COMPUTER GRAPHICS *forum*

DOI: 10.1111/j.1467-8659.2009.01352.x

COMPUTER GRAPHICS *forum*
Volume 28 (2009), number 6 pp. 1618–1631

Evenly Spaced Streamlines for Surfaces: An Image-Based Approach

Benjamin Spencer[1], Robert S. Laramee[1], Guoning Chen[2] and Eugene Zhang[2]

[1]Department of Computer Science, Swansea University, SA2 8PP Wales, United Kingdom
{csbenjamin, R.S.Laramee}@swansea.ac.uk
[2]School of Electrical Engineering and Computer Science, Oregon State University, 2111 Kelley Engineering Center, Corvallis, OR 97331
{chengu, zhange}@eecs.oregonstate.edu

Abstract

We introduce a novel, automatic streamline seeding algorithm for vector fields defined on surfaces in 3D space. The algorithm generates evenly spaced streamlines fast, simply and efficiently for any general surface-based vector field. It is general because it handles large, complex, unstructured, adaptive resolution grids with holes and discontinuities, does not require a parametrization, and can generate both sparse and dense representations of the flow. It is efficient because streamlines are only integrated for visible portions of the surface. It is simple because the image-based approach removes the need to perform streamline tracing on a triangular mesh, a process which is complicated at best. And it is fast because it makes effective, balanced use of both the CPU and the GPU. The key to the algorithm's speed, simplicity and efficiency is its image-based seeding strategy. We demonstrate our algorithm on complex, real-world simulation data sets from computational fluid dynamics and compare it with object-space streamline visualizations.

Keywords: flow visualization, vector field visualization, streamline seeding, surfaces

1. Introduction

The problem of developing fast and intuitive methods of vector field visualization has received a great deal of attention in recent years. The analysis of flow in computational fluid dynamics models is of particular importance since modern solvers generate very large, complex data sets.

One popular approach to the task is the class of texture-based methods. Examples include spot noise [vW91], LIC [CL93] and more recently, LEA [JEH01] and IBFV [vW02]. These methods operate by mapping a noise function and texture onto every point of the data set, generating a dense and highly detailed view of motion within the vector field.

A second family of techniques is based around streamlines; curves in the domain that are tangent to the velocity of the flow field. The use of streamlines to depict motion in vector fields is of key interest in many areas of flow visualization. The low visual complexity of the technique coupled with scalable density means that important flow features and behaviour may be expressed elegantly and intuitively, in both static and interactive applications. Since one of the primary appeals of using streamlines is their visual intuitiveness, a great deal of prior research has focussed on effective seeding and placement within the vector field. All streamline-based flow visualization techniques have to face the seeding problem, that is, finding the optimal distribution of streamlines such that all the features in the vector field are visualized. One popular approach to this problem stems from the use of evenly spaced streamlines, i.e. streamlines that are distributed uniformly in space. Specifically, this work has centred around ensuring streamlines are evenly spaced, of an optimal length and are spatio-temporally coherent (Figure 1).

Until relatively recently, the task of distributing streamlines uniformly onto 3D surfaces has received comparatively little attention. This is due in part to the numerous difficulties encountered when performing particle tracing in 3D space. In this paper, we propose a novel and conceptually simple method of seeding and integrating evenly spaced streamlines for surfaces by making use of image space. In previous approaches, streamlines are first seeded and integrated in object

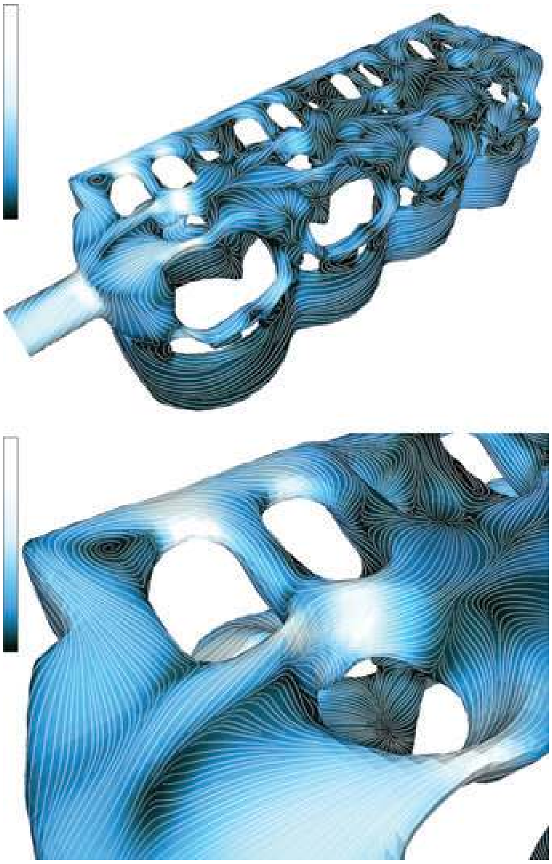

Figure 1: *Visualization of flow at the surface of a cooling jacket. The upper image presents an overview of the surface. The lower image focuses on the bottom left-hand corner of the jacket. The mesh is comprised of approximately 227 000 adaptive resolution polygons. Detailed images of sample grids have been presented earlier [Lar04].*

space. The result is then projected onto the image plane. In our approach, we reverse the classic order of operations by projecting the vector field onto the image plane, then seeding and integrating the streamlines. The advantages of this approach are as follows:

- Streamlines are always evenly spaced in image space, regardless of the resolution, geometric complexity or orientation of the underlying mesh (Figure 1).
- Streamlines are never generated for occluded or otherwise invisible regions of the surface.
- Various stages of the process are accelerated easily using programmable graphics hardware (Section 3.3).
- The user has a precise and intuitive level of control over the spacing and density of the streamlines.
- The algorithm is fast, resulting in support for user-interaction such as zooming, panning and rotation (Section 5).
- The distribution of the streamlines remains constant, independent of the user's viewpoint, e.g. zoom level.
- The algorithm decouples the complexity of the underlying mesh from the streamline computation and so does not require any parametrization of the surface (Section 3).
- The algorithm is simple and intuitive and thus could be incorporated into any visualization library.

However, in order to obtain these characteristics, certain challenges, both technical and perceptual, must first be overcome. We describe these in detail in the sections that follow. To our knowledge, this is the first general solution with an accompanying description to the problem of seeding evenly spaced streamlines for surfaces in 3D since the fast and popular 2D algorithm was presented by Jobard and Lefer [JL97] over 10 years ago.

The rest of this paper is organized as follows. In Section 2, we review previous related literature. In Section 3, we break down our method into multiple stages and describe each one in detail. We then propose several visual enhancements in Section 4 which help accentuate the perception of 3D space, the motion of the flow and the visual appeal of the streamlines. In Section 5, we demonstrate our technique by providing images and performance timings of the algorithm at work, using data generated by CFD solvers. Finally, we conclude in Section 6 with a summary of our method together with several promising avenues of future research.

2. Previous Work

In our review of the literature, we focus on automatic streamline seeding strategies as opposed to manual or interactive techniques [BL92]. See Laramee et al. [LHD*04] and Post et al. [PVH*03] for more comprehensive overviews of flow visualization literature.

2.1. Evenly spaced streamlines in 2D

Turk and Banks introduce the first evenly spaced streamline strategy [TB96]. The algorithm is based on an iterative optimization process that uses an energy function to guide streamline placement. Their work is extended to parametric surfaces (or curvilinear grids) by Mao et al. [MHHI98]. They adapt the aforementioned energy function to work in 2D computational space analogous to the way that Forssell and Cohen [FC95] extended the original LIC algorithm [CL93] to curvilinear grids.

The Turk and Banks algorithm [TB96] is enhanced by Jobard and Lefer [JL97] who introduce an accelerated version of the automatic streamline seeding algorithm. This algorithm uses the streamlines to perform what is essentially a search process for spaces in which streamlines have not already been seeded. Animated [JL00] and multiresolution versions of the algorithm [JL01] have been implemented.

Mebarki et al. [MAD05] introduce an alternative approach to that of Jobard and Lefer [JL97] by using a search strategy that locates the largest areas of the spatial domain not containing any streamlines. Liu and Moorhead [LM06] present another alternative approach capable of detecting closed and spiraling streamlines. Li et al. [LHS08] describe a seeding approach that resembles hand-drawn streamlines for a flow field.

2.2. Evenly spaced streamlines in 3D

Mattausch et al. [MT*03] implement an evenly spaced streamline seeding algorithm for 3D flow data and incorporate illumination. The technique does not generate evenly spaced streamlines in image space however, but object space.

Li and Shen describe an image-based streamline seeding strategy for 3D flows [LS07]. The goal of their work is to improve the display of 3D streamlines and reduce visual cluttering in the output images. Their algorithm does not however, necessarily, result in evenly spaced streamlines in image space. Streamlines may overlap one another after projection from 3D to 2D. Furthermore, unnecessary complexity is introduced by performing the integration in object space.

We also note the closely related, automatic streamline seeding strategies of Verma et al. [VKP00] and Ye et al. [YKP05]. These techniques seed streamlines first by extracting and classifying singularities in the vector field and then applying a template-based seeding pattern that corresponds to the shape of the singularity. Chen et al. [CML*07] also use a topology-based seeding strategy.

We have chosen to base our work on that of Jobard and Lefer [JL97] due to its clarity of exposition and elegant implementation. Their evenly spaced seeding algorithm has become a well-known classic flow visualization technique.

3. Evenly Spaced Streamlines for Surfaces

Here we present the details of our algorithm starting with a short discussion of why we chose an image-based approach.

3.1. Object versus parameter versus image space

In order to seed evenly spaced streamlines for surfaces, several challenges must first be addressed. To begin with, perspective projection can destroy the evenly spaced property of the streamlines (for example, Figure 14). Undesirable visual clutter may arise due to bunching of projected streamlines on surfaces near-perpendicular to the view vector. Secondly, visualization in 3D space is view-independent. Streamlines are likely to be generated for occluded surfaces which incurs a much greater computational overhead. Thirdly, zooming and panning of the viewpoint introduces problems with arbitrary levels of detail. In order to keep the coverage of the vector field constant, new streamlines would need to be integrated whenever the view changes. At high levels of magnification streamlines would need to lie relatively close together in object space, requiring a complex algorithm to detect which parts of the model were in view and which were not. Avoiding the generation of streamlines for non-visible regions outside the viewing frustum creates difficulty and introduces another layer of complexity.

The process of tracing streamlines over a 3D surface is made complex by the problem of particle tracing, which can be performed in either object or parameter space. In object space, a data structure is required which permits migration of particles from one polygon to another as they move around the surface. The process is made more difficult when polygons differ in area by six or more orders of magnitude as they typically do in meshes from CFD (for example, Figure 1). Tracing streamlines on surfaces demands robust intersection testing and numerically stable methods of handling special cases, such as when streamlines pass through vertices. Finally, checking for collisions between streamlines may require geodesic distance checking; a process which is typically very computationally expensive.

In parameter space, the mesh is treated as a locally Euclidean two-manifold. While this approach simplifies the process of particle advection, the task of parametrizing the surface is still very complex, especially when the structure is topologically intricate. Furthermore, parametrization introduces a distortion when mapping back onto physical space, and can also produce errors in the vector field.

3.2. Method overview

Our algorithm overcomes all of these difficulties by performing streamline integration in image space utilizing a multipass technique that is both conceptually simple and computationally efficient. It operates by projecting flow data onto the view plane, selecting and tracing seed candidates to generate the streamlines, and finally rendering both geometry and streamlines to the framebuffer. To generate our images we use a 3D polygonal model of a flow data set. Technically, the velocity is defined as 0 at the boundary (no slip condition) so we have extrapolated the velocity from just inside the boundary for visualization purposes. Each vertex describes the direction and magnitude of the flow at that point on the surface. An overview diagram describing each conceptual stage of the algorithm can be seen in Figure 2.

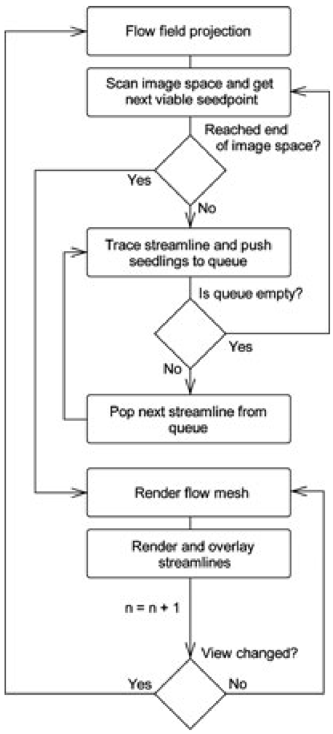

Figure 2: *An Overview diagram for generating evenly spaced streamlines on surfaces. Here, n is the frame number.*

3.3. Flow data projection

One of the key goals of our algorithm is to map the flow data into a structure over which streamlines may be traced effectively. To accomplish this we use a technique in which the flow information at each vertex is encoded into the colour and alpha channels of the frame buffer. This approach has several useful properties.

- The sparse flow data stored at each vertex of the mesh is automatically interpolated to an arbitrary level of detail using graphics hardware.
- Occluded surfaces are automatically discarded by z-testing and frustum culling.
- Rendering may be carried out in hardware. This is both fast and allows for the use of pixel and vertex shaders to compute flow projection.
- The complex problem of integrating evenly spaced streamlines over a 3D mesh is reduced to a simpler 2D problem.

Since perspective projections are view-variant, each velocity vector has to be transformed to homogeneous clip space at each frame before being encoded and rendered to the frame buffer. Transforming, projecting and encoding each flow vector is a task ideally suited to programmable graphics hardware since the computationally expensive matrix multiplications involved can be carried out using the GPU.

To pass the flow data to the graphics card, we store each flow velocity vector f as a float3 texture coordinate at each vertex. We also scale f by the reciprocal of the maximum magnitude $|f_{\max}|$ in the data set, thus mapping $|f|$ to the range [0, 1]. Flow data encoding and rendering is performed on the GPU by a single pass vertex and pixel shader. The vertex shader performs the following operations on each vertex v:

1. Add f to p, where f is the flow vector belonging to v and p is the position of v in object space.
2. Transform f and p to homogeneous clip space using a world-view-projection matrix such that each vector is represented by a homogeneous (x, y, z, w) matrix.
3. Store the w component of the vector f
4. Transform f and p into inhomogeneous screen space by dividing by w_f and w_p, respectively.
5. Subtract f from p to localize the flow velocity to the origin.
6. Multiply f by the stored value of w thereby reversing the foreshortening effect of the transform by cancelling out the perspective divide. This step is important for two reasons: first, the precision of the encoded data on distant surfaces is not diminished. Secondly, particle advection step length does not become a function of z-depth, thus reducing computation time.
7. Map the x and y components of f to the red and green channels of the framebuffer by outputting the diffuse vertex colour as:

$$r = 2^{d-1} + f_x(2^{d-1} - 1) \tag{1}$$

$$g = 2^{d-1} + f_y(2^{d-1} - 1). \tag{2}$$

Here, d is the bit depth of each colour channel. The output from this projection represents a velocity image of the flow field (Figure 3, top left-hand panel). Every pixel with an appropriate depth value stores a vector that can be decoded on an as-needed basis during the streamline seeding and integration process (Section 3.5).

In addition to encoding the flow velocity, we also store a 16-bit representation of the z-depth at each pixel using the remaining blue and alpha channels. Regions of the framebuffer which are not filled by triangles have a zero depth value and can be used to determine which pixels do not lie on the surface. Depth information is important when both seeding the streamlines and detecting discontinuities, aspects which are described in Section 4.

There are several additional options available if greater numerical accuracy is desired. We can normalize the

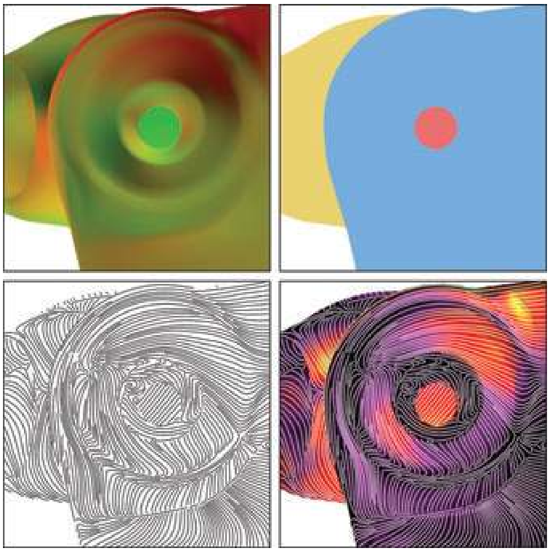

Figure 3: *Different stages of our algorithm: (upper left-hand panel) the velocity image result from vector field projection, (upper right-hand panel) the local regions (and their local boundary edges) of the geometry after projection to image space, (lower left-hand panel) evenly spaced streamlines in image-space with edge detection, (lower right-hand panel) a composite image of the streamlines and a shaded rendering of the intake port.*

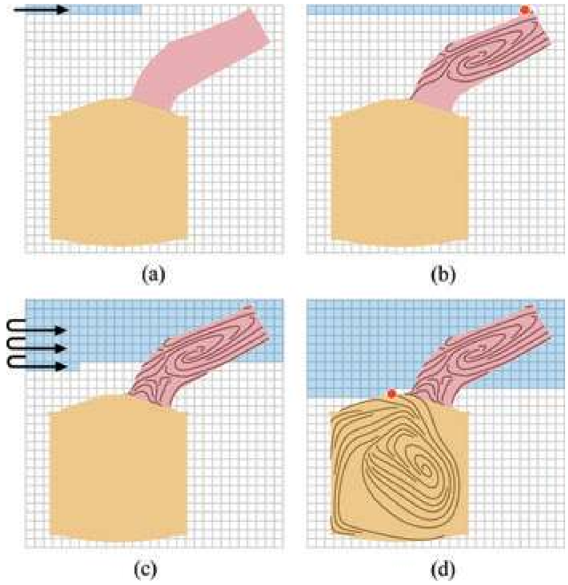

Figure 4: *Image-based seeding. (a) The image is sequentially scanned at intervals of d_{sep} from left to right and top to bottom. (b) A viable seed point is found and streamlines are traced in that distinct region. (c) Scanning continues down the image. (d) Another viable grid-based seed point is found and the rest of the vector field is visualized with streamlines.*

vector samples in an attempt to reduce loss of precision since the magnitude information is not critical (only the directional) for computing streamlines. We can also realize arbitrary levels of numerical precision by using a high dynamic range display buffer or multiple rendering passes. For example, we can store a 16-bit (or greater) representation of any arbitrary vector component, v_n, by using both the r and g channels (or more) for storage of v_n. Longer streamlines could be shortened to streamlets. However, we have found these measures unnecessary. A simple, single-pass, hardware-based projection produces streamlines with the same accuracy as an object-spaced, CPU approach. This can be seen in Figures 14 and 15 which compare the CPU, object-space and GPU, image-based approaches side-by-side. The image-based streamlines follow the same paths and depict the same flow characteristics as the object-space streamlines and are very suitable for visualization purposes. If an engineer is interested in exact velocity values, they simply click on the mesh at the point of interest to retrieve it (rather than using streamlines).

3.4. Rasterized image space search

Once the velocity image has been rendered to the framebuffer, we can proceed by identifying suitable points on the mesh at which we can seed the streamlines. The problem of properly visualizing multiple, locally discontinuous regions is depicted in Figure 3 (upper right-hand paper). Here, each coloured zone represents a local region of the geometry after projection to 2D space, in which we render streamlines. To ensure that we find discontinuities, we divide up the frame buffer using a grid and sequentially attempt to seed a streamline at the centre of each cell. The dimensions of the grid cells should ideally be the same as the user-supplied distance of separation between streamlines, d_{sep}. A cell is deemed to be a suitable seed candidate if both of the following conditions are met:

- The z-depth is non-zero, indicating that the seed point lies on the mesh.
- There are no streamlines that lie closer than d_{sep} to the seed point.

Starting from the top-left hand corner of the image, our algorithm sequentially scans across and downwards until the bottom-right cell has been reached (Figure 4). We call the seeds resulting from the rasterized image-spaced search, grid-based seeds. In practice, only a few streamlines are seeded in this fashion, the rest being placed by the vector field-based seeding strategy described in the following section.

3.5. Streamline integration

When a property is defined and advected on visible portions of a surface in 3D space, its end position is independent of whether its projection to the image plane takes place before or after the integration [LvWJH04]. As each grid-based seed point is determined (shown in red in Figure 5), we perform a modified version of the 2D evenly spaced streamline algorithm of Jobard and Lefer [JL97]. From the grid-based seed, the particle tracer integrates forwards and backwards through the flow. Upon termination, the algorithm then attempts to seed new streamlines. We call these type of seeds vector field-based seeds. At regular intervals along the length of the streamline curve, new candidate seedlings at a perpendicular distance d_{sep} are tested. Whenever a valid seed point is discovered, a new streamline is traced from it and pushed onto a queue. When no more seedlings can be found, the next streamline is popped from the queue and the process is repeated until the queue is empty. A more detailed overview of this algorithm can be found in previous literature [JL97].

At each iteration, the flow velocity at the position of the particle is computed by looking up the pixel value h at the corresponding position in image space. Inverting the transform used to originally encode the data into the velocity image results in the flow velocity vector $f(u, v)$:

$$f_u = \frac{h_r - 2^{d-1}}{2^{d-1} - 1} \qquad f_v = \frac{h_g - 2^{d-1}}{2^{d-1} - 1}. \tag{3}$$

The particle tracer terminates when the proximity between two neighbouring streamlines drops below the user-specified threshold $d_{\text{test}}(d_{\text{test}} \approx \frac{d_{\text{sep}}}{2})$, or when the z-depth drops to zero indicating the edge of the mesh has been reached. Proximity testing is accelerated using a static grid, the cells of which contain pointers to the streamline elements already placed. The size of each cell is d_{sep}, making the proximity test a simple matter of checking the cells adjacent to and containing the current element.

The problem of a streamline immediately terminating by incorrectly detecting immediate proximity to itself is addressed by introducing another condition into the proximity check (Figure 6). Specifically, neighbouring samples may only be considered when the dot product between the direction of motion and the relative position of the neighbouring sample is greater than zero. That is, when:

$$\frac{\delta x_\alpha}{\delta t}(x_\beta - x_\alpha) + \frac{\delta y_\alpha}{\delta t}(y_\beta - y_\alpha) > 0, \tag{4}$$

where α is the sample belonging to the current streamline and β a neighbouring sample tested for proximity checking. x and y represent the position of the samples on the grid and t the interval over which the particle is integrated. This solution bears resemblance to previous work [LS07].

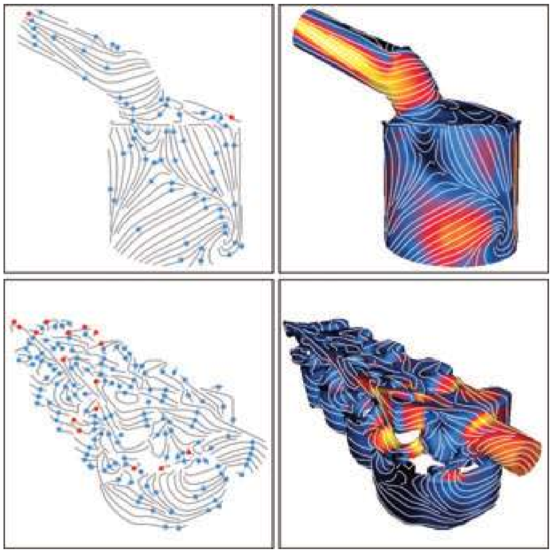

Figure 5: *Seed points: (top row) A gas engine simulation. (bottom row) A cooling jacket data set. The red circles represent seed points placed by our scanline algorithm (grid-based seeds). Blue circles are those seeded by other streamlines (vector field-based seeds).*

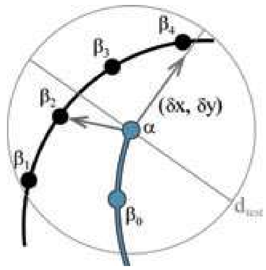

Figure 6: *Proximity testing: The dot product between the direction of motion $(\delta x, \delta y)$ and the relative position of the neighbouring sample $\beta_n - \alpha$, determines whether β_n is tested. In this case, β_0 to β_2 have negative products and are not tested. β_3 and β_4, however, have positive products and will subsequently trigger the termination of the streamline.*

3.6. Discontinuity detection

Tracing streamlines over a projected mesh differs from conventional integration over a planar flow field due to potential geometric discontinuities arising from edges and occluding surfaces. If the particle tracer is not aware of these features, undesirable artefacts may appear when integrating the streamlines. An example of such an artefact can be seen when a streamline abruptly changes direction due to a geometric discontinuity generated by one surface partially occluding another. Conversely, two such overlapping surfaces may

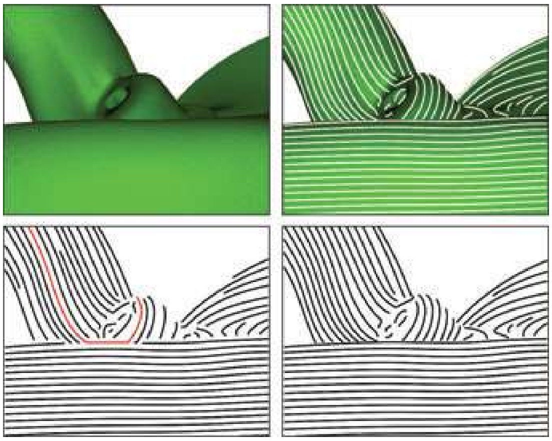

Figure 7: *Edge detection. Top left-hand panel: a close-up view of an edge from the diesel engine data set. Top right-hand panel: the same view with streamlines. Bottom left-hand panel: no edge-detection applied. Notice how the streamlines run off the edge of the upper surface then suddenly change direction due to the flow over the lower surface (highlighted in red). Bottom right-hand panel: the same scene with edge detection. The underlying structure of the flow mesh is now more clearly reflected by the streamlines.*

exhibit the same flow direction allowing a streamline to seamlessly cross from one surface to the other (Figure 7, bottom left-hand panel).

Given that our algorithm operates in image space, it is important to detect discontinuities when integrating streamlines so as to reflect the edges and silhouettes of the mesh. Our solution to this problem is to use the encoded output from the z-buffer stored in the blue and alpha channels of the frame buffer to track the depth value of each streamline element. Using this information, we augment the set of termination criteria to take into account surface geometry. In addition to the conditions specified by Jobard and Lefer [JL97], streamlines are also terminated when either:

1. The z-depth drops to zero indicating that the edge of the model has been reached.
2. The z-depth changes too abruptly indicating that the edge of two overlapping regions has been reached. More specifically, if the change in the depth between two samples exceeds a user-defined threshold then the streamline is terminated. This termination condition can also be expressed as follows:

$$\left|\frac{dz}{df}\right| > \epsilon. \tag{5}$$

Here z is the z-depth, f is the absolute position of the streamline on the flow field and ϵ is a user-defined threshold. Choosing a suitable value for ϵ depends on the distance between the near and far clipping planes. In our implementation, we adjusted the range of the z-depth information to fit closely to the dimensions of the mesh. We found a value of $\epsilon \approx 0.3\%$ yields good results. Alternatively, one could include the z-component into the distance calculation.

We found that using the gradient of the depth buffer worked well for edge detection, however achieving clean edges still proved difficult since streamlines would often terminate before reaching an edge boundary. This was caused by the proximity of streamlines on the other side of the edge falling below d_{sep}, thereby interfering with the edge detection. To solve this problem, we impose a constraint on the streamline distance check whereby streamline elements must be at approximately the same z-depth before proximity checking is applied. This approach is effective at producing clean edges and thus preserving the sense of depth and discontinuity (Figure 7).

3.7. Visual coherency

Our decoupling of object space and image space results in a streamline visualization with an unusual characteristic: new streamlines are generated whenever the viewpoint changes. One might presume that such a decoupling will result in disturbing visual artefacts during exploration of the simulation results. However, the algorithm yields streamlines with a high level of visual coherency as a result of the image-based grid used during the rasterized image space search described in Section 3.4. The grid-based seeds resulting from this rasterized search tend to remain in the same place (in image space) when the viewpoint changes slowly. Since the seeding algorithm starts with the grid-based seeds the vector field-based seeds ultimately stemming from the grid-based seeds tend to maintain a pleasing level of visual coherency. However, we also offer an option for users wishing to suppress the streamline generation process during changes to the viewpoint. This offers even faster interaction and the streamlines only require subsecond computation times after the user has modified the viewing parameters.

4. Streamline Rendering and Enhancements

Once the streamlines have been computed for the visible portions of the mesh, the next stage deals with rendering the data to the framebuffer. Before rendering and compositing the final image, the velocity image used to compute the streamlines can first be replaced. While our colour encoding describes the velocity and direction of the flow, the mapping is relatively complex and it is not intuitive or easily decipherable at a glance. Furthermore, the lack of any directed lighting means that a full sense of depth is not always conveyed.

We use a colour gradient mapped to the magnitude of the velocity and which is used to compute the diffuse colour at each vertex of the flow mesh. Re-rendering using diffuse illumination coupled with specular highlighting greatly improves the appearance of the scene at the expense of another rendering pass (as in Figure 14).

4.1. Flow animation

Animating the streamlines at interactive frame rates so as to better convey a sense of the downstream direction in the flow is a useful feature. We accomplish this by adapting the method described by Jobard et al [JL97] whereby a periodic intensity function $f(i)$ is mapped onto the streamline:

$$f(i) = \frac{1}{2}\left\{1 + \sin\left[2\pi\left(\frac{i}{N}\right) + \theta\right]\right\}. \tag{6}$$

Here, i is a given sample on the streamline, N is the size of the wave period interval and θ is the wave phase. The output from this function can be used to vary a number of attributes including streamline width, alpha value and colour. We define θ per frame as $\theta = 2\pi\frac{n}{M}$, where n is the current frame number and M is the number of frames per period. The effect resulting from varying θ is a series of visual cues moving along the length of the streamline in the direction of motion.

Our implementation utilizes a particle tracer with an adaptive step-size which distributes samples along the length of the streamline regardless of the magnitude of the flow. One drawback with the sinusoidal function described above is that the flow velocity appears constant at all points on the streamline. This is as a result of information lost by the uniform spacing of samples and results in an non-optimal perception of the flow field magnitude.

To correct this, we store the time taken by the integrator to reach each streamline element and use this information to independently calculate the value of the parameter, i. This means that as θ is varied, the resulting wave pattern travels faster in regions of higher velocity magnitude. A demonstration of this feature can be found on the video that accompanies this paper.

4.2. Perspective foreshortening

A desirable result of perspective projection is the sense of depth produced by foreshortening. Using a fixed value of d_{sep} in image space means that distant surfaces have the same projected density of streamlines as those nearby. When overlaid and rendered onto the underlying mesh, the resulting effect could be considered counter-intuitive in certain situations.

To address this, we propose dynamically adjusting the value of d_{sep} depending on the distance between the surface and the view plane. The desired outcome is to alter the density of streamlines in image space so that it more closely resembles a projection of evenly spaced streamlines in world space. For a given point with depth z on the flow mesh, we calculate the new separating distance d'_{sep} as being:

$$d'_{\text{sep}} = \frac{d_{\text{sep}}(z_{\max} + z_{\min})}{2z} \tag{7}$$

Where $z_{\min}$ and $z_{\max}$ are equal to the minimum and maximum visible points on the surface, respectively. Depending on the position and orientation of the mesh, the overhead from proximity checking is generally slightly higher than usual. Figure 8 demonstrates perspective foreshortening with

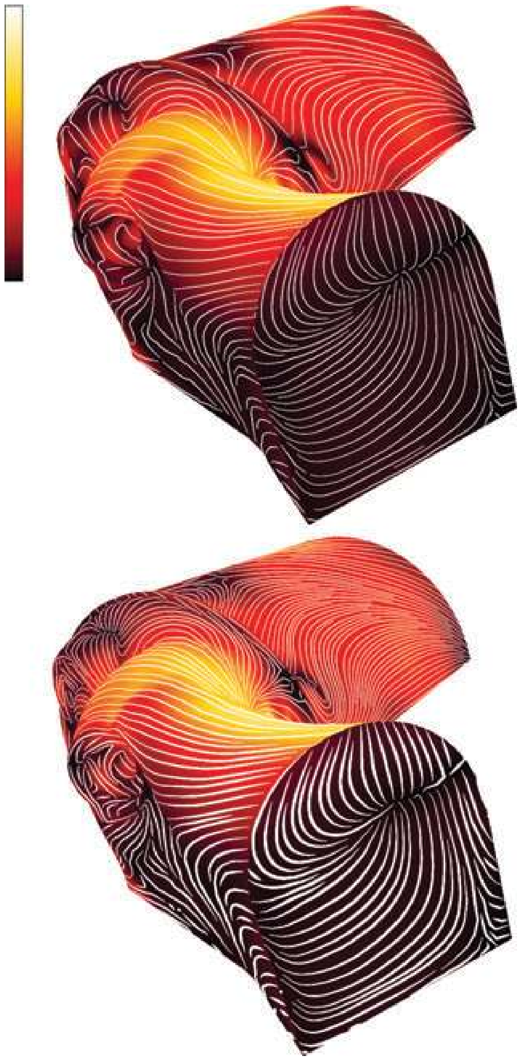

Figure 8: *Top panel: a top view of the gas engine simulation without perspective foreshortening. Bottom panel: the same scene with perspective foreshortening. Notice how the streamline density is much higher towards the rear of the model. In addition to varying d_{sep}, we have also adjusted the thickness of the streamlines to compensate for the change in density.*

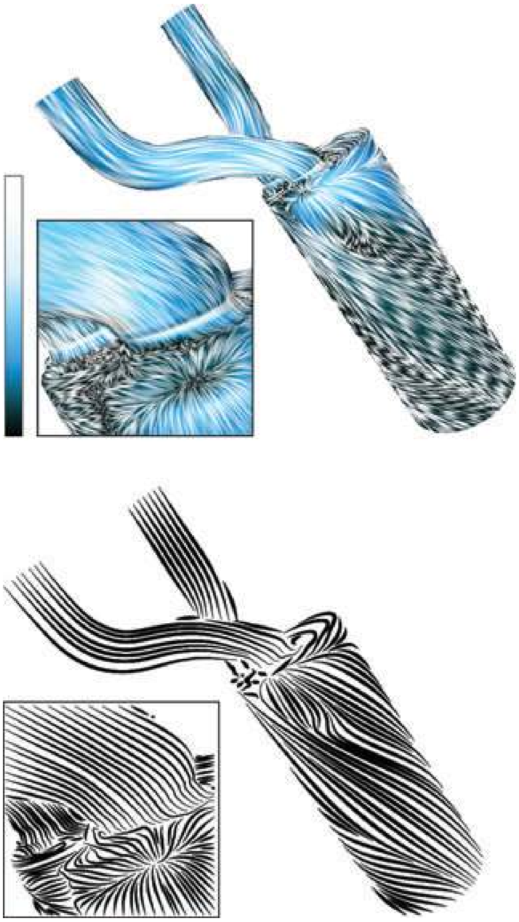

Figure 9: *Top: visualization of a diesel engine simulation with coarse, heavily weighted streamlines. Bottom: with fine, tapering streamlines. The output from the periodic intensity function has been used to determine the thickness of each streamline.*

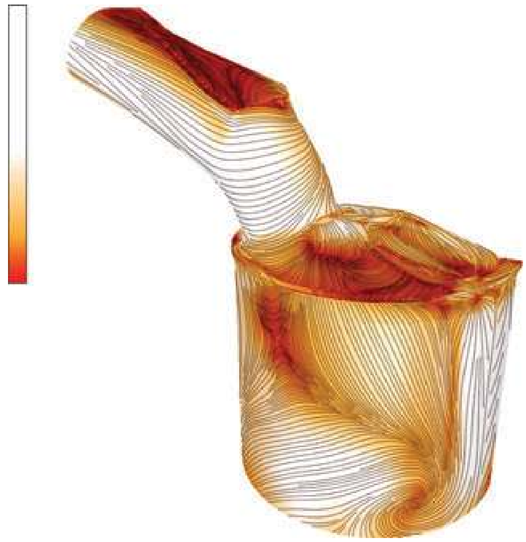

Figure 10: *Visualization of flow through the gas engine simulation. Here, d_{sep} is set as a function of flow importance. Regions of slower flow exhibit a higher streamline density.*

one image exhibiting a variable density of streamlines. Notice how the evenly spaced property is still maintained while conveying a more realistic sense of depth.

4.3. Variable width and tapering streamlines

Varying the width of the streamlines can be used, both to enhance certain flow features, as well as improve aesthetic appearance. Figure 9 demonstrates two examples of variable width. The right-hand example defines the width $w(i)$ at any given point i on the streamline w as:

$$w(i) = 1 + (w_{\max} - 1)\sin\left(2\pi \frac{i}{i_{\max}}\right), \tag{8}$$

where $i_{\max}$ is the length of the streamline and $w_{\max}$ is the maximum width. In this example, the value of d_{sep} is relatively low the and the streamlines relatively thick so as to create an artistic, hand-painted appearance. Figure 9 also demonstrates the result of varying the streamline width using the output from the periodic intensity function $f(i)$. In the neighbouring example $w_{\max}$ is equivalent to d_{test} so as to create a very dense and highly detailed flow effect. The first example uses the output from $w(i)$ to vary the streamline thickness.

4.4. Increasing information content

There are several ways in which the information conveyed by our seeding algorithm may be increased. To begin with we can extend the technique outlined in 4.2 by decreasing d_{sep} in regions of greater importance or increasing flow complexity. For example, flow velocity, vorticity and proximity to critical points all yield scalar values and may be used to control streamline density. By encoding these values into the velocity image, we can guide the streamline placement algorithm into rendering areas of greater definition where appropriate. Figure 10 demonstrates the effect of mapping d_{sep} to flow importance. In this example, regions of the field with low magnitude correspond to a higher streamline density. Notice how the density of the streamlines increases in the regions containing flow features. Here a saddle point, a sink and the separatrix connecting them are emphasized by the streamlines that are no longer evenly spaced.

By decreasing d_{sep} to approximately one pixel we can also obtain complete coverage of the vector field. This is highlighted in Figure 11 where the colour of each streamline is mapped to the flow velocity. In addition we can also reduce

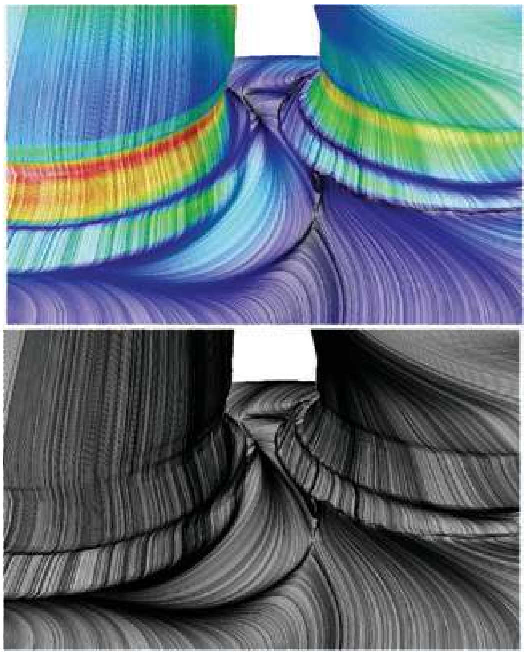

Figure 11: *A close-up visualization of flow through the diesel engine simulation. In this example* d_{sep} *has been set to* 2.0 *and* d_{test} *to* $0.1 \times d_{\text{sep}}$.

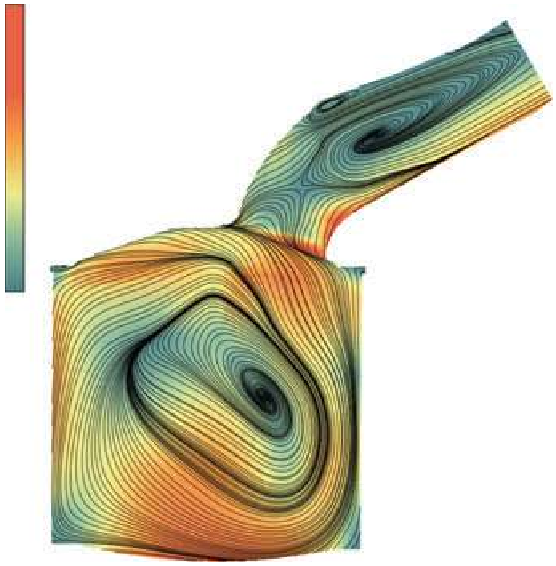

Figure 12: *Visualization of flow through the gas engine simulation. By setting* d_{test} *to* $0.05 \times d_{\text{sep}}$*, streamlines bunch together highlighting loops within the flow field.*

d_{test} to a arbitrarily small value. This has the effect of relaxing the separation constraints, allowing streamlines to converge and bunch together. By setting $d_{\text{test}} << d_{\text{sep}}$ streamlines tent to converge on areas of increasing flow complexity and singularities in the vector field. An example can be seen in Figure 12. In this instance, vortex cores and periodic orbits within the flow are highlighted by the increase in density. We have verified the correctness of this result based on our experience of extracting these same features directly [CML*07].

Figure 3 shows d_{sep} mapped to depth of field. Figure 10 shows d_{sep} mapped to velocity magnitude. Figure 11 shows $d_{\text{sep}} \approx 2$ pixels in order to gain complete coverage of the vector field, thus maximizing information content. Figure 14 shows d_{sep} mapped to the dot product of the view vector with the surface normal which can give the appearance of streamline density as a function of distance to the surface, i.e. the object-space method. We emphasize that in fact, d_{sep} can be mapped to any property or scalar field of the data set arbitrarily. What is shown here are only a few of the possibilities.

5. Implementation and Results

We tested our algorithm on a range of data sets taken from complex CFD simulations. To obtain high-quality results, we use a second-order Runge–Kutta particle tracer with an adaptive step size in the sub-pixel range. Our test system included an Intel Core 2 Duo 6400 processor with 2GB RAM and an nVidia GeForce 7900 GS graphics card. Given that the flow projection and mesh rendering passes are handled by the GPU, we found that increasing the complexity of the underlying model did not adversely affect the time taken to generate an image. Except when the number of polygons was relatively high, our graphics card capped the frame rate at 60Hz. In order to render the streamlines to the framebuffer, however, the memory associated with the device needed to be locked at each frame. Reading from video memory typically incurs a read-back penalty (we encountered it to be approximately 570ms per megabyte of framebuffer data) which adversely affects performance. However the net gain of offloading computationally expensive tasks onto the graphics hardware meant that this was an acceptable trade-off. In all our examples, the underlying colour gradient is mapped to flow velocity.

Figure 13 demonstrates a ring surface. In this example, d_{sep} has been reduced to 2.0 and the alpha channel is varied using Equation (6). This gives the appearance of an image-based flow effect such as LIC, although each visible fibre is actually a streamline.

Figure 14 uses high-detail data from the computed flow through two intake ports. Here, the colour scheme has been chosen to highlight slow-moving flow. Notice how the streamlines fit well around the small holes on top of each of the two intake lines. We also compare our algorithm with an object-based approach (middle image). There is no visible difference in terms of the accuracy between each method of

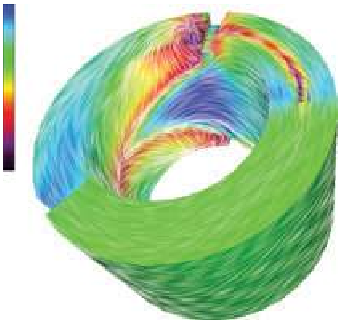

Figure 13: *The simulation of flow through a ring. The visible spectrum colour gradient has been overlaid with dense streamlines, The widths of which are varied by a sinusoidal, periodic intensity function.*

streamline integration. Further results are illustrated in the accompanying video.

The data set in Figure 1 is a snapshot from a simulation of fluid flow through an engine cooling jacket. The adaptive resolution mesh is composed of over 227 000 polygons and contains many holes, discontinuities and seeding zones. Despite the high level of geometric complexity, our algorithm computes evenly spaced streamlines cleanly and efficiently. In this instance, using a technique based on surface parametrization would be especially difficult owing to the complex topology of the shape.

In Figure 15, we demonstrate the flexibility of our algorithm in handling arbitrary levels of magnification. The left-most image shows a profile view of a gas engine simulation cut-away with object-based streamlines. The next image shows the same data set rendered using image-based streamlines. The remaining images show progressively higher factors of magnification with the small square in the first frame corresponding to the field of view in the final frame. Note how the spacing of the streamlines automatically remains uniform, independent of the level of magnification.

Table 1 compares the time taken to integrate streamlines over the velocity image for each of the four models described above. The figures describing the size of the flow field are calculated by summing the number of visible pixels belonging to the flow mesh that are rasterized onto the framebuffer. Our performance times are comparable to previous 2D seeding algorithms. Furthermore, our algorithm is approximately two orders of magnitude faster than the CPU, object-based method owing to the reduced computational complexity.

The CPU, object-based method requires more than 60 s of computation time (several minutes). It is worth noting that our implementation of the original evenly spaced streamline algorithm is not fully optimized. Several enhancements and improvements have been proposed that both speed up and refine seeding and placements of streamlines [LM06], however we have deliberately kept our implementation simple so as to concentrate on extending it to a higher spatial dimension.

6. Accuracy

There are several additional options available if greater numerical accuracy is desired. We can normalize the vector samples since the magnitude information is not critical (only the directional) for computing streamlines. We can also realize arbitrary levels of numerical precision by using a high dynamic range display buffer or multiple rendering passes.

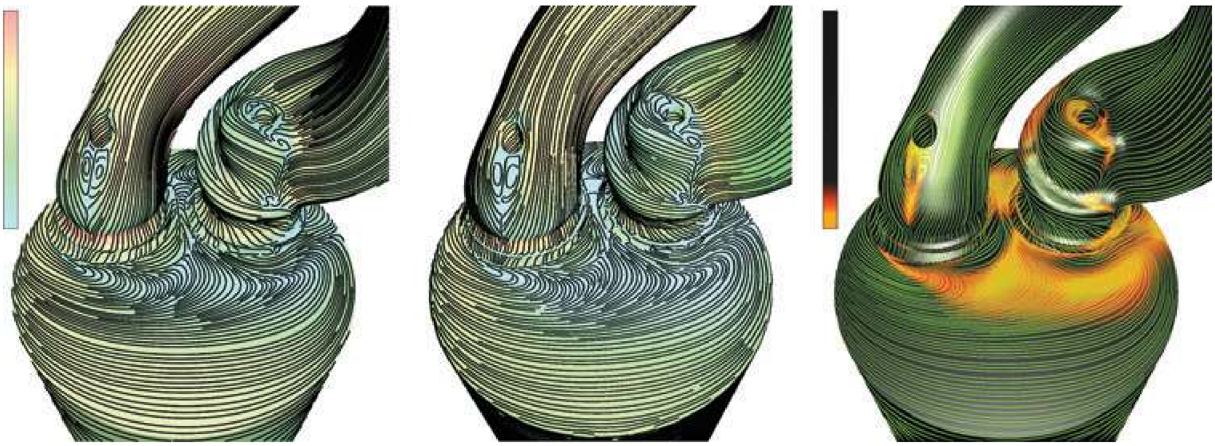

Figure 14: *The visualization of flow at the boundary surface of two intake ports. (Left) With our novel, image-based streamlines. (Middle) With full-precision, object-based streamlines computed on the CPU. (Right) High-contrast, image-based streamlines. This mesh is comprised of approximately 222 000 polygons at an adaptive resolution.*

Table 1: *Streamline generation timing figures for a variable value of d_{sep}. In these examples, the integration step size is set to 1 pixel. Foreshortening and edge detection is enabled. The dimensions of the framebuffer upon which each mesh is rendered is 500^2 pixels. % is the amount of the image plane covered by the geometry after projection.*

		d_{sep} (pixels)					
Scene	%	1.0	2.0	4.0	8.0	16.0	32.0
Gas Engine	39.3%	1977.6 ms	627.49 ms	244.21 ms	95.27 ms	46.76 ms	18.93 ms
Diesel Engine	39.6%	1455.08 ms	456.58 ms	166.28 ms	73.44 ms	29.69 ms	16.20 ms
Ring Surface	41.3%	1392.65 ms	368.39 ms	132.23 ms	53.79 ms	24.41 ms	12.59 ms
Cooling Jacket	39.3%	3774.56 ms	1345.37 ms	592.95 ms	250.81 ms	126.44 ms	46.55 ms

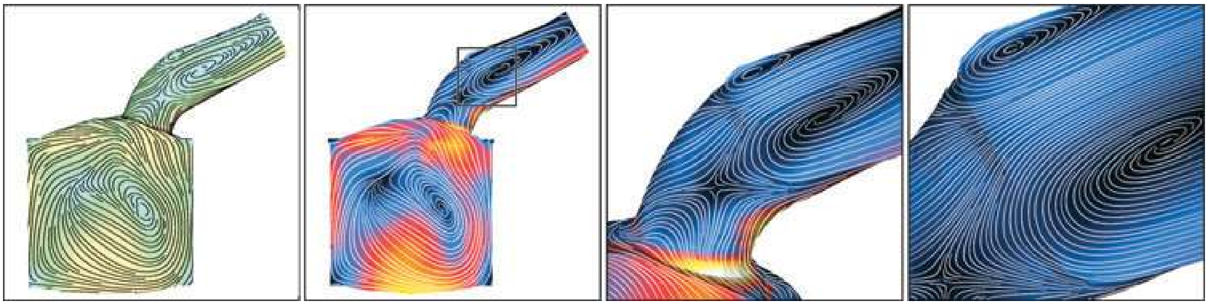

Figure 15: *Zooming: Visualization of the flow at the surface of a gas engine simulation at progressively higher levels of magnification. The left-most image was generated using a full floating-point, object-based algorithm computed on the CPU. The successive images were generated using our novel, image-based technique.*

For example, we can store a 16-bit (or greater) representation of any arbitrary vector component, v_n, by using both the r and g channels (or more) for storage of v_n. Longer streamlines could be shortened to streamlets. However, we have found these measures unnecessary. A simple, single-pass, hardware-based projection produces streamlines with the same accuracy as an object-spaced, CPU approach. This can be seen in Figures 14 and 15 which compare the CPU, object-space and GPU, image-based approaches side-by-side.

The full precision, floating point, object space technique we use for comparison is adapted directly from Jobard and Lefer's original algorithm [JL97]. We implement particle advection using the local coordinate frame of the occupied triangle to interpolate flow velocity. Migration to adjacent polygons is also tracked. Proximity checking is accomplished by computing the geodesic distance across the surface to neighbouring streamlines. Algorithm 1 describes pseudocode outlining the particle tracing step of this process. While an object space approach is very accurate, geodesic distance checking incurs a high performance penalty. As a result, complete coverage of the surface takes considerably longer than our image space approach.

Notice how the image-based streamlines follow the same paths and depict the same flow characteristics as the object-space streamlines making them very suitable for visualization purposes (Figures 14 and 15). If an engineer is interested in exact velocity values, they simply click on the mesh at the point of interest to retrieve it (rather than using streamlines).

We also point out that no exact method for tracing individual trajectories exists. Visualization of vector fields using individual trajectories can raise questions with respect to accuracy in general, due to the discrete nature of the simulation data. First, data samples are only given at discrete locations such as cell centres or cell vertices. Interpolation schemes are then used to reconstruct the vector field between the given samples. Secondly, the given data samples themselves are numerical approximations, e.g. approximate solutions to a set of partial differential equations. Thirdly, the given flow data are often only a linear approximation of the underlying dynamics. Finally, the visualization algorithms themselves, e.g. streamline integrators, have a certain amount of error inherent associated with them.

In summary, approaches on how to handle such error is a topic for other papers [CML*07, CMLZ08].

7. Conclusion and Future Work

In this paper, we have proposed a novel, image-based technique for generating evenly spaced streamlines over surfaces—a problem that has remained unsolved for more than 10 years. We have shown that our algorithm effectively places streamlines on data sets with arbitrary topological and

Algorithm 1 TRACESTREAMLINE*(SeedPoint, MaxSteps).*

```
Particle ← SeedPoint
Streamline.Add(Particle) {Add initial segment to streamline}
for i = 0 to MaxSteps do
   if IsClosedLoop(Streamline) then
      break
   end if
   {Get a list of triangles within the geodesic range of Particle}
   PolyList ← FastMarchingGeodesic(Particle, d_test)
   {If Particle passes too close to a streamline in PolyList}
   if IsCloseToStreamline(Particle, PolyList, d_test) then
      break
   end if
   {If Particle passes too close to a singularity in Polylist
   if IsCloseToSingularity(Particle, PolyList, d_test) then
      break
   end if
   Particle ← IntegrateParticle(Particle)
   Streamline.Add(Particle) {Add segment to streamline}
end for
return Streamline
```

geometric complexity. We have also demonstrated how a sense of depth and volume can be conveyed while preserving the desirable evenly spaced property of the algorithm's 2D counterpart. Our results show that an image-based projection approach and seeding strategy can automatically handle zooming, panning and rotation at arbitrary levels of detail. The efficiency of the technique is also highlighted by the fact that streamlines are never generated for invisible regions of the data set. The accuracy of the visualization is demonstrated by comparing the results of image- and object-based approaches.

As future work we would like to explore the possibility of implementing the entire algorithm on programmable graphics hardware. Finally, we would like to investigate the feasibility of parallelizing the streamline integration step, so as to take advantage of the increasing availability of multi-core processors. We would also like to extend our algorithm to handle unsteady flow.

Acknowledgement

This research was funded in part by the EPSRC Research Grant EP/F002335/1.

References

[BL92] BRYSON S., LEVIT C.: The virtual wind tunnel. *IEEE Computer Graphics and Applications 12*, 4 (1992), 25–34.

[CL93] CABRAL B., LEEDOM L. C.: Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH' 1993*, Annual Conference Series (1993), pp. 263–272.

[CML*07] CHEN G., MISCHAIKOW K., LARAMEE R. S., PILARCZYK P., ZHANG E.: Vector field editing and periodic orbit extraction using morse decomposition. *IEEE Transactions on Visualization and Computer Graphics 13*, 4 (2007), 769–785.

[CMLZ08] CHEN G., MISCHAIKOW K., LARAMEE R. S., ZHANG E.: Efficient morse decompositions of vector fields. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (2008), 1–15.

[FC95] FORSSELL L. K., COHEN S. D.: Using line integral convolution for flow visualization: curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (1995), 133–141.

[JEH01] JOBARD B., ERLEBACHER G., HUSSAINI M. Y.: Lagrangian-Eulerian advection for unsteady flow visualization. In *Proceedings IEEE Visualization '01* (2001), IEEE Computer Society, pp. 53–60.

[JL97] JOBARD B., LEFER W.: Creating evenly-spaced streamlines of arbitrary density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing '97* (1997), vol. 7, pp. 45–55.

[JL00] JOBARD B., LEFER W.: Unsteady flow visualization by animating evenly spaced streamlines. In *Computer Graphics Forum (Eurographics 2000)* (2000), vol. 19(3), pp. 21–31.

[JL01] JOBARD B., LEFER W.: Multiresolution flow visualization. In *WSCG 2001 Conference Proceedings* (Plzen, Czech Republic, 2001), pp. 33–37.

[Lar04] LARAMEE R. S.: *Interactive 3D Flow Visualization Using Textures and Geometric Primitives*. PhD thesis, Vienna University of Technology, Institute for Computer Graphics and Algorithms, Vienna, Austria, 2004.

[LHD*04] LARAMEE R. S., HAUSER H., DOLEISCH H., POST F. H., VROLIJK B., WEISKOPF D.: The state of the art in flow visualization: dense and texture-based techniques. *Computer Graphics Forum 23*, 2 (2004), 203–221.

[LHS08] LI L., HSIEH H.-S., SHEN H.-W.: Illustrative streamline placement and visualization. In *IEEE Pacific Visualization Symposium 2008* (2008), IEEE Computer Society, pp. 79–85.

[LM06] LIU Z. P., MOORHEAD, II R. J.: An advanced evenly spaced streamline placement algorithm. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 965–972.

[LS07] Li L., Shen H.-W.: Image-based streamline generation and rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 630–640.

[LvWJH04] Laramee R. S., van wijk J. J., Jobard B., Hauser H.: ISA and IBFVS: Image space based visualization of flow on surfaces. *IEEE Transactions on Visualization and Computer Graphics 10*, 6 (2004), 637–648.

[MAD05] Mebarki A., Alliez P., Devillers O.: Farthest point seeding for efficient placement of streamlines. In *Proceedings IEEE Visualization 2005* (2005), IEEE Computer Society, pp. 479–486.

[MHHI98] Mao X., Hatanaka Y., Higashida H., Imamiya A.: Image-guided streamline placement on curvilinear grid surfaces. In *Proceedings IEEE Visualization '98* (1998), pp. 135–142.

[MT*03] Mattausch O., Theussl T., Hauser H., Gröller E.: Strategies for interactive exploration of 3d flow using evenly spaced illuminated streamlines. In *Proceedings of the 19th Spring Conference on Computer Graphics* (2003), pp. 213–222.

[PVH*03] Post F. H., Vrolijk B., Hauser H., Laramee R. S., Doleisch H.: The state of the art in flow visualization: feature extraction and tracking. *Computer Graphics Forum 22*, 4 (2003), 775–792.

[TB96] Turk G., Banks D.: Image-guided streamline placement. In *ACM SIGGRAPH 96 Conference Proceedings* (1996), pp. 453–460.

[VKP00] Verma V., Kao D., Pang A.: A flow-guided streamline seeding strategy. In *Proceedings IEEE Visualization 2000* (2000), pp. 163–170.

[vW91] van Wijk J. J.: Spot noise-texture synthesis for data visualization. In *Computer Graphics (Proceedings of ACM SIGGRAPH'91)* (1991), Sederberg T. W., (Ed.), vol. 25, pp. 309–318.

[vW02] van Wijk J. J.: Image based flow visualization. *ACM Transactions on Graphics 21*, 3 (2002), 745–754.

[YKP05] Ye X., Kao D., Pang A.: Strategy for seeding 3D streamlines. In *Proceedings IEEE Visualization 2005* (2005), pp. 471–476.