Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustr. 7, D-14195 Berlin-Dahlem

Detlev Stalling Malte Zöckler Hans-Christian Hege

Fast Display of Illuminated Field Lines

Preprint SC-96-58 (December 1996)

Fast Display of Illuminated Field Lines

Detlev Stalling   Malte Zöckler   Hans-Christian Hege

Abstract

A new technique for interactive vector field visualization using large numbers of properly illuminated field lines is presented. Taking into account ambient, diffuse, and specular reflection terms as well as transparency and depth cueing, we employ a realistic shading model which significantly increases quality and realism of the resulting images. While many graphics workstations offer hardware support for illuminating surface primitives, usually no means for an accurate shading of line primitives are provided. However, we show that proper illumination of lines can be implemented by exploiting the texture mapping capabilities of modern graphics hardware. In this way high rendering performance with interactive frame rates can be achieved. We apply the technique to render large numbers of integral curves of a vector field. The impression of the resulting images can be further improved by a number of visual enhancements, like transparency and depth-cueing. We also describe methods for controlling the distribution of field lines in space. These methods enable us to use illuminated field lines for interactive exploration of vector fields.

Preprint SC-96-58 (December 1996)

Contents

1 Introduction

The visual representation of vector fields is subject of ongoing research in scientific visualization. A number of sophisticated methods have been proposed to tackle this problem, ranging from particle tracing [8, 18, 12] over icon based methods [9, 14] to texture based approaches [3, 2, 4, 17, 10]. A straightforward, powerful and therefore popular technique is to depict field lines also called integral curves or stream lines. However, using this method the user is confronted with the following problems. First, on common graphics workstations field lines either have to be displayed using flat-shaded line segments, impairing the spatial impression of the image, or they have to be represented by polygonal tubes, strongly limiting the number of field lines that can be displayed in a scene. Second, it is usually not quite obvious how to distribute field lines in space in order to get expressive pictures without missing important details of the field. In this paper we present ideas that can help to overcome both problems.

It is a well-known fact that quality and realism of computer generated images depend to a high degree on the accurate modeling of light interacting with the objects in a scene. Shading effects are perhaps the most important cues for spatial perception. Consequently much research has been performed to develop realistic illumination and reflection models in computer graphics. A widely used compromise between computational complexity and resulting realism is Phong's reflection model [13] which assumes point light sources and approximates the most important reflection terms by simple expressions. Traditionally this model is applied to surface elements. Today many graphics workstations offer hardware support for this kind of illumination.

The shading model can also be generalized to line primitives in $\mathbb{R}^3$. In this paper we will make direct use of such a generalization. However, on current graphics workstations there is no direct hardware support for display of Phong-shaded line primitives. We achieve a fast and accurate illumination of line segments by exploiting texture mapping capabilities of modern graphics hardware. Applying this new shading technique large numbers of field lines distributed throughout the data volume can be rendered. Taking light reflection on line primitives into account increases significantly spatial impression of the resulting images, and therefore is of particular significance for scientific visualization. Image quality can be further improved by drawing semi-transparent field lines and employing depth cueing. This allows the user to get a better understanding of the spatial structure of a field. Transparency and animation of field lines also make it possible to distinguish between forward and backward direction of the field vectors.

The large number of field lines that may be displayed simultaneously also facilitates their placement. We employ statistical methods to select seed points. Given some scalar quantity that loosely describes the degree of interest in the vector field at some location, field lines are placed automatically such that the relative degree of interest is matched qualitatively. This is possible for user-selected spatial subvolumes as well as two and one dimensional sub-manifolds.

In scientific visualization the goal is not to render natural scenes in a photo-realistic way, but to generate images which provide maximal insight into numerical or experimental data. Nevertheless, shading effects are at least as important for the spatial interpretation of artificial images as in traditional computer graphics. Shading provides the observer with a minimum of realism in a world of cutting planes, isosurfaces, and symbols. Unfortunately there are a number of visualization techniques which aren't based on surface primitives, and which therefore can't make use of the hardware shading capabilities of current graphics workstations. As an example consider the various volume rendering techniques. While interactive frame rates can be achieved for simple emission-absorption models by exploiting graphics hardware, in general this isn't yet possible if some sort of gradient dependent shading is included. Although rendering of line primitives is not as complex as volume rendering, the situation is similar. Traditionally, either flat shading has to be used or significant parts of the illumination calculation have to be computed without support by dedicated hardware.

After discussing illumination of line primitives in more detail, we show in section 3 how it can be implemented using texture mapping techniques. In section 4 we describe several visual enhancements, like use of color, transparency, and depth cueing, as well as animation of field lines. In the next section some aspects of numerical field line integration and interpolation are explained. In section 6 we show how to distribute field lines in space in order to enhance interesting features of a vector field. In the final sections we present results and conclusions.

2 Illumination of Lines in $\mathbb{R}^3$

Surfaces can be characterized locally by a distinct outward normal vector $\boldsymbol{N}$. This normal vector plays an important role when describing the interaction of light with surface elements. In the following we will shortly review the popular reflection model of Phong. Let $\boldsymbol{L}$ denote the light direction, $\boldsymbol{V}$ the viewing direction and $\boldsymbol{R}$ the unit reflection vector (the vector in the $\boldsymbol{L}$-$\boldsymbol{N}$-plane with the same angle to the surface normal as the incident light). Then light intensity at a particular surface point is given by

$$\begin{aligned} I &= I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}} \\ &= k_a + k_d\, \boldsymbol{L}\cdot\boldsymbol{N} + k_s\, (\boldsymbol{V}\cdot\boldsymbol{R})^n \end{aligned} \tag{1}$$

The first term, a global one, represents the ambient light intensity due to multiple reflections in the environment. The second term describes diffuse reflection due

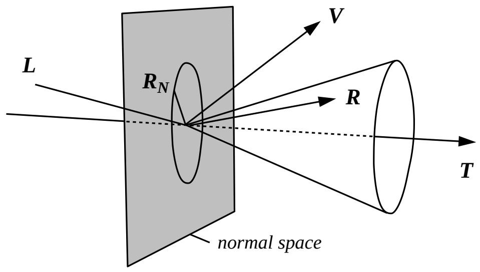


Figure 1: For line primitives there are infinitely many possible reflection vectors $\boldsymbol{R}$ lying on a cone around $\boldsymbol{T}$. For the actual lighting calculation we choose the one contained in the $\boldsymbol{L}$-$\boldsymbol{T}$-plane.

to Lambert's law. Diffuse light intensity does not depend on the viewing vector, i.e. diffuse reflecting objects look equally bright from all directions. The last term in Eq. (1) describes specular reflections on a surface. Specular reflections or highlights are centered around the reflection vector $\boldsymbol{R}$. The width of the highlights is controlled by the exponent n, also called shininess.

Let us now consider line primitives. In this case we can no longer define unique normal and reflection vectors. Instead there are two-dimensional manifolds containing infinitely many possible normal and reflection vectors. Mathematically lines in $\mathbb{R}^3$ are said to have codimension 2. Fortunately common surface reflection models can be generalized to higher codimensions in a straightforward way. These generalizations have been discussed in detail by Banks [1]. For lines in $\mathbb{R}^3$ the results are quite obvious. From all possible normal vectors we simply have to select the one which is coplanar to the light vector $\boldsymbol{L}$ and the tangent vector $\boldsymbol{T}$. Taking this particular normal vector we compute the diffuse reflection term as for surfaces using Eq. (1). Likewise, from all possible reflection vectors we choose the one coplanar to $\boldsymbol{L}$ and $\boldsymbol{T}$. Again, taking this particular reflection vector we use Eq. (1) to compute the specular reflection term. The relevant vectors for line illumination are illustrated in Fig. 1.

Instead of relying on a specially selected and explicitely calculated normal vector we would rather like to express diffuse light intensity for line segments solely in terms of $\boldsymbol{L}$ and $\boldsymbol{T}$. Therefore we first project the light vector into the line's normal and tangent spaces, yielding an orthogonal decomposition $\boldsymbol{L} = \boldsymbol{L}_N + \boldsymbol{L}_T$. As illustrated in Fig. 2, by applying Pythagoras's theorem we obtain

$$\boldsymbol{L}\cdot\boldsymbol{N} = |\boldsymbol{L}_N| = \sqrt{1 - |\boldsymbol{L}_T|^2} = \sqrt{1 - (\boldsymbol{L}\cdot\boldsymbol{T})^2}. \qquad (2)$$

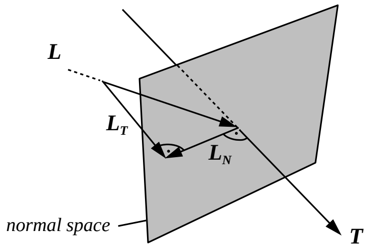


Figure 2: The light vector $\boldsymbol{L}$ can be decomposed into two orthogonal components $\boldsymbol{L}_T$ and $\boldsymbol{L}_N$ corresponding to the projection on the line's tangent and normal space, respectively.

Using similar arguments we can express the inner product $\boldsymbol{V}\cdot\boldsymbol{R}$ responsible for specular reflection solely in terms of $\boldsymbol{L}$, $\boldsymbol{V}$, and $\boldsymbol{T}$, i.e. without refering to $\boldsymbol{N}$. First, observe that $\boldsymbol{R}_N = -\boldsymbol{L}_N$ and $\boldsymbol{R}_T = \boldsymbol{L}_T$. We therefore have

$$
\begin{aligned}
\boldsymbol{V}\cdot\boldsymbol{R} &= \boldsymbol{V}\cdot(\boldsymbol{L}_T - \boldsymbol{L}_N) \\
&= \boldsymbol{V}\cdot((\boldsymbol{L}\cdot\boldsymbol{T})\boldsymbol{T} - (\boldsymbol{L}\cdot\boldsymbol{N})\boldsymbol{N}) \\
&= (\boldsymbol{L}\cdot\boldsymbol{T})(\boldsymbol{V}\cdot\boldsymbol{T}) - (\boldsymbol{L}\cdot\boldsymbol{N})(\boldsymbol{V}\cdot\boldsymbol{N}) \\
&= (\boldsymbol{L}\cdot\boldsymbol{T})(\boldsymbol{V}\cdot\boldsymbol{T}) - \\
&\qquad \sqrt{1-(\boldsymbol{L}\cdot\boldsymbol{T})^2}\sqrt{1-(\boldsymbol{V}\cdot\boldsymbol{T})^2}. \qquad (3)
\end{aligned}
$$

Here we have replaced $\boldsymbol{L}\cdot\boldsymbol{T}$ by Eq. (2). A similar expression has been used to rewrite $\boldsymbol{V}\cdot\boldsymbol{T}$.

3 Rendering Illuminated Lines

Despite the fact that the illumination equation looks the same for lines and surfaces, use of standard hardware shading techniques is impaired because for each new view or light direction a suitable normal vector has to be computed without utilizing graphics hardware. In the following we show how Eqs. (2) and (3) can be effectively evaluated using texture mapping capabilities of modern graphics hardware, thereby avoiding explicit normal vector computation. The technique allows us to achieve high frame rates even when large numbers of line segments have to be rendered.

3.1 Texture Mapping

We assume to have a graphics API available similar to OpenGL. In this graphics library at each vertex a homogeneous vector of texture coordinates can be specified. Usually the first components of this vector are taken as indices into a one-, two-, or three-dimensional texture map. A texture map may contain colors and/or transparencies which can be used to modify in various ways the original color of a fragment in the graphics pipeline. In addition it is possible to change texture coordinates using a 4×4 texture transformation matrix. This texture transformation is the key feature which makes it possible to employ texture mapping hardware for shading calculations.

3.2 Diffuse Reflection

Looking at Eq. (2) we note that the diffuse light intensity of a line segment is a function of $\boldsymbol{L}\cdot\boldsymbol{T}$ only. Specifying a texture vector $\boldsymbol{t}_0$ equal to the line's tangent vector $\boldsymbol{T}$ at each vertex, this inner product can be computed in hardware using the following texture transformation matrix:

$$M = \frac{1}{2} \begin{pmatrix} \boldsymbol{L}_1 & 0 & 0 & 0 \\ \boldsymbol{L}_2 & 0 & 0 & 0 \\ \boldsymbol{L}_3 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 \end{pmatrix}$$

The first component of the transformed homogeneous texture vector $\boldsymbol{t} = \boldsymbol{t}_0 M$ then evaluates to

$$t_1 = \frac{1}{2}(\boldsymbol{L}\cdot\boldsymbol{T} + 1).$$

Note, that t_1 always lies in the range $0 \ldots 1$. Therefore this value can be used as an index into a one-dimensional texture map $P(t_1)$. The value of the texture map at location t_1 is chosen such that it resembles the diffuse light intensity corresponding to $\boldsymbol{L}\cdot\boldsymbol{T} = 2t_1 - 1$, namely

$$P(t_1) = I_{\text{diffuse}} = k_d \sqrt{1 - (2t_1 - 1)^2}. \qquad (4)$$

Using a texture mode which takes the color of a line fragment to be equal to its texture color $P(t_1)$ we obtain an image which accurately shows line segments diffusely illuminated by a single point light source. If the light direction changes we simply have to update the texture transformation matrix. Vertices and texture coordinates of the line segments remain constant. This means that we can make use of OpenGL display lists to further increase rendering speed. Display lists allow one to specify multiple vertex and texture definitions using a single graphics library call.

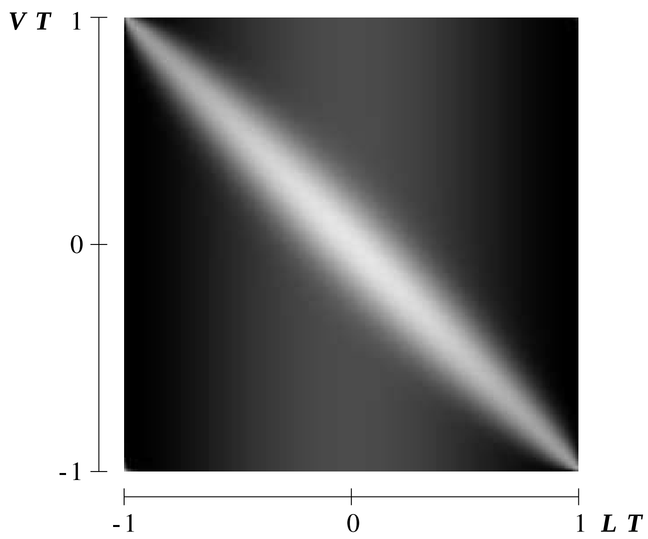


Figure 3: Two-dimensional texture map used to implement Phong's reflection model for line segments. Parameter values are $k_a = 0.1, k_d = 0.3, k_s = 0.6$, and $n = 40$.

3.3 Specular Reflection

The specular reflection term does not only depend on $\boldsymbol{L}\cdot\boldsymbol{T}$ but also on $\boldsymbol{V}\cdot\boldsymbol{T}$, as can be seen from Eq. (3). To compute this additional inner product we initialize the second column of the texture transformation matrix with the current viewing direction:

$$M = \frac{1}{2}\begin{pmatrix} \boldsymbol{L}_1 & \boldsymbol{V}_1 & 0 & 0 \\ \boldsymbol{L}_2 & \boldsymbol{V}_2 & 0 & 0 \\ \boldsymbol{L}_3 & \boldsymbol{V}_3 & 0 & 0 \\ 1 & 1 & 0 & 2 \end{pmatrix}$$

While the first transformed texture component remains the same, for the second component we now get

$$t_2 = \frac{1}{2}(\boldsymbol{V}\cdot\boldsymbol{T} + 1).$$

In order to obtain the correct light intensity corresponding to $\boldsymbol{L}\cdot\boldsymbol{T} = 2t_1 - 1$ and $\boldsymbol{V}\cdot\boldsymbol{T} = 2t_2 - 1$ we use a two-dimensional texture map $P(t_1, t_2)$. Adding a constant ambient term k_a as well as the diffuse contribution from Eq. (4) we can perform the whole shading calculation for a single light source in texture hardware. Fig. 3 shows an example of a resulting two-dimensional texture map. One can clearly identify the highlight appearing at different angle positions on top of a diffuse background. If no highlight were present color would not depend on the viewing direction $\boldsymbol{V}$, as stated by Lambert's law.

It is worthwhile to note that there is an important special case, which allows one to use a one-dimensional texture even when specular reflection is present. This is the case of a headlight, i.e. a point light source located at the same position as the camera. In this case light vector and viewing vector are identical. Equation (3) simplifies to

$$\boldsymbol{V}\cdot\boldsymbol{R} = 2(\boldsymbol{L}\cdot\boldsymbol{T})^2 - 1.$$

Headlights are quite useful because they always guarantee an adequate illumination of the scene, irrespectively of the actual viewing direction. The user has not to bother with a tedious setup of light conditions. However, as will be shown later, also situations occur where other light positions are favourable.

Of course it is also possible to use the third column of the texture transformation matrix to compute an additional inner product. This would require the use of a three-dimensional texture map. Three different inner products would allow the illumination of lines by two point light sources located at arbitrary positions including specular reflection. Alternatively one might discard specular reflection and instead introduce a third purely diffuse illuminating light source.

3.4 Excess Brightness

Banks [1] pointed out that there is a general problem when illuminating objects with codimension > 1. The overall intensity of an image increases and becomes more uniform, thus disturbing spatial perception. In case of lines in $\mathbb{R}^3$ this can be understood by the following consideration: We know that the normal vector is not a constant one, but is given by the projection of the light vector into the line's normal space. Choosing such a vector means minimizing the angle between light vector and normal. Therefore in general the angle between these two vectors is smaller compared to the case of a fixed normal. This results in a more uniform brightness than we are used to perceive in real world. As suggested by Banks, we compensate the effect qualitatively by exponentiating the diffuse intensity term:

$$\hat{I}_{\text{diffuse}} = k_d\ (\boldsymbol{L}\cdot\boldsymbol{N})^p \tag{5}$$

In [1] a value of $p = 4.8$ was proposed. For the images in this paper we have used a value of $p = 2$, which produced nicer results.

4 Visual Enhancements

There are a number of ways to enhance and modify the rendering of illuminated field lines as discussed in section 3. With color coding it is possible to depict an additional scalar quantity. Transparency can either be used to draw anti-aliased line primitives, to highlight particular regions in space, or to encode the directional sign of a field line. Animation is an even better way to indicate the orientation of a field line. Finally depth cueing can be used to further improve the understanding of complex spatial structures. In the following we will discuss all these topics in more detail.

4.1 Color

Color coding is a common method in visualization. Applying color to individual field lines would enable us to depict some scalar quantity in addition to vector field structure. Such a quantity could be field magnitude or potential strength, or something more unrelated scalar variable like pressure in a fluid flow. Ideally we would like to modify the curve's ambient and diffuse color components according to a given color lookup table. However, in our case color is directly taken from a texture map. Since we use the same texture map for all field lines it is not possible to set these components locally in a straight-forward way. Nevertheless, by using an alternative texture mapping mode it is possible to modulate, i.e. multiply, texture color with the object's base color. The latter can be defined for each vertex separately. This yields the desired effect with the restriction that also the specular highlight gets colored instead of remaining constant. Fig. 4 suggests that this is only a minor limitation. Despite being differently colored the highlights can be identified clearly throughout the whole image while still improving spatial perception. At the same time color accurately encodes an additional scalar variable.

4.2 Transparency

Transparency is a powerful concept which can be utilized in a number of ways. However, it requires geometric primitives to be rendered in a depth-sorted way. We will first discuss some applications of transparency, before we describe how to deal with the depth-sorting problem.

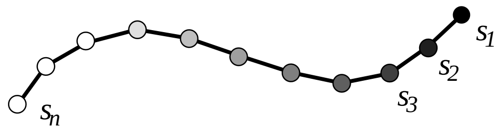
Anti-aliasing. Lines on a raster display may appear rather jagged, if a binary scan-conversion algorithm is used. These alias effects can be suppressed effectively by rendering pixels which are covered only partially by a line with an opacity proportional to the actual amount of overlap. This causes the final pixel color to be a mixture of the line's color and the color of the underlying object. Anti-aliasing of lines is directly supported in OpenGL. It improves image quality significantly. Jags tend to appear at different locations in sucessive frames with slightly different view directions. Since this is quite disturbing anti-aliasing is even more important for interactive applications and animations.

Highlighting. Transparency can be used to highlight important features of a vector field. Like with color in our application we can use an independent scalar field to define the transparency of a field line at each vertex. In Fig. 10 a simple model of the magnetic field around the earth is shown. While the field of the earth itself is assumed to be a magnetic dipole, the field of the sun is approximated by a constant term. Choosing transparency proportional to the logarithm of vector field magnitude reveals the characteristic dipol structure and still lets you discern the constant field in the outer region.

Directional sign. To encode unambiguously the directional sign of a vector field imagine small particles traversing the vector field and leaving a veil of haze. For a stationary field the particles will just follow the field lines. Assuming that the haze disappears according to an exponential law, opacity or alpha values for equidistant points s_n of a field line are given by

$$\alpha(s_n) = \alpha_0 q^{n-1}. \tag{6}$$

Here the factor q controls how much of the haze disappears per unit step. A resulting semi-transparent field line is illustrated in the following figure:



The sign of vector field direction would not become visible if field lines were rendered symetrically in forward and backward direction. Fig. 8 ilustrates the use of transparency to encode the directional sign of a field line. The figure also compares flat shaded and illuminated field lines.

Depth sorting. Drawing a transparent pixel of opacity α and color C causes the current color in the frame buffer to be updated according to

$$C_{\text{new}} = (1 - \alpha)C_{\text{old}} + \alpha C. \tag{7}$$

In general if multiple transparent objects are present the final color depends on the ordering of the individual objects. Correct results are obtained using a back to front traversal. The situation is simplified if all objects are of equal color C. In this case all traversal orders yield the same result. This has been exploited by Max, Crawfis, and Grant [11], who applied flat shaded line bundles for vector field visualization. However, for illuminated lines color isn't constant. Therefore individual lines have to be rendered in a depth-sorted way.

In general it is impossible to achieve an exact depth ordering for extended curves in 3D, because mutual coverings may occur. Therefore we split each field line into many small line segments, which are sorted and rendered individually. To avoid resorting line segments each time the view direction changes we use the following simplified algorithm: Three lists of pointers to field line segments are created. The lists are sorted in order of increasing x-, y-, and z-coordinates, respectively. During rendering the list that most closely resembles the viewing direction is traversed, either from back to front or from front to back. Although this method is not exact, it produces excellent results which can not be distinguished visually from the exact images. Experiments have shown, that typically only about 1% of all pixels receive somewhat incorrect color values.

4.3 Field Line Animation

Animated particles provide a very intuitive mean of visualization, especially when velocity fields are to be visualized. Following the idea of particles leaving a veil of haze, animation sequences can be obtained in the following way.

Stream lines are created at different times t_i with an initial length of 0. In each time step, all field lines are extended by one point, while opacity of all the points already drawn is modified by the factor q, c.f. Eq. (6). This gives the illusion of moving particles producing a slowly disappearing veil of haze, like comets. A periodic animation sequence can be created by assuring that the period T is long enough so that points on a field line can disappear completely within this interval (i.e. $q^T \approx 0$). Then a field line that has been created at time t_i can be restarted at the same location at time $t_i + T$, since it is no longer visible then. This results in a continuous animation loop of period T.

4.4 Depth Cueing

Depth cueing is a well-established technique to improve the spatial perception of complex three-dimensional scenes. It means that the color of objects is adjusted according to their distance from the camera. The underlying idea is that bright objects of high contrast usually appear closer than dark or washed-out ones. In our context the method turns out to be especially useful in situations where illuminated field lines are arranged such that they constitute surface-like structures. Since the illumination model does not distinguish between inside and outside, sometimes the spatial structure of such pseudo-surfaces may not be clear on the first sight. Depth cueing helps to identify near and far parts of the geometry and therefore improves spatial perception. This becomes obvious by comparing the middle and right image of Fig. 9.

5 Field Line Integration

For numerical field line integration we use a fourth-order Runge-Kutta method with error monitoring and adaptive step size control, as described in [17]. Use of an adaptive method allows us to control the error of the solution. Such methods are also necessary to detect singularities. At these points field line integration has to be terminated. Singularities, i.e. sinks and sources, commonly occur for example in electrostatic fields. Examples are shown in Figs. 9 and 7.

A common problem with adaptive integrators is that the step sizes usually are so large that the resulting curve cannot be approximated by straight line segments anymore. Instead additional samples have to be computed between the solution vectors. Additional samples are obtained by evaluating an interpolation polynomial whose degree should conform to the order of the integrator. For a fourth-order integrator cubic Hermite interpolation is an appropriate choice [7]. It retains the exact values of the tangent vectors at the endpoints of an interval in addition to the location of the endpoints itself. The tangent vectors itself are interpolated by a quadratic polynomial. To quickly evaluate the interpolation polynomials at equidistant steps we are using forward differences.

The step size of an adaptive integrator is closely related to the curvature of the solution curve. In areas of high curvature small steps are taken, while in areas of little curvature large steps are taken. Therefore it makes sense to subdivide each interval using an equal number of intermediate samples. The number of intermediate samples determines how closely the original geometry is approximated. We obtained good results with 3-5 intermediate samples.

Instead of precomputing intermediate samples during field line integration it is also possible to defer evaluation of the interpolation polynomials until rendering. OpenGL provides means for sampling a polynomial curve at equi-distant locations. Instead of specifying vertices or texture coordinates directly, so-called evaluators can be used to compute these quantities from a polynomial. For this the curve has to be specified in terms of linearly independent Bernstein polynomials. For a cubic curve the Bernstein polynomials are $(1-u)^3$, $3(1-u)^2u$, $3(1-u)u^2$, and u^3. The advantage of letting OpenGL interpolate the intermediate samples is that the number of subdivisions can be easily changed without any recomputation. For example it would be possible to adjust the resolution to the current view (level of detail). However, in general faster rendering times are obtained by computing the intermediate samples in a preprocessing step and then drawing the field lines directly as straight line segments. Only for a large number of subdivisions (> 25) OpenGL's evaluator interface becomes preferable.

In some situations it is also desirable to render the field lines with equidistant samples. For example if an independent scalar field is used to obtain color or opacity values for each vertex, the required resolution doesn't depend on field line curvature alone, but also on the characteristics of the scalar field. If nothing is known about this field, using equidistant samples seems to be the best choice.

6 Seed Point Selection

The proper choice of seed points for field line integration is a common problem in vector field visualization. On the other hand the fast texture based rendering technique described above allows us to generate images with thousands of lines at interactive rates. This means that the positioning of an individual field line becomes less important. This allows us to apply statistical methods for distributing seed points in the data volume.

It should be mentioned that actually the distribution of the field lines itself should be controlled rather then the distribution of seed points. If the directional field has a non-vanishing divergence, field line density will not remain constant. Instead field lines will run together in some areas, resulting in an increased local density, or they will expand in other areas, resulting in a decreased local density. These variations are less dominant if the total length of a line is limited and if the field lines are integrated an equal distance in forward and backward direction. Using this approach, we obtained reasonable results with just controlling seed point density instead of field line density. More elaborated strategies compute an optimized distribution by taking into account some kind of repulsion between different lines, as described for the 2D case in [16]. However, this is a rather expensive computational process.

6.1 The Interest Function

Often it is intuitive to have a field line distribution proportional to some scalar quantity p. Such a field p may be interpreted as the degree of interest the user wants to put in a region. For example a constant p would result in a homogenous distribution of seed points, while a value of p proportional to vector magnitude would emphasize regions of large field strength.

In general it is not a trivial task to find a good interest function p. For example, in electrostatic data sets field strength often varies over several orders of magnitude. Instead of choosing p exactly proportional to field strength we would rather like to have a more homogenous distribution which resembles vector field magnitude only qualitatively. Such an effect can be obtained using a histogram equalization approach. This technique is well known from the image processing literature [6], but in our case may also be used to modify the degree of interest p in a suitable way. Let us define a sum histogram in the following way:

$$S(p) = \frac{\text{number of cells with } p_i < p}{\text{total number of cells}} \tag{8}$$

Based on the sum histogram we can assign each cell a new equalized degree of interest p_i' by

$$p_i' = S(p_i). \tag{9}$$

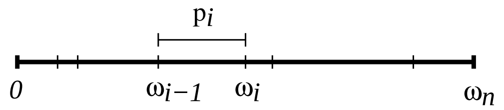
Of course other probability distributions can be useful to emphasize special features of the field. We have implemented a symbolic interface which allows us to specify p_i as a function of any given set of scalar fields. Within this interface functions like logarithm or square root as well as threshold operators can be used to specify the degree of interest p.

6.2 Seed Volumes

To actually generate seed points with a given density p, we subdivide a box-shaped seed volume into n uniform cells. For each cell we compute a value p_i describing the local degree of interest for that cell. The accumulated degree of interest is defined by

$$\omega_i = \sum_{j=1}^{i} p_j \ . \tag{10}$$

We assume all cells being arranged in a sequence based on some arbitrary numbering. We choose cells randomly with a probability proportional to p_i. This is done by taking a random number r uniformly distributed in the range $0 \ldots \omega_n$. The first value $\omega_i > r$ determines which cell is taken.



Within a selected cell we place a new seed point at a random position. Because the values ω_i are monotonely increasing, the cell lookup procedure has a complexity of $O(\log(n))$ and therefore can be performed quite fast.

To get an overview of the global field structure, it is a good idea to choose an initial seed volume that fully encloses the data set. The user then may further constrain the seed volume. For this we use so-called dragger provided by the Open Inventor graphics toolkit. Draggers are interactive components that may be translated and rescaled directly in a three-dimensional scene. They provide a very flexible and intuitive interface. An example of an Inventor-style selection box is shown in Fig. 5.

6.3 Seed Surfaces

Instead of using seed volumes, field lines may also be started on two-dimensional manifolds. For example to visualize the structure of an electrostatic field seed points may be distributed on an iso-potential surface. Since an electrostatic field is always oriented perpendicular to an iso-potential surface, this method is of particular interest. It may reveal important features of the field.

We assume a surface S to be given by a set of triangles. To distribute n_p seed points homogenously on S, we first determine how many point have to be placed in a particular triangle T_i of the surface. The larger a triangle, the more seed points it should contain, i.e.

$$n_i = \frac{Area(T_i)}{Area(S)}\, n_p \;. \tag{11}$$

We always generate at least $\lfloor n_i \rfloor$ points in each triangle T_i. To deal with non-integer numbers n_i an additional point is generated with a probability given by the fractional part $n_i - \lfloor n_i \rfloor$. Again seed point density may also be chosen proportional to some interest function $p(\boldsymbol{x})$ defined on the manifold. In this case equation (11) has to be replaced by

$$n_i = \frac{\int_{T_i} p(\boldsymbol{x})dA}{\int_S p(\boldsymbol{x})dA}\, n_p \;. \tag{12}$$

Inside a triangle all points are distributed uniformly. This is a reasonable approximation also for non-constant density function $p(\boldsymbol{x})$, if the triangles are small enough. For iso-surfaces obtained by a marching-cubes algorithm this is usually the case; there the number of triangles typically is even much larger than the number of seed points. To obtain a uniform point distribution within a triangle, we choose two uniform numbers u and v in the range $[0..1]$. If $u + v > 1$ then $u := 1 - u$ and $v := 1 - v$ are used instead. The final position of a point inside a triangle is given by $\boldsymbol{P} = \boldsymbol{A} + u(\boldsymbol{B} - \boldsymbol{A}) + v(\boldsymbol{C} - \boldsymbol{A})$, where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ are the vertices of the triangle. If the final distribution is not sufficiently homogeneous, the points can be moved on the surface using a point-repulsion approach, as described in e.g. [15]. An example of field lines placed uniformly on iso-potential surfaces is shown in Fig. 6. The image reveals the structure of the electrostatic field of a water molecule.

Instead of placing seed points on complex geometries like iso-surfaces we have also implemented a small surface probe, wich can be moved and rotated interactively within the scene. Again we utilized Open Inventor components. An example of a surface probe together with a box-style dragger is shown in Fig. 5.

6.4 Seed Lines

Another way for seed point selection is to start field lines on one dimensional manifolds, i.e. lines. Instead of using simple geometric primitives like straight line segments or circles such manifolds may be defined conveniently by intersecting some other three-dimensional geometry with a cutting plane. Again iso-surfaces are suitable candidates for such an operation. We use draggers defined in Open Inventor to implement a cutting plane which can be translated and rotated easily in a three-dimensional scene. Open Inventor also provides a so-called triangle callback action which allows us to actually compute the intersection.

An example of a set of field lines started along a one-dimensional manifold defined by an intersection operation is depicted in Fig. 9. This way of placing seed points is especially suited to emphasize possible symmetries in a field. The image also shows the effect of depth cueing as discussed in section 4.4.

7 Implementation and Results

The algorithms presented in this paper have been implemented in C++ by subclassing the Open Inventor toolkit. Using Inventor makes it easy to display illuminated field lines in combination with other geometries. The rendering code itself is built on top of the OpenGL graphics library. It is embedded into an object-oriented visualization system developed at ZIB. The object-oriented design allows us to process 3D vector fields defined in various ways using the same interface. Examples are analytically defined fields (Fig. 10), fields defined on curvilinear grids (Fig. 4) or fields on regular grids as in the molecular datasets.

We have applied our methods to visualize vector fields from various disciplines like computational fluid dynamics, quantum chemistry, and astro physics. In most cases the default values for seed point distribution provide a good first impression of the vector field. The fast rendering speed offers the possibility to interactively rotate and zoom the geometry. This is an important feature for understanding the complex vector field structures.

Fig. 4 shows the air flow around a wing, obtained from a CFD simulation. The vector field is defined on a curvilinear grid. Color is used to encode velocity magnitude. Blue depicts regions of high velocity, while yellow and red show slowly flowing parts of the field. The scene contains 14,200 line primitives and can be rendered at a frame rate of 25 frames per second. All performance measurements have been done on an SGI Indigo^2 with Maximum Impact Graphics and 250 MHz R4400 CPU. Performance gains of about 10-20% can be achieved by using OpenGL display lists. In a display list multiple graphics commands are compiled into a single function call. Due to the depth-sorting approach in our case 6 independent display lists have to be generated. Therefore the start-up time needed to generate the lists as well as the memory overhead usually outweighs the performance gain.

In Fig. 10 a simple model of the magnetic field around the earth is shown. This field is given analytically. Transparency modulation is used to enhance the dipole structure of the earth field, as described in Sec. 4.2 (27,000 lines, 15 fps).

Fig. 7 shows the electrostatic field of a benzene molecule. The field is computed using the NAO-PC method (Natural Atomic Orbitals - Point Charge). This quantum-classical method aproximates atomic orbitals by a set of discrete fractional point charges. The location of some of these point charges can be clearly identified in the images (19,100 lines, 18 fps).

An example of a velocity field from a CFD application is shown in Fig. 5. The data represents a fluid flow over a backward facing step. The turbulent region behind the step is characterized by a very complex field structure. A 3D dragger is used to highlight this part of the field. In addition some field lines are seeded on a probe-surface (25,000 lines, 15 fps).

8 Conclusion

The visual representation of 3D vector fields is one of the current challenges in scientific visualization. Of particular interest are methods that provide an overview of the global field structure and that also depict fine details.

In this paper we have presented a fast method for visualizing 3D vector fields based on the display of field lines, i.e. integral curves of the field. The method gives a good impression of the field structure and enables us to resolve visually rather fine details, like small vortices. A texture mapping technique is used to accurately illuminate the field lines. Light reflection on field lines improves spatial perception and thereby facilitates the understanding of the inner structure of a field.

We have shown how high quality field line images can be generated at interactive speed using hardware supported texture mapping. This offers new opportunities for interactive visualization. Using a simple Monte-Carlo method lines are placed automatically such that the relative degree of interest, defined by some scalar field, is matched qualitatively.

Some interesting topics of further research are improvement of the seed point selection strategies such that characteristic features of the field are detected and enhanced automatically or the application of the shading technique to time dependent vector fields. In the latter case particle paths or streak lines should be used in favour of field lines.

References

[1] D.C. Banks, *Illumination in Diverse Codimensions*, Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics* Annual Conference Series, 1994, ACM SIGGRAPH, pp. 327-334.

[2] B. Cabral, L. Leedom, *Imaging vector fields using line integral convolution*, Proceedings of SIGGRAPH '93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics* 27, 1993, ACM SIGGRAPH, pp. 263-272.

[3] Roger Crawfis, Nelson Max, *Textured Splats for 3D Scalar and Vector Field Visualization*, Proceedings of Visualization '93, Nielson and Bergeron, Eds., IEEE Computer Society Press, 1993, pp. 261-272.

[4] L.K. Forsell, *Visualizing Flow over Curvilinear Grid Surfaces unsing Line Integral Convolution*, Proceedings of Visualization '94, Bergeron and Kaufman, Eds., IEEE Computer Society Press, 1994, pp. 240-247.

[5] Allen Van Gelder, Jane Wilhelms, *Interactive Animated Visualization of Flow Fields*, Proceedings of ACM Workshop on Volume Visualization, 1992, pp. 47-54.

[6] R.C. Gonzales, P. Wintz, *Digital Image Processing*, Addison Wesley, Second Edition, 1987, pp. 146–152.

[7] Ernst Hairer, Syvert Paul Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I, Nonstiff Problems*. Springer Verlag, Berlin, Heidelberg, New York, Tokyo, 1987.

[8] Andrea J. S. Hin and Frits H. Post, *Visualization of turbulent flow with particles*. In *Visualization '93*, IEEE Computer Society Press, pp. 46-52.

[9] W.C. de Leeuw, J.J. van Wijk, *A probe for local flow field visualization*, Proceedings of Visualization '93, Nielson and Bergeron, Eds., IEEE Computer Society Press, 1993, 39-45.

[10] W.C. de Leeuw, J.J. van Wijk, *Enhanced Spot Noise for Vector Field Visualization*, Proceedings of Visualization '95, Nielson and Silver, Eds., IEEE Computer Society Press, 1995, pp. 233-239.

[11] N. Max, R. Crawfis, C. Grant, *Visualizing 3D Velocity Fields Near Contour Surfaces*, Proceedings of Visualization '94, Bergeron and Kaufman, Eds., IEEE Computer Society Press, 1994, pp. 248-255.

[12] Kwan-Liu Ma and Philip J. Smith, *Virtual smoke: An interactive 3d flow visualization technique*, In *Visualization '92*, IEEE Computer Society Press, pp. 46-52.

[13] Bui-T. Phong, *Illumination for Computer Generated Pictures*, Communications of the ACM, June 1975, pp. 311-317.

[14] F.J. Post, T. van Walsum, F.H. Post, *Iconic Techniques for Feature Visualization*, Nielson and Silver, Eds., Proceedings of Visualization '95, pp. 288-295.

[15] G. Turk, *Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion*, Proceedings of SIGGRAPH '91 (Las Vegas, 28 July - 2 August). In *Computer Graphics* 25:4, ACM SIGGRAPH, pp. 289-298.

[16] G. Turk, D. Banks, *Image-Guided Streamline Placement*, Proceedings of SIGGRAPH '96 (New Orleans, Aug 4-9). In *Computer Graphics* Annual Conference Series, 1996, ACM SIGGRAPH, pp. 453-460.

[17] D. Stalling, H.C. Hege, *Fast and Resolution Independent Line Integral Convolution*, Proceedings of SIGGRAPH '95 (Los Angeles, California, August 6-11, 1995). In *Computer Graphics* Annual Conference Series, 1995, ACM SIGGRAPH, pp. 249-256.

[18] Jarke J. van Wijk, *Rendering surface-particles*, In *Visualization '92*, IEEE Computer Society Press, pp. 54-61.

wing5.85mm.eps

Figure 4: The image shows the flow around a wing. Color is used to encode the velocity magnitude.

stepSeedProbe.85mm.eps

Figure 5: Velocity field from a CFD simulation. Open Inventor draggers are used to define a seed volume (turbulent region) and a surface probe (upper part).

waterCombined.85mm.eps

Figure 6: Electrostatic field of a water molecule. Seed points are distributed uniformly on two iso-potential surfaces. A directional light shining from the upper right is used for illumination. If light and view direction were coincident, it would be hard to identify lines facing to the camera.

benzene.85mm.eps

Figure 7: Electrostatic field of a benzene molecule. The field line seeds were distributed proportionally to field strength.

waterTexTrans.eps

Figure 8: In the left image fully opaque, flat shaded field lines are drawn. In the middle and right images transparency is used to encode the directional sign of the field lines. Finally the right image shows the effect of proper illumination.

cutter.eps

Figure 9: A cutting plane is used to interactively define a cut through an iso-potential surface (left image). Seed points are placed on the resulting line. In the left and middle image depth cueing is used to improve depth perception.

magdipolRot2.85mm.eps

magdipolRot1.85mm.eps

Figure 10: Simple model of the magnetic field around the earth. In the right image transparency is used to emphasize the dipole structure of the earth's field, suppressing the constant contribution from the sun.