Interactive Visualization Of 3D-Vector Fields Using Illuminated Stream Lines

Malte Zöckler, Detlev Stalling, Hans-Christian Hege

Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)[1]

Abstract

A new technique for interactive vector field visualization using large numbers of properly illuminated stream lines is presented. Taking into account ambient, diffuse, and specular reflection terms as well as transparency, 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 in a vector field. The impression of the resulting images can be further improved by making the curves partially transparent. We also describe methods for controlling the distribution of stream lines in space. These methods enable us to use illuminated stream lines within an interactive visualization environment.

1 Introduction

The visual representation of vector fields is subject of ongoing research in scientific visualization. A number of sophisticated methods has been proposed to tackle this problem, ranging from particle tracing [7, 15, 11] over icon based methods [8, 13] to texture based approaches [3, 2, 4, 14, 9]. A straightforward, popular and still very powerful method is the concept of depicting stream lines. However, when using stream lines for visualization the user is confronted with a number of problems. First, on a common graphics workstation stream 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 stream lines that can be displayed in a scene. Second, it is usually not quite obvious how to distribute stream 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. To achieve a fast and accurate illumination of line segments we exploit the texture mapping capabilities of modern graphics hardware. We apply this new shading technique to render large numbers of stream lines distributed throughout a vector field. Taking into account light reflection on stream lines is of great significance for scientific visualization because it very much increases the spatial impression of the resulting images. Image quality can be further improved by making parts of a stream line semi-transparent. This allows us to get a better understanding of the inner structure of a field. It also makes it possible to distinguish between forward and backward direction. To facilitate the placement of a large number of stream lines we employ statistical methods. Given some scalar quantity that loosely describes the degree of interest in the vector field at some location, stream lines are placed automatically such that the relative degree of interest is matched qualitatively.

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 cue 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 [12] which assumes point light sources and approximates the most important reflection terms by simple expressions. Traditionally the model is applied to surface elements. Today many graphics workstations offer hardware support for this kind of illumination. However, the model can also be generalized to line primitives, and in this paper we will make direct use of such a generalization.

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

[1]Takustr. 7, D-14195 Berlin, Germany
E-mail: {zoeckler,stalling,hege}@zib-berlin.de

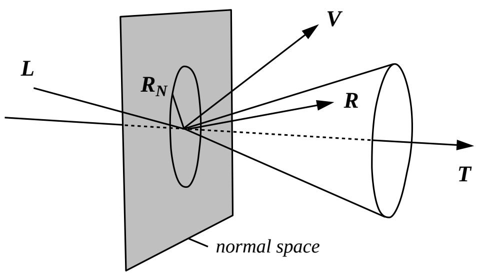


Figure 1: For line primitives there are infinitely many possible reflection vectors $\boldsymbol{R}$ lying on a cone around $\boldsymbol{T}$. For the actual lighting calculation we choose the one contained in the $\boldsymbol{L}$-$\boldsymbol{T}$-plane.

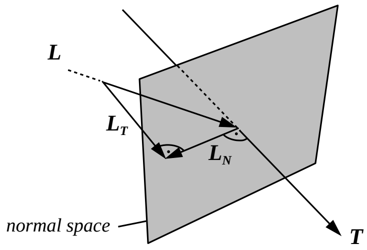


Figure 2: The light vector $\boldsymbol{L}$ can be decomposed into two orthogonal components $\boldsymbol{L}_T$ and $\boldsymbol{L}_N$ corresponding to the projection on the line's tangent and normal space, respectively.

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, in section 3 we show how it can be implemented using texture mapping techniques. In section 4 we describe how to distribute stream lines in space in order to enhance interesting features within a vector field. In the final sections we present results and conclusions.

2 Illumination of Lines

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 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 onto a specially selected 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

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

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

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

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

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, avoiding the need of 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

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

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.

3.3 Specular Reflection

The specular reflection term does not only depend on $\boldsymbol{V}\cdot\boldsymbol{T}$ but also on $\boldsymbol{L}\cdot\boldsymbol{T}$, 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 can 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 was 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. In fact, all of the color plates in this paper were rendered using a headlight.

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

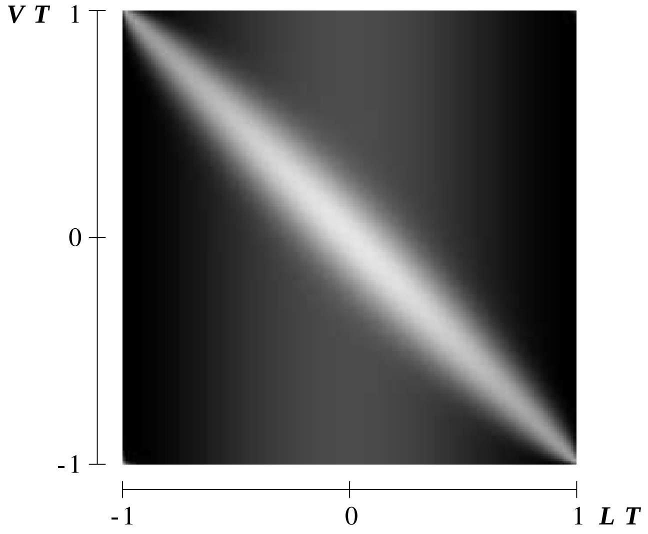


Figure 3: Two-dimensional texture map used to implement Phong's reflection model for line segments. Parameter values are $k_a = 0.1, k_d = 0.3, k_s = 0.6$, and $n = 40$.

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 Color Coding

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 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. 7 suggests that this is only a minor limitation. Despite being differently colored the highlight can be identified clearly throughout the whole image while still improving spatial perception. At the same time color accurately encodes an additional scalar variable.

3.5 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:

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

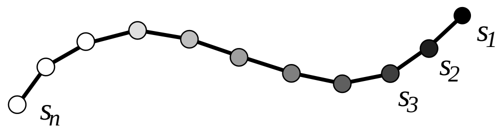
In [1] a value of $p = 4.8$ was proposed. For the images in this paper we have used a value of $p = 2$, which produced nicer results.

3.6 Transparency

Shading of line segments as described above provides important cues for the spatial impression of stream line images. However, image quality can be further improved by use of transparency. Let us imagine the image of a stream line is produced by a small particle traversing the vector field and leaving a veil of haze. Assuming that the haze disappears according to an exponential law, opacity or alpha value for a point s_n at the curve is 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 stream line is illustrated in the following figure:



Use of transparency has two advantages: First, stream lines near to the camera do not completely hide those being more far away. This allows the observer to gain deeper insight into the inner structure of the vector field. Second, the sign of vector field direction becomes visible in a static image. This is not the case when stream lines are rendered symetrically in forward and backward direction.

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 [10], who applied constant shaded line bundles for vector field visualization. However, for illuminated lines color isn't constant anymore. 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 stream 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 stream 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 from the exact images visually. Experiments have shown, that only about 1% of all pixels receive somewhat incorrect color values.

3.7 Stream 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 stream lines are extended by one point, while opacity of all the points already drawn is modified by the factor q (compare Eq. (7)). 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 stream line can disappear completely within this interval (i.e. $q^T \approx 0$). Then a stream 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 Distributing Stream Lines

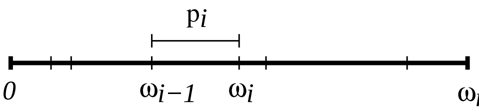
When using stream lines for vector field visualization a common problem is to select proper seed points for path tracking. The fast texture based shading technique described above allows us to render images containing thousands of stream lines at interactive rates. Working with a large number of stream lines has the advantage, that the positioning of an individual line becomes less important. Instead we can apply statistical methods to distribute seed points throughout the field. In particular we would like the distribution to resemble some sort of scalar quantity p, which loosely corresponds to the degree of interest the user wants to put in some 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 have the effect that more seed points are placed in regions of large magintude.

4.1 Monte-Carlo Selection

To generate seed points with a density proportional to p, we subdivide to whole data 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

$$\omega_i = \sum_{j=1}^{i} p_j \tag{8}$$

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.

4.2 An Equalization Strategy

In general it is not a trivial task to find a good scalar quantity p. For example, choosing p equal to vector field magnitude may not have the desired effect when this quantity varies over multiple orders of magnitude. Instead of exactly being proportional to p, we would rather like to have a density distribution which resembles vector field magnitude qualitatively, but in general places seed points more homogenously. 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:

$$S(p) = \frac{\text{number of cells with } p_i < p}{\text{total number of cells}} \tag{9}$$

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

$$p_i' = S(p_i). \tag{10}$$

Of course other probability distributions can be used to emphasize special features of the field. We have implemented a symbolic interface which allows us to specify p_i as a function of vector field magnitude and other optional scalar fields. Within this interface analytic functions like logarithm or square root as well as threshold operators can be used to modify p. Together with a three-dimensional selection box, which may be positioned interactively to spatially confine the region of interest, it is possible to explore very quickly the overall characteristics as well as the details of a vector field.

4.3 Divergence Compensation

If the vector field has a divergence different from zero, the stream line density will not remain constant even if the seed points are distributed uniformily. In some areas stream lines will run together, resulting in an increased local density. In other areas they will expand, resulting in a decreased local density. In our case stream lines are computed with a fixed maximum length. Experience shows that a sufficiently uniform stream line density is obtained by placing seed points in the middle of a stream line segment, and integrating equally far in forward and backward direction. Of course, better results could be obtained by adaptively terminating existing lines or creating new ones based on local stream line density.

4.4 Streamline Computation

For numerical stream line integration we use a fourth-order Runge-Kutta method with error monitoring and adaptive step size control, as described in [14]. 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 stream line integration has to be terminated. Singularities, i.e. sinks and sources, commonly occur for example in electrostatic fields. Examples are shown in Figs. 4-7.

5 Interaction

Due to its high rendering speed our method is dedicated to be used in an interactive visualization framework. To interactively define regions of interest we apply so-called draggers which are provided by the Open Inventor graphics toolkit. Two such box-type draggers are depicted in Fig. 4. Each dragger defines an rectangular or spherical volume in which stream lines are seeded.

We have also implemented methods to place seed points along curves. The curves may be created by intersecting an arbitrary geometry, e.g. an isosurface, and a user-defineable plane. This method is particularly suited to highlight possible symmetries within the data set.

Further refinements can be achieved by using scalar fields that define local seed point density or local degree of interest. Such fields can be obtained from numerical simulation. In addition we use a symbolic interface to define such fields in terms of vector field magnitude, cartesian coordinates and other available quantities.

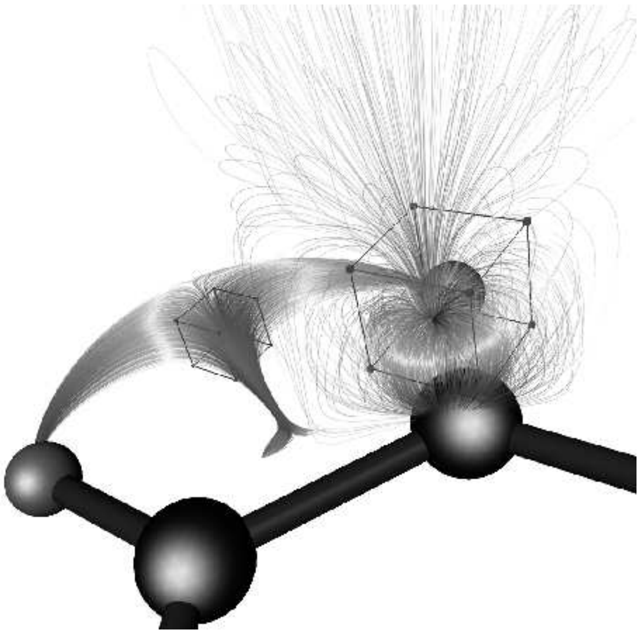


Figure 4: Interactive definition of seed volumes using Open Inventor draggers.

6 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 shaded stream lines in combination with other geometries. The shading itself makes use of the OpenGL graphics library.

On a SGI Indigo^2 desktop workstation with Maximum Impact graphics and a 250 MHz R4400 CPU scenes containing 3000 stream lines each consisting of 120 transparent line segments can be rendered at a rate of 10 frames per second. These results can be improved by 10%-20% if OpenGL display lists are used. However, display lists cause the rendering to be delayed when the scene is drawn the first time. Therefore, in our implementation the user can choose whether to use display lists or not. Also, the integration of our algorithm into the Open Inventor rendering scheme may be further optimized.

We have applied our methods to visualize vector fields from various disciplines like computational fluid dynamics, quantum chemistry, and medicine. In most cases the default values for seed point distribution (eventually accompanied with the histogram equalization technique) 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 an improved spatial perception.

Fig. 5 and 6 show the electrostatic field of a benzene molecule. The field is computed using the NAO-PC method (Natural Atomic Orbitals - Point Charge). This quantumclassical 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.

Fig. 7 also depicts parts of an electrostatic field of a molecule. In addition stream lines have been color coded as described in section 3.4. In this example color depicts the electrostatic potential. The field lines connect several positive point charges (magenta) with a single negative charge (green-orange).

An example of a velocity field from a CFD application is shown in Fig. 8. The data represents a fluid flow over a backward facing step. The turbulent region emphasized by the visualization is characterized by a very complex field structure.

Finally Fig. 9 illustrates the power of accurate line shading: While only a poor three-dimensional impression is obtained from the middle and right images, the spatial structure of the field is clearly revealed in the left image.

7 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 stream lines, i.e. integral curves of the field. The method gives a good impression ofthe field structure and enables us to resolve visually rather fine details, like small vortices. A texture mapping technique is used to accurately illuminate the stream lines. Light reflection on stream lines improves spatial perception and thereby facilitates the understanding of the inner structure of a field.

We have shown how high quality stream 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. Additional use of of a histogram equalization approach allows us to automatically place stream line segments more homogenously.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

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