Appears in Proc. SIGGRAPH 1993,
ACM, June, 1993. pp. 249-252.

Fast Shadows and Lighting Effects Using Texture Mapping

Mark Segal
Carl Korobkin
Rolf van Widenfelt
Jim Foran
Paul Haeberli
Silicon Graphics Computer Systems*

Abstract

Generating images of texture mapped geometry requires projecting surfaces onto a two-dimensional screen. If this projection involves perspective, then a division must be performed at each pixel of the projected surface in order to correctly calculate texture map coordinates.

We show how a simple extension to perspective-correct texture mapping can be used to create various lighting effects. These include arbitrary projection of two-dimensional images onto geometry, realistic spotlights, and generation of shadows using shadow maps[10]. These effects are obtained in real time using hardware that performs correct texture mapping.

CR Categories and Subject Descriptors: I.3.3 **[Computer Graphics]:** Picture/Image Generation; I.3.7 **[Computer Graphics]:** Three-Dimensional Graphics and Realism - *color, shading, shadowing, and texture*

Additional Key Words and Phrases: lighting, texture mapping

1 Introduction

Producing an image of a three-dimensional scene requires finding the projection of that scene onto a two-dimensional screen. In the case of a scene consisting of texture mapped surfaces, this involves not only determining where the projected points of the surfaces should appear on the screen, but also which portions of the texture image should be associated with the projected points.

If the image of the three-dimensional scene is to appear realistic, then the projection from three to two dimensions must be a perspective projection. Typically, a complex scene is converted to polygons before projection. The projected vertices of these polygons determine boundary edges of projected polygons.

Scan conversion uses iteration to enumerate pixels on the screen that are covered by each polygon. This iteration in the plane of projection introduces a homogeneous variation into the parameters that index the texture of a projected polygon. We call these parameters *texture coordinates*. If the homogeneous variation is ignored in favor of a simpler linear iteration, incorrect images are produced that can lead to objectionable effects such as texture "swimming" during scene animation[5]. Correct interpolation of texture coordinates requires each to be divided by a common denominator for each pixel of a projected texture mapped polygon[6].

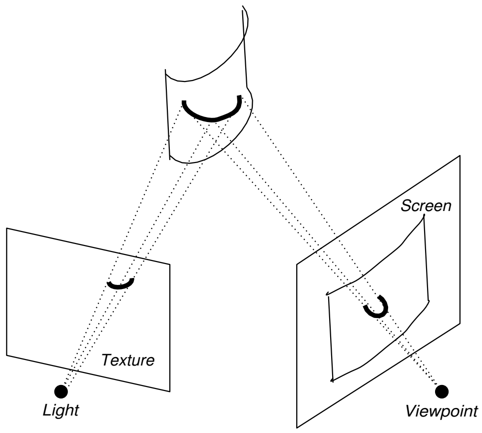

Figure 1. Viewing a projected texture.

We examine the general situation in which a texture is mapped onto a surface via a projection, after which the surface is projected onto a two dimensional viewing screen. This is like projecting a slide of some scene onto an arbitrarily oriented surface, which is then viewed from some viewpoint (see Figure 1). It turns out that handling this situation during texture coordinate iteration is essentially no different from the more usual case in which a texture is mapped linearly onto a polygon. We use *projective textures* to simulate spotlights and generate shadows using a method that is well-suited to graphics hardware that performs divisions to obtain correct texture coordinates.

2 Mathematical Preliminaries

To aid in describing the iteration process, we introduce four coordinate systems. The *clip* coordinate system is a homogeneous representation of three-dimensional space, with x, y, z, and w coordinates. The origin of this coordinate system is the viewpoint. We use the term clip coordinate system because it is this system in which clipping is often carried out. The *screen* coordinate system represents the two-dimensional screen with two coordinates. These are obtained from clip coordinates by dividing x and y by w, so that screen coordinates are given by $x^s = x/w$ and $y^s = y/w$ (the s superscript indicates screen coordinates). The *light* coordinate system is a second homogeneous coordinate system with coordinates x^l, y^l, z^l, and w^l; the origin of this system is at the light source. Finally,

*2011 N. Shoreline Blvd., Mountain View, CA 94043

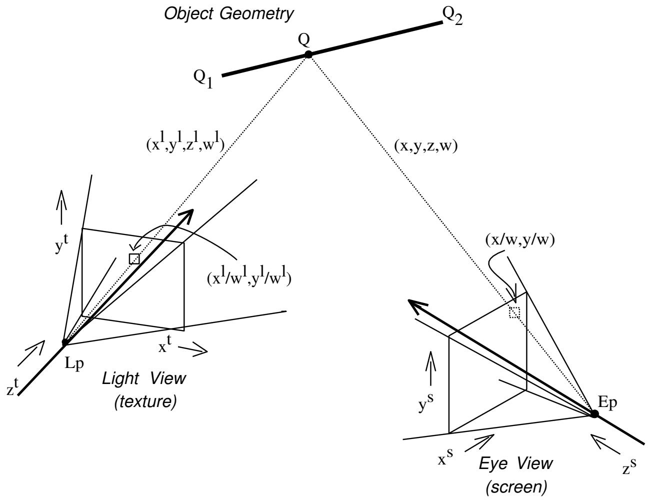

Figure 2. Object geometry in the light and clip coordinate systems.

the *texture* coordinate system corresponds to a texture, which may represent a slide through which the light shines. Texture coordinates are given by $x^t = x^l/w^l$ and $y^t = y^l/w^l$ (we shall also find a use for $z^t = z^l/w^l$). Given (x^s, y^s), a point on a scan-converted polygon, our goal is to find its corresponding texture coordinates, (x^t, y^t).

Figure 2 shows a line segment in the clip coordinate system and its projection onto the two-dimensional screen. This line segment represents a span between two edges of a polygon. In clip coordinates, the endpoints of the line segment are given by

$$\mathbf{Q}_1 = (x_1, y_1, z_1, w_1) \qquad \text{and} \qquad \mathbf{Q}_2 = (x_2, y_2, z_2, w_2).$$

A point $\mathbf{Q}$ along the line segment can be written in clip coordinates as

$$\mathbf{Q} = (1-t)\mathbf{Q}_1 + t\mathbf{Q}_2 \tag{1}$$

for some $t \in [0, 1]$. In screen coordinates, we write the corresponding projected point as

$$\mathbf{Q}^s = (1-t^s)\mathbf{Q}_1^s + t^s\mathbf{Q}_2^s \tag{2}$$

where $\mathbf{Q}_1^s = \mathbf{Q}_1/w_1$ and $\mathbf{Q}_2^s = \mathbf{Q}_2/w_2$.

To find the light coordinates of $\mathbf{Q}$ given $\mathbf{Q}^s$, we must find the value of t corresponding to t^s (in general $t \neq t^s$). This is accomplished by noting that

$$\mathbf{Q}^s = (1-t^s)\mathbf{Q}_1/w_1 + t^s\mathbf{Q}_2/w_2 = \frac{(1-t)\mathbf{Q}_1 + t\mathbf{Q}_2}{(1-t)w_1 + tw_2} \tag{3}$$

and solving for t. This is most easily achieved by choosing a and b such that $1 - t^s = a/(a+b)$ and $t^s = b/(a+b)$; we also choose A and B such that $(1-t) = A/(A+B)$ and $t = B/(A+B)$. Equation 3 becomes

$$\mathbf{Q}^s = \frac{a\mathbf{Q}_1/w_1 + b\mathbf{Q}_2/w_2}{(a+b)} = \frac{A\mathbf{Q}_1 + B\mathbf{Q}_2}{Aw_1 + Bw_2}. \tag{4}$$

It is easily verified that $A = aw_2$ and $B = bw_1$ satisfy this equation, allowing us to obtain t and thus $\mathbf{Q}$.

Because the relationship between light coordinates and clip coordinates is affine (linear plus translation), there is a homogeneous matrix M that relates them:

$$\mathbf{Q}^l = M\mathbf{Q} = \frac{A}{A+B}\mathbf{Q}_1^l + \frac{B}{A+B}\mathbf{Q}_2^l \tag{5}$$

where $\mathbf{Q}_1^l = (x_1^l, y_1^l, z_1^l, w_1^l)$ and $\mathbf{Q}_2^l = (x_2^l, y_2^l, z_2^l, w_2^l)$ are the light coordinates of the points given by $\mathbf{Q}_1$ and $\mathbf{Q}_2$ in clip coordinates.

We finally obtain

$$\begin{aligned} \mathbf{Q}^t &= \mathbf{Q}^l/w^l \\ &= \frac{A\mathbf{Q}_1^l + B\mathbf{Q}_2^l}{Aw_1^l + Bw_2^l} \\ &= \frac{a\mathbf{Q}_1^l/w_1 + b\mathbf{Q}_2^l/w_2}{a(w_1^l/w_1) + b(w_2^l/w_2)}. \end{aligned} \tag{6}$$

Equation 6 gives the texture coordinates corresponding to a linearly interpolated point along a line segment in screen coordinates. To obtain these coordinates at a pixel, we must linearly interpolate x^l/w, y^l/w, and w^l/w, and divide at each pixel to obtain

$$x^l/w^l = \frac{x^l/w}{w^l/w} \qquad \text{and} \qquad y^l/w^l = \frac{y^l/w}{w^l/w}. \tag{7}$$

(For an alternate derivation of this result, see [6].)

If w^l is constant across a polygon, then Equation 7 becomes

$$s = \frac{s/w}{1/w} \qquad \text{and} \qquad t = \frac{t/w}{1/w}, \tag{8}$$

where we have set $s = x^l/w^l$ and $t = y^l/w^l$. Equation 8 governs the iteration of texture coordinates that have simply been assigned to polygon vertices. It still implies a division for each pixel contained in a polygon. The more general situation of a projected texture implied by Equation 7 requires only that the divisor be w^l/w instead of $1/w$.

3 Applications

To make the various coordinates in the following examples concrete, we introduce one more coordinate system: the *world* coordinate system. This is the coordinate system in which the three-dimensional model of the scene is described. There are thus two transformation matrices of interest: M_c transforms world coordinates to clip coordinates, and M_l transforms world coordinates to light coordinates. Iteration proceeds across projected polygon line segments according to equation 6 to obtain texture coordinates (x^t, y^t) for each pixel on the screen.

3.1 Slide Projector

One application of projective texture mapping consists of viewing the projection of a slide or movie on an arbitrary surface[9][2]. In this case, the texture represents the slide or movie. We describe a multi-pass drawing algorithm to simulate film projection.

Each pass entails scan-converting every polygon in the scene. Scan-conversion yields a series of screen points and corresponding texture points for each polygon. Associated with each screen point is a color and z-value, denoted c and z, respectively. Associated with each corresponding texture point is a color and z-value, denoted c_τ and z_τ. These values are used to modify corresponding values in a framebuffer of pixels. Each pixel, denoted p, also has an associated color and z-value, denoted c_p and z_p.

A color consists of several indepenedent components (e.g. red, green, and blue). Addition or multiplication of two colors indicates addition or multiplication of each corresponding pair of components (each component may be taken to lie in the range $[0, 1]$).

Assume that z_p is initialized to some large value for all p, and that c_p is initialized to some fixed ambient scene color for all p. The slide projection algorithm consists of three passes; for each scan-converted point in each pass, these actions are performed:

Pass 1 If $z < z_p$, then $z_p \leftarrow z$ *(hidden surface removal)*

Pass 2 If $z = z_p$, then $c_p \leftarrow c_p + c_\tau$ *(illumination)*

Pass 3 Set $c_p = c \cdot c_p$ *(final rendering)*

Pass 1 is a z-buffering step that sets z_p for each pixel. Pass 2 increases the brightness of each pixel according to the projected spotlight shape; the test ensures that portions of the scene visible from the eye point are brightened by the texture image only once (occlusions are not considered). The effects of multiple film projections may be incorporated by repeating Pass 2 several times, modifying M_l and the light coordinates appropriately on each pass. Pass 3 draws the scene, modulating the color of each pixel by the corresponding color of the projected texture image. Effects of standard (i.e. non-projective) texture mapping may be incorporated in this pass. Current Silicon Graphics hardware is capable of performing each pass at approximately 10^5 polygons per second.

Figure 3 shows a slide projected onto a scene. The left image shows the texture map; the right image shows the scene illuminated by both ambient light and the projected slide. The projected image may also be made to have a particular focal plane by rendering the scene several times and using an accumulation buffer as described in [4].

The same configuration can transform an image cast on one projection plane into a distinct projection plane. Consider, for instance, a photograph of a building's facade taken from some position. The effect of viewing the facade from arbitrary positions can be achieved by projecting the photograph back onto the building's facade and then viewing the scene from a different vantage point. This effect is useful in walk-throughs or fly-bys; texture mapping can be used to simulate buildings and distant scenery viewed from any viewpoint[1][7].

3.2 Spotlights

A similar technique can be used to simulate the effects of spotlight illumination on a scene. In this case the texture represents an intensity map of a cross-section of the spotlight's beam. That is, it is as if an opaque screen were placed in front of a spotlight and the intensity at each point on the screen recorded. Any conceivable spot shape may be accommodated. In addition, distortion effects, such as those attributed to a shield or a lens, may be incorporated into the texture map image.

Angular attenuation of illumination is incorporated into the intensity texture map of the spot source. Attenuation due to distance may be approximated by applying a function of the depth values $z^t = z^l/w^l$ iterated along with the texture coordinates (x^t, y^t) at each pixel in the image.

This method of illuminating a scene with a spotlight is useful for many real-time simulation applications, such as aircraft landing lights, directable aircraft taxi lights, and automotive headlights.

3.3 Fast, Accurate Shadows

Another application of this technique is to produce shadows cast from any number of point light sources. We follow the method described by Williams[10], but in a way that exploits available texture mapping hardware.

First, an image of the scene is rendered from the viewpoint of the light source. The purpose of this rendering is to obtain depth values in light coordinates for the scene with hidden surfaces removed. The depth values are the values of z^l/w^l at each pixel in the image. The array of z^t values corresponding to the hidden surface-removed image are then placed into a texture map, which will be used as a *shadow map*[10][8]. We refer to a value in this texture map as z_τ.

The generated texture map is used in a three-pass rendering process. This process uses an additional framebuffer value α_p in the range $[0, 1]$. The initial conditions are the same as those for the slide projector algorithm.

Pass 1 If $z < z_p$, then $z_p \leftarrow z$, $c_p \leftarrow c$ *(hidden surface removal)*

Pass 2 If $z_\tau = z^t$, then $\alpha_p \leftarrow 1$; else $\alpha_p \leftarrow 0$ *(shadow testing)*

Pass 3 $c_p \leftarrow c_p + (c$ modulated by $\alpha_p)$ *(final rendering)*

Pass 1 produces a hidden surface-removed image of the scene using only ambient illumination. If the two values in the comparison in Pass 2 are equal, then the point represented by p is visible from the light and so is not in shadow; otherwise, it is in shadow. Pass 3, drawn with full illumination, brightens portions of the scene that are not in shadow.

In practice, the comparison in Pass 2 is replaced with $z_\tau > z^t + \epsilon$, where ϵ is a bias. See [8] for factors governing the selection of ϵ.

This technique requires that the mechanism for setting α_p be based on the result of a comparison between a value stored in the texture map and the iterated z^t. For accuracy, it also requires that the texture map be capable of representing large z_τ. Our latest hardware posseses these capabilites, and can perform each of the above passes at the rate of at least 10^5 polygons per second.

Correct illumination from multiple colored lights may be produced by performing multiple passes. The shadow effect may also be combined with the spotlight effect described above, as shown in Figure 4. The left image in this figure is the shadow map. The center image is the spotlight intensity map. The right image shows the effects of incorporating both spotlight and shadow effects into a scene.

This technique differs from the hardware implementation described in [3]. It uses existing texture mapping hardware to create shadows, instead of drawing extruded shadow volumes for each polygon in the scene. In addition, percentage closer filtering [8] is easily supported.

4 Conclusions

Projecting a texture image onto a scene from some light source is no more expensive to compute than simple texture mapping in which texture coordinates are assinged to polygon vertices. Both require a single division per-pixel for each texture coordinate; accounting for the texture projection simply modifies the divisor.

Viewing a texture projected onto a three-dimensional scene is a useful technique for simulating a number of effects, including projecting images, spotlight illumination, and shadows. If hardware is available to perform texture mapping and the per-pixel division it requires, then these effects can be obtained with no performance penalty.

Acknowledgements

Many thanks to Derrick Burns for help with the texture coordinate iteration equations. Thanks also to Tom Davis for useful discus-

Figure 3. Simulating a slide projector.

Figure 4. Generating shadows using a shadow map.

sions. Dan Baum provided helpful suggestions for the spotlight implementation. Software Systems provided some of the textures used in Figure 3.

References

[1] Robert N. Devich and Frederick M. Weinhaus. Image perspective transformations. *SPIE*, 238, 1980.

[2] Julie O'B. Dorsey, Francois X. Sillion, and Donald P. Greenberg. Design and simulation of opera lighting and projection effects. In *Proceedings of SIGGRAPH '91*, pages 41–50, 1991.

[3] Henry Fuchs, Jack Goldfeather, and Jeff P. Hultquist, et al. Fast spheres, shadows, textures, transparencies, and image enhancements in pixels-planes. In *Proceedings of SIGGRAPH '85*, pages 111–120, 1985.

[4] Paul Haeberli and Kurt Akeley. The accumulation buffer: Hardware support for high-quality rendering. In *Proceedings of SIGGRAPH '90*, pages 309–318, 1990.

[5] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, UC Berkeley, June 1989.

[6] Paul S. Heckbert and Henry P. Moreton. Interpolation for polygon texture mapping and shading. In David F. Rogers and Rae A. Earnshaw, editors, *State of the Art in Computer Graphics: Visualization and Modeling*, pages 101–111. Springer-Verlag, 1991.

[7] Kazufumi Kaneda, Eihachiro Nakamae, Tomoyuki Nishita, Hideo Tanaka, and Takao Noguchi. Three dimensional terrain modeling and display for environmental assessment. In *Proceedings of SIGGRAPH '89*, pages 207–214, 1989.

[8] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *Proceedings of SIGGRAPH '87*, pages 283–291, 1987.

[9] Steve Upstill. *The RenderMan Companion*, pages 371–374. Addison Wesley, 1990.

[10] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH '78*, pages 270–274, 1978.