# Lights from Highlights and Shadows

Pierre Poulin
Alain Fournier

Department of Computer Science
University of British Columbia
{poulin | fournier} @cs.ubc.ca

## Abstract

Designing the illumination of a scene is a difficult task because one needs to render the whole scene in order to look at the result. Obtaining the correct lighting effects may require a long sequence of modeling/rendering steps. We propose to use directly the highlights and shadows in the modeling process. By creating and altering these lighting effects, the lights themselves are indirectly modified. We believe this new technique to design lighting is more intuitive and can lead to a reduction of the number of modeling/rendering steps required to obtain the desired image.

**CR Categories and Subject Descriptors:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism. Interaction techniques.
**General Terms:** Algorithms.
**Additional Key Words and Phrases:** extended light source, shadow volume, soft shadows, hard shadows, interactive light modeling.

## 1 Introduction

An important research area of computer graphics consists in simulating realistic pictures. Reality is modeled by observing and measuring its attributes. In a next step, the models are rendered onto an image. In that sense, computer graphics models the causes and renders the effects onto an image. On the other hand, computer vision is interested in analyzing an image. It tries to isolate certain effects in an image in order to identify the causes. While the two processes might seem to go on totally opposite directions, it is interesting to consider how advances in one direction might actually help the reverse process.

In computer vision, highlight information has been used to determine light direction or local shape orientation. Babu et al. [babu85] study contours of constant intensity in an image to determine the orientation of planar surfaces under the illumination of a directional light source. Buchanan [buch87] fits ellipses to the highlights to obtain the same information for planar surfaces illuminated by point light sources.

One important aspect of most of these algorithms consists in identifying the highlight area. This is not an easy task as many of the algorithms for shape from shading [horn88] require almost entirely diffuse surfaces.

When techniques are not restricted to diffuse surfaces, they often rely on some kind of thresholding. The unfortunate reality with thresholding is that different values of threshold can lead to relatively different shape of the highlight and therefore, to different shape/light recovery. Other techniques like Wolff's use of polarization [wolf91] are promising although require the presence of polarizing lenses on the cameras capturing the scene.

Much useful information can also be extracted from the shadow areas in an image [walt75] [shaf85]. These areas provide additional information on the shape of the object casting a shadow and even on the shape of the object on which the shadow is cast. Moreover, they provide information on the direction and the shape of the light sources. Unfortunately, very little work has been involved in recovering the shape of an extended light source, as recovering shape from shading under a directional or a point light source is already a difficult task.

Shadows are not easy to extract from an image. Detecting shadows can be done in a similar way than edge detection by applying various edge enhancing filters. For extended lights, the shadow edges are soft and the shadow must be detected based on changes in the gradients of the shading. Gershon [gers87] use gradients in color space to determine if the region corresponds to a shadow region or simply to a change of material. Textures can also defeat most of the techniques and must be carefully handled.

While modeling a scene, a user has access to important information unavailable to computer vision, i.e. the geometry of the scene and the viewing projection parameters. To better understand a 3D scene, the user can therefore move the camera around, use at the same time several views of the same scene, move objects, remove hidden surfaces, and all of this in real time; however, so far, few applications use information about highlights and shadows in order to improve on the modeling step in computer graphics.

This paper proposes to investigate how we can use highlight and shadow information in order to help a user to define the shape and position of a light source. It does not preclude the previous ways of defining and positioning the light sources, but enhances the whole process.

## 2 Defining and Manipulating Light Sources

With the advent of high performance graphics hardware, it becomes possible to interactively create and manipulate more and more complex models with a higher degree of realism. Yesterday's simple wireframe models can now be replaced by flat shaded polygons, Gouraud shaded and even Phong shaded polygons, allowing for real time interaction with the models. Hanrahan and Haeberli [hanr90] demonstrate with their system how today's graphics hardware could be used to "paint" textures and various other surface parameters (transparency, perturbation of surface normals, etc.) in a fully interactive system. This increase in rendering power provides us with the possibility to investigate light definition and manipulation from the highlights and shadows it produces.

### 2.1 Lights from Highlights

In this section, the process of defining a light from its highlights is described. Its advantages are demonstrated and its restrictions explained so one could better understand the implications of using such a process.

Highlights are usually defined in the reflection models by the specular term. Consider the specular term of Phong's shading [phon75] as expressed by Blinn [blin77]:

$$\left(\vec{N} \cdot \vec{H}\right)^n \tag{1}$$

where $\vec{N}$ is the surface normal at a given point[1]
$\vec{H}$ is the bisector vector of the eye direction and the light direction
$n$ is the surface roughness coefficient.

This formulation tells us that for a given point on the surface specified as the *maximum intensity* of the highlight, a unique directional light source can be determined as

$$\vec{L} = 2\left(\vec{N} \cdot \vec{E}\right)\vec{N} - \vec{E} \tag{2}$$

where $\vec{E}$ is the eye direction.

The term *maximum intensity* is not properly correct if we think of it in the context of a complete shading model. However we will use it here meaning maximizing equation (1). It is interesting to note that other points on the surface might reach this maximum but will never surpass it.

This simple relationship between the maximum intensity of the highlight and light direction has been used in the past. Hanrahan and Haeberli [hanr90] mention how they can specify a light direction by dragging a highlight on a sphere. This technique has also been previously implemented in some modelers like a light modeler developed in 1983 at NYIT by Paul Heckbert (manipulating highlights on a sphere) and a light editor written by Richard Chuang around 1985 at PDI, which was used among others, to get highlights to appear at the right time on flying logos. It also came to the attention of the authors that a similar approach to Chuang's was used at LucasFilm to get the glare to appear at the crucial moment on a sword in the movie *Young Sherlock Holmes*.

Our technique extends the basic approach in the above systems by indirectly and interactively determining the surface roughness coefficient $n$ in relation with the size of the highlight. Here is how it works.

[1] All vectors in this paper are assumed normalized

Once the maximum intensity point of the highlight has been chosen, the user drags the cursor away from this point. At a new position on the surface, the surface normal is computed. This new point is used to determine the boundary of the highlight, i.e. where the specular term of (1) reaches a fixed threshold $t$. To satisfy this threshold, $n$, the only unknown, is easily computed as

$$n = \frac{\log t}{\log(\vec{N} \cdot \vec{H})}. \tag{3}$$

While only these two points on a surface are necessary to orient a directional light source and establish the surface roughness coefficient, they give almost no information on the shape of the highlight. To approximate the contour of the highlight, the pixel with the maximum intensity is used as a seed point and the neighboring pixels covered by this surface are visited in a *boundary fill* fashion until pixels on both sides of the threshold are identified or until the boundary of the surface is found. With this technique, the second point might not appear within the contour of the highlight determined from the seed point. If this happens, the second point is also used as a seed. Unfortunately, unless each pixel covered by this surface is visited, some of the other highlights produced by this light on this surface might be missed. If the position of every highlight is necessary, the whole surface is visited by the filling algorithm only on request from the user because such a request can lead to considerable increase in computation time.

When $n$ has already been determined for a given surface, care must be taken in order to keep a unique value for $n$. If another highlight is created on this surface, as soon as the point with the maximum intensity is selected, the contour of this new highlight is computed with the previous value for $n$. However this value for $n$ and the position of the highlights are not fixed and can be interactively changed because some information is kept in a temporary frame buffer. In this frame buffer, each previously visited pixel contains information about its surface normal. The contour can therefore be scaled down (i.e. a smaller highlight but a larger value for $n$) very efficiently. If the contour is increased, only the unvisited pixels need to have their surface normal determined. Moving the contour on the surface is also possible although more expensive if the highlight is moved to a completely different location on the surface as many surface normals might have to be computed. On some graphics hardware like the VGX from SGI, information on the surface normals can be obtained directly from the hardware and therefore allows for even faster highlight manipulation. Figure 1 shows the highlight produced by a directional light source over a patch of the teapot. The white segment within the highlight region represents the point of maximum intensity and points towards the light direction.

Unfortunately, highlight information is dependent on the eye position. Therefore, if the camera is moved, every highlight in the scene must be recomputed. Also, the points of maximum intensity are not valid any more and consequently every surface has to be scanned to recover every highlight, an expensive process that one should try to avoid as much as possible. This means also that a highlight computed in one window would have a different definition in another window with a different projection. To avoid confusion and increasing too much the computing time, we decided to remove every highlight information when the viewing parameters are changed although we keep the light definitions. These highlights are recomputed on request from the user.

Figure 1: Creating a light by its highlight

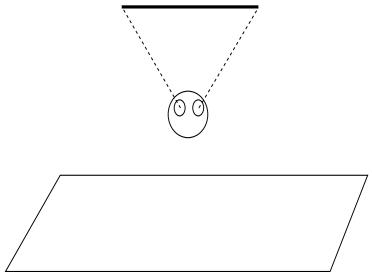

Figure 2: Incomplete highlight information

Another limitation of using highlight information to describe a light source resides in the fact that a highlight specifies only a direction. We therefore need more constraints to use it to determine other types of light source. Such constraints exist for instance for polygonal light sources. Assume a plane on which a polygonal light resides. By adding a highlight, a direction is established. The intersection between this direction and such a plane[2] defines a point light source, a vertex of a linear or polygonal light source.

To represent highlights created by extended light sources, the contribution of each vertex of the light is not sufficient to determine the shape of the complete highlight. To display this information, the boundary fill algorithm would have to compute the specular integral for a linear light [poul91] or a polygonal light [tana91] for each pixel to visit. Such integrals are rather expensive to compute and in order to achieve real time, cheaper approximations based on precomputed tables could be of some use here. We did not investigate this approach in the context of this paper, relying solely on the partial information provided by the light vertices as shown in figure 2.

As it can be observed, highlight information can be very useful to specify directional light sources and surface roughness coefficients. With extra constraints, they can even be used to define point, linear and polygonal light sources although creating an arbitrary plane in 3D is not necessarily an easy task. Another technique, more flexible for extended light sources, consists in using the shadow information to define the light sources.

[2] Note that there might not be any intersection

## 2.2 Lights from Shadows

Shadows are very important clues to help understanding the geometry of the scene and the interrelationship between objects; in the context of this paper, shadows can reveal important information about the nature of the light sources. We will define light sources by manipulating their *shadow volumes*.[3] These shadow volumes have the advantage to depend only on the lights and objects positions. Therefore, as opposed to the case of the highlights, the camera position can be changed without altering their description. Moreover, their definition is consistent for every projection, allowing for multiple windows open with different orthographic and perspective projections as used in most of the modeling systems.

The shadow volume created by an object illuminated by a directional light source consists of a sweep of the object silhouette in the direction the light source shines. This silhouette can be analytically determined for simple primitives, computed for moderately complicated objects with algorithms like in [bonf86], sampled by studying the variation of surface normals at the vertices of a tessellated object or sampled using the information in a z-buffer projection of this object. Specifying the direction of a directional light is simply a question of choosing two arbitrary, although different, points in the scene. The second point will be along the shadow cast by the first one. To move this shadow volume once defined, one needs to select a point on the shadow

[3] A shadow volume formed by a single object and a directional or a point light is the 3D volume within which every point is in shadow of this object [crow77] [berg86]. For extended light sources (linear, polygonal), the shadow volume is the 3D volume within which every point is at least partly in shadow of this object.

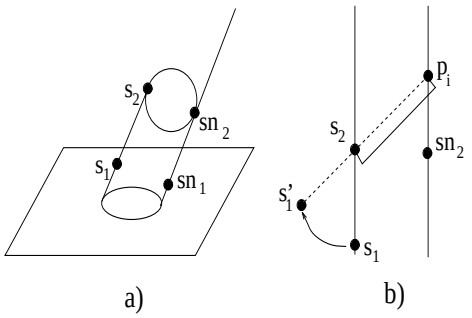


Figure 4: Going from a directional light source to a point light source

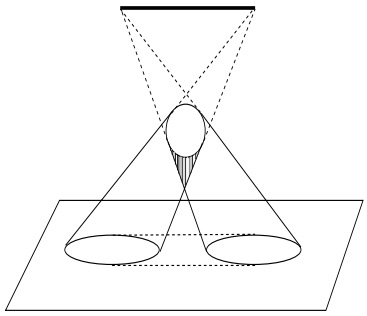

Figure 5: Umbra region in hatched undetected in the projection domain

volume. The point on the object casting this shadow is then identified. By dragging the cursor to a new location, a new direction is computed, the direction of a directional light source. Figure 3 shows a cylinder illuminated by a directional light source. For some primitives, computing the exact silhouette can be expensive and not carry much more information. In the case of this cylinder, each polygon vertex forming the cylinder is simply projected in the direction the light shines.

A directional light source can be viewed as a point light source at infinity. If the point light source is not at infinity, the silhouette defining the shadow volume can be different than the silhouette defined by a directional light source. Figure 4 illustrates the process of going from a directional light source (figure 4a) to a point light source (figure 4b) by modifying its shadow volume.

A point $sn_1$ on the shadow volume is chosen. The point $sn_2$ on the silhouette casting shadow on the point $sn_1$ is identified. This *shadow segment* $sn_1 - sn_2$ will now be considered as nailed and the point light source will reside on the line extending this segment. By selecting another point $s_1$ on the shadow volume, the point $s_2$ casting this shadow on this point is identified. The nailed segment $sn_1 - sn_2$ and the point $s_2$ define a plane $(sn_1 - sn_2 - s_2)$. By moving the cursor, a point $s_1'$ on this plane is located. $s_1'$ now is on the shadow cast by $s_2$. The point light source is therefore moved to $p_i$ as shown in figure 4b.

Once a point light source is created, it can be manipulated in the scene by manipulating its volume shadow. This can be done by fixing any shadow segment as previously explained, or, if no shadow segment is nailed, by adding a new constraint to the system by assuming for instance the distance $d$ from the light $p_i$ to the point $s_2$ casting a shadow is constant. Combinations of these two actions are sufficient to position almost any point light source in a scene.

In some rare configurations of a scene, some positions might not be accessible. For instance, assume a scene is made of a single flat polygon and of a directional light parallel to the plane of the polygon. In such a situation, the light will never be able to escape the plane of the polygon. Fortunately, this situation does not occur often in general 3D scenes, and so far combinations of moving the shadow volumes with and without nailed segments proved to be sufficient to position our lights.

It is important to note that the point $s_2$ might not lie on the boundary of the shadow volume while the point light source is moved around. However the real shadow volume is always displayed so the user has a direct view of the altered shadow.

To create extended light sources like linear or polygonal, new point light sources are needed, defining the vertices of the light source. The shadow volumes of each light vertex are handled as normal point light sources although for polygonal light sources with more than three vertices, care must be taken so each light vertex will reside on the light plane.

Shadows of extended light sources are formed by the umbra and penumbra regions. The whole shadow region is defined by the convolution of the object and the light source in the projection domain [guib83]. The umbra is defined by the intersection of each shadow volume (one shadow volume per light vertex); the penumbra is the difference between the whole shadow and the umbra. Nishita et al. [nish83] studied the various parts of these shadow regions in 2D, once projected onto polygonal surfaces for shadow culling purposes. Some problems occur when neither the object casting the shadow or the light are limited to being convex. It can be shown however that if both the light and the object are divided into convex elements, the whole shadow is the *union* in 3D of all the shadow convex hulls as:

```
For each convex light element
    For each convex object element
        Compute the convex hull of the shadow
        volumes created by these two elements
Compute the 3D union of all these convex hulls
```

For now on, assume a polygonal convex light and a convex object.

Assume an object does not intersect the light plane. All the shadows lie on a plane parallel to the light plane but located at infinity. As such, 2D convex hull algorithms can be used to determine which part of the shadow volumes form the 3D convex hull of the shadow volumes.

However computing the umbra region, i.e. the intersection of the convex hulls for each light vertex cannot be done in 2D. Figure 5 shows an exemple where using only the information in the 2D projection plane would fail to identify the umbra region showed in hatched.

To recover the umbra region, one could intersect each shadow polygon[4] of a light vertex shadow volume with each other shadow volume of the other vertices of a single light. This process can be very expensive as it is $O((ps)^2)$ where $p$ is the number of vertices of the light and $s$ is the number

[4]The silhouette of the object can be discretized. Each point cast its shadow in one direction. Two consecutive points on this silhouette and their shadow direction define a quadrilateral with two of its vertices at infinity.

Figure 3: Creating a directional light by its shadow

of shadow polygons forming the shadow volume. However some improvements can be obtained by first projecting the shadow quadrilateral onto the plane containing the convex hull on the 2D projection plane.

Since we use Graham's 2D convex hull algorithm [sedg90], the points of the shadow quadrilateral, once projected in 2D, are converted in pseudo angles and an efficient combination of angle comparisons and boxing allows for faster intersection culling.

This process could also be improved by using a different data structure that might be more suitable for faster intersections of half planes defined by the shadow quadrilaterals. In object space, the binary space subdivision algorithm handling shadow volumes as presented by Chin and Feiner [chin89] would be a good candidate to investigate, while in screen space the algorithm described by Fournier and Fussell [four88] could be of use.

## 3 Results

A very simple modeler has been implemented in order to test the techniques presented in this paper. The modeler includes primitives like conics (sphere, disk, cone, cylinder), squares, cubes, triangular meshes and Bézier patches. Figure 6 shows a global view of the modeler itself.

The code, far from being optimized, is written under GL and was developed and tested on an Iris 4D/20 with z-buffer. This machine handles well a few primitives ($\approx 10$) but as the scene complexity increases, a 4D/240 VGX becomes very handy. The VGX also allows for real time Phong shading which is very useful to model a scene and when creating/manipulating shadows, but it can lead to some minor difficulties when creating highlights, because the threshold $t$ must be adjusted to the SGI's Phong's shading implementation.

Figures 7 to 9 show a cone under a triangular light source. At first, no convex hull is applied. In this image (figure 7), it is easier to associate each shadow with a light vertex. Once the convex hull is applied (figure 8), the silhouette of the penumbra is easier to detect. Notice the umbra region just under the cone, within the penumbra region. In figure 9, the umbra and penumbra volumes are filled with a semi-transparent mask. This representation gives a more complete impression of the shadows that can not really be shown here with a single image.

## 4 Conclusion

In this paper, we investigated using lighting effects, i.e. highlights and shadows, to define the lights themselves and specify their location. We showed some inherent limitations with these approaches but also demonstrated a powerful new technique. This technique allows a user to interactively manipulates highlights and shadows, which can be very important when designing a scene. In previous modeling systems, these effects were too often neglected. Therefore a user needed to iterate between rendering the whole scene and modifying the lights. It is a process that can be expensive depending of the quality of the rendering required. Incorporating highlights and shadows in the modeling process adds more information on the geometry of the scene and its illumination which should help the user to understand better the scene before even rendering it.

Our system, although simple, gives during the modeling process direct information to the user on the lighting effects since these effects are the objects being manipulated. This direct manipulation is crucial as getting the right effect by manipulating the causes is generally more difficult than manipulating the effects themselves.

We foresee that, as the graphics hardware improves and as the CPU becomes faster, more and more effects available once only at the rendering stage will become an inherent part of the modeling stage itself. Real time Phong shading is now becoming common with high-end modelers. These improvements will lead us to investigate more intuitive ways of defining and controlling these special effects. Although the separation between computer graphics and computer vision is still strong, we believe this will lead us to more and more

Figure 6: Global view of the modeler

Figure 7: Cone under a triangular light: No convex hull

Figure 8: Cone under a triangular light: Convex hull applied

Figure 9: Cone under a triangular light: Convex hull with filled shadows

graphics in vision and more and more vision in graphics for greater benefits to realism in graphics and scene analysis of natural phenomena in vision.


### Acknowledgements

Thanks to Mikio Shinya for interesting "brainstorming" sessions while the first author spent a summer at NTT Japan. Thanks also go to Paul Heckbert for informing us on related but unpublished work and for commenting on previous drafts of this paper. We acknowledge financial support from NSERC, UGF and the University of British Columbia.


## References


[babu85] Mohan D.R. Babu, Chia-Hoang Lee, and Azriel Rosenfeld. "Determining Plane Orientation from Specular Reflectance". *Pattern Recognition*, Vol. 18, No. 1, pp. 53–62, January 1985.

[berg86] P. Bergeron. "A General Version of Crow's Shadow Volumes". *IEEE Computer Graphics and Applications*, Vol. 6, No. 9, pp. 17–28, September 1986.

[blin77] James F. Blinn. "Models of Light Reflection For Computer Synthesized Pictures". *Computer Graphics (SIGGRAPH '77 Proceedings)*, Vol. 11, No. 2, pp. 192–198, July 1977.

[bonf86] L. Bonfigliolo. "An Algorithm for Silhouette of Curved Surfaces based on Graphical Relations". *Computer-Aided Design*, Vol. 18, No. 2, pp. 95–101, March 1986.

[buch87] Craig Stuart Buchanan. "Determining Surface Orientation from Specular Highlights". M.Sc. Thesis, Department of Computer Science, University of Toronto, August 1987.

[chin89] Norman Chin and Steven Feiner. "Near Real-Time Shadow Generation Using BSP Trees". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 99–106, July 1989.

[crow77] Franklin C. Crow. "Shadow Algorithms for Computer Graphics". *Computer Graphics (SIGGRAPH '77 Proceedings)*, Vol. 11, No. 2, pp. 242–248, July 1977.

[four88] Alain Fournier and Donald Fussell. "On the power of the frame buffer". *ACM Transactions on Graphics*, Vol. 7, No. 2, pp. 103–128, April 1988.

[gers87] Ron Gershon. *The use of color in computational vision.* Ph.D. thesis, Dept. of Computer Science, University of Toronto, 1987.

[guib83] Leo Guibas, Lyle Ramshaw, and Jorge Stolfi. "A kinetic framework for computational geometry". *Proceedings of the 24th Annual IEEE Symposium on the Foundations of Computer Science*, pp. 100–111, 1983.

[hanr90] Pat Hanrahan and Paul Haeberli. "Direct WYSIWYG Painting and Texturing on 3D Shapes". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 215–223, August 1990.

[horn88] Berthold K. P. Horn and M.J. Brooks, editors. *Shape from Shading.* MIT Press, 1988.

[nish83] Tomoyuki Nishita and Eihachiro Nakamae. "Half-Tone Representation of 3-D Objects Illuminated by Area or Polyhedron Sources". *Proc. of IEEE Computer Society's Seventh International Computer Software and Applications Conference (COMPSAC83)*, pp. 237–242, November 1983.

[phon75] Bui-T. Phong. "Illumination for Computer Generated Pictures". *Communications of the ACM*, Vol. 18, No. 6, pp. 311–317, June 1975.

[poul91] Pierre Poulin and John Amanatides. "Shading and shadowing with linear light sources". *Computers and Graphics*, Vol. 15, No. 2, pp. 259–265, 1991.

[sedg90] Robert Sedgewick. *Algorithms in C.* Addison-Wesley, 1990.

[shaf85] Steven A. Shafer. *Shadows and Silhouettes in Computer Vision.* Kluwer Academic Publishers, 1985.

[tana91] Toshimitsu Tanaka and Tokiichiro Takahashi. "Shading with Area Light Sources". *Eurographics '91*, pp. 235–246, September 1991.

[walt75] David Waltz. "Understanding Line Drawings of Scenes with Shadows". *The Psychology of Computer Vision*, pp. 19–91. Mc-Graw Hill, New York, 1975.

[wolf91] Lawrence B. Wolff. *Polarization Methods in Computer Vision.* Ph.D. thesis, Columbia University, 1991.