DOI: 10.1111/j.1467-8659.2012.03082.x
Eurographics Conference on Visualization (EuroVis) 2012
S. Bruckner, S. Miksch, and H. Pfister
(Guest Editors)

Volume 31 (2012), Number 3

Multi-layer illustrative dense flow visualization

R. Carnecky, B. Schindler, R. Fuchs and R. Peikert

ETH Zurich, Switzerland

Abstract

We present a dense visualization of vector fields on multi-layered surfaces. The method is based on the illustration buffer, which provides a screen space representation of the surface, where each pixel stores a list of all surface layers. This representation is implemented on the GPU using shaders and leads to a fast, output sensitive technique. In our approach, we first use procedural noise to create an initial spot pattern on the surface that has both an almost constant screen space frequency and is view independent. Then, we perform anisotropic diffusion simultaneously on all surface layers using a discretization scheme that maintains second order convergence while only accessing the four neighboring pixels. Finally, we enhance this result with illustrative techniques and composite the final image. Our method works with time-evolving surfaces, time-dependent vector fields, and moving cameras. We apply our method to CFD data sets from engineering and astronomy as well as synthetic velocity fields.

1. Introduction

Computational fluid dynamics engineers are regularly faced with the problem of analyzing complex three-dimensional flows. Visualizing such flows is difficult, since reducing the entire flow volume to a flat image inevitably leads to occlusion and clutter. Fortunately, the visualization can often be restricted to the flow along special surfaces of interest. These can include physical object boundaries, but also virtual surfaces, such as stream surfaces, pressure isosurfaces, or Lagrangian coherent structures (LCS) [Hal01]. But even if we restrict the problem to visualizing flow on surfaces instead of volumes, occlusion has to be handled carefully. Twists and folds induced by turbulent flow will almost always lead to the surface having multiple self-occluding layers in the image.

One approach to solve this problem is to use cutaways and focus the visualization of the flow to a specific layer or part of the surface. This, however, requires tedious user interaction for moving the cutaway primitives or manipulating degree-of-interest functions. More importantly, cutting away large parts reduces the usefulness of the method for an explorative analysis of the data set without a priori knowledge of what the user is interested in.

Another approach is to sparsely place arrow glyphs on the surface, indicating the local flow direction. If the glyphs are sparse enough and the surfaces transparent, all layers will be visible, but the glyphs will cause difficulties with depth perception. Additionally, finding an optimal placement for the glyphs is not trivial and even then, important features in between the glyphs may be missed. For $2D$ flows and single layer surfaces, this problem has been solved by dense vector field visualization. These methods usually start with a random noise image and "smear" it along the flow, resulting in the surface being densely covered by streaks aligned to the local flow direction. Several of these methods could be used for multi-layered surfaces, but require a surface parametrization or the knowledge of the entire vector field. In particular, the first requirement is not always given: LCS surfaces are often topologically complex and non-orientable, and a global parametrization might not always exist.

We propose a novel dense visualization method that addresses the aforementioned shortcomings. The method is designed for visualizing flows along multi-layered surfaces, where both the shape of the surface and the flow direction at all layers is important. In contrast to previous work, it only requires as input a triangle mesh with the vector field values specified at the mesh vertices. The mesh does not need to have a parametrization and both the mesh and the vector field can undergo arbitrary changes over time. This format is suitable for portable data sets and readily available in many visualization frameworks. Moreover, the method is output sensitive, i.e., its run time depends mainly on the size and quality of the output. Additionally, it is not limited to a fixed view point and produces patterns of a constant screen space frequency, regardless of the current view or zooming factor.

2. Background and related work

Dense and texture-based flow visualization

One of the major categories of numerical flow visualization methods are the dense and texture-based methods [LHD*04]. Methods falling into this category are on one hand techniques that generate anisotropic noise textures indicating the flow direction and on the other hand texture advection methods, where a moving texture is used to visualize the flow. Both approaches can be applied to steady and unsteady flow, but for performance reasons, anisotropic noise textures are typically used for steady flow, while for unsteady flow, a combination of the two approaches is most often used. Anisotropic noise textures can be produced with a number of techniques, including line integral convolution (LIC) [CL93] and anisotropic diffusion [DPR00].

The goal of our method is a 2.5D dense flow visualization. Unfortunately most dense flow visualization methods work essentially in 2D and an extension to multi-layered surfaces is not straightforward. One option would be to cover the surface with a texture and compute the visualization in texture (parameter) space. However, this requires a parametrization of the surface. A second option would be to compute the visualization in physical (object) space. This leads to a limited resolution, especially during close up views. A third option would be to compute the visualization in image (screen) space. This approach is taken by our method. Note that this requires a special handling of multiple layers in the image. Some common approaches such as depth peeling do not work, as explained in Section 3.1.

Line integral convolution

The idea of the LIC method is to visualize a 2D vector field by covering the domain with a white noise texture image and filtering it using 1D kernels that follow the streamlines. The first publication by Cabral and Leedom [CL93] was followed by numerous later research providing extensions and improvements. FastLIC, by Stalling and Hege [SH95], reuses results of streamline integration in order to cut computational costs. UFLIC, by Shen and Kao [SK97], is an extension to unsteady flow. For temporal coherence, a dense set of particles is used which carry the texture information and which are advected with the flow. Liu and Moorhead presented AUFLIC [LM02], an accelerated version, based on the idea of FastLIC. Li et al. [LTH06] reformulated the UFLIC algorithm for an efficient GPU implementation. Since all of the methods based on UFLIC are 2D methods, they cannot be easily applied to 2.5D visualizations. A different approach is taken by Battke et al. [BSH97]. By packing triangles individually into textures and computing the LIC on each triangle separately in object space, their method can handle arbitrary surfaces. Unlike our method however, the frequency of the LIC pattern is not constant in screen space and suffers from a limited texture resolution, especially during close up views.

Anisotropic diffusion

Nonlinear diffusion has been widely used in image processing to selectively smooth image parts while maintaining sharp feature edges. An overview of diffusion in image processing is given by Weickert [Wei97]. Diewald et al. [DPR00] apply anisotropic diffusion to the dense visualization of flow fields. They also extend the diffusion to arbitrary surfaces by using a fine mesh as the underlying discretization and computing the diffusion in object space. While our method uses a similar diffusion equation, it works in screen space and is therefore not affected by the resolution of the mesh. A different diffusion-reaction equation is used by Sanderson et al. [SJK04] to create a dense set of stream-aligned and oriented spots. The discretization of the diffusion equation can significantly affect the quality of the result. Scharr and Weickert [SW00] present a fast explicit second-order scheme that uses two staggered 3x3 stencils, which is very similar to the discretization used in our method.

Texture advection

The first flow visualization based on texture advection was done by Max et al. in 1992 [MCW92]. The LEA (Lagrangian-Eulerian advection) algorithm by Jobard et al. [JEH02] is a hybrid scheme where texture advection is combined with particle advection for higher accuracy. Van Wijk proposed image-based flow visualization (IBFV) [vW02], a fast and simple texture advection method for generating streaklines. Unsteady flow advection-convolution (UFAC), by Weiskopf et al. [WEE03], is a texture advection method combined with a LIC-like spatial filtering. For velocity fields on curved surfaces, a variant of IBFV was proposed by van Wijk [vW03]. Image-space advection (ISA) by Laramee et al. [LJH03] was the first screen-space method for texture advection on surfaces. This screen-space approach is well suited for a GPU implementation [GL05], but it cannot correctly handle moving cameras, because the texture is associated with screen pixels, not surface primitives as in van Wijk's method. Weiskopf and Ertl [WE04] solved this problem by using 3D noise textures. Our initial noise generation is based on their method, except that we use procedural noise instead of textures.

Multiple layers and illustrative techniques

Despite the advanced state of the art, none of the above mentioned texture-based flow visualization methods has been extended to transparent multi-layer surfaces. Extensions for multiple layers exist, however, for other categories, in particular the geometric and integration-based methods. A general approach to improve perception of curved surfaces, by Interrante et al. [IFP97], is based on adding strokes indicating principal curvature directions. Krishnan et al. [KGJ09] propose a rendering of streak surfaces using strong transparency in combination with sparse (streak) lines on the surface.

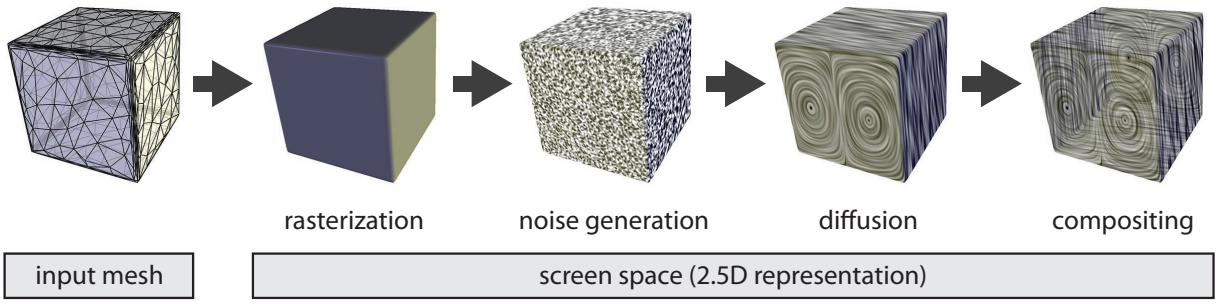

Figure 1: Overview of our method. All steps are performed on a 2.5D screen space representation of the surface.

Hummel et al. [HGH*10] use adaptive transparency based on the surface normal variation to improve the perception of surface shape. Born et al. [BWF*10] use cutaway techniques, combined with surface glyphs, for an illustrative visualization of stream surfaces. The two later techniques are capable of illustrating the local flow direction, however they are limited to naturally parametrized stream surfaces. Urness et al. [UIL*06] describe strategies for the display multiple colocated vector fields by overlaying texture, glyph, and line based visualizations. However, their approach is limited to a small fixed number of 2D vector fields. These techniques are able to render surfaces in a way that makes a good number of layers recognizable.

Noise texture generation

Procedural noise has been introduced to computer graphics by Perlin [Per85] and is commonly known as Perlin noise. Later, the author addressed numerous shortcomings of this function and proposed the simplex noise [Per01], which is used by our method.

3. Our method

Our method targets the visualization of flow along multi-layered surfaces. Our general approach consists of the following steps, illustrated in Figure 1.

1. A screen space representation of the surface is built. In this *illustration buffer*, each pixel stores a linked list of all surface layers at that pixel, along with a number of surface properties and explicit links to neighboring pixels. All subsequent operations are performed on this representation. It is described in Section 3.1.
2. A procedural noise function is used to assign a grayscale color value to each surface point. The function depends on both view-dependent and view-independent properties, such that the noise pattern is coherent under view changes and has a configurable, constant screen space frequency. It is described in Section 3.2.
3. Anisotropic diffusion is performed simultaneously on all surface layers. The diffusion process is guided by the vector field, resulting in the surface being covered by grayscale streaks that are aligned to local streamlines. A suitable finite difference discretization of this process is presented in Section 3.3.
4. The surface color is converted into transparency, and surface layers are composited using alpha blending. This final step is described in Section 3.5.

The illustration buffer is a very flexible data structure and can be used to extend our method with various illustrative effects, as described in Section 3.6.

3.1. Illustration buffer

The *illustration buffer* [CFM*11] is a pixel-based representation of a rendered image that supports a variable and unbounded number of surface layers for each pixel. It is similar to the concept of the A-buffer [Car84] and thanks to recent advancements in graphics hardware, it can be implemented on the GPU. As a complete description of this data structure is beyond the scope of this paper, we will provide a simplified functional description and list its main properties. First, we shall define important terms used in the illustration buffer: let a *pixel* be the smallest addressable image element and a *fragment* the intersection of the rendered surface with a ray going through the pixel center.

Figure 2(a) depicts a cross section of a surface with dashed lines indicating view rays (one for each pixel). The illustration buffer consists of one buffer with one element per pixel, indicated by red boxes in Figure 2(b). Each pixel contains a linked list of surface fragments, indicated by blue boxes. Each surface fragment contains a fixed number of data channels where arbitrary values can be stored for deferred rendering purposes. Additionally, each fragment contains explicit links to its four geodesically neighboring fragments (indicated by horizontal lines between fragments). Note that these neighbor links follow the surface and are not affected by occluding silhouettes.

Figure 2 also shows why a naïve depth peeling approach is not appropriate for our problem. In depth peeling, the front-most fragments for each pixel are extracted and then

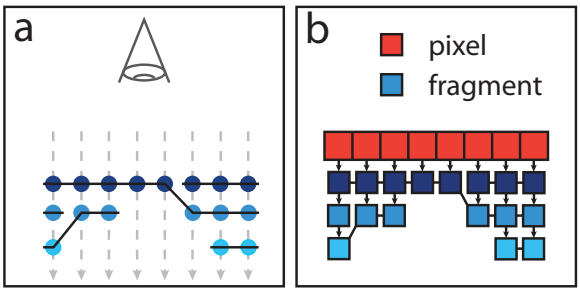

Figure 2: The illustration buffer. (a) A cross section of a surface with three layers. (b) The corresponding buffer layout.

processed as a flat $2D$ image. This is repeated for the next fragments along the view rays until no more fragments are available. These depth peeling layers, illustrated by different shades of blue, contain adjacent fragments from different surfaces, and surfaces may pass through different depth peeling layers. The diffusion process would therefore need to exchange information with other depth peeling layers, effectively creating a $2.5D$ data structure similar to the illustration buffer. Additional disadvantages of depth peeling are that the whole input mesh has to be rendered once per depth peeling layer, and that the number of fragments per pixel is very unevenly distributed, leading to deeper depth peeling layers containing very few fragments.

The illustration buffer is filled in a first step by rendering the surface with a fragment shader that inserts data into the illustration buffer data structure. Both the noise generation and diffusion process are then implemented by rendering a full screen quad with a fragment shader that iterates through all fragments for that pixel. Note that we do not need a special treatment for pixels with multiple fragments. Whenever a fragment at pixel coordinates (i, j) needs to access the data from its neighboring fragment at $(i+1, j)$, it simply follows its right neighbor pointer.

3.2. Noise generation

In order to get an initial noise pattern that is invariant to camera rotation, we want a noise that "sticks to the surface". In our case we use a procedural noise function that maps the world coordinates of each surface point to a noise value $\rho^0 \in [-1, 1]$, using the basic approach

$$\rho^0(\mathbf{x}) = \texttt{snoise3}(s \cdot \mathbf{x}) \tag{1}$$

where `snoise3` is the three-dimensional simplex noise function [Per01] and s a parameter that scales the input coordinates. This function has a number of useful properties: it is reasonably fast, C^2 continuous, visually isotropic, and produces spot noise like patterns, where the size of the spots depends on s. An additional advantage is that it can be extended to arbitrary dimensions. For example, a fourth parameter can be used to include time and create continuously animated noise patterns.

The above approach has the disadvantage that for a constant s, the noise is uniquely determined by the world coordinates of the surface and thus the screen space frequency of the noise pattern changes when zooming in and out. This is undesirable and can lead to aliasing. To fix this, we would like to compute for each pixel the optimal value of s, such that the screen space frequency of the noise is constant. We first compute the ratio of the area of one pixel in screen space to its projected area in world space using partial derivatives of the surface world coordinates, which are provided by the GPU. Then, since the frequency of the local maxima for simplex noise is equal to one per spatial unit, we set s to one times the square root of the inverse area ratio. This value will result in a constant screen space frequency of one white spot per pixel, which can then further be scaled to the desired frequency. Note that this method of computing s reduces the noise frequency for surface parts that are almost parallel to the view direction and thus prevents aliasing of the noise pattern. Depending on the application requirements, other methods for computing s might be more appropriate, such as methods based on the surface distance to the camera.

Unfortunately, using the optimal value for s makes the initial noise highly view dependent and would lead to a noise pattern that moves along the surface as the view is zoomed in or out. In order to retain coherency under changing views, we instead define a fixed set of discrete scaling values s_l with $s_{l+1} = 2s_l$. Then, for each pixel we first compute the optimal value for s, choose the two closest scaling values s_l and s_{l+1}, and linearly blend the two corresponding noise patterns ρ^0_l and ρ^0_{l+1}. This approach is similar to traditional mipmapping and trilinear filtering [Hec83].

When blending two noise images, we should make sure that the result has a similar appearance as a single noise image. Figures 3(a) and 3(b) show the noise patterns for different values of s. By analyzing the histogram of the images, we can see that the noise values for different s follow approximately the same distribution with some unknown variance σ^2. Assuming noise values for different s are unrelated, a linear combination of noise images $\rho^0 = (1-\alpha)\rho^0_l + \alpha\rho^0_{l+1}$ will have the same distribution with a reduced variance $((1-\alpha)^2 + \alpha^2)\sigma^2$. In order to maintain a constant contrast, we scale the blended noise by $1/\sqrt{(1-\alpha)^2 + \alpha^2}$. The effect of this scaling is demonstrated in Figures 3(c) and 3(d).

The whole algorithm for the initial noise is summarized in Algorithm 1, where `log2` and `exp2` are the logarithm and exponential to the base of 2, `dFdx` and `dFdy` derivatives with respect to screen coordinates, and `clamp` a function that clamps its first argument to the range given by the last two arguments. The parameter *spotsize* is the desired approximate diameter of the noise spots, given in screen pixels. Note that the initial noise depends on the world coordinates of each fragment. These coordinates are stored as per-

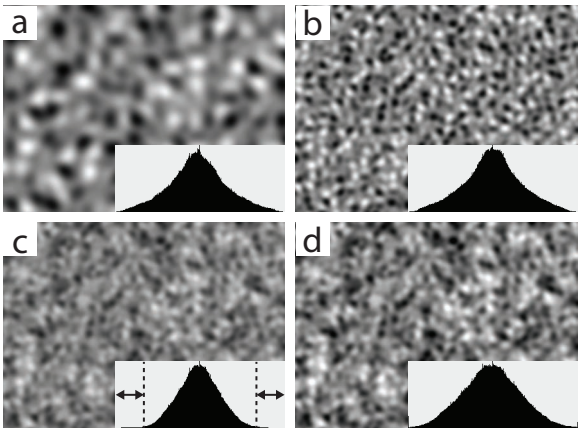

Figure 3: Simplex noise for $\mathbf{x} \in [-1,1]^2$. (a) Noise for $s = 5$. (b) Noise for $s = 10$. (c) The average of the two above images. (d) The average with applied contrast correction.

Algorithm 1 Computing the initial noise

$s \leftarrow 1/(spotsize \cdot \sqrt{||\texttt{dFdx}(\mathbf{x}) \times \texttt{dFdy}(\mathbf{x})||})$
$ls \leftarrow \lfloor \texttt{log2}(s) \rfloor$
$s_1 \leftarrow \texttt{exp2}(ls)$
$s_2 \leftarrow \texttt{exp2}(ls+1)$
$\rho_1 \leftarrow \texttt{snoise3}(s_1\mathbf{x})$
$\rho_2 \leftarrow \texttt{snoise3}(s_2\mathbf{x})$
$\alpha \leftarrow (s-s_1)/(s_2-s_1)$
$c \leftarrow 1/\texttt{sqrt}(\alpha^2+(1-\alpha)^2)$
return $\texttt{clamp}(c((1-\alpha)\rho_1+\alpha\rho_2),0,1)$

fragment data when filling the illustration buffer. All subsequent operations operate in screen space and therefore only use the screen space coordinates of the fragments.

3.3. Anisotropic diffusion

Similar to many approaches in dense vector field visualization, we "smear" the initial noise along the vector field, so that the color of neighboring pixels along a streamline is correlated, while the color of neighboring streamlines is not. Since our 2.5D image representation contains multiple layers and only allows access to neighboring fragments, we formulate our smearing process as an anisotropic diffusion where each fragment only exchanges data with neighboring fragments. Since we are interested in the flow along the surface, we first project the original vector field onto the surface and then transform it to screen space. The resulting vector field $\mathbf{u}$ is used to guide the anisotropic diffusion, given by the partial differential equation

$$\frac{\partial \rho}{\partial t} = \nabla \cdot (\mathbf{A} \nabla \rho) \tag{2}$$

with the initial and boundary conditions

$$\rho(\mathbf{x},0) = \rho^0(\mathbf{x}), \quad \mathbf{A}\nabla\rho \cdot \mathbf{n} = 0 \text{ on } \partial\Omega \tag{3}$$

where $\mathbf{A}$ is a diffusion tensor and $\mathbf{n}$ the screen space normal to the surface silhouette $\partial\Omega$. Expressing the diffusion tensor in terms of diffusivity along and perpendicular to the local vector field leads to

$$\mathbf{A}(\mathbf{u}) = \begin{pmatrix} u & \hat{u} \\ v & \hat{v} \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & \hat{a} \end{pmatrix} \begin{pmatrix} u & v \\ \hat{u} & \hat{v} \end{pmatrix} \tag{4}$$

where $\mathbf{u} = (u,v)^T$ is the local flow direction, $\hat{\mathbf{u}} = (\hat{u},\hat{v})^T$ a vector normal to $\mathbf{u}$, a the diffusivity along the flow direction, and $\hat{a}$ the diffusivity perpendicular to the flow direction. Using the above decomposition, Equation 2 can also be written as

$$\frac{\partial \rho}{\partial t} = \nabla \cdot [a\mathbf{u}(\mathbf{u} \cdot \nabla\rho) + \hat{a}\hat{\mathbf{u}}(\hat{\mathbf{u}} \cdot \nabla\rho)] \tag{5}$$

where $\mathbf{u}(\mathbf{u} \cdot \nabla\rho)$ is the projection of the gradient onto the local vector field direction. In our applications, we set $\hat{a} = 0$ in order to maintain a maximal contrast between streamlines.

We discretize Equation 5 using finite differences, more precisely, using second order approximations for the gradient and the divergence, and an explicit forward time stepping scheme. In order to limit the stencil of the spatial discretization to the four neighboring fragments, we first compute for each fragment the discrete gradient of ρ, multiply it with the diffusion tensor, and store it inside the fragment. In the next iteration, we compute the discrete divergence of these gradient vectors and use it to update ρ using the following explicit forward time scheme

$$\begin{pmatrix} g \\ h \end{pmatrix}^k_{i,j} = a\mathbf{u} \left[\mathbf{u} \cdot \begin{pmatrix} D_x(\rho)^k_{i,j} \\ D_y(\rho)^k_{i,j} \end{pmatrix} \right] \tag{6}$$

$$\rho^{k+1}_{i,j} = \rho^k_{i,j} + \Delta t \left[D_x(g)^k_{i,j} + D_y(h)^k_{i,j} \right] \tag{7}$$

where $\rho^k_{i,j}$ is the value of ρ at position (i,j) and time step k, $(g,h)^T$ the gradient field, and D_x and D_y the discrete derivatives given by the central difference operators

$$D_x(f)^k_{i,j} = \frac{1}{2}\left(f^k_{i+1,j} - f^k_{i-1,j}\right) \tag{8}$$

$$D_y(f)^k_{i,j} = \frac{1}{2}\left(f^k_{i,j+1} - f^k_{i,j-1}\right) \tag{9}$$

A necessary condition for the stability of this scheme is given by the Courant-Friedrichs-Lewy (CFL) condition $\Delta t \leq ||\mathbf{u}||^2/a$ [Str04]. In order to guarantee a fast convergence, we normalize the vector field and use $a = 1$ and $\Delta t = 0.95$. Using the non-normalized field would require limiting the time step to satisfy the CFL condition, leading to slow convergence, or using a more sophisticated discretization method that does not have such a condition. Note that Equation 7 depends on gradients computed by Equation 6. This is implemented by executing both equations in separate render passes. One may be tempted to instead execute both equations in one pass and use the gradients from the previous time step. This will however halve the maximal stable time step, thus bringing no real improvement.

Figure 4: Discretizations of the gradient. (a) Full 3x3 stencil. (b) Forward and backward differences. (c) Central differences.

Instead of central differences, other discretizations could be used. Figure 4(a) shows the result of a full 3x3 stencil, as used in [SW00]. This operator is more precise at the cost of accessing diagonal elements, which is expensive in the illustration buffer, as it requires two indirect loading operations (e.g., follow the left neighbor of the upper neighbor to get the upper left diagonal element). Figure 4(b) shows the result of first order forward differences for the gradient and backward differences for the divergence. Using this scheme, Equations 6 and 7 can be implemented in a single pass, however this scheme suffers from severe numerical diffusion. The central differences, shown in Figure 4(c), are a good compromise between speed and quality. An analysis of similar five point schemes [SH07] revealed that central differences have the lowest numerical diffusivity at the cost of checkerboard patterns appearing near large gradients. A checkerboard has a constant zero gradient for the central differences operator and thus will not be removed by the diffusion process. Fortunately, our input noise is very smooth. For smooth vector fields and spot sizes of at least four pixels, these patterns are usually not noticeable. If necessary, more sophisticated schemes as presented in [SH07] could be used to avoid this problem.

A very important aspect of the discretization is the correct handling of boundary conditions. Since the computational domain Ω is the set of all connected fragments, we first have to find boundary fragments and estimate the domain normal $\mathbf{n}$. We use the following numerical approximation for the outer normal

$$\mathbf{n} = -\sum_{\Delta\mathbf{i}\in\mathcal{N}_{i,j}} \Delta\mathbf{i} \tag{10}$$

where $\mathcal{N}_{i,j} \subseteq \{(-1,0),(1,0),(0,-1),(0,1)\}$ is the set of offsets for which pixel (i,j) contains valid neighbors. This is illustrated in Figure 5(a), where the red cells are fragments outside of the surface and the blue cells are fragments inside the surface. Note that this approach does not work for degenerate fragments where the normal is not uniquely defined, which we ignore since such fragments will already have zero gradient across the domain boundary.

When accessing fragments outside of the domain in Equations 8 and 9, we use reflecting boundary conditions. These conditions state that the function is virtually reflected at the boundary, i.e., $f_R^k = f_L^k$ and $D_x(f)_R^k = -D_x(f)_L^k$ in Figure 5(a). This guarantees $\nabla\rho\cdot\mathbf{n} = 0$, but not $\mathbf{D}\nabla\rho\cdot\mathbf{n} = 0$. From Equation 5 we see that at a boundary, the projected gradient must not have a component parallel to $\mathbf{n}$. We enforce this condition by modifying $\mathbf{u}$ at the boundary so that it is perpendicular to $\mathbf{n}$. We have found this approach to be more stable than using first order differences at the boundary and removing the normal component from the flux $\mathbf{D}\nabla\rho$. Figure 5(b) demonstrates that the diffusion produces correct results for arbitrary boundaries. The only numerical difficulties we have encountered are checkerboard patterns near boundaries, as shown in Figure 5(c). We have not found them to be disturbing enough to justify more sophisticated discretizations as explained above.

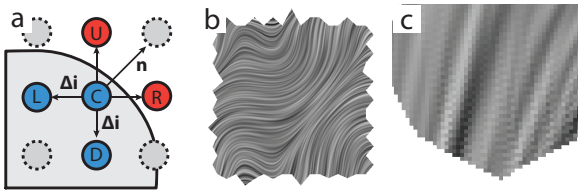

Figure 5: Boundary conditions. (a) Estimating the boundary normal. (b) 2000 diffusion iterations with a complex surface boundary. (c) Checkerboard pattern near boundaries.

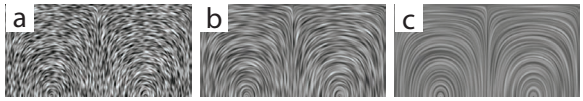

Figure 6: Number of iterations on a 512×256 pixel image. (a) 20 iterations. (b) 100 iterations. (c) 1000 iterations.

The diffusion process is repeated for a number of iterations chosen by the user. Higher numbers will result in longer streaks and generally a higher image quality, however increase the run time proportionally. A comparison of different diffusion times is shown in Figure 6. In practice, the number of iterations should be adapted to the desired level of interactivity, i.e., decreased for interactive data exploration and increased for still images.

3.4. Relation with line integral convolution

In this section we analyze how the anisotropic diffusion is related to line integral convolution. Ignoring numerical diffusion, Equation 2 with $\hat{a} = 0$ can only transport information along streamlines and thus corresponds to a one-dimensional diffusion along a streamline. The analytic solution of the one-dimensional diffusion equation is

$$\rho(x,T) = \frac{1}{\sqrt{4\pi aT}}\int_{-\infty}^{\infty} e^{\frac{-(x-y)^2}{4aT}}\rho(y,0)dy \tag{11}$$

which closely resembles the one-dimensional line integral convolution given by

$$\rho(x) = \int_{-L}^{L} k(x-y)\rho_0(y)dy \tag{12}$$

where $k(x)$ is the convolution kernel and L the convolution length. This comparison shows that the anisotropic diffusion after a time T corresponds to a line integral convolution with an infinite convolution length and a Gaussian kernel $k(x)$. Note that $k(x)$ is equal to the probability density function of the normal distribution with variance $\sigma^2 = 2aT$. Since 99.7% of the area of $k(x)$ lies in the range $[-3\sigma, 3\sigma]$, the convolution length can be safely limited to $L \approx 3\sqrt{2aT}$. This relation can be used to express parameters for our method in terms of parameters for existing LIC based visualizations in order to obtain similar results.

3.5. Compositing

The last stage of our method is the compositing of the illustration buffer into the final image. Our basic approach is to convert the fragment intensity $\rho \in [-1,1]$ into the fragment transparency $\alpha \in [0,1]$ and composite the fragments using alpha blending. We do not change the color of the fragments, leaving the choice of surface and background color to the user. This will result in the surface being covered by opaque line-like streaks, interleaved by transparent streaks. In order to facilitate the perception of the depth ordering for multiple layers, we want the streaks to be fully transparent and fully opaque, respectively, i.e., we want the image to have a high contrast. Since the diffusion process reduces the contrast of the image, we perform a simple contrast stretching

$$\alpha = \texttt{min}(\texttt{max}(r \cdot 0.5 \cdot \rho + 0.5, 0), 1) \tag{13}$$

with a stretching parameter r. In our applications, we use values between 2 and 4.

When illustrating multiple layered surfaces, care has to be taken when choosing the colors for the surfaces and the background. Dense vector field visualization methods only work well if there is sufficient contrast in the final image. Since the image will contain streaks from all surface layers interleaved by the background, we want the set of all used colors to contain elements with pairwise high contrast. Example choices are white surfaces on a black background, or cyan, magenta, and yellow surfaces on black background. The maximal number of layers that can be visualized greatly depends on the data. In general, a free choice of surface color and animated images increase the number of layers.

3.6. Illustrative techniques

The core of our method is the solution of a partial differential equation on a screen space representation of the surface with multiple layers. This framework is very flexible and can be used to implement a number of extensions. In order to enhance the perception of the surface shape and depth ordering, we use an illustrative method for the enhancement of surface boundaries [CFM*11]. This effect uses a separate diffusion process to compute for each fragment its screen space distance to the nearest surface silhouette. The distance is then used to update the intensity ρ just before compositing with

$$\rho \leftarrow \rho + k_1 e^{-k_2 \beta^2} \tag{14}$$

where β is the fragment distance to the nearest silhouette and $k_1 \in [0,2]$ and $k_2 > 0$ user-defined parameters modifying the strength and width of the illustrative enhancement. An additional advantage of the illustration buffer is that the number of surface layers is known for each pixel. We use this information to adaptively lower the density of surface streaks for image parts that contain many layers. This is implemented by lowering and rescaling the initial noise value ρ^0 with

$$\rho^0 \leftarrow \frac{\rho^0 - (k_3 + k_4 n)}{1 - (k_3 + k_4 n)} \tag{15}$$

where n is the number of layers and k_3 and k_4 user-defined parameters. The effect of these enhancements is shown in Figure 10.

4. Implementation and performance

We have implemented our method on a desktop computer using OpenGL 4.2. All operations, including building the illustration buffer, generating the initial noise, the anisotropic diffusion, and the compositing, were written as custom GLSL shader programs. Note that due to the random memory access required by the illustration buffer, the method needs OpenGL 4 compatible hardware, such as the NVIDIA GeForce 400 or AMD Radeon 5000 series.

Table 1 shows the timings of our method on an NVIDIA GeForce 580 GPU. The times were measured as an average over 1000 random camera positions with a fixed camera distance. The method scales linearly with the number of fragments in the image and the number of iterations used and is therefore highly output sensitive. Note that our performance numbers are slower than the ones presented in [CFM*11]. We attribute this in part to our implementation that was written with focus on highest flexibility instead of performance, and in part to the use of the very latest OpenGL standard, which might not be used in the most optimal way. We expect that the presented performance numbers can be improved significantly.

Scene	Medium		High	
Figure 1	148 ms	18MB	1016 ms	71MB
Figure 7	529 ms	59MB	3770 ms	237MB
Figure 9	686 ms	83MB	10917 ms	333MB
Figure 10	229 ms	23MB	1640 ms	95MB

Table 1: Average render times and graphics memory consumption for different scenes. Medium quality is 512×512 pixels image resolution with 100 diffusion iterations, high quality is 1024×1024 pixels with 200 iterations.

5. Results

In this section, we will present results of our method applied to three different problems: simulated atmospheric flow on an extrasolar planet, Lagrangian coherent structures in a revolving door simulation, and a stream surface of a synthetic velocity field. For higher print quality, images were rendered using a large number of iterations.

5.1. Exoplanet

In the first application, we visualize the results of a weather simulation on an extrasolar planet [HMP11]. The planet is roughly the size of Jupiter and is tidally-locked, i.e., one side of the planet is always facing the star. Due to the physical nature of the problem, the atmospheric flow is mostly horizontal and limited to near-spherical surfaces, which were also used in the discretization of the computational domain. Therefore, a visualization of the flow along these surfaces is of great interest to the researchers.

Figure 7 illustrates the flow along two layers of the atmosphere. The lower layer, located about 2000 km above the planetary surface, is where infrared light is escaping. The upper layer, located about 7000 km above the surface, is not visible to the eye. Since the temperature is an important property in this problem, the surfaces were colored according to the temperature, with values ranging from 600K (blue) over 1300K (green) to 2000K (red). Note how the atmosphere on the left (day) side of the planet is warmer and more bulgy than the atmosphere on the right (night) side.

In this application, we have chosen a high frequency initial noise and long diffusion times, resulting in long, thin streaks. In order to achieve the highest quality for these thin features, we have used the full 3×3 stencil for the gradient discretization instead of the central difference discretization that was used in all other results. The planet surface was rendered as an opaque black object in order to provide contrast for the atmosphere layers. When presented to domain experts, our results have been assessed as extremely useful because one can then get an idea of how the planetary flow is reacting to the starlight at different layers.

5.2. Revolving door

In the second application, we visualize Lagrangian coherent structures (LCS) [Hal01] in a simulation of a revolving door [SFB*11]. In order to minimize energy losses from cold air entering through the revolving door, an air curtain was added, blowing warm air upwards from the floor behind the door. LCS have proved to be very useful in finding energy leaks in this scenario and are therefore important to visualize. Since by definition, the flux through the LCS is negligible, a visualization of the flow along these surfaces will provide physically relevant information.

Figures 8 and 9 show the LCS, where blue color denotes repelling structures and red attracting ones. The main problem with LCS-based visualizations is dealing with clutter as LCS tend to have complicated structures. In [SFB*11], cutaways were used to reveal the structure close to the air curtain, as seen in Figure 8. Even though cutting away half of the mesh does a very good job at revealing the structure of the LCS, cutting meshes is cumbersome and finding proper cut locations takes time. In Figure 9, we can see how our dense flow visualization helps understanding the global structure of the LCS even without using any cutaways. Since understanding the shape of the LCS is very important, we have chosen a sparse, low frequency initial noise and have applied an illustrative, non-local silhouette enhancement as described in [CFM*11]. Note that our visualization shows only the instantaneous flow direction and care has to be taken when interpreting this information on the time-dependent LCS surfaces.

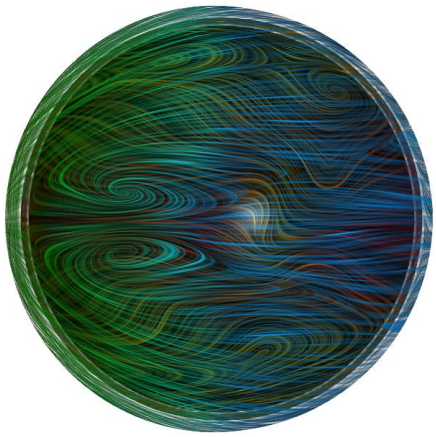

Figure 7: Two-layered dense visualization of the atmosphere of a tidally-locked exoplanet. The streaks are colored using the temperature. 800×600 pixels, 500 iterations.

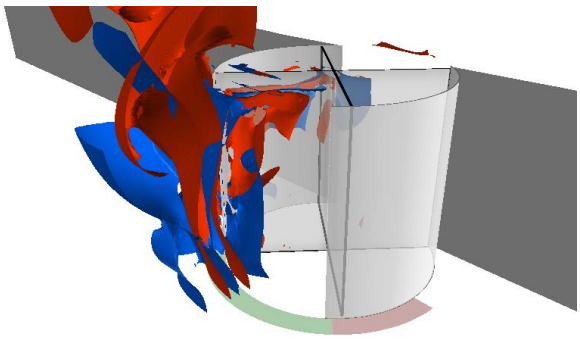

Figure 8: LCS based visualization of the simulation of a revolving door with air curtain using cutaways. The air curtain is modeled by an air outlet, as shown in green. On both sides of the air curtain, air intakes are placed to compensate for air blown in by the air outlet.

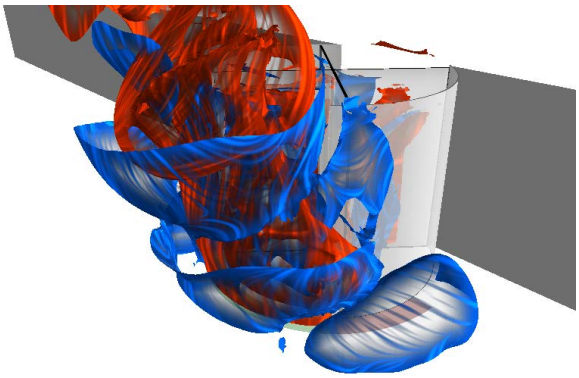

Figure 9: LCS based visualization of the simulation of a revolving door with an air curtain using our method. Even without cutaways, the general structure of the LCS is much more easily understood. 800×600 pixels, 250 iterations.

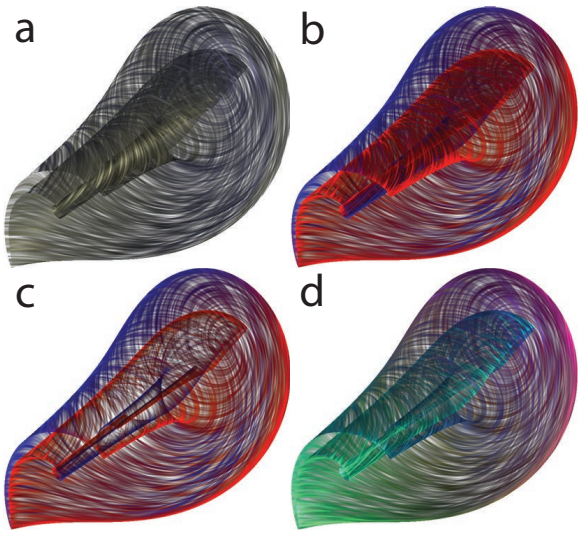

Figure 10: Visualization of the flow along a stream surface. (a) Base result. (b) Cool-warm shading and silhouette enhancement. (c) Adaptively reduced noise density. (d) Automatic surface coloring. 1024×1024 pixels, 1200 iterations.

5.3. Stream surface

In the third application, we visualize a swirling flow. The vector field is an analytic modification of Hill's spherical vortex, and the surface was produced using high quality numerical integration. Since there is by definition no flow through a stream surface, and recirculating flows produce twisted and folded structures, this setting is suitable to explore the expressiveness of our visualization.

Figure 10 demonstrates four different variations of our method. In Figure 10(a), the method was applied without any extensions. In Figure 10(b), an illustrative cool-warm shading [GGSC98] and a non-local silhouette enhancement according to Equation 14 with $k_1 = 0.5$ and $k_2 = 30$ was added. In Figure 10(c), the initial noise was adapted to the number of surface layers, according to Equation 15 with $k_3 = -0.4$ and $k_4 = 0.4$. In Figure 10(d), the color of the surface was computed from the geodesic distance to three uniformly distributed points on the surface, resulting in a better color difference between streaks that are geodesically far apart.

6. Conclusion and future work

In this paper, we have presented a dense flow visualization method that is applicable to surfaces with multiple self-occluding layers, where both the surface shape and the flow along the surface is of importance. To our knowledge our method is the first to transfer the efficiency of 2D screen-space methods to 2.5D. Furthermore, our 2.5D data structure allows for seamless integration with illustrative techniques. The method works with arbitrary triangle meshes, does not need a parametrization, and only requires the vector field values to be known at the mesh vertices. By using both view-dependent and object space properties, it generates patterns of constant screen space frequency configurable by an intuitive user parameter – the screen space diameter of the noise spots. We have found that with our visualization, two surface layers are almost always discernible in animations, and usually more in interactive viewing. In still images, the visualization works best if the colors of the layers vary and the flows are not parallel. The perception of multiple layers might be improved by an optimal choice of surface color and noise density that maximizes the number of discernible layers, which is an interesting open problem.

The advantages of our method come at a cost: the 2.5D data structure is complex to implement and requires recent graphics hardware. Although it scales linearly with the number of image pixels, it may be too slow for real time application. Additionally, it may not be suitable for very fragmented and noisy surfaces, where the neighboring fragment search is difficult and the streamlines very short. Finally, a pure anisotropic diffusion can only produce streamlines. For time dependent flows, pathlines, streaklines, or pattern advection might be more useful. In future work, we wish to extend our method to advection-diffusion schemes in order to address this problem. Since the core of our method is the solution of a partial differential equation on multi-layered surfaces, other equations can easily be investigated as well.

Acknowledgments

The authors would like to thank Kevin Heng, Stefan Barp and Matthias Röthlin for the CFD datasets. This work was funded by the Swiss National Science Foundation under grant 200021_127022, the ETH grant 12 09-3, and the Future and Emerging Technologies (FET) program of the European Commission, under FET-Open grant number 226042.

References

[BSH97] BATTKE H., STALLING D., HEGE H.-C.: *Fast line integral convolution for arbitrary surfaces in 3D*. Springer-Verlag, Heidelberg, 1997, pp. 181–195. 2

[BWF*10] BORN S., WIEBEL A., FRIEDRICH J., SCHEUERMANN G., BARTZ D.: Illustrative stream surfaces. *IEEE transactions on visualization and computer graphics 16*, 6 (2010), 1329–38. 3

[Car84] CARPENTER L.: The A-buffer, an antialiased hidden surface method. In *Proc. SIGGRAPH* (1984), ACM, pp. 103–108. 3

[CFM*11] CARNECKY R., FUCHS R., MEHL S., JANG Y., PEIKERT R.: *Smart transparency for illustrative visualization of complex flow surfaces*. Tech. Rep. 746, ETH Zurich, http://www.inf.ethz.ch/research/disstechreps/techreports, 5 2011. 3, 7, 8

[CL93] CABRAL B., LEEDOM L.: Imaging vector fields using line integral convolution. *Computer Graphics 27* (1993), 263–272. 2

[DPR00] DIEWALD U., PREUSSER T., RUMPF M.: Anisotropic diffusion in vector field visualization on Euclidean domains and surfaces. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (2000), 139–149. 2

[GGSC98] GOOCH A., GOOCH B., SHIRLEY P., COHEN E.: A non-photorealistic lighting model for automatic technical illustration. In *Proc. SIGGRAPH* (1998), ACM, pp. 447–452. 9

[GL05] GRABNER M., LARAMEE R. S.: Image space advection on graphics hardware. In *Proc. Spring Conference on Computer Graphics* (2005), ACM, pp. 77–84. 2

[Hal01] HALLER G.: Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Phys. D 149*, 4 (2001), 248–277. 1, 8

[Hec83] HECKBERT P.: Texture mapping polygons in perspective. *NYIT Computer Graphics Lab Tech. Memo 13* (1983). 4

[HGH*10] HUMMEL M., GARTH C., HAMANN B., HAGEN H., JOY K. I.: IRIS: illustrative rendering for integral surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1319–28. 3

[HMP11] HENG K., MENOU K., PHILLIPPS P. J.: Atmospheric circulation of tidally locked exoplanets: a suite of benchmark tests for dynamical solvers. *Monthly Notices of the Royal Astronomical Society 413*, 4 (2011), 2380–2402. 8

[IFP97] INTERRANTE V., FUCHS H., PIZER S.: Conveying the 3D shape of smoothly curving transparent surfaces via texture. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (1997), 98–117. 2

[JEH02] JOBARD B., ERLEBACHER G., HUSSAINI M. Y.: Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics 8* (2002), 211–222. 2

[KGJ09] KRISHNAN H., GARTH C., JOY K.: Time and Streak Surfaces for Flow Visualization in Large Time-Varying Data Sets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1267 –1274. 2

[LHD*04] LARAMEE R., HAUSER H., DOLEISCH H., VROLIJK B., POST F., WEISKOPF D.: The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum 23*, 2 (2004), 143–161. 2

[LJH03] LARAMEE R., JOBARD. B., HAUSER H.: Image Space Based Visualization of Unsteady Flow on Surfaces. In *Proc. IEEE Visualization* (2003), IEEE Computer Society, pp. 131–138. 2

[LM02] LIU Z., MOORHEAD R. J.: AUFLIC: An Accelerated Algorithm For Unsteady Flow Line Integral Convolution. In *Proc. Joint Eurographics – IEEE TCVG Symp. on Visualization* (2002), Eurographics Association, pp. 43–52. 2

[LTH06] LI G.-S., TRICOCHE X., HANSEN C. D.: GPUFLIC: Interactive and accurate dense visualization of unsteady flows. In *Proc. EuroVis* (2006), Eurographics Association, pp. 29–34. 2

[MCW92] MAX N., CRAWFIS R., WILLIAMS D.: Visualizing wind velocities by advecting cloud textures. In *Proc. IEEE Visualization* (1992), IEEE Computer Society, pp. 179–184. 2

[Per85] PERLIN K.: An image synthetizer. In *Proc. SIGGRAPH* (1985), ACM, pp. 287–296. 3

[Per01] PERLIN K.: Noise hardware. In *SIGGRAPH 2001 Course Notes*, Olano M., (Ed.). ACM, 2001, ch. 9. 3, 4

[SFB*11] SCHINDLER B., FUCHS R., BARP S., WASER J., POBITZER A., MATKOVIĆ K., PEIKERT R.: Lagrangian Coherent Structures for Design Analysis of Revolving Doors. *Submitted* (2011). 8

[SH95] STALLING D., HEGE H.-C.: Fast and resolution independent line integral convolution. In *Proc. SIGGRAPH* (1995), ACM, pp. 249–256. 2

[SH07] SHARMA P., HAMMETT G. W.: Preserving monotonicity in anisotropic diffusion. *J. Comput. Phys. 227*, 1 (2007), 123–142. 6

[SJK04] SANDERSON A., JOHNSON C., KIRBY R.: Display of vector fields using a reaction-diffusion model. In *Proc. IEEE Visualization* (2004), IEEE Computer Society, pp. 115–122. 2

[SK97] SHEN H.-W., KAO D. L.: UFLIC: a line integral convolution algorithm for visualizing unsteady flows. In *Proc. IEEE Visualization* (1997), IEEE Computer Society, pp. 317–322. 2

[Str04] STRIKWERDA J. C.: *Finite Difference Schemes and Partial Differential Equations*. SIAM, 2004. 5

[SW00] SCHARR H., WEICKERT J.: An anisotropic diffusion algorithm with optimized rotation invariance. In *Proc. DAGM-symposium* (2000), Springer, pp. 460–467. 2, 6

[UIL*06] URNESS T., INTERRANTE V., LONGMIRE E., MARUSIC I., O'NEILL S., JONES T.: Strategies for the visualization of multiple 2D vector fields. *Computer Graphics and Applications, IEEE 26*, 4 (2006), 74–82. 3

[vW02] VAN WIJK J. J.: Image based flow visualization. *ACM Trans. Graph. 21* (2002), 745–754. 2

[vW03] VAN WIJK J. J.: Image based flow visualization for curved surfaces. In *Proc. IEEE Visualization* (2003), IEEE Computer Society, pp. 23–130. 2

[WE04] WEISKOPF D., ERTL T.: A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In *Proc. Graphics Interface* (2004), Canadian Human-Computer Comm. Soc., pp. 263–270. 2

[WEE03] WEISKOPF D., ERLEBACHER G., ERTL T.: A texture-based framework for spacetime-coherent visualization of time-dependent vector fields. In *Proceedings IEEE Visualization* (2003), IEEE Computer Society, pp. 107–114. 2

[Wei97] WEICKERT J.: A review of nonlinear diffusion filtering. In *Scale-Space Theory in Computer Vision*, vol. 1252 of *Lecture Notes in Computer Science*. Springer, 1997, pp. 1–28. 2