Vector Glyphs for Surfaces: A Fast and Simple Glyph Placement Algorithm for Adaptive Resolution Meshes

Zhenmin Peng, Robert S. Laramee

Department of Computer Science
Swansea University, Wales UK
Email: {cszp, r.s.laramee}@swansea.ac.uk

Abstract

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. 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.

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

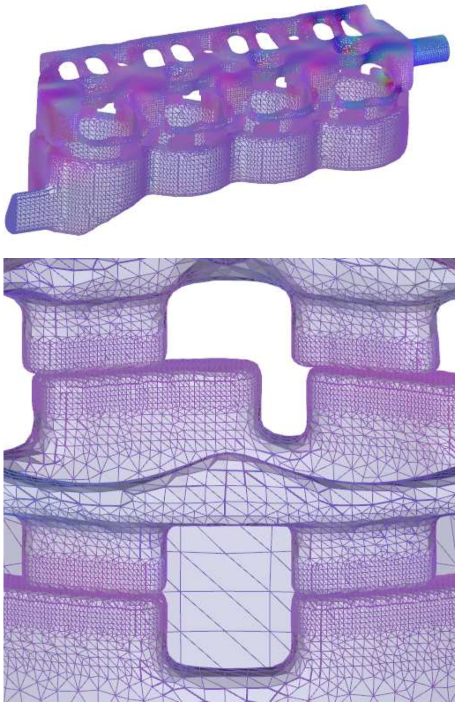

Figure 1: *The unstructured adaptive resolution boundary grid of a cooling jacket from a CFD simulation. The upper image is an overview of the boundary mesh, and the bottom is a close-up. These images illustrate how complex a typical mesh from CFD can be.*

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 [10] [12], glyphs for vector field visualization have received relatively little attention. This may be due to the difficulties in placing glyphs on unstructured, adaptive resolution boundary meshes from the complex CFD data sets and perceptual problems like visual complexity and occlusion. In order to address these challenges, we 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.

The rest of the paper is organized as follows: Section 2 provides an overview of related research work. The placement algorithm and user options are described in multiple stages in Section 3. Section 4 gives the performance and visualization results. Conclusions and suggestions for future work are presented in Section 5.

2 Related Work

Ward [12] 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. We emphasize the glyph placement related techniques in two main categories: tensor field and vector field data. Within each category, techniques are discussed with respect to the dimensionality of the given data: 2D, 2.5D (for surfaces in 3D) and 3D.

2.1 Tensor Field Glyph Placement

The majority of related work has not focused on vector field glyph placement but rather tensor field glyph placement. Laidlaw et al. apply an elliptical tensor field glyph placement algorithm for the visualization of 2D Diffusion Tensor Image (DTI) data from the spinal cord of a mouse [7]. The regular array of ellipsoids are normalized by size for a visualization that is more easily deciphered. Instead of placing tensor glyphs on a regular-Cartesian array, Kindlmann and Westin present a glyph placement algorithm that shapes and positions the glyphs in a smooth and continuous fashion resulting in a visualization free of holes and without overlapping glyphs [5]. The artifacts of the underlying grid structure then disappear. Hlawitschka et al. [2] present an accelerated version of the tensor field glyph packing algorithm. The goal of their algorithm is to support interactive data exploration.

Additionally, a surface-based (2.5D) glyph placement strategy for medical visualization is proposed by Ropinski et al. [9]. The algorithm works in object space and is based on isosurfaces. The volume is searched voxel-by-voxel for locations through which the chosen isosurface passes. Afterwards a glyph is placed at every cell that encompass the given isovalue such that it's located on the specified isosurface. Axis aligned rays are cast into the volume in order to detect visible portions of the isosurface. Then only visible portions remain in the final rendering. An approach that requires volume searching, isosurfacing, and ray casting is overly complex and not optimized for speed. Sigfridsson et al. [11] present a hybrid volume rendering and glyph-based visualization for 3D tensor data based on interactive glyph placement. The glyphs are placed manually with a 3D cursor.

2.2 Vector Field Glyph Placement

Vector field glyph placement has received comparatively little attention. A vector glyph placement approach is described by Klassen and Harrington [6]. 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 [1] 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 [3] 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 [8]. 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.

3 Intuitive Glyphs on Surfaces

This section presents the details of the algorithm starting with a short discussion of why we chose an image-based approach.

3.1 Object Space vs. Parameter Space vs. Image Space

In order to construct a fast and simple glyph placement algorithm on surfaces, we develop a reasonable approach which can deal with large and complex flow data sets from CFD efficiently and interactively.

One possible solution to depict the flow data on the surfaces is to render glyphs directly in object space. Placing a glyph at each data sample point constructs the well-known hedgehog visualization. However, typical drawbacks are obvious. Most of the glyphs may be occluded. This is especially true of the example shown in Figure 1 (The cooling jacket). Furthermore, glyphs will either be too large, resulting in visual clutter or too small to perceive.

Secondly, parameter space is another possible approach. If a global parameterization of the surface can be computed then the challenges posed by glyph placement are simplified. But the process of parameterizing the surface globally is very complex. CFD data sets contain a large number of polygons. Some involve an especially complex topology. Also parameterization may result in some distortion when the parameterized surface is mapped back on to physical (3D) space.

Image space is a good alternative to address these challenges. With the use of image space, a 3D vector field can be projected onto the 2D image plane to simplify the problem. That means only visible polygons are sampled and no extra time is spent on generating glyphs for the polygons hidden from the user's view-point. The problem of how to properly place glyphs to represent the vector field on the surfaces in 3D space is then greatly simplified to finding the optimal placement in 2D space. By exploiting this approach, user interaction techniques, which would otherwise not be possible, may be applied. However using an image-based approach does bring new challenges, both conceptual and technical. We describe these challenges in detail in the sections that follow.

3.2 Method Overview

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 2. 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 in the sub-sections that follow.

3.3 Vector Data Encoding and Projection

The vector field values of the boundary are stored at the vertices of the polygonal CFD mesh. A key step into our algorithm is to project the vector field defined at the boundary surface to the image plane. In order to realize this, we use the approach in which the *x*, *y*, and *z* component values of the vector stored at each vertex of the boundary surface are encoded into *r*, *g*, and *b* color values in framebuffer respectively. The formula we use to encode the vector components is the following:

$$c_{r,g,b} = \frac{v_{x,y,z} - v_{min(x),(y),(z)}}{v_{max(x),(y),(z)} - v_{min(x),(y),(z)}} \qquad (1)$$

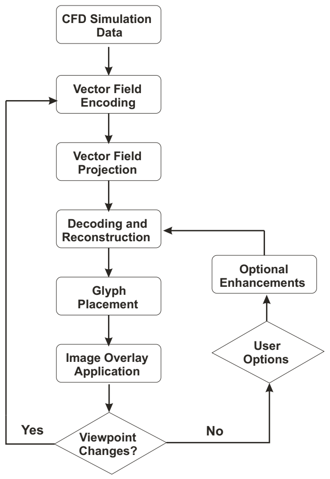

Figure 2: *An overview chart of our algorithm for the fast generation and simple placement of vector field glyphs for surfaces.*

Encoding the vector component values in this way yields the following benefits:

- Occluded or otherwise hidden portions of the geometry are automatically culled and are thus eliminated from any further processing.
- Linear interpolation of the vector field is performed automatically by the graphics card hardware.
- No further computation time is spent on polygons whose size is less than one pixel, the occurrence of which is high for CFD meshes (see Figure 1).
- The complexity of placing glyphs in object space is reduced to a much simpler problem in image space.

After the component-wise encoding of the vector values, a velocity image is generated. And it's also worth pointing out that the interpolation of velocity image is necessary for vector field reconstruction. With the help of hardware-assisted interpolation, we are able to decode the vector field values within the original boundary mesh polygons in addition to the vertices. Transferring the velocity image to the main memory completes the vector field projection process.

3.4 Decoding

Following the projection of the vector field defined at the surface, reconstruction for the vector field is an essential stage for developing an optimal glyph placement algorithm. Before discussing the details of the reconstruction, we describe the decoding phase for the reconstruction. The process of decoding the velocity vector values *x* from the velocity image is performed according to the following:

$$v_x = c_r \cdot (v_{max(x)} - v_{min(x)}) + v_{min(x)} \quad (2)$$

Vector values of *y* and *z* can be achieved in the same way like equation (2). The decoded vector field values used to reconstruct the flow and render the glyphs are then projected onto the image plane. Technically, the velocity at the boundary surface is defined to be zero (no slip boundary condition). What we see here is an extrapolation of the vector field just under the surface to the boundary.

3.5 Vector Field Reconstruction

After the vector field has been projected to the image plane we perform flow reconstruction. There are many different options when considering the best approach to representing the vector field including sub-sampling, using first-order or bilinear interpolation, and using box, linear and Gaussian interpolation filter functions. Representations may be optimized for speed or for accuracy. In our implementation, we offer options optimized for both. We describe the reconstruction options in more details in the sub-sections that follow.

3.5.1 Sub-sampling

The fastest and simplest way to represent the vector field with glyphs is to use sub-sampling. A rectilinear grid, the resolution of which is defined by the user, is placed in image space. The vector field is then sampled at the center of each grid cell using the decoding described in the previous section. A vector glyph is rendered at the center of each grid cell based on the sample. In the implementation, some measure must be taken to ensure that no glyphs are rendered at cell centers with the background color. This can be handled either by an explicit test for background color or by testing the depth buffer for its maximum value. We chose the option of testing the z-buffer value. No glyphs are rendered for cell centers where $z_{depth} = 1.0$.

The advantage of this approach is speed. The user may sample the vector field at several frames per second, rotating the resampling grid, changing its resolution and sliding the grid in image space in order to place the glyphs precisely where the user chooses. We note that these user options of specifying the resampling grid resolution, translating and rotating the grid are not arbitrary. They were

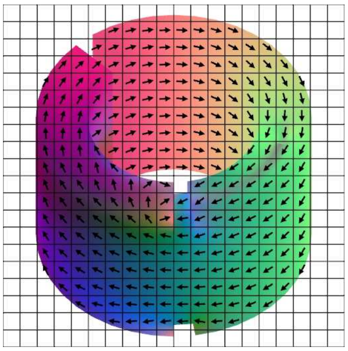

Figure 3: *Glyphs are rendered at the resampling grid cell centers for visible portions of the boundary geometry and its associated vector field. This example shows a simple ring geometry with a* 20^2 *resampling grid.*

specifically requested by fluid engineers we talked to when developing an earlier tool [8]. Fluid engineers want precise control over glyph placement and resampling parameters.

3.5.2 Average-based representation

Although the above sub-sampling approach provides the desired speed to investigate the simulation result, it does not construct the most accurate representation of the flow since only a sub-set of samples are taken into account. Therefore we also provide the option of rendering glyphs based on the average vector field value of each user-defined resampling grid cell. Instead of sampling the vector field only at the center point of each cell, this approach samples the vector field pixel-by-pixel over the whole cell. However, a complication can arise with this approach due to discontinuities on the surface. If we do not take edges in the geometry into account then we may end up including undesired velocity values into the final average. To address this problem, we also sample the depth value at each pixel during the average computation. If the depth values of the center grid cell point, $\mathbf{p}_{center}$, and another sample within the same cell, $\mathbf{p}_{sample}$, differ by more than a threshold value, ε_{depth}, then the pixel with $\mathbf{p}_{sample}$ is not included in the final average. This approach separates the image into distinct regions for accurate flow reconstruction. We emphasize that, in all of our filters, any pixels beyond edge discontinuities are left out of the final result. We do so in order to separate different regions of the geometry that may be at the same depth.

The averaging approach provides high accuracy for representing the flow. The user can gain a precise overview of the vector field based on the boundary surface via the intuitive center glyph without missing any potentially interesting values. One drawback with this approach however is its cost in computing time.

3.5.3 Reconstruction using filters

In order to construct an optimal approach which combines the speed of sub-sampling and the accuracy of an averaging-based representation, we use footprint functions with various interpolation filters. We can use various filters (or filter kernels) to reconstruct the flow at the boundary surface. There are many possible interpolation functions, namely n^{th} order filters where $n = 0,1,2,\ldots$ all with relative advantages and disadvantages (see [4] for an overview). We have implemented linear (or first order) and Gaussian filters in our framework. In order to accelerate the computation, we have implemented these filters as footprint functions similar to Westover [13]. Our image-based resampling approach makes this a natural choice for simple, symmetric 2D footprint functions. Using a 2D footprint function with an elliptical extent, vector field reconstruction is accelerated by pre-computing the contribution of each sample in the footprint's extent and storing it in a look-up table. The extent of each footprint simply encompasses one grid cell in the user-defined resampling grid as shown in Figure 4. Furthermore, the footprint represented by each glyph is the same except for a screen space offset. An illustration showing an example footprint and filter kernel is shown in Figure 4.

While we believe the use of filter functions implemented using footprint tables represents a balanced trade-off between accuracy and speed, complications arise due to discontinuities in the vector field stemming from mesh boundaries and edges.

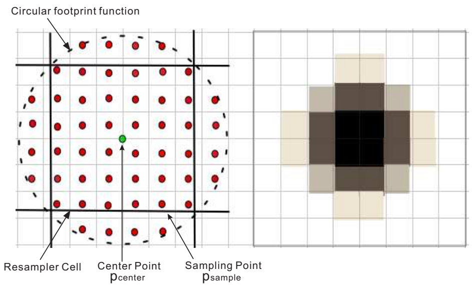
Circular footprint function
Resampler Cell
Center Point p_{center}
Sampling Point p_{sample}

Figure 4: *Left is the circular footprint function using 8^2 sized footprint-grid. On the right side, an 8^2 footprint look-up table with a Gaussian filter kernel.*

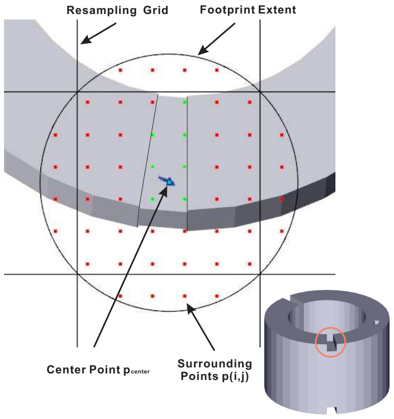

Figure 5: *The circular footprint function with Gaussian filter kernel is applied at the surface of the ring which has edges as shown. The blue point is the center point of the resampling cell. Green and red points are sample points defined by the footprint function. Green contribute to the final representation whilst red have been filtered out by equation (3).*

If we simply contribute each sample's vector values to the filter function we may not get an accurate visualization of the flow at each resampling grid cell center. This is because the center points may lie on a different portion of the mesh than surrounding sample points. Figure 5 shows a center-point $\mathbf{p}_{center}$ where a glyph representing the flow is rendered. Also shown is the extent of the footprint function surrounding $\mathbf{p}_{center}$. The extent of the footprint encompasses more samples including undesired ones like $\mathbf{p}_{i,j}$ (shown in red) around $\mathbf{p}_{center}$. Thus we must introduce an additional filtering operation to disregard the contribution of red points $\mathbf{p}_{i,j}$ to $\mathbf{p}_{center}$. We achieve this by taking the depth gradient between $\mathbf{p}_{center}$ and $\mathbf{p}_{i,j}$ into account. If

$$\varepsilon_{depth} > |depth(p_{center}) - depth(p_{i,j})| \qquad (3)$$

then samples $\mathbf{p}_{i,j}$ contribution is not added to the final value at $\mathbf{p}_{center}$. Here ε_{depth} is a user-defined threshold. In practice, we have found a value of 0.003 to be a good threshold value.

In order to implement the filtering operation, we use a neighbor-based selection method to select sampling points $\mathbf{p}_{i,j}$ accurately and efficiently. Equation (3) helps us detect relevant portions of the boundary geometry to sample and which samples to filter out. We test each sample within the extent of the center point's footprint kernel. We start at

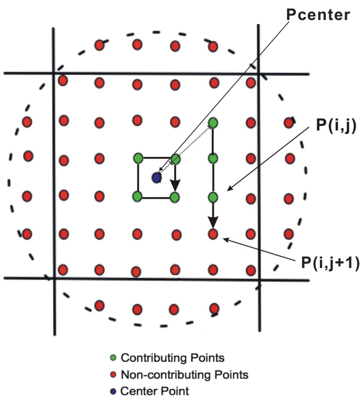

Figure 6: *Candidate samples are found by searching the kernel extent in a looping fashion starting at $\mathbf{p}_{center}$. Sample points $\mathbf{p}_{i,j}$ do not contribute if equation (3) is true.*

$\mathbf{p}_{center}$ and test each adjacent neighbors in a looping fashion. If we find contributing points $\mathbf{p}_{i,j}$ in the first loop of neighbors surrounding $\mathbf{p}_{center}$ then we increase the radius of the loop by one unit and repeat the process. If a discontinuity is found, all neighbors beyond the discontinuity are filtered out. This iterative process stops when either (1) a loop with no contributing samples is found or (2) the extent of the footprint is reached. An illustration is shown in Figure 6. $\mathbf{p}_{i,j+1}$ is a neighbor from $\mathbf{p}_{i,j}$. Once $\mathbf{p}_{i,j+1}$ is rejected, the unvisited neighbor points beyond $\mathbf{p}_{i,j+1}$ are not needed.

We point out that we have implemented a sub-set of possible filters. Our framework however offers any number of filters to be plugged in and used in a natural and easy way.

3.6 Image Overlay Application

Along with the glyph placement, an optional image overlay is used for the resulting visualization of the vector field on surfaces by applying color, shading, or any attribute mapped to color. In the implementation, we generate the image overlay following the construction of the velocity image once for each static scene. Once the view-point is changed, image overlay needs to be regenerated. By exploiting OpenGL's glDrawPixels() function, rendering an image is much faster than rendering the complex triangulated object each time a user parameter is changed.

3.7 Glyph Placement, User Options and Enhancements

Many user options can be applied to enhance the usability and flexibility of glyph placement. The

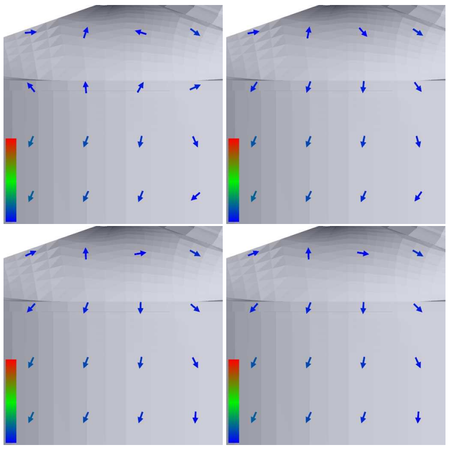

Figure 7: *A close-up view of the vector field on the surface of an 79K polygonal gas engine simulation mesh. Here we illustrate a comparison of various vector field reconstruction options: (top, left) Sub-sampling, (top, right) Averaging, (bottom, left) Linear filtering and (bottom, right) Gaussian filtering on red color circulated area. It's worth pointing out that accuracy of depicting vector field around the edge between the cap and the cylinder via Linear and Gaussian filter is quite similar to the result from the averaging, but with much less computational cost than averaging.*

user options enable engineers to gain more insight of the flow on boundary surfaces. We describe these user options in more detail individually. Many of the user-interaction techniques we describe would not be possible with an object space approach.

- User-Defined Resampling Grid Resolution: The user may interactively specify the resolution of the resampling grid in image space. The higher the resolution, the more accurate the linear and Gaussian filter kernels become. The lower the resolution, the faster the interaction becomes. Users desire faster speed for interaction and higher accuracy for analysis and presentation.
- Grid Translation and Rotation: In order to precisely place the vector field glyphs at user defined points, we provide the option of moving the resampling grid around as well as rotating the grid around the center point. These options were specifically requested by CFD engineers, as well as control of the resampling grid resolution.
- Glyph Scaling: This user option adjusts the size of each glyph in order to avoid overlap-

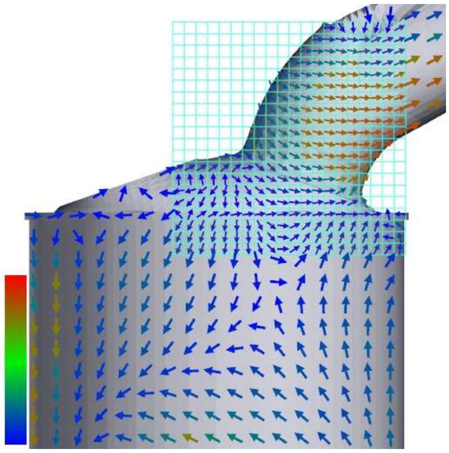

Figure 8: *A colored multi-resolution visualization of low resolution and high resolution glyphs applied with Gaussian filter is rendered to visualize the flow at the surface of a gas engine simulation. Color is mapped to velocity magnitude.*

ping and occlusion of glyphs. With the help of glyph scaling, glyphs can be rendered in proper size to make the vector field easily perceived.

- Multi-Resolution Flow visualization: Our tool also offers a multi-resolution representation of the flow. The user may define a sub-grid, with its own higher (or lower) grid resolution. The user can then position this sub-grid over any area of interest interactively. All of the options built into our framework can be applied to the sub-grid as well: different reconstruction options, as well as grid scaling and rotation (see Figure 8).

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 visualization 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.*

We also point out that the accuracy of the vector field representation increases automatically when the user zooms in on a boundary. The higher sampling frequency is a natural benefit of the approach. One of the consequences of using an image-based approach is that the glyphs remain fixed in their positions as the object moves under rotation or translation. This can be handled by unprojecting the vector glyphs back to the object-space surface. However glyphs are not generated for portions of the geometry that are occluded or outside the current view. Finding a perfect solution to this problem is a part of our future work.

Data Set	Resampling Rate (FPS)							
	5^2		10^2		20^2		50^2	
	SS	Gaussian	SS	Gaussian	SS	Gaussian	SS	Gaussian
Ring (10K)	59(29)	59(28)	59(29)	59(20)	59(29)	15(11)	58(23)	2.5(2.4)
Combustion Chamber (79K)	59(20)	59(19)	59(20)	58(20)	59(20)	13(9.5)	58(20)	2.3(2)
Intake Port (221K)	59(11)	59(10)	59(11)	58(8.5)	59(9.7)	15(6.2)	58(10)	2.4(1.7)
Cooling Jacket (228K)	59(10)	58(9.4)	59(9.5)	58(7.4)	59(10)	14(6.5)	59(9.4)	2.4(1.9)

Table 2: *Sample frame rates for the visualization algorithm applied with sub-sampling (SS), a Gaussian filter function (Gaussian), varying the resolution of user-defined resampling grid and about 75% image space area covered.*

4 Performance and Results

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 9 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 top picture, 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 bottom, 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 10), especially compared to a hedgehog visu-

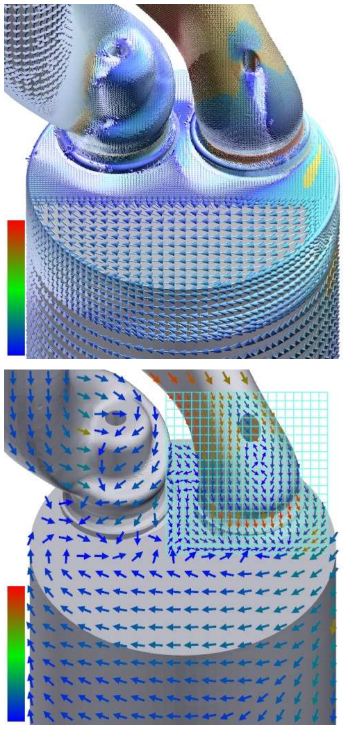
Figure 9: *The comparison of brute-force hedgehog visualization (top) and our multi-resolution glyph-based visualization which is powered by Gaussian filter (bottom) applied in order to depict the flow at a surface of an intake port mesh composed of unstructured, adaptive-resolution 221K polygons. Notice how the glyphs are cluttered using the hedgehog approach (top). 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.*

alization. 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

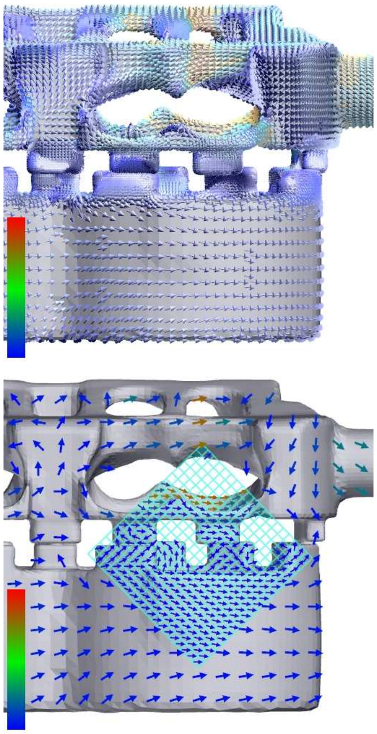
Figure 10: *Another comparison of brute-force hedgehog flow visualization (top) and glyph-based flow visualization which is powered by Gaussian filter and multi-resolution (bottom) applied at the surface of a cooling jacket - a composite of 228K unstructured, adaptive-resolution polygons.*

illustrated in the FPS column are for the static case of (no change to the view point) only changes to the user options from section 3.7. The times shown within parenthesis depict the dynamic case of changes to the viewpoint. In terms of the overview chart presented in Figure 2, 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.

Table 2 shows performance times in order to compare sub-sampling and a Gaussian filter function with various resolutions of the user-defined resampling grid. The performance time of our algorithm depends on the resolution of the user defined resampling grid used to place relevant glyphs and the number of polygons in the original surface mesh. But in the static case, the algorithm depends mostly on the coverage of the area in image space (75% is covered in our cases) rather than the number of polygons in the original surface mesh. Table 2 indicates that the higher the resolution of the resampling grid, the lower the performance. When the resolution is higher than 20^2, the performance speed drops off. Hence we use a 20^2 default resolution like in Figures 9 and 10 of the resampling grid for good performance and high accuracy. In general, our goal is to provide users for fast performance times for interaction and exploration. For presentation and analysis, users then have the option of increasing the accuracy.

5 Conclusion and Future Work

In 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.

We would like to extend the work to visualization of unsteady 3D 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. The authors would like to thank Tony McLoughlin for his help in proofreading the manuscript.

References

[1] D. Dovey. Vector Plots for Irregular Grids. pages 248–253. IEEE Visualization, 1995.

[2] M. Hlawitschka, G. Scheuermann, and B. Hamann. Interactive Glyph Placement for Tensor Fields. In *ISVC07*, pages I: 331–340, 2007.

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

[4] A. Kaufman and K. Mueller. *Overview of Volume Rendering*, chapter 1. Visualization Handbook. 2005.

[5] G. Kindlmann and C. Westin. Diffusion Tensor Visualization with Glyph Packing. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization 2006)*, 12(5):1329–1335, September-October 2006.

[6] R.V. Klassen and S.J. Harrington. Shadowed Hedgehogs: A Technique for Visualizing 2D Slices of 3D Vector Fields. *Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on*, pages 148–153, 22-25 Oct 1991.

[7] D.H. Laidlaw, E.T. Ahrens, D. Kremers, M.J. Avalos, R.E. Jacobs, and C. Readhead. Visualizing Diffusion Tensor Images of the Mouse Spinal Cord. In *IEEE Visualization '98*, pages 127–134, 1998.

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

[9] T. Ropinski, J. Meyer-Spradow, M. Specht, K.H. Hinrichs, and B. Preim. Surface Glyphs for Visualizing Multimodal Volume Data. In *Proceedings of the 12th International Fall Workshop on Vision, Modeling, and Visualization (VMV07)*, pages 3–12, nov 2007.

[10] 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)*, 2008.

[11] A. Sigfridsson, T. Ebbers, E. Heiberg, and L. Wigstrom. Tensor Field Visualisation Using Adaptive Filtering of Noise Fields Combined with Glyph Rendering. *Visualization, 2002. VIS 2002. IEEE*, pages 371–378, 1-1 Nov. 2002.

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

[13] L. Westover. Footprint Evaluation for Volume Rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 367–376, New York, NY, USA, 1990. ACM.