

A Hybrid Physical/Device-Space Approach for Spatio-Temporally Coherent Interactive Texture Advection on Curved Surfaces

Daniel Weiskopf

Thomas Ertl

Institute of Visualization and Interactive Systems
University of Stuttgart

Abstract

We propose a novel approach for a dense texture-based visualization of vector fields on curved surfaces. Our texture advection mechanism relies on a Lagrangian particle tracing that is simultaneously computed in the physical space of the object and in the device space of the image plane. This approach retains the benefits of previous image-space techniques, such as output sensitivity, independence from surface parameterization or mesh connectivity, and support for dynamic surfaces. At the same time, frame-to-frame coherence is achieved even when the camera position is changed, and potential in-flow issues at silhouette lines are overcome. Noise input for texture advection is modeled as a solid 3D texture and constant spatial noise frequency on the image plane is achieved in a memory-efficient way by appropriately scaling the noise in physical space. For the final rendering, we propose color schemes to effectively combine the visualization of surface shape and flow. Hybrid physical/device-space texture advection can be efficiently implemented on GPUs and therefore supports interactive vector field visualization. Finally, we show some examples for typical applications in scientific visualization.

Key words: Flow visualization, vector field visualization, surface visualization, textures, GPU programming.

1 Introduction

Vector field visualization plays an important role in computer graphics and in various scientific and engineering disciplines alike. For example, the analysis of CFD (computational fluid dynamics) simulations in the aerospace and automotive industries relies on effective visual representations. Another field of application is the visualization of surface shape by emphasizing the principal curvature vector fields [5]; hatching lines are often guided by the principal curvature directions for the non-photorealistic rendering of pen-and-ink drawings.

In this paper, we focus on vector—or flow—visualization techniques that compute the motion of massless particles advected along the velocity field to obtain characteristic structures like streamlines or streak-

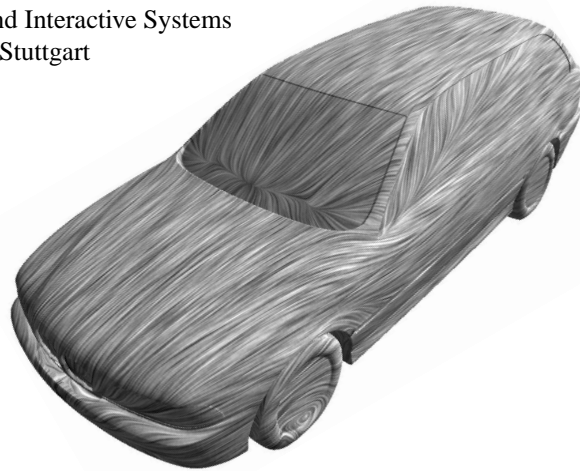


Figure 1: Flow visualization on a curved surface for an automotive CFD simulation.

lines. The fundamental problem of finding appropriate initial seed points can be overcome by a dense representation, i.e., by densely covering the domain with particle traces. This approach gives good results for planar 2D domains, but has intrinsic problems of clutter and occlusion in 3D [8]. Flow visualization on curved 2D hypersurfaces through the complete 3D data set, however, avoids most of the occlusion problems and provides a more flexible representation than on planar slices. Interesting fluid behavior often occurs in the direct neighborhood of a curved boundary and can be displayed by this 2.5D approach. A typical example is the visualization of the air flow around an aircraft wing or an automobile (see Figure 1).

Texture-based visualization techniques for 2D planar flow can be extended to 2.5D by considering a C-space (computational space) approach. Here, a parameterization of the surface has to be known, and all computations take place in the Cartesian 2D coordinate system. However, many surfaces are not equipped with a parameterization and thus do not allow for a direct application of C-space methods.

In this paper, we rather follow an approach guided by image-space computations. Our technique is inspired and strongly influenced by the recent work by Laramee et al. [11] and Van Wijk [21], who apply texture advection and image-based flow visualization on the image plane.

The image-space approach has a number of important advantages: Neither the parameterization nor the connectivity information of a surface mesh are required; it is well supported by the GPU (Graphics Processing Unit); it does not need sophisticated data structures and is rather simple to implement; the main part of the algorithm—advection and blending—is output-sensitive, i.e., the performance is determined by the viewport size. Unfortunately, the restriction to pure image-space texture advection also causes some disadvantages: Frame-to-frame coherence cannot be guaranteed when the camera position is changed; silhouette lines are inflow regions on the image plane, and it is difficult to maintain constant contrast in these inflow areas; silhouette lines have to be identified by an image-space edge detector; only an exponential filter kernel is supported for Line Integral Convolution.

The idea of this paper is to combine the image-space methods with some aspects of object-space methods. The main contributions are: First, a hybrid Lagrangian particle tracing in physical and device spaces that retains all aforementioned benefits of the image-space approach and avoids its disadvantages. Second, a solid texturing of the input 3D noise which fixes the noise structure in object space and thus guarantees temporal coherence under camera motion while, at the same time, a constant spatial noise frequency is maintained on the image plane. Third, an approach to effectively combine the visualization of surface shape and flow by using appropriate color schemes. Fourth, a mapping of our algorithm to GPUs, which leads to interactive rendering. Finally, we demonstrate how this approach can be used for typical applications in scientific visualization.

2 Previous Work

A large body of research has been published on noise-based and dense vector field visualization. For a comprehensive presentation we refer to the review articles by Sanna et al. [17] and Hauser et al. [6]. Spot noise [19] and Line Integral Convolution (LIC) [2] are early texture-synthesis techniques for dense flow representations, and serve as the basis for many subsequent papers that provide a variety of extensions and improvements to these original methods. Many recent techniques for unsteady 2D flow are based on the closely related concept of texture advection, the basic idea of which is to represent a dense collection of particles in a texture and transport this texture along the vector field [16]. Lagrangian-Eulerian Advection (LEA) [10] visualizes unsteady flows by a Lagrangian integration of particle positions and a Eulerian advection of particle colors. Image Based Flow Visualization (IBFV) [20] is a variant of 2D texture advection in which a second texture is blended into the advected

texture at each time step.

Since dense representations need a large number of computations, graphics hardware can often be exploited to increase visualization performance. For example, GPU-based implementations are known for steady flow LIC [7], for different techniques for unsteady 2D flows [9, 20, 23], and for 3D vector fields [13, 18, 24].

The first techniques for dense flow visualization on curved surfaces were based on 2D curvilinear grids, which directly provide a parameterization that can be used for particle tracing in C-space. For example, LIC can be applied in this fashion on curvilinear grids [3]. The effects of different cell sizes in such grids can be compensated by using multi-granularity input noise [14]. An arbitrary surface can always be approximated by triangulation and, therefore, visualization on generic curved surfaces usually relies on a triangle mesh representation. Battke et al. [1] describe a steady flow LIC technique for triangle surfaces that uses the mesh connectivity to trace a particle path from one triangle to an adjacent one and that builds a packing of triangles into texture memory. Mao et al. [15] avoid this triangle packing by tightly connecting the LIC computation in texture space with the view-dependent rendering of the mesh; however, they also need the connectivity information for particle tracing. As already mentioned in the introductory section, Van Wijk [21] and Laramee et al. [11] propose an image-space approach for texture advection on curved surfaces which has directly influenced the development of our visualization technique.

3 Lagrangian Particle Tracing on Surfaces

In this paper, a Lagrangian approach to particle tracing is adopted. Each single particle can be identified individually and the properties of each particle depend on time t . The path of a single massless particle is determined by the ordinary differential equation

$$\frac{d\mathbf{r}(t)}{dt} = \mathbf{u}(\mathbf{r}(t), t) \quad , \quad (1)$$

where $\mathbf{r}(t)$ describes the position of the particle at time t and $\mathbf{u}(\mathbf{r}, t)$ denotes the time-dependent vector field. In the case of a flat 2D or 3D domain, the points or vectors (marked as boldface letters) are 2D or 3D, respectively.

In this paper, we focus on domains that can be represented as 2D curved manifolds embedded in flat 3D space. Curved n D manifolds are often described by an atlas, which is the collection of charts; a chart consists of a coordinate system that is a subset of the Cartesian space \mathbb{R}^n , and of a mapping from the manifold to the coordinate system. Finding an appropriate set of charts—a set of parameterizations—for an arbitrary 2D manifold is

a formidable and expensive task that is part of ongoing research (see, for example, the papers [5, 12]). We do not want to assume or construct a parameterization and therefore choose another representation for the embedded manifold: just the set of points \mathbf{r} in \mathbb{R}^3 that belong to the surface. To build a tangential vector field, the normal component (along the normal direction of the hypersurface) of the 3D vector \mathbf{u} has to vanish. A tangential vector field can either come from a direct construction, such as a fluid flow computation on a surface, or from the projection of a non-tangential 3D vector field. For a tangential vector field, Eq. (1) leads to curves that stay on the surface.

So far, the vector and point quantities were given with respect to physical space (P-space). Along the rendering pipeline of computer graphics, however, a large number of additional coordinate systems is used. In GPU-based interactive rendering, a vertex of the input geometry is usually transformed from its original object coordinates into subsequent world, eye, and clip coordinates by respective affine or projective transformations. After rasterization, a fragment undergoes the transformation from clip space into normalized device coordinates, via a homogeneous division by the w clip coordinate. In this paper, the view frustum is assumed to be $[0, 1]^3$ in normalized device space (D-space).

In this notation, P-space coordinates are identical to their object coordinates. For a stationary surface geometry, P-space and world coordinates are related by a constant affine transformation that is described by the model matrix. On the one hand, a P-space approach is well-suited for representing everything that should be fixed with respect to object coordinates: For example, the noise input for LIC-like convolution (more details on specifying such a noise texture are given in Section 4) should be modeled in P-space; in this way, a frame-to-frame coherent display of the input noise is automatically achieved even when the virtual camera is moved. On the other hand, D-space is the natural representation for everything that is directly related to, or governed by aspects of, the image plane. For example, a dense vector field visualization is ideally computed on a per-pixel basis with respect to the image plane in order to achieve an output-sensitive algorithm and a uniform density on the image plane.

The basic idea of our algorithm is to combine the advantages of P-space and D-space representations by computing particle paths in both spaces simultaneously and tightly connecting these two points of view. Figure 2 illustrates the coupled P/D-space approach. The starting point is Eq. (1), which is solved by explicit numerical integration. A first-order explicit Euler scheme is employed in our current implementation, but other, higher-order

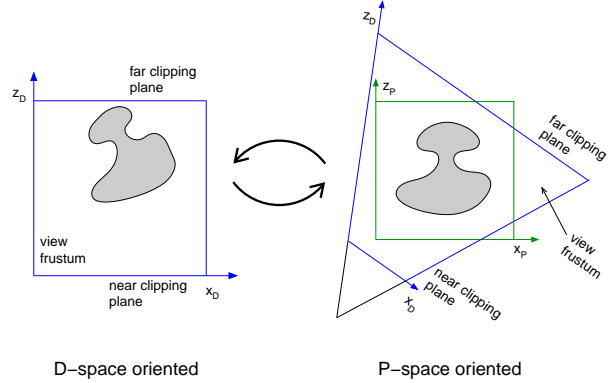


Figure 2: Coupled D-space (left) and P-space (right) representations of the same scene. The y axes are left out to simplify the illustration.

schemes (e.g., Runge-Kutta) could be used as well. The numerical solver works with P-space coordinates $\mathbf{r}_P \equiv \mathbf{r}$ and the original tangential vectors $\mathbf{u}_P \equiv \mathbf{u}$. After each integration step, the corresponding position in D-space is computed by applying the model, view, projection, and homogeneous division operations, T_{MVP} :

$$\mathbf{r}_D = T_{MVP}(\mathbf{r}_P) \quad . \quad (2)$$

An important point is that the vector field is no longer given on a P-space but a D-space domain, i.e., we have different representations for the vector components and the associated point on the surface. This alternative description with respect to D-space is given by $\mathbf{v}_P(\mathbf{r}_D, t) = \mathbf{u}_P(T_{MVP}^{-1}(\mathbf{r}_D), t)$. The differential equation then becomes

$$\frac{d\mathbf{r}_P(t)}{dt} = \mathbf{v}_P(\mathbf{r}_D(t), t) \quad , \quad (3)$$

in combination with Eq. (2).

The crucial step in making the integration process efficient is to reduce the 3D representation of the quantities that depend on \mathbf{r}_D to a 2D representation with respect to the x_D and y_D components of \mathbf{r}_D . Since the flow fields are assumed to live on opaque surfaces, only the closest surface layer is considered and the z_D component of \mathbf{r}_D (i.e., the depth component) can be neglected.

The goal of all LIC-oriented techniques is to generate a texture I by starting particle traces at each texel and computing the convolution integral along these traces. This texture can be parameterized by 2D device coordinates because only the visible surface parts are of interest. Since each particle trace can be uniquely identified by its corresponding seed point at the starting time t_0 , the P-space positions along a path can be labeled by the x and y device coordinates of the seed point (x_D^0, y_D^0) : $\mathbf{r}_P(t - t_0; x_D^0, y_D^0)$. Finally, the vector field with respect to

```

// Initialize the 2D texture for  $\mathbf{v}_P(\mathbf{r}_D)$ :
while rendering the surface with attached vector field:
  make vectors tangential
  transform vectors to device coordinates
  → write result into  $\text{tex2D\_v}_P(x_D^0, y_D^0)$ 

// Set the 2D texture for the initial positions  $\mathbf{r}_P$ :
render surface and write  $\mathbf{r}_P$  into 2D texture  $\text{tex2D\_r}_P(x_D^0, y_D^0)$ 

// Initialize the convolution texture:
 $\text{tex2D\_I}(x_D^0, y_D^0) \leftarrow 0$ 

// Iterate explicit solver until maximum length is reached:
for i=1 to imax
  for each visible texel  $(x_D^0, y_D^0)$ :
    // Transform from P-space to D-space
     $(x_D, y_D) \leftarrow T_{\text{MVP}}(\text{tex2D\_r}_P(x_D^0, y_D^0))$ 
    // Single Euler integration step:
     $\text{tex2D\_r}_P(x_D^0, y_D^0) \leftarrow \text{tex2D\_r}_P(x_D^0, y_D^0)$ 
       $+ \Delta t \text{ tex2D\_v}_P(x_D, y_D)$ 
    accumulate convolution in  $\text{tex2D\_I}(x_D^0, y_D^0)$ , based on
      P-space noise  $\text{tex3D\_noise}(\text{tex2D\_r}_P(x_D^0, y_D^0))$ 
  endfor
endfor

```

Figure 3: Pseudo code for the iterative computation of surface particle paths.

2D device coordinates, $\mathbf{v}_P(x_D^0, y_D^0, t)$, can be computed by projecting the surface geometry onto the image plane. In summary, all quantities that are needed to solve Eq. (3) can be stored in 2D textures with respect to the texel positions (x_D^0, y_D^0) .

Figure 3 shows the complete pseudo code for texture-based Lagrangian particle tracing on surfaces. The algorithm is split in two major parts. In the first part, the 2D textures for \mathbf{r}_P and \mathbf{v}_P are initialized by rendering the mesh representation of the hypersurface. The closest depth layer is extracted by the z test. The P-space positions are set according to the surface’s object coordinates (i.e., the standard vertex coordinates); interpolation during scanline conversion provides the positions within the triangles. Similarly, the vector field texture is filled by \mathbf{v}_P , which either comes from slicing through a 3D vector field texture or from vector data attached to the vertices of the surface. If the vector field is not tangential from construction, it is made tangential by removing the normal component, which is computed according to the normal vectors of the surface mesh. In the second part, Eq. (3) is solved by iterating Euler integration steps. This part works on the 2D sub-domain of D-space; it successively updates the coordinate texture \mathbf{r}_P along the particle traces, while simultaneously accumulating the contribution of the convolution integral in texture I . A contribution from backward particle traces can be computed by additionally executing the complete algorithm with negated step size.

The algorithm in Figure 3 supports only steady vector fields. In the case of a time-dependent flow, the projection

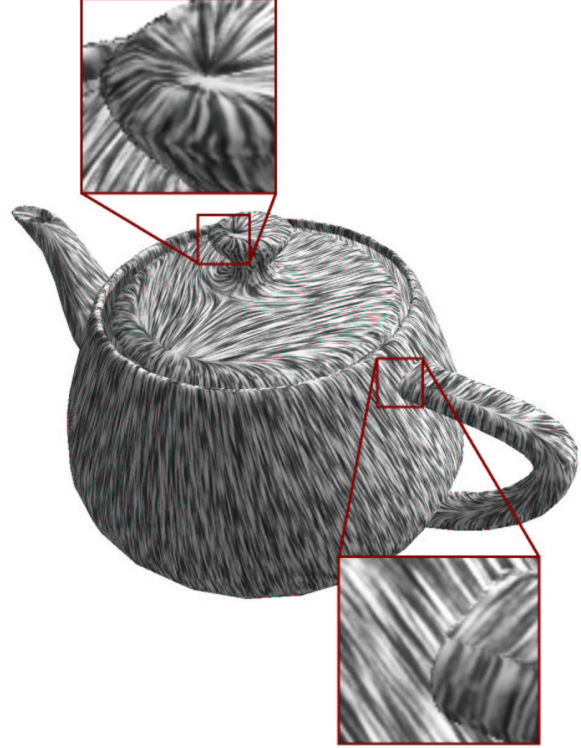


Figure 4: Demonstrating the treatment of silhouette lines and object boundaries for surface flow on a teapot.

of the vector data has to be performed within the integration loop. Note that the details of the noise accumulation step are covered in the following section.

By restricting the device coordinates to the image plane during the texture lookup in the 2D vector field, we essentially imitate the behavior of the image-space techniques [11, 21], i.e., the velocity values are evaluated only at the surface and not at arbitrary 3D points. On the other hand, the positions in P-space are not projected onto the hypersurface. This is one crucial difference to the image-space methods. In this way, we avoid the inflow boundaries that are generated by silhouette lines on the image plane and that cause some of the artifacts in [11, 21]. Moreover, the P-space representation provides a better numerical representation of positions in the vicinity of silhouette lines by taking into account three coordinates instead of only two. Finally, the full 3D representation of P-space points allows us to distinguish two objects that touch each other in image space, but are at different depths. Therefore, it is automatically guaranteed that flow structures do not extend across object boundaries. Figure 4 shows that the flow structure is correctly generated at silhouette lines and boundaries. Even discontinuities of the tangential vector field are well represented. Figure 5 illustrates a uniform vertical 3D flow projected

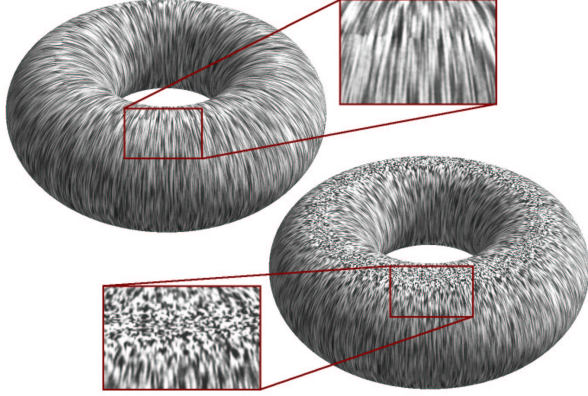


Figure 5: Vertical flow projected onto two tori: LIC visualization (left) vs. representation of the magnitude of the vectors (right).

onto two tori. The LIC approach (left image) normalizes the projected vector field to unit length and therefore produces a discontinuity of the flow on the top and bottom rings of the torus. Even these problematic parts can be treated by our approach. The right image in Figure 5 shows the same vector field without normalization to unit length. The streaks are shorter at the top of the torus due to the small magnitude of the tangential vector field. We recommend to view the accompanying electronic videos on our web page [22] because the frame-to-frame coherence provided by our approach becomes apparent only in animations.

4 Visual Mapping and Noise Injection

So far, the focus has been on the computation of particle paths. But how can these paths serve as the basis for generating effective images?

For dense flow visualization, we adopt the idea of LIC on 2D planar surfaces [2] to produce patterns of different colors or gray-scale values. The basis is a noise input image. By computing the convolution along the characteristic curves, high correlation is achieved along the lines, but no or only little correlation perpendicular to the lines. Generalizing the LIC idea to 3D and taking into account time-dependent noise input similarly to IBFV [20], we obtain

$$I(x_D^0, y_D^0) = \int_{-\infty}^{\infty} k(t - t_0) N(\mathbf{r}_P(t - t_0; x_D^0, y_D^0), t) dt, \quad (4)$$

where $\mathbf{r}_P(t - t_0; x_D^0, y_D^0)$ describes the particle path starting at point (x_D^0, y_D^0) at time t_0 , k is the filter kernel, and $N(\mathbf{r}_P, t)$ is the time-dependent 3D noise input. After dis-

cretizing the integral, we obtain

$$I(x_D^0, y_D^0) = \sum_i k^i N^i(\mathbf{r}_P^i(x_D^0, y_D^0)) \Delta t, \quad (5)$$

where the superscripts i indicate the time dependency. Since procedural noise is (not yet) supported by GPUs, we model all textures as sampled textures. In a naive implementation, the spacetime noise N^i with its four dimensions would require a large amount of texture memory. By adopting the time representation from IBFV [20], the temporal dimension is replaced by a random temporal phase per texel. Furthermore, the memory for the remaining three spatial dimensions can be reduced by periodically repeating the 3D texture along the main axes (i.e., texture wrapping). From experience, noise texture sizes of 64^3 or 128^3 are appropriate for typical applications. Repeating noise structures may become apparent only for planar slices—as long as the surface is at least slightly curved, texture wrapping leads to good results.

However, one problem is introduced by our solid noise texture: A constant spatial frequency in physical space leads to different frequencies after perspective projection to image space. We overcome associated aliasing issues by appropriately scaling the noise texture in physical space and thus compensating the effects of perspective projection. Mao et al. [15] use a uniform scaling that is based on the distance of the surface from the camera, which is appropriate for generating a non-animated LIC image of an object with limited depth range. We add the following two features to allow for temporally coherent, animated visualizations of large objects. First, the scaling factor is computed for each particle seed point independently to ensure a constant image-space frequency for objects with large depth ranges. Second, the noise is only scaled with discretized scaling factors and snapped to fixed positions to achieve frame-to-frame coherence under camera motions.

We assume a model in which noise with all spatial frequencies, \tilde{N} , is constructed by the sum of band-pass filtered noise:

$$\tilde{N}(\mathbf{r}) = \sum_{i=0}^{\infty} N_{\text{band}}(2^i \mathbf{r}), \quad (6)$$

where N_{band} serves as role model for the noise and the multiplication by 2^i results an increase of the spatial frequencies by 2^i . When a band-pass filter is applied to \tilde{N} , low and high frequencies are removed and therefore the infinite sum is reduced to a finite sum of $N_{\text{band}}(2^i \mathbf{r})$ terms. As an approximation, we assume a rather narrow band filter and consider only the linear superposition of two

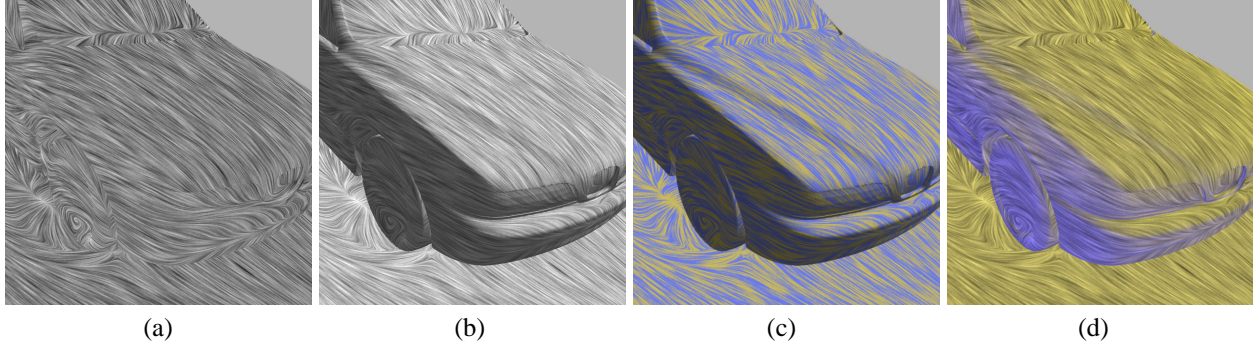


Figure 6: Different shading approaches for object and flow rendering: (a) gray-scale flow visualization without additional surface shading, (b) gray-scale flow modulated with illuminated gray object, (c) yellow/blue flow visualization modulated with illuminated gray object, and (d) gray-scale flow modulated with cool/warm shaded surface.

neighboring frequency bands,

$$N_{\text{filtered}}(\mathbf{r}) = \sum_{i=i_{\text{scale}}}^{i_{\text{scale}}+1} \alpha_i N_{\text{band}}(2^i \mathbf{r}) \quad . \quad (7)$$

The two discrete scaling factors are determined by i_{scale} , which is computed from the distance d between particle seed point and camera (by using $\log_2 d$). The weights α_i provide a linear interpolation based on the fractional part of $\log_2 d$. This model can directly be realized by sampled textures in combination with a fragment program: A single copy of N_{band} is stored as 3D texture, the measure i_{scale} is computed for each seed point, two texture lookups with scalings $2^{i_{\text{scale}}}$ and $2^{(i_{\text{scale}}+1)}$ are performed, and finally these values are linearly interpolated.

Since the noise frequency spectrum is variable in P-space, the step size of Lagrangian integration should be as well. We change the step size in a similar way as for the noise. In this case, however, we do not need the snapping to discrete scaling factors, but can directly use the distance d as measure.

5 Effective Object and Flow Rendering

For the final rendering, both the LIC-like texture that represents the flow, and the shape and orientation of the hypersurface should be taken into account. That is, two types of information need to be visualized at the same time: object shape and flow structure. Figure 6 (a) demonstrates that the geometry of the surface is extremely hard to recognize if the flow texture is projected onto the image plane without any modifications.

Since we are restricted to displaying the image on a 2D image plane, visual cues are essential to allow the user to recognize shape and structure. We think that colors play a crucial role because shape recognition heavily depends on the color variations introduced by illumination. Figure 6 (b) shows a straightforward way of

combining colors from the illumination with the structure from the flow texture: Gray-scale values are computed according to the diffuse illumination of the gray object surface and then modulated (i.e., multiplied) with the gray-scale values from the LIC-like texture. This image gives a good impression of both flow and object shape. However, both aspects are coded only by luminance variations and, therefore, other color dimensions are not used. Based on the tristimulus theory, colors are given with respect to a three-dimensional space. According to perception-oriented color systems, luminance, hue, and saturation can be distinguished as three dimensions. From this point of view, the approach of Figure 6 (b) completely neglects hue and saturation. Therefore, we propose an alternative method: The LIC-like texture is first mapped from gray-scale variations to either hue or saturation variations. Afterwards, the transformed flow texture is modulated with the illuminated gray surface. Figure 6 (c) demonstrates a mapping to hue variations between blue and yellow. From our experience, we prefer hue gradients because they result in a better visual contrast than saturation gradients.

Building on the idea of using different color dimensions for coding flow and shape structure, we propose another, alternative technique in which the roles of hue and luminance are exchanged. Here, the LIC-like texture is represented by luminance variations and the surface by hue variations. Cool/warm shading [4] is a well-established hue-based illumination model that leads to an intuitive recognition of shape. In Figure 6 (d), cool/warm shading with yellowish and bluish colors is modulated with the gray-scale LIC-like texture.

We think that the different approaches from Figure 6 (b)–(d) have specific advantages and disadvantages. A pure gray-scale representation (b) allows us to code additional properties in the hue and saturation channels and thus is useful for multivariate or multi-field visualization.

Furthermore, gray-scale images are, of course, a good basis for black-and-white printing. Since luminance variations are a strong visual cue for shape recognition, the technique from Figure 6 (c) provides a good impression of the geometry. On the other hand, the flow texture is hard to see in dimly illuminated parts of the surface. Finally, cool/warm surface shading is excellent in showing the flow structure in all visible surface regions, but gives a weaker impression of the surface geometry.

6 Implementation

Our implementation is based on C++ and DirectX 9.0, and was tested on a Windows XP machine with an ATI Radeon 9800 Pro GPU (256 MB). GPU states and programs (i.e., vertex and pixel shader programs) are configured within effect files. A change of this configuration can be included by changing the clear-text effect files, without recompiling the C++ code. All shader programs are formulated with high-level shading language (HLSL) to achieve a code that is easy to read and maintain. Since the descriptions of Lagrangian integration in Section 3 and noise representation in Section 4 are already based on textures, most parts of our visualization approach can be readily mapped to the functionality of a DirectX 9.0 compliant GPU. A comparable implementation should be feasible with OpenGL and its vertex and fragment program support.

An advantage of our approach is that most operations take place on a texel-by-texel level. This essentially reduces the role of the surrounding C++ program to allocating memory for the required textures and executing the pixel shader programs by drawing a single domain-filling quadrilateral. The early z test allows us to skip the pixel shader programs for the pixels that are not covered by the projection of the surface onto the image plane. Textures are updated by using the render-to-texture functionality of DirectX. As an example, the HLSL code for a single Lagrangian particle integration step is given in Figure 7, which demonstrates that Eqs. (2) and (3) can be readily transferred into a corresponding pixel shader program. Similarly, the other elements of the algorithm can be mapped to shader programs in a direct way. Additional technical information and HLSL codes can be found on our web page [22].

Some of the textures hold data that is replaced by new

Table 1: Performance measurements in fps.

| Domain size | 600 ² | | 950 ² | |
|-------------|------------------|------|------------------|-----|
| | 25 | 70 | 25 | 70 |
| Teapot | 26.2 | 10.1 | 13.1 | 5.1 |
| Automobile | 12.6 | 5.7 | 8.0 | 3.4 |

```

// Parameters:
float4x4 matMVP; // Model-view-projection matrix
float4 stepSize; // Integration step size

struct VS_Output { // Tex coords for viewport-filling quad
    float2 TexCoord: TEXCOORD0;
};

struct PS_Output { // Homogeneous P-space position
    float4 RGBA : COLOR0;
};

// High-level pixel shader program
PS_Output IntegratePS (VS_Output In) {
    PS_Output Output;

    // Lookup current P-space position in texture PosTex
    float4 posP = tex2D(PosTex, In.TexCoord);

    // One integration step, split into three parts:
    // (1) Transform from P-space into D-space
    // Applies model-view-projection matrix:
    float4 posD = mul(posP, matMVP);
    posD = posD / posD.w; // Homogeneous division
    posD.x = .5 * posD.x + .5; // Maps x from [-1,1] to [0,1]
    posD.y = -.5 * posD.y + .5; // Maps y from [1,-1] to [0,1]
    // (2) Get P-space velocity from the 2D texture
    float3 velocity = tex2D(FlowField2dTex, (float2) posD);
    // (3) Compute new P-space position with Euler integration
    float3 posNew = posP + velocity * stepSize;

    // Output new position in homogeneous P-space coordinates
    Output.RGBA = float4 (posNew, 1.0);

    return Output;
}

```

Figure 7: High-level pixel shader code for one integration step in hybrid P/D-space particle tracing.

values during each iteration step. For example, the P-space coordinates are updated, based on the current position. For such a texture, ping-pong rendering is applied: Two copies of the texture are used, one as the destination texture (i.e., the render target) and the other one as the data source. The roles of the two textures are exchanged after each iteration step. To limit the required texture memory and make full use of the internal memory bandwidth of the GPU, the depth of the color channels in the textures should be reduced to the smallest possible level. From our experience, we find the following choices appropriate: 32 bit floating-point resolution for P-space positions, 16 bit fixed-point numbers for the accumulated noise values, 16 bit fixed-point or floating-point numbers for the flow data, and 8 bit fixed-point numbers for the input noise and random phases.

Performance measurements for the aforementioned hardware configuration are shown in Table 1. Filter length describes the number of integration steps to compute the Line Integral Convolution. The teapot scene from Figure 4 is an example for a surface with a low polygon count, while the automobile scene from Figures 1 and 6 with its 1 143 796 triangles and 597 069 vertices

represents a realistic industry data set. Please note that the performance numbers for the two scenes should not directly be compared with each other because the two scenes cover different ratios of the domain; the car almost completely fills the viewport, while the teapot covers a smaller percentage. However, the numbers for each scene show that the complexity of the surface geometry plays only a small role and that the performance depends on the domain size and filter length in a nearly linear fashion. Therefore, our implementation allows the user to balance speed against visualization quality by gradually changing the filter length and/or the viewport size.

7 Conclusion and Future Work

We have presented a novel approach for texture-based visualization of vector fields on curved surfaces. The texture advection mechanism relies on a simultaneous Lagrangian particle tracing in physical and device spaces. Its built-in key features are: a high degree of output sensitivity, independence from surface parameterization and mesh connectivity, frame-to-frame coherence in animations, and a simple and efficient implementation on GPUs. Noise input for texture advection is modeled as a solid 3D texture, and a constant noise frequency on the image plane is achieved by appropriately scaling the noise in physical space. Finally, we have presented a number of alternatives to effectively convey surface shape and flow structure in the same image.

In future work, it will be interesting and beneficial to introduce dye advection in our concept of texture advection. However, this will be a challenging task because dye needs a long history of its path to be stored, which might require to extend the 2D representation beyond the closest surface.

Acknowledgements

We thank the anonymous reviewers for helpful remarks to improve the paper. The data set that was used for Figures 1 and 6 was provided by the BMW Group. Special thanks to Martin Schulz for his help. The first author acknowledges support from the Landesstiftung Baden-Württemberg.

References

- [1] H. Battke, D. Stalling, and H.-C. Hege. Fast Line Integral Convolution for arbitrary surfaces in 3D. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics*, pages 181–195. Springer, 1997.
- [2] B. Cabral and L. C. Leedom. Imaging vector fields using Line Integral Convolution. In *Proceedings of ACM SIGGRAPH 93*, pages 263–272, 1993.
- [3] 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, 1995.
- [4] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH 1998 Conference Proceedings*, pages 101–108, 1998.
- [5] G. Gorla, V. Interrante, and G. Sapiro. Texture synthesis for 3D shape representation. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):512–524, 2003.
- [6] H. Hauser, R. S. Laramee, and H. Doleisch. State-of-the-art report 2002 in flow visualization. TR-VRVis-2002-003, VRVis, 2002.
- [7] 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, 1999.
- [8] V. Interrante and C. Grosch. Strategies for effectively visualizing 3D flow with volume LIC. In *IEEE Visualization '97*, pages 421–424, 1997.
- [9] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Hardware-accelerated texture advection for unsteady flow visualization. In *IEEE Visualization '00*, pages 155–162, 2000.
- [10] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):211–222, 2002.
- [11] R. S. Laramee, B. Jobard, and H. Hauser. Image space based visualization of unsteady flow on surfaces. In *IEEE Visualization '03*, pages 131–138, 2003.
- [12] 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.
- [13] G. S. Li, U. Bordoloi, and H. W. Shen. Chameleon: An interactive texture-based framework for visualizing three-dimensional vector fields. In *IEEE Visualization '03*, pages 241–248, 2003.
- [14] X. Mao, L. Hong, A. Kaufman, N. Fujita, and M. Kikukawa. Multi-granularity noise for curvilinear grid LIC. In *Graphics Interface*, pages 193–200, 1998.
- [15] X. Mao, M. Kikukawa, N. Fujita, and A. Imamiya. Line Integral Convolution for 3D surfaces. In *EG Workshop on Visualization '97*, pages 57–70, 1997.
- [16] 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.
- [17] A. Sanna, B. Montrucchio, and P. Montuschi. A survey on visualization of vector fields by texture-based methods. *Recent Res. Devel. Pattern Rec.*, 1:13–27, 2000.
- [18] A. Telea and J. J. van Wijk. 3D IBFV: Hardware-accelerated 3D flow visualization. In *IEEE Visualization '03*, pages 233–240, 2003.
- [19] J. J. van Wijk. Spot noise – texture synthesis for data visualization. *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, 25(4):309–318, 1991.
- [20] J. J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002.
- [21] J. J. van Wijk. Image based flow visualization for curved surfaces. In *IEEE Visualization '03*, pages 123–130, 2003.
- [22] D. Weiskopf. Texture-based flow visualization, 2004. <http://www.vis.uni-stuttgart.de/textflowvis>.
- [23] D. Weiskopf, G. Erlebacher, and T. Ertl. A texture-based framework for spacetime-coherent visualization of time-dependent vector fields. In *IEEE Visualization '03*, pages 107–114, 2003.
- [24] D. Weiskopf and T. Ertl. GPU-based 3D texture advection for the visualization of unsteady flow fields. In *WSCG 2004 Short Papers*, pages 259–266, 2004.