Lighting Controls for Computer Cinematography

Ronen Barzel
Pixar Animation Studios

PUBLISHED IN: *Journal of Graphics Tools* 2(1), 1997, PP.1–20
http://www.acm.org/jgt/papers/Barzel97

Abstract

Lighting is an essential component of visually rich cinematographic images. However, the common computer graphics light source models, such as a cone-shaped spotlight, are not versatile enough for cinematographic-quality lighting. In this paper we describe the controls and features of a light source model for lighting computer graphics films. The model is based on generalized light cones, emphasizing independent control over the shape and texture of lights and shadows. While inspired by techniques of real-world cinematography, it is tailored to the needs and capabilities of computer graphics. The model has been used successfully in production over the past few years, to light many short works and the movie *Toy Story.*

1 Introduction

Photorealism is an important and much-studied goal in computer graphics imagery and illumination. But photographers and cinematographers know that when it comes to lighting, realism is not the only goal.

The purposes of lighting for cinematography are to contribute to the storytelling, mood, and image composition, and to direct the viewer's eye. The "practical" light sources on a real-world movie set, such as desk or ceiling lamps, are rarely major contributors to the illumination. Instead, various types of lamps and spotlights are placed off-camera, in order to create the desired illumination effect. A lighting designer will use whatever techniques, tricks, and cheats are necessary, such as: suspending a cloth in front of a light to soften shadows; positioning opaque cards or graded filters to shape a light; focusing a narrow "tickler" light to get an extra highlight; or hiding a light under a desk to fill in dark areas under a character's chin.

This paper presents a lighting model that has developed over several years in response to the needs of computer graphics film production, in particular for making *Toy*

Story. The model gives the CG lighting designer control over the shape, placement, and texture of lights, so that the designer's real-world cinematographic talent can be applied to computer images.

The emphasis of the model is not on realism nor on physically simulating the tools of real-world cinematography. Rather, we take advantage of various sorts of unreality available to us in the CG world in order to get the desired cinematographic effects.[1] Thus, while real-world effects provide motivation, our lighting model is ultimately based on effects that are useful in computer graphics.

Note that this paper discusses the technology of cinematography, not the artistry; the art of cinematography is beyond the scope of this paper (and this author). We refer interested readers to cinematography texts such as [Lowell92] or [Malkiewicz86]. Of particular relevance, [Calahan96] discusses the art of cinematography as applied to computer graphics.

Related Work

[Warn83] developed the cone spotlight model that has since become standard in computer graphics (as well as a simple barn door light), "based in part on observation. . . of the studio of a professional photographer," in order to be able to create better-lit images than were available with then-standard tools. We continue in this vein, extending the model to meet increasingly ambitious cinematic lighting needs. Doubtless others have met similar needs, but we are not aware of other published work in this area.

End-user graphics systems typically have fairly powerful lighting tools. For example, [Alias95] supports assorted light types, such as spot, point, and volume-restricted. Our model is more flexible in that any parameter or effect can be used with any shape, and we place greater emphasis on light textures and manipulation of shadows.

Our model does not include finite-area light sources (see, e.g., [Verbeck, Greenberg84] or [Cook,Porter,Carpenter84]), and it would likely benefit from them; however our light-shaping methods can to a large extent fake their soft lighting effects.

Global illumination methods can yield subtly and realistically lit images, but they typically support limited lighting controls and emphasize light from practical sources. However, the cinematographic emphasis on off-camera lighting and other trickery is not incompatible with global illumination methods. For example, [Dorsey,Sillion, Greenberg91] models opera lighting using off-camera lights and projected images. [Gershbein,Schröder,Hanrahan94], and [Arvo95] also address textured light sources. Since our model is implemented via RenderMan shaders [Upstill90], it can in principle work with a RenderMan-compliant global illumination renderer such as BMRT [Gritz, Hahn96], although we have not pursued this. Pragmatically, however, global illumination is still too computationally expensive for regular use in computer graphics film production.

[1] In fact, real-world cinematographers would doubtless use CG tricks if they could, e.g., have light emanate out of nowhere, cut off a light after a certain distance, or change the direction of shadows.

The task of a lighting designer can be aided by tools that speed up the process of choosing and varying lighting parameters. For example, [Bass81] and [Dorsey, Arvo,Greenberg95] describe techniques to quickly recompute and redisplay images as lighting parameters change; [Schoeneman *et al.*93] and [Kawai,Painter,Cohen93] determine lighting parameters given user-specified objectives. Tools such as these could be used in conjunction with our lighting model.

Overview

Sec. 2 describes the controls and features of the lighting model, and Sec. 3 sketches its implementation. Sec. 4 presents and discusses several sample *Toy Story* images.

We will not discuss the user interface for interactively placing and manipulating the light sources; it is straightforward but beyond the scope of this paper.

2 The Lighting Model

The lighting model provides control over several aspects of each light source: *selection, shape, shadowing, texture, dropoff, direction,* and *properties.* We describe and illustrate these capabilities below. For clarity, each feature is illustrated in isolation, although the model allows them to be used in any combination. Also, we illustrate with static images, but all parameters can of course be animated.

Fig. 1 shows a sample scene lit only by fill lights; Fig. 2 shows the same scene with a simple conical key light. (We use the cinematography terms *key light* and *fill light* for major and minor sources of illumination, respectively.) Fog is introduced to illustrate the effect of the light throughout space.[2]

2.1 Selection

Computer graphics lights can be enabled or disabled on a per-object basis (Fig. 3). The ability to selectively illuminate objects in a scene is a powerful feature, which has no analog in real-world lighting. In our experience, per-object selection is used frequently, in particular to adjust illumination separately for the characters and the set.

2.2 Shape

The basic task of lighting is the placement and shape of light in a scene. Real-world cinematography commonly uses spotlights and barn door (rectangular) lights to achieve desired shapes; our model provides a generalization of these.

[2]The fog is calculated by computing the incident light on hypothetical fog particles along each viewing ray, and accumulating the total light scattered back to the viewer. The fog has no effect on the light incident on the objects.

Generalized cone/pyramid. The light affects a region whose cross-section is a superellipse, continuously variable from purely round, through rounded-rectangle, to pure rectangle (Figs. 4–6). The slope of the pyramid can be varied until at the limit the sides are parallel. The pyramid may be truncated, as if it were originating from a flat lamp face, and may be sheared, for window and doorway effects (Fig. 6).

Soft edges. To soften the edge of the light, we define a boundary zone in which the light intensity drops off gradually. The width and height of the boundary zone can be adjusted separately. Thus we have two nested pyramids: the light is at full intensity inside the inner pyramid, and has no effect—i.e., 0 intensity—outside the outer pyramid, with a smooth falloff between them (see Figs. 7,8 and appendix A).

Cuton and cutoff. The light shape can further be modified by specifying near and far truncation, again with adjustable-width smooth dropoff zones (Figs. 9,10). These have no real-world physical analog, but are very useful to soften a lighting setup, and to keep the light from spilling onto undesired parts of the scene.

Being able to adjust the shape and edge dropoff of lights easily allows for soft lighting of a scene, faking area-light penumbra effects.[3] In our experience, the majority of lights are chosen to be mostly rectangular, with large edge zones to provide smooth gradation of lighting; conical lights are used mostly if the scene includes a practical source, such as a flashlight.

The light shape can of course be rigidly rotated, scaled, and placed anywhere in space. This is a strength of CG lighting over real-world lights: we are not encumbered by physical structures and mechanisms that must be hidden or placed off-camera; CG lights can emanate spontaneously anywhere in space, giving the lighting designer great freedom.

Finally, another choice for the shape is to have no shape at all: the light affects all of space.

2.3 Shadowing

Shadows and shadow placement are an important part of cinematography. Computer graphics has great freedom to control shadows for artistic effect. In the following discussion, it is convenient to think of shadow projection as defining a "volume of darkness" inside of which illumination is inhibited; this volume can be manipulated as an independent entity.

Shadow selection. A light doesn't necessarily have to cast shadows.[4] In Fig. 11, the key light casts shadows, but the background fill lights do not. Shadows may also be disabled on a per-object basis, as in Fig. 12. Thus a difficult problem in real-world cinematography, suppressing unwanted shadows, is trivial in computer graphics.

[3]The lack of actual area lights in the model can manifest itself, however, in the shapes of shadows and surface highlights.

[4]For most rendering algorithms it is of course cheaper and easier *not* to cast shadows.

Shadow direction. The direction that shadows are cast doesn't necessarily have to follow the light—each "volume of darkness" can be aimed as needed. For example, the light in Fig. 13 is the same as in Fig. 11, but the shadows have been shifted so that the torus casts a shadow on the cylinder. It is perhaps surprising just how far shadow directions can be shifted from true without incurring visual dissonance. "Cheating" the shadow directions can be a powerful tool for controlling image composition; in our experience, background shadows are often "cheated."

Shadow sharing. A seemingly bizarre capability is for a light to share its shadows with other lights (Fig. 14). That is, a "volume of darkness" defined by a given light and object can inhibit illumination from other lights as well. This allows the lighting designer to strengthen shadows that might otherwise be washed out by nearby lights. In our experience shadows are often shared.

Fake shadows. It is often useful to create extra shadows, to imply nonexistent offscreen objects or simply to darken a scene where needed. In real-world lighting, opaque cards can be placed in front of a light. In our model, *blockers* can similarly be defined; each is specified by a 2D superellipse that can be placed anywhere in space (Fig. 15). As with ordinary shadows, the direction that the blocker casts its shadows can be adjusted, and a blocker can be shared among several lights. In our experience, blockers are heavily used, sometimes several per light.

Shape trimming. A blocker can be made large and placed so as to trim the shape of a light, as in Fig. 16. Animating a large blocker can be an easy way to fake a door-opening-offscreen effect.

Shadow softening. To keep shadows from being too harsh, any shadow or blocker can be made translucent, to only partially inhibit illumination. Shadow edges can be softened as well: for blockers, this is done via an edge-zone dropoff in the same manner as the light shape; for object shadows, the boundary of the "volume of darkness" is blurred. Finally, rather than going to black, a shadow can be assigned a color; subtle use of colored shadows can add richness to an image.

2.4 Texture

Just as images are used in computer graphics to create texture on surfaces, they can be used to create texture in lights via projection.

Cookie. A single-channel matte image can be used as a "cookie cutter,"[5] to get cross-sectional shapes other than the built-in superellipses (Fig. 17), or, more subtly, to fake complex shadows from offscreen objects (Fig. 18).

Slide. A full-color image yields a slide-projector effect (Fig. 20). An unfocused projection (such as from a television set) can be achieved by applying a blur filter whose width increases as the projection distance increases.

[5] "Cookie" is colloquial for "cucaloris," the technical term for an opaque card with cutouts, used to block a light.

Noise. In addition to stored image files, the light can be projected through a 2D noise function that modifies the intensity or color, yielding "dirty" lights.

As with shadows and blockers, it is possible to "cheat" the origin and direction of an image projection, to blur it, and to adjust its intensity.

2.5 Dropoff

The intensity of the light can vary, in the familiar manner of computer graphics lighting:

Beam distribution. Fig. 21 illustrates dropoff of intensity across the beam using the usual exponentiated cosine function; however, the angle is normalized so that the intensity attenuates to 0 at the edge of the beam.

Distance falloff. Fig. 22 illustrates dropoff of intensity with distance from the light source location, using the usual inverse-power attenuation; to keep the intensity well-defined at the light origin we provide a smooth clamp to a maximum intensity (see appendix B).

In our experience, choosing exponential dropoff parameters is not visually intuitive; it is often easier to have no *a priori* dropoff, and to gradate lighting by using soft shape, cutoff, and blocker edges.

2.6 Direction

The light ray direction has the two options common in computer graphics (Fig. 23):

Radial. The rays emanate from a point at the apex of the pyramid.

Parallel. The rays are parallel to the centerline of the light pyramid, as per a very distant light.

Fig. 23 illustrates parallel and radial rays in a light pyramid. The combination of parallel rays with widening shape yields a non-physical, but still useful effect: light rays are created along the edges of the shape. One can also imagine circumstances in which it would be useful to "cheat" the ray direction in other ways, e.g. to have rays parallel to a direction other than the pyramid centerline, or to define an arbitrary curvilinear vector field for the ray direction, but we have not needed such cheats in our standard model.

Note that by choosing a single well-defined light direction, we are implicitly assuming a point source or infinitely-distant source. To support finite-area lights, a mechanism such as ray distribution ([Cook,Porter,Carpenter84], [Veach,Guibas95]) would be need to be introduced to the model.

2.7 Properties

The previous sections have discussed where and in what direction the light reaches the surfaces in the scene. Finally, we have the properties of the "photons" that are incident

on the surfaces:

Intensity. The nominal intensity of the light is specified at its origin or at a target point within the shape. This value is attenuated by the shape boundary, shadows, cookies, noise, and dropoff.

Color. The color of the light is expressed as a standard 3-channel RGB value, which can be filtered by a slide, noise, or colored shadow.

Effect. The three standard CG illumination effects are available: ambient flat lighting, diffuse shape lighting, and specular highlighting. They may be used in combination, with a scale factor for each. (For ambient lighting, the ray direction is of course irrelevant.) In practice, diffuse-only lights are usually used for soft fills (Fig. 1), while key lights use both diffuse and specular effects. A small ambient component to a light is useful to keep regions from going completely black.

Other information. Depending on what can be supported by the surface-shading model, additional information can be carried to surfaces. For example: a fourth channel of color can contain "ultraviolet" that a "fluorescent" surface will react to by self-illuminating; or we may have an identifier for each light source so that surfaces can react differently to specific lights.

3 Implementation

The lighting model is implemented as a RenderMan light source shader [Upstill90]. The programmability of shaders has been invaluable in developing and extending the model.

Following the RenderMan paradigm, a light source is considered to be a functional unit: given a point on a surface anywhere in space, a light source computes the direction and color of the "photons" incident on that point due to that source. Fig. 24 gives an overview of the computation. The actual RenderMan code is somewhat messier, however, in particular because RenderMan does not support arrays; there are a maximum number of shadowmaps, blockers, and so forth, and each **foreach** in Fig. 24 is actually a series of tests against each parameter.

For efficiency, if any features aren't used, we skip the corresponding steps of the computation. Also, if at any step the attenuation factor becomes 0, the remainder of the steps can be skipped. Finally, we use a mechanism that generates special-purpose shaders based on the chosen parameters, analogous to but simpler than the method of [Guenter,Knoblock,Ruf95].

Fig. 24 uses shadowmaps for shadow generation ([Williams78], [Reeves,Salesin, Cook87]), as this is the mechanism supported by our renderer. With shadowmaps, the shadow trickery of Sec. 2.3 is straightforward: objects can be selectively included in shadowmaps; shadow directions can be cheated by adjusting the shadow camera before computing the map; and shadows can be shared by referencing the same shadowmap file in several lights. For other shadowing algorithms, these tricks may need to be

coded into the renderer.

4 Results

Plates 1 through 6 show several *Toy Story* frames. We discuss some illustrative features of the lighting in each.

Plate 1 is a simple scene that illustrates the significance of light shape and placement for image composition: the character is framed by a rectangular profile of a barn door light. The light nominally shines through a window offscreen to the right, but the placement was chosen for visual effect, not necessarily consistent with the relative geometries of the window, the sun, and the bed. Notice also the soft (partial-opacity) shadows of the bed on the wall.

Plate 2 has stripes of light on the characters, that are not shadows of the venetian blind slats, but are generated using a cookie that is adjusted for dramatic effect; The slats are lit from below by a separate light, in order to obtain the desired highlighting.

Plate 3 also uses a cookie (the same one used in Figs. 18,19) to generate a dappled leaf effect. A separate cone-shaped light, with its own cookie, spotlights the area around the soldier crouching in the rear. The characters in the foreground are lit with an additional, blue light acting only on them for the blue highlights. This scene also includes a practical light, the red LED, which is used for effect but not as a key source of illumination.

Plate 4 features a practical light source, the flashlight, also using a cookie for the lens ring effect. The flashlight actually includes two light sources, one that illuminates the character and one used only for a fog calculation; the latter has a cutoff so that the fog doesn't obscure the character. This scene also features an "ultraviolet" light (as described in Sec. 2.7), responsible for the blue glow of the white shirt.

Plate 5 illustrates the importance of shadow placement—the shadows of the milk crate were carefully adjusted so that an "X" would fall on the character's face without obscuring the eyes or mouth, since the character is speaking. Blockers were also used to darken the background, both to darken the mood and to accentuate the character's face in the foreground.

Plate 6 contains a variety of techniques: the key light has soft edges; the character's shadow direction is cheated for compositional effect; a light at a grazing angle from the left illuminates only the desktop, to accentuate its texture; separate lights illuminate only the character to provide extra highlighting; and a cookie is animated from one frame to the next, to provide a falling-rain effect.

5 Conclusion

The lighting model we describe provides a convenient and powerful encapsulation of control for cinematography. It has been developed in response to needs and requests of

lighting designers doing computer graphics film production: the features that we have described are those that have proven useful in practice, and almost all lighting effects used in *Toy Story* were achieved with the available features.

Our lighting model is a straightforward collection and extension of common computer graphics methods. It (or a variation) is perhaps a candidate for inclusion in standard graphics libraries. We would be particularly interested in seeing support for controllable light shape and texture in rendering hardware, in order to be able to interactively create cinematographically-lit images.

An intriguing question is to what extent lighting models such as ours—practical but non-physical—can be incorporated into physically-based global-illumination rendering schemes. The combination of controlled lighting and sophisticated rendering has potential for creating images of a quality as yet unseen.

Acknowledgements

This lighting model is an extension of earlier work by Yael Milo and others. The blockers are are based on work of Larry Aupperle and Oren Jacob. The fog effect is based on work of Mitch Prater. This model would not have been developed without feedback and use by the *Toy Story* lighting team. Kurt Fleischer, Sharon Calahan, and Uriel Barzel provided valuable suggestions for this paper.

References

[Alias95] Alias | Wavefront, a division of Silicon Graphics Canada Limited, Toronto, 1995.

[Arvo95] James Arvo. Applications of irradiance tensors to the simulation of non-Lambertian phenomena. In *Computer Graphics* proceedings, Annual Conference Series, ACM SIGGRAPH, 1995, pp. 335–342.

[Bass81] Daniel H. Bass. Using the video lookup table for reflectivity calculations: specific techniques and graphics results. *Computer Graphics and Image Processing*, Vol. 17, 1981, pp. 249–261.

[Calahan96] Sharon Calahan. Storytelling through lighting: a computer graphics perspective. SIGGRAPH course notes, 1996.

[Cook,Porter,Carpenter84] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(3) (Proc. SIGGRAPH), July 1984, pp. 137–145.

[Dorsey,Arvo,Greenberg95] Julie Dorsey, James Arvo, and Donald Greenberg. Interactive design of complex time-dependent images. *IEEE Computer Graphics and Applications*, 15(2), March 1995, pp. 26–36.

[Dorsey,Sillion,Greenberg91] Julie O'B. Dorsey, François X. Sillion, and Donald P. Greenberg. Design and simulation of opera lighting and projection effects. *Computer Graphics* 25(4) (Proc. SIGGRAPH), July 1991, pp. 41–50.

[Gershbein,Schröder,Hanrahan94] Reid Gershbein, Peter Schröder, and Pat Hanrahan. Textures and radiosity: controlling emission and reflection with texture maps. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 1994, pp. 51–58.

[Gritz,Hahn96] Larry Gritz and James K. Hahn. A global illumination of the RenderMan standard. Submitted to *Journal of Graphics Tools*, 1996.

[Guenter,Knoblock,Ruf95] Brian Guenter, Todd B. Knoblock, and Erik Ruf. Specializing shaders. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 1995, pp. 343–350.

[Kawai,Painter,Cohen93] John K. Kawai, James S. Painter, and Michael F. Cohen. Radioptimization—goal based rendering. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 1993, pp. 147–154.

[Lowell92] Ross Lowell. *Matters of Light & Depth.* Broad Street Books, 1992.

[Malkiewicz86] Kris Malkiewicz. *Film Lighting.* Prentice Hall Press, 1992.

[Reeves,Salesin,Cook87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics* 21(4) (proc. SIGGRAPH), July 1987, pp. 283–291.

[Schoeneman *et al.*93] Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo, and Donald Greenberg. Painting with light. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 1993, pp. 143–146.

[Upstill90] Steve Upstill. *The RenderMan Companion.* Addison-Wesley, Reading, MA, 1990.

[Veach,Guibas95] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for Monte Carlo rendering. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 1995, pp. 419–428.

[Verbeck,Greenberg84] Channing P. Verbeck and Donald P. Greenberg. A comprehensive light-source description for computer graphics. *IEEE Computer Graphics and Applications* 4(7), July 1984, pp. 66–75.

[Warn83] David R. Warn. Lighting controls for synthetic images. *Computer Graphics* 17(3) (Proc. SIGGRAPH), July 1983, pp. 13–21.

[Williams78] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics* 12(3) (Proc. SIGGRAPH), August 1978, pp. 270–274.

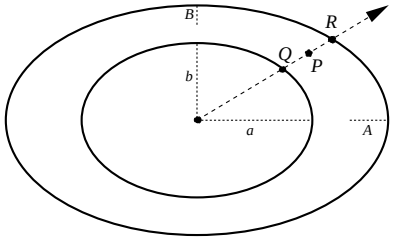
Appendices

A Superellipses

A superellipse is a figure that varies between an ellipse and (in the limit) a rectangle, given by:

$$(\frac{x}{a})^{\frac{2}{d}} + (\frac{y}{b})^{\frac{2}{d}} = 1$$

where a and b are the x and y radii, and d is a "roundness" parameter varying the shape from pure ellipse when $d = 1$ to a pure rectangle as $d \to 0$ (Fig. 4).



We want to soft-clip a point P to a shape specified by two nested superellipses, having radii a, b and A, B. That is, given P, compute a clip factor of 1 if it is within the inner superellipse and 0 if it is outside the outer superellipse, varying smoothly value in between. We assume that the 3-space point has been projected into the first quadrant of the canonical plane of the ellipse.

We express the ray through P as $\mathcal{P}(s) = sP$, and intersect it with the inner superellipse at Q, and with the outer at R. To find the points P, Q, and R, we express them as:

$$P = \mathcal{P}(p), \quad Q = \mathcal{P}(q), \quad R = \mathcal{P}(r).$$

Trivially, $p \equiv 1$. To compute q, we derive:

$$\begin{aligned}
(\tfrac{qP_x}{a})^{\frac{2}{d}} + (\tfrac{qP_y}{b})^{\frac{2}{d}} &= 1 \\
q^{\frac{2}{d}}((\tfrac{P_x}{a})^{(\frac{2}{d}} + \tfrac{P_y}{b})^{\frac{2}{d}}) &= 1 \\
q^{\frac{2}{d}} &= ((\tfrac{P_x}{a})^{(\frac{2}{d}} + \tfrac{P_y}{b})^{\frac{2}{d}})^{-1} \\
q &= ((\tfrac{P_x}{a})^{\frac{2}{d}} + (\tfrac{P_y}{b})^{\frac{2}{d}})^{-\frac{d}{2}} \\
&= ab((bP_x)^{\frac{2}{d}} + (aP_y)^{\frac{2}{d}})^{-\frac{d}{2}}
\end{aligned}$$

and similarly $r = AB((BP_x)^{\frac{2}{d}} + (AP_y)^{\frac{2}{d}})^{-\frac{d}{2}}$. The final clip factor is given by $1 - \texttt{smoothstep}(q, r, p)$, where RenderMan's `smoothstep` [Upstill90] computes:

$$\texttt{smoothstep}(q, r, p) = \begin{cases} 0, & p < q \\ \textit{Hermite} \\ \textit{interpolation,} & q \le p \le r \\ 1, & p > r \end{cases}$$

For a pure rectangle, $d = 0$, we simply compose x and y clipping to compute a clip factor:

$$(1 - \texttt{smoothstep}(a, A, P_x)) * (1 - \texttt{smoothstep}(b, B, P_y))$$

This gives a different falloff at the corners than the limit of the round calculation, but suits our purposes.

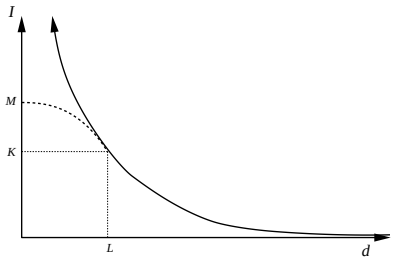
Other falloff functions than the Hermitian `smoothstep` could be useful in some circumstances, but we have not experimented with any. It could also be useful to support asymmetric edge widths; but we have not as yet done so.

B Intensity falloff curve

The common inverse power formula for light intensity can be expressed as

$$I(d) = K \left(\frac{L}{d}\right)^{\alpha}$$

where d is distance from the light source, α is attenuation exponent, and K is the desired intensity at a canonical distance L. This expression grows without bound as d decreases to 0 (solid line):



A common solution is to clamp the intensity to a maximum value; however this yields a curve with discontinuous derivative, potentially causing Mach banding. Instead, we use a Gaussian-like curve (dashed line) when inside the canonical distance:

$$I(d) = \begin{cases} Me^{s(\frac{d}{L})^{\beta}}, & d < L \\ K(\frac{L}{d})^{\alpha}, & d > L \end{cases}$$

where $s \equiv \ln(\frac{K}{M})$ and $\beta \equiv -\frac{\alpha}{s}$ are chosen so that the two curves have matching value $I(d) = K$ and slope $I'(d) = -\frac{K\alpha}{L}$ at distance $d = L$.

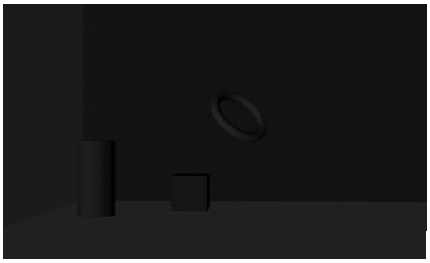


Figure 1: Cylinder and cube on the floor, torus in midair.

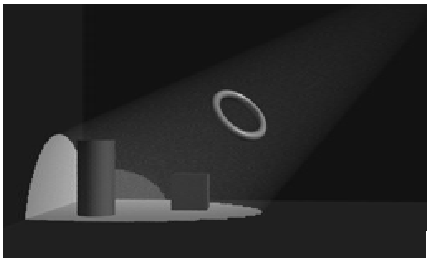


Figure 2: Same as Fig. 1, with a conical key light.

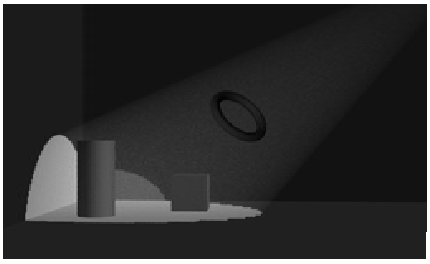


Figure 3: Selection. Same as Fig. 2, but the torus is unaffected by the key light.

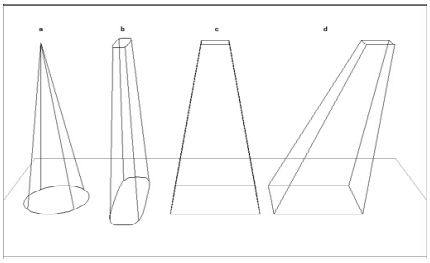


Figure 4: Shape. A superellipse profile is swept into a pyramid, which may be truncated or sheared.

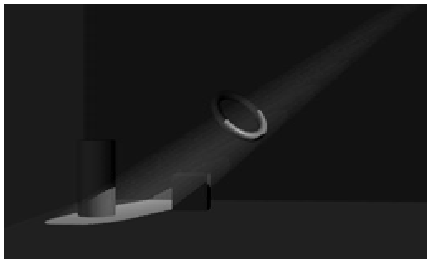


Figure 5: Rounded rectangle shape, as in Fig. 4b.

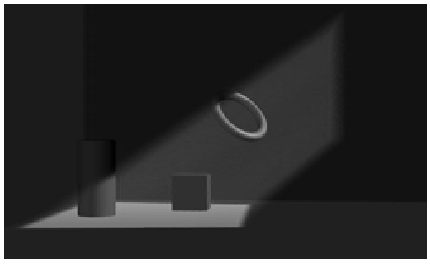


Figure 6: A sheared barn door light, as in Fig. 4d.

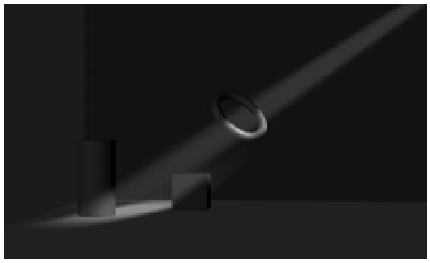


Figure 7: Same as Fig. 5, but with soft edges (Fig. 8).

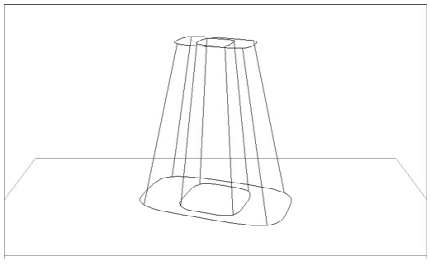


Figure 8: Nested pyramids define the soft edges in Fig. 7.

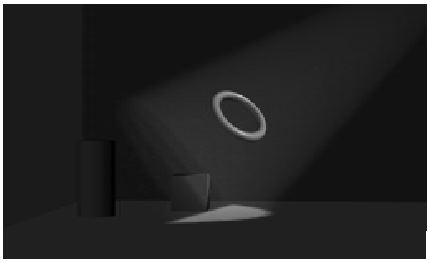


Figure 9: same as Fig. 2, but with a sharp cutoff.

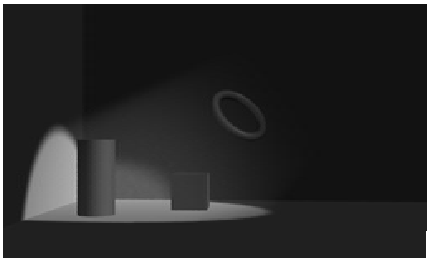


Figure 10: Same as Fig. 2, but with a gradual cuton.

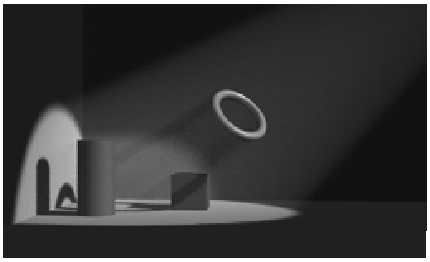


Figure 11: Basic shadows. Same as Fig. 2, but with shadows from the key light.

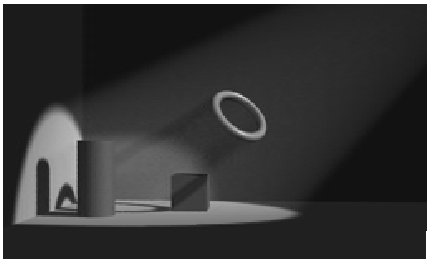


Figure 12: Shadow selection. Same as Fig. 11, but the cube casts no shadow.

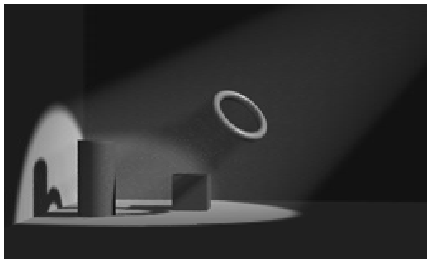


Figure 13: Cheated shadows. All light parameters are the same as in Fig. 11, but the shadow directions have been cheated so that the torus slightly shadows the cylinder.

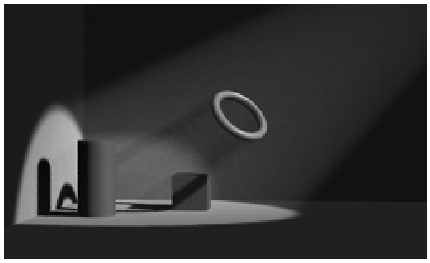


Figure 14: Shared shadows. Same as Fig. 11, but the key light shares its shadows with the fill lights.

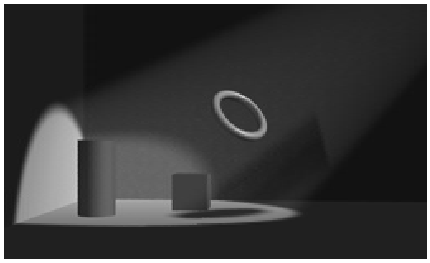


Figure 15: Faked shadow. Same as Fig. 2, but with a blocker that casts a shadow.

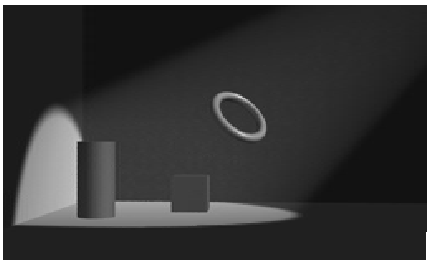


Figure 16: Shape trimming. Same as Fig. 2, but a large blocker has been placed just in front of the rear wall, to eliminate unwanted illumination of the wall.

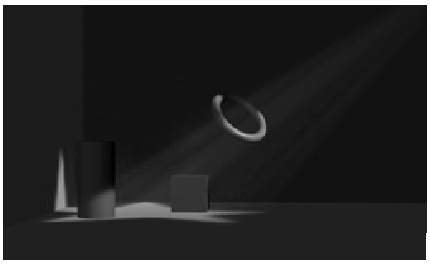


Figure 17: Projecting a matte image as a "cookie cutter" to get alternate light shapes (Fig. 19).

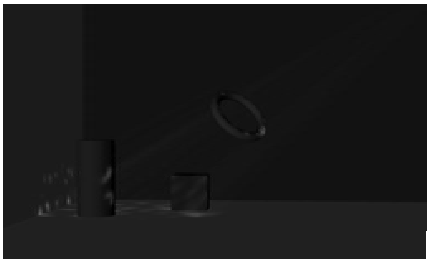


Figure 18: Projecting a matte image to get simulated shadows, here for a dappled-leaf effect (Fig. 19).

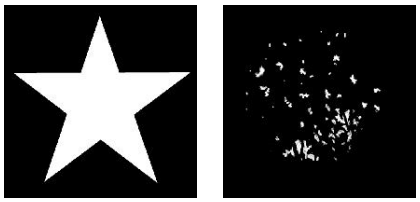


Figure 19: The matte images used in Fig. 17 and Fig. 18.

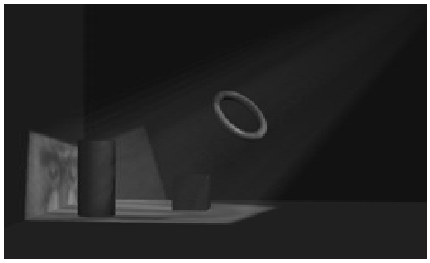


Figure 20: Projecting a color image (the well-known mandrill) to get a slide effect.

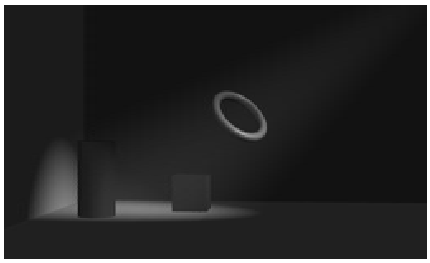


Figure 21: Intensity distribution across beam.

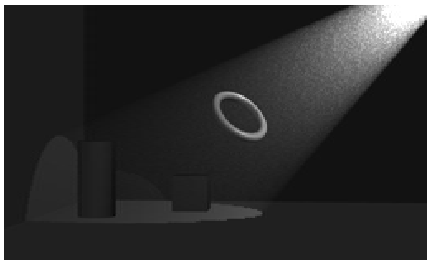


Figure 22: Intensity falloff with distance.

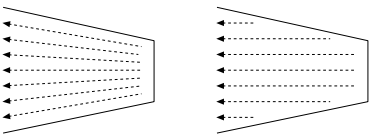


Figure 23: Cross section of light shape, with radial and parallel light rays.

GIVEN: POINT P ON SURFACE
COMPUTE: COLOR AND DIRECTION OF INCIDENT RAY

Each light is defined in local coordinates such that the pyramid is along the z *axis with its apex at the origin, simplifying clipping and falloff calculations.*

```
Pl = transform P to light's coords
atten = 1.0
// Clip to near/far planes
atten *= step(znear-nearedge, znear, zcomp(Pl))
atten *= step(zfar, zfar+faredge, zcomp(Pl))
// Clip to shape boundary
atten *= clipSuperellipse(Pl) // see appendix A
// Apply blockers
foreach blocker
       Pb = project P into plane of blocker
       atten *= clipSuperellipse(Pb)
end
// Apply cookies
foreach cookie
       Pm = project P into plane of cookie
       atten *= texture(cookie.filename, Pm)
end
// Apply slide filters
foreach slide
       Ps = project P into plane of slide
       color *= texture(slide.filename, Pm)
end
// Apply noise
foreach noise texture
       Ps = project P into noise space
       atten or color *= noise(Ps)
end
// Apply shadows
foreach referenced shadowmap
       Ps = project P into plane of shadow
       atten or color *= test if in shadow
end
// Intensity dropoff
atten *= Falloff(zcomp(Pl))
atten *= BeamDistribution(Pl)
// Final output
output(color) = intensity*color*atten
output(rayDirection) = P (radial) or z (parallel)
```

Figure 24: Overview of light computation.

Plate 1

Plate 2

Plate 3

Plate 4

Plate 5

Plate 6