, pp 589-596.
- Cenedese C, Nokes R and Hyatt J, 2016, “Lock-exchange gravity current over rough bottoms”, *Environmental Fluid Mechanics*, DOI [10.1007/s10652-016-95010-0](https://doi.org/10.1007/s10652-016-95010-0)
- Cline A K and Renka R L, 1984, “A storage-efficient method for construction of a Thiessen triangulation”, *Rocky Mountain Journal of Mathematics*, **14**, No. 1, pp 119-139.
- 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
- Fernando H J S (Ed), 2013, “Handbook of Environmental Fluid Dynamics, Volume Two”, *CRC Press/Taylor & Francis Group, LLC*. ISBN: 978-1-4665-5601-0.
- Heays K, Friedrich H, Melville B and Nokes R, 2014, “Quantifying the dynamic evolution of graded gravel beds using particle tracking velocimetry”, *Journal of Hydraulic Engineering*, **140**, No. 7.
- 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.

Kikkert G A, Davidson M J, Nokes R I, 2010, “Buoyant jets with three-dimensional trajectories”, *Journal of Hydraulic Research*, 48, 3, pp292-301, DOI: 10.1080/00221681003726270.

Maas H, Gruen A and Papantoniou D, 1993, “Particle tracking velocimetry in three-dimensional flows. Part I: Photogrammetric determination of particle coordinates”, *Experiments in Fluids*, **15**, pp 279-294.

Malik N, Dracos Th and Papantoniou D, 1993, “Particle tracking velocimetry in three-dimensional flows. Part II: Particle tracking”, *Experiments in Fluids*, **15**, pp 279-294.

Nikora V, Nokes R, Veale W, Davidson M, and Jirka G, 2007 “Large-scale turbulent structure of uniform shallow free-surface flows ” *Environmental Fluid Mechanics*, **7**, 2, pp 159-172.

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, 2021, “A practical guide to particle tracking systems – incorporating Streams 3.03.” to appear.

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, Kompenhans J, 1998, “Particle Image Velocimetry”, *Springer-Verlag*, Berlin, Germany, pp 253.

Renka R L and Cline A K, 1984, “A triangle-based C1 interpolation method”, *Rocky Mountain Journal of Mathematics*, **14**, No. 1, pp 223-237.

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.

Wilson D 1981, "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes", *The Computer Journal*, **24**, No. 2, pp 167-172.