ISA and IBFVS: Image Space Based Visualization of Flow on Surfaces

Robert S. Laramee, Jarke J. van Wijk, Bruno Jobard, and Helwig Hauser

***Abstract*— We present a side-by-side analysis of two recent image space approaches for the visualization of vector fields on surfaces. The two methods, Image Space Advection (ISA) and Image Based Flow Visualization for Curved Surfaces (IBFVS) generate dense representations of time-dependent vector fields with high spatio-temporal correlation. While the 3D vector fields are associated with arbitrary surfaces represented by triangular meshes, the generation and advection of texture properties is confined to image space. Fast frame rates are achieved by exploiting frame-to-frame coherency and graphics hardware. In our comparison of ISA and IBFVS we point out the strengths and weaknesses of each approach and give recommendations as to when and where they are best applied.**

***Index Terms*— Unsteady flow visualization, computational fluid dynamics (CFD), surface representation, surface rendering, texture mapping**

I. Introduction

UNTIL recently, 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 Spot Noise [27] and LIC [3] in the early 1990s. The main advantage of this class of algorithms is their *complete* depiction of the flow field while their drawbacks are, in general, the computational time required to generate the results, lack of flow orientation (upstream vs. downstream), and applicability to 3D flow. Two new algorithms, namely, those introduced by Laramee et al. [13] (named here as ISA, Image Space Advection) and IBFVS (Image Based Flow Visualization for Curved Surfaces) by Van Wijk [29] have recently been introduced and overcome the computation time hurdle by generating a dense representation of flow on surfaces at fast frame rates, even for unsteady flow. ISA and IBFVS generate dense representations of fluid flow on complex surfaces without relying on a parameterization, e.g., Figure 1 top.

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 [26]. Here, we alter this classic order of operations. First we project the surface geometry and its associated vector field to image space and then apply texturing. In other words, while conceptually texture properties are advected on boundary surfaces in 3D, in fact the algorithms realize texture advection in image space. In particular, the methods have the following characteristics: They (1) generate a dense representation of unsteady flow on surfaces (see Figure 1, bottom), (2) visualize flow on complex surfaces composed of polygons whose number is on the order of 250,000 or more, (3) can handle arbitrary, complex meshes without relying on a parametrization, (4) support user-interaction such as rotation, translation, and zooming while

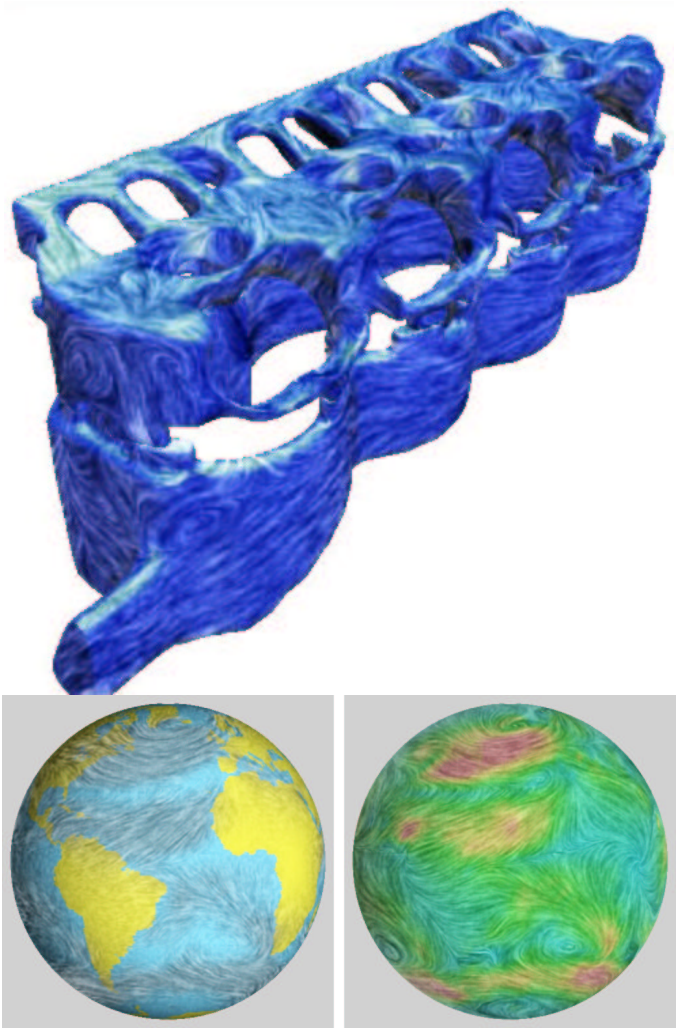

Fig. 1. (top) Visualization of flow at the complex surface of a cooling jacket–a composite of over 228,000 polygons. (bottom) The wind stress over the month of January, averaged over more than 100 years, visualized at the earth's surface.

Manuscript received February 26, 2004

Robert S. Laramee and Helwig Hauser are with the VRVis Research Center in Vienna, Austria, E-Mail: {Laramee,Hauser}@VRVis.at. Jarke J. van Wijk is with the Technische Universiteit Eindhoven, the Netherlands, E-mail: VanWijk@win.tue.nl. Bruno Jobard is with University of Pau, France, E-Mail: bjobard@univ-pau.fr.

maintaining a constant, high spatial resolution, (5) deliver high performance, i.e., several frames per second, and (6) visualize flow on dynamic meshes with time-dependent geometry and topology.

We present a framework that both unifies and compares ISA and IBFVS. We identify where the two approaches overlap as well as where they separate and discuss the resulting consequences. We also offer our perspectives on when ISA and IBFVS are best applied and why. The rest of the paper is organized as follows: In Section III we provide a joint model for ISA and IBFVS. Section IV presents a side-by-side comparison of the algorithms and the resulting overlapping as well as divergent components of their implementations. Section V provides applications of the techniques while Sections V-C and VI compare the relative performance of ISA and IBFVS and give some guidelines with respect to when to apply the algorithms. Finally, Section VII draws some conclusions and outlines future work.

II. Related Work

Our work focuses on texture-based representations of unsteady flow on complex, non-parameterized surfaces. The challenge of visualizing time-dependent vector fields on surfaces at fast frame rates remains unsolved in surveys of the research literature [22], [23], [26]. However, several techniques have been proposed to successfully resolve parts of the problem. In the next two sections we describe the two main categories of approaches for dense representations on surfaces and dense representations of unsteady 2D vector fields.

A. Texture-Based Flow Visualization on Surfaces: Object Space Approaches

Previous research with a focus on representations of the vector field on boundary surfaces is generally restricted to steady-state flow. This is mainly due to the prohibitive computational time required. These techniques generally use an object space approach of generating a dense representation of the flow.

Some approaches are limited to curvilinear surfaces, i.e., surfaces that can be parameterized using 2D coordinates. The original Spot Noise paper [27] showed how flow aligned texture can be generated on parametric surfaces. Forssell and Cohen [6] extended LIC to curvilinear surfaces with animation techniques and added magnitude and directional information. Battke et al. [1] described an extension of LIC for arbitrary surfaces in 3D. Mao et al. [18] presented an algorithm for convolving solid white noise on triangle meshes in 3D space and extended LIC for visualizing a vector field on arbitrary surfaces in 3D. Stalling provided a helpful overview of LIC techniques applied to surfaces [26] in 1997. In particular, a useful comparison of parameterized vs. non-parameterized surfaces is given.

B. 2D, Unsteady Flow

Much work has been done in order to speed up texture-based flow visualization in 2D. Cabral and Leedom [2] present a parallel processing implementation of LIC. Max and Becker were early to introduce the idea of moving textures in order to visualize vector fields [19]. Heidrich et al. [8] exploit pixel textures to accelerate LIC computation.

Jobard et al. introduced a Lagrangian-Eulerian texture advection technique for 2D vector fields at interactive frame rates [11], [12]. The algorithm produces animations with high spatio-temporal correlation. Each still frame depicts the instantaneous structure of the flow, whereas an animated sequence of frames reveals the motion of a dense collection of particles as if released into the flow. Particle paths are integrated as a function of time, referred to as the Lagrangian step, while the color distribution of the image pixels is updated in place (Eulerian step).

Image Based Flow Visualization (IBFV) by Van Wijk [28] is one of the most recent algorithms for synthesizing dense, 2D, unsteady vector field representations. It is based on the advection and decay of textures in 2D. Each frame of the visualization is defined as a blend between the previous image, distorted according to the flow direction, and a number of background images composed of filtered white noise textures. Fast performance times are achieved through effective use of the graphics hardware.

III. Method Background

Before we describe visualization on surfaces, we present the framework upon which ISA and IBFVS are built, with emphasis on the synthesis of dense textures in two dimensions. More details are given by Van Wijk [28].

A. Texture Synthesis in 2D

Suppose we have an unsteady, two-dimensional vector field $\mathbf{v}(\mathbf{x};t) \in \mathcal{R}^2$ with

$$\mathbf{v}(\mathbf{x};t) = [v_x(x,y;t), v_y(x,y;t)] \tag{1}$$

defined for $t \geq 0$, $\mathbf{x} \in \Omega$, and $\Omega \subset \mathcal{R}^2$. A pathline is the path of a massless particle, advected by the flow over time. Such a pathline, $\mathbf{p}(t)$, is the solution of the differential equation

$$d\mathbf{p}(t)/dt = \mathbf{v}(\mathbf{p}(t);t) \tag{2}$$

for a given start position $\mathbf{p}(0)$. A first-order Euler approximation of $\mathbf{p}(t)$ is

$$\mathbf{p}_k = \mathbf{p}_{k-1} + \mathbf{v}(\mathbf{p}_{k-1};t)\Delta t \tag{3}$$

with $k \in \mathcal{N}$ and $t = k\Delta t$. We use the frame number k to denote time. Consider a field $F(\mathbf{x};k)$ that represents some property advected by the flow. Here F represents an image, hence $F(\mathbf{x};k)$ is typically an RGB-tuple. The property is advected just like a particle, so a first-order approximation of the transport of F can be given by:

$$F(\mathbf{p}_k;k) = \begin{cases} F(\mathbf{p}_{k-1};k-1) \text{ if } \mathbf{p}_{k-1} \in \Omega \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

Eventually, some or even all of $F(\mathbf{x};k)$ will likely disappear since it may eventually be advected beyond the spatial boundaries of the domain. Hence, at each time step a combination of F and another image G is taken:

$$F(\mathbf{p}_k;k) = (1-\alpha)F(\mathbf{p}_{k-1};k-1) + \alpha G(\mathbf{p}_k;k) \tag{5}$$

where the points $\mathbf{p}_k$ are defined by equation (3), and where $\alpha = \alpha(\mathbf{x}; k) \in [0, 1]$ defines a blending mask. A typical value for α is 10–20% of the total opacity. Equation 5 defines the image generation process. By varying G many different visualization methods can be emulated. To produce dense textures an interpolated grid with random values is used, i.e.,

$$G(\mathbf{x}; k) = \sum_{i,j} h(\frac{x}{s} - i)h(\frac{y}{s} - j)G_{i,j;k} \tag{6}$$

where s is a parameter that controls the spatial frequency of the texture and where $h(x)$ is a triangular pulse $h(x) = max(0, 1 - |x|)$. A typical value for the texture scale is anywhere from 1–10 pixels in the spatial domain. The instantaneous grid values $G_{i,j;k}$ are defined by

$$G_{i,j;k} = w(v_g k + \phi_{i,j}) \tag{7}$$

To each grid point a random phase $\phi_{i,j}$ is assigned. The value of $G_{i,j;k}$ cycles according to some periodic function $w(t)$, for instance a square wave or a saw tooth. The term v_g denotes the rate of image change.

The preceding equations can be mapped directly to standard graphics operations, leading to a fast algorithm. The flow field is represented by a rectangular or triangular mesh. First, the mesh is distorted and rendered, using the preceding image as a texture map. Second, fresh ink (or noise) is added by blending in a polygon, texture mapped with a scaled, interpolated, precomputed pattern. Third, the image is stored in texture memory for the next round. Finally additional graphics are rendered and the resulting image is shown to the user.

B. Texture Synthesis for Surfaces

Next we consider the dense visualization of flow on surfaces: ISA and IBFVS. Both techniques use a triangular mesh as a geometric representation for boundary surfaces. Suppose that for each vertex i its position $\mathbf{r}_i = (r_{ix}, r_{iy}, r_{iz})$, a normal vector $\mathbf{n}_i = (n_{ix}, n_{iy}, n_{iz})$, and a velocity vector $\mathbf{v}_i = (v_{ix}, v_{iy}, v_{iz})$ is given. Typically, this velocity is a sample of a 3D flow field. For simplicity we assume that this velocity is confined to the surface (possibly by projection), i.e., $\mathbf{v}_i \cdot \mathbf{n}_i = 0$. The more general case is discussed by Laramee et al. [14] and Van Wijk [29].

Given a model, viewing, and perspective transformation, the mesh can be projected on the screen. We denote this by $\mathbf{x}' = T(\mathbf{x})$, where $\mathbf{x}'$ is a 2D point on the screen and where T denotes the perspective transformation. We use apostrophes to denote projected quantities throughout the paper. One key to ISA and IBFVS is the simple observation that when some property is defined and advected on a surface in 3D space, equation 4 also holds after projection of this surface to the image plane, i.e.,

$$F(\mathbf{p}'_k; k) = \begin{cases} F(\mathbf{p}'_{k-1}; k-1) \text{ if } \mathbf{p}'_{k-1} \in \Omega' \\ 0 \text{ otherwise} \end{cases} \tag{8}$$

where Ω' is the projection of the surface onto the image F, and where we assume that pathline $\mathbf{p}$ is visible. We use this to define a process for the synthesis of flow texture on surfaces. Two further aspects must be addressed. Firstly, fresh ink should be added only to the projection of the surface and not to the background. Secondly, a shaded image $F_s(., k)$ has to be blended in. For this, we use a separate texture image $F_t(., k)$, which is defined on exactly the same space as the image F itself, i.e., texture space coincides with image space. The process can now be defined as follows:

$$F_t(\mathbf{p}'_k; k) = (1 - \gamma(\mathbf{p}'_k))F_t(\mathbf{p}'_{k-1}; k-1) + \gamma(\mathbf{p}'_k)G(\mathbf{p}'_k; k) \tag{9}$$

$$F(\mathbf{x}'; k) = \beta F_t(\mathbf{x}'; k) + (1 - \beta)F_s(\mathbf{x}'; k) \tag{10}$$

where β denotes the strength of the texture with respect to the shading, and where γ is introduced to constrain the fresh ink G to the projected surface. Its value is defined by

$$\gamma(\mathbf{x}') = \begin{cases} \alpha(\mathbf{x}') \text{ if } \mathbf{x}' \in \Omega' \\ 0 \text{ otherwise} \end{cases} \tag{11}$$

where α is defined above.

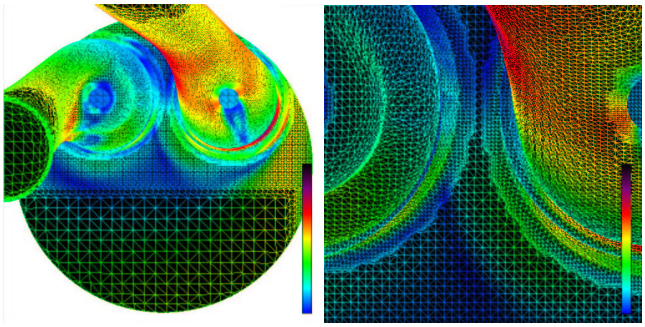

Fig. 2. A wire frame view of the surface of two intake ports showing its 221,000 polygonal composition: (left) an overview from the top, note that many polygons are cover less than one pixel (right) a close-up view of the mesh between the two intake ports.

IV. ISA AND IBFVS

In this section we describe ISA and IBFVS in detail, starting with a discussion of why we chose an image space approach.

A. Parameter Space vs. Object 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 in depth, e.g., by Gorla et al. [7] or Levy et al. [15]. According to Stalling [26], 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 [5], [6]. 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 [16] and most unstructured surface meshes used for CFD simulations. Figures 1 top, 2 and 12 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 [24]. This would have the advantage of

simplifying the mapping between texture and physical space and would 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. Finally, the amount of texture memory required would exceed that available on many graphics cards.

Alternatively, the surface patches can be packed into texture space via a triangle packing algorithm in the manner described by Stalling [26] or Carr et al. [4]. However, the packing problem becomes complex since CFD meshes are usually composed of many scalene triangles as opposed to the equilateral and isosceles triangles often found in computational geometry. For CFD meshes, triangles generally have very disparate sizes. Many triangles would have to be packed that cover less than one texel (cf. Figure 2). In addition, the problem of advecting texture properties across triangle edges would have to be addressed. To by-pass this, the surfaces could be divided into patches which could be stored into a texture atlas [15]. However, much computation time would be spent generating texels which cover polygons hidden from the current point of view because packing does not consider the current viewing projection.

B. Side-by-Side Overview of Both Methods

ISA and IBFVS simplify the problem by confining the synthesis and advection of texture properties to image space. Both methods project the surface mesh and associated vector data to image space and then apply a series of textures. Summarizing, the methods are comprised of the following steps:

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 mesh and its vector field onto the image plane
3) advect texture properties according to the vector field in image space
4) inject and blend noise
5) overlay optional visualization cues such as showing a semi-transparent representation of the surface with shading

These stages are depicted schematically for ISA and IBFVS in Figure 3. 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 change to the viewing parameters.

The ISA and IBFVS implementation pipelines are shown side-by-side in Figure 3 in order to illustrate overlap and divergence. Conceptually, the algorithms share several overlapping components such as vector field projection, advection mesh computation, texture-mapping, noise injection and blending, and the addition of shading or a color map. The main difference between the two methods is that ISA uses an image-based mesh in order to advect the textures, whereas the texture advection in IBFVS is driven by the original 3D mesh. As a result, differences arise stemming from parts of the algorithm that use image space vs. object space. These decisions result in advantages and disadvantages for both methods. We examine each of these stages in more detail in the sections that follow.

C. Vector Field Projection, Image vs. Object Space

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. Projecting the vector field to image space makes the advection computation and noise blending simpler, thus ISA and IBFVS inherit advantages from the original LEA and IBFV, e.g., simple noise blending and fast frame rates.

1) ISA: Image Space Vector Field Projection: ISA uses an image space approach that takes advantage of the graphics hardware to project the vector field to the image plane. A color whose R, G, and B values encode the x, y, and z components of the local vectors is assigned to each vertex of the boundary surface respectively. The velocity-colored geometry is rendered to the framebuffer using Gouraud shading in order to fill the projected triangles with interpolated velocity values. We use the term *velocity image* (Figure 4 top, left) to describe the result of rendering the mesh, with colors encoding velocities. The velocity image is interpreted as the vector field and is used for the texture advection in image space.

The de-coded velocity vectors used to compute the advection mesh (Section IV-D) are then projected onto the image plane. In the ISA implementation, the projection of the vectors to the image plane is done after velocity image construction for two reasons: (1) not *all* of the vectors have to be projected (Sec. IV-D), thus saving computation time and (2) ISA may use the vectors from 3D world space in order to construct a velocity mask [13].

The ISA approach yields the following benefits: (1) the vector field and polygon mesh are decoupled, thereby freeing up expensive computation time dedicated to polygons smaller than a single pixel and (2) conceptually, this is performing hardware-accelerated occlusion culling, since all polygons hidden from the viewer are eliminated from any further processing. Saving the velocity image to main memory is simple, fast, and easy. On the graphics card we tested (Section V-C), reading the framebuffer required less than one millisecond. We note that a full-resolution read-back of the framebuffer is not a fundamental requirement of ISA, only of its current implementation. A sample velocity image is shown in Figure 4 (top, left).

The use of a velocity mask results in quantization of the velocity field. However, our experience shows that we are unable to see quantization effects resulting from the use of a velocity image in the ISA implementation. ISA has been implemented in the framework of a commercial software and has been tested on a wide variety of data sets with velocity magnitudes varying by up to three orders of magnitude.

2) IBFVS: Object Space Vector Field Projection: In IBFVS the velocity vectors stored at the mesh vertices are projected to image space on a per-mesh-vertex basis by the CPU, making the projection process closer to an object-space approach. That is, for each velocity vector, its projected quantity is computed

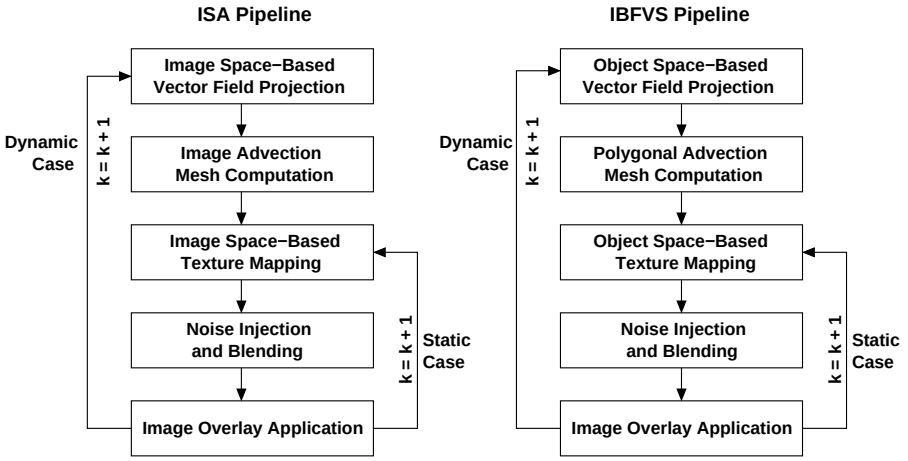

Fig. 3. Flow diagrams of the two texture-based flow visualization algorithms, side-by-side. On the left is the ISA pipeline, on the right is IBFVS. The edge detection process of the original ISA algorithm [13] is not included in order to highlight the major differences between the algorithms.

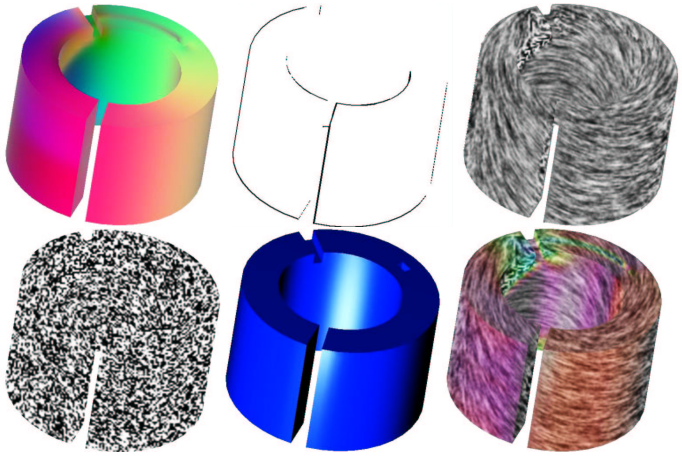

Fig. 4. The five component images, plus a sixth composite image, used for the visualization of surface flow: (top, left) the velocity image (ISA only), (top, middle) the geometric edge boundaries drawn in black for illustration (ISA only), (top, right) advected and blended textures, (bottom, left) a sample noise image, (bottom, middle) an image overlay, (bottom, right) the result of the composited images with an optional color map.

in software. This has the advantage of simplicity, plus no resampling and interpolation of the vector field is performed.

Additionally, IBFVS avoids some of the disadvantages of the ISA approach. Namely, (1) no separate velocity image has to be constructed, and (2) it avoids read-back of the framebuffer, an operation that can be time consuming. One disadvantage of the IBFVS approach appears in the case of very dense meshes. In this case, many velocity vectors may be projected onto the same pixel in image space, thus creating redundant computation. Also, velocity vectors occluded from the viewer are also projected onto the image plane.

D. Image vs. Polygonal Texture Advection

After the projection of the vector field, both ISA and IBFVS compute the texture coordinates used to advect the textures in image space. Both ISA and IBFVS use backward coordinate integration (introduced by Max and Becker [20]):

$$\mathbf{p}_{k-1} = \mathbf{p}_k - \mathbf{v}'(\mathbf{p}_k; t)\,\Delta t \tag{12}$$

where $\mathbf{v}'$ represents a magnitude and direction after projection to the image plane. In other words, the texture is advected over the mesh, instead of moving the mesh with the texture attached to it. This approach has advantages because texture properties are not pushed outside the geometry and because the mesh remains static.

1) ISA: Image Mesh Texture Advection: The meshes used to compute the advected texture coordinates in image space are different for ISA and IBFVS. This is a central difference between the two approaches.

ISA distorts a regular, rectilinear mesh defined in 2D image space, that conceptually overlaps the velocity image. The velocities at the mesh vertices are found by a look up in the velocity image followed by a projection to the image plane. In this case the texture coordinates located at the backward distorted mesh positions are mapped to the mesh vertices. We can summarize this as an *image mesh texture advection* approach. The resulting texture-mapping takes place in image space. Some advantages of using an image mesh texture advection approach are: (1) no computation time is spent on mesh polygons covering an area of less than one pixel, (2) no computation time is spent computing texture coordinates for occluded polygons, and (3) the resolution of the texture advection remains constant during user zooming. This is important because as the user zooms towards an object, the ISA approach implicitly samples the vector field at a higher spatial frequency in object space.

2) IBFVS: Polygonal Mesh Texture Advection: IBFVS starts with the original surface mesh, projects each vertex to image space, and then warps the texture according to the velocity vectors stored at the projected mesh vertices. In this case the texture coordinates located at the backward distorted mesh positions are mapped to the projected mesh vertices. These vertices are derived from the mesh in object space, projected to the image plane. We can summarize this as a *polygonal mesh texture advection* approach. The resulting texture-mapping returns back to object space. The distorted texture coordinates are mapped to the original 3D surface mesh vertices.

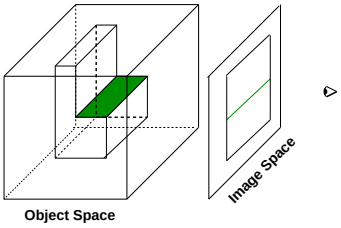

Fig. 5. A 3D surface geometry (left) is projected, to image space (right). If flow aligned texture properties are advected across the colored edge, an artificial flow continuity results.

One advantage of the polygonal mesh texture advection approach is that the density of the polygonal mesh used to advect the texture is the same as the density of the original mesh. Hence, in areas with more detail, more polygons are used. Also, the performance time of IBFVS can be accelerated by discarding back-facing polygons or polygons far outside the clipping frustum.

For both ISA and IBFVS, when the view is changed or when the object is moved, the projection of the patterns is unaltered in viewing space, hence the viewer can briefly observe that the texture patterns are not fixed to the surface, but reside in image space. For animated textures the effect is less strong however. The viewer tends to follow the moving texture properties instead of fixing the view to a point in screen space.

Measures can be taken to decrease this effect that stems from the textures being visually disconnected from the 3D geometry when changing the viewing direction. First, when the user changes the view, the projection of the texture can be changed as well. The original IBFVS implementation provides the option that the textures in image space are translated according to the motion of the mouse pointer. This improves the perceived imagery significantly for translation, reasonably for rotation, but not for scaling. Such an option can also be added to an ISA implementation.

Another approach is to stop the update of the texture image. If the calculation of new texture coordinates and animation are skipped, the IBFVS algorithm reduces to standard texture mapping, using the last generated noise pattern as a fixed texture map. When the user stops the manipulation, texture coordinate computation and texture advection animation are enabled again. The last pattern is used as a start for the new animation. This method works well for scaling and dragging, but is less effective for rotation because sometimes a good texture mapping has not been generated for occluded portions of the surface.

For both ISA and IBFVS, when the view is changed, it takes a short time before the image or animation stabilizes again. With a frame rate of 50 FPS, the image stabilizes in about one half of a second.

E. Edge Detection and Blending

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 is not desired. A sample case is shown in Figure 5. 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.

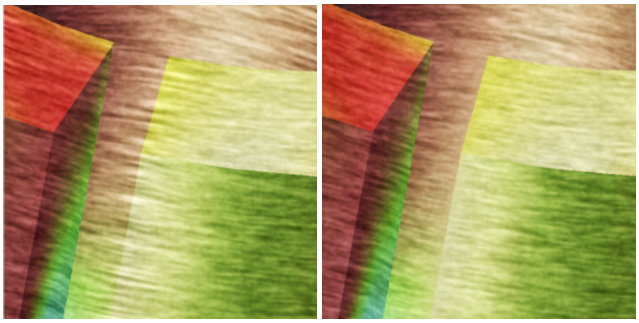

Fig. 6. A close-up example of ISA geometric edge detection: on the left side, geometric edge detection is disabled, on the right side enabled.

1) ISA: Image Space Edge Detection and Blending: To handle this, ISA incorporates a geometric edge detection process that, although using depth information from object space, can be implemented in image space. During the image integration computation, ISA compares spatially adjacent depth values during one integration and advection step. ISA compares the associated depth values, z_{k-1} and z_k in object space of $\mathbf{p}_{k-1}$ and $\mathbf{p}_k$ from equation (12), respectively. If

$$|z_{k-1} - z_k| > \varepsilon \cdot |\mathbf{p}_{k-1} - \mathbf{p}_k| \tag{13}$$

where ε is a threshold value, then ISA identifies an edge. All positions, $\mathbf{p}_k$, for which equation (13) is true, are classified as edge advection point. 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 4 (top, middle) shows one resulting set of edges from the ISA detection process. We term this result an *edge mask*. The edge mask is created and stored during the dynamic visualization case and additional blending is applied during the static case. During the edge blending phase of the algorithm, ISA introduces 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.

Some results of the ISA edge detection and blending phase are illustrated in Figure 6. In our data sets an ε of 1–2% of depth buffer is practical when comparing depth values from Equation (13). The ISA edge mask improves the visualization result of texture advection on surfaces. Plus, the image space approach takes advantage of information already provided by the graphics hardware. However, one disadvantage of the ISA implementation is that it requires read-back of the depth buffer.

2) IBFVS: Object Space Edge Detection and Blending: Originally IBFVS did not include an edge detection and blending process (as indicated in Figure 3, right). However, we have implemented and experimented with an object space approach that can be incorporated into IBFVS.

Creating an edge mask for IBFVS can be done as follows. Firstly, silhouette edges are identified. These are either boundary edges or edges between a backward and forward facing

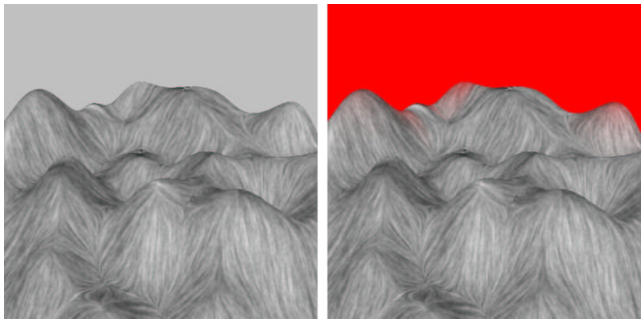

Fig. 7. Boundary inflow from background. With the red background, artifacts show up more clearly.

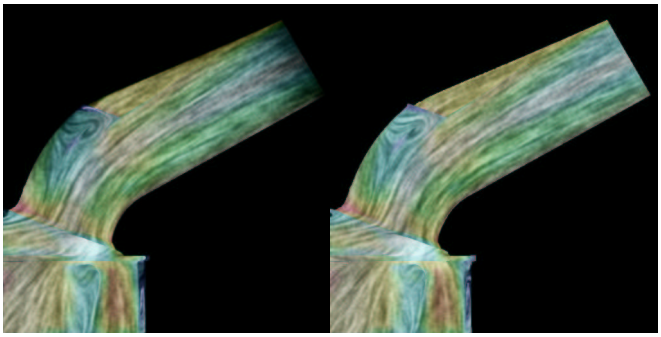

Fig. 8. At the top we see the inlet of an intake port from a CFD simulation. On the left, with no edge blending, the background color shows through boundary areas with incoming flow. On the right, with ISA edge blending, these artifacts are no longer a distraction. Also, the edges are crisper.

polygon. Next, background-colored lines with high opacity are rendered on top of the synthesized texture. As a result, the texture is dimmed and the artificial continuity is diminished.

One advantage of this approach is that it avoids read-back of the depth buffer. One disadvantage of this approach is that, since it is an object space approach, it may have a negative impact on overall performance time in the case of large surface meshes.

F. Boundaries and Silhouettes

Another problem with performing the texture synthesis in image space relates to boundary artifacts in regions of incoming flow. In areas where no texture is present, the background color may blend in. This is illustrated in Figure 7, right. Without special treatment, 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 IV-G. In short, the background color shows through more in areas of incoming flow because not as much noise has been blended in these areas.

1) ISA Boundary Treatment: The same edge detection and blending benefits described in Section IV-E.1 also benefit the treatment of incoming boundary flow. Figure 8, left, shows a surface mesh from a CFD simulation with incoming boundary flow coming in through inlet from the top, right. Note that the edge of the inlet appears dim. Figure 8, right, shows the same inlet with ISA 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 an important role while an object is rotating. Without special treatment, contours in image space become blurred when different portions of a surface geometry overlap.

2) IBFVS Boundary Treatment: IBFVS uses a gray background color by default instead of black (or red). As this color is close to the average value of the texture, artifacts are less visible. This is illustrated in Figure 7, left. Arguably, boundary artifacts here are not as disturbing relative to the original IBFV. Firstly, the contrast of the texture is less than with the original IBFV and hence errors show up less clearly. Secondly, near boundaries and silhouette edges, high gradients in the shading are common. These edges draw the attention of the viewer. Thirdly, the edge mask described in Section IV-E.2 can be added to reduce these artifacts.

G. Noise Injection and Blending

By reducing the image generation process back to two dimensions, the noise injection and blending phase falls in line with the original IBFV. The process is purely image based for both ISA and IBFVS. Namely, an image, F, is related to a previous image, G, by [28]

$$F(\mathbf{p}_k; k) = \alpha \sum_{i=0}^{k-1} (1-\alpha)^i G(\mathbf{p}_{k-i}; k-i) \qquad (14)$$

where $\mathbf{p}_k$ represents a pathline and α defines a blending coefficient. Thus a point, $\mathbf{p}_k$, of an image F_k, is the result of a convolution of a series of previous images, $G(\mathbf{x}; i)$, along the pathline through $\mathbf{p}_k$, with a decay filter defined by $\alpha(1-\alpha)^i$.

Figure 4 (top, right) shows a sample blended image (F) and Figure 4 (bottom, left) shows a sample noise image (G). More details about the noise injection process can be found in previous work [13], [28].

H. Image Overlay Application

For both ISA and IBFVS the rendering of the advected image and the noise blending may be followed by an image overlay. An overlay enhances the resulting texture-based representation of the surface flow by applying color, shading, or any attribute mapped to color (Fig. 4, bottom, right). In implementation, ISA and IBFVS generate the image overlay following the vector field projection. The overlay is constructed once for the dynamic case 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. Figure 4 (top, right) shows only the flow while Figure 4 (bottom, middle) shows only the surface.

I. A Perceptual Consequence: Non-Uniform Density

One of the primary advantages of performing texture synthesis in image space is the gain in performance. However, generating and advecting textures in image space while visualizing object space has perceptual consequences as mentioned previously in Section IV-D.2.

For both ISA and IBFVS, the density of the texture is constant in image space. One can object that this is unnatural

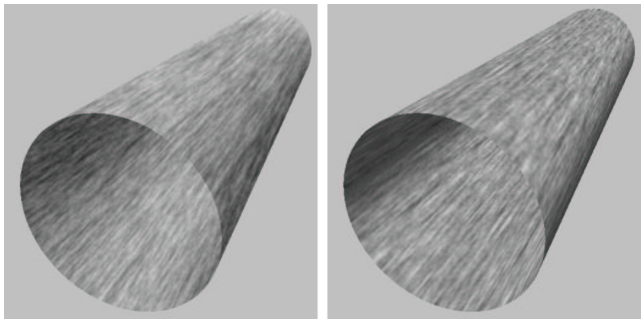

Fig. 9. Uniform density of texture in image space (left) and in object space (right).

and a constant density in world space seems more appropriate. There are three situations where this is visible. Firstly, the density does not depend on the orientation of the surface, i.e., when viewed under an oblique angle the density is the same as when viewed perpendicular. Secondly, the density is independent of the perspective, i.e., surfaces close to the viewer have the same density as surfaces far away. In Figure 9, two textured cylinders are shown with flow along the axis, on the left with a uniform density in image space (produced by IBFVS), on the right with a uniform density in surface space (produced by texture mapping a noise pattern). The difference between these images is small, and the left image does not necessarily appear unnatural.

The third situation where this is visible is in the case of zooming. With image-based texture, the density of the texture remains constant in image space. This is highly advantageous for visualization purposes, but unnatural from a physical point of view. In the real world, texture is scaled when we approach an object and other higher frequency details become visible. With computer-generated imagery however, a uniform density in object space can lead to unnatural effects. When a textured surface is close to the viewer, the texture is scaled strongly and appears blurred, whereas surface parts far away have a high density and usually appear sharp. Although distant textures may also suffer aliasing artifacts, especially in the case of high frequency textures. ISA and IBFVS do not suffer from these such aliasing artifacts.

J. ISA Texture Clipping and IBFVS Texture Interpolation

In ISA, 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., 128^2, for faster performance or a fine resolution mesh, e.g., 512^2, for higher accuracy. However, if the resolution of the advection mesh is too coarse in image space, artifacts appear. Figure 10, left, illustrates these artifacts zoomed in on the edge of a surface. In order to minimize the jagged edges created by coarse resolution texture quadrilaterals, ISA applies a texture clipping function. Subsets of textured quadrilaterals that do not cover the surface are clipped from the visualization as shown in Figure 10, right. This can be implemented simply with the image overlay by assigning unity to the opacity wherever the depth buffer value is maximized, i.e., wherever there is a great depth.

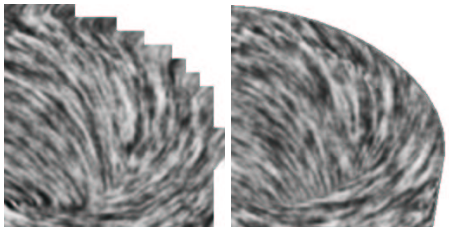

Fig. 10. The result of, left, a coarse resolution advection mesh with artifacts and, right, the application of texture clipping. The resolution of the advection mesh shown on the left is 32×32 for illustration.

In IBFVS the inserted noise also has to be clipped to the projection of the mesh. This is realized by using a texture mapped rectangle at great depth, where the z-buffer test is set such that it is only visible when an object is in front. The previous image is texture mapped directly on the 3D mesh, which sometimes gives rise to a texture interpolation artifact. The scan conversion of a triangle involves interpolation of edges, colors, and also texture coordinates. Hardware offers the option to interpolate texture coordinates perspectively correct. Artifacts become visible when large triangles are rendered with a high depth gradient. The texture synthesis process is two-dimensional, assuming a linear interpolation of texture coordinates. One solution could be simply to turn off perspectively correct texture interpolation. Another solution is to use only small triangles or to switch to isometric projection for close-up views.

V. Applications of ISA and IBFVS

For applications, we start with flow visualization, the original impetus for this work, followed by a discussion of how the perception of surfaces can be enhanced by adding texture.[1]

A. Flow Visualization

Our visualization techniques have been applied to large, highly irregular, adaptive resolution meshes commonly resulting from CFD simulations.

1) Computational Fluid Dynamics: Figure 12 illustrates ISA applied to a surface of the intake port mesh shown in Figure 2 composed of 221K polygons. The intake port is composed of polygons with highly varying sizes for which no global parameterization is readily available. The methods here allow the user to zoom in at arbitrary view points always maintaining a high spatial resolution visualization.

The methods apply equally well to meshes with time-dependent geometry and topology. Figure 11 shows the surface of a piston cylinder with the piston head (not shown) defining the bottom of the surface. The methods here enable the visualization of fuel intake as the piston head slides down the cylinder. The resulting flow visualization has a smooth spatio-temporal coherency.

[1]For supplementary material, including MPEG animations, please visit: `http://www.VRVis.at/ar3/pr2/tvcg04/`

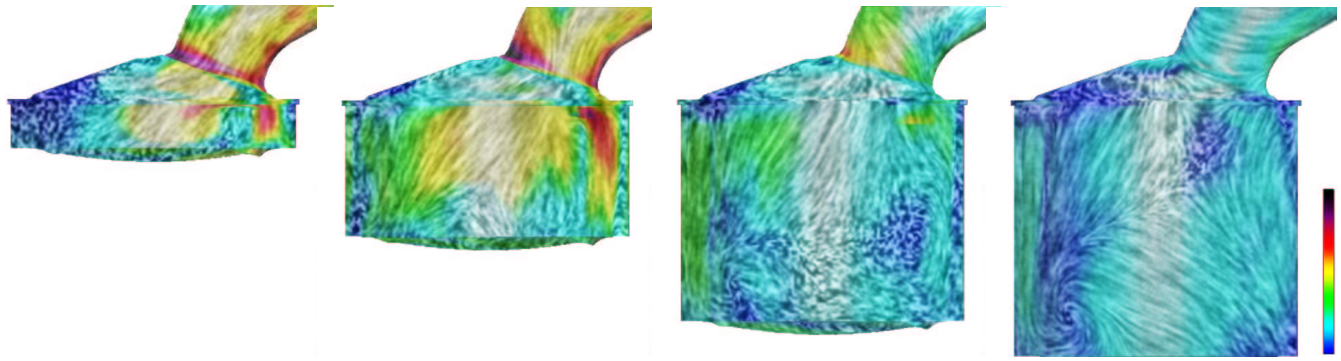

Fig. 11. Snapshots from the visualization of a time-dependent surface mesh composed of 79K polygons with dynamic geometry and topology.

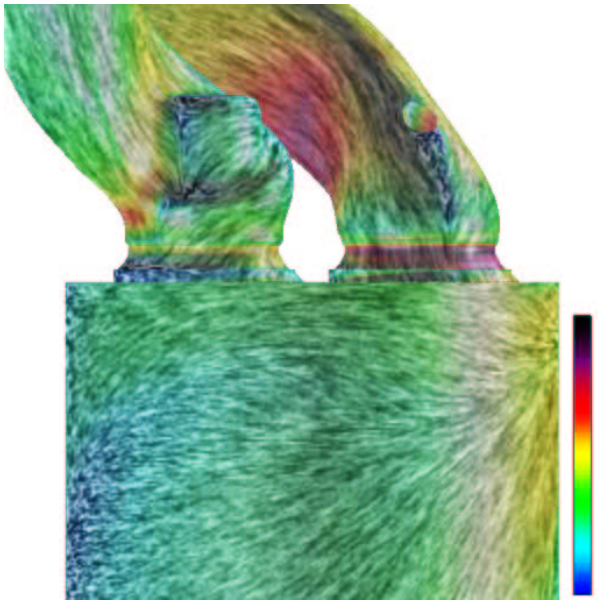

Fig. 12. A side view of the surface of an 221K polygonal intake port mesh. Texture-based flow visualization is applied to the surface.

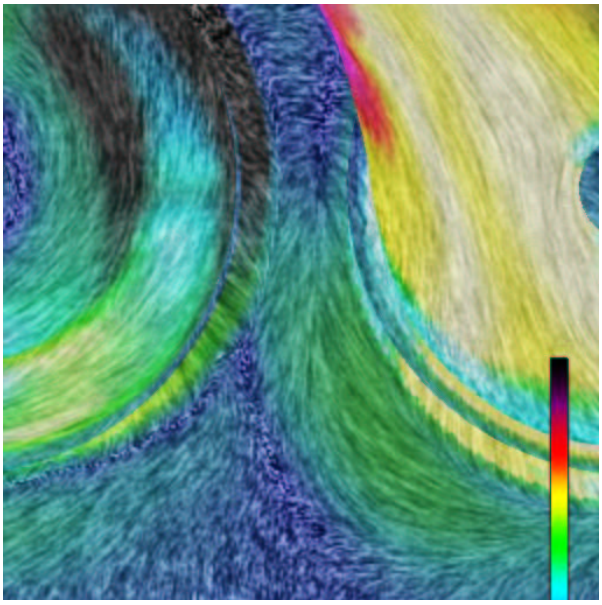

Fig. 13. A close-up, top view of the surface of the intake port mesh shown in Figure 12. Here we illustrate user-supported zooming with automatic, on the fly recalculation of the flow texture.

2) Meteorological Data: Image-based flow visualization also applies to meteorological data. Figure 1 bottom, shows the average wind stress field averaged over more than 100 years for the month of January. On the left, a map of the world is shown, and the strength of the texture is modulated with the magnitude of stress. On the right, the magnitude is visualized via color. Various features, especially vortices, show up clearly. The user can rotate and zoom in on the globe and view the variation of the flow over the year. Features like the monsoon in India and the circulation around Antarctica are clearly visible.

3) Medical Visualization: Our algorithms also have applications in the field of medicine. Figure 14 shows the circulation of blood at the junction of three blood vessels. An abnormal cavity has developed that hinders the optimal distribution of blood. ISA and IBFVS have also been applied to the visualization of surface topology [29] and isosurfaces [14].

B. Surface Visualization

The rendering of surfaces can be considered a visualization problem. The definition of the characteristics of surfaces has been studied intensively in differential geometry. We start with a very brief summary. A local first-order approximation of a surface is a tangent plane. A local second-order approximation is given by two orthogonal principal directions. Along these principal directions the curvature (1/radius) of a line on the surface through the tangent point is either minimal or maximal. We denote these curvatures by κ_{min} and κ_{max}.

Interrante et al. have presented a variety of methods to visualize surface shape based on differential geometry [9]. They have shown how valleys and ridges on the surface can be detected and emphasized by lines [10]. They have studied how texture, aligned with principal directions, can aid in understanding surface shape [7]. Another approach to visualizing surface features is releasing particles on the surface and moving them according to a principal direction [17].

Inspired by these results, we have studied if ISA and IBFVS

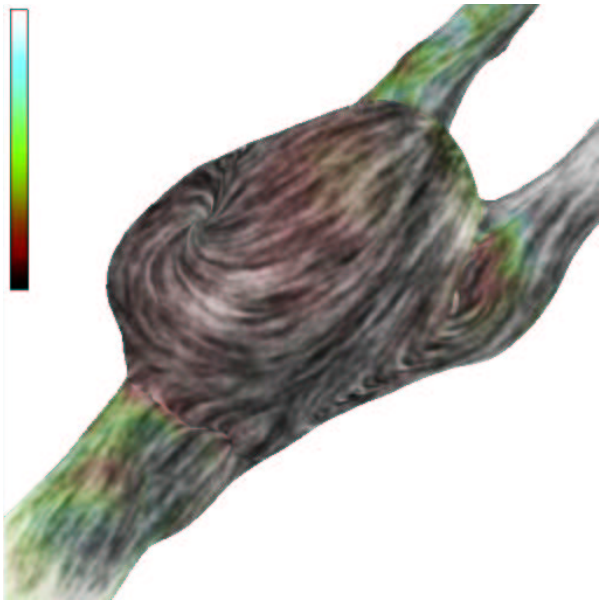

Fig. 14. Medical visualization: blood flow at the surface of the junction of three blood vessels. Stagnant blood flow may occur within the abnormal pocket at the junction.

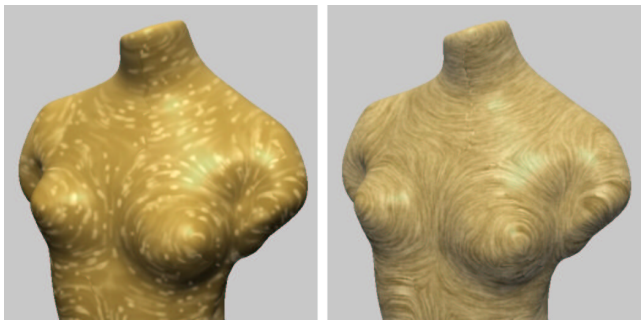

Fig. 15. On the left, moving particles and on the right a dense texture.

can be used for this purpose.

Moving particles can be generated by using a special setting of the input noise texture G. Figure 15, left, shows a result. We used a sequence of short spikes for $w(t)$, such that only a small fraction of the grid values $G_{i,j,k}$ are white. Furthermore, to achieve a higher brightness we added the texture instead of blending it. The flow field is aligned with the direction of κ_{min}.

A more standard setting of G produces a dense texture, aligned with the principal directions, e.g., Figure 15, right. One problem here is that differential geometry only gives an orientation and not a signed direction for the principal directions. We have experimented with several solutions for this. The best results were obtained by aligning the principal directions with a user defined vector (typically a coordinate axis), followed by a rotation over 90 degrees around the normal. However, a globally optimal perfect solution can, by definition, not be achieved. Hence, a separation line on the breast can be seen.

For smooth surfaces such as the torso, good results can be obtained. For more irregular surfaces the results are not as encouraging. An improved result is obtained when we modulate the strength β of the texture with respect to the shading by κ_{max}, and use κ_{max} also to modulate the color of the surface. In other words, we use a combination of two methods of Interrante et al. The images give the impression that a dirty surface has superficially been cleaned (Figure 16). It can be shown that this effect is similar to local accessibility shading [21]. An effect of eroded stone is obtained when parameterization with respect to κ_{min} is used. Note that the input consists only of a triangular mesh. All detail is generated automatically. We found that default settings for colors and bounds for the parameterization can be defined that give good results for a variety of geometries. Further examples of surface visualization are illustrated by Van Wijk [29].

C. Performance

We have implemented both ISA and IBFVS in the same software application in order to facilitate comparison of the two. Our implementation is based on OpenGL 1.1. Performance for both ISA and IBFVS was evaluated on a PC with an Nvidia 980XGL Quadro graphics card, a 2.8 GHz dual-processor and 1 GB of RAM. The performance times reported in Table 1 support interactive exploration of unsteady flow on surfaces. Triangle strips and OpenGL display lists were employed to accelerate the frame rates.

For comparing relative performance, we use the following notation: N denotes the number of polygons contained in the object space mesh, M denotes the effective number of polygons used by ISA in the image mesh, and C indicates a constant time factor. For ISA, M is a user-defined constant function of the (image space) texture advection resolution and the percentage of image space covered by the object. More precisely, $M = r_m^2 p$, where r_m^2 is the texture advection mesh resolution and p is the percentage of image space covered by the surface mesh after rendering. In our test cases shown in Table 1, we covered about 75% of image space, with the exception of the cooling jacket, which covers only 39% of image space.

The first times reported (in Table 1) in the FPS columns are for the static cases of steady-state visualization and the absence of changes to the view point. The times shown in parentheses indicate the dynamic cases of unsteady flow and interactive zooming and rotation illustrated in Figure 3. We include geometric edge detection in the ISA frame rates reported in Table 1. It does not introduce significant overhead since it is easily built into the ISA advection process.

Table 2 shows for each step the dependency on the number of polygons N in the original mesh and the number of polygons M of the image mesh used by ISA. The performance time of IBFVS depends linearly on N for both the static and dynamic case. For the dynamic case, projections for all vertices have to be calculated and updated texture coordinates must be sent to the GPU, leading to a significantly lower performance in the dynamic case.

For ISA, the performance depends linearly on N and M for the dynamic case, but only on M for the static case. As a result, we see that again the dynamic case has a lower performance. For the static case, ISA outperforms IBFVS when $M < N$, i.e., when using coarse image meshes for

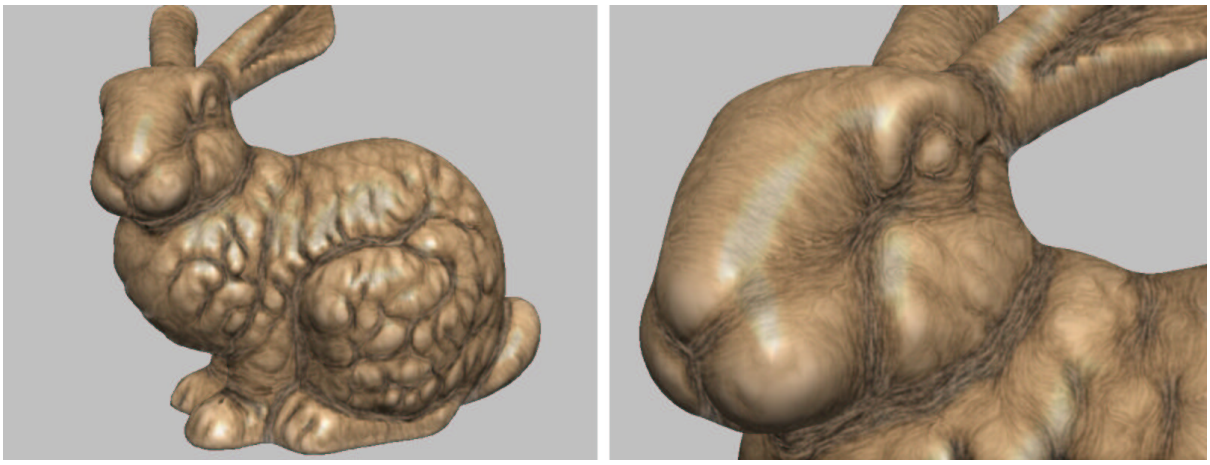

Fig. 16. (left) texture modulated with respect to shading and color for κ_{max} and (right) the same scene but zoomed in.

data set	no. of polys	IBFVS FPS	ISA adv. mesh res.	ISA FPS
ring (Fig 4)	10K	64 (49)	64^2	64 (38)
			128^2	64 (20)
			256^2	31 (7.5)
			512^2	14 (2.7)
combustion chamber (Fig 11)	79K	56 (38)	64^2	64 (38)
			128^2	64 (18)
			256^2	30 (7.5)
			512^2	15 (2.7)
intake port (Fig 12)	221K	13 (3.0)	64^2	64 (13)
			128^2	64 (10)
			256^2	30 (5.3)
			512^2	15 (1.9)
cooling jacket (Fig 1 top)	228K	13 (2.7)	64^2	64 (15)
			128^2	64 (13)
			256^2	47 (8.0)
			512^2	20 (3.1)

TABLE I
SAMPLE FRAME RATES FOR THE ISA AND IBFVS VISUALIZATION ALGORITHMS.

Stage	ISA	IBFVS
Dynamic Case:		
project vector field	N	N
compute texture coordinates	M	N
Static Case:		
texture map mesh	M	N
inject and blend noise	C	C
save image to texture memory	C	C
render image overlay	C	C

TABLE II
THE PERFORMANCE DEPENDENCY OF EACH STAGE OF THE ISA AND IBFVS PIPELINES: N INDICATES THE NUMBER OF POLYGONS IN OBJECT SPACE AND M INDICATES THE NUMBER OF POLYGONS IN IMAGE SPACE.

small polygon meshes, or when the original polygon meshes are larger than ≈200K polygons.

VI. Discussion and Conclusions

The choice of whether to apply ISA or IBFVS depends on the complexity of the model. For surface visualization like that shown in Figure 16, IBFVS is a good choice. Polygons generally cover several pixels in image space and the amount of computation time per-pixel is low. For visualization of flow on large meshes such as Figures 1 top, and 12, ISA is a good choice. For these meshes, many polygons cover less than a pixel and many polygons are occluded. ISA avoids spending computation time on these pixels. Also, in terms of software development, IBFVS is generally easier to implement because it is a more straightforward extension of IBFV.

We also point out that ISA and IBFVS are much faster than previous attempts to depict both steady and unsteady flow on surfaces using texture synthesis. This includes the previous work by Forssell and Cohen [6] to extend LIC to curvilinear grids, Battke et al. [1] to extend Fast LIC to arbitrary surfaces, and Mao et al. [18] who extend LIC to arbitrary triangular meshes with steady-state flow. In addition, we have not seen much work in the area of texture-based flow visualization on surfaces in general since the introduction of UFLIC (Unsteady Flow LIC) by Shen and Kao [25] in 1998, which also used a parameterization.

One open question concerning the performance times of ISA and IBFVS arises in the case of programmable graphics hardware. In the case of surfaces the algorithms might be even faster if they did not use the mesh vertices for warping, i.e., if they used a velocity image as per-pixel texture coordinate offsets, even though dependent texture reads are slower than default texture mapping. There is however at least one good reason for not choosing this route and that is namely portability. ISA and IBFVS do not rely on proprietary programmable graphics hardware and thus are not bound to any specific graphics card.

VII. Summary and Future Work

We have presented a side-by-side comparison of two novel techniques for texture synthesis of unsteady flow on boundary surfaces. We have identified where ISA and IBFVS overlap and where they diverge. We also identify the relative strengths and weaknesses of each approach and offer our views as to where the methods are best applied. The algorithms support visualization of flow on arbitrary surfaces, at over 60 FPS in

many cases. ISA and IBFVS support exploration and visualization of flow on large, unstructured polygonal meshes, and on time-dependent meshes with dynamic geometry and topology. While the vector fields are defined in 3D and associated with arbitrary triangular surface meshes, the generation and advection of texture properties is derived in image space.

Dense flow-aligned textures on surfaces, generated at high speed, are useful for many application areas. In the application sections we have presented examples for flow visualization, medical visualization, meteorology, and for more general surface visualization.

Future work can go in many directions including visualization of unsteady 3D flow, something we expect to see soon. Challenges will include both interactive performance time and perceptual issues. Future work also includes the application of more specialized graphics hardware features like programmable per-pixel operations in the manner of Weiskopf et al. [30].

ACKNOWLEDGMENTS

Portions of this work have been done via a cooperation between two research projects of the VRVis Research Center (`www.VRVis.at`) which is funded by the Austrian research program Kplus (`www.kplus.at`) in cooperation with AVL (`www.avl.com`). We would also like to extend a special thanks to Jürgen Schneider of AVL for his valuable insight into the CFD simulation data sets. Thanks to Jeroen van der Zijp and the FOX Windowing Toolkit (`www.fox-toolkit.org`) for help with the implementation. Thanks to Michael Mayer for the medical simulation data and Markus Hadwiger for his careful proofreading of the manuscript. All CFD simulation data presented here is courtesy of AVL.

REFERENCES

[1] H. Battke, D. Stalling, and H.C. Hege. Fast Line Integral Convolution for Arbitrary Surfaces in 3D. In *Visualization and Mathematics*, pages 181–195. Springer-Verlag, Heidelberg, 1997.

[2] B. Cabral and C. Leedom. Highly Parallel Vector Visual Environments Using Line Integral Convolution. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 802–807, Philadelphia, PA, February 1995. SIAM.

[3] 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. ACM Press / ACM SIGGRAPH, 1993.

[4] N. A. Carr and J. C. Hart. Meshed Atlases for Real-Time Procedural Solid Texturing. *ACM Transactions on Graphics*, 21(2):106–131, April 2002.

[5] L. K. Forssell. Visualizing Flow over Curvilinear Grid Surfaces Using Line Integral Convolution. In *Proceedings IEEE Visualization '94*, pages 240–247, Los Alamitos, CA, October 1994. IEEE Computer Society.

[6] 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.

[7] G. Gorla, V. Interrante, and G. Sapiro. Texture Synthesis for 3D Shape Representation. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):512–524, October/December 2003.

[8] W. Heidrich, R. Westermann, H.-P. Seidel, and T. Ertl. Applications of Pixel Textures in Visualization and Realistic Image Synthesis. In *ACM Symposium on Interactive 3D Graphics*, 1999.

[9] V. Interrante. Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution. In *Proceedings of ACM SIGGRAPH 97*, Annual Conference Series, pages 109–116. ACM SIGGRAPH, ACM Press, August 1997.

[10] V. Interrante, H. Fuchs, and S. Pizer. Enhancing Transparent Skin Surfaces with Ridge and Valley Lines. In *Proceedings IEEE Visualization '95*, pages 52–59, 1995.

[11] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-Eulerian Advection for Unsteady Flow Visualization. In *Proceedings IEEE Visualization '01*, San Diego, California, October 2001. IEEE.

[12] B. Jobard, G. Erlebacher, and Y. Hussaini. Lagrangian-Eulerian Advection of Noise and Dye Textures for Unsteady Flow Visualization. In *IEEE Transactions on Visualization and Computer Graphics*, volume 8(3), pages 211–222, 2002.

[13] R. S. Laramee, B. Jobard., and H. Hauser. Image Space Based Visualization of Unsteady Flow on Surfaces. In *Proceedings IEEE Visualization '03*, pages 131–138. IEEE Computer Society, 2003.

[14] R. S. Laramee, J. Schneider, and H. Hauser. Texture-Based Flow Visualization on Isosurfaces from Computational Fluid Dynamics. In *Joint Eurographics-IEEE TVCG Symposium on Visualization (VisSym '04)*, Konstanz, Germany, May 19-21 2004. forthcoming.

[15] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least Squares Conformal Maps for Automatic Texture Atlas Generation. In *SIGGRAPH 2002 Conference Proceedings*, Annual Conference Series, pages 362–371, 2002.

[16] W. E. Lorensen and H. E. Cline. Marching Cubes: a High Resolution 3D Surface Construction Algorithm. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, pages 163–170. ACM, July 27–31 1987.

[17] E. B. Lum, A. Stompel, and K. L. Ma. Kinetic Visualization: A Technique for Illustrating 3D Shape and Structure. In *Proceedings IEEE Visualization 2002*, pages 435–442. IEEE Computer Society, October 27– November 1 2002.

[18] X. Mao, M. Kikukawa, N. Fujita, and A. Imamiya. Line Integral Convolution for 3D Surfaces. In *Visualization in Scientific Computing '97. Proceedings of the Eurographics Workshop*, pages 57–70. Eurographics, 1997.

[19] N. Max and B. Becker. Flow Visualization Using Moving Textures. In *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data*, September 1995.

[20] N. Max and B. Becker. Flow Visualization Using Moving Textures. In *Data Visualization Techniques*, pages 99–105, 1999.

[21] G. Miller. Efficient Algorithms for Local and Global Accessibility Shading. In *Proceedings of ACM SIGGRAPH 94*, Annual Conference Series, pages 319–326. ACM SIGGRAPH, ACM Press / ACM SIGGRAPH, 1994.

[22] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. Feature Extraction and Visualization of Flow Fields. In *Eurographics 2002 State-of-the-Art Reports*, pages 69–100, Saarbrücken Germany, 2–6 September 2002. The Eurographics Association.

[23] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The State of the Art in Flow Visualization: Feature Extraction and Tracking. *Computer Graphics Forum*, 22(4):775–792, 2003.

[24] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive exploration of volume line integral convolution based on 3D-texture mapping. In *Proceedings IEEE Visualization '99*, pages 233–240, San Francisco, 1999. IEEE Computer Society.

[25] H.W. Shen and D. L. Kao. A New Line Integral Convolution Algorithm for Visualizing Time-Varying Flow Fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2), April – June 1998.

[26] D. Stalling. LIC on Surfaces. In *Texture Synthesis with Line Integral Convolution*, pages 51–64. ACM SIGGRAPH 97, International Conference on Computer Graphics and Interactive Techniques, 1997.

[27] J. J. van Wijk. Spot noise-Texture Synthesis for Data Visualization. In Thomas W. Sederberg, editor, *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, volume 25, pages 309–318. ACM, 1991.

[28] J. J. van Wijk. Image Based Flow Visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002.

[29] J. J. van Wijk. Image Based Flow Visualization for Curved Surfaces. In *Proceedings IEEE Visualization '03*, pages 123–130. IEEE Computer Society, 2003.

[30] D. Weiskopf, G. Erlebacher, and T. Ertl. A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields. In *Proceedings IEEE Visualization '03*, pages 107–114. IEEE Computer Society, 2003.