GLYPH AND STREAMLINE PLACEMENT ALGORITHMS FOR CFD SIMULATION DATA

Zhenmin Peng[1], Robert S. Laramee[1], Guoning Chen[2], Eugene Zhang[2]

[1]Visual and Interactive Computing Group, Department of Computer Science, Swansea University, Swansea, UK. Email {cszp,r.s.laramee}@swansea.ac.uk

[2]School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331. Email: {chengu,zhange}@eecs.oregonstate.edu.

THEME
Virtual Reality, Visualization

KEYWORDS
glyph placement, streamline placement, streamline seeding, flow visualization, vector field visualization, CFD simulation data

SUMMARY
Visualization of flow on boundary surfaces from computational flow dynamics (CFD) is challenging due to the complex, adaptive resolution nature of the meshes used in the modeling and simulation process. Part one of this paper presents a fast and simple glyph placement algorithm in order to investigate and visualize flow data based on unstructured, adaptive resolution boundary meshes from CFD. The algorithm has several advantages: (1) Glyphs are automatically placed at evenly-spaced intervals. (2) The user can interactively control the spatial resolution of the glyph placement and their precise location. (3) The algorithm is fast and supports multi-resolution visualization of the flow at surfaces. The implementation supports multiple representations of the flow–some optimized for speed others for accuracy. Furthermore the approach doesn't rely on any pre-processing of the data or parameterization of the surface and handles large meshes efficiently. The result is a tool that provides engineers with a fast and intuitive overview of their CFD simulation results.

In part two, we introduce an 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 parameterization, 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.

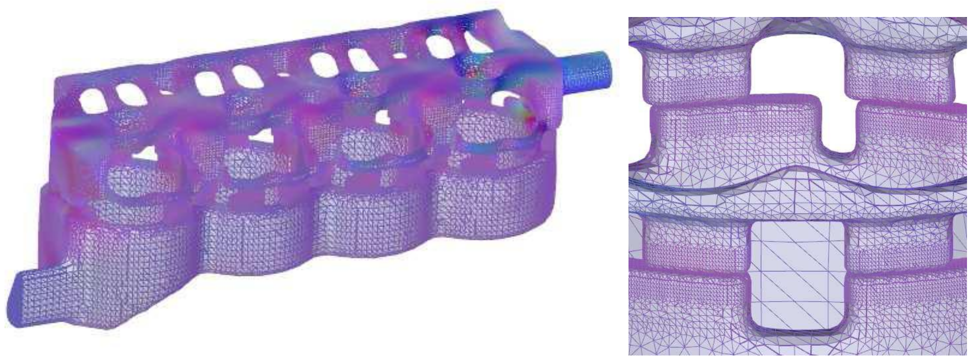

Figure 1: The unstructured adaptive resolution boundary grid of a cooling jacket from a CFD simulation. The first image is an overview of the boundary mesh, and the second is a close-up. These images illustrate how complex a typical mesh from CFD can be.

1 Introduction

Ever increasing attention is invested in order to find reasonable and efficient solutions for analyzing and visualizing the flow from computational fluid dynamics in last three decades. As the size of simulation data sets increases, so does the need for effective visualizations that provide insight into the data. A tremendous amount of time and money is spent on simulation in order to speed up the manufacturing process. Constructing objects in software should be faster than building their real hardware counterparts.

Out of all the possible visualization techniques that can be used to investigate the simulation results, vector glyphs and color-coding are the most popular tools used by engineers. Vector glyphs offer several advantages. They are intuitive – the depiction of the underlying flow is universally understood. Secondly, they do not accumulate error in the same way that geometric techniques do. Integration-based visualizations such as streamlines have in an inherent error associated with them stemming from the approximations made in the underlying computation. Thirdly, glyphs are easy to implement. No complicated algorithms or data structures are needed. Thus they are featured in every software application. However, glyphs also have their drawbacks. Optimal vector field glyph placement is a challenge, especially in the context of CFD applications. Figure 1 shows a typical, triangulated boundary mesh produced from a CFD model. Its unstructured, adaptive resolution characteristics make the placement of vector glyphs difficult. If we naively place a vector glyph at every sample point on the surface, then the glyphs are either too small to see or so large that they overlap and result in clutter. Another drawback is that the density of glyphs corresponds with the density of mesh polygons. This variation is unrelated to the vector values themselves. Also, the user has no control over the glyph placing. Furthermore, rendering so many glyphs degrades performance time greatly. Most of the glyphs would be occluded.

While glyph-based visualization has been widely applied to tensor field and medical visualization [19] [23], glyphs for vector field visualization have received relatively little attention. This may be due to the difficulties in placing glyphs evenly on unstructured, adaptive resolution boundary meshes from the complex CFD data sets and perceptual problems like visual complexity and occlusion (as indicated above). In order to address these challenges, we

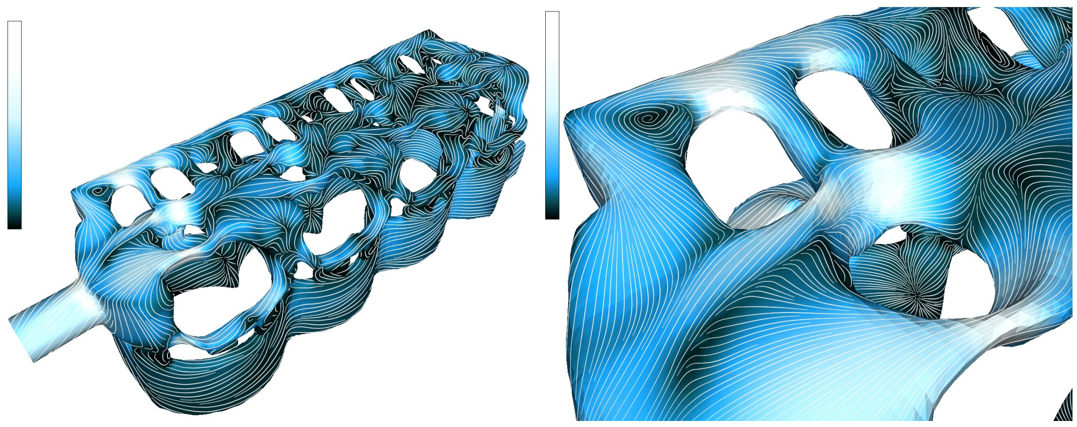

Figure 2: Visualization of flow at the surface of a cooling jacket. The first image presents an overview of the surface. The second image focuses on the bottom left-hand corner of the jacket. The mesh is comprised of approximately 227,000 adaptive resolution polygons. Detailed images of sample grids have been presented earlier [11].

present a fast and simple glyph placement algorithm to investigate and visualize flow data based on unstructured, adaptive resolution boundary meshes from CFD yielding the following benefits:

- Glyphs are automatically placed at evenly-spaced intervals, independent of how complex or dense the underlying adaptive resolution mesh is.
- The user can interactively and intuitively control the spatial resolution of the glyph placement as well as their precise location.
- Multi-resolution visualization of the flow at surfaces can be applied to increase detail in areas deemed interesting by the user.
- Glyphs are never generated for occluded or otherwise invisible regions of the surfaces.
- The algorithm is fast, enabling novel user interaction such as zooming, translating and rotation.
- Our approach enables various representations of the flow, optimized for either speed or accuracy, in a natural way.

The algorithm relies neither on pre-processing of the data nor on parameterization of the surface. It also handles large numbers of polygons efficiently. The key to the algorithms speed and simplicity is transferring computation that would normally take place in object space to image space. The approach is especially useful because engineers often start their investigation of simulation results by looking at the surface for an overview.

A second family of visualization 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 behavior 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 2).

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 describe a 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 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 that:

- Streamlines are always evenly-spaced in image space, regardless of the resolution, geometric complexity or orientation of the underlying mesh.
- 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.
- 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.
- 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 parameterization of the surface.
- 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.

2 Related Work

Ward [23] states that glyph-based visualization has been widely used to convey various information simultaneously by employing intuitive graphs to depict corresponding various variables from abstract data sets. Our work focuses on applying this intuitive depiction in image-space as well as developing an efficient and fast glyph placement algorithm to illustrate the vector field accurately. Previously, related techniques have been proposed in order to improve glyph-based visualization. In this section we describe these related techniques.

2.1 Vector Field Glyph Placement

Vector field glyph placement has received comparatively little attention. A vector glyph placement approach is described by Klassen and Harrington [9]. Three-dimensional glyphs are placed at regularly-spaced intervals on a 2D plane. Shadows on the plane are added to the glyphs to highlight their orientation. In order to depict the vector fields on curvilinear and unstructured grids, Dovey [3] presents a vector glyph placement algorithm for slices through 3D curvilinear and unstructured grids. He describes two different object-space approaches for resampling a vector field defined on a 3D unstructured or curvilinear grid onto a regular planar slice. The most computationally expensive part of the procedure for interpolating a simulation result value onto an arbitrary new point is locating the cell that contains the point. This process can be very costly in terms of processing time even when spatial data structures are used to accelerate the search. Hong et al [5] use volume rendered vector glyphs which are generated from pre-voxelized icon templates to describe regular, structured vector fields in 3D space. Incremental image updates which re-compute only those pixels on the image plane affected by user input make visualization of the scalar and vector field faster and more interactive. Laramee describes an object-space approach using resampling and vector glyph placement for slices through unstructured, 3D CFD meshes [10]. The algorithm we describe here is conceptually similar but raises the spatial dimensionality to surfaces (as well as planar slices). Our algorithm is also faster, simpler, and more efficient. In fact we are surprised not to find any previous work that provides an elegant and fast solution to the basic problem we are addressing.

2.2 Evenly-Spaced Streamlines in 2D and 3D

Turk and Banks introduce the first evenly-spaced streamline strategy [21]. 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. [15]. They adapt the aforementioned energy function to work in 2D computational space analogous to the way that Forssell and Cohen [4] extended the original LIC algorithm [1] to curvilinear grids. The Turk and Banks algorithm [21] is enhanced by Jobard and Lefer [6] 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 [7] and multiresolution versions of the algorithm [8] have been implemented. Mebarki et al. [17] introduce an alternative approach to that of Jobard and Lefer [6] by using a search strategy that locates the largest areas of the spatial domain not containing any streamlines. Liu and Moorhead [14] present another alternative approach capable of detecting closed and spiraling streamlines. Li et al. [12] describe a seeding approach that resembles hand-drawn streamlines for a flow field.

Mattausch et al. [16] 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 [13]. 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. [22] and Ye et al. [24]. 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. [2] also use a topology-based seeding strategy.

3 Glyph and Streamline Placement Algorithms

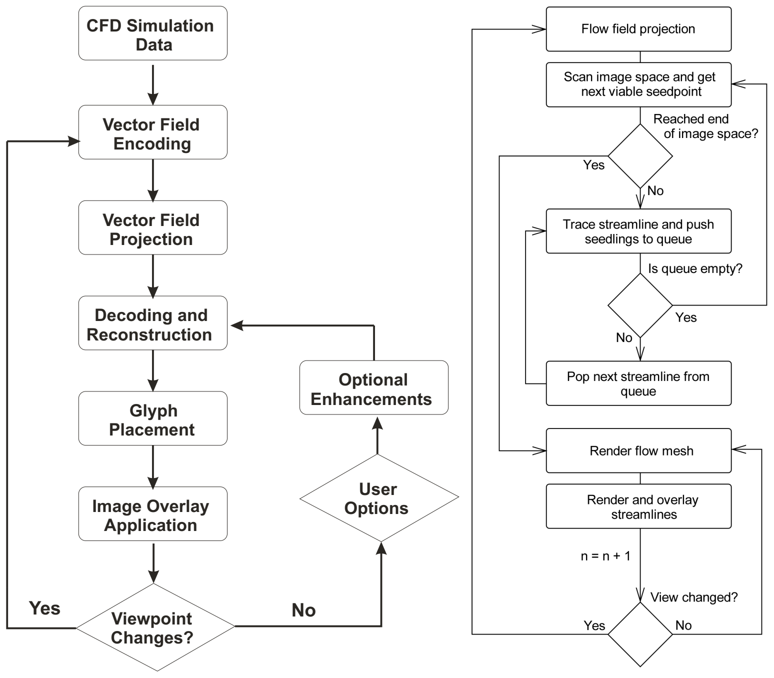
CFD Simulation Data
Vector Field Encoding
Vector Field Projection
Decoding and Reconstruction
Glyph Placement
Image Overlay Application
Viewpoint Changes?
Yes
No
User Options
Optional Enhancements
Flow field projection
Scan image space and get next viable seedpoint
Reached end of image space?
Yes
No
Trace streamline and push seedlings to queue
Is queue empty?
Yes
No
Pop next streamline from queue
Render flow mesh
Render and overlay streamlines
n = n + 1
View changed?
Yes
No

Figure 3: (first) An overview chart of the glyph placement algorithm for the fast generation and simple placement of vector field glyphs for surfaces. (second) An overview diagram for generating evenly-spaced streamlines on surfaces. Here, n is the frame number.

For the glyph placement algorithm, first the vector field is projected from 3D object space to 2D image space, this is done by exploiting graphics hardware. The vector field on the boundary surface from the CFD data set is encoded into the frame buffer. This is followed by both flow reconstruction and glyph placement. The vector field is reconstructed based on the user-defined resolution of an image-based Cartesian mesh. Then the vector glyphs are rendered along with the original surface geometry image overlay. An overview of this process is depicted in Figure 3. Several enhancements can be added including various interaction techniques as well as multi-resolution visualizations. Many different user options are available following the reconstruction and glyph placement phases in order to depict the vector field accurately and interactively. It's also worth mentioning that if viewpoint is changed after the final glyph rendering, the next pass will start from the encoding phase. Only a subset of the algorithm is required, starting with decoding and reconstruction if the user-defined resampling parameters are changed. More details are given by Peng and Laramee [18].

The streamline placement algorithm overcomes many difficulties by performing streamline integration in image space utilizing a multi-pass 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 3. More details are given by Spencer et al. [20].

4 Performance and Results

Data Set	Resampling Rate (FPS)			
	Sub-sampling	Average	Linear	Gaussian
Ring (10K)	59(29)	2.5(2.0)	30(17)	30(16)
Combustion Chamber (79K)	59(20)	1.9(1.8)	29(11)	29(12)
Intake Port (221K)	59(11)	2(1.5)	29(8)	30(7.5)
Cooling Jacket (228K)	59(9.5)	1.9(1.7)	29(8.2)	29(7.8)

Table 1: Sample frame rates for the glyph placement algorithm applied with 15^2 fixed resolution of user-defined resampling grid with about 75% image space area covered. An image of 512^2 pixels is used.

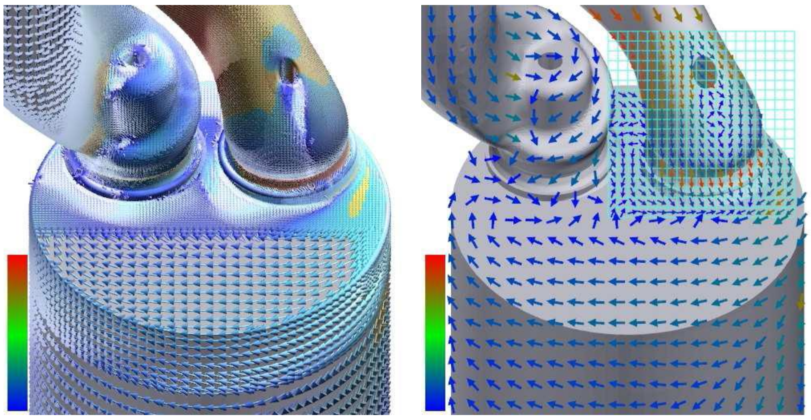

Figure 4: The comparison of brute-force hedgehog visualization (first) and our multi-resolution glyph-based visualization which is using a Gaussian filter (second) applied in order to depict the flow at a surface of an intake port mesh composed of unstructured, adaptive-resolution 221K polygons. Notice how the glyphs are cluttered using the hedgehog approach (first image). Also notice that artifacts appear resulting from the underlying mesh that have nothing to do with the actual flow. Glyphs are color-coded according to velocity magnitude.

As our glyph-based visualization is focused on unstructured, adaptive resolution boundary meshes from the complex CFD data sets, we evaluate our visualization on simulation data sets with these characteristics. Figure 4 shows a comparison of brute-force hedgehog placement and our glyph-based method applied on a surface of an intake port mesh composed of 221K polygons. The intake port has highly adaptive resolution boundary surface and for which no global parameterization is easily computed. As we can see from the first picture,

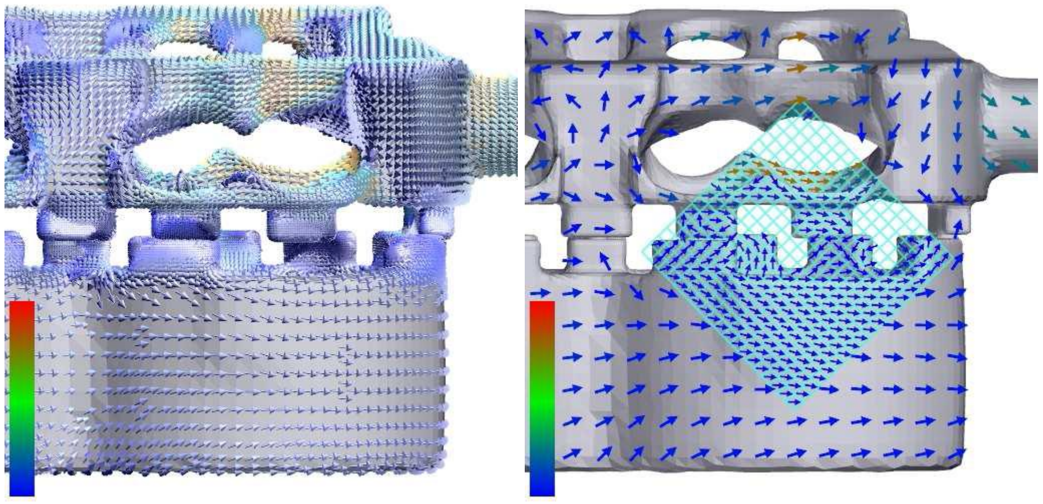

Figure 5: Another comparison of brute-force hedgehog flow visualization (first) and glyph-based flow visualization which is powered by Gaussian filter and multi-resolution (second) applied at the surface of a cooling jacket - a composite of 228K unstructured, adaptive-resolution polygons.

most glyphs overlap or are occluded. Using a hedgehog approach 664k glyphs are rendered. However, our approach renders only about 400 glyphs. Also, the distribution of glyphs is uneven. These artifacts are a result of the underlying mesh and have no relation to the flow itself. In the second, our method places glyphs in an intuitive and efficient fashion enabling engineers to get a fast and clear overview of the flow on the surface. At the same time, with the help of a multi-resolution option, more details on the interesting areas can be obtained. The vector field on the complex cooling jacket boundary meshes (from Figure 1) can be also efficiently visualized by our intuitive glyph-based method (Figure 5), especially compared to a hedgehog visualization. Because of the fast speed of our method this glyph-based visualization allows users to translate, rotate and zoom in the object interactively to get better insight of the CFD data sets. We encourage the reader to view the supplementary video for more results.

In order to compare the various reconstruction options implemented in our framework, we evaluated sub-sampling, averaging, the linear filter function and the Gaussian filter function on a PC with an Nvidia Geforce 8600GT graphics card, a 2.66 GHz dual-processor and 4 GB of RAM. The performance times reported in Table 1 were obtained using a fixed 15^2 resolution resampling grid with about 75% image space coverage. The first times illustrated in the FPS column are for the static case of (no change to the view point) only changes to the user options. The times shown within parenthesis depict the dynamic case of changes to the viewpoint. In terms of the overview chart presented in Figure 3, the construction of a velocity image, image overlay, reconstruction of vector field, as well as glyph placement need to be computed in the dynamic case. From Table 1, we can see that sub-sampling is the fastest while averaging is the slowest. Linear and Gaussian filter functions are in the middle as a balance between computation speed and high accuracy. More detail about the performance of the glyph placement algorithm can be found in Peng and Laramee [18].

We tested the streamline placement algorithm on a range of datasets taken from complex CFD simulations. To obtain high-quality results, we use a second-order Runge-Kutta par-

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

Table 2: 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.

ticle 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 off-loading 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 6 uses high-detail data from the computed flow through two intake ports. Here, the color 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 streamline integration. Further results are illustrated in the accompanying video.

The dataset in Figure 2 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 parameterization would be especially difficult owing to the complex topology of the shape.

In Figure 7 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 dataset 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 2 compares the time taken to integrate streamlines over the velocity image for each of

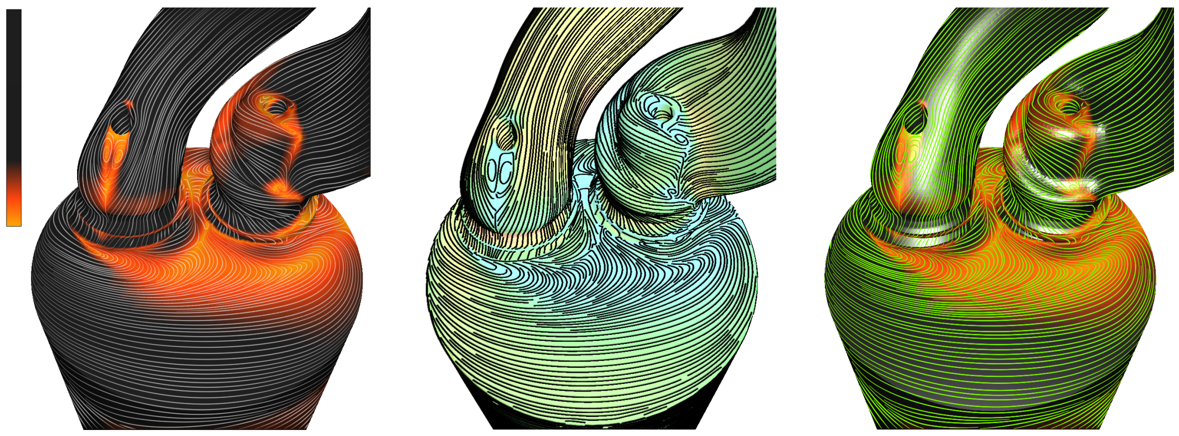
Figure 6: The visualization of flow at the boundary surface of two intake ports. (Left) With our novel, image-based streamlines. (Middle) With full-precision, object-based streamlines computed on the CPU. (Right) High-contrast, image-based streamlines. This mesh is comprized of approximately 222,000 polygons at an adaptive resolution.

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 seconds 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 [14], however we have deliberately kept our implementation simple so as to concentrate on extending it to a higher spatial dimension.

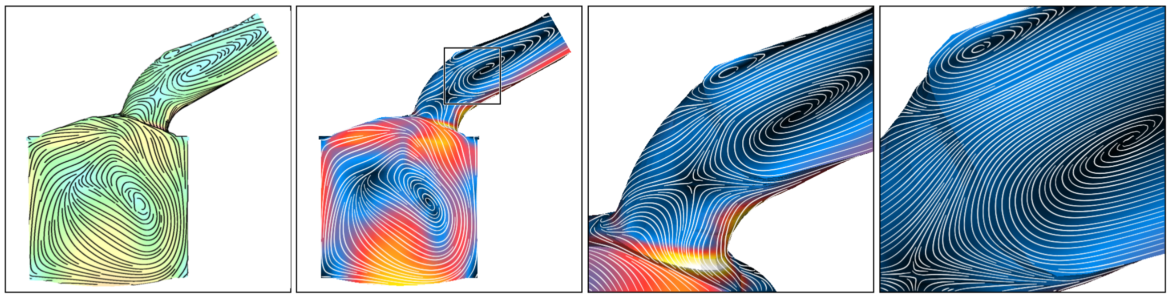
Figure 7: Zooming: Visualization of the flow at the surface of a gas engine simulation at progressively higher levels of magnification. The left-most image was generated using a full floating-point, object-based algorithm computed on the CPU. The successive images were generated using our novel, image-based technique.

5 Conclusion and Future Work

In the first part of this paper we propose a fast and simple glyph placement algorithm for investigating and visualizing boundary flow data based on unstructured, adaptive resolution boundary meshes from CFD. We show that the algorithm effectively and automatically places glyphs at evenly-spaced intervals, independent of geometric and topological complexity of the underlying adaptive resolution mesh. We have also demonstrated that the spatial resolution and precise location of the glyph placement can be interactively and intuitively adjusted by the user in order to gain better visualization results. In addition, multi-resolution visualization can be applied to highlight details in areas deemed interesting by the user. Furthermore, the efficiency of our algorithm is reinforced by the fact that no computation time is wasted on occluded polygons or polygons covering less than one pixel. Due to the efficiency and speed of the algorithm user interaction such as zooming, translating and rotation is enabled. The framework supports various representations of the flow optimized for both speed and accuracy. No pre-processing of the data or parameterization is required.

The second constituent of this paper describes an, image-based technique for generating evenly-spaced streamlines over surfaces. We have shown that our algorithm effectively places streamlines on datasets with arbitrary topological and 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 dataset. The accuracy of the visualization is demonstrated by comparing the results of image- and object-based approaches.

As future work we would like to extend the work to visualization of unsteady, 3D (volumetric) flow. Challenges stem from both the resampling performance time and perceptual issues. Future work also includes using floating-point texture in order to encode and decode the vector field.

6 Acknowledgements

This work was supported by EPSRC research grant EP/F002335/1.

References

[1] B. Cabral and L. C. Leedom. Imaging Vector Fields Using Line Integral Convolution. In *Poceedings of ACM SIGGRAPH 1993*, Annual Conference Series, pages 263–272, 1993.

[2] G. Chen, K. Mischaikow, R. S. Laramee, P. Pilarczyk, and E. Zhang. Vector Field Editing and Periodic Orbit Extraction Using Morse Decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):769–785, Jul/Aug 2007.

[3] D. Dovey. Vector Plots for Irregular Grids. In *Proceedings IEEE Visualization '95*, pages 248–253, 1995.

[4] L. K. Forssell and S. D. Cohen. Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):133–141, June 1995.

[5] L. Hong, X. Mao, and A. E. Kaufman. Interactive Visualization of Mixed Scalar and Vector Fields. In *Proceedings IEEE Visualization '95*, pages 240–247, 1995.

[6] B. Jobard and W. Lefer. Creating Evenly–Spaced Streamlines of Arbitrary Density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing '97*, volume 7, pages 45–55, 1997.

[7] B. Jobard and W. Lefer. Unsteady Flow Visualization by Animating Evenly-Spaced Streamlines. In *Computer Graphics Forum (Eurographics 2000)*, volume 19(3), pages 21–31, 2000.

[8] B. Jobard and W. Lefer. Multiresolution Flow Visualization. In *WSCG 2001 Conference Proceedings*, pages 33–37, Plzen, Czech Republic, February 2001.

[9] R. V. Klassen and S. J. Harrington. Shadowed hedgehogs: A technique for visualizing 2D slices of 3D vector fields. In *Proceedings IEEE Visualization '91*, pages 148–153, 1991.

[10] R. S. Laramee. FIRST: A Flexible and Interactive Resampling Tool for CFD Simulation Data. *Computers & Graphics*, 27(6):905–916, 2003.

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

[12] L. Li, H.-S. Hsieh, , and H.-W. Shen. Illustrative Streamline Placement and Visualization. In *IEEE Pacific Visualization Symposium 2008*, pages 79–85. IEEE Computer Society, 2008.

[13] L. Li and H.-W. Shen. Image-Based Streamline Generation and Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):630–640, 2007.

[14] Z. P. Liu and R. J. Moorhead, II. An Advanced Evenly-Spaced Streamline Placement Algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):965–972, September 2006.

[15] X. Mao, Y. Hatanaka, H. Higashida, and A. Imamiya. Image-Guided Streamline Placement on Curvilinear Grid Surfaces. In *Proceedings IEEE Visualization '98*, pages 135–142, 1998.

[16] O. Mattausch, T. Theussl, , H. Hauser, and E. Gröller. Strategies for Interactive Exploration of 3D Flow Using Evenly-Spaced Illuminated Streamlines. In *Proceedings of the 19th Spring Conference on Computer Graphics*, pages 213–222, 2003.

[17] A. Mebarki, P. Alliez, and O. Devillers. Farthest Point Seeding for Efficient Placement of Streamlines. In *Proceedings IEEE Visualization 2005*, pages 479–486. IEEE Computer Society, 2005.

[18] Z. Peng and R. S. Laramee. Vector Glyphs for Surfaces: A Fast and Simple Glyph Placement Algorithm for Adaptive Resolution Meshes. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2008*, pages 61–70, 2008.

[19] T. Ropinski and B. Preim. Taxonomy and Usage Guidelines for Glyph-based Medical Visualization. In *Proceedings of the 19th Conference on Simulation and Visualization (SimVis08)*, pages 121–138, 2008.

[20] B. Spencer, R. S. Laramee, G. Chen, and E. Zhang. Evenly-Spaced Streamlines for Surfaces: An Image-Based Approach. *Computer Graphics Forum*, 28, 2009. forthcoming.

[21] G. Turk and D. Banks. Image-Guided Streamline Placement. In *ACM SIGGRAPH 96 Conference Proceedings*, pages 453–460, August 1996.

[22] V. Verma, D. Kao, and A. Pang. A Flow-guided Streamline Seeding Strategy. In *Proceedings IEEE Visualization 2000*, pages 163–170, 2000.

[23] M.O. Ward. A Taxonomy of Glyph Placement Strategies for Multidimentional Data Visualization. *Information Visualization*, 1(3-4):194–210, 2002.

[24] X. Ye, D. Kao, and A. Pang. Strategy for Seeding 3D Streamlines. In *Proceedings IEEE Visualization 2005*, pages 471–476, 2005.