Shading and Shadowing with Linear Light Sources

Pierre Poulin† and John Amanatides‡

†Imager,
Department of Computer Science,
University of British Columbia,
Vancouver, British Columbia
Canada V6T 1W5
(604) 228-2218
poulin@cs.ubc.ca

‡Department of Computer Science
York University
4700 Keele St.
North York, Ontario
Canada M3J 1P3
(416) 736-5053
amana@yetti.yorku.ca

In virtually all rendering systems, linear light sources are modeled with a series of point light sources that require considerable computing resources to produce realistic looking results. A general solution for shading surfaces illuminated by a linear light source is proposed. A formulation allowing for faster computation of the diffuse component of light reflection is derived. By assuming Phong's specular component, simple, inexpensive and convincing results are produced with the use of a Chebyshev approximation. A shadowing algorithm is also presented. As shadowing from linear light sources is expensive, two acceleration schemes, extended from ray tracing, are evaluated.

1 Introduction

One of the requirements for generating realistic images is the capability of simulating a wide variety of light sources. Small changes in the intensity patterns of these illuminants can significantly effect the appearance of a scene. To get the right effect, artists can easily add dozens of lights. Light which originates from neon lights can be simulated by linear light sources. In this paper we introduce two solutions for rendering surfaces illuminated by linear light sources. The first is an analytic solution that is exact though a little expensive. The second allows us to compute the effects of these light sources inexpensively yet avoiding the sampling problems of standard approaches. When extending the light source to a line, the shadows have to be handled differently in order to capture the variation of intensity within the shadow region. An algorithm to compute the umbra and penumbra regions of shadows is presented. This algorithm allows the objects in a scene to not be limited to polygons only. We first review shading and the various light sources currently in use in computer graphics. The two solutions are then derived and the shadowing algorithm is introduced. Finally, results are presented and discussed.

2 Concepts and Previous Work

2.1 Shading

The light reflection off a surface can be broken down into two components: diffuse and specular[1]. When light hits an ideal diffuse surface, it is re-radiated equally in intensity in all directions. Chalk and flat paints are examples of real surfaces that re-radiate light mostly in a diffuse way. Purely specular surfaces only re-radiate light in one direction, the reflected light direction. Mirrors are

[1]This division of reflection into two components is usually well accepted amongst the computer graphics community. In fact, highly diffuse or specular material are closely approximated by this subdivision. However it is important to mention that reflection off some surfaces cannot adequately be represented this way. The more general bidirectional reflection functions [Cabr87] should be used to characterise these surfaces.

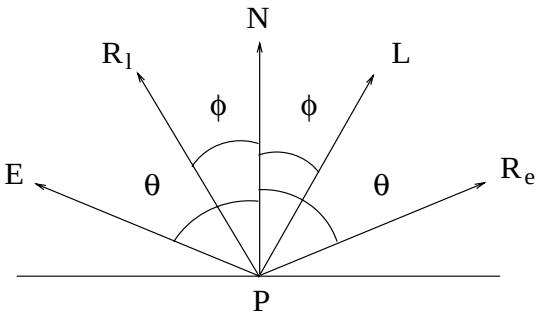

Figure 1: General Reflection

examples of specular surfaces. A physical explanation of the difference between these two components is that light bounces off a specular surface while for a diffuse surface, light penetrates the surface and is scattered internally before emerging again. The reflection of the light from real objects contains both diffuse and specular components and both must be modeled to create realistic images. Consider figure 1. $\vec{E}$ and $\vec{L}$ are unit vectors that point to the eye and to a point on a light source, respectively, $\vec{N}$ is the unit vector normal to the surface at P and $\vec{Re}$ and $\vec{Rl}$ point in the reflected eye direction and reflected light direction, respectively. Computing the diffuse component is very simple; it is proportional to $\vec{N} \cdot \vec{L}$, which is the well-known Lambert's Law. Note that in diffuse reflection, since light is radiated equally in all directions, the position of the eye is not required by the computation and the maximum intensity occurs when a surface facing the light is perpendicular to the light source direction.

The specular component is harder to compute. Real objects are non-ideal specular reflectors and some light is also reflected slightly off axis from the ideal reflected light direction ($\vec{Rl}$). A possible explanation is that a real surface is never perfectly flat but contains microscopic deformations. Popular models of the specular component have been proposed by Phong [Phon75], Blinn [Blin77] and Cook and Torrance [Cook81]. Our selection of a particular illumination model (Phong's $(\vec{Re} \cdot \vec{L})^n$) for modeling linear light sources was based on its relative accuracy with reality and the possibility to compute and integrate it at a moderately low cost.

2.2 Light Sources

The simplest light source in use in computer graphics is the directional light source. This models the parallel light coming from an infinitely distant light source. There are several reasons for its use. First, the vector $\vec{L}$, which points to the light source, is constant and does not have to be computed at each point to shade. This vector, in fact, defines the directional light source. Because the vector $\vec{L}$ is constant, the shading computation for polygons can be simplified as many computations can be removed from the inner loops [Phon75]. Also, the intensity of the light source is constant and independent of the position of an object in space.

The second most popular light source is the point light source, which is defined geometrically by a point in space. The following equation can be used to model the shade of a surface when illuminated by a point source:

$$I_{pixel} = \frac{I}{r^2}\Big(k_d(\vec{N} \cdot \vec{L}) + k_s R_s(\vec{E}, \vec{N}, \vec{L})\Big)$$

where k_d and k_s determine the ratio of the diffuse and specular reflection, respectively, and R_s is a function defining the specular intensity reflected. To compute $\vec{L}$ we subtract the position of the surface we are shading from the point source position and normalise the resulting vector. The intensity of this light source is proportional to $\frac{1}{r^2}$ though confusion on exactly how to apply this formula has been reported [Fole82]. Some people include the distance from a surface to the eye in the intensity calculation. This is not necessary. As a surface moves further away from the viewer, the increased distance is compensated by the increase surface area that now occupies the pixel. Thus, the only factor that must be taken into account is the distance from the light source to the surface.

The directional and point light sources are generally modeled in most rendering systems due to their simplicity. Unfortunately, they are difficult to use creatively. Researchers have introduced modifications to the intensity distribution of point sources that give the designer more creative freedom. We outline some of these attempts below.

Warn [Warn83] attempts to create more realistic light sources by mimicking the lights used by photographers. His extensions include making the intensity of the light source a function of direction (to produce spotlights) and providing flaps that can cut off the light in certain directions. Verbeck and Greenberg [Verb84] and Nishita et al. [Nish85] continue Warn's work by providing more sophisticated lighting design tools. They allow the user to specify the shape of the intensity distribution by drawing goniometric diagrams[2].

Other kinds of light sources have also been defined. For instance in cone tracing [Aman84], spherical lights have been simulated. In various attempts to solve the global illumination with the radiosity algorithm [Gora84] [Cohe85], lights are formed of polygons. In another rendering algorithm driven by light propagation [Four89], any type of light can be simulated. However in all these approaches, lights are inherent to a particular rendering algorithm and as such, cannot easily be generalised to other common rendering techniques.

Linear light sources simulating neon-like lights open a new dimension to the effects that can be produced. These lights can be modeled by the following expression:

$$I_{pixel} = \int_{length} \frac{I_l}{r_l^2} \Big(k_d(\vec{N} \cdot \vec{L_l}) + k_s R_s(\vec{E}, \vec{N}, \vec{L_l}) \Big)\, dl$$

where l is the variable of integration. By assuming that the intensity I_l is constant over the whole length ($length$) of the light source, we can take out the intensity from the integral part. This integral can also be broken down into diffuse and specular components:

$$I_{pixel} = Ik_d \int_{length} \frac{(\vec{N} \cdot \vec{L_l})}{r_l^2}\, dl + Ik_s \int_{length} \frac{R_s(\vec{E}, \vec{N}, \vec{L_l})}{r_l^2}\, dl$$

The linear sources were modeled by Verbeck and Greenberg [Verb84] by a series of collinear point sources. Area sources were similarly modeled. There are some problems associated with this approach to model linear light sources. For instance, a large number of point sources must be used or sampling problems will ensue. These problems will be related to the shading of a surface as well as the shadowing within the penumbra region. Since shading and shadowing computations are very important parts of the rendering process, this approach can be very expensive.

Nishita et al. [Nish85] use a slightly different formulation than the one introduced in this paper. They derive an analytic solution for the diffuse component if the light source is parallel or perpendicular to the surface. However, multiple point sources are used to approximate the specular integral.

By looking more closely at the reflection expressions given above, it becomes possible to extract an analytic solution from them. In the next sections we derive our solutions for the diffuse and the specular integrals.

3 The Diffuse Integral

To solve the diffuse integral we transform the light source into a coordinate system in which the surface is at the origin and the surface normal $\vec{N}$ is along the Z axis. In this coordinate system, $\vec{N} \cdot \vec{L_l}$ can be replaced by $\frac{z_l}{r_l}$ and thus the diffuse integral is transformed to:

[2]Goniometric diagrams specify relative intensity as a function of direction. See the IES Lighting Handbook [Kauf81] for a good source of lighting definitions. Note: the user must exercise care when drawing these diagrams since sharp discontinuities in the resulting curves are a source of aliasing artifacts.

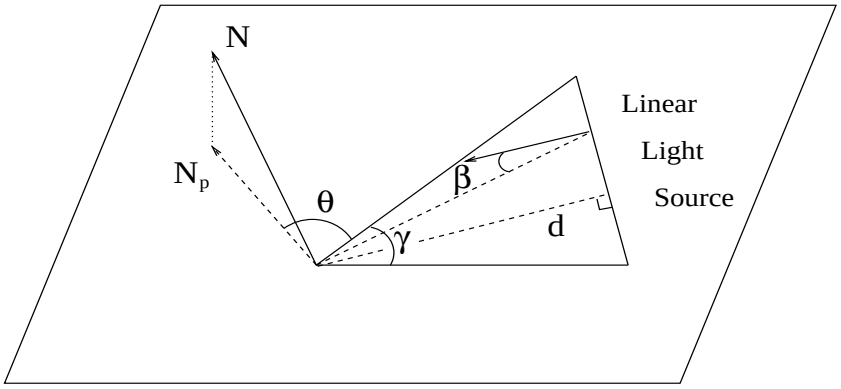

Figure 2: Integrating over the angle γ

$$\int_{length} \frac{z_l}{r_l^3}\, dl \qquad \text{or} \qquad \int_{length} \frac{z_l}{\sqrt{x_l^2 + y_l^2 + z_l^2}^3}\, dl.$$

Let the light source be defined by $(\vec{u} + t\vec{v})$ for $0 \leq t \leq 1$. By substituting the light source line equation into the integral, we have an integral of the following form that can be solved analytically [Grad65] as:

$$\int_0^1 \frac{Dt + E}{\sqrt{At^2 + Bt + C}^3}\, dt = \left. \frac{2(D(Bt + 2C) - E(2At + B))}{(B^2 - 4AC)\sqrt{At^2 + Bt + C}} \right|_0^1.$$

4 The Specular Integral

To lay the groundwork for the solution of the specular integral we will redevelop the diffuse integral in a different coordinate system. In this coordinate system, the solution is more expensive than the one developed above. Nevertheless, its development will simplify the explanation of our solution to the specular integral.

Let us consider the problem in 2D. We transform the coordinate system such that P is at the origin and the light source lies on the plane $z = 0$. The diffuse integral can be expressed in this system as

$$\int_{length} \frac{(\vec{N} \cdot \vec{L_l})}{r_l^2}\, dl = (\vec{N} \cdot \vec{N_p}) \int_{length} \frac{(\vec{N_p} \cdot \vec{L_l})}{r_l^2}\, dl$$

where $\vec{N_p}$ is the projection of $\vec{N}$ onto the plane $z = 0$.

Take an infinitely small segment dl on the light source. As seen from P, this dl has a length $dq = dl \cos\beta$ where β is the angle between the direction perpendicular to dl and the direction from dl to P. On another hand, take a circle of radius r_l centered at the origin. The length of the arc of $d\phi$ is $dq = r_l\, d\phi$. By combining these two equations, we can express dl as a function of $d\phi$ as

$$dl = \frac{r_l\, d\phi}{\cos\beta}.$$

So now, we can replace the integral along the length of the light source by an integral along the angle subtended by the light source at P. If γ is the angle made by joining the two end points of the light source to P (two light vectors) and θ is the angle between $\vec{N_p}$ and the closest light vector[3], the integral becomes

[3]If $\vec{N_p}$ is within γ, we can take any end points light vector. In such a case, θ is negative.

$$(\vec{N} \cdot \vec{N_p}) \int_{\theta}^{\theta+\gamma} \frac{\cos\phi}{r_l^2} \frac{k\, r_l}{\cos\beta}\, d\phi \tag{1}$$

Figure 2 illustrates this situation.

In this new formulation, k, the intensity of the light source per unit length, has been added for more flexibility to the diffuse integral previously given. Expressing r_l as a function of ϕ gives $r_l = \frac{d}{\cos\beta}$ where d is the distance between the infinite line along the light source and P.

This formulation is valid to model an infinity of point light sources over a linear light source where each point on the light source radiates light equally in all directions. This final integral is

$$\frac{k(\vec{N} \cdot \vec{N_p})}{d} \int_{\theta}^{\theta+\gamma} \cos\phi\, d\phi. \tag{2}$$

Similarly, assuming the function $R_s = (\vec{Re} \cdot \vec{L})^n$ is used to represent the specular reflection component, the specular integral is established as

$$\frac{k(\vec{Re} \cdot \vec{Re_p})^n}{d} \int_{\theta}^{\theta+\gamma} \cos^n\phi\, d\phi. \tag{3}$$

It is important to note that if P is along the line formed by the light source, the integral is undeterminate (γ and d are both zero). In this situation, the integral can be easily handled as a special case.

4.1 Chebyshev Approximation

The integral for the diffuse term has an exact solution that is easily solved. However, the integral for the specular term is harder to integrate exactly in an efficient manner.

An alternative is to approximate $\cos^n\phi$ between 0 and $\frac{\pi}{2}$ since it is nicely behaved. Several approximation techniques can be used (various spline bases, etc.), but in our case, we chose the Chebyshev approximation for its simplicity and its property to return, for a given degree, the best polynomial approximating a curve.

In order to reduce the degree of the polynomial and since we deal with pixels with discrete values, we cut the function $\cos^n\phi$ at the angle where function is less than a chosen ϵ. This value is function of the intensity of the light source. Then the curve evaluated at a value of ϕ higher than this value would be considered zero. In practice, we have found that a polynomial of degree 6 approximates within $\epsilon = \frac{1}{256}$ the specular intensity curve[4]. The degree of the polynomial can be reduced if the ϵ is larger. Then for each of the coefficients n in a scene, the Chebyshev polynomials are computed and stored. They are used at the rendering stage to easily approximate the specular integral.

5 Shadowing

The shadowing of linear light sources is very important to achieve a better realism. The solution of simulating the linear light by a series of collinear points is still an alternative but as mentioned earlier, the shadowing would be subject to sampling problems. Nishita et al. [Nish85] describe an algorithm for a scene made exclusively of polygons. Given a linear light source and a polygon casting a shadow on a surface, the polygon vertices are projected onto the surface, taking as center of projection the two end points of the light. Once the contour lines of both shadows are determined, the convex hull of these shadow polygons is established and stored. Later during the scanline process, if a point is inside a penumbra region, the polygons casting these shadows are projected back onto the light

[4]If we consider a light source with an intensity of 1 unit and a given function $\cos^n\phi$ approximated by a degree 6 polynomial, the choice of $\epsilon = \frac{1}{256}$ insures us that the intensity computed using the Chebyshev polynomial will have an approximative maximum error of $\frac{1}{256}$.

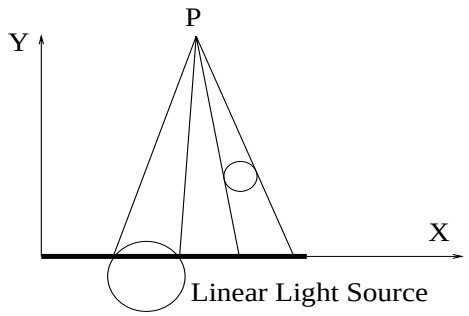

Figure 3: Visibility of the Light Source

source and the shading is computed for only the segments of the light source that are visible from the surface.

Our approach is similar to this algorithm but extends the scene to other primitives. When point P has to be shaded, the light source is first *cut* by the plane tangent to the normal at P because no light can shine on P from an angle of more than $\frac{\pi}{2}$. If a portion of the light source is visible from P, a triangle (the *light triangle*) is formed by the end points of the linear light and P. If an object intersects this triangle, its intersection is projected onto the light and the shading is computed only for the *visible* segments of the light. In the case of the specular integral, the visible segments must also be *cut* such that for all segment i, $|\theta_i| < \frac{\pi}{2}$ and $|\theta_i + \gamma_i| < \frac{\pi}{2}$ in equation 3.

Let O be an object that may cast a shadow on P. The first test consists in intersecting the plane containing the light triangle (the *light plane*) with object O. If the computations involved are expensive, the test can be done onto the bounding volume of the object. If it is determined the object does not intersect the light plane, then object O does not cast any shadow on P. Otherwise, the light triangle is first transformed into the object coordinate system where more accurate tests can be performed. Afterwards, if indeed the object O intersects the light plane, it is transformed in such position that the light lies on the X axis with one end at the origin and the intersection point P lies on the plane $z = 0$ on the positive side of Y (see figure 3). This involves only a series of rotations and translations, leaving the basic definition (shape) of the primitive unchanged. The remaining work consists in identifying the intersection between a primitive and the plane $z = 0$ and projecting in 2D this intersection onto the X axis.

Let us take a sphere as an example. The sphere can be transformed so it is not spherical any more. The light triangle is therefore transformed in the sphere's own coordinate system where it is easier to determine if the sphere intersects the light plane. The intersection between the plane and a perfect sphere is either null, a point or a circle. In the first two cases, no shadow is produced. For a circle, the two tangent points from P or the intersection between the circle and the X axis defines the visible segments of the light source. Figure 3 illustrates these two cases.

For simple objects like polygons, the transformation in the object coordinate system is not required and the transformation matrix bringing the light triangle onto the plane $z = 0$ needs to be computed only once per intersection point P. For certain objects like patches, this whole process can be very complicated. In our implementation, when such a situation occurs, we simply rely on shooting rays to approximate the projection of the object onto the light, thus avoiding any object transformation. The number of rays is determined as a function of the angle at P (with the two end points of the light). In a first pass, a regular set of rays are shot to roughly determine the two edges (assuming convex objects only) of the projection. In a second pass, additional rays are used to refine the projection edges. At this stage, some jittering is added to the ray directions, substituting aliasing by noise, which was very effective within the penumbra region.

It is important to note here that as soon as the whole light is hidden, the shadowing process is stopped. The intersection/projection scheme described above can be relatively expensive. It is therefore essential to eliminate the objects not intersecting the light triangle as quickly as possible. The next section will describe two algorithms that we implemented to reduce the number of objects candidate to cast a shadow on the point to be shaded. Some results will also be discussed.

Table I: Linear vs Points

Primitive	Shading Only		Shading and Shadowing	
	Function	Equivalent	Function	Equivalent
Square	$0.39 + 0.14\ l$	4.36	$0.16 + 0.10\ l$	8.35
Cube	$0.34 + 0.11\ l$	6.00	$0.15 + 0.10\ l$	8.50
Sphere	$0.31 + 0.11\ l$	6.27	$0.16 + 0.10\ l$	8.40
Disk	$0.40 + 0.14\ l$	4.29	$0.09 + 0.06\ l$	16.28
Patch	$0.50 + 0.08\ l$	6.25	$0.06 + 0.04\ l$	23.50

6 Results of the Shading and Shadowing Algorithms

Replacing a linear light source by a series of point light sources is, as we said earlier, prone to sampling problems. The problems can appear on both the shading and the shadowing. Table I investigates the relative cost of substituting a linear light source as described in this paper by a series of point light sources. Plates 1 and 2 illustrates the typical scene we used.

A linear light source of length 10 is positioned at 5 units above a primitive located at 1 unit above a plane. Each primitive has been scaled so the shadow area would cover a similar area. The specular coefficient n for $(\vec{Re} \cdot \vec{L})^n$ is 64 for the primitive and 32 for the plane. The relative timings are given from our implementation on our local ray tracer. The timings are normalised by the time required to render the same scene with a linear light source. For instance, rendering a sphere (ray casting) in our scene without any light source takes only 31% of the time required to shade it with a linear light source. If point light sources are used, 11% must be added for each light. Therefore in our benchmark, substituting a linear light source by more than 7 point light sources would be more expensive. If shadows are considered, this number increases to 9. Plate 1 was computed with 7 point light sources and plate 2 with our approximate linear light source. On a 24-bit planes display, the same scene was rendered with as much as 41 point light sources and Mach bands within the penumbra region were still visible.

For a patch primitive, the number of point light sources equivalent, cpu-wise, to a linear light source is as high as 24 because we relied on using additional rays (two passes solution described in the previous section) to determine the projection of the patch onto the light source.

6.1 Speeding up Shadow Determination

In the scene used in Table I, only two objects are used. As the number of objects increases, the complexity of shadowing increases as the square of the number of objects. Avoiding to compute unnecessary and expensive projections onto the light source becomes primordial.

We present two different schemes we implemented to achieve this goal. First, we use the regular grid traversal described in Amanatides and Woo [Aman87]. Once an intersection point is found, a light triangle is formed. This light triangle is then scan-converted in the 3D grid of the scene. The objects occupying a same voxel than the light triangle are gathered and tested for uniqueness. This smaller set of objects is then tested and projected onto the light source if necessary.

In our second scheme, a modified light buffer for linear light sources has been implemented. Our linear light buffer can be represented by an infinite cylinder oriented along the light. This cylinder is subdivided in arcs along its radius. Each arc has a list of objects contained (at least in parts) within the limit of its angle. To improve the performance of the linear light buffer, we also divide it in three regions: the `left`, `center` and `right` side of the light source. This is illustrated in figure 4. When an intersection point P is found, it is located within an angular section of the linear light buffer. If it is positioned in the `center` region, only the objects at least partly in this angular section and in the `center` region need to be tested. If P is in the `left` region, only the objects in the `left` and

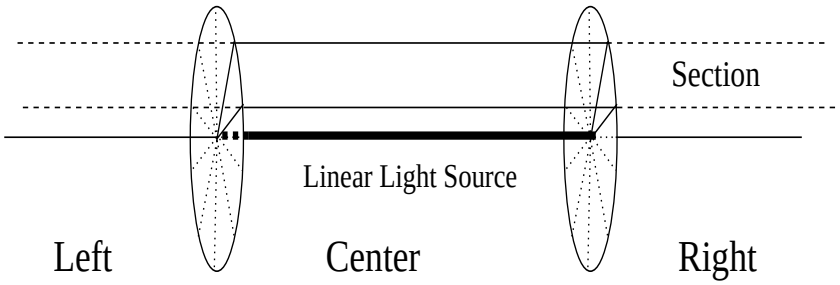

Figure 4: The Linear Light Buffer

`center` regions need to be tested, and similarly for the `right` region.

In the scan-conversion of the light triangle in the 3D grid, the only additional memory is a link list of pointers to the objects. No preprocessing is required but the scan-conversion process is expensive. Table II and III illustrate our results for two test scenes[5]. In Table III, the rendering times have been normalised by the time required to render the same scene with a grid of 5^3 voxels and without using the scan-conversion of light triangles. In the tetrahedron scene, having a grid resolution of 5^3 is 25%more expensive than having no grid at all. This is due to the facts that just a few voxels contain lots of small triangles and that identifying if a triangle intersects the light plane is relatively a cheap process. However when the grid resolution increases, the number of tests on triangles is greatly reduced in comparison with the extra processing of scan-conversion and gathering of the candidates, reducing the rendering cost by as much as 75%. Notice that increasing the grid resolution does not result always in a speed up factor. In the tetrahedron scene, a grid of 50^3 is less cost effective than 25^3. The sphere flakes scene is more problematic and shows some unstability with the algorithm of scan-converting the light triangle. The bottom square is much larger than the spheres that are agglomerated in the center of the scene. Because of this situation, the scan-conversion is highly sensible to variations in the grid resolution, as demonstrated by its non-predictable results.

In the linear light buffer, the pointers to the objects for each region within each section of each linear light buffer need to be allocated. The assignment of an object to the appropriate sections and regions is done in preprocessing and a simple location of the right section and region is added to the rendering process. An adaptive subdivision of the angular sections can be extended for the linear light buffer but this has not been done yet in our implementation.

In Table III, the interest of the linear light buffer is evaluated. All the rendering times are normalised by the rendering time using no linear light buffer (a single section per linear light buffer) and no grid. It is interesting to note that unlike the scan-conversion of the light triangle, this algorithm is very stable for both scenes. As the grid resolution increases, the ratio of the ray intersection time over the whole rendering time decreases and as such the importance of the speeding up of the linear light buffer is better shown.

Table II: Light Triangle

Database	Light Triangle	Grid Resolution					
		5^3	10^3	15^3	20^3	25^3	50^3
flakes	Inactive	1.0000	0.9059	0.9022	0.8863	0.8742	0.8629
	Active	1.2342	0.7969	0.9722	0.8245	0.8696	1.5742
tetrahedron	Inactive	1.0000	0.8926	0.8719	0.8657	0.8615	0.8642
	Active	1.2532	0.5239	0.2925	0.2689	0.2385	0.3740

[5]These two scenes, the sphere flakes and the tetrahedron, have been suggested by Haines [Hain87] in an attempt to help benchmarking rendering techniques. The sphere flakes consists of 91 spheres defined recursively above a square while the tetrahedron is represented by 4096 triangles forming a pyramid. To test our algorithms, a linear light source is located above each scene.

Table III: Linear Light Buffer

Database	Grid Resolution	Linear Light Buffer Resolution						
		1	24	36	72	144	360	720
flakes	1^3	1.0000	0.6656	0.6291	0.5751	0.5505	0.5332	0.5281
	5^3	1.0270	0.6834	0.6449	0.5916	0.5678	0.5471	0.5497
	10^3	0.9304	0.5809	0.5446	0.4876	0.4676	0.4479	0.4466
	15^3	0.9266	0.5782	0.5392	0.4858	0.4644	0.4449	0.4411
	20^3	0.9103	0.5606	0.5229	0.4796	0.4481	0.4280	0.4258
	25^3	0.8979	0.5486	0.5112	0.4688	0.4347	0.4156	0.4130
	50^3	0.8863	0.5366	0.4974	0.4579	0.4212	0.4031	0.3990
tetrahedron	1^3	1.0000	0.9732	0.9508	0.9425	0.9236	0.9120	0.9127
	5^3	0.1429	0.0455	0.0410	0.0359	0.0333	0.0318	0.0314
	10^3	0.1275	0.0303	0.0256	0.0206	0.0180	0.0166	0.0160
	15^3	0.1246	0.0418	0.0227	0.0177	0.0151	0.0138	0.0132
	20^3	0.1237	0.0384	0.0223	0.0168	0.0142	0.0128	0.0123
	25^3	0.1231	0.0341	0.0214	0.0163	0.0138	0.0122	0.0117
	50^3	0.1235	0.0534	0.0209	0.0159	0.0133	0.0118	0.0114

7 Conclusions

In this paper, solutions have been presented for the diffuse and specular components of surface shading when illuminated by linear light sources. These solutions are also valid for any position of the light source and any type of primitives. On a simple test scene, we observed that for a few primitives, shading and shadowing with a linear light source is equivalent, cpu-wise, to replace the linear light source by 10 point light sources. However using this few point light sources results in pictures showing strong aliasing within the penumbra region. For primitives as complicated as patches, the equivalent number of point light sources required is around 25.

An algorithm is introduced to correctly handle shadowing with more complex primitives than polygons. This algorithm adds more flexibility in the primitives casting shadows from a linear light source, and this, at the cost of more expensive computations. In order to reduce the additional cost of using this shadowing algorithm, two algorithms based on ray tracing's efficiency schemes are given. The scan-conversion algorithm has the advantage of requiring only an additional link list of pointers to the objects. However the scan-conversion process is rather expensive and it is difficult to evaluate the dimension of the grid subdivision that would provide a good speed up of the shadowing calculations. The linear light buffer requires additional memory for each linear light source. However its stability and the speed up observed made us choose this method for rendering many of our scenes.

For certain primitives, determining the visible portion of the linear light source is very complicated. A general approach based on shooting rays to determine the visible portion of the light is used. With such a process, we expect that the tradeoff of both algorithms over no culling will become even more worthwhile.

The overall result is more flexibility for artists to create special effects with commonly used rendering techniques (see Plates 3 and 4). The authors are investigating whether similar approaches can be used for other 2D and 3D light sources.

8 Acknowledgements

The authors would like to thank Alain Fournier, John Buchanan and Andrew Woo for their suggestions. They are also grateful to NSERC and OGS for their financial support.

References

[Aman84] Amanatides, J., "Ray Tracing with Cones", *Computer Graphics*, (Proc. SIGGRAPH 84), Vol. 18, No. 3, July 1984, pp. 129-135.

[Aman87] Amanatides, J. and Woo, A., "A Fast Voxel Traversal Algorithm for Ray Tracing", *Eurographics 87*, August 1987, pp. 1-10.

[Blin77] Blinn, J., "Models of Light Reflection for Computer Synthesized Pictures", *Computer Graphics*, (Proc. SIGGRAPH 77), Vol. 11, No. 2, July 1977, pp. 192-198.

[Cabr87] Cabral, B., Max, N. and Springmeyer, R., "Bidirectional Reflection Functions from Surface Bump Maps", *Computer Graphics*, (Proc. SIGGRAPH 87), Vol. 21, No. 4, July 1987, pp. 273-281.

[Cohe85] Cohen, M.F. and Greenberg, D.P., "The Hemi-Cube, A Radiosity Solution for Complex Environments", *Computer Graphics*, (Proc. SIGGRAPH 85), Vol. 19, No. 3, July 1985, pp. 31-40.

[Cook81] Cook, R., Torrance, K., "A Reflectance Model for Computer Graphics", *Computer Graphics*, (Proc. SIGGRAPH 81), Vol. 15, No. 3, August 1981, pp. 307-316.

[Fole82] Foley, J.D. and Van Dam, A., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, 1982.

[Four89] Fournier, A., Fiume, E., Ouellette, M. and Chee, C., "FIAT: Light Driven Global Illumination", DGP Technical Memo DGP89-1, Dynamic Graphics Project, University of Toronto, 1990.

[Gora84] Goral, C.M., Torrance, K.E., Greenberg, D.P. and Battaile, B., "Modeling the Interaction of Light Between Diffuse Surfaces", *Computer Graphics*, (Proc. SIGGRAPH 84), Vol. 18, No. 3, July 1984, pp. 213-222.

[Grad65] Gradshteyn, I.S. and Ryzhik, I.M., *Table of Integrals, Series and Products*, Academic Press, New York, 1965.

[Hain87] Haines, E.A., "A Proposal for Standard Graphics Environments", *IEEE Computer Graphics and Applications*, Vol. 7, No. 11, November 1987, pp.3-5.

[Kauf81] Kaufman, J.E. and Haynes, H., *IES Lighting Handbook*, Illuminating Engineering Society of North America, New York, 1981.

[Nish85] Nishita, T., Okamura, I. and Nakamae, E., "Shading Models for Point and Linear Sources", *ACM Transaction on Graphics*, Vol. 4, No. 2, April 1985, pp. 124-146.

[Phon75] Phong, B., "Illumination for Computer Generated Pictures", *Communications of the ACM*, Vol. 18, No. 6, June 1975, pp. 311-317.

[Verb84] Verbeck, C.P. and Greenberg, D.P., "A Comprehensive Light-Source Description for Computer Graphics", *IEEE Computer Graphics and Applications*, Vol. 4, No. 7, July 1984, pp. 66-75.

[Warn83] Warn, D.R., "Lighting Controls for Synthetic Images", *Computer Graphics*, (Proc. SIGGRAPH 83), Vol. 17, No. 3, July 1983, pp. 13-21.

Plate 1: A sphere under 7 point light sources

Plate 2: A sphere under a linear light source

Plate 3: A typical cafeteria lit by 12 linear light sources

Plate 4: A desk under a linear light source