Efficient Rendering of Anisotropic Surfaces Using Computer Graphics Hardware

Wolfgang Heidrich and Hans-Peter Seidel

Computer Graphics Group, University of Erlangen
Am Weichselgarten 9, D-91058 Erlangen, Germany
Email: heidrich@informatik.uni-erlangen.de

Abstract

There are surprisingly many anisotropically reflecting objects in the real world. Some examples are brushed metal, cloth, CDs, and even many brands of paper.

Despite the importance of anisotropic reflections in the real world, most of todays computer graphics systems cannot deal with them appropriately. This is particularly true for interactive and hardware-accelerated rendering systems, which is mostly due to the fact that so far there has been no model for simulating anisotropy with commonly available computer graphics hardware.

In this paper we describe a novel approach for simulating anisotropic reflections on OpenGL-based architectures in real time. The rendering of an anisotropic surface with one light source is exactly as expensive as the rendering of the same surface with traditional texturing and lighting. It is therefore possible in real time, even on low-end workstations and PCs.

1 Introduction

Many of the objects we know from day-to-day life have anisotropic surfaces. Often, this anisotropy is caused by a micro structure, in which long, thin features are aligned in one predominant direction. For example, brushed metal objects have been polished in such a way that there are a lot of parallel scratches in the surface. On CDs and records, the micro structure is given by the tracks carrying the music or data. Woven cloth consists of fibers oriented in certain directions, and paper contains wooden grains, which, due to the production process, are often oriented mostly in one direction [8].

The anisotropy in all these examples is created because the distribution of the surface normals along the scratches or fibers is different from the distribution across them. If the observer is distant enough from the surface, these features cannot be seen individually. Instead, the impression of a homogeneous surface surface is created (see Figure 1).

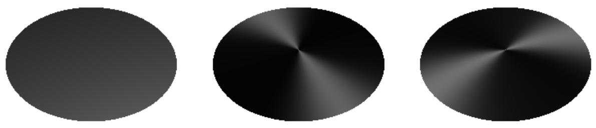

Figure 1: An example of a disk shaded with isotropic reflection (left), anisotropic reflection with circular features (center), and radial features (right).

A model for this kind of surface has been described in [2]. The scratches of fibers are assumed to be lines or curves on the surface. With this assumption, existing lighting models for the illumination of lines can be applied to rendering anisotropic surfaces.

The remainder of this paper is organized as follows. In Section 2 we briefly summarize the method for illuminating lines described in [2]. In Section 3 we describe a way to efficiently illuminate lines with the help of computer graphics hardware and the OpenGL API [6]. This part is based on work presented in [11] and [9]. Afterwards we discuss how self-shadowing of surfaces can be handled, and present multipass methods for isotropic components and multiple light sources in Section 4.

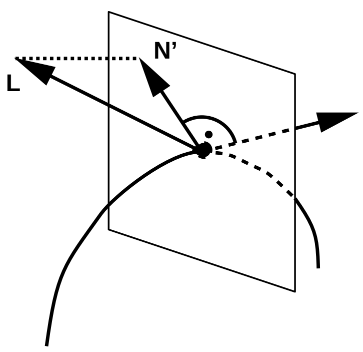

Figure 2: In order to find the shading normal N, the light vector L is projected into the normal plane.

2 Illumination

The illumination of lines and line segments in 3-dimensional space has been discussed in several places [2, 11, 9]. Other publications (for example [4, 3]) have researched the same topic in the context of rendering hair and fur.

Before we discuss the illumination of lines, we briefly review the Phong illumination model of surfaces. The intensity I_o of any surface point is given as the sum of the ambient, diffuse and specular components:

$$\begin{aligned} I_o =& I_{ambient} + I_{diffuse} + I_{specular} \\ =& k_a I_a + (k_d \cos\phi + k_s (\cos\theta)^n) \cdot I_i \\ =& k_a I_a + (k_d \langle L, N\rangle + k_s \langle V, R\rangle^n) \cdot I_i \end{aligned}$$

where k_a, k_d, and k_s are the ambient, diffuse, and specular reflection coefficients, $1/n$ is the surface roughness, I_a is the ambient light in the scene, I_i is the incoming light at the surface point, N is the surface normal, L is the unit vector towards the light, R is the reflection of L at N, and finally V is the unit vector towards the viewpoint.

The fundamental difference between the illumination of curves and the illumination of surfaces is that every point on a curve has an infinite number of normal vectors. Thus, every vector that is perpendicular to the tangent vector of the curve is a potential candidate for use in the illumination calculation.

For reasons described in [2], and [9], the vector N' selected from this multitude of potential normal vectors is the projection of the light vector L into the normal plane, as depicted in Figure 2.

As described in [11, 9], the cosine $\langle L, N'\rangle$ used for the diffuse component can be expressed in terms of the tangent T, and the light vector V for this specific normal vector N'. Since L, T, and N' are all coplanar, the angle $\phi = \angle(L, N')$ is identical to $\pi/2 - \angle(N', T)$, and thus

$$\langle L, N'\rangle = \cos\phi = \sin(\angle(N', T)) = \sqrt{1 - \langle L, T\rangle^2}.$$

Similarly, for the specular component we get

$$\langle V, R\rangle = \sqrt{1 - \langle L, T\rangle^2}\sqrt{1 - \langle V, T\rangle^2} - \langle L, T\rangle \langle V, T\rangle .$$

With this transformation we can describe the brightness of a point as a bivariate function $I_o(\langle L, T\rangle, \langle V, T\rangle)$, which only depends on the two cosines $\langle L, T\rangle$ and $\langle V, T\rangle$.

With the exception of self-shadowing effects, which are described in Section 3.1, this method for illuminating lines is directly applicable for the illumination of anisotropic surfaces as described above.

3 OpenGL Implementation

For an efficient implementation of this model in OpenGL, let us assume we had a way of specifying $\langle L, T\rangle$ and $\langle V, T\rangle$ as the *texture coordinates* s and t (actually, in order to make for valid OpenGL texture coordinates, the cosines first have to be mapped to the range $[0, 1]$). In order to illuminate any given point of a curve or surface with this illumination model, we could then simply precompute a texture containing the bivariate function $I_o(\langle L, T\rangle, \langle V, T\rangle)$.

One way of using $1/2(\langle L, T\rangle + 1)$ and $1/2(\langle V, T\rangle + 1)$ as texture coordinates, is to recompute and specify these values for each frame individually. However, this would mean that the geometry of the scene would have to change for each frame, which would result in low performance. A more clever way of achieving the same goal is to use a texture coordinate matrix for computing the cosines [9]. If we use the tangent vector $T = (t_x, t_y, t_z)^T$ as the initial texture coordinate vector, then an appropriately chosen texture matrix performs the necessary transformations:

$$\frac{1}{2}\begin{bmatrix} l_x & l_y & l_z & 1 \\ v_x & v_y & v_z & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} t_x \\ t_y \\ t_z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(\langle L, T\rangle + 1) \\ \frac{1}{2}(\langle V, T\rangle + 1) \\ 0 \\ 1 \end{bmatrix}$$

Here, the l_i and v_i are the components of the L, and V vector, respectively.

Note that this allows us to specify the texture coordinates only once, while changes in view point and light position can be taken care of by manipulating the texture matrix. As a consequence, the whole geometry remains fixed for all frames, and can be stored in a display list. Thus, anisotropic surfaces can be rendered at exactly the cost of rendering the same surface with traditional texture mapping enabled.

Figure 3 shows a sphere rendered with this method from different viewpoints. The sphere resembles a satin ball like the ones used for Christmas trees. The satin fibers range from pole to pole on the sphere. The light comes from a 45 degree angle from the right and from behind the viewer.

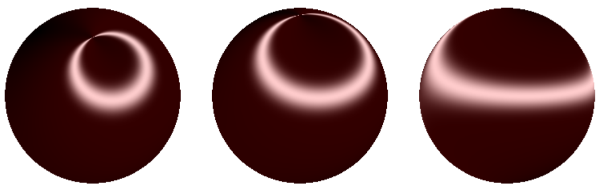

Figure 3: A "satin ball" rendered with the proposed method from three different views.

3.1 Self-Shadowing

Although this image already shows the typical effects of anisotropy, the shading looks unrealistic, because the left side of the sphere is illuminated despite it pointing away from the light source. This behavior is correct for the illumination of lines, because these do not have back faces.

For surfaces, however, areas that point away from the light should not receive any light. Moreover, areas that are illuminated under glancing angles should be darker than areas onto which the light shines perpendicularly. This is due to the fact that, for glancing angles, the fibers (or scratches) cast shadows onto each other. In order to account for this effect, Banks [2] proposes to modify the illumination model as follows:

$$I_o = I_{ambient} + clamp(\langle -N, L\rangle\,(I_{diffuse} + I_{specular}),$$

where N is the actual geometric normal at the surface point, and the function *clamp* is zero for negative values, and the identity function otherwise.

This modification can be incorporated into the OpenGL implementation with the help of the standard OpenGL lighting mechanism. In the OpenGL rendering pipeline, the result of the texture lookup operation for each pixel can be multiplied by the result of the lighting calculation ("fragment color") for the same pixel. If we use a diffuse, white surface material, and a white, parallel light, the resulting lighting computations performed by OpenGL result exactly in the clamped cosine value required above. Figure 4 shows some results for this modified algorithm.

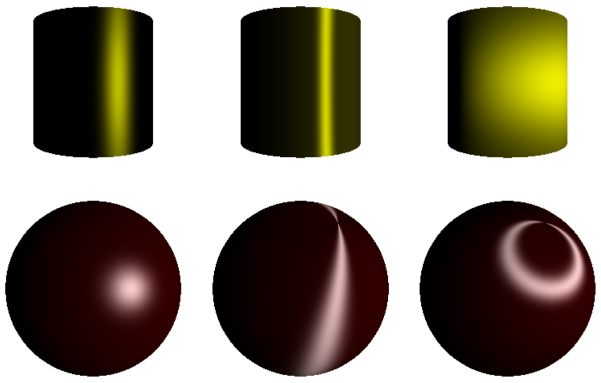

Figure 4: Sphere and cylinder with isotropic reflection (left), anisotropic reflection with radial features (center), and vertical features (right).

This modified method slightly increases the cost of the rendering algorithm. Now, not only texturing, but also lighting has to be performed by OpenGL.

4 Multipass Techniques

So far, we have only considered the illumination through a single light source. If we add a second light source, say in direction L', the illumination becomes a function in three variables: $I_o\left(\langle L, T\rangle, \langle L', T\rangle, \langle V, T\rangle\right)$, which could be represented as a 3-dimensional texture map. 3-dimensional texturing is available as an extension on many OpenGL platforms today, and is proposed to become a standard feature of OpenGL version 1.2 [1]. The modified texture matrix for two light sources is as follows:

$$\frac{1}{2}\begin{bmatrix} l_x & l_y & l_z & 1 \\ l'_x & l'_y & l'_z & 1 \\ v_x & v_y & v_z & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Unfortunately, 3-dimensional textures can grow quite large, which, in turn, causes performance problems. Also, this method fails for more than two light sources. A better way to illuminate anisotropic surfaces with more than one light source is to use a multipass approach.

For this multipass method, the object is rendered multiple times, once for each light source. The results of each rendering pass are added to the image using alpha blending. Obviously, the rendering time increases linearly with the number of light sources. Nonetheless, for a relatively small number of lights, or a small percentage of anisotropic surfaces, the method is still fast, even on low end machines.

Another application of multipass techniques is the addition of isotropic reflection components. The model described so far is purely anisotropic. Many surfaces, however, have both an isotropic and an anisotropic reflection component, for example when the scratches or fibers on the surface have a certain, non-zero distance from each other. A single rendering pass with the standard OpenGL lighting model can add this isotropic component for all the light sources in the scene.

Other surfaces have anisotropies in multiple directions. For example woven cloth consists of fibers oriented in two perpendicular directions. Again it is possible to use multiple passes, one for each direction and light source, to account for these effects.

5 Conclusion

In this paper we have presented a novel approach for rendering surfaces with anisotropic reflection characteristics in OpenGL. The method requires one rendering pass per light source, and each pass is no more expensive than rendering the same object with traditional lighting and texture mapping switched on. Therefore, the method is well suited even for PCs and low end workstations, as long as the number of light sources is not too large.

References

[1] OpenGL ARB. *OpenGL Specification, Draft Version 1.2*, 1997.

[2] David C. Banks. Illumination in diverse codimensions. In *Computer Graphics (Proceedings of Siggraph '94)*, pages 327–334, July 1994.

[3] Agnes Daldegan, Nadia Magnenat Thalmann, Tsuneya Kurihara, and Daniel Thalmann. An integrated system for modeling, animating, and rendering hair. In *Eurographics '93*, pages 211–221, 1993.

[4] Ken ichi Anjyo, Yoshiaki Usami, and Tsuneya Kurihara. A simple method for extracting the natural beauty of hair. In *Computer Graphics (Proceedings of Siggraph '92)*, pages 111–120, July 1992.

[5] James T. Kajiya. Anisotropic reflection models. In *Computer Graphics (Proceedings of Siggraph '85)*, pages 15–21, 1985.

[6] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide*. Addison Wesley, 1993.

[7] Pierre Poulin and Alain Fournier. A model for anisotropic reflection. In *Computer Graphics (Proceedings of Siggraph '90)*, pages 273–282, August 1990.

[8] Morgan Schramm, Jay Gondak, and Gary Meyer. Light scattering simulations using complex surface models. In *Graphics Interface '97*, pages 56–67, 1997.

[9] 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.

[10] Gregory J. Ward. Measuring and modeling anisotropic reflection. *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 265–273, July 1992.

[11] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Interactive visualization of 3D-vector fields using illuminated stream lines. In *IEEE Visualization '96*, pages 107–113, 1996.