LIC on Surfaces

Detlev Stalling

Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)

1 Introduction

In this lecture we will discuss how line integral convolution can be generalized to depict vector fields defined on surfaces in 3D. The surfaces are assumed to be topologically equivalent to a 2D plane – at least in a local neighbourhood. In principle, any 2D imaging technique can be applied to such a surface as well. In particular, it is possible to map standard 2D LIC images onto the surface to visualize directional information. This is very similar to conventional texture mapping in computer graphics. Applying LIC on surfaces implies a number of interesting questions, e.g. how to compute field lines on a surface, how to define a suitable input noise, or how to perform the texture mapping in detail. Different methods have been proposed to solve these problems. Basically, surface LIC algorithms can be divided into two groups depending on whether they operate in parameter space or in physical space. After introducing tangent curves on surfaces mathematically, we will discuss and compare both groups of algorithms in detail.

Of course, line integral convolution is not the only method to depict directional information on a surface. Other methods for 2D vector field visualization can be generalized to surfaces as well, e.g. arrows or contour lines. Such tools are provided by many visualization systems today. Spot noise methods have also been applied on surfaces for some time. Van Wijk already described texture mapping on parametric surfaces in his 1991 spot noise paper [10]. Improvements are described in [2]. In both, spot noise and surface LIC, a main issue is to ensure that the mapped texture is not distorted and faithfully represents the desired information. The solutions proposed for both methods are quite similar. We will come back to this topic below. In general, the advantages of texture based vector field visualization methods on surfaces are the same as in the flat case. Compared to conventional visualization techniques much more information can be encoded in a single image. The global structure of a surface flow is easily understood, especially if the surface can be zoomed and rotated interactively.

2 Tangent Curves on Surfaces

Conventional line integral convolution is a method for visualizing a set of curves in a plane, namely the integral curves of a 2D vector field. With the methods discussed in this lecture we get images of so-called tangent curves on surfaces. Now, what exactly are these curves and how are they related to vector fields ?

Tangent curves can be defined in two ways. Let us assume that a surface M can be locally parametrized by Euclidean coordinates $u, v \in \mathbb{R}^2$. First, we consider a 2D vector field $\boldsymbol{f}$ defined in parameter space G.

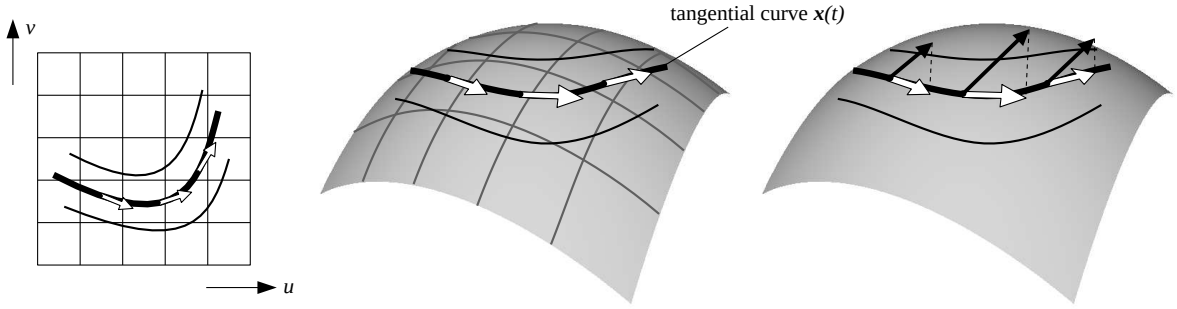


Figure 1: A vector field $\boldsymbol{f}$ in parameter space defines a set of tangent curves on a surface. The map of $\boldsymbol{f}$ onto the surface produces a tangential vector field $\boldsymbol{F}$. Tangent curves can also be defined by projecting a general vector field $\boldsymbol{V}$ onto the surface.

The field and its integral curves $\boldsymbol{u}(t)$ are related by

$$\boldsymbol{f} = \frac{d\boldsymbol{u}}{dt}, \quad \boldsymbol{f} : G \mapsto \mathbb{R}^2 \tag{1}$$

By mapping the integral curves of $\boldsymbol{f}$ onto the surface we obtain tangent curves $\boldsymbol{x}(t)$ on M. The map of $\boldsymbol{f}$ itself produces a tangential vector field $\boldsymbol{F}$ on M and we have

$$\boldsymbol{F} = \frac{d\boldsymbol{x}}{dt}, \quad \boldsymbol{F} : \boldsymbol{x} \ni M \mapsto TM_{\boldsymbol{x}}, \tag{2}$$

where $TM_{\boldsymbol{x}}$ denotes the tangent space of M at the point $\boldsymbol{x}$. The relationship between the vector field $\boldsymbol{f}$ in parameter space and its map $\boldsymbol{F}$ is illustrated in Fig. 1.

Alternatively, we may also consider a 3D vector field $\boldsymbol{V} : M \mapsto \mathbb{R}^3$ over the surface. A tangent curve $\boldsymbol{x}(t)$ defined by $\boldsymbol{V}$ is a curve on M where the tangential vector $d\boldsymbol{x}/dt$ and the projection of $\boldsymbol{V}$ into the tangent space $TM_{\boldsymbol{x}}$ are equal for any point of the curve. Obviously, we obtain the same tangent curves as in the previous definition if the tangential part of $\boldsymbol{V}$ is equal to $\boldsymbol{F}$, i.e.

$$\boldsymbol{F} = \boldsymbol{V} - (\boldsymbol{V} \cdot \boldsymbol{n})\, \boldsymbol{n}. \tag{3}$$

Here $\boldsymbol{n}$ denotes the normalized surface normal vector. In principle tangent curves on surfaces can either be computed in parameter space by integrating Eq. (1), or in physical space by integrating Eq. (2). In the latter case a general 3D vector field has to be projected onto the surface using Eq. (3). It is the tangential part of $\boldsymbol{V}$ which can be visualized using line integral convolution. Fields which differ only with respect to their normal component will produce the same LIC pattern. This is a general limitation of LIC on surfaces. To visualize the normal component of a vector field, one might apply color coding or may vary the LIC filter length appropriately. However, with these methods the 3D structure of a field is often hard to understand.

Fortunately, there are a number of interesting applications where vector fields are inherently defined in the tangential space of a surface or where meaningful results can be obtained by analyzing the surface projection of a vector field. For example in CFD simulations it is of interest to visualize the projection of a 3D velocity field on a body in the flow. In Computer Aided Geometric Design or in differential geometry lines of curvature and other tangent curves have to be visualized. Finally, two-dimensional dynamical systems may be defined on a non-planar domain and the trajectories have to be represented on such a domain. Examples of theses applications will be presented below.

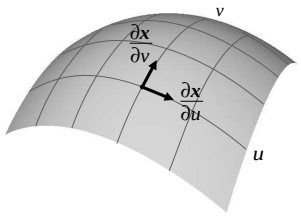
3 LIC in Parameter Space

Applying line integral convolution to surfaces becomes particularly easy when the *whole* surface can be parametrized globally by two-dimensional Euclidean coordinates. In this case the LIC texture can be computed completely in parameter space using conventional 2D LIC algorithms. The resulting image can be mapped back onto the surface using standard texture mapping techniques as provided by modern 3D graphics libraries or rendering software. This method has been proposed for the first time by Lisa Forssell in 1994 [3]. In the following let us first introduce parametric surfaces, before discussing the algorithm itself as well as methods to compensate texture distortion.

3.1 Parametric Surfaces

A parametric surface is defined by

$$\boldsymbol{x}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}, \quad \boldsymbol{u} = \begin{pmatrix} u \\ v \end{pmatrix} \in \mathbb{R}^2. \tag{4}$$



The cartesian coordinates x, y, z of a surface point are assumed to be differentiable functions of the parameters u and v. The derivatives $\partial\boldsymbol{x}/\partial u$ and $\partial\boldsymbol{x}/\partial v$ lie in the tangent space of the surface. A parametrization is said to be regular, if both vectors are linearly independent, i.e. if they form a basis of the tangent space.

In many situations the tangential vector field $\boldsymbol{F}$ to be visualized will be given in physical space, i.e. $\boldsymbol{F} = d\boldsymbol{x}/dt$. Since the LIC calculation is to be performed in parameter space, we need to find the corresponding 2D vector field $\boldsymbol{f} = d\boldsymbol{u}/dt$. The vector field's tangent curve in physical space is given by $\boldsymbol{x}(u(t), v(t))$. We decompose $d\boldsymbol{x}/dt$ into

$$\frac{d\boldsymbol{x}}{dt} = \frac{\partial\boldsymbol{x}}{\partial u}\frac{du}{dt} + \frac{\partial\boldsymbol{x}}{\partial v}\frac{dv}{dt}. \tag{5}$$

Eq. (5) is a linear system consisting of three equations and two unknowns du/dt and dv/dt. Since $d\boldsymbol{x}/dt$ lies in the tangent plane spanned by $\partial\boldsymbol{x}/\partial u$ and $\partial\boldsymbol{x}/\partial v$, we can find a unique solution for any regularly parametrized surface. Formally we may introduce a linearly independent vector in normal direction,

$$\frac{\partial\boldsymbol{x}}{\partial w} \equiv \frac{\partial\boldsymbol{x}}{\partial u} \times \frac{\partial\boldsymbol{x}}{\partial v}, \tag{6}$$

to obtain a 3×3 system. The resulting regular system can be solved for example using Cramer's rule. While the component belonging to $\partial\boldsymbol{x}/\partial w$ will be zero, the 2D vector in parameter space is given by

$$\frac{du}{dt} = \frac{\det\left[\frac{d\boldsymbol{x}}{dt}, \frac{\partial\boldsymbol{x}}{\partial v}, \frac{\partial\boldsymbol{x}}{\partial w}\right]}{\det\left[\frac{\partial\boldsymbol{x}}{\partial u}, \frac{\partial\boldsymbol{x}}{\partial v}, \frac{\partial\boldsymbol{x}}{\partial w}\right]}, \qquad \frac{dv}{dt} = \frac{\det\left[\frac{\partial\boldsymbol{x}}{\partial u}, \frac{d\boldsymbol{x}}{dt}, \frac{\partial\boldsymbol{x}}{\partial w}\right]}{\det\left[\frac{\partial\boldsymbol{x}}{\partial u}, \frac{\partial\boldsymbol{x}}{\partial v}, \frac{\partial\boldsymbol{x}}{\partial w}\right]}. \tag{7}$$

This equation may also be evaluated for a physical space vector with non-vanishing normal component. The resulting parameter space vector automatically corresponds to the tangential part of the input vector.

An interesting application area for surface LIC algorithms are CFD simulations. Often such simulations are performed on three-dimensional curvilinear grids. Such grids conform to the shape of the body in the flow. Parametric surfaces can be extracted by keeping one grid index fixed. To project the flow field on such a surface and to obtain the corresponding 2D parameter space vectors, again Eq. (7) can be used. However,

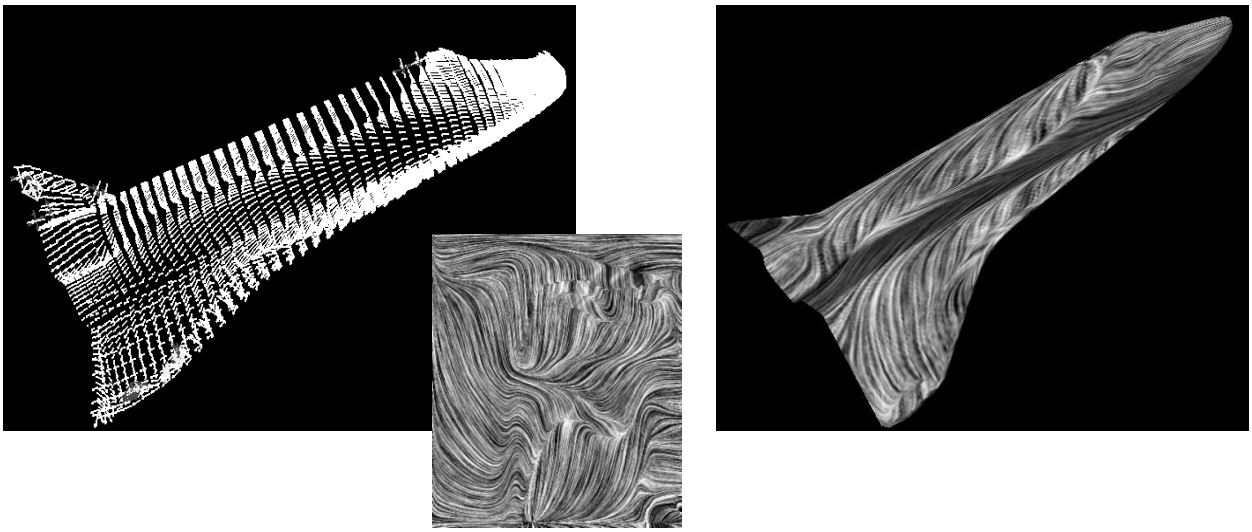


Figure 2: Line integral convolution on a parametric surface: A conventional 2D LIC image is computed in parameter space and is mapped back onto the original surface. Images by courtesy of Lisa Forssell.

in this case the normal component $\partial \boldsymbol{x}/dw$ need not to be computed using Eq. (6) but is inherently being defined by the curvilinear grid. The matrix

$$J = \left[\frac{\partial \boldsymbol{x}}{\partial u}, \frac{\partial \boldsymbol{x}}{\partial v}, \frac{\partial \boldsymbol{x}}{\partial w} \right] \tag{8}$$

is just the Jacobian matrix which transforms between physical space and parameter space.

It should be mentioned, that the mapping between physical space and parameter space may introduce some errors. First, in parameter space usually the vectors are interpolated linearly, although the mapping itself is non-linear. Second, often only a discrete approximation of the Jacobian is known. However, both types of errors can be controlled by choosing the resolution of the surface grid to be sufficiently fine.

3.2 The Algorithm

After the vector field has been transformed into the surface's parameter space a conventional LIC image is computed. For this any standard 2D LIC algorithm can be used. The resulting image is mapped back onto the surface using the texture mapping techniques provided by modern graphics libraries like OpenGL. The algorithm is illustrated in Fig. 2. The image shows a simulated flow field around a space shuttle, projected onto a plane of a curvilinear grid. Today texture mapping is supported in hardware on an increasing number of graphics workstations. On such machines the final object can be displayed at truely interactive frame rates. This is an important feature for analyzing the structure of the surface flow.

3.3 Compensating Texture Distortions

The main drawback of the surface LIC approach described above are the distortions introduced by the non-isometric mapping between parameter space and physical space. Usually the cell sizes of the parametric surface differ significantly. In some areas of the surface the mapped LIC texture is compressed while in others it is stretched. This is clearly visible from Fig. 2. Near the top of the space shuttle the texture is

much finer than on the wings. For visualization purposes it is often desireable that the texture has equal characteristics across the whole surface. A perspective viewing projection involves an additional scaling of the object. Therefore, a varying feature size of the surface texture may also cause a wrong depth perception.

As a first remedy Forssell suggested to adjust the length of the LIC filter kernel so that it becomes constant in physical space [3, 4]. The idea behind this strategy is that the filter length determines the feature size of the resulting LIC texture. If the filter length is scaled appropriately in parameter space the distortions of the mapping can be compensated – at least partially. The method also helps to compensate errors in the perceived texture speed in surface LIC animations. The filter length formula suggested by Forssell was

$$L(\boldsymbol{x}) = a + b\, r(\boldsymbol{x}), \quad \text{with} \quad r(\boldsymbol{x}) = \frac{|d\boldsymbol{u}/dt|}{|d\boldsymbol{x}/dt|}. \tag{9}$$

The quantity r just denotes the ratio between vector magnitude in parameter space and vector magnitude in physical space. This ratio is low in sparse areas of the grid and high in dense areas. The constant contribution a is a non-vanishing minimum filter length.

Better results can be obtained if not only the filter lengths are scaled, but if also the characteristics of the input texture were adapted locally according to the grid density. The frequencies of the input noise should be lower in areas of the grid which are mapped into a small region in physical space. Ideally such a scaling should not be isotropic but should take into account the distortions in u and v direction separately. The goal is to have an input noise which has equal properties everywhere after being mapped onto the surface.

While the exact compensation of the mapping distortions seems to be difficult, quite satisfactory results have been obtained by Thomas Loser using isotropically modified input noise [6]. He uses the technique described by Kiu and Banks [5] to generate multi-frequency noise. To determine the frequency at a particular location it is computed how much a unit vector perpendicular to the original surface vector is scaled by the mapping. A unit vector perpendicular to $d\boldsymbol{x}/dt$ is obtained by normalizing

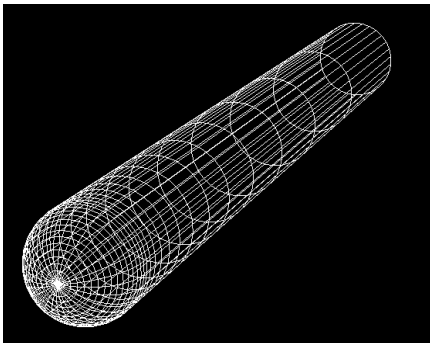
$$\boldsymbol{d} = \frac{d\boldsymbol{x}}{dt} \times (\frac{\partial \boldsymbol{x}}{\partial u} \times \frac{\partial \boldsymbol{x}}{\partial v}). \tag{10}$$

The reason for choosing the direction $\boldsymbol{d}$ is that it controls the thickness of the streaks in a LIC image, while the length can be adjusted using Eq. (9). An example of a multi-frequency input texture is shown in Fig. 3. The figure also shows the corresponding LIC image both in parameter space and after being mapped onto the surface. The data represents the simulated flow around a hemisphere cylinder.

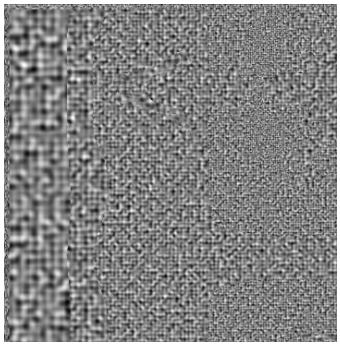
4 LIC in Physical Space

Computing surface LIC images in parameter space is a simple and elegant approach. Unfortunately, for a large number of surfaces no global 2D parametrization is available. Examples are isosurfaces in scalar data sets or stream surfaces in 3D flow fields. These surfaces may have an arbitrary complex topology. Usually only triangular approximations are generated, e.g. using the marching cubes algorithm in the case of isosurfaces. Surfaces occuring in CAGD often consist of smoothly joined parametric patches, but they may as well have a complex topology and therefore in general can't be parametrized globally. Finally, for surfaces obtained from subdivision schemes usually there is also no global parametrization available. In all these examples the surface LIC technique described in the previous section cannot be applied.

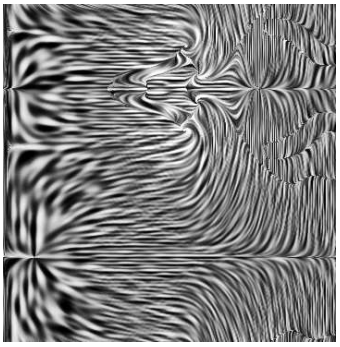
As an alternative approach therefore algorithms have been developed which perform the convolution operation on a given triangulation *directly in physical space*. In addition to being more generally applicable,



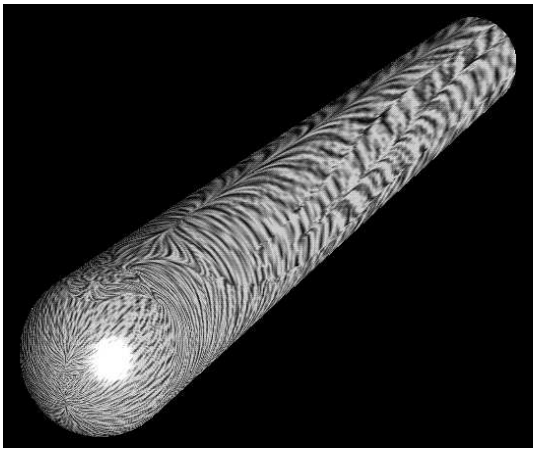
parametric surface



multi−frequency noise



optimized LIC texture



standard texture mapped on surface

optimized texture mapped on surface

Figure 3: An optimized LIC texture can be generated by adjusting the frequency of the input noise according to the grid density. In this case the flow field around a hemisphere cylinder is shown. In areas of high grid density a correspondingly coarser LIC texture is generated. Images by courtesy of Thomas Loser, Yuval Levy, and Lambertus Hesselink (Stanford University, Department of Electrical Engineering).

in these approaches no texture distortions occur because the input noise is also defined in physical space. Methods to compensate distortion are not needed anymore.

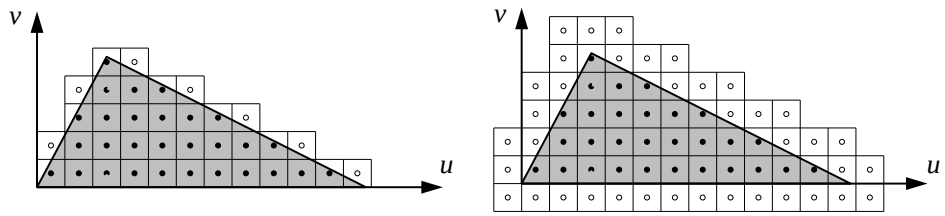
In the following we will first outline two different variants of physical space surface LIC algorithms. After the principles of these methods have become clear, we will describe common details like the definition of input noise or how to perform field line integration on triangular surfaces.

4.1 Local Texture Approach

A single polygon of a triangulated surface can be parametrized without any distortion at all using the identity mapping. This leads to the idea of computing a separate piece of LIC texture for each triangle [1]. In this way surfaces of arbitrary topology can be processed. To produce a continuous LIC pattern, field lines have to be followed across neighbouring triangles. However, within the triangles convolution can be performed in a standard way. In particular, it is possible to apply the fast LIC algorithm described in [9]. The whole process is summarized in the upper part of Fig. 4.

As with parameter space methods, the texture mapping capabilities of modern graphics workstations can be exploited to project the individual LIC images onto the surface. The final object can then be displayed at truely interactive frame rates. However, this requires a rather unnatural cartesian texture grid to be used for the triangles. In general the texture grids of neighbouring triangles will not match exactly. Therefore some artifacts can occur at the triangle edges. Fortunately, these artifacts do not affect visual impression, unless a rather large zoom factor is chosen. The resolution of the local LIC images is a user-definable parameter. The choice of this parameter as well as the definition of the input noise will be discussed below.

The lower part of Fig. 4 shows the influence of the texture interpolation model being used. The interpolation model determines how the textures are resampled during the final rendering step. The OpenGL graphics library supports constant as well as bilinear interpolation. With bilinear interpolation artifacts due to non-conforming local texture grids are less evident. Both, constant as well as bilinear interpolation require the texture area to be somewhat bigger than the triangle itself. Some pixels from the outside are needed to interpolate intensities inside a triangle. Intensities of the outer pixels can be determined from neighbouring triangles in a post-processing step. The actual extent of a triangle texture is shown in the following figure:



constant interpolation bilinear interpolation

4.2 Ray-Surface Intersection Approach

A view-dependent physical space surface LIC algorithm has been proposed recently by Xiaoyang Mao and coworkers [7]. Their method is not tainted by any artifacts due to misaligned local texture grids or due to interpolation of LIC images during rendering. Instead, rays are shot from the viewpoint through screen pixels into the 3D surface. The convolution integral is evaluated separately at all visible ray-surface intersections. The obtained intensities are used to render the corresponding screen pixels. An additional shading of the surface is also taken into account. Fig. 5 shows an image produced by this algorithm.

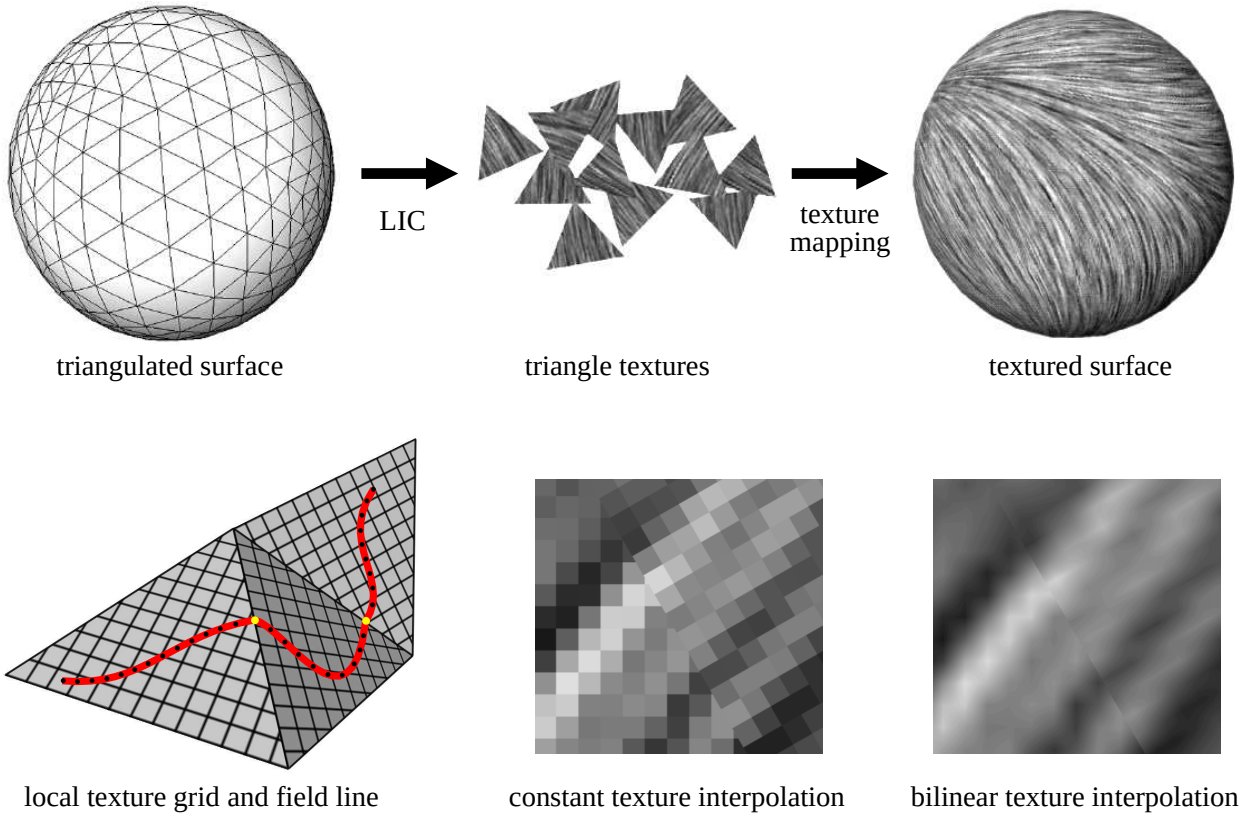


Figure 4: Local texture approach for surface LIC in physical space: For each triangle a separate LIC image is computed. Convolution is performed across neighbouring triangles. In order to make use of standard graphics libraries, cartesian triangle textures have to be used. Edge artifacts due to non-conforming texture grids are only noticeable in a close-up view. They are even less visible if bilinear texture interploation is used.

The main difference between the ray-casting approach and the local texture algorithm described above are the locations at which the convolution integral is evaluated. In the one case, these are exactly the surface points which correspond to screen pixels. In the other case, these are the nodes of intermediate local triangle textures. The intermediate textures have to be resampled (interpolated) during rendering. This will introduce some errors, especially if the texture size is small compared to the size of the screen area it is projected on. However, since texture mapping is implemented very efficiently on modern graphics computers, interactive frame rates can be achieved. On the other hand, the ray-surface intersection approach is more precise. It doesn't involve any interpolation at all at the expense of redoing the LIC calculation for each new viewing direction.

4.3 Solid Noise

Physical space LIC algorithms require an input noise function to be defined at every point on the surface. The noise function should have equal characteristics at all points. This is most easily achieved by evaluating an isotropic 3D noise function. In principle also a predefined noise on a discrete lattice might be used, but this would need a huge amount of memory. Perlin described how to define solid noise procedurally [8]. An efficient implementation of the Perlin noise function is given in [11]. The method works by mapping floating point coordinates to an integer grid. From the integer indices a pseudo-random value is generated using arithmetic and binary operations. To get a continuous noise function, random values of neighbouring grid points have to be interpolated. However, input noise for LIC need not to be continuous. Therefore the interpolation step can be discarded, and the implementation becomes very simple. Particular noise implementations for surface LIC are given in [1, 7].

An important parameter is the granularity of the input noise, i.e. the resolution of the integer grid of pseudo-random values. This parameter determines how much detail is contained in the final LIC texture. Fig. 6 compares two surface LIC images obtained by convolving a solid noise of low and high granularity, respectively. In both cases, the same surface and the same vector field have been used.

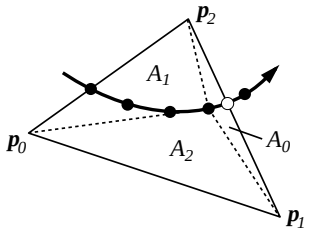
The optimal choice of noise granularity also depends on how the surface is projected onto the screen. If the surface is to small in screen space, the directional information of the LIC texture disappears and aliasing effects occur. Aliasing can be prevented by using local triangle textures together with a proper minification filter. However, also in this case the directional structure disappears and the surface color becomes an average grey. Of course, the resolution of the local triangle textures should be adapted to the frequency of the input noise. If it is to small, the input noise cannot be sampled properly. If it is to large, texture memory would be wasted and computing time increases.

4.4 Field Line Integration on Triangular Surfaces

In this section we will discuss how the field lines of a tangential vector field can be computed on a triangulated surface. Like in conventional 2D LIC a number of different numerical algorithms can be used. The integration may be performed in 3D space or in the local parameter space of the triangles. In any case the integrator needs to check whether the current triangle has been left or not. If it has, a neighbouring triangle needs to be determined where the integration is continued.

Often, a 3D vector field will be defined at the vertices of the triangulation. In this case the vectors have to be projected into the triangles first. Inside a triangle the field can be interpolated linearly. If $\boldsymbol{p}_i$ are the vertices of a triangle, we can express a point $\boldsymbol{p}$ inside the triangle in barycentric coordinates, i.e.

$$\begin{aligned} \boldsymbol{p} &= \boldsymbol{p}_0 + \lambda_1 (\boldsymbol{p}_1 - \boldsymbol{p}_0) + \lambda_2 (\boldsymbol{p}_2 - \boldsymbol{p}_0) \\ &= (1 - \lambda_1 - \lambda_2)\, \boldsymbol{p}_0 + \lambda_1 \boldsymbol{p}_1 + \lambda_2 \boldsymbol{p}_2. \end{aligned} \tag{11}$$



Introducing $\lambda_0 = 1 - \lambda_1 - \lambda_2$, the linearly interpolated field vector at $\boldsymbol{p}$ is given by

$$\boldsymbol{f}(\boldsymbol{p}) = \lambda_0 \boldsymbol{f}(\boldsymbol{p}_0) + \lambda_1 \boldsymbol{f}(\boldsymbol{p}_1) + \lambda_2 \boldsymbol{f}(\boldsymbol{p}_2). \tag{12}$$

It is easy to see that the barycentric coordinates λ_i are just proportional to the signed area parts A_i of the triangle, as illustrated in the above figure. This means, that they can also be used to quickly perform the point-in-triangle test. If one of the three coordinates λ_i gets smaller than zero, then the current triangle has been left. In this case the intersection of the field line with the triangle boundary has to be computed. This is necessary in order to find the correct starting point for field line integration in the neighbouring triangle. Let $\boldsymbol{p}$ be the last sample inside the triangle and $\boldsymbol{q}$ the first one outside. We look for a number s such that $\boldsymbol{p} + s(\boldsymbol{q} - \boldsymbol{p})$ lies on a triangle edge. In order to quickly find s, the triangle edges can be stored in implicit form, i.e. $ax + by + c = 0$. Then s is given by

$$s = \frac{ap_x + bp_y + c}{a(p_x - q_x) + b(p_y - q_y)}. \tag{13}$$

If two barycentric coordinates are smaller than zero this equation has to be evaluated for both edges. The smallest value s corresponds to the nearest intersection and determines the actual exit point. Here we have assumed, that the calculation is performed in 2D space. In 3D space 4 numbers are necessary to define a triangle edge, i.e. $ax + by + cz + d = 0$. Consequently, in Eq. (13) also the z-components of the position vectors have to be included.

Inside a triangle the field lines might be found analytically, since the vector field is interpolated linearly. However, this requires to compute a coefficient matrix, its eigenvalues and eigenvectors, and the evaluation of transcendental functions. It is much simpler and more stable to apply numerical methods. Often simple Euler integration is adequate. Higher order methods like the ones applied in the fast LIC algorithm [9] are not as efficient as in the flat case, since the integration has to be restarted at each triangle boundary. Therefore, only relatively small steps can be taken.

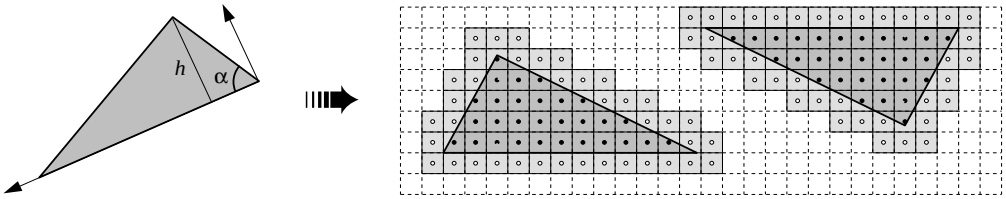
4.5 Triangle Packing

In this section we will consider an important issue of the local texture surface LIC approach, namely how to organize the triangle textures in memory. Modern graphics computers provide fast hardware-supported texture mapping. However, the total size of the textures is limited. Moreover, width and height of individual textures are restricted to powers of 2, i.e. textures have to be $2^i \times 2^j$ pixels large. In some implementations i and j even must be equal.

Obviously, under these constraints it is not a good idea to allocate a separate block of texture memory for every element of a surface triangulation. A large amount of expensive texture memory would be wasted. In addition using many small textures instead of a few big ones may affect rendering speed. Therefore we would like to place many small triangular patches into a single block of texture memory. The packing should be as dense as possible.

An approximate solution of this problem has been proposed in [1]. Since the packing problem is NP-hard, a number of simplifications have been made. The method relies on arranging similar triangles within

rows. Only a limited set of triangle orientations is considered. The longest side of a triangle is either aligned to the bottom or to the top of a row. If necessary, the triangle is flipped so that the second largest angle lies either in the lower left or in the upper right. In total, there are two only possible orientations:



At the beginning the heighest triangle is put in the lower left corner of a quadratic texture region. Then the algorithm proceeds in a greedy way, i.e. it tries to find a triangle which optimally fits to the end of a row. The criterium is to minimize the wasted area between two successive triangles. The wasted area can be determined on a discrete level, i.e. by counting the number of unused pixels. Alternatively, the "most similar" triangle may be found by comparing a weighted sum of the squared angle and height differences. In this case, the next triangle is the one which minimizes

$$d^2 = c_\alpha(\alpha_0 - \alpha)^2 + c_h(h_0 - h)^2. \tag{14}$$

Here α_0 is the angle of the last triangle and h_0 is the height of the row. If one row of texture memory is filled, a new one is started.

This heuristic approach produces quite efficient packings. Typically 85-95% of all pixels in a quadratic texture block are covered. Examples of resulting arrangements are shown in Fig. 7. The textures in these images belong to the surface shown in Fig. 8. The efficiency of the packing algorithm could be further improved by filling the empty regions at the front and back of a row. For this it would be necessary to allow 90° rotations of the triangles as well.

5 Extensions and Future Directions

The methods surveyed in this lecture allow one to compute standard LIC textures on surfaces. We concentrated on the fundamental principles of the algorithms, and did not discuss more sophisticated extensions. Obviously, there are number of possible directions. Many of them are familiar from conventional 2D LIC. For example, Forssell [3, 4] animated surface LIC images to indicate the direction of a flow as well as its magnitude. Of course, such additional scalar information can also be visualized using straight-forward color coding or by varying characteristic features of the LIC texture, e.g. the length of the filter kernel.

The results of 2D LIC can be improved in an attractive way by applying an emboss filter to the final LIC image. In surface LIC this is not as easy, since different surface orientations and lighting conditions may occur. A very similar effect can be obtained using bump mapping, which is provided by many advanced 3D rendering system. Fig. 8 shows an example of a bump-mapped LIC texture. Unfortunately, bump mapping is not supported in OpenGL. Therefore the technique is not amenable for interactive applications.

As with 2D LIC the real challenge for surface LIC is to depict time-dependent data. There are a number of open questions related to this topic. What curves should be used for convolution ? How can we maintain the correlation of LIC textures in sucessive frames ? In surface LIC not only the vector field may change in time, but the whole surface itself. Therefore, by animating surfaces properly, it might also be possible to better understand the structure of 3D data sets.

References

[1] BATTKE, H., STALLING, D., AND HEGE, H. Fast line integral convolution for arbitrary surfaces in 3d. In *Visualization and Mathematics* (1997), H. Hege and K. Polthier, Eds., Springer, pp. 181–195. Also published as ZIB Preprint SC-96-59, available at *http://www.zib.de/Visual/publications/surfaceLIC*.

[2] DE LEEUW, W. C., AND VAN WIJK, J. J. Enhanced spot noise for vector field visualization. In *Visualization '95* (1995), IEEE Computer Society, pp. 233–239.

[3] FORSSELL, L. K. Visualizing flow over curvilinear grid surfaces using line integral convolution. In *Visualization '94* (1994), IEEE Computer Society, pp. 240–247.

[4] FORSSELL, L. K., AND COHEN, S. D. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transaction on Visualization and Computer Graphics 1*, 2 (June 1995).

[5] KIU, M.-H., AND BANKS, D. C. Multi-frequency noise for LIC. In *IEEE Visualization '96* (Oct. 1996), IEEE. ISBN 0-89791-864-9.

[6] LOSER, T. Vector and tensor field visualization using textures. Master's thesis, Institut fü Verfahrenstechnik, University of Hannover, Germany, 1996.

[7] MAO, X., KIKUKAWA, M., FUJITA, N., AND IMAMIYA, A. Line integral convolution for arbitrary 3d surfaces through solid texturing. In *Proceedings of Eighth Eurographics Workshop on Visualization in Scientific Computing* (1997), to appear.

[8] PERLIN, K. An image synthesizer. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (July 1985), B. A. Barsky, Ed., vol. 19, pp. 287–296.

[9] STALLING, D., AND HEGE, H. Fast and resolution independent line integral convolution. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), R. Cook, Ed., Annual Conference Series, ACM SIGGRAPH, pp. 249–256. held in Los Angeles, California, 06-11 August 1995.

[10] VAN WIJK, J. J. Spot noise. *Computer Graphics 25*, 4 (July 1991), 309–318.

[11] WARD, G. A recursive implementation of the perlin noise function. In *Graphics Gems 2*, J. Arvo, Ed. Academic Press Inc., 1991, pp. 396–401.

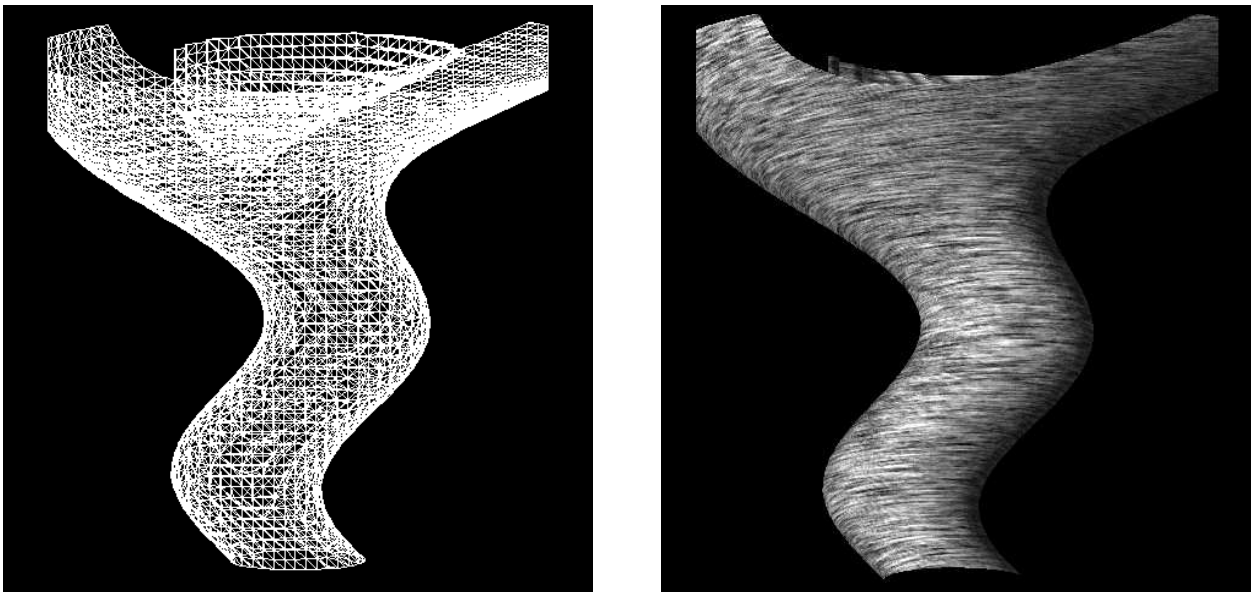


Figure 5: Physical space LIC algorithms can operate on arbitrary triangulations. Here a contour surface has been extracted from a tornado data set using the marching cubes algorithm. The LIC image has been produced by computing ray-surface intersections for screen pixels first. At all visible intersection points line integral convolution is done. Images by courtesy of Xiaoyang Mao.

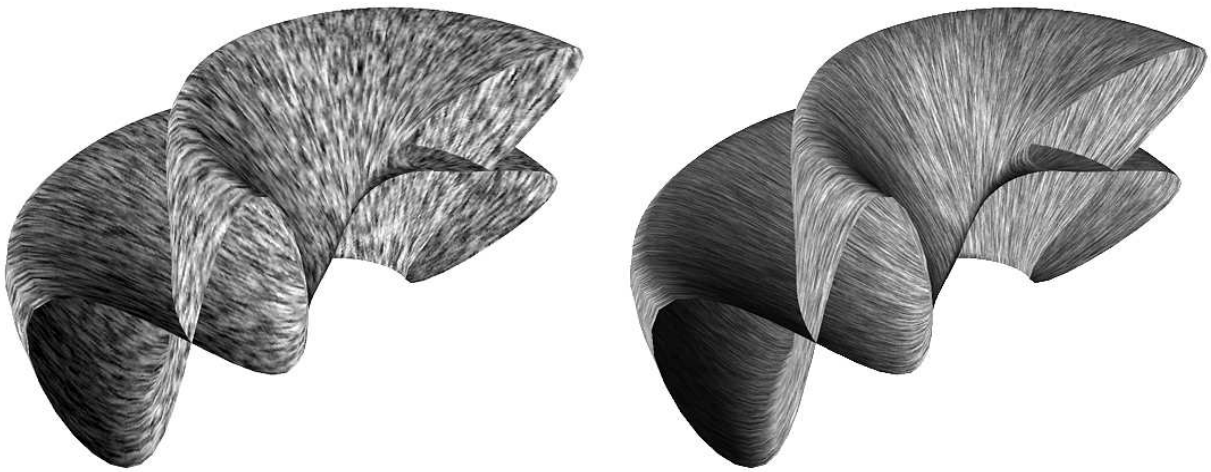


Figure 6: The resolution of the input noise determines how much detail is contained in the final surface texture. The images show contour lines on parts of a surface with the topology of a Klein bottle. A separate LIC texture has been computed for every triangle.

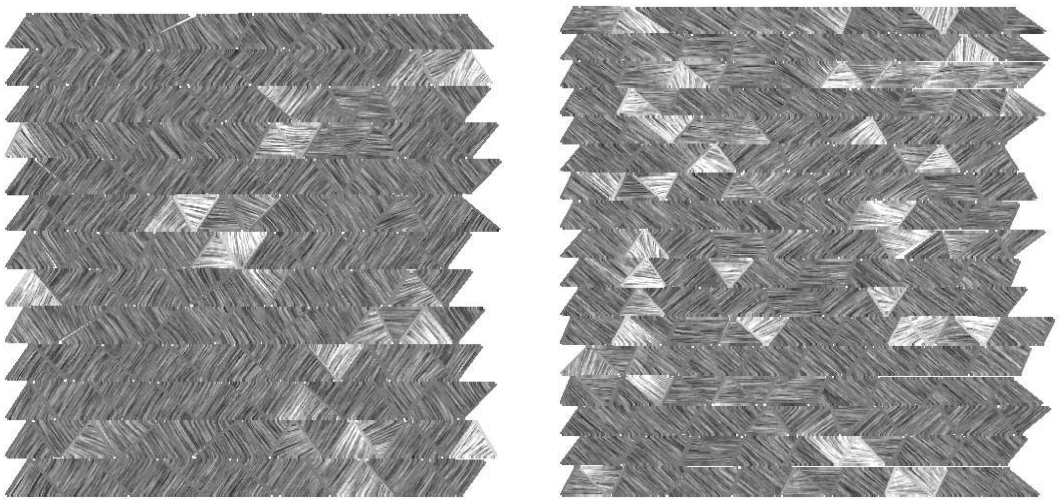


Figure 7: Packing of local triangle textures: To make optimal use of texture memory similar triangle patches are arranged in rows.

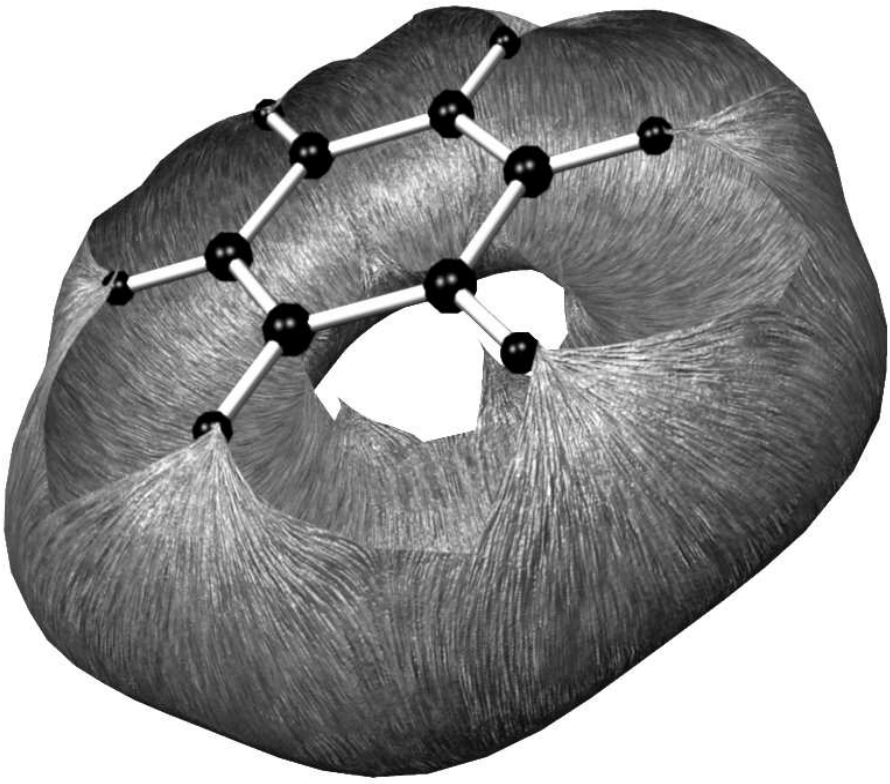


Figure 8: Example of a bump-mapped LIC texture on a surface. The surface contains field lines of the electrostatic field of a benzene molecule. It has been generated by applying an advancing front stream surface algorithm. The rendering was done using the Alias/Wavefront software.