Journal of Advances in Science and Technology

Vol. IV, No. VII, November-2012, ISSN 2230-9659

REVIEW ARTICLE

ISA: IMAGE SPACE BASED VISUALIZATION OF UNSTEADY FLOW ON SURFACES

www.ignited.in

Journal of Advances in Science and Technology
Vol. IV, No. VII, November-2012, ISSN 2230-9659

Isa: Image Space Based Visualization of Unsteady Flow on Surfaces

Aparna Sharma

Research Scholar, CMJ University, Shillong, Meghalaya

----------♦----------

Dense, texture-based, unsteady flow visualization on surfaces has remained an elusive problem since the introduction of texture-based flow visualization algorithms themselves. The class of fluid flow visualization techniques that generate dense representations based on textures started with the Spot Noise and LIC. The main advantage of this class of algorithms is their complete depiction of the flow field while their primary drawback is, in general, the computational time required to generate the results.

Recently, two new algorithms, namely Lagrangian-Eulerian Advection (LEA) and Image Based Flow Visualization (IBFV) , have been introduced that overcome the computation time hurdle by generating two-dimensional flow visualization at interactive frame rates, even for unsteady flow. This paves the way for the introduction of new algorithms that overcome the same problems on boundary surfaces and in three dimensions. In this chapter which has also been published elsewhere we present a new algorithm, ISA Image Space Advection that generates dense representations of arbitrary fluid flow on complex, non-parameterized surfaces, more specifically, surfaces from computational fluid dynamics (CFD). However, the algorithm is general enough to apply to other vector field data associated with a surface such as blood vessel flow.

Traditional visualization of boundary flow using texture mapping first maps one or more 2D textures to a surface geometry defined in 3D space. The textured geometry is then rendered to image space. Here, we alter the classic order of operations. First we project the surface geometry to image space and then apply texturing. In other words, conceptually texture properties are advected on boundary surfaces in 3D but in fact our algorithm realizes texture advection solely in image space. The result is a versatile visualization technique with the following characteristics:

_ generates a dense representation of unsteady flow on surfaces

_ visualizes flow on complex surfaces composed of polygons whose number is on the order of 200,000 or more

_ visualizes flow on dynamic meshes with time-dependent geometry and topology

_ visualizes flow independent of the surface mesh's complexity and resolution

Figure: Visualization of flow on the surface of an intake manifold. The ideal intake manifold distributes flow evenly to the piston valves.

_ supports user-interaction such as rotation, translation, and zooming always maintaining a constant, high spatial resolution

_ the technique is fast, realizing up to 20 frames per second

The performance is due, among other reasons, to the exploitation of graphics hardware features and utilization of frame-to-frame coherency. The rest of the chapter is organized as follows: in Section we discussed related work, Section details unsteady flow visualization on surfaces from CFD. Implementation details are described in Section while results and conclusions are discussed in Section .

PHYSICAL SPACE VS. PARAMETER SPACE VS. IMAGE SPACE

One approach to advecting texture properties on surfaces is via the use of a parameterization, a topic that has been studied ad nauseam e.g., Levy . According to Stalling , applying LIC to surfaces becomes particularly easy when the whole surface can be parameterized globally in two dimensions, e.g., in the manner of Forssell and Cohen. However, there are drawbacks to this approach. Texture distortions are introduced by the mapping between parameter space and physical space and, more importantly, for a large number of surfaces, no global parameterization is available such as isosurfaces from marching cubes and most unstructured surface meshes resulting from CFD. Surface meshes from CFD may consist of smoothly joined parametric patches, but can have a complex topology and therefore, in general, cannot be parameterized globally. Figures are examples of surfaces for which a global parameterization is not easily derived.

Another approach to advecting texture properties on surfaces would be to immerse the mesh into a 3D texture, then the texture properties could be advected directly according to the 3D vector field. This would have the advantages of simplifying the mapping between texture and physical space and would

Figure : Visualization of flow at the complex surface of a cooling jacket -a composite of over 250,000 polygons.

result in no distortion of the texture. However, this visualization would be limited to the maximum resolution of the 3D texture, thus causing problems with zooming. Also, this approach would not be very efficient in that most of the texels are not used. The amount of texture memory required would also exceed that available on our graphics card, e.g., we would need approximately 500MB of texture memory if we use 4 bytes per texel and a 5123 resolution texture.

Can the problem be reduced to two dimensions? The surface patches can be packed into texture space via a triangle packing algorithm in the manner described by Stalling. However, the packing problem becomes complex since our CFD meshes are composed of many scalene triangles as opposed to the equilateral and isosceles triangles often found in computational geometry. The problem of packing scalene triangles has been studied by Carr. For CFD meshes, triangles generally have very disparate sizes. For a given texture resolution, many triangles would have to be packed that cover less than one texel. To by-pass this, the surfaces could be divided into several patches which could be stored into a texture atlas. In any case, computation time would be spent generating texels which cover polygons hidden from the current point of view. The preceding discussion leads us to an alternative solution that, ideally, has the following characteristics: works in image space, efficiently handles large numbers of surface polygons, spends no extra computation time on occluded polygons, does not spend computation time on polygons covering less than a pixel, and supports user interaction such as zooming, translation, and rotation.

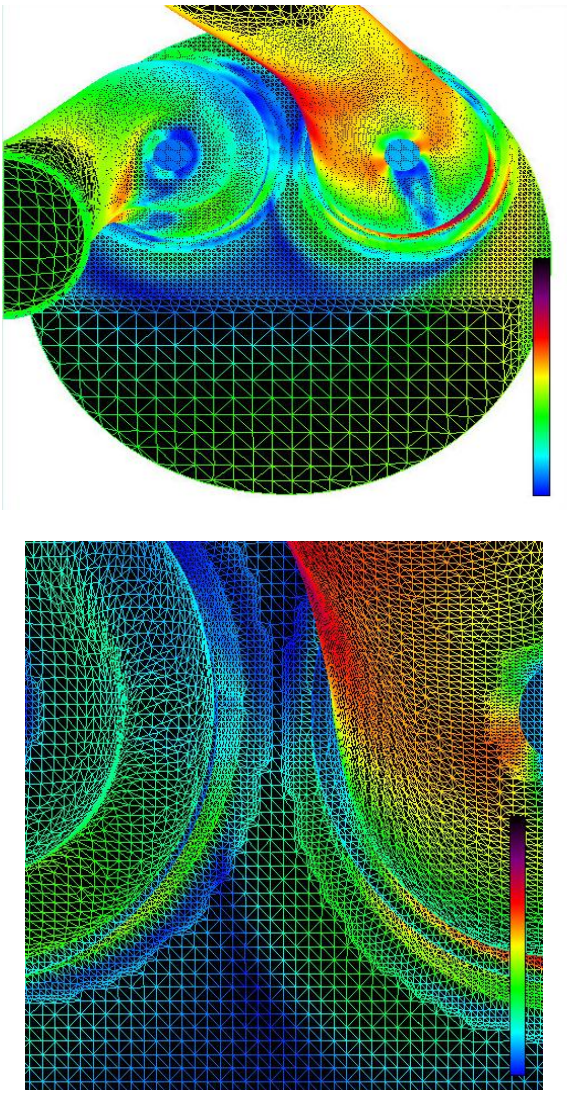


Figure : A wire frame view of the surface of two intake ports showing its 221,000 polygonal composition:(left) an overview from the top, note that many polygons are cover less than one pixel

(right) a close-up view of the mesh between the two intake ports.

METHOD OVERVIEW

The algorithm presented here simplifies the problem by confining the advection of texture properties to image space. We project the surface geometry to image space and then apply a series of textures. This order of operations eliminates portions of the surface hidden from the viewer. In short, our proposed method for visualization of flow on surfaces is comprised of the following procedure:

1. associate the 3D flow data with the polygons at the boundary surface i.e., a velocity vector is stored at each polygon vertex of the surface

2. project the surface and its vector field onto the image plane

3. identify geometric discontinuities

4. advect texture properties according to the vector field in image space

5. inject and blend noise

6. apply additional blending along the geometric discontinuities previously identified

7. overlay all optional visualization cues such as showing a semi-transparent representation of the surface with shading

These stages are depicted schematically in Figure 4.4. Each step of the pipeline is necessary for the dynamic cases of unsteady flow, time-dependent geometry, rotation, translation, and scaling, and only a subset is needed for the static cases involving steady-state flow and no changes to the view-point. We consider each of these stages in more detail in the sections that follow.

VECTOR FIELD PROJECTION

In order to advect texture properties in image space, we must project the vector field associated with the surface to the image plane, taking into account that the velocity vectors are stored at the polygon vertices. We chose to take advantage of the graphics hardware to project the vector field to the image plane. We assign a color whose R, G, and B values encode the x, y, and z components of the local vectors to each vertex of the boundary surface respectively. The velocity-colored geometry is rendered to the frame buffer.

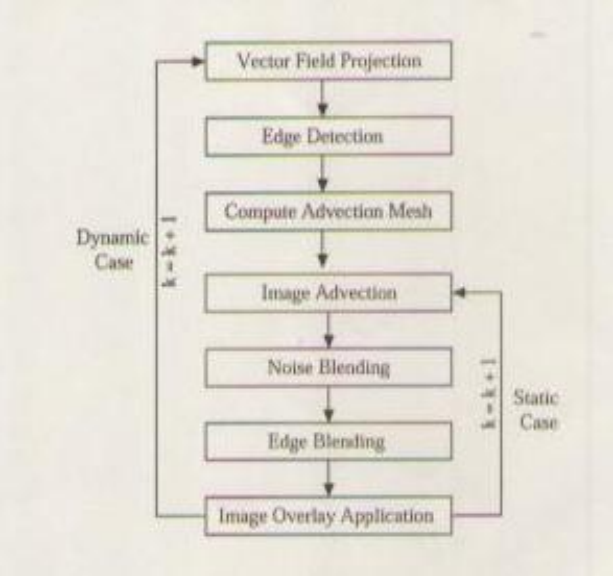


Figure : Flow diagram of texture-based flow visualization on complex surfaces -k represents time as a frame number.

We use the term velocity image to describe the result of encoding the velocity vectors at the mesh vertices into color values. The velocity image is interpreted as the vector field and is used for the texture advection in image space. More precisely, the color assignment can be done with a simple scaling operation. For each color component, hrgb, we assign a velocity, vxyz component according to:

hr = vx - vmin x / vmax x - vmin x

hg = vy - vmin y / vmax y - vmin y

hb = vz - vmin z / vmax z - vmin z

The minimum velocity component is subtracted for each color component respectively, in an effort to minimize loss of accuracy. The use of a velocity image yields the following benefits: the advection computation and noise blending is simpler in image space, thus we inherit advantages from the LEA and IBFV, the vector field and polygon mesh are decoupled, thereby freeing up expensive computation time dedicated to polygons smaller than a single pixel, conceptually, this is performing hardware-accelerated occlusion culling, since all polygons hidden from the viewer, are immediately eliminated from any further processing, and this operation is supported by the graphics hardware. Saving the velocity image to main memory is simple, fast, and easy. A sample velocity image is shown in Figure.

The construction of the velocity image allows us to take advantage of hardware-accelerated flow field reconstruction. During the construction of the velocity

image, we enable Gouraud Shading, also supported by the graphics hardware. Since each velocity component is stored as hue at each polygon vertex of the surface, the smooth interpolation of hue amounts to hardware-accelerated vector field reconstruction. This is important for a minimum of two reasons. First, the polygonal primitive we choose at image advection time is independent of the original mesh polygons more in Section . In other words, the vertices of the mesh we use to distort the image are not the same vertices where the original velocity vectors are stored. Second, interpolation is essential for flow field reconstruction. When the surface is rendered with velocity encoded as hue, the vertices of some polygons are clipped during the projection process. However, we still need to access the vector field values inside those polygons, and not just at the vertices, hence the need for reconstruction. We also note that we are not necessarily limited to linear interpolation for reconstruction.

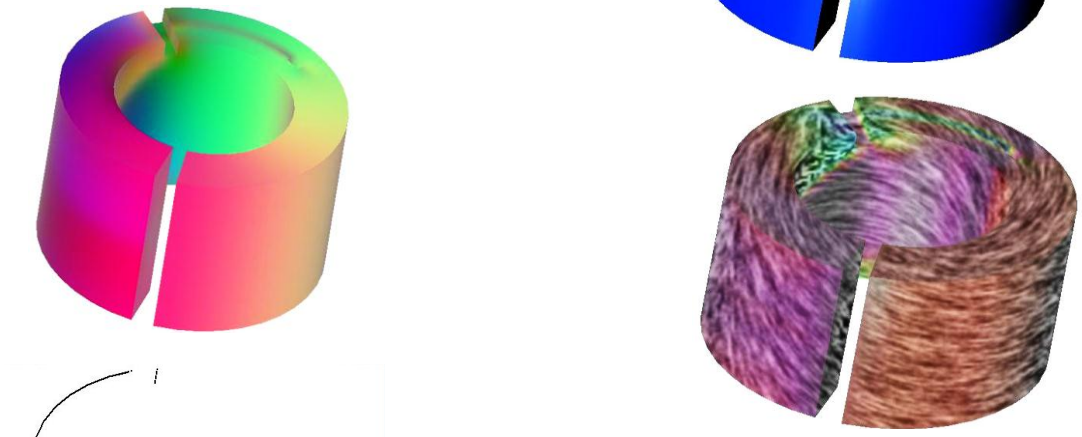


Figure: The 5 component images, plus a 6th composite image, used for the visualization of surface flow on a ring: (top, left) the velocity image, (top, middle) the geometric edge boundaries, (top, right) the advected and blended textures, (bottom, left) a sample noise image, (bottom, middle) an image overlay, (bottom, right) the result of the composited images with an optional velocity color map. The geometric edge boundaries are drawn in black for illustration.

Higher order interpolation schemes can be supported by graphics hardware.

The velocity vectors are de-coded from the velocity image according to:

vx = hr . (vmax x - vmin x) + vmin x

vy = hg . (vmax y - vmin y) + vmin y

vz = hb . (vmax z - vmin z) + vmin z

The de-coded velocity vectors used to compute the advection mesh are then projected onto the

image plane.

The magnitude of the velocity vectors at those parts of the surface orthogonal to the image plane may be shortened as a result of perspective projection, i.e., if the dot product between the image plane normal and the 3D surface normal is zero or close to zero. This can reduce the visual clarity of the vector field's direction during animation. In our implementation, we added an option that allows the user to apply a bias to the velocity vectors that are shortened close to zero due to the projection. We can use this bias to reduce the scaling effect for curved surfaces. Conceptually it is like applying a reverse velocity clamp. The projection of the vectors to the image plane is done after velocity image construction for 2 reasons:

(1) not all of the vectors have to be projected , thus saving computation time

(2) we use the original 3D vectors for the velocity mask.

ADVECTION MESH COMPUTATION AND BOUNDARY TREATMENT

After the projection of the vector field we compute the mesh used to advect the textures similar to IBFV. We distort a regular, rectilinear mesh according to the velocity vectors stored at mesh grid intersections. The distorted mesh vertices can then be computed by advecting each mesh grid intersection according to the discretized Euler approximation of a pathline, p, the same as a streamline for steady flow expressed as:

pk+1 = pk + vp(pk; t) ∆t

where vp represents a magnitude and direction after projection to the image plane. The texture coordinates located at the regular, rectilinear mesh vertices are then mapped to the forward distorted mesh positions. The distorted mesh positions are stored for fast advection of texture properties for static scenes. Special attention must be paid in order to handle flow at geometric boundaries of the surface.Figure shows an overview of the original IBFV process. During the visualization, each frame is advected, rendered, and blended in with a background image. If we look carefully at the distort phase of the algorithm, we notice that there is nothing to stop the image from being advected outside of the physical boundary of the geometry. While this is not a problem when the geometry covers the entire screen, this can lead to artifacts for geometries from CFD, especially in the case of boundaries with a non-zero outbound flow, e.g., flow outlets.

To address this problem we borrow a notion from LEA that treats non-rectangular flow domains, namely, the use of backward coordinate integration. Using backward integration, equation becomes:

Pk-1 = pk - vp(pk-1; t)∆t

In this case the texture coordinates located at the backward distorted mesh positions are mapped to the regular, rectilinear mesh vertices. Backward integration does not allow advection of image properties past the geometric boundaries.

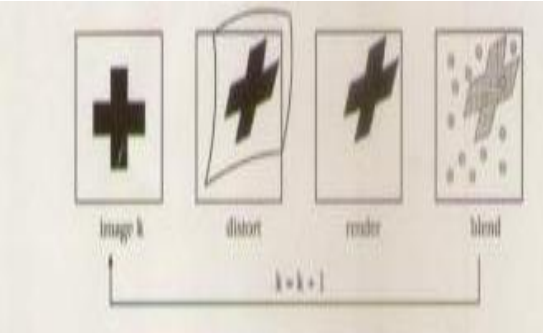


Figure : An overview of the original image based flow visualization

EDGE DETECTION AND BLENDING

While we gain many advantages by decoupling the image advection process with the 3D surface geometry, artifacts can result, especially in the case of geometries with sharp edges. If we look carefully at the result of advecting texture properties in image space, we notice that in some cases a visual flow continuity is introduced where it may be undesirable. A sample case is shown in Figure. A portion of the 3D geometry, shown colored, is much less visible after the projection onto the image plane. If the flow texture properties are advected across this edge in image space, also shown colored, an artificial continuity results. To handle this, we incorporate a geometric edge detection process into the algorithm. During the image integration computation, we compare spatially adjacent depth values during one integration and advection step. We compare the associated depth values, zk-1 and zk in world space of pk- 1 and pk from equation , respectively.

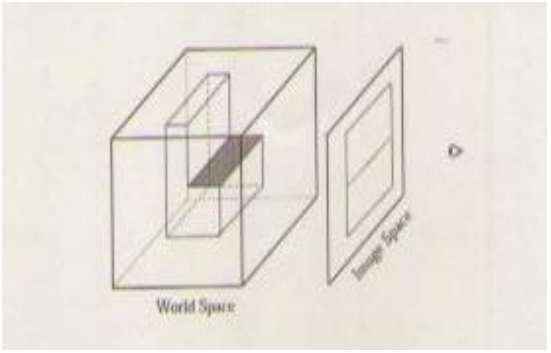


Figure : When a 3D surface geometry (left) is projected, continuity is created in image space (right). If the flow aligned texture properties are advected across this edge, an artificial flow continuity may result.

If **| zk - 1 - zk | > E | pk- 1 – pk |**

where zk is a threshold value, then we identify an edge. All positions, p, for which equation is true, are classified as edge crossing start points. Special treatment must be given when advecting texture properties from these points. This process does not detect all geometric edges, only those edges across which flow texture properties should not be advected.

Figure top, middle shows one set of edges from the detection process. The geometric edges are identified and stored during the dynamic visualization case and additional blending is applied. During the edge blending phase of the algorithm we introduce discontinuities in the texture aligned with the geometric discontinuities from the surface, i.e., gray values are blended in at the edges. This has the effect of adding a gray scale phase shift to the pixel values already blended. This could obviously be handled in different ways, e.g., choosing a random noise value to advect or inverting the noise value already present. Some results of the edge detection and blending phase are illustrated in Figure . In our data sets an " of 1-2% of depth buffer is practical. However, the users may set their own value if fine tuning of the visualization is needed. The same edge detection and blending benefits incoming boundary flow. Also an artifact of the IBFV algorithm, geometric boundaries with incoming flow may appear dimmer than the rest of the geometry.

This is a result of the noise injection and blending process described in Section . In short, the background color shows through more in areas of incoming flow because not as much noise has been blended in these areas. Figure top, shows a 2D slice through a 3D mesh from a CFD simulation with incoming boundary flow coming in through the narrow inlet from the right. Note that the edge of the inlet appears dim. Figure bottom, shows the same slice with edge blending turned on. The boundary artifacts of the noise injection and blending process are no longer a distraction. Edge detection and blending also plays in important role while an object is rotating. Without special treatment, contours in image space become blurred when different portions of a surface geometry overlap, such as when blood vessels in Figure overlap during rotation.

NOISE BLENDING

By reducing the image generation process back to two dimensions, the noise injection and blending phase falls in line with the original IBFV technique, namely, an image, F, is related to a previous image,

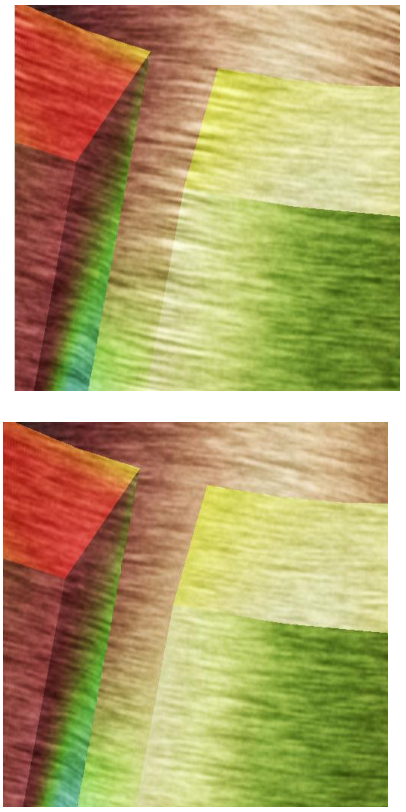


Figure : A close-up example of geometric edge detection: on the left side, geometric edge detection is disabled, on the right side enabled.

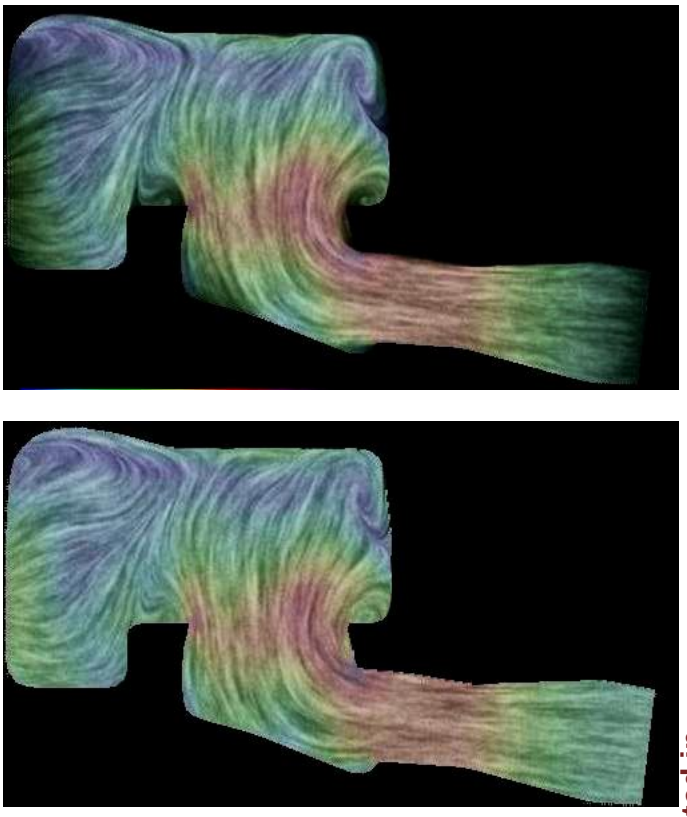


Figure : Here we see a 2D slice through a 3D geometry from a CFD simulation. (top) With no edge blending, the background color shows through boundary areas with incoming flow.

(bottom). With edge blending, these artifacts are no longer a distraction.

G, by [164]:

$$F(p; k) = \infty \sum k\text{-}1\ Xi = 0\ (1 - \infty)^t\ G(pk - i)$$

where **p** represents a pathline, ∞ defines a blending coefficient, and k represents time as a frame number. Thus a point, pk, of an image Fk, is the result of a convolution of a series of previous images, G(x; i), along the pathline through pk, with an decay filter. The blended noise images have both spatial and temporal characteristics. In the spatial domain, a single noise image, g(x), is described as a linearly interpolated sequence of n random values, Gi, in the range ,

$$g(x) = \sum hs(x - is)Gi \bmod n$$

where the spacing, s, between noise samples is generally greater than or equal to the distance traversed by an image property in one advection step and hs represents a triangular black and white pulse function. Here x represents a location in the flow domain. In practice, we give the user control of s, resulting in multi-frequency texture resolutions in the spacial domain. The background textures used for blending also vary in time. In the temporal domain, each point, Gi in the background texture, periodically increases and decays according to a profile, w(t), e.g.,

$$Gi;k = w((k/M + \varphi i) \bmod 1)$$

where **φi** represents a random phase, drawn from the interval [0,1],M is the total number of background noise images used, and where w(t) is defined for all time steps. We use a square wave profile, i.e., w(t) = 1 if t < 1=2 and 0 otherwise. In our application, the user has the option of varying M. Smaller values ofM result in higher frequency noise in the temporal domain whereas higher values M result in a lower temporal frequency. Figure shows a sample blended image and Figure shows a sample noise image.

IMAGE OVERLAY APPLICATION

The rendering of the advected image and the noise blending may be followed by an optional image overlay. An overlay enhances the resulting texture-based representation of surface flow by applying color, shading, or any attribute mapped to color. In implementation, we generate the image overlay following the construction of the velocity image. This overlay may render any CFD simulation attribute mapped to hue. The overlay is constructed once for each static scene and applied after the image advection, edge blending, and noise blending phases. Since the image advection exploits frame-to-frame coherency, the overlay must be applied after the advection in order to prevent the surface itself from being smeared. Also worthy of mention, is that the opacity value of the image overlay is a free parameter we provide to the user.

IMPLEMENTATION

In this section we consider some aspects of the algorithm not previously discussed which are important for implementation. Our implementation is based on the highly portable OpenGL library.

TEXTURE CLIPPING

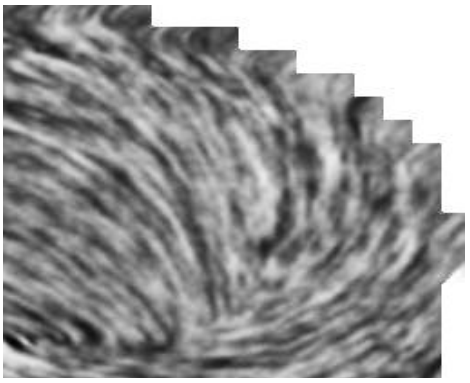
In our application, the resolution of the quadrilateral mesh used to warp the image can be specified by the user. The user may specify a coarse resolution mesh, e.g., 128 x 128, for faster performance or a fine resolution mesh, e.g., 512 x 512, for higher accuracy. However, if the resolution of the advection mesh is too coarse in image space, artifacts begin to appear. Figure illustrates these artifacts zoomed in on the edge of a surface. In order to minimize the jagged edges created by coarse resolution texture quadrilaterals, we apply a texture clipping function. Subsets of textured quadrilateral that do not cover the surface are clipped from the visualization as shown in Figure. This can be implemented simply with the image overlay by maximizing the opacity wherever the depth buffer value is maximized, i.e., wherever there is a great depth.

VELOCITY MASK

In order to dim high frequency noise in low velocity regions, the user also has the option of applying a velocity mask. We adopt the velocity mask of Jobard et al. [63] for our purposes here, namely:

$$\infty = 1 - (1 - v)^m$$

where ∞ decreases as a function of velocity magnitude. In our case, the image overlay becomes more opaque in regions of low velocity and more transparent in areas of high velocity. With the velocity mask



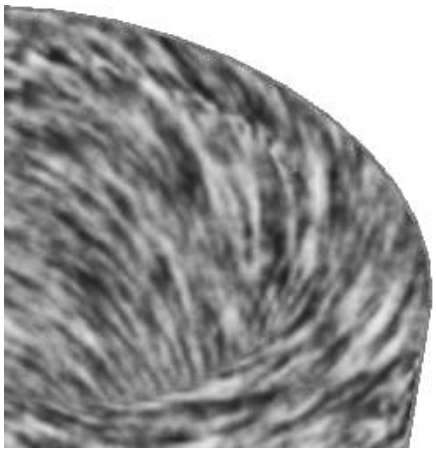


Figure :The result of, left, a coarse resolution advection mesh with artifacts and, right, the application of texture clipping. The resolution of the advection mesh shown on the left is 32 x 32 for illustration.

enabled, the viewer's attention is drawn away from areas of stagnant flow, and towards areas of high flow velocity. We note that in the context of CFD simulation data, engineers are often very concerned about areas of stagnant flow. In the case of a cooling jacket, stagnant flow may represent a region of the geometry where the temperature is too high, possibly leading to boiling conditions thus reducing the effectiveness of the cooling jacket itself. Therefore, in our case the engineers may disable the velocity mask or use the velocity mask to highlight areas of flow, e.g., make the hue brighter in areas of low velocity.

PERFORMANCE AND RESULTS

Our visualization technique is applied primarily to large, highly irregular, adaptive resolution meshes commonly resulting from computational fluid dynamics simulations.The ideal intake manifold supplies an equal amount of fluid flow to each piston valve. Visualizing the flow at the surface gives the engineer insight into any imbalances between the inlet pipes, in this case, the long narrow pipes of the geometry. Figure shows our method applied to a surface of an intake port mesh composed of 221K polygons. The intake port mesh is composed of highly adaptive resolution surface polygons and for which no global parameterization is readily available. The method described here allows the user to zoom in at arbitrary view points always maintaining a high spatial resolution visualization. The algorithm applies equally well to meshes with time-dependent geometry and topology. Figure shows the surface of a piston cylinder with the piston head defining the bottom of the surface. The method here enables the visualization of fuel intake as the piston head slides down the cylinder. The resulting flow visualization has a smooth spatio-temporal coherency. Our algorithm also has applications in the field of medicine. Figure shows the circulation of blood at the junction of blood vessels. An abnormal cavity has developed that may hinder the optimal distribution of blood.