

Image Based Flow Visualization for Curved Surfaces

Jarke J. van Wijk
Technische Universiteit Eindhoven

Abstract

A new method for the synthesis of dense, vector-field aligned textures on curved surfaces is presented, called IBFVS. The method is based on Image Based Flow Visualization (IBFV). In IBFV two-dimensional animated textures are produced by defining each frame of a flow animation as a blend between a warped version of the previous image and a number of filtered white noise images. We produce flow aligned texture on arbitrary three-dimensional triangle meshes in the same spirit as the original method: Texture is generated directly in image space. We show that IBFVS is efficient and effective. High performance (typically fifty frames or more per second) is achieved by exploiting graphics hardware. Also, IBFVS can easily be implemented and a variety of effects can be achieved. Applications are flow visualization and surface rendering. Specifically, we show how to visualize the wind field on the earth and how to render a dirty bronze bunny.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.6.6 [Simulation and Modelling]: Simulation Output Analysis

Keywords: Flow visualization, texture mapping, line integral convolution, surface rendering

1 Introduction

Visualization is indispensable to achieve insight in the large data sets produced by Computational Fluid Dynamics (CFD) simulations. Many methods for flow visualization have been developed, ranging from arrow plots and streamlines to dense texture methods. The latter class of methods produces clear visualizations, but its use in practice has been hampered by a poor efficiency.

Recently we have introduced Image Based Flow Visualization (IBFV) [26]. This method provides a single framework to generate a wide variety of visualizations of two-dimensional fluid flow, varying from moving particles, streamlines, moving textures, to topological images; it can handle arbitrary grids and unsteady flow; a high efficiency is achieved by exploiting graphics hardware; and finally the method is easy to implement.

In this article we present how IBFV can be extended to produce vector field aligned textures on curved surfaces. We named the new method IBFVS. The texture mapping is solved in line with the original method: Textures are directly generated in image space. As a result, IBFVS inherits the advantages of IBFV.

The typical application domain is flow visualization, but vector field aligned textures can also be used for other purposes. We show how surface details can be depicted realistically by textures derived from and modulated by local surface curvature.

In the next section related work is discussed, in section 3 the new method is described and analyzed extensively. In section 4 applications are presented. The implementation and performance are described in section 5, in section 6 the results are discussed. Finally, conclusions are drawn.

2 Related work

Many methods have been developed to visualize flow. Arrow plots are a standard method, but it is hard to reconstruct the flow from discrete samples. Streamlines and advected particles provide more insight, but also here important features of the flow can be overlooked.

The visualization community has spent much effort in the development of more effective techniques. Van Wijk [25] introduced the use of texture to visualize flow with Spot Noise. Cabral and Leedom [1] introduced *Line Integral Convolution* (LIC), which gives a much better image quality.

Many extensions to and variations on these original methods, and especially LIC, have been published. The main issue is improvement of the efficiency. Improved numerical schemes [24], the use of parallel processing [2, 30, 22], and the use of graphics hardware have been proposed [3, 18, 8, 12, 13]. A major advancement was the introduction of Image Based Flow Visualization (IBFV) [26]. IBFV is based on warping and blending images, thereby modeling the transport, decay, and injection of ink. Moving textures are produced by insertion of scaled white noise patterns. Van Wijk shows how artifacts can be prevented by low-pass filtering the input textures in space and time. IBFV has many advantageous features, but is limited to two-dimensional fluid flow fields.

How can we use this method to visualize flow on curved surfaces? This would drastically enlarge the scope of the method. In many CFD simulations flow on curved surfaces, such as car and airplane bodies, plays an important role, and also, flow aligned textures can be used for other purposes, such as visualization of surface shape [7].

The obvious approach is to use standard texture mapping: A texture is defined as a separate image in *texture space*, mapped on the surface in 3D in *world space*, and finally the surface is projected to *image space*. This route has been pursued for Spot Noise [25] as well as LIC [5]. Can we use this path also for IBFV? An exhaustive treatment of all methods for texture mapping and texture synthesis is far beyond the scope of this paper. Instead, we mention a few approaches as an illustration.

Both for Spot Noise and for LIC textures are displayed on parametrized surfaces: The given parametrization of the surface is used to relate points on the surface with points on the texture. This requires that the distortion from parametric space to geometric space is taken into account to achieve a uniform density. Also, CFD data often have a strongly varying density of the parametrization. Straightforward use of parameter space as texture space would then imply a waste of texture memory. Possibly a reparametrization can be used to compensate for this.

Finite element simulations often do not have a given parametrization, in which case some parametrization must be derived. This problem has been studied heavily, see for instance [4, 23, 15, 7]. Irregular topologies introduce complex problems, to find a parametrization without too much distortion requires much care. A next source of problems for the standard approach concerns the density of the texture. In an interactive session a user wants to switch from overviews to close-ups, for instance to study turbulent flow on different scales. One would prefer here that the density, i.e. the graininess of the texture is constant in image space:

When zooming in, automatically a more fine grained texture in world space is shown. The standard approach leads here either to the use of very fine textures, or to ad hoc generation of texture, which requires again quite some effort to implement.

In summary, via the standard texture mapping approach "generation of texture on arbitrary surfaces in 3D should be feasible" for IBFV, quoting [26], but it is not an easy route, and it is doubtful if a similar performance and generality as for the 2D case can be achieved.

We propose a different approach, based on the following observations. Firstly, the texture should be dense in image space. Secondly, the most efficient approach is to produce texture only where needed: in the image. Finally, image coherence can be exploited, also when the image shows flow on a curved surface. All these observations point in the same direction. Entirely in the spirit of the original method, let us produce the texture directly in image space, or, in other words, let texture space and image space coincide. In the next section we elaborate on this.

3 Method

In this section we start with a summary of 2D IBFV, with an emphasis on the synthesis of dense textures. More details are given in the original paper [26]. Next our new method IBFVS is described, followed by a discussion of various aspects in more detail.

3.1 2D IBFV

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

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

defined for $t \geq 0$, and for $\mathbf{x} \in \Omega$, where $\Omega \subset \mathbb{R}^2$. A pathline is the path of a particle, advected by the flow, which path is the solution of the differential equation

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

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

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

with $k \in \mathbf{N}$ and $t = k\Delta t$. In the remainder we use the frame number k as parameter of 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-triplet. The property is advected just like a particle, so a first order approximation of the transport of F can hence be given by:

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

Sooner or later most of $F(\mathbf{x}; k)$ will have disappeared, so at each time step a convex combination of F and another image G is taken:

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

where the points \mathbf{p}_k are again defined by equation (3), and where $\alpha = \alpha(\mathbf{x}; k) \in [0, 1]$ defines a blending mask. 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.

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

where s is a parameter that controls the scale of the texture, where $h(x)$ is a triangular pulse

$$h(x) = \begin{cases} 1 - |x| & \text{if } |x| \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

and where the instantaneous grid values $G_{i,j;k}$ are defined by

$$G_{i,j;k} = w(v_g k + \phi_{i,j}). \quad (8)$$

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; 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 covered with a rectangular or triangular mesh. First, a mesh is distorted and rendered, using the preceding image as a texture map; second, fresh ink is added by blending in a polygon, texture mapped with a scaled, interpolated, pre-computed pattern; third, the image is stored in texture memory for the next round; and finally additional graphics is drawn and the image is shown on the screen.

3.2 IBFV for curved surfaces

We next consider the visualization of flow on surfaces. We use a triangular mesh as the geometric model for surfaces. Triangular meshes are a generic format, in the sense that almost all surface representations can be converted to triangular meshes, and by using more triangles an arbitrary level of detail can be achieved. Hence we claim that the following is applicable to arbitrary surfaces. Another advantage is obviously that triangles are a basic primitive for graphics hardware.

Suppose that for each vertex i its position $\mathbf{r}_i = (r_{ix}, r_{iy}, r_{iz})$, a normal vector \mathbf{n}_i 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. We assume that this velocity is confined to the surface (possibly by projection), i.e. $\mathbf{v}_i \cdot \mathbf{n}_i = 0$, later on we will discuss the more general case.

Given a composite modeling, viewing, and perspective transformation, the mesh can be projected on the screen. We denote this by $\mathbf{x}' = T(\mathbf{x})$, where $\mathbf{x}' = (x_s, y_s)$ is a 2D point on the screen and where T denotes the transformation. We use apostrophes to denote projected quantities. The key to 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, i.e.

$$F(\mathbf{p}'_{k+1}; k+1) = \begin{cases} F(\mathbf{p}'_k; k) & \text{if } \mathbf{p}'_k \in \Omega' \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

where Ω' is the projection of the surface on the image F , and where we assume that the path line \mathbf{p} is visible.

We use this to define a process for the synthesis of flow aligned texture on curved surfaces. Two further aspects must be taken care of. Firstly, fresh ink has to be added only to the projection of the surface and not to the background. Secondly, as an additional cue for the shape of the surface, a shaded image $F_s(\cdot, k)$ has to be blended in. We use a separate texture image $F_t(\cdot, k)$, 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:

$$F_t(\mathbf{p}'_k; k) = (1 - \gamma(\mathbf{p}'_k))F_t(\mathbf{p}'_{k-1}; k-1) + \gamma(\mathbf{p}'_k)G(\mathbf{p}'_k; k), \quad (10)$$

$$F(\mathbf{x}'; k) = \beta F_t(\mathbf{x}'; k) + (1 - \beta)F_s(\mathbf{x}'; k), \quad (11)$$

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

$$\gamma(\mathbf{x}') = \begin{cases} \alpha(\mathbf{x}') & \text{if } \mathbf{x}' \in \Omega' \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

where α has the same meaning as in standard IBFV.

3.3 Algorithm

Just as with standard IBFV, the preceding equations can be translated into a fast algorithm that exploits graphics hardware. The steps are as follows:

1. Initialize the texture image F_t with a background color B ;
2. Initialize the image F with B ;
3. Calculate texture coordinates $\mathbf{t}_i = (t_{ix}, t_{iy})$ for all vertices;
4. Render the mesh, texture mapped with F_t , without shading;
5. Blend in fresh noise G on the surface;
6. Store the result in F_t ;
7. Render the mesh shaded, and blend it with the image F ;
8. Draw additional graphics;
9. Show the image on the screen;
10. Update state and go back to step 2.

Figure 1 shows the main steps (4-7) of the algorithm schematically. In the following we consider the algorithm in more detail. Step 1 is an initialization. In step 2 the image F is cleared. This is different from standard IBFV, Van Wijk stated that he obtained the best results with respect to boundary artifacts if the image was not cleared in between. However, in his case the complete image was filled, whereas here the image is only partially filled with a projection of the mesh. Hence, we must clear the image per frame to prevent ghost images when for instance the view point is changed.

Step 3 and 4 are crucial. Here the pattern is advected in screen space by warping the texture. The texture coordinates are

$$\mathbf{t}_i = T(\mathbf{r}_i - \mathbf{v}_i \Delta t), \quad (13)$$

in other words, as texture coordinates we use the projected position of the previous point on the path line through the vertex \mathbf{r}_i . These texture coordinates are computed in software by the central processor. As a result, the texture is slightly warped. Just like in standard IBFV, care has to be taken that the displacement in image space is below a threshold to prevent artifacts. In contrast to standard IBFV, here we prefer to warp the texture, instead of warping the mesh. We found that for time dependent flow a warped mesh gives a distracting motion of silhouette edges and extra textures on the surface. More specific, in an example that we present in a following section, we found that Iceland moved around when the mesh is warped. New texture coordinates only have to be calculated when the viewing transformation changes or when the flow field is dynamic, hence step 3 can be skipped for stationary flow, observed from a static viewpoint. In step 4 no shading is used, just the texture itself to maintain a high contrast and to prevent advection of shade.

In step 5 fresh noise is blended in. For this the same approach as in standard IBFV is used. A sequence of small patterns is pre-computed and stored in texture memory, next these patterns are texture mapped on a rectangle that covers the screen, using scaling and

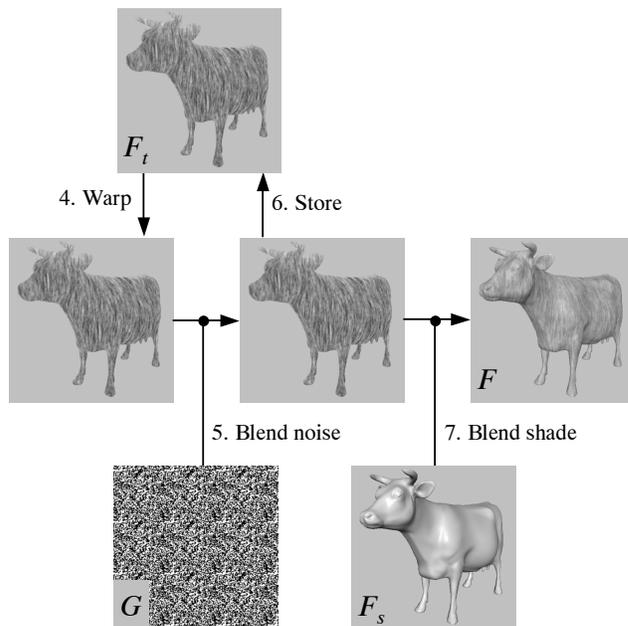


Figure 1: Pipeline IBFVS

interpolation to prevent high frequencies. The texture has to be put only on the (projection of the) surface, noise on the background is very distracting. Fortunately, at this stage we already have exact information available which parts of the image are covered and which are not: in the z-buffer. We exploit this by rendering the rectangle at a great depth, just in front of the back clipping plane, and using a reversed z-buffer test: Only where the current z-value is smaller than the new value, the fresh noise has to be blended in. An alternative could be to use a stencil buffer for this purpose, but we have not explored this further.

In step 6 the intermediate image, which only contains the texture, is stored for step 4 in the next round. This intermediate image is visually not attractive, hence we enhance it in step 7 and 8. In step 7 a shaded version of the mesh is blended in. Many options can be used for this. In the simplest way, just a shaded version is used, but also a texture mapped mesh can be used, with color representing some scalar value, or by showing for instance a geographical map. Also, β (the strength of the texture relative to shading) can be varied over the surface, for instance to show only texture where some variable is above a certain limit. In step 8 extra imagery can be superimposed: grids, vectors, markers, etc. When the image is ready, we show it on the screen in step 9. Finally, the state of the process is updated (user input is processed, possibly a new view transform is derived, new flow field data are obtained, etc.), and we can start with the next image.

This completes our algorithm, which produces flow aligned texture for arbitrary surfaces and arbitrary flow fields at a high speed. Can we get away with this simple approach? Aren't there many artifacts that spoil the result? In the following subsections we consider various aspects in more detail. In general, the method described so far is indeed effective and efficient; only in section 3.7 do we describe a problem for which an additional measure has to be taken.

3.4 Contrast

Lack of contrast was mentioned as a problem for standard IBFV. In 2D, the texture is all that can be seen, and hence a high contrast is attractive. For flow on surfaces, the situation is somewhat differ-

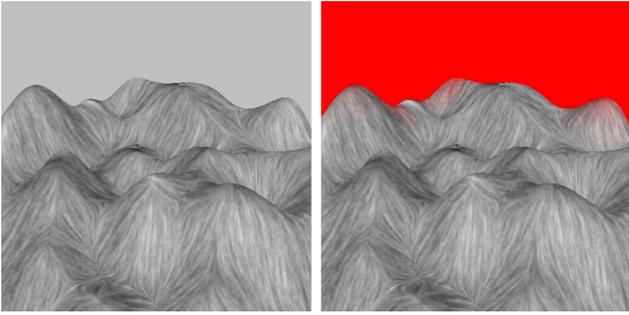


Figure 2: Boundary inflow

ent. Here also the shape of the surface itself is important, depicted by shading, and a high contrast texture disturbs the spatial effect. We found that the use of $\beta = 0.5$ usually gives a good compromise between surface shade and flow texture, and that the decreased contrast is not a big problem here, a subtle texture can still be perceived well on the screen, especially when a moving texture is used. Subtle textures however sometimes do not survive printing or video recording, but here the contrast can be improved separately. For none of the images shown in this paper was extra contrast enhancement used.

3.5 Boundaries and silhouettes

Another problem of standard IBFV is boundary artifacts. In areas near the boundary, no texture is present to be advected, and the background color blends in. A partial solution for standard IBFV was to refrain from clearing the screen in between, such that areas outside the mesh are filled with texture, but, as mentioned earlier, here this solution cannot be used. Surprisingly however, we found for IBFV on surfaces these artifacts to be less disturbing, and even often absent. Several reasons can be given for this. Firstly, the contrast of the texture is less than with standard IBFV and hence also errors show up less clearly. Secondly, near boundaries and silhouette edges often high gradients occur in the shading, which draw the attention of the viewer. Thirdly, when the flow is confined to the surface, and not normalized to a maximum displacement, a strong inflow perpendicular to the boundary does not occur. Finally, by default we use a grey background color instead of black. As this color is closer to the average value of the texture, artifacts are less visible.

As an example, figure 2 shows a landscape with the flow aligned downward with the gradient. The flow is normalized in image space, and $\alpha = 0.1$ and $\beta = 0.4$ were used. When red is used as background color, some inflow across the horizon is visible, when grey is used, no artifacts are visible.

3.6 Non-uniform density

The density of the texture, i.e. the average frequency of strokes on the surface, is constant in image space. One can object that this is unnatural, 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. Consider now figure 3. 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

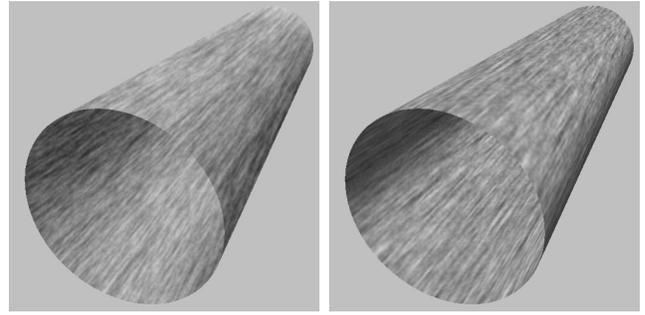


Figure 3: Uniform density in screen (left) and in world space (right)

noise pattern). The difference between these images is small, and the left image does not appear unnatural. Note that the direction of the texture follows the perspective.

The third situation where this is visible, related to the previous one, is when we zoom in on an object. 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 geometric space often leads 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 appear sharp. This is opposite to the natural cue that objects in front should appear sharp and with much detail, whereas objects far away should appear dimmed.

The image on the left does show another small problem. At the front edge of the cylinder, the interior texture makes a smooth transition to the exterior texture. In practice, this is a rare situation that only occurs when the flow in front of and behind a silhouette edge have the same direction.

3.7 Projected texture

Another way to describe our algorithm is that we project noise patterns on the surface, which are advected by subsequent warping. 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 observe that the texture patterns are not fixed to the surface, but are fixed in image space.

This effect is indeed visible. For moving textures the effect is less strong however, the viewer tends to follow the moving texture instead of fixing the view to a point in screen space. Furthermore, several measures can be taken to decrease this effect. Consider an interactive setting, where the user can drag, rotate, and scale the object by dragging the mouse.

First, when the user changes the view, the projection of the texture can be changed as well. We provide the option that both F_t and G are translated according to the motion of the mouse pointer. This improves the perceived imagery strongly for translation, reasonably for rotation, but not for scaling. Secondly, the update of the texture image can be stopped. If we skip step 3 (calculation of new texture coordinates) and step 6 (saving the new texture), the algorithm reduces to standard texture mapping, using the last generated noise pattern as a fixed texture map. When the user stops the manipulation, step 3 and 6 are enabled again, and the last pattern is used as a start for a new animation. This method works well for scaling and dragging, but is less effective for rotation. When the object is rotated, new parts of the object appear for which not yet a sensible texture has been generated.

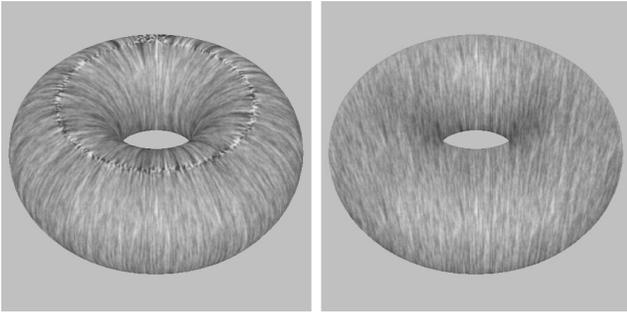


Figure 4: Velocity projected on surface (left) and in 3D (right)

This last method, freezing the texture, can also be used for other purposes, for instance to produce texture maps that have to be processed in a standard way. The method gives both the texture map and the texture coordinates, but is limited to objects where the complete surface is visible in one view and where the variation in depth is not too strong. Later on we will give an example.

When the view is changed, it takes some time before the image or animation stabilizes again. For animations ($v_g \neq 0$) this happens faster than for still images ($v_g = 0$). Consider the worst case, a still image, where the intensity of a pixel has to change from 0 to 1 (the maximum value of $G_{i,j}$). The change ϵ in intensity for the N th step is then equal to $\alpha(1 - \alpha)^N$. Hence, $N = \log(\epsilon/\alpha) / \log(1 - \alpha)$. For $\epsilon = 0.01$ and $\alpha = 0.1$ we find $N \approx 22$, hence with a frame rate of 50 frames per second, the image stabilizes after about half a second.

3.8 Non surface-aligned flow

We so far assumed that the flow was aligned with the surface. How to deal with velocities that have a component perpendicular to the surface? A simple approach is just to use exactly the same formula (eq. 13) for this. Figure 4 shows the result. Both images depict a torus in a vertical flow field, on the left the velocity is projected on the surface, whereas on the right it is not. Which image is correct?

The image on the left shows the surface shape clearly, at the top of the torus the separation line is clearly visible. On the other hand, the image on the right does show the flow field better: The message that this is a simple and uniform flow field is conveyed effectively. We can consider this visualization as the result of using a wire mesh as a flow probe, where the mesh contains a lot of little hairs that all are aligned with the 3D flow. In summary, also non surface-aligned flow can be visualized, but its interpretation is somewhat less easy than for surface-aligned flow.

3.9 Texture interpolation

A final problem to be mentioned concerns texture interpolation. The scan conversion of a triangle involves interpolation of edges, colors, and also texture coordinates. Nowadays hardware offers by default the option to interpolate texture coordinates perspectively correct. Unfortunately, this causes artifacts here. The texture synthesis process works completely in 2D, assuming a linear interpolation of texture coordinates. One solution could be simply to turn off perspectively correct texture interpolation, but this hint is ignored by the drivers and hardware we used. Artifacts become visible when large triangles are rendered in perspective with a high variation in depth. Solutions are therefore to use only small triangles, and/or to switch to isometric projection when close-up views are made.

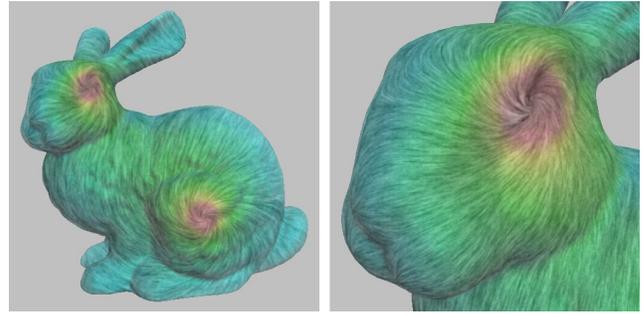


Figure 5: Potential flow field on bunny

4 Applications

Now that we can produce texture aligned with vector fields on curved surfaces, we consider various applications. We start with flow visualization, the original impetus for this work, followed by a discussion how the perception of surfaces can be enhanced by adding texture. More examples, animations, and demo software can be found on the web [27].

4.1 Flow visualization

For testing purposes we have implemented potential flow fields, in line with [29]. A flow field is defined by the superposition of a linear flow field and a number of fields defined by flow elements. A flow element is defined as a line. The user can interactively position these flow elements, and specify how much rotational, tangential, and repelling or attracting flow is desired. Meanwhile, the resulting flow on the surface is shown in real time. Figure 5 shows an example using the Stanford bunny. The flow field is defined by two sources, a repelling and rotating line source through the eyes and an attracting and rotating line source through the legs. When we zoom in, automatically a more detailed texture is produced. Another application is shown in figure 6. Here the average wind stress field on the earth is shown for January [9]. In the left image a map of the world is shown, and the strength of the texture is modulated with the magnitude of the stress; in the right image 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. In the accompanying video we show that features like the monsoon in India and the circulation around Antarctica are clearly visible.

An example of topological decomposition is shown in figure 7. The geometry is a height field, defined as a B-spline surface; and a flow field aligned downward with the gradient is defined. If we

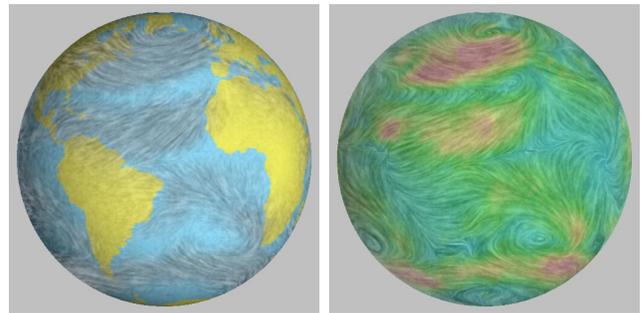


Figure 6: Average wind stress in January

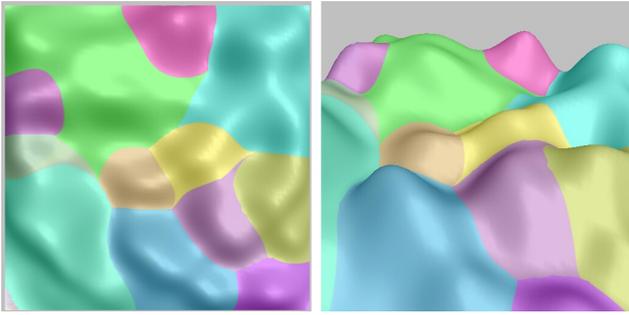


Figure 7: Topological decomposition

view the surface from top, set $\alpha = 0$, and initialize F_t with random colors, the colors are transported downhill from local maxima. After about 30-50 steps, each pixel is assigned a color that is the same as the local maximum that is reached by walking upwards as steeply as possible. Finally, we can lock the generated texture (see section 3.7) and view the surface from oblique angles. The boundaries between the areas are so-called topographic valley lines [20]. If the direction of the flow is reversed, ridge lines are produced, the use of other flow fields can be used to produce for instance Voronoi diagrams.

4.2 Surface visualization

The rendering of surfaces can be considered as a visualization problem. How can we add cues such that we can easily understand a shape and its features? This requires first that we are able to detect the structure and features on the surface. This topic has been studied intensively in differential geometry. We give 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 ($1/\text{radius}$) of a line on the surface through the tangent point is either minimal or maximal.

Interrante and her co-workers have presented a variety of methods to visualize surface shape based on differential geometry. They have shown how valleys and ridges on the surface can be detected and emphasized by lines [11, 10], and they have studied how texture, aligned with principal directions, can aid in understanding surface shape [7]. Another approach to visualize surface features is to drop particles on the surface and let them move according to a principal direction [17].

Inspired by these results, we have studied if IBFVS can be used for this purpose as well. Moving particles can be generated by using a special setting of the input noise G . Figure 8a shows a result. We used a sequence of short spikes for $w(t)$, such that only a small fraction of the grid values $G_{i,j,k}$ is bright white. Furthermore, to achieve a higher brightness we added the texture here 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 (fig. 8b). 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, and for instance on the breast a separation line can be seen.

For smooth surfaces such as the torso, good results can be obtained, for more irregular surfaces the results are somewhat disap-

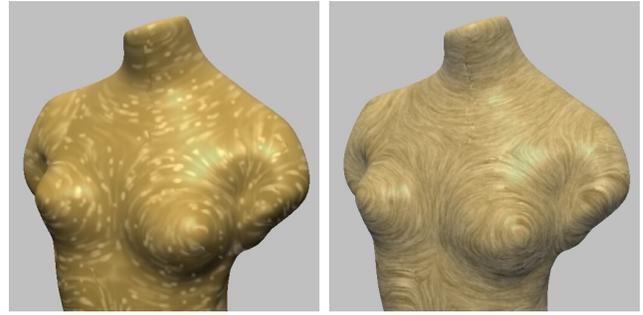


Figure 8: Moving particles (left) and a dense texture (right)

pointing. However, 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 (fig. 9a and b). It can be shown that this effect is similar to local accessibility shading [19]. An effect of eroded stone is obtained when parametrization 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 parametrization can be defined that give good results for a variety of geometries.

5 Implementation

We have implemented IBFVS in an interactive visualization system, which is freely available ([27]). The application was implemented in Delphi 5, using ObjectPascal. It consists of about 5,000 lines of code. All images shown are produced with this system. As input geometry we use triangular meshes: coordinates of vertices and the vertex numbers of triangles. Furthermore, time-dependent flow fields can be read in.

The geometry is processed in various ways after it has been read in. Normals, curvatures and principal directions are calculated using the recommendations and method of Goldfeather [6]. Optionally, the mesh can be refined by application of Loop subdivision [16]. For coarse meshes such as the cow (5800 triangles) or the torso (1400 triangles) this considerably improves the result. Furthermore, triangle strips are constructed to speed up the rendering. For meshes in the order of 100K triangles this preprocessing typically takes a few seconds.

A variety of parameters can be adjusted: colors, scales, mappings, etc. to achieve different effects. We found tuning the visualization to be relatively easy. All parameters have a natural meaning, and the effect of changes is visible immediately. Furthermore, for standard cases, such as shown in figure 8, 9 and ??, we have defined default settings. A wide range of geometries can be visualized in these styles with a single button press.

To access the graphics hardware we used OpenGL1.1. No extensions are needed. We optimized the rendering for the static case, with a constant flow field and constant projection. All rendering bulk data can be put in display lists and named textures, and hence generation of animations can be done at high speed. Table 1 shows performance results for the static case, and between parentheses for the dynamic case (rotating object). We varied image resolution (256^2 , 512^2 , 1024^2) and the resolution of the mesh (5.8K, 23.2K, and 92.8K triangles). As geometric model we used the cow, and for the rendering the base case, as shown in fig. 1. More complex cases, involving for instance modulation of β , are in the order of 5-25 % slower. These results are representative for all images shown

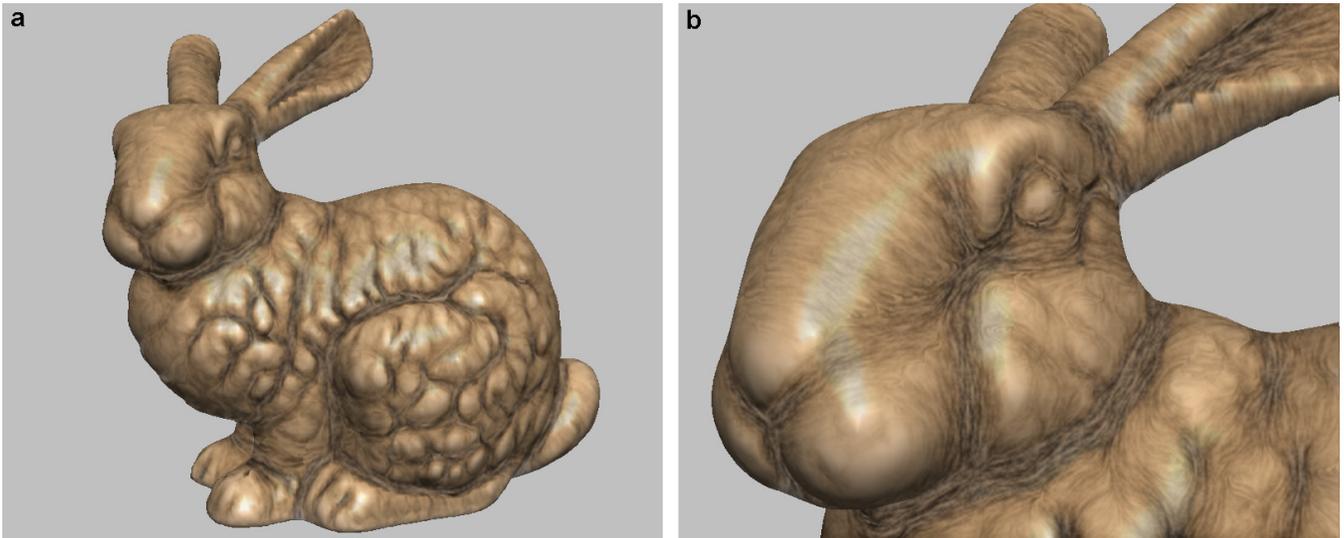


Figure 9: (a) texture modulated for κ_{\max} ; (b) zoomed in

in this paper. Rendering was done on a desktop PC with an AMD Athlon XP1700 processor, and a GeForce4 Ti200 graphics card.

Resolution	5.8 K Δ	23.2 K Δ	92.8K Δ
256×256	600 (76.9)	228 (22.7)	64.0 (6.6)
512×512	290 (76.9)	165 (22.7)	58.8 (6.6)
1024×1024	88.0 (76.9)	72.4 (22.7)	40.0 (6.6)

Table 1: Performance results for texture synthesis in frames per second. Static case, dynamic case between parentheses

Due to faster hardware and possibly a more efficient implementation, the results outperform those of the original method. High-resolution imagery is produced in real-time. For the static case the frame rate depends on the resolution of the image and the number of triangles used. The dynamic case is much slower and dominated by the number of triangles. Here per frame texture coordinates have to be recalculated and resent per frame. We found that resending vertex information took most (about 90 %) of the extra overhead. A much higher performance can possibly be achieved by delegating the texture coordinate calculation to the graphics hardware also.

6 Discussion

We discuss various aspects of IBFVS: Efficiency, versatility, simplicity, and novelty.

IBFVS is about two to three orders of magnitude faster than other methods for the synthesis of dense texture animations of flow on curved surfaces. Our method is the first to exploit graphics hardware for this. Similar as for IBFV, efficiency comes also from three other effects: few steps in the generation of images, exploitation of frame to frame coherence, and sampling of the flow field on the original mesh.

Concerning versatility, we can produce visualizations for arbitrary, time-dependent flow fields defined on arbitrary meshes. Similar as with the original method, various kinds of texture can be produced by variation of the scale s of the texture, the type of time profile $w(t)$, the image variation speed v_g and the blending factor α . By varying the strength of the texture β as a function of properties of the surface, a variety of special effects can be achieved.

Also with respect to simplicity IBFVS inherits all properties from standard IBFV. The method is easy to understand and implement, and does not depend on peculiar features of the graphics hardware.

Concerning the contribution of this work, we have introduced a novel approach to generate dense textures on curved surfaces. By generating these directly in image space, many problems of the standard approach for texture mapping (see section 2) are circumvented. Parametrization is reduced to a simple problem, and, very useful for visualization purposes, a uniform density in image space is achieved.

Dense flow aligned textures on surfaces, generated at high speed, are useful for many application areas. In the application sections we have not only presented examples for flow visualization, but also showed how the method can be used to depict surface detail and to generate artificial drawings. The high performance of the method invites experiments: to try different modulations and combinations of settings, and to use it for different applications.

6.1 Future work

Concerning flow visualization, the next challenge is to develop a similar method for 3D volumes. In [28] such an approach is presented for particles. One fundamental problem here is perceptual. How to depict three-dimensional flow directly in an understandable way is by lack of suitable metaphors still an unsolved problem.

We have used IBFVS for different applications, but it can probably also be used for other types of applications. One area where we were not successful yet, is the rendering of hair and fur. The standard textures produced look somewhat furry, but they cannot compete with existing methods for this, such as [14]. Soft silhouettes are one of the missing features, but possibly this can be remedied. One promising application area is medical visualization: figure 9 already suggests that the method could be used to visualize surface detail of organs such as the human brain. Other possible extensions of the method are the use of multiple layers, flow fields defined by solid texturing [21], spatial or temporal modulation of α , and spatial variation of density.

7 Conclusions

We have presented a simple, efficient, effective, and versatile method for the rendering of dense, vector-field aligned textures on three-dimensional curved surfaces. Unsteady flow on arbitrary surfaces is visualized as animations of texture, at fifty or more frames per second on a PC, using standard features of consumer graphics hardware. The method can be used for a variety of applications, such as flow visualization, surface rendering, and non-photorealistic rendering.

References

- [1] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, pages 263–272, Anaheim, California, August 1993.
- [2] W.C. de Leeuw and R. van Liere. Divide and conquer spot noise. In *Proceedings SuperComputing'97*, 1997.
- [3] W.C. de Leeuw and J.J. van Wijk. Enhanced spot noise for vector field visualization. In *Proceedings IEEE Visualization'95*, 1995.
- [4] M. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3), 1997.
- [5] 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.
- [6] J. Goldfeather. Understanding errors in approximating principal direction vectors. Technical Report 01-006, University of Minnesota, Computer Science and Engineering, 2001.
- [7] G. Gorla, V. Interrante, and G. Sapiro. Texture synthesis for 3d shape representation. *IEEE Transactions on Visualization and Computer Graphics*, 2002.
- [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*, pages 127–134, April 1999.
- [9] S. Hellerman and M. Rosenstein. Normal monthly wind stress over the world ocean with error estimates. *Journal of Physical Oceanography*, 13:1093–1104, 1983.
- [10] V. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. *Computer Graphics*, 31(Annual Conference Series):109–116, 1997.
- [11] V. Interrante, H. Fuchs, and S. M. Pizer. Enhancing transparent skin surfaces with ridge and valley lines. In *Proceedings IEEE Visualization'95*, pages 52–59, 1995.
- [12] B. Jobard, G. Erlebacher, and M.Y. Hussaini. Hardware-accelerated texture advection for unsteady flow visualization. In *Proceedings IEEE Visualization 2000*, pages 155–162, October 2000.
- [13] B. Jobard, G. Erlebacher, and M.Y. Hussaini. Lagrangian-eulerian advection for unsteady flow visualization. In *Proceedings IEEE Visualization 2001*, pages 53–60, October 2001.
- [14] J. Lengyel, E. Praun, A. Finkelstein, and H. Hoppe. Real-time fur over arbitrary surfaces. In *ACM 2001 Symposium on Interactive 3D Graphics*, 2001.
- [15] B. Levy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21(3):362–371, 2002. Proceedings of ACM SIGGRAPH 2002.
- [16] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah, 1987.
- [17] E.B. Lum, A. Stoppel, and K.-L. Ma. Kinetic visualization: A technique for illustrating 3d shape and structure. In *Proceedings IEEE Visualization 2002*, pages 435–442, 2002.
- [18] N. Max and B. Becker. Flow visualization using moving textures. In *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data*, pages 77–87, 1995.
- [19] G. Miller. Efficient algorithms for local and global accessibility shading. In *Proceedings of SIGGRAPH '94, Annual Conference Series*, pages 319–326, 1994.
- [20] R. Peikert and M. Roth. The parallel vectors operator – a vector field visualization primitive. In *Proceedings IEEE Visualization '99*, pages 263–270, 1999.
- [21] K. Perlin. An image synthesizer. *Computer Graphics*, 19(3), 1985. Proceedings of ACM SIGGRAPH 85.
- [22] 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):98–108, 1998.
- [23] C. Soler, M.-P. Cani, and A. Angelidis. Hierarchical pattern mapping. *ACM Transactions on Graphics*, 21(3):673–680, 2002. Proceedings of ACM SIGGRAPH 2002.
- [24] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Proceedings of ACM SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 249–256, August 1995.
- [25] J. J. van Wijk. Spot noise: Texture synthesis for data visualization. *Computer Graphics*, 25:309–318, 1991. Proceedings ACM SIGGRAPH 91.
- [26] J. J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002. Proceedings of ACM SIGGRAPH 2002.
- [27] J.J. van Wijk. Ibfvs: examples and demo software, 2003. <http://www.win.tue.nl/~vanwijk/ibfvs>.
- [28] D. Weiskopf, M. Hopf, and T. Ertl. Hardware-accelerated visualization of time-varying 2d and 3d vector fields by texture advection via programmable per-pixel operations. In *Vision, Modeling, and Visualization VMV '01 Conference Proceedings*, pages 439–446, November 2001.
- [29] J. Wejchert and D. Haumann. Animation aerodynamics. *Computer Graphics*, 25:19–22, 1991. Proceedings ACM SIGGRAPH 91.
- [30] M. Zöckler, D. Stalling, and H.-C. Hege. Parallel line integral convolution. *Parallel Computing*, 23(7):975–989, 1997.