CSIRO PUBLISHING

www.publish.csiro.au/journals/pasa *Publications of the Astronomical Society of Australia*, 2007, **24**, 159–173

SPLASH: An Interactive Visualisation Tool for Smoothed Particle Hydrodynamics Simulations

Daniel J. Price[A]

[A] School of Physics, University of Exeter, Exeter EX4 4QL, UK.
Email: dprice@astro.ex.ac.uk

Received 2007 June 19, accepted 2007 August 31

Abstract: This paper presents SPLASH, a publicly available interactive visualisation tool for Smoothed Particle Hydrodynamics (SPH) simulations. Visualisation of SPH data is more complicated than for grid-based codes because the data are defined on a set of irregular points and therefore requires a mapping procedure to a two dimensional pixel array. This means that, in practise, many authors simply produce particle plots which offer a rather crude representation of the simulation output. Here we describe the techniques and algorithms which are utilised in SPLASH in order to provide the user with a fast, interactive and meaningful visualisation of one, two and three dimensional SPH results.

Keywords: hydrodynamics — methods: numerical

1 Introduction

Smoothed Particle Hydrodynamics (SPH, for recent reviews see Price 2004; Monaghan 2005) is a Lagrangian particle method for solving the equations of fluid dynamics. It has found widespread use in astrophysics due to the ability to simulate complicated three dimensional flow geometries and free surfaces with relative ease and the natural coupling with N-body techniques for self-gravitating problems. For example, SPH is used widely for simulations of cosmological structure formation (e.g. Frenk et al. 1999; Springel 2005), for problems related to star (e.g. Bate et al. 2003) and planet (e.g. Mayer et al. 2002) formation and in simulating astrophysical accretion discs (e.g. Smith et al. 2007) and stellar collisions (e.g. Freitag & Benz 2005; Dale & Davies 2006) and publicly-available SPH codes such as GADGET-2 by Springel (2005) have found widespread application.

However, visualisation of SPH data is not a straightforward process, since the data are defined on a set of moving points which follow the fluid motion and derivatives are evaluated by interpolation from neighbouring points weighted by a smoothing kernel. In practise many authors simply present particle plots which are a rather crude representation of the data. For example the widely used and publicly available TIPSY[1] visualisation tool, though written for N-body simulations, is often used for SPH visualisation where the only procedure possible is to colour the particles according to the value of a scalar field such as density.

A faithful visualisation of SPH data is much more complicated than for grid-based fluid codes since, for a smooth representation, a mapping procedure from the particles to a two dimensional array of pixels is required. Using commercial visualisation packages (e.g. IDL) for this procedure is often inefficient because, for example, they require simply interpolating to a 3D grid first rather than mapping directly from the particles to the two dimensional pixel array required for a particular visualisation. Also, given that interpolation lies at the heart of SPH, consistency suggests use of the same interpolation algorithms as part of the visualisation procedure. Because fluid particles in SPH preserve their identity, there are also certain visualisation procedures which are possible which cannot be used in an Eulerian context, such as tracing the history of a portion of the flow via its component particles and tracking of particular objects. These aspects of SPH visualisation give strong motivation for a dedicated software tool designed to visualise SPH data using SPH algorithms. This paper presents the software design and algorithms implemented in exactly such a tool, which we have called 'SPLASH'.

SPLASH differs from other visualisation tools because it is designed specifically for SPH visualisation and works both interactively and non-interactively (see the discussion relating to the software design below). For example IFRIT[2] is a publicly-available tool written to visualise ionisation fronts in cosmological simulations (including those using particles) but allows only an interactive visualisation and lacks many of the features of SPLASH such as the ability to visualise in one, two and three dimensions, to select and hide particles and to track portions of the flow across multiple dump files. SPLASH allows plotting to both interactive and non-interactive devices allowing both a mouse-click driven visualisation as well as a 'pipeline'

[1] http://www-hpcc.astro.washington.edu/tools/tipsy/tipsy.html

[2] http://home.fnal.gov/~gnedin/IFRIT/

10.1071/AS07022 1323-3580/07/03159

mode for producing the same visualisation from a series of dump files (without the need for any kind of scripting). Similarly SPLOTCH[3] is a raytracing utility to visualise SPH simulations in a manner similar to the 'surface rendering' technique implemented in SPLASH (see Section 3.2) but does not allow other visualisation techniques and does not have any interactive capabilities.

Other publicly available tools such as SUPERMONGO and GNUPLOT implement primitive plotting functionality at a much lower level and would require a script of similar length to the SPLASH source code to achieve similar functionality in terms of visualising SPH data (equivalent to SPLASH's use of the PGPLOT library for actually plotting the results of the rendering operations). SPLASH can also be used to visualise remotely from the same location as the data are produced (e.g. on a remote supercomputing facility), installation on which is straightforward since the only requirement is a FORTRAN compiler which can also be used to compiler the PGPLOT libraries. Using a commercial package, this would not always be possible because it would require the remote facility to have the appropriate license (this in particular applies to IDL). Furthermore many visualisation tools require some form of scripting to achieve the desired functionality (in IDL's case, to the level of an entire programming language). Since SPLASH is specifically tailored to visualise SPH simulations with settings changed via a series of command-line based menus, no scripting is required even for complicated tasks such as producing a sequence of plots from multiple dump files (either interactively or non-interactively).

The paper is organised as follows: In Section 2 we discuss the basic requirements driving the software design and present the design in detail; in Section 3 we discuss the basic methods for visualising SPH data and how these are incorporated into SPLASH and in Section 4 we discuss the details of the interpolation algorithms implemented. Some additional features are described in Section 5 and the code's performance and memory usage are described in Section 6. A summary is given in Section 7.

2 Software Design

The basic requirements I set for an SPH visualisation tool (based largely on my own experience of performing SPH simulations) were the following:

1. Capable of producing sufficiently annotated, appropriately labelled figures suitable for inclusion in research papers.
2. Capable of producing a sequence of images for making animations.
3. Capable of reading data directly from binary code dumps from users' SPH codes.
4. Visualisation of SPH data in 1, 2 and 3 dimensions.
5. Algorithms should be consistent with the basic SPH method.
6. Should be easy to apply the same visualisation to different dump files (either interactively or non-interactively).
7. Visualisation of both scalar and vector fields defined on the particles.
8. Visualisation should be interactive so the user can rapidly understand the data and find the best representation.
9. Remote visualisation capability, since simulation data are often produced remotely on supercomputing facilities from which data transfer is awkward and time-consuming.
10. Written in a programming language familiar to users.

SPLASH is a program designed to meet these basic visualisation requirements in the most efficient manner possible. Each of the above requirements have strongly constrained the software design. For example the requirement that the visualisation be interactive means that simple but inefficient procedures such as interpolating from the particles to a 3D grid before using standard grid-based visualisation techniques cannot be utilised.

The basic software design which achieves all of the above is outlined in Figure 1. The code (written in FORTRAN 90) is built around a command-line menu structure (designed so as to meet the requirement for remote visualisation) with the actual plotting performed via the PGPLOT graphics subroutine library[4] (thus satisfying the requirements for production of figures for papers via the postscript device drivers; for movies via bitmap device drivers such as PNG and GIF; for interactivity via interactive devices such as the X-windows driver). The use of a graphics library not only facilitates the easy reproduction of the same plots on different devices but also means that SPLASH can be focussed on the data-input and manipulation side of the visualisation procedure rather than the implementation of primitive plotting functionality.

Plot settings are changed either non-interactively via a series of sub-menus accessed on the command line from the main menu; or interactively using the mouse and/or pressing particular keystrokes with the cursor in the plotting window (this is the 'interactive mode' indicated in Figure 1).

Rather than requiring the user to convert data to an intermediate format (e.g. ASCII files), data are read directly from the binary code dump files – this is a crucial requirement for rapid visualisation and makes for significantly reduced disk space requirements (since no intermediate storage is required), which can be a major constraint on many systems for simulations involving $\gtrsim 10^6$ particles. The filenames are read from the command line, making it easy to read all files from a simulation by using wildcards (e.g. '*splash dump**'). Read routines are supplied for several widely used SPH codes (e.g. GADGET, Springel 2005; VINE, Wetzstein in preparation; and Matthew Bate's SPH code, Bate 1995). Optionally, a further set of derived

[3] http://dipastro.pd.astro.it/~cosmo/Splotch/

[4] http://www.astro.caltech.edu/~tjp/pgplot

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

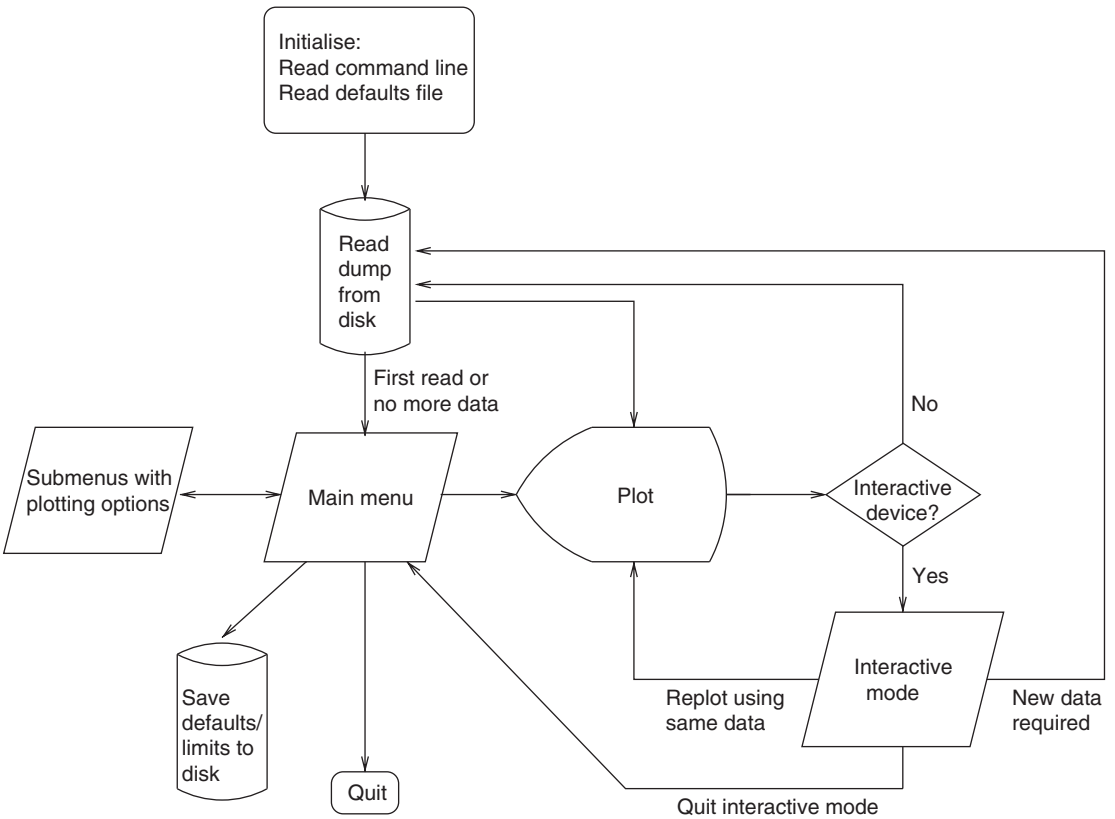

Figure 1 Basic software design.

quantities can be calculated from the data read. For a typical SPH data set this would include the radius, the magnitude of all vector quantities and the entropy. These quantities appear as 'extra columns' as if they had been read from the dump file.

The first file listed on the command line is read on entry (see Figure 1) and this determines the basic parameters used for the visualisation such as the number of data columns, column labels and unit settings where appropriate. The data are then listed by column in the main menu, where the column number corresponds to the position of a variable in the data read. This means that any two parameters can be plotted against one another (for example density versus x would be plotted by typing 6 for the y-axis and 1 for the x-axis assuming column 1 contains the particle x-position and column 6 contains the density). Where two of these columns correspond to particle coordinates we refer to this as a 'coordinate plot' which (provided particle masses, density and smoothing lengths have been read from the dump file) can be plotted either as particle plots or with a third quantity 'rendered' to a pixel array. Thus, for example, a plot of density in a two dimensional domain is a plot of y versus x with density rendered. If vector quantities are present in the data (specified in the data read corresponding to that particular data format) a fourth quantity can also be plotted over the rendered plot in the form of an arrow plot. In 3D these plots can either be projection (using all particles) or cross section plots (using only particles contributing to a slice positioned in the third coordinate direction). Similarly two dimensional 'rendered' plots are either plots using all of the particles or line plots tracing an oblique cross section through the computational domain. The interpolation procedures used to map from the particle data to a rendered image are described below and the algorithms are presented in Section 4.

The plotting is directed to a particular device via a PGPLOT prompt. For interactive devices, the program then enters 'interactive mode', where the user can manipulate the data interactively either using the mouse (to zoom, change colour bar limits, select and colour particles and move legend positions) or via keystrokes pressed in the plot window, giving access to a wide range of options such as rotating the particles, moving the 3D observer, adapting plot limits, plotting smoothing circles, labelling particles, changing the colour scheme, adjusting the length of arrows on vector plots, setting up animation sequences, finding the gradient of a line and (most importantly) moving forwards and backwards through timesteps. For example pressing the space bar moves forwards to the next dump file, whereupon the same plot is repeated (and repeatedly pressing the spacebar produces a crude 'animation' dependent of course, on the speed at which data can be read from disk and plotted). This is indicated by the loop in Figure 1 which proceeds from interactive mode via the data read back to the 'plot' step and finally returning to interactive

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

mode. Where no data read is required the plot is simply re-plotted with the changed settings (perhaps recalculating the interpolation to pixels where necessary).

A key feature facilitating the easy production of animations is that, when plotting is directed to a non-interactive device, the plotting cycles automatically through all of the dump files on the command line. This is indicated by the loop in Figure 1 proceeding from the 'plot' step back to the data read (if the device is non-interactive) and returning plot the same figure for the next dump file with settings unchanged.

The settings for a particular plot can be saved to disk by pressing 's' from the main menu (see Figure 1). This saves a file in the current working directory containing (in FORTRAN 90 NAMELIST format) all of the current plot settings. This file is then read automatically on the next invocation of SPLASH such that plot settings can be restored. A 'full save' (implemented by pressing 'S' from the main menu) saves both the plot settings and the current minimum and maximum limits set for each column (in a simple two-column ASCII file), so that *exactly* the same plot can be reproduced on the next invocation of SPLASH. Additional files are also saved where physical units have been applied to the data columns or animation sequences have been set.

The plot settings are structured into FORTRAN 90 modules which contain the parameters which may be changed via a particular submenu together with the subroutine implementing the submenu itself. Each settings module contains it's own namelist for those parameters which should be saved to disk. Thus the 'save' operation simply saves all of the namelists in order into a single file. This structure means that, for the programmer, it is a straightforward task to add additional menu options affecting particular plotting functions (e.g. settings related to vector plots are changed in a 'vector plot options' submenu and both the settings and the submenu are contained in the same FORTRAN 90 module. This module is then used only in the subroutines which implement the plotting of vector plots, so any parameters changed via options in the vector submenu will be automatically available near where they will be used to make plotting decisions and automatically saved to the defaults file, provided they have been added to the namelist).

3 Plot Types

The 'central engine' of the visualisation procedure is encapsulated in the 'plot' step in Figure 1. An expanded outline of this step is shown in Figure 2. There are essentially two types of plots: particle plots or rendered plots, where a further rendered plot of vector arrows can be plotted on top of either of these. The procedure for each of these is described in turn below. Note that transformations such as log, rotation, 3D perspective and change of co-ordinate systems are applied to the particle data prior to calling any interpolation routines.

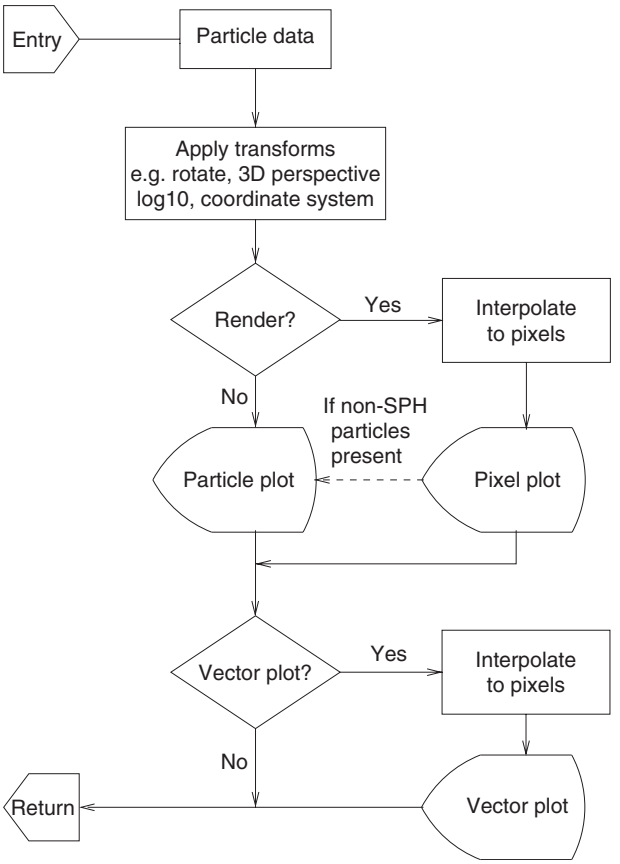

Figure 2 The plotting pipeline.

3.1 Particle Plots

If rendering is not being used (i.e. the plot is not a coordinate plot or no third quantity has been selected), the plot can simply proceed by plotting the particle positions directly on the plotting device (Figure 2), using markers which can be chosen dependent on the particle types (set via the submenus accessed from the main menu, see Figure 1). A simple particle plot example is shown in the top panel of Figure 3. Particle colours can be changed in a variety of ways. For example, selecting particles with the mouse and pressing keys 1–9 whilst in interactive mode colours the selected particles with colours corresponding to the respective PGPLOT colour indices. Since colours allocated to particles are retained in all subsequent plots, this can be used to select ranges of a particular parameter (e.g. by selecting particles on a density-versus-x plot) with the colours still appearing on a different plot (e.g. a coordinate plot of y versus x). Similarly particles can be coloured using data from a dump corresponding to the initial conditions and, provided particles retain their identity between dumps, the same particles will still appear coloured when plotted in subsequent dumps.

Particles can also be coloured according to the value of a particular quantity by setting an option which renders via particle colours instead of interpolating to pixels, although the latter method (see below) is strongly preferred as a method of visualisation. However there are circumstances

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

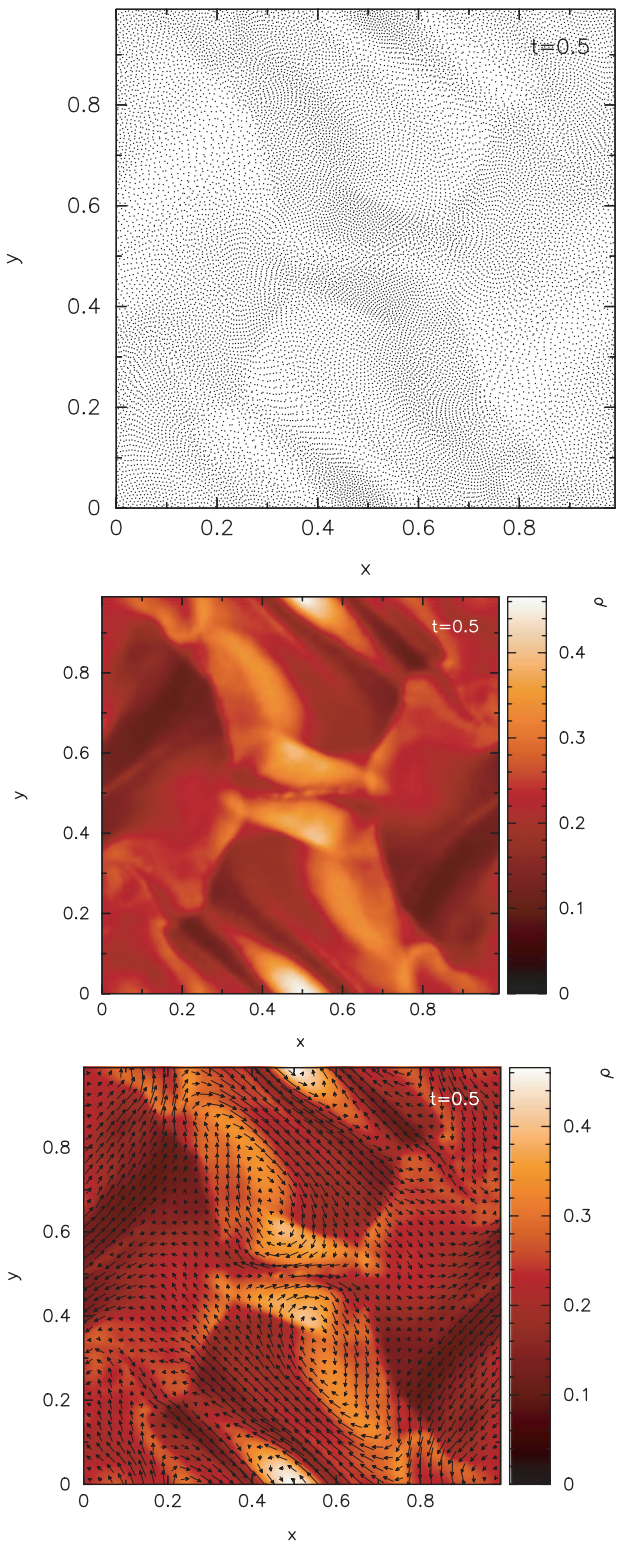

Figure 3 Top panel: A simple particle plot produced from a 2D simulation simply by plotting all of the particle positions. Middle panel: The same data but plotted with data rendered to a pixel array instead of plotting particles. Bottom panel: with a vector plot additionally overlaid (in this case showing the magnetic field in the simulation).

where it may be desirable to see the actual values of a quantity on the particles themselves.

Where the 'cross section' option has been set from the menu, particles are plotted in a thin slice of finite (although user-adjustable) thickness around the user-defined cross section slice position.

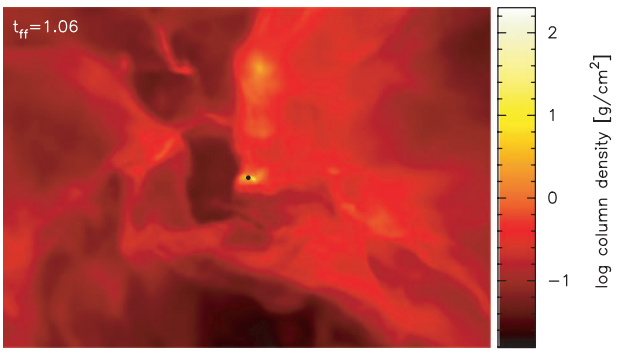

Figure 4 Visualisation of a large scale star cluster formation calculation (Price & Bate 2007) via a 3D rendered plot showing the density integrated through the z-direction.

3.2 Rendered Plots

'Rendered' or 'pixel' plots proceed in a similar manner to particle plots but with an intermediate step where the particle data are interpolated to the two-dimensional pixel array corresponding to the viewing surface (Figure 2). In 3D rendered plots are either projections (integration along the line of sight), cross section slices or surface rendered plots (see Section 4.3.3). In 2D the plot can either be a projection (a straight interpolation to a 2D pixel array) or a cross section (a 1D line plot drawn arbitrarily through the 2D domain). Rendered plots do not apply in the case of 1D data. An example of a 2D rendered plot is shown in the middle panel of Figure 3, where the same data shown in the particle plot (top panel) has been used. Note the striking difference between the visualisation using pixels compared to the raw particle plot (this kind of plot is often used as a representation of the density field). One of the goals of SPLASH is to make visualisation of SPH data in this manner a straightforward task for the user.

A slight complication here is that often simulations contain particles of multiple types, some SPH (e.g. different types of gas particle) and some non-SPH (e.g. sink or N-body particles). In this case the interpolation is performed using all of the available SPH particles, provided plotting of that type has been turned on via the submenu options. Particles of non-SPH types can optionally be plotted on top of rendered plots (e.g. sink particles appear on top of a rendered plot of gas density). This is indicated by the dashed pathway in Figure 2. An example of a three dimensional rendered projection (i.e. showing in this case column density) of a large scale star formation calculation (similar to that described in Bate et al. 2003) is shown in Figure 4, where additionally a sink particle has been plotted over the rendered array.

One further type of rendering is available in SPLASH for three dimensional data, which we refer to as 'surface rendering' (the algorithm is described in Section 4.3.3, below). This type of rendering provides an 'optically thick' view of the particles (as opposed to the 'optically

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

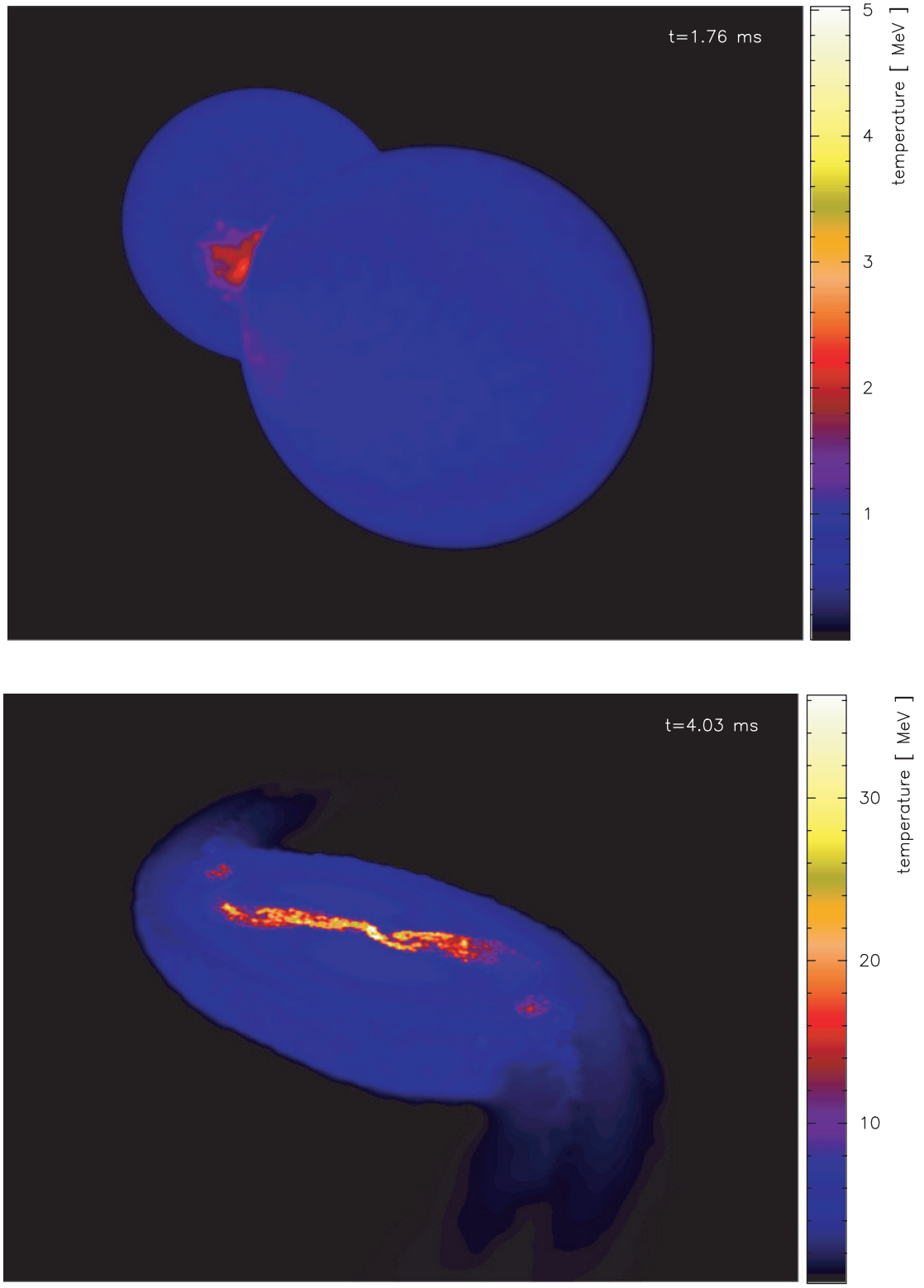

Figure 5 Visualisation of a neutron star merger calculation (Price & Rosswog 2006) via a 3D surface rendered plot showing the temperature on a 'surface of last scattering'. The top panel shows results near the start of the simulation using all of the particles, whereas in the bottom panel only particles below the mid-plane have been used in the interpolation, producing a 'cut-away' effect.

thin' view provided by the column integrated rendering), showing the value of a particular parameter on the 'surface of last scattering', determined by a user-defined opacity which is proportional to the particle density. Generally this type of visualisation works best for simulations where there is a well-defined surface and/or the range of densities in the simulation is not too high. An example is shown in Figure 5 showing gas temperature during the

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

merger of two neutron stars similar to those described in Price & Rosswog (2006). The top panel shows a surface rendering near the start of the simulation where all of the particles have been used in the interpolation. The bottom panel shows a similar plot but where only particles below the midplane have been used in the calculation, giving a 'cut-away' effect.

3.3 Vector Plots

Vector quantities are visualised using arrow plots, although more advanced visualisations may be possible in future. Whilst in principle an arrow could be plotted for each SPH particle with length proportional to the value of the vector on that particular particle, these type of plots quickly become cluttered when large numbers of particles are used in the simulation. Thus vector plots in SPLASH are implemented by first interpolating each component of the vector quantity to the two-dimensional pixel array corresponding to the viewing surface, where in 3D the plot can be an integration of each component along the line of sight or where vector arrows are plotted in a cross section slice (depending on whether cross sections or projections have been selected in the menu options, also affecting rendered plots).

An example of a vector plot is shown in the lower panel of Figure 3 where the arrows are shown overlaid on the rendered plot of density (otherwise identical to the middle panel). A preliminary feature has also been implemented whereby streamlines can be calculated for the interpolated vector field and plotted as contours (instead of plotting arrows). As presently implemented (see Section 4.2.4) this works quite well when the vector field is smooth but gives poor results where the field has strong gradients. A strongly desirable feature for future implementation would be an algorithm for tracing three dimensional field lines through SPH particle data.

4 Interpolation Algorithms

4.1 SPH Interpolation

The heart of the SPH method (see e.g. Monaghan 1992; Price 2004; Monaghan 2005 for reviews) is the following identity for an arbitrary function $A(\mathbf{r})$ defined on spatial coordinates $\mathbf{r}$:

$$A(\mathbf{r}) = \int A(\mathbf{r}')\delta(|\mathbf{r}-\mathbf{r}'|)\mathrm{d}\mathbf{r}', \qquad (1)$$

where δ is the Dirac delta function. This integral is approximated in SPH by replacing the delta function with a smooth function W with finite characteristic width h which reduces to a delta function in the limit $h \to 0$, giving the SPH 'integral interpolant' in the form

$$A(\mathbf{r}) = \int A(\mathbf{r}')W(|\mathbf{r}-\mathbf{r}'|,h)\mathrm{d}\mathbf{r}' + \mathcal{O}(h^2), \qquad (2)$$

where the error in the representation of A is of order h^2 provided the kernel function W is even and the kernel function is normalised such that the volume integral of the kernel is unity. This integral is discretised onto the particles by replacing the integral with a summation over neighbouring particles and replacing the mass element $\rho\mathrm{d}\mathbf{r}'$ with the neighbouring particle mass m, i.e.

$$A(\mathbf{r}) \approx \sum_{j=1}^{N} \frac{m_j}{\rho_j} A_j W(|\mathbf{r}-\mathbf{r}_j|,h), \qquad (3)$$

where the subscript j refers to a quantity defined on particle j. The expression given above is the SPH 'summation interpolant', forming the basis of the SPH approach and therefore the basis of the interpolation algorithms used in SPLASH for SPH visualisation. A normalised version of this interpolant is achieved by dividing the result by the interpolation of unity, given by

$$1 \approx \sum_{j=1}^{N} \frac{m_j}{\rho_j} W(|\mathbf{r}-\mathbf{r}_j|,h). \qquad (4)$$

Many different forms are possible for the smoothing kernel W, but the most commonly used is the cubic spline kernel (see Monaghan 1992):

$$W(r,h) = \frac{\sigma}{h^\nu}\begin{cases} 1-\frac{3}{2}q^2+\frac{3}{4}q^3, & 0 \le q < 1; \\ \frac{1}{4}(2-q)^3, & 1 \le q < 2; \\ 0 & q \ge 2 \end{cases} \qquad (5)$$

where $q = |\mathbf{r}_a - \mathbf{r}_b|/h$, ν is the number of spatial dimensions and the normalisation constant σ_ν is given by $\sigma_1 = 2/3$, $\sigma_2 = 10/(7\pi)$ and $\sigma_3 = 1/\pi$. This kernel satisfies the basic requirements that it is Gaussian-like and has smooth first derivatives which tend smoothly to zero as $q \to 2$ and is zero beyond $q = 2$. The quantity h is the smoothing length, which in most astrophysical applications is a spatially variable quantity set in such a way as either to fix (either exactly or approximately) the number of nearest neighbours (Hernquist & Katz 1989; Benz et al. 1990), or via an analytic relation to the (number) density (Monaghan 2002; Springel & Hernquist 2002; Price & Monaghan 2007).

By default the interpolations used in SPLASH are non-normalised. The reason for this is that, at a free surface the normalised interpolation (that is, using Equation 3 and dividing the result by Equation 4) looks odd, whereas an interpolation using Equation (3) falls away smoothly. An example is shown in Figure 6 which shows a cross section slice of density from a three dimensional neutron star merger calculation (Price & Rosswog 2006). The top panel shows the results using a non-normalised interpolation whereas the bottom panel shows the results when the interpolated array is normalised (by dividing by the interpolation of unity). The normalised interpolation performs poorly at the edges, where the effects of individual particle smoothing spheres are visible. However, using a

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

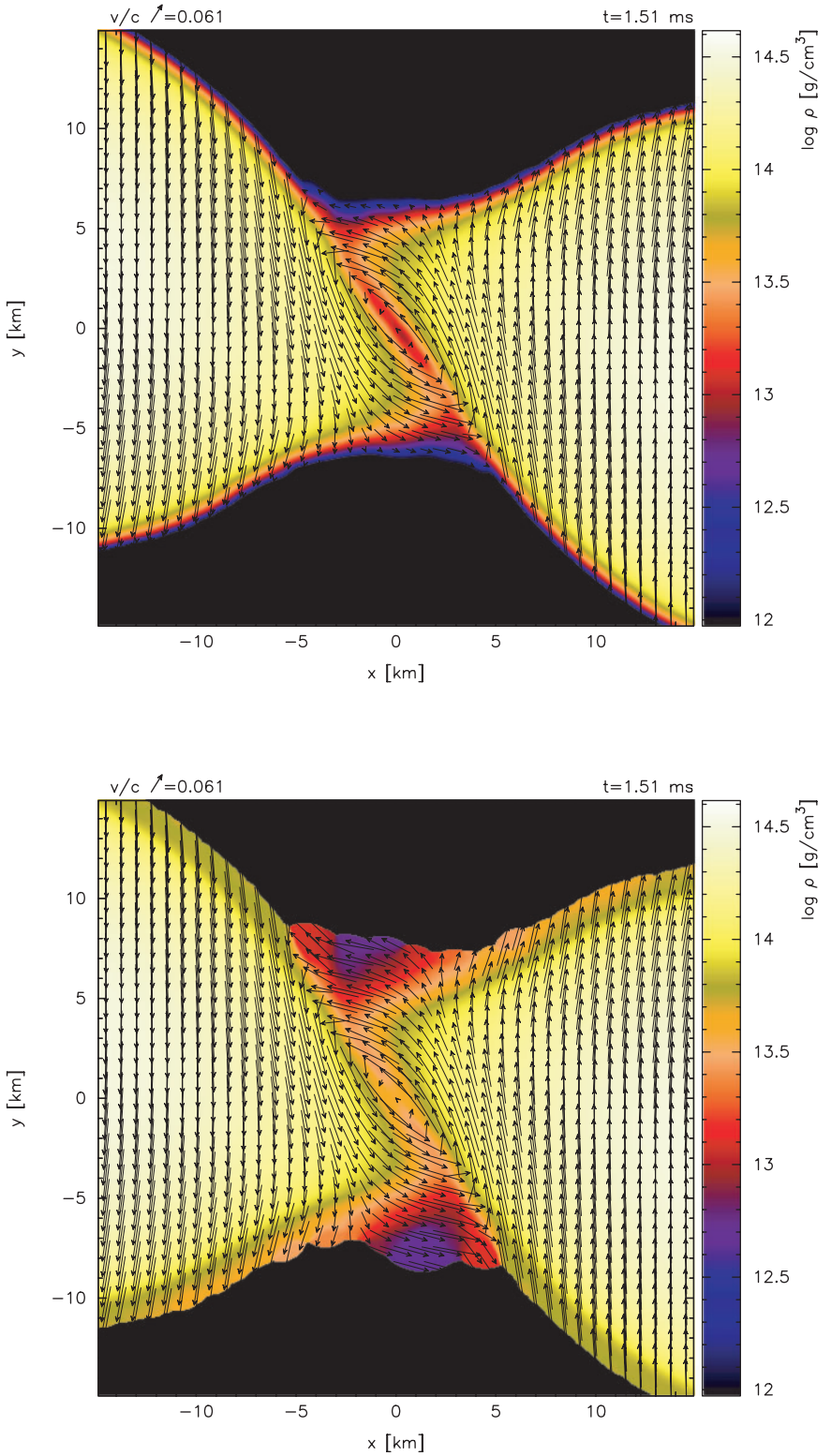

Figure 6 Cross section slice of density (and velocity arrows) in a neutron star merger calculation (Price & Rosswog 2006) showing the difference between non-normalised (top) and normalised (bottom) interpolation. Normalised interpolation is turned off by default as it produces spurious effects due to individual particles at free surfaces (bottom panel).

normalised interpolation improves the accuracy of volume rendered quantities on the pixels by removing effects due to the particle distribution. Thus it is recommended that a normalised interpolation should always be used if there are no free surfaces.

To avoid round-off error in interpolation calculations (done in single precision), we write the summation interpolant in the simpler form:

$$A(\mathbf{r}) \approx \sum_{j=1}^{N} w_j A_j \mathcal{W}(r/h). \qquad (6)$$

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

where w_j is the dimensionless weight given by

$$w_j \equiv \frac{m_j}{\rho_j h_j^\nu}, \tag{7}$$

where ν is the number of spatial dimensions and $\mathcal{W}$ refers to the dimensionless part of the kernel function, such that

$$W(|\mathbf{r} - \mathbf{r}_j|, h) = \frac{1}{h^\nu}\mathcal{W}(r/h), \tag{8}$$

(i.e. we have incorporated the $1/h^\nu$ part of the usual kernel definition into the weight).

With this definition a normalised interpolation is given by

$$A(\mathbf{r}) \approx \frac{\sum_{j=1}^{N} w_j A_j \mathcal{W}(r/h)}{\sum_{j=1}^{N} w_j \mathcal{W}(r/h)}. \tag{9}$$

As an interesting aside, it is worth noting that the usual formula for varying the smoothing length in SPH codes is given by

$$h = \eta \left(\frac{m}{\rho}\right)^{1/\nu}, \tag{10}$$

where η is a constant and ν refers to the number of spatial dimensions. Enforcing this relation rigourously (e.g. as in Springel & Hernquist 2002; Price & Monaghan 2004, 2007) thus corresponds to using constant weights (Equation 7) in the interpolation with the value related to the parameter η. Thus strictly, only knowledge of the (constant) weight value and the smoothing length is required for interpolation of any quantity in these codes.

4.2 Rendering of 2D Data

4.2.1 Interpolation to Pixels

Rendering of 2D data involves a straightforward application of Equation (6) to the interpolation of data from the particles to a two dimensional grid of pixels. Thus we have

$$A(x, y) = \sum_j w_j A_j \mathcal{W}(r/h), \tag{11}$$

where

$$r = \sqrt{(x - x_j)^2 + (y - y_j)^2}, \tag{12}$$

the summation is over contributing particles and we take the smoothing length as

$$h = \max(h_j, \Delta/2), \tag{13}$$

that is, the maximum of the particle smoothing length and half of the pixel width (the latter thus being used generally only when few pixels are used in the interpolated plot). The interpolation is performed as a 'scatter' operation from the particles, that is, for each particle b, we find the range of pixels to which the particle should contribute

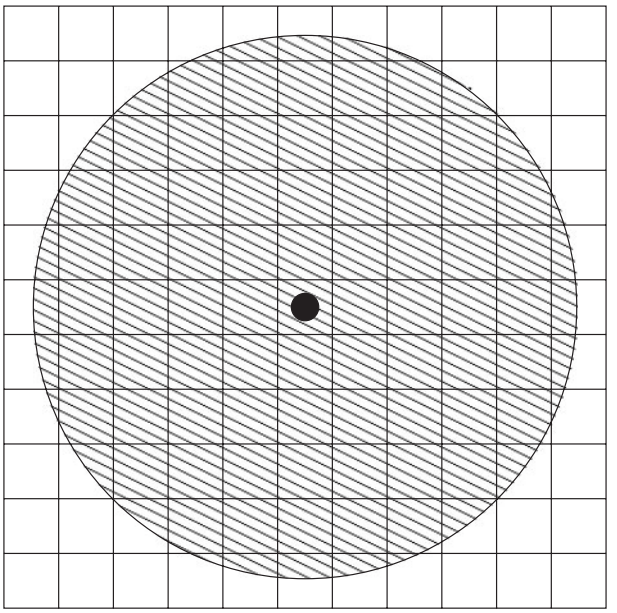

Figure 7 Interpolation of 2D data: for each particle we perform a loop over the pixels (in x and y) to which it contributes, adding the contribution from that particle to the pixel array.

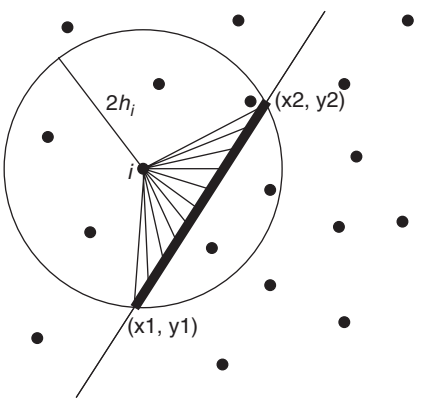

Figure 8 Computation of a one dimensional cross section through 2D data. Each particle contributes to a sequence of pixels along the section of the cross-section line (if any) that intersects the smoothing circle.

(in both x and y) and add the contribution from particle b to all of those pixels. Note that this is much more efficient than attempting to perform the summation over particles in Equation (11) for every pixel. The procedure is illustrated in Figure 7 and examples of 2D interpolation are shown in Figure 3.

4.2.2 Cross Sections of 2D Data

The cross-sectioning algorithm for 2D data (giving a 1D line) is completely general and can be used for arbitrary oblique (or straight) cross sections. The cross section is defined by two points (x1, y1) and (x2, y2) through which the line should pass. These points are converted to give the usual equation for a line

$$y = mx + c. \tag{14}$$

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

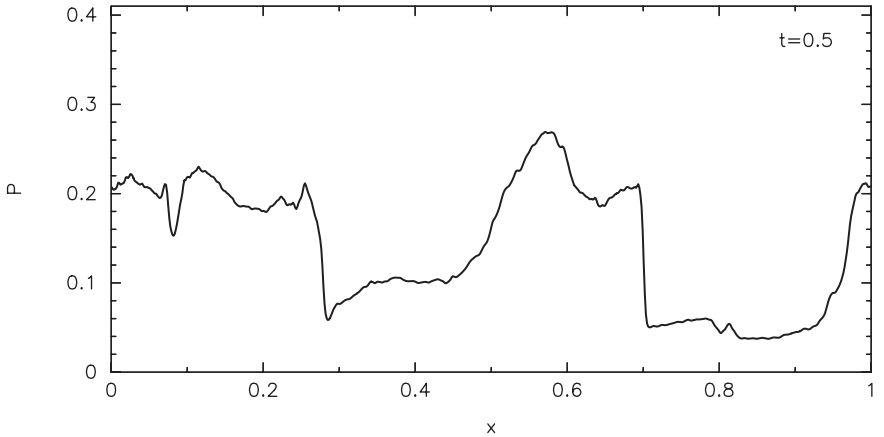

Figure 9 Example of a one dimensional cross-section through 2D data, in this case showing the pressure distribution along a $y = 0.3125$ cut through a high resolution version of the simulation shown in Figure 3.

This line is then divided evenly into pixels to which the particles may contribute. The contributions along this line from the particles is computed as follows: For each particle, the points at which the cross section line intersects the smoothing circle are calculated (illustrated in Figure 8). The smoothing circle of particle i is defined by the equation

$$(x - x_i)^2 + (y - y_i)^2 = (2h)^2. \tag{15}$$

The x-coordinates of the points of intersection are the solutions to the quadratic equation

$$(1 + m^2)x^2 + 2(m(c - y_i) - x_i)x + (x_i^2 + y_i^2 - 2cy_i + c^2 - (2h)^2) = 0. \tag{16}$$

For particles which do not contribute to the cross section line, the determinant is negative. For the particles that do, it is then a simple matter of looping over the pixels which lie between the two points of intersection, calculating the contribution to each pixel using the 1D SPH summation interpolant, i.e.

$$A(x) = \sum_j w_j A_j \mathcal{W}(|x - x_j|/h_j). \tag{17}$$

An example of a 1D cross section through 2D data is shown in Figure 9. In principle a similar method could be used for oblique cross sections through 3D data. In this case we would need to find the intersection between the smoothing sphere and the cross section plane. However in 3D it is simpler just to rotate the particles first and then take a straight cross section as described above.

4.2.3 2D Vector Plots

Vector plots of 2D data are produced by interpolating the x- and y-components of the vector separately to the pixel array, which are then used to plot an array of arrows centred on the pixels, with length proportional to the vector magnitude. Each component is interpolated exactly as for scalar 2D data, i.e.

$$A_x(x, y) = \sum_j w_j A_{x,j} \mathcal{W}(r/h), \tag{18}$$

$$A_y(x, y) = \sum_j w_j A_{y,j} \mathcal{W}(r/h), \tag{19}$$

$$r = \sqrt{(x - x_j)^2 + (y - y_j)^2}, \tag{20}$$

$$h = \max(h_j, \Delta/2). \tag{21}$$

The main difference between interpolation for vector plots and that used for rendered plots is that far fewer pixels are used for the arrow plots (otherwise arrows become indistinguishable). Thus in general the interpolation for vector plots is more like a smoothing procedure rather than an interpolation (i.e. there are far more particles than pixels). Since we only calculate distances to the centres of pixel cells, this is where the minimum smoothing length given by Equation (21) becomes particularly important in providing a smooth representation of the data. An example of a 2D vector plot is shown in the lower panel of Figure 3.

4.2.4 Streamlines

For a two dimensional vector map, streamlines ('field-lines') of the vector field can be plotted by integrating the vector field to find the stream function, contours of which provide the field lines. The stream function is given by

$$\Phi(x, y) = \int v_x(x, y)\mathrm{d}y - \int v_y(x, y)\mathrm{d}x, \tag{22}$$

such that

$$v_x = \frac{\partial \Phi}{\partial y}, \tag{23}$$

$$v_y = -\frac{\partial \Phi}{\partial x}. \tag{24}$$

In SPLASH we compute the integral based on the *interpolated* velocity field on the pixel array using a simple trapezoidal-rule integration. As presently implemented, this procedure works quite well when the vector field is smooth but performs poorly where there are strong gradients present.

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

4.3 Rendering of 3D Data

In three dimensions we must take either a projection through the whole domain or a cross section slice.

4.3.1 Projections (Line-of-Sight Integration)

In the projection case we wish to obtain an integral of the rendered quantity along the line of sight. We begin with the 3D SPH summation interpolant in the form

$$A(x, y, z) = \sum_j m_j \frac{A_j}{\rho_j} W(x-x_j, y-y_j, z-z_j, h_j), \quad (25)$$

where W is the usual (3D) cubic spline kernel Equation (5). Taking the integral of both sides along the line of sight (assumed to be along the z axis) we have

$$\int A(x, y, z)\mathrm{d}z = \sum_j m_j \frac{A_j}{\rho_j} \int W(x - x_j, y - y_j, z - z_j, h_j)\mathrm{d}z. \quad (26)$$

This shows that the line-of-sight integration for three dimensions can be written as a two dimensional interpolation

$$\mathcal{A}(x, y) = \int A(x, y, z)\mathrm{d}z = \sum_j m_j \frac{A_j}{\rho_j} Y(x - x_j, y - y_j, h_j), \quad (27)$$

where the 2D kernel (denoted Y) is the 3D kernel integrated through one spatial dimension, i.e.

$$Y(x, y) = \int W(x, y, z)\mathrm{d}z. \quad (28)$$

For practical purposes we write Y in the form

$$Y(r_{xy}, h) = \frac{1}{h^2} F(q_{xy}), \quad (29)$$

where $q_{xy} = r_{xy}/h$ and $F(q_{xy})$ is the dimensionless 2D kernel given by

$$F(q_{xy}) = \int_{-(R^2-q_{xy}^2)^{1/2}}^{(R^2-q_{xy}^2)^{1/2}} \mathcal{W}(q)\mathrm{d}q_z, \quad (30)$$

where $q_z = z/h$, $q^2 = q_{xy}^2 + q_z^2$, R is the kernel radius (=2 for the cubic spline) and $\mathcal{W}$ is the usual dimensionless kernel function for the cubic spline, i.e.

$$\mathcal{W}(q) = \frac{1}{\pi} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3, & 0 \le q < 1; \\ \frac{1}{4}(2-q)^3, & 1 \le q < 2; \\ 0 & q \ge 2. \end{cases} \quad (31)$$

The integral (30) is not (obviously) tractable analytically (apart from at $q_{xy} = 0$). However it is straightforward to perform this integration numerically (for all q_{xy} from $0 \to 2$) and store the results in a table for the interpolation calculation. This is the method adopted in SPLASH. An alternative would be to use a different kernel in the visualisation for which the above integral can be calculated analytically.

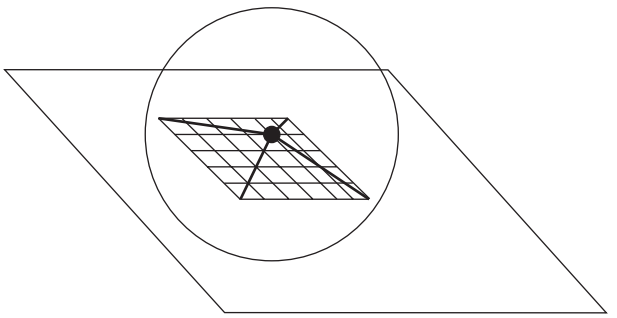

Figure 10 Computation of a two dimensional cross section through 3D data: each particle contributes to pixels in the cross section plane that lie within the smoothing sphere.

As previously, to avoid problems with round-off error we use the dimensionless weights defined in Equation (7), thus writing the final interpolant (as implemented in the code) in the form

$$\mathcal{A}(x, y) = \int A\mathrm{d}z = \sum_j w_j h_j A_j F(r_{xy}/h), \quad (32)$$

where as previously we take

$$h = \max(h_j, \Delta/2). \quad (33)$$

An example of a 3D column-integrated plot is shown in Figure 4, showing the results of a large scale star cluster formation calculation (in this case showing integrated density, i.e. column density).

In the case of vector quantities each component is interpolated separately in the form

$$\mathcal{A}_x(x, y) = \int A_x\mathrm{d}z = \sum_j w_j h_j A_{x,j} F(r_{xy}/h), \quad (34)$$

$$\mathcal{A}_y(x, y) = \int A_y\mathrm{d}z = \sum_j w_j h_j A_{y,j} F(r_{xy}/h), \quad (35)$$

where again

$$h = \max(h_j, \Delta/2). \quad (36)$$

This results in a line-of-sight integrated vector map which can be plotted on top of a rendered plot or as a standalone plot.

4.3.2 Cross Sections of 3D Data

A cross section can be taken of three dimensional data by summing the contributions to each pixel in the cross section plane from all particles within $2h$ of the plane (Figure 10). In the implementation used in SPLASH the cross section is always at a fixed value of the third co-ordinate (i.e. for xy plots the cross section is in the z direction).

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

Oblique cross sections can be taken by rotating the particles first (the combination of settings can be achieved easily in SPLASH's interactive mode by drawing a cross section plane with the mouse, from which the rotation angle and slice position are automatically calculated and the cross section subsequently plotted). The interpolation for cross sections (e.g. in z) takes the form

$$A(x, y, z_0) = \sum_j w_j A_j \mathcal{W}(r/h), \tag{37}$$

where

$$r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z_0 - z_j)^2} \tag{38}$$

$$h = \max(h_j, \Delta/2), \tag{39}$$

and z_0 refers to the position of the cross section slice. As previously, vector plots are achieved by interpolating each component separately. Examples of 3D cross section plots are shown in Figure 6 of both scalar and vector fields.

4.3.3 3D Surface Rendering of SPH Data

A further option for visualisation of 3D data is to use surface rendering (see Section 3.2). The idea is to produce a visualisation of the surface of a data set by performing a ray-trace through the SPH particles, with the density distribution giving the optical depth and the rendered quantity providing the colour. Thus low-density regions will be transparent whilst high density regions will be opaque.

For a homogeneous medium the transport equation for a ray traced from $0 \rightarrow D$ is

$$I_\nu(D) = I_\nu(0)\mathrm{e}^{-\tau_\nu(D)} + S_\nu(1 - \mathrm{e}^{-\tau_\nu(D)}), \tag{40}$$

where I_ν is the (frequency dependent) intensity, S_ν is the source function along the ray and τ is the monochromatic optical depth. The first term in (40) represents absorption (intensity decreases by $\mathrm{e}^{-\tau}$) whilst the second term represents emission. For example, at large optical depth ($\tau \rightarrow \infty$) everything is obscured and all we see is the source function (i.e. light emitted from D), whereas at low optical depth $\tau \rightarrow 0$ the source function contributes nothing and all we see is the previous intensity $I(0)$.

The optical depth τ is given by

$$\tau(D) = \int \kappa\rho \mathrm{d}s, \tag{41}$$

where ρ is the density and κ is the opacity (with dimensions of 'cross section per unit mass').

For SPH visualisation the procedure is as follows. First of all we sort the particles in 'z', where z represents the distance from the observer to the particle. Then starting from the furthest particles, we consider the attenuation of a ray through each particle. Since what we are after is a final 2D pixel map, what we do in practise is take one ray for each pixel, but rather than taking a ray at a time (and looping over particles), we loop over all of the particles (from back to front), calculating the contribution of that particle to all rays ('pixels') in the final pixel map. The optical depth through the particle is given by

$$\tau(x, y) = \kappa \int \rho \mathrm{d}z, \tag{42}$$

where we have assumed that the opacity κ is independent of z. Using the SPH summation for the density, we have

$$\tau(x, y) = \kappa \sum_j m_j \int W(|\mathbf{r} - \mathbf{r}_j|, h)\mathrm{d}z, \tag{43}$$

giving just a summation involving the SPH kernel integrated through one spatial dimension, which is the same as is used in the 3D projections (see 4.3.1 for details of how we compute this). All that remains is to adjust κ appropriately to give the desired surface position. In SPLASH an approximate value for κ is computed according to

$$\kappa = \frac{\pi \bar{h}^2}{(\bar{m} Y(0) d)}, \tag{44}$$

where $\bar{h}$ and $\bar{m}$ are estimates for the average smoothing length and particle mass, calculated from the current (fixed) plot limits according to $\bar{h} = 0.5(h_{\min} + h_{\max})$ (similarly for $\bar{m}$, the important aspect here is that these values do not change between dump files and can be restored from saved settings) and $Y(0)$ is the value of the integrated kernel function (Section 4.3.1) at the origin. The dimensionless parameter d is then a user defined value giving approximately the surface depth in terms of 'number of smoothing lengths'.

Actually, rather than computing the sum in Equation (43) for the whole ray, we consider just the attenuation of the ray through one particle at a time, using the optical depth for that particle alone. Looping over each particle, we calculate the contribution to all rays (pixels) within the kernel radius $2h$. That is we have, for each particle

$$I(x, y) = I_0(x, y)\mathrm{e}^{-\tau_i(x,y)} + S_i(1 - \mathrm{e}^{-\tau_i(x,y)}), \tag{45}$$

where S_i is the source function (discussed below) and the optical depth through the particle's reach is

$$\tau_i(x, y) = \kappa m_i Y(x - x_i, y - y_i, h), \tag{46}$$

where Y is the integrated kernel function as in Section 4.3.1.

In the computation of the surface rendering, there are two ways of proceeding. The first option is to assign each particle an actual red, green and blue colour corresponding to the particle's value of the rendered quantity (i.e. from the colour table). The source function then consists of a red, green and blue intensity $S_{i(\mathrm{r})}$, $S_{i(\mathrm{g})}$, $S_{i(\mathrm{b})}$. Then we would add up (i.e. using Equation 45) the intensities in each colour (red, green and blue) to get final red, green and blue values at each pixel. The effect of this is to 'blend' colours (so a red plus blue would make purple), which

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

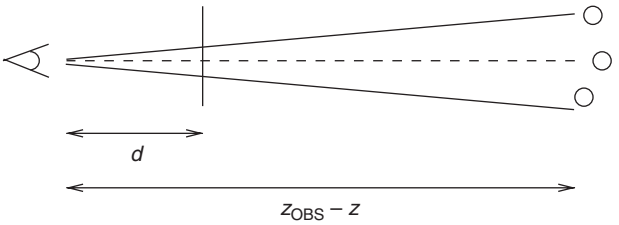

Figure 11 3D perspective: Objects at a distance d from the observer appear with unit magnification, whereas objects further away appear progressively smaller depending on their distance from the observer.

is more like what happens in a real gas, but is meaningless in the sense that the colours produced may no longer correspond to those in the colour table.

The alternative is to use a 'monochromatic' intensity, that is where the source function S_i for each particle is just the value of the rendered quantity at the particle location. Alongside this a 'total' optical depth is computed along each ray. Again, we add up the intensities according to Equation (40), but now there is only a single value of I for each pixel, which corresponds to a final 'ray-averaged' value of the rendered quantity. The pixel map can then be rendered in the usual manner using the ray-averaged values (which represent the values of the rendered quantity at the 'last scattering surface'). The only complication here is that we must make the particles optically thin to the background. Thus the final colours must be faded to the background colour (i.e. black) according to the total optical depth computed for each pixel. The latter method is the one used in SPLASH. However here we run into a limitation of the PGPLOT libraries, namely that the devices are limited to 256 colours, whereas we require 256 colours also at various degrees of blackness. Thus at present a non-faded version is returned to the PGPLOT device whilst a full (faded) version is written directly as a .ppm file (although without axes and annotation). This is one of the limitations that would make it desirable to change the back-end graphics library in future.

An example of 3D surface rendering is shown in Figure 5, showing temperature in a simulation of the merger of two neutron stars.

4.4 Rotation and 3D Perspective

Added perspective can be given to 3D plots by rotating the particles ('parallel projection') or using a depth-dependent 3D perspective (that is, so that objects further away appear smaller). For SPH visualisation it is straightforward to apply these transformations to the particle positions prior to the interpolation procedure. The algorithm for 3D perspective is described below.

4.4.1 3D Perspective

3D perspective (illustrated in Figure 11) is defined by two parameters: a distance to the observer (which we will call z_{OBS}) and a distance between the observer and a screen placed in front of the observer (which we will call d). The transformation from usual x and y to screen x' and y' is then given by

$$x' = x * d/(z_{OBS} - z),$$
$$y' = y * d/(z_{OBS} - z). \quad (47)$$

This means that objects at the distance d will have unit magnification, objects closer than the screen will appear larger (points diverge) and objects further away will appear smaller (points converge). The SPLASH default is a 1/10 reduction at the typical distance of the object (i.e. observer is placed at a distance of 10 times object size with distance to screen of 1 times object size). SPLASH sets this as default using the current 'z' plot limits as the 'object size'.

When using 3D perspective on interpolated plots the smoothing lengths of the particles are also modified by the 3D perspective, although the smoothing length used to give the z length scale on integrated plots (Equation 32) remains unchanged.

5 Other Useful Techniques

5.1 Fast Particle Plotting

Without using hardware graphics rendering, plotting large numbers of particles to the screen can be quite slow (certainly too slow for interactive work) and produces unnecessarily large files on vector plotting devices (e.g. postscript). Whilst one of the prime motivations behind SPLASH is to remove the need for raw particle plots as a poor man's SPH visualisation, plots showing correlations between certain variables or radial profiles can still require projected plots of large numbers of particles.

SPLASH uses a simple trick to speed up this kind of plotting by dividing the plot surface into an array of pixels (typically 500×500) and plotting up to a maximum of two particles in each cell. This results in a substantial speed increase with almost no loss in visible information. Note that upon zooming the same criterion is applied to the zoomed-in view surface, so the effective resolution is increased appropriately.

5.2 Accelerated Rendering

The slowest of the rendering techniques is the calculation of a 3D projection through particles (Section 4.3.1) and the 3D surface rendering (Section 4.3.3) since they both involve contributions from all of the particles in the simulation, not just a subset. The former has the advantage that it can be easily parallelised (done so using OpenMP in SPLASH) whilst the latter is more complicated to implement in parallel (since for the surface rendering the contributions at each z must be added in order). However a simple optimisation can be applied in both cases by taking advantage of the spherical symmetry of the kernel function.

For example, considering the interpolation to the pixels shown in Figure 7 it is apparent that, provided we assume that the particle lies in the centre of the pixel which contains it, that the contribution to each quarter of the domain

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

will be the same. Thus we can perform the interpolation to the top quarter of pixels only and copy the result to the remaining three quarters, providing an in-principle speedup of 4 for particles contributing to large numbers of pixels. The caveat is the assumption that the particle lies in the centre of the pixel. In practise the optimisation works well (that is, the results are visually identical to the non-optimised version) except where the particles are regularly distributed in the domain (e.g. on a lattice in the initial conditions), in which case the shift in the particle positions can produce unwanted grid patterns in the interpolation. For this reason the 'accelerated rendering' option is off by default but can be turned on by the user.

6 Performance and Memory Usage

As discussed above, the slowest rendering techniques used in SPLASH are the calculation of a 3D projection through particles and the 3D surface rendering. However, even these are sufficiently fast to be performed interactively. The algorithmic cost of the interpolation scales like $N_{\rm part} \times N_{\rm pix}^2$, where $N_{\rm pix}$ is the number of pixels to which each particular particle contributes. Thus larger images are more expensive. In SPLASH the default number of pixels is set quite low (i.e. 200×200), with the idea being that a smaller number of pixels can be used for interactive work with the final step in producing the finished image to use a larger number of pixels.

Whilst it is difficult to give precise timings (because the exact time taken for the rendering depends, amongst other things, on how many pixels each particle contributes to and thus how clustered the data are), SPLASH is easily able to handle very large data sets interactively in reasonable times. For example, producing a rendered projection of column density from a three dimensional simulation containing 135 million particles to a 600×600-pixel image takes approximately 55 seconds on a single processor of our local supercomputer. Using the (shared-memory) parallel version on 8 cores of the same machine takes approximately 12 seconds. Similarly a 100 million particle simulation of a galactic disc takes approximately 26 seconds to render to a 1000×1000-pixel image on a single processor and around 7 seconds on 8 cores. Using the accelerated rendering technique described above (Section 5.2) results in a factor of 2–3 speedup on these timings. Surface rendering is somewhat slower – approximately a factor of two more expensive than a column-integrated projection and currently not implemented in parallel. However the surface rendering technique is also not as widely applicable to different types of simulation.

In terms of memory use, by default SPLASH reads into memory an entire dump file, converted to a two-dimensional single precision array (where the dimensions are the number of particles times the number of columns). Thus for a typical 'full dump file' from a simulation of 10^6 particles with 10 quantities (x, y, z, v_x, v_y, v_z, particle mass, smoothing length, density and thermal energy) this would require approximately 40 MB of storage (and hence 400 MB for 10^7 particles, 4 GB for 10^8 particles, etc.). Additionally a 4-byte integer colour index is stored for each particle and temporary memory is allocated for the two dimensional pixel array which is rendered to the screen, the size of which depends on the number of pixels chosen by the user (for example a 1000×1000 image would require a further 4 MB). A low-memory mode for large data sets where memory is only allocated for those columns actually required to make a particular plot is currently being implemented (though applicable only to binary formats where data columns can be read independently). In this mode the data are re-read from disk every time a different plot is made (e.g. when plotting a z–x projection of column density instead of an x–y projection). Also plotting functionality which requires additional storage (such as particle colouring) will eventually be disabled in this mode.

For smaller data sets, SPLASH can also be set to 'buffer' all of the dumpfiles into memory, thus with memory requirements similar to the above times the number of dump files buffered. This provides a faster visualisation across multiple files for small data sets (since data do not have to be read from disk), provided sufficient memory is available.

For applications involving of the order of 10^6 particles (typical of many current SPH simulations), the slowest part of movie-making (i.e. applying the same visualisation to a series of data files) is reading the data from disk. To speed up the visualisation in this case SPLASH flags whether or not each particular column is required for the image being produced. For data reads where columns can be read independently (including that for the GADGET code) this is then used to read only the required subsection of the data from the dump file, resulting in a much faster data read.

7 Summary/Roadmap

In this paper we have presented SPLASH, a software tool for the visualisation of data from astrophysical SPH simulations. The program is fully interactive, reads data direct from code dumps and can be used to visualise both scalar and vector SPH data in one, two and three dimensions both to the screen and also to a variety of plotting devices provided by the PGPLOT graphics library. The software is designed to provide the user with a rapid feel for the output of a simulation and a variety of efficiently-implemented visualisation techniques unique to SPH with which to represent the results. There are many other features of SPLASH not discussed in this paper which we leave the reader to discover for themselves (and are described in the SPLASH userguide). These include:

- setting of animation sequences between frames in a movie;
- exact solutions to common test problems (e.g. hydrodynamic shock tubes, Sedov blast wave);
- transformation to different coordinate systems (e.g. cylindrical, spherical and toroidal coordinates);

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press

- particle tracking limits;
- multiple plots per page (appropriately tiled where so desired);
- calculation of quantities not dumped.

Development of and improvements to the algorithms in SPLASH continues apace, particularly as a result of user feedback which has already helped to improve certain aspects of the program substantially. In terms of future developments, most notable is the absence of a routine for plotting stream/field lines through 3D SPH data and it would be highly desirable to be able to do this efficiently directly from the particles (rather than highly inefficiently via interpolation to a 3D grid).

Secondly in several places SPLASH has outgrown the capacities of the PGPLOT graphics library. There are now several other graphics libraries layered on similar interfaces to PGPLOT (e.g. PLPLOT and S2PLOT, Barnes et al. 2006) and migrating the back-end library to one of these would not represent a formidable challenge. A more challenging alternative would be to move directly to OPENGL rendering, primarily for the speedup but also for the ease with which complicated 3D graphics can be manipulated. However the difficulty with at least the last two of these (OPENGL and S2PLOT) at present is that inherent in the SPLASH design is *also* the ability to produce, non-interactively, appropriately annotated graphics for use in research papers. Similarly the visualisation should apply as easily to a series of dump files (non-interactively) as it does to a single file (interactively or not).

In summary, SPLASH is an efficient and capable software package which makes the visualisation of SPH data a straightforward and enjoyable task for the user. SPLASH is publicly available[5] and is released under the terms of the Gnu General Public Licence.

Acknowledgments

My knowledge of the techniques used for SPH visualisation presented here owes a great deal to many useful discussions with Matthew Bate. Useful conversations with Richard West and Klaus Dolag have also contributed to my understanding of the surface rendering technique. My research at the University of Exeter is supported by a UK PPARC/STFC postdoctoral research fellowship. Thanks also to the many users who have given feedback which has helped to improve SPLASH substantially and to the referee(s) for useful suggestions on this paper.

References

Barnes, D. G., Fluke, C. J., Bourke, P. D. & Parry, O. T., 2006, PASA, 23, 82
Bate, M. R., Bonnell, I. A. & Bromm, V., 2003, MNRAS, 339, 577
Bate, M. R. B., 1995, PhD thesis, University of Cambridge, Cambridge, UK
Benz, W., Cameron, A. G. W., Press, W. H. & Bowers, R. L., 1990, ApJ, 348, 647
Dale, J. E. & Davies, M. B., 2006, MNRAS, 366, 1424
Freitag, M. & Benz, W., 2005, MNRAS, 358, 1133
Frenk, C. S., et al., 1999, ApJ, 525, 554
Hernquist, L. & Katz, N., 1989, ApJS, 70, 419
Mayer, L., Quinn, T., Wadsley, J. & Stadel, J., 2002, Sci, 298, 1756
Monaghan, J. J., 1992, ARA&A, 30, 543
Monaghan, J. J., 2002, MNRAS, 335, 843
Monaghan, J. J., 2005, Rep. Prog. Phys., 68, 1703
Price, D. J., 2004, PhD thesis, University of Cambridge, Cambridge, UK, astro-ph/0507472
Price, D. J. & Monaghan, J. J., 2004, MNRAS, 348, 139
Price, D. J. & Monaghan, J. J., 2007, MNRAS, 374, 1347
Price, D. J. & Rosswog, S., 2006, Sci, 312, 719
Smith, A. J., Haswell, C. A., Murray, J. R., Truss, M. R. & Foulkes, S. B., 2007, MNRAS, 378, 785
Springel, V., 2005, MNRAS, 364, 1105
Springel, V. & Hernquist, L., 2002, MNRAS, 333, 649

[5] `http://www.astro.ex.ac.uk/people/dprice/splash/`

https://doi.org/10.1071/AS07022 Published online by Cambridge University Press