Illuminated Lines Revisited

Ovidio Mallo* Ronald Peikert† Christian Sigg‡ Filip Sadlo§

ETH Zürich

ABSTRACT

For the rendering of vector and tensor fields, several texture-based volumetric rendering methods were presented in recent years. While they have indisputable merits, the classical vertex-based rendering of integral curves has the advantage of better zooming capabilities as it is not bound to a fixed resolution. It has been shown that lighting can improve spatial perception of lines significantly, especially if lines appear in bundles. Although OpenGL does not directly support lighting of lines, fast rendering of illuminated lines can be achieved by using basic texture mapping. This existing technique is based on a maximum principle which gives a good approximation of specular reflection. Diffuse reflection however is essentially limited to bidirectional lights at infinity. We show how the realism can be further increased by improving diffuse reflection. We present simplified expressions for the Phong/Blinn lighting of infinitesimally thin cylindrical tubes. Based on these, we propose a fast rendering technique with diffuse and specular reflection for orthographic and perspective views and for multiple local and infinite lights. The method requires commonly available programmable vertex and fragment shaders and only two-dimensional lookup textures.

CR Categories: Picture/Image Generation I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture; I.3.8 [Computer Graphics]: Applications;

Keywords: Field lines, illumination, vector field visualization, texture mapping, graphics hardware

1 INTRODUCTION

A core topic of scientific visualization is the representation of vector and tensor fields in three-dimensional space. The dimensionality of the problem forbids a direct visualization such as color coding and requires an abstraction to be made, classically either arrows or integral curves. The latter kept their place in scientific visualization even though a wide spectrum of innovative techniques have evolved over the years (see e.g. [11]). The reason is certainly that field lines have a physical meaning for most kinds of simulation or measurement data such as electric and magnetic fields, velocity and vorticity fields. Beyond field lines, streaklines and trajectories can be used for visualizing transient flow, whereas integral curves of eigenvectors is a way of visualizing tensor fields.

While explicit computation of field lines has been practiced since the early days of scientific visualization, newer techniques provide an implicit way of visualizing field lines. Crawfis and Max [4] were the first to adapt direct volume rendering to vector fields. A few years later, the line integral convolution method of Cabral and Leedom [3] was extended to three dimensions for volume rendering by Interrante and Grosch [6]. More recently, a texture-based method exploiting programmable graphics hardware was presented by Li et al. [8]. For the visualization of tensor fields, methods based on anisotropic volume rendering have been developed by Sigfridsson et al. [14] and by Schussman and Ma [13], the latter being also applicable to general sets of lines.

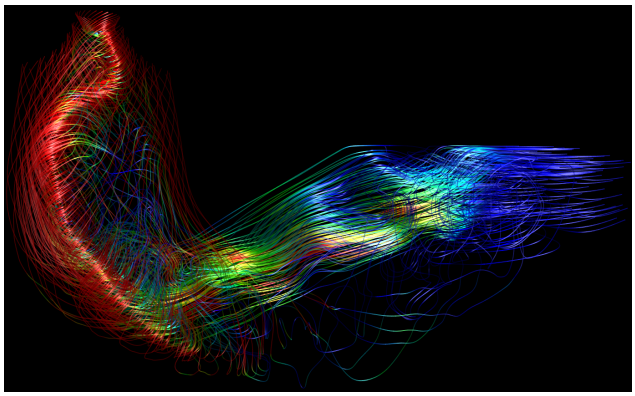

Figure 1: Flow in a Francis draft tube visualized by streamlines regularly seeded on a cone and colored by speed. Streamlines are illuminated based on cylinder averaging. In the vertical part of the tube, a vortex rope is visible.

In other areas of computer graphics, the display of lines in space is of much less importance nowadays. This probably explains why graphics libraries such as OpenGL do not offer an automatic access to high-quality rendering of polylines comparable to that of polygonal surfaces. In fact, OpenGL's entire lighting system is targeted at polygons. But it is obvious that lighting is as important for the spatial perception of line bundles as it is for surfaces. The rendering of lines as tubes is a theoretical solution but in practice it is too costly and produces poor quality caused by thin elongated polygons. Nevertheless, the cylindrical tube is the basis for two line rendering techniques that have been described in literature. Banks [1] introduced the idea of maximizing the reflected light over the perimeter of an infinitesimally thin cylinder, treating diffuse and specular reflection separately. Zöckler, Stalling and Hege [17, 15] found a fast way of rendering polylines illuminated according to this maximum reflection principle in OpenGL. Their method requires only 2D textures and texture transformations, concepts that were commonly available on graphics cards of that time. Under these hardware constraints, their method is probably the optimal tradeoff between speed and quality. The idea of using texture transformations for lighting has been studied in the context of polygon rendering by Heidrich and Seidel [5]. A variation of the maximum reflection illumination model has been applied to the visualization of diffusion tensor imaging data by Wenger et al. [16].

The maximum reflection only approximates the reflection from the cylinder as calculated by integrating over its perimeter. In the case of high gloss specular reflection, the approximation works

*e-mail:ovidiom@student.ethz.ch
†e-mail:peikert@inf.ethz.ch
‡e-mail:sigg@inf.ethz.ch
§e-mail:sadlof@inf.ethz.ch

quite well since the angles near the maximum contribute most to the integral. The method produces best results if the reflection type is chosen to be mostly specular. Diffuse reflection is not well approximated by the maximum which is not even sensitive to the sign of the light direction. This means that any (infinite) light source is effectively bidirectional, in the sense that there is a second light source in the opposite direction. Bidirectional lights are less disturbing if the light direction is close to the viewing direction. In fact, the use of a headlight is also recommended in [15].

As an alternative to the maximum reflection principle, cylinder averaging can be used for illumination of lines. Here, the spatial curve is treated as the limit of a cylindrical tube of a radius approaching zero. The cylinder is considered opaque and has therefore self-occlusion which eliminates much of the diffusely reflected light. Diffuse and specular reflection are calculated by integrating over the visible part of the perimeter. Such a lighting model has been used by Schussman and Ma [13]. The diffuse component of their model is a view dependent version of the one introduced by Kajiya and Kay [7]. Early work on numerically computed diffuse and specular reflection from cylinders was done by Miller [9] for the rendering of hair and by Poulin and Fournier [12] for the purpose of modeling surface anisotropy.

Illuminated lines based on maximum reflection was an optimal way of utilizing a past generation of graphics hardware. It is the goal of this work to find out how illuminated lines can be improved with the features offered by current graphics cards. We aim at line rendering at a speed and quality comparable to standard OpenGL rendering of polygons. Therefore, our focus is on efficiency, not on realistic lighting in a physical sense. Our contributions are simplified expressions for cylinder averaging, allowing the computation of diffuse and specular reflection for orthographic and perspective views, multiple local and infinite lights and variable gloss without the need for 3D textures. We implemented our polyline rendering as a replacement for OpenGL's `glMultiDrawArrays` function, making it easy to switch between standard rendering and illuminated lines. Finally, we demonstrate on a few examples how the improved diffuse reflection can help to better perceive spatial structures visualized by bundles of field lines.

2 Line Illumination

If a point $\mathbf{P}$ on a *surface* is to be lit, the standard Phong [10] and Phong/Blinn [2] lighting models require besides the material and light properties the three unit vectors $\mathbf{V}$ (pointing from $\mathbf{P}$ towards the camera), $\mathbf{L}$ (pointing from $\mathbf{P}$ towards the light source) and $\mathbf{N}$ (oriented surface normal).

The Phong lighting model for a single white light source and a single channel is

$$I = I_a + I_d + I_s = k_a + k_d \mathbf{L}\cdot\mathbf{N} + k_s (\mathbf{V}\cdot\mathbf{R})^n \qquad (1)$$

where k_a, k_d and k_s denote the ambient, diffuse and specular reflection coefficients and n the specular exponent. $\mathbf{R}$ is the reflection of $\mathbf{L}$ at $\mathbf{N}$. The intensities I_d and I_s are clamped between 0 and 1 and in the case of multiple or colored lights they are weighted sums.

The Phong/Blinn model avoids computing $\mathbf{R}$ and uses instead the halfway vector $\mathbf{H} = (\mathbf{V}+\mathbf{L})/\|\mathbf{V}+\mathbf{L}\|$. It is

$$I = I_a + I_d + I_s = k_a + k_d \mathbf{L}\cdot\mathbf{N} + k_s (\mathbf{H}\cdot\mathbf{N})^n \qquad (2)$$

and approximates Eq. 1 with a specular exponent of $\frac{n}{4}$. This model is used by OpenGL as it is more efficiently computed.

For a point $\mathbf{P}$ on a *curve in space*, the situation differs in that a curve does not have a uniquely defined normal. Lighting must instead be computed from the three unit vectors $\mathbf{V}$, $\mathbf{L}$ and the tangent $\mathbf{T}$ to the curve at the point $\mathbf{P}$. Throughout this paper we will use a curve/view-aligned coordinate frame $(\mathbf{T},\mathbf{N},\mathbf{B})$, with the binormal $\mathbf{B} = \mathbf{T}\times\mathbf{V}/||\mathbf{T}\times\mathbf{V}||$ and the normal $\mathbf{N} = \mathbf{B}\times\mathbf{T}$. If the curve is understood as an infinitesimally thin cylinder, it has at $\mathbf{P}$ the normal vectors $\mathbf{N}_\theta = \mathbf{N}\cos\theta + \mathbf{B}\sin\theta$ where the phase is chosen such that $\mathbf{N_0} = \mathbf{N}$. This is illustrated in Fig. 2.

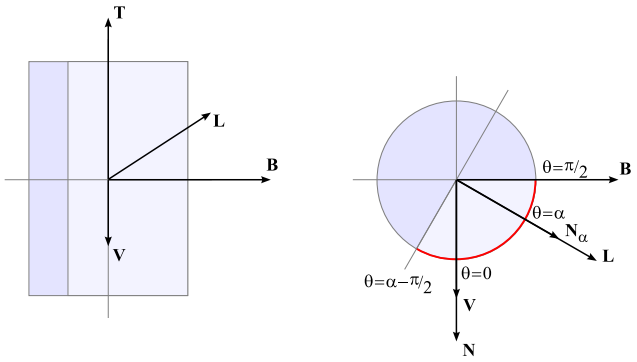

Figure 2: Front and top view of cylinder, unit vectors. Visible sector of diffuse reflection (red arc and lighter shading).

2.1 Maximum reflection principle

Banks [1] resolved the ambiguity in the choice of the normal vector by choosing those angles θ for which the dot products occurring in the diffuse and specular terms of Eq. 1 are maximal. In general, two different angles result for the diffuse and the specular term. Their derivation is recapitulated in the next two subsections.

2.1.1 Diffuse term

Calculation of the diffuse reflection is based on the assumption of a Lambertian reflector, meaning that diffuse reflection does not depend on the viewing direction. According to Phong's model, diffuse reflection is $\mathbf{L}\cdot\mathbf{N}_\theta$, the product of the light vector and the facet normal. In the basis $(\mathbf{T},\mathbf{N},\mathbf{B})$, these vectors are

$$\begin{aligned} \mathbf{L} &= \begin{pmatrix} L_T \\ L_N \\ L_B \end{pmatrix} = \begin{pmatrix} L_T \\ \sqrt{1-L_T^2}\cos\alpha \\ \sqrt{1-L_T^2}\sin\alpha \end{pmatrix} \\ \mathbf{N}_\theta &= \begin{pmatrix} 0 \\ \cos\theta \\ \sin\theta \end{pmatrix} \end{aligned} \qquad (3)$$

where α is the angle between $\mathbf{N}$ and the projection of $\mathbf{L}$ onto the $(\mathbf{N},\mathbf{B})$ plane. The dot product of the two vectors is

$$\mathbf{L}\cdot\mathbf{N}_\theta = \sqrt{1-L_T^2}\cos(\theta-\alpha). \qquad (4)$$

The maximum is reached at $\theta = \alpha$, making the diffuse term

$$I_d = k_d\sqrt{1-L_T^2}. \qquad (5)$$

2.1.2 Specular term

The specular term, if computed according to the Phong model, requires the view vector

$$\mathbf{V} = \begin{pmatrix} V_T \\ \sqrt{1-V_T^2} \\ 0 \end{pmatrix} \tag{6}$$

and the reflection of the vector $\mathbf{L}$ at $\mathbf{N}_\theta$

$$\mathbf{R}_\theta = -\mathbf{L} + 2(\mathbf{L}\cdot\mathbf{N}_\theta)\mathbf{N}_\theta = \begin{pmatrix} -L_T \\ \sqrt{1-L_T^2}\cos(2\theta-\alpha) \\ \sqrt{1-L_T^2}\sin(2\theta-\alpha) \end{pmatrix}. \tag{7}$$

The dot product to be maximized is then

$$\mathbf{V}\cdot\mathbf{R}_\theta = -V_T L_T + \sqrt{1-V_T^2}\sqrt{1-L_T^2}\cos(2\theta-\alpha) \tag{8}$$

and its maximum is reached at $\theta = \alpha/2$, making the specular term:

$$I_s = k_s\left(-V_T L_T + \sqrt{1-V_T^2}\sqrt{1-L_T^2}\right)^n. \tag{9}$$

In [1] Eq. 9 is given with opposite signs. The first sign gets positive due to the different definition of $\mathbf{L}$, but the sign preceding the square roots is nevertheless positive. The expression is given correctly in [15] though its derivation does not apply to the general case.

2.1.3 Rendering

As is described in [15], the diffuse and specular parts of this lighting model can be implemented in OpenGL for standard shaders. 3D textures can be avoided by restricting to orthographic view and lights at infinity. Then the vectors $\mathbf{V}$ and $\mathbf{L}$ are constants. The three components of the tangent $\mathbf{T}$ are taken as original texture coordinates from which the dot products $\mathbf{L}\cdot\mathbf{T}$ and $\mathbf{V}\cdot\mathbf{T}$ are computed by multiplying with an appropriate texture transformation matrix. Finally, the diffuse and specular intensities are looked up in textures built up from Eq. 5 and 9, respectively.

The main drawback of lighting based on the maximum principle is its unrealistic modeling of diffuse reflection. As is shown in Fig. 3, lateral lighting from the left and from the right cannot be discerned if only diffuse reflection is present. A more realistic diffuse lighting of lines has to be view dependent.

2.2 Cylinder averaging

Schussman and Ma [13] calculate diffuse and specular reflection by integrating the reflection from the infinitesimally thin facets of a cylinder. The range of integration consists of those facets which are both visible and lit. It depends on the angle α between the projections of $\mathbf{V}$ and $\mathbf{L}$ onto the $(\mathbf{N},\mathbf{B})$ plane. To get simpler formulas, we can force α to lie between 0 and π by making the third component of $\mathbf{L}$ nonnegative, i.e. by possibly reflecting $\mathbf{L}$ at the $(\mathbf{T},\mathbf{V})$ plane. This way the integration bounds are always $\alpha-\frac{\pi}{2}$ and $\frac{\pi}{2}$.

The contribution of each facet to the total reflected light depends on the projected area as seen from the $\mathbf{V}$ direction. For a cylinder of unit height and unit radius, the facet perpendicular to $\mathbf{N}_\theta$ is projected to an area of

$$\mathbf{V}\cdot(\mathbf{N}\cos\theta + \mathbf{B}\sin\theta)d\theta = \mathbf{V}\cdot\mathbf{N}\cos\theta\, d\theta. \tag{10}$$

The total projected area is $2\mathbf{V}\cdot\mathbf{N}$, therefore the integrand has to be weighted with $\frac{\cos\theta}{2}$.

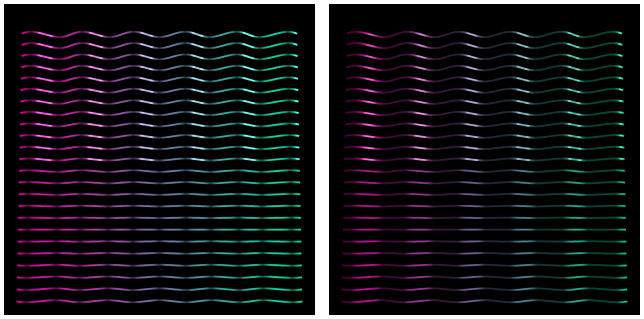

Figure 3: Stack of horizontal sine curves with directional lighting from the right. Left image: Maximum reflection produces bidirectional diffuse lighting (bottom half), a strong specular component is needed to disambiguate light direction (top half). Right image: Cylinder averaging. The light direction is clear even without a specular component (bottom half).

2.2.1 Diffuse term

The diffuse term for an infinitesimally thin cylinder facet is given by Eq. 4. As explained above, this term has to be multiplied with the weight $\frac{\cos\theta}{2}$ and integrated from $\alpha-\frac{\pi}{2}$ to $\frac{\pi}{2}$. The integral can be resolved analytically, giving:

$$\begin{aligned} I_d &= k_d \int_{\alpha-\pi/2}^{\pi/2} (\mathbf{L}\cdot\mathbf{N}_\theta)\frac{\cos\theta}{2}d\theta \\ &= k_d\sqrt{1-L_T^2}\int_{\alpha-\pi/2}^{\pi/2}\cos(\theta-\alpha)\frac{\cos\theta}{2}d\theta \\ &= k_d\sqrt{1-L_T^2}\,\frac{\sin\alpha+(\pi-\alpha)\cos\alpha}{4}. \end{aligned} \tag{11}$$

An equivalent but less compact expression was published in [13]. Our expression lets us recognize the square root term as obtained with the maximum reflection principle (Eq. 5), but multiplied with a factor between 0 and $\frac{\pi}{4}$ depending on the angle α between projected view and projected light. By taking the dot product of $\mathbf{L}$ and $\mathbf{V}$ given in Eq. 3 and Eq. 6, respectively, this angle can be computed as

$$\alpha = \arccos\frac{\mathbf{V}\cdot\mathbf{L} - V_T L_T}{\sqrt{1-V_T^2}\sqrt{1-L_T^2}}. \tag{12}$$

In the special case of constant vectors $\mathbf{V}$ and $\mathbf{L}$ (i.e. for orthographic view and directional light) the technique of [17] can be applied. First, V_T and L_T are computed by a texture transform from $\mathbf{T}$, then I_d is looked up in a 2D texture.

In the general case, $\mathbf{V}$ and $\mathbf{L}$ must be computed per vertex which can be done in a vertex program.

2.2.2 Specular term

Cylinder averaging of the specular term can be done for both the Phong and the Phong/Blinn model. However, both lead to integrals which need to be solved numerically. In the case of the Phong model, the integral of Eq. 8 depends on the three parameters V_T, L_T and α if the specular exponent n is considered fixed. The three parameters describe the relative orientation of $\mathbf{T}$, $\mathbf{V}$ and $\mathbf{L}$ and are independent in general. If the numerical integrals are precomputed and stored in lookup textures, this would imply 3D textures which we find an inappropriate use of resources for the sole purpose of line illumination.

Therefore, we follow [13] and use the Phong/Blinn model leading to simpler expressions and finally to 2D textures only. From $\mathbf{V}$ and $\mathbf{L}$ we first compute the halfway vector

$$\mathbf{H} = \frac{\mathbf{V}+\mathbf{L}}{\|\mathbf{V}+\mathbf{L}\|} = \begin{pmatrix} H_T \\ \sqrt{1-H_T^2}\cos\beta \\ \sqrt{1-H_T^2}\sin\beta \end{pmatrix}. \tag{13}$$

Note that the angle β between projected view and projected halfway vector can vary between 0 and π, (not just $\frac{\pi}{2}$) [1].

Using the weighting term and the integration bounds as above, the specular term can be calculated as:

$$\begin{aligned} I_s &= k_s \int_{\alpha-\pi/2}^{\pi/2} (\mathbf{H}\cdot\mathbf{N}_\theta)\frac{\cos\theta}{2}d\theta \\ &= k_s\sqrt{1-H_T^2}^n \int_{\alpha-\pi/2}^{\pi/2} \cos^n(\theta-\beta)\frac{\cos\theta}{2}d\theta. \end{aligned} \tag{14}$$

Further simplification yields:

$$\begin{aligned} I_s = k_s\sqrt{1-H_T^2}^n \Bigg(&\frac{\sin\beta}{2(n+1)}\left(\sin^{n+1}\beta - \sin^{n+1}(\alpha-\beta)\right) + \\ &\frac{\cos\beta}{2}\left(S_{n+1}(\pi-\alpha+\beta) - S_{n+1}(\beta)\right)\Bigg) \end{aligned} \tag{15}$$

where the function

$$S_{n+1}(x) = \int_0^x \sin^{n+1} t\,dt \tag{16}$$

can be precomputed and tabulated for $x = 2\pi h/2^m$, $h = 0,1,\cdots,2^m$. This means that numerics is needed only for generating a 1D lookup table, assuming a fixed specular exponent n. The 2D lookup table (or texture map) can then be computed analytically. As an alternative, it is also possible to treat n as variable. Then, computing I_s involves a sequence of 2D lookups for $S_{n+1}(x)$ and for powers of trigonometric functions.

Eq. 15 is equivalent to formulas (16) and (17) in [13] up to the plus signs in the integration bounds which should be minus signs.

2.3 Brightness adjustment

2.3.1 Excess brightness

Diffuse and specular lighting based on maximum reflection leads to high average brightness of rendered scenes. To compensate for this, Banks [1] proposed an artificial brightness reduction factor. Lighting based on cylinder averaging does not have this problem. By definition of the lighting model, a random set of lines has the same brightness distribution as a random set of polygons.

[1] An example is

$$\mathbf{V} = \begin{pmatrix} \sqrt{3}/2 \\ 1/2 \\ 0 \end{pmatrix}, \mathbf{L} = \begin{pmatrix} -1/2 \\ -\sqrt{3}/2 \\ 0 \end{pmatrix}, \mathbf{H} = \begin{pmatrix} \sqrt{2}/2 \\ -\sqrt{2}/2 \\ 0 \end{pmatrix}, \cos\beta = -1$$

.

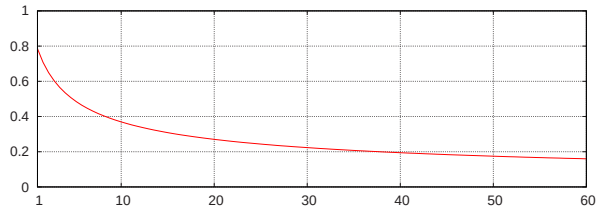

Figure 4: Maximum of specular term as a function of the specular exponent n.

2.3.2 Stretching of dynamic range

Because of the averaging process in the infinitesimal cylinder method, both the diffuse and the specular term have a maximum less than 1. This offers the option to stretch the dynamic range by multiplying the reflection term with a constant, which has the effect of brighter light sources. For the diffuse term (Eq. 11) the maximum is $\frac{\pi}{4}$ at $\alpha = 0, L_T = 0, k_d = 1$. Therefore, it is possible to stretch the dynamic range by multiplying Eq. 11 with $\frac{4}{\pi}$.

Likewise, the specular term (Eq. 14) has a maximum at $k_s = 1$, $H_T = \alpha = \beta = 0$, of

$$\frac{\sqrt{\pi}}{2}\Gamma\left(\frac{n+1}{2}\right)\Big/\Gamma\left(\frac{n+3}{2}\right). \tag{17}$$

This is a monotonically decreasing function (see Fig. 4) which simplifies to

$$\frac{2}{3}\frac{4}{5}\frac{6}{7}\cdots\frac{n}{n+1}$$

for (positive) even n and to

$$\frac{\pi}{2}\frac{1}{2}\frac{3}{4}\frac{5}{6}\frac{7}{8}\cdots\frac{n}{n+1}$$

for odd n.

This maximum decreases rapidly with increasing n, therefore it is advisable to stretch the dynamic range at least for the typically high specular exponents.

3 Implementation

The lighting model presented in this paper has been implemented in C++ on top of the OpenGL graphics library. Since our main goal was to allow for a transparent and seamless integration of our code into already existing applications, the interface provided by our implementation consists of a replacement of the `glMultiDrawArrays` function which is part of the OpenGL API. This makes it possible to easily switch between standard rendering and illuminated lines.

As opposed to OpenGL, our lighting model will be evaluated on a per-pixel basis. For maximum performance, this is done by a combination of traditional texture mapping and modern programmable Graphics Processing Units (GPUs).

3.1 Lighting Textures

For evaluating the lighting model, two 2D lighting textures are used, one for the diffuse, the other for the specular component of the lighting model. We will refer to the values stored inside those textures through the functions F_d and F_s for the diffuse and specular part, respectively. For F_d, we have

$$F_d(\cos\alpha, L_T) = \sqrt{1-L_T^2}\,\frac{\sin\alpha - (\pi-\alpha)\cos\alpha}{4}. \tag{18}$$

This basically corresponds to the full diffuse term as given in Eq. 11, without the reflection coefficient k_d, which is not stored inside the texture. For the specular part, we only store the integral given in Eq. 14 in the texture, which needs to be integrated numerically, since storing the full specular term would require 3D textures to be used. Thus, for F_s, we have

$$F_s(\cos\alpha,\cos\beta) = \int_{\alpha-\pi/2}^{\pi/2} \cos^n(\theta-\beta)\frac{\cos\theta}{2}d\theta. \qquad (19)$$

For actually computing this texture during the preprocessing step, Eq. 15 is used for better efficiency.

Evidently, the expressions $\cos\alpha$, $\cos\beta$ and L_T used in Eq. 18 and Eq. 19 all lie in $[-1,+1]$. Therefore, they need to be range compressed to $[0,1]$ in order to use them as texture coordinates. If x is any of those expressions, this can simply be done by the transformation $(x+1)/2$.

Note that we use the cosine of α and β instead of the angles themselves for accessing the textures since the arc cosine function is usually not directly supported by programmable GPUs and therefore very slow.

Using these textures, for every incoming fragment with color C_{in}, the final color of the illuminated fragment can be computed as

$$\begin{aligned} C_{out} &= C_{in}\left(k_a + k_d F_d(\cos\alpha, L_T)\right) \\ &\quad + k_s\sqrt{1-H_T^2}^{\,n} F_s(\cos\alpha,\cos\beta). \end{aligned} \qquad (20)$$

3.2 Shader Programs

For evaluating Eq. 20, we first of all need to compute the tangent vector at each vertex. With today's powerful programmable GPUs, this can be done efficiently at each frame in a vertex program. To do so, for each vertex $\mathbf{P}_i$, we pass $\mathbf{P}_{i-1}$ and $\mathbf{P}_{i+1}$ as texture coordinates to our vertex program, where the tangent vector $\mathbf{T}$ is then computed as $\mathbf{T} = (\mathbf{P}_{i+1}-\mathbf{P}_{i-1})/||\mathbf{P}_{i+1}-\mathbf{P}_{i-1}||$. Of course, this does not work for the first and last vertex of each polyline. For those vertices, the tangent vectors are computed in software.

Furthermore, if $\mathbf{O}$ is the position of our light, the vertex program computes in camera coordinates the light vector $\mathbf{L} = \mathbf{O} - \mathbf{P}$ and the viewing vector $\mathbf{V} = -\mathbf{P}$.

$\mathbf{L}$, $\mathbf{V}$ and $\mathbf{T}$ are then passed to a fragment program. Since per-vertex data is only interpolated linearly between vertices before being passed to the fragment processing unit, those vectors need to be normalized at each fragment. After this normalization, the curve/view-aligned coordinate frame $(\mathbf{T},\mathbf{N},\mathbf{B})$ described in section 2 as well as the halfway vector $\mathbf{H}$ are computed.

The expressions needed for accessing the lighting textures according to Eq. 18 and Eq. 19 can then be computed as $\cos\alpha = (\mathbf{L}\cdot\mathbf{N})/\sqrt{1-(\mathbf{L}\cdot\mathbf{T})^2}$, $\cos\beta = (\mathbf{H}\cdot\mathbf{N})/\sqrt{1-(\mathbf{H}\cdot\mathbf{T})^2}$ and $L_T = \mathbf{L}\cdot\mathbf{T}$, which in turn allows us to easily evaluate our lighting model according to Eq. 20.

3.3 Multiple Lights

Throughout the discussion of our lighting model, we have restricted ourselves to the case of a single light source in the scene. Nevertheless, a generalization for supporting multiple lights is straightforward. In Eq. 18 and Eq. 19, we can see that the values stored in the lighting textures are independent of any light source. Therefore, no additional texture resources are needed for supporting multiple lights.

The vertex program only needs to be changed in that, instead of computing and passing a single light vector $\mathbf{L}$ to the fragment program, it has to do so for each light source, yielding a set of light vectors $\mathbf{L}_i$.

In the fragment program, we see that the local coordinate frame $(\mathbf{T},\mathbf{N},\mathbf{B})$ does not need to be computed for each light source, since it only depends on $\mathbf{T}$ and $\mathbf{V}$, but the halfway vector and the texture coordinates actually do depend on the light vector $\mathbf{L}_i$. The number of light sources must be fixed and is of course limited by performance requirements and size of the fragment program.

By adding an index i to each expression dependent on the light vector $\mathbf{L}_i$, the generalized version of Eq. 20 supporting multiple lights reads

$$\begin{aligned} C_{out} &= C_{in}\sum_i \left(k_{a_i} + k_{d_i} F_d(\cos\alpha_i, L_{T_i})\right) \\ &\quad + \sum_i k_{s_i}\sqrt{1-H_{T_i}^2}^{\,n} F_s(\cos\alpha_i,\cos\beta_i). \end{aligned} \qquad (21)$$

3.4 Depth Sorting

Data sets which need to be visualized often result from areas such as computational fluid dynamics (CFD) and other numerical simulations, which typically produce a vast amount of data. In such situations, where very dense line bundles need to be rendered, the use of alpha blending can considerably improve the visual perception of the scene. Unfortunately, this makes it necessary to depth-sort the scene as mentioned in [15], since alpha blending only yields correct results if the objects are drawn in back to front order.

Although this need for sorting the scene is not limited to the rendering of lines, but a general issue related to alpha blending itself, we have integrated a sorting routine in our implementation. We have opted for performing a full depth sort instead of using approximations like the one described in [15] where real sorting is actually avoided.

For the actual sorting, we use the `sort` function provided by the STL library of the C++ programming language. For simplicity reasons, we sort the individual line segments of the polylines according to the averaged depth of their end points.

There is another less apparent scenario which is more tightly related to line rendering and which makes depth sorting necessary: antialiasing. Due to the way antialiased lines are realized in rendering APIs like OpenGL, i.e. using alpha blending for smoothing out the edges of the lines, depth sorting of the scene becomes necessary although no transparent lines are actually drawn. Since antialiasing is visually very important when rendering lines and since full depth sorting is by far the most expensive step during the rendering process, it is an option to replace the OpenGL line antialiasing mechanism by a full scene antialiasing technique using multisampling. This makes depth sorting superfluous and yields good results at significantly higher frame rates.

3.5 Vertex Buffer Objects

Usually, the rendering of lines does not involve a complex setup of the rendering pipeline, but the problem typically consists in an efficient visualization of a large amount of data. In such situations, the bottleneck usually lies in sending the data from main memory to the GPU. With the traditional use of vertex arrays (VAs), this requires sending the data at each frame over the bus to the GPU, even for static scenes.

To avoid this, at least for static datasets, we have found the concept of so called vertex buffer objects (VBOs) very useful. While VBOs are conceptually very similar to VAs, the data stored in VBOs is cached in high-performance graphics memory directly on the GPU, thereby increasing the rate of data transfers during the rendering process. The use of VBOs often led to important performance gains, as can be seen in Table 1, by comparing the "default" and "dynamic data" rows. Table 1 also reveals, that the actual benefit of using VBOs is GPU specific.

Table 1: Performance measurements for various rendering modes. Frames per second are given for the ATI (fps^1) and nVidia cards (fps^2).

Dataset	Lorenz		El. Field		Draft tube	
Vertices	600,000		883,564		1,431,599	
	fps^1	fps^2	fps^1	fps^2	fps^1	fps^2
default	51.8	22.4	39.4	14.5	25.7	11.7
dynamic data	17.0	21.8	16.9	13.6	6.8	10.7
1024x1024 pixels	36.1	15.8	26.2	11.1	22.7	9.0
high quality	5.7	4.7	3.8	3.0	2.0	1.5
no multisampling	58.1	50.5	41.7	21.3	26.8	30.7
directional	51.5	79.4	60.2	58.1	34.2	36.1
no illumination	94.3	113.6	99.0	78.1	54.3	54.1

4 RESULTS

For validating our implementation we selected a number of datasets with different characteristics in terms of size and distribution of lines in space. The Lorenz attractor $\mathbf{u} = (10(y-x), 28x-y-xz, -\frac{8}{3}z+xy)$ was chosen for its near planar substructures near its two foci. It is visualized by uniformly seeded streamlines which are colored by seed location. The electrostatic field of the water molecule is another simple test dataset which we chose for its high degree of symmetry. It is computed by the superposition of three potentials of point charges and the field lines are seeded on a recursively subdivided sphere centered at the oxygen atom. The draft tube dataset is computed from industrial CFD data. As a test visualization, instantaneous streamlines are seeded on seven rings close to the (roughly conical) hub of the rotor. For all three datasets, the classical fourth-order Runge-Kutta integration was used to generate polylines from the given seed points.

In Fig. 5 the three datasets are rendered with illumination based on maximum reflection and on cylinder averaging. In all image pairs the expected good positional match of the specular highlights between the two methods is confirmed. The highlights were adjusted to roughly equal brightness by stretching the dynamic range as described in section 2.3.2. This makes the different distribution of diffuse reflection clearly visible in that there are more and larger dark regions in the right column of images. Even though this difference is present in all three image pairs, from the point of view of spatial perception the improvement is most noticeable in some areas of the draft tube data where the effect of Fig. 3 comes into play. The same data is visualized in Fig. 1 with fewer but wider lines and a different light position.

4.1 Performance

We measured frame rates on two PCs with ATI Radeon X800 Pro and nVidia GeForce 6800 graphics cards. The CPUs were Pentium 4 with 3.0 and 3.2 GHz, respectively. The datasets are those depicted in Fig. 5 ranging from 600,000 to 1,431,599 vertices. We placed the objects such that the image was roughly filled, but such that no frustum culling occurred.

In Table 1 the "default" rendering mode refers to a 512x512 pixel image rendered with maximal multisampling (6x for ATI, 8x for nVidia) and using vertex buffer objects. In this mode, interactive frame rates can be obtained even with large datasets.

In the "dynamic data" mode VBOs are not used and the vertex arrays are filled with new data for each frame. The higher frame rates achieved with the default mode demonstrate mostly the benefits of VBOs which pays off especially on the ATI system.

The "1024x1024" mode is the same as "default" up to the image resolution. It is remarkable that even though most of the processing is done within the fragment program, the frame rates drop by less than a third when the resolution is doubled.

In "high quality" mode, lines are semitransparent and antialiased and consequently also depth sorted. This mode is required only if multisampling is not available. Due to the sorting step, it is not suitable for interactive use, but can serve for redrawing a static scene.

"No multisampling" is the same as "default" but with multisampling turned off. The frame rates indicate that it is not worth turning multisampling off.

In "directional" mode, illumination is computed for constant $\mathbf{V}$ and $\mathbf{L}$ which is done without vertex/fragment programs. On the nVidia system, this mode is significantly faster than the default mode and is an option if directional light and orthographic viewing is sufficient.

Finally, "no illumination" simply draws Gouraud shaded lines. As can be seen, the price to pay for line illumination is a factor of about 2 to 6 on our two systems.

5 CONCLUSION

In this paper, we have presented a method for rendering illuminated lines in three-dimensional space. The method results from a view-dependent lighting model based on averaged Phong/Blinn lighting over the surface of an infinitesimally thin cylinder. Orthographic and perspective views, as well as multiple local and infinite lights are supported.

We have derived simpler expressions for the given lighting model and implemented it on top of the OpenGL API using traditional resources such as texture mapping, but also exploiting modern hardware capabilities provided by todays graphics processing units, in particular shader programs [2]. Several issues related to the rendering of field lines such as antialiasing, but also depth sorting in conjunction with the use of alpha blending have been discussed, and solutions for an efficient and high-quality rendering have been provided.

REFERENCES

[1] David C. Banks. Illumination in diverse codimensions. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 327–334. ACM Press, 1994.

[2] James F. Blinn. Models of light reflection for computer synthesized pictures. In *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 192–198. ACM Press, 1977.

[3] B. Cabral and L. Leedom. Imaging vector fields using line integral convolution. In *Proc. of SIGGRAPH-93: Computer Graphics*, pages 263–270, Anaheim, CA, 1993.

[4] Roger Crawfis and Nelson Max. Direct volume visualization of three-dimensional vector fields. In *VVS '92: Proceedings of the 1992 workshop on Volume visualization*, pages 55–60, New York, NY, USA, 1992. ACM Press.

[5] Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. *Computer Graphics*, 33(Annual Conference Series):171–178, 1999.

[6] Victoria Interrante and Chester Grosch. Strategies for effectively visualizing 3d flow with volume lic. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 421–424., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[2] A simple implementation can be downloaded from http://graphics.ethz.ch/flowvis/illumlines.

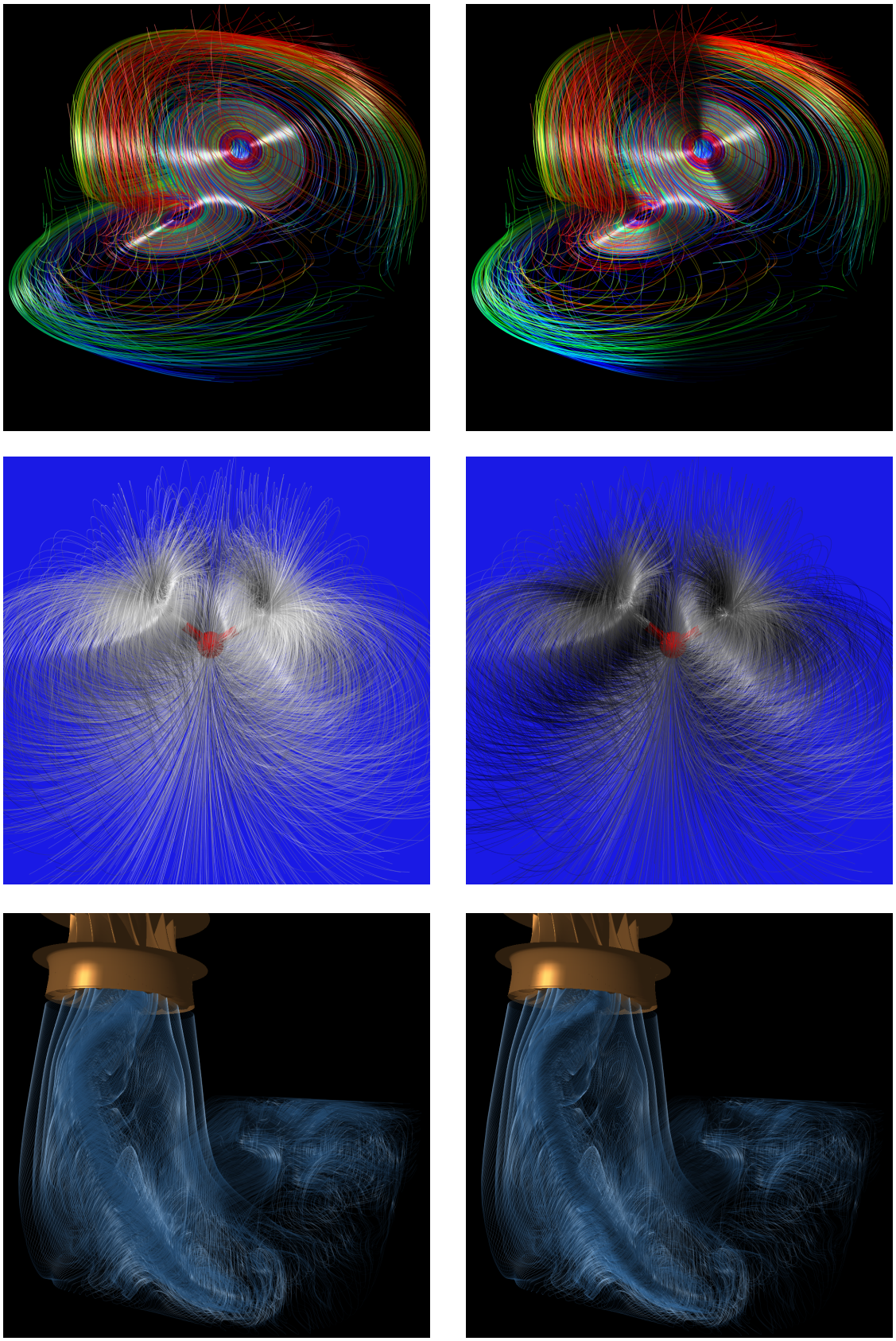

Figure 5: Comparison of illumination based on maximum reflection (left) and on cylinder averaging (right). Visualized datasets are (from top): Lorenz attractor, electrostatic field, drafttube

[7] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. *Computer Graphics*, 23(3):271–280, July 1989.

[8] G. Li, U.D. Bordoloi, and H. Shen. Chameleon: An interactive texture-based rendering framework for visualizing three-dimensional vector fields. In *Proceedings of IEEE Visualization 2003*, pages 241–248, Oct 2003.

[9] G. S. P. Miller. From wire-frames to furry animals. In *Proceedings on Graphics interface '88*, pages 138–145, Toronto, Ont., Canada, Canada, 1988. Canadian Information Processing Society.

[10] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975.

[11] Frits H. Post, Benjamin Vrolijk, Helwig Hauser, Robert S. Laramee, and Helmut Doleisch. Feature extraction and visualization of flow fields. In *EUROGRAPHICS 2002, State of the Art Reports*, pages 69–100, 2002.

[12] Pierre Poulin and Alain Fournier. A model for anisotropic reflection. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 273–282, New York, NY, USA, 1990. ACM Press.

[13] Greg Schussman and Kwan-Liu Ma. Anisotropic volume rendering for extremely dense, thin line data. In *VIS '04: Proceedings of the IEEE Visualization 2004 (VIS'04)*, pages 107–114. IEEE Computer Society, 2004.

[14] A. Sigfridsson, T. Ebbers, E. Heiberg, and L. Wigstrom. Tensor field visualisation using adaptive filtering of noise fields combined with glyph rendering. In *Proc. IEEE Visualization*, pages 371–378, 2002.

[15] Detlev Stalling, Malte Zöckler, and Hans-Christian Hege. Fast display of illuminated field lines. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):118–128, 1997.

[16] Andreas Wenger, Daniel Keefe, Song Zhang, and David H. Laidlaw. Interactive volume rendering of thin thread structures within multi-valued scientific datasets. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):664–672, November/December 2004.

[17] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Interactive visualization of 3d-vector fields using illuminated stream lines. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 107–114. IEEE Computer Society Press, 1996.