Light-Dependent Texture Mapping

Dan Gelb, Tom Malzbender, Kevin Wu
Client and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-98-131 (R.1)
April 26th , 2001*

E-mail: malzbend@hpl.hp.com

texture mapping, image based rendering, computer graphics, bump mapping

In this paper we present a method for performing light-dependent texture mapping that can be implemented in current texture mapping hardware with low additional cost. Our method gives the impression of finely modeled surface detail that is properly illuminated as lighting moves relative to the object. As an image-based technique, it requires no complex geometric models as input, only a set of photographs or renderings of an object under varying light conditions. Light-dependent variations are captured by per-texel polynomial functions designed to be efficiently evaluated via slightly modified multi-texturing hardware.

* Internal Accession Date Only Approved for External Publication

Light-Dependent Texture Mapping

Dan Gelb[1], Tom Malzbender[2], Kevin Wu

Visual Computing Department
Hewlett-Packard Laboratories
Palo Alto, California

June 15, 1998

Abstract

In this paper we present a method for performing light-dependent texture mapping that can be implemented in current texture mapping hardware with low additional cost. Our method gives the impression of finely modeled surface detail that is properly illuminated as lighting moves relative to the object. As an image-based technique, it requires no complex geometric models as input, only a set of photographs or renderings of an object under varying light conditions. Light-dependent variations are captured by per-texel polynomial functions designed to be efficiently evaluated via slightly modified multi-texturing hardware.

1. Introduction

Traditional texture mapping is used to give the impression of geometric detail in a model using an image. For example, a photograph of a brick wall may be used as a texture map on a planar surface to avoid modeling the complex surface detail of the brick. However, if the lighting in the synthetic environment where the texture map is used is different from the lighting the texture map was captured under the resulting rendering will appear incorrect and unrealistic. In addition, if an image is used as a texture map on an object with a different shape than the input image then the lighting effects captured in the texture may not match the lighting in the synthetic geometric environment. If the texture is blended with the calculated lighting of a geometric surface then the resulting rendering will look very flat and smooth to the viewer (see Figure 1).

[1] Currently at Cornell University, Ithaca, NY. E-mail: dgelb@graphics.cornell.edu

[2] Hewlett Packard Laboratories, 1501 Page Mill Rd, M.S. 3U-4, Palo Alto, CA 94304. Tel.: 1-650- 857-6760. E-mail: malzbend@hpl.hp.com

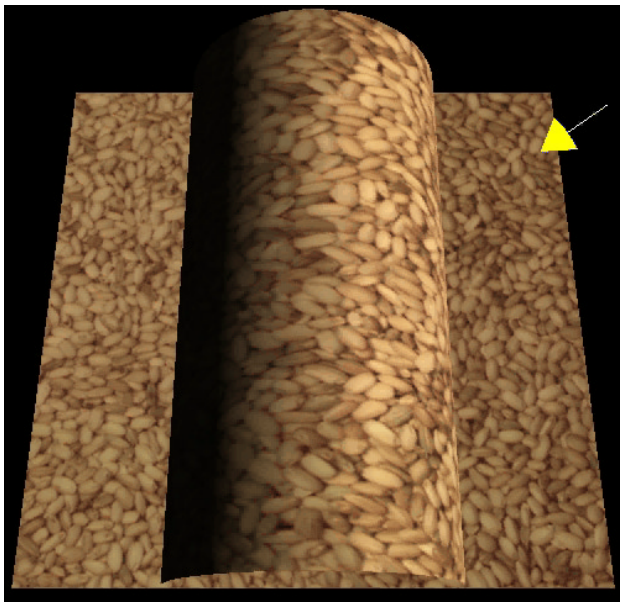

Figure 1: Texture map modulated with Phong illuminated cylinder and plane. Yellow arrow indicates light source direction.

Bump mapping [Blinn 78] is one proposed solution to this problem. Bump mapping is a technique that perturbs the surface normals of the underlying geometry according to a bump map. Introducing variations in the surface normals causes the lighting method to render the surface as though it had local surface variations instead of just a smooth surface. As a result, as the light is moved around the object highlights appear due to the bump map and the surface appears to be rough or grooved or similarly modified as desired.

In general, bump maps are either hand modeled, or more typically calculated procedurally. Creating a bump map to be used with real world textures from photographs is generally difficult. Methods have been developed that attempt to automatically generate bump maps from a set of input images under known light directions [Rushmeier 97]. These methods have difficulty with generating bump maps for objects with large surface variations that cause self-shadowing and intra-object interreflections. In addition, current bump map rendering techniques do not render shadows due to surface variation or brightened regions due to interreflections.

Our technique in contrast is an image-based technique that requires no modeling of complex geometry or bump maps. The input data required is a set of images of the desired object to be used as a texture, each one under illumination from a different known direction, all captured from the same view point. The input light directions need not be uniformly spaced in the hemispherical set of possible light directions.

Interpolating the input images to create textures from arbitrary light directions would be very costly both in memory and bandwidth. For each texture element (texel) in our texture map we would have to store a color sample for all input light positions. Instead of storing a large set of color samples at each texel, we use a simple second order polynomial at each texel to approximate the change in color

of the texel as a function of light direction. Our method currently uses a polynomial to approximate the brightness (non-linear luminance) of each texel, keeping the chromaticity constant.

The result of our method is a texture map that properly reproduces the effects of variations in the illuminant direction relative to the object, whether due to the surface orientation of the texture mapped object, or to changing the location of the source. Renderings using our method are very realistic, and require little or no user input once the input images are acquired.

2. Data Acquisition and Direct Rendering

The input to our system consists of a large number (40 in our experiments) of images of the object to be used as a texture map, each image taken with the light at a different direction relative to the object. Each light direction is a sample from the set of all possible light directions, the hemisphere above the sample object surface[3]. In order to simplify our data acquisition, we created a once subdivided icosahedral dome, and positioned the light at the center of each triangular face of the dome (Figure 2). The camera was in a fixed position at the top of the dome. The data we capture is equivalent to a set of samples of a bidirectional texture function (BTF) [Dana 97], except we only have samples for one view direction. Therefore, our texture maps assume the object exhibits no view-dependent effects.

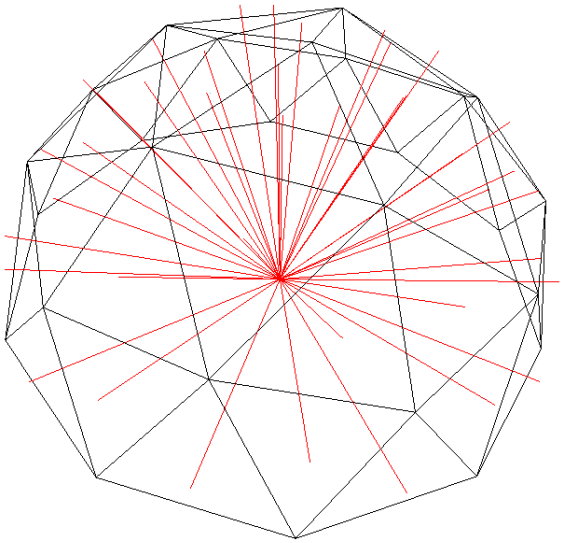

Figure 2: Black lines indicate the edges of the subdivided icosahedron. Red lines indicated light direction relative to sample and the center of the dome.

We can render texture maps directly from the input data using a technique analogous to view-dependent texture mapping [Debevec 96]. If we chose the color of a texel with a particular input

[3] We assume the object receives no light when the illuminant is behind the object.

light direction based solely on the nearest input sample the rendering would exhibit discontinuities between pixels and flicker with changing light directions because the nearest sample direction would vary abruptly as light direction changed. Instead we triangulate the space of input light directions, and choose a texel's color based on the weighted average of the three sample points at the vertices of the triangle the light vector intersects. We perform a Delaunay triangulation to generate a triangular mesh based on our input sample directions. If we determined the weights for each of the three input samples based on Euclidean distance we would still experience discontinuities as the sample light direction crossed triangle boundaries. To resolve this problem we used Barycentric coordinates to determine the weighting. Figure 3 illustrates an example rendering directly from the input data using this technique.

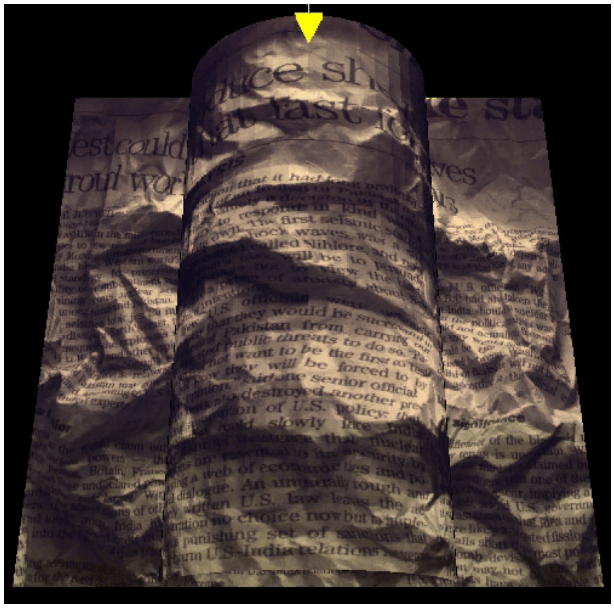

Figure 3: Direct light-dependent texture mapping of a crumpled newspaper texture

3. Fitting and Color conversion

Instead of storing a large number of sample color values, one for each input light direction for every texel, our goal is to approximate the change in color with a polynomial that is a function of the light direction. We model the color Y (assuming one color channel for now) by the polynomial:

$$Y = Av_x^2 + Bv_y^2 + Cv_x v_y + Dv_x + Ev_y + F$$

Where v_x and v_y are the projections of the vector to the light in the local texture coordinate space, and A-F are six fitted coefficients. Figure 4 shows how v_x and v_y are calculated from the normal N and texture axes s and t, based on the vector to the light L. Parameterizing light direction by the projection of the vector into the texture space has the advantage of having no discontinuities as L varies over the hemisphere. A more traditional parameterization by angle and elevation has a

discontinuity between 0 and 360 degrees. One disadvantage of this projection based method is that a special check must be included to differentiate between light directions in the upper and lower hemispheres.

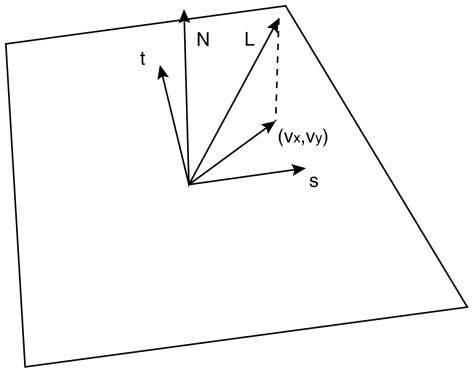

Figure 4: Calculation of v_x and v_y by projecting L into local texture space.

The polynomial coefficients, A through F, are calculated independently for every texel. Instead of storing color values at every texel in our texture map, we are storing 6 polynomial coefficients. As a result every texel holds a polynomial that approximates the appearance of the texel as a function of the light direction. The coefficients are calculated based on a least squares fit of all the input samples. We currently use a simple normal equations method [Press 92]. Figure 5 shows a comparison of rendering directly from the input images in the top half and from fitted polynomials (one per color channel) in the bottom half. The only visibly noticeable difference is slight smoothing of lighting detail in the polynomial based image.

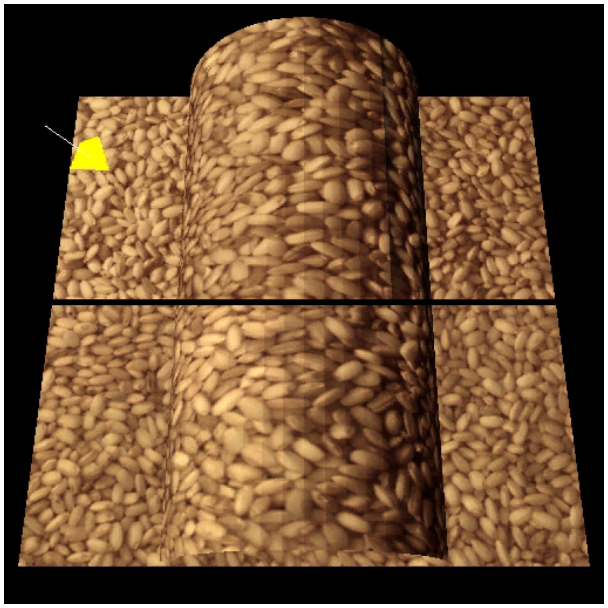

Figure 5: Top half: Light dependent texture mapping directly from input images. Bottom half: Light dependent texture mapping using three polynomials per texel.

The result of the fitting operation is six floating-point coefficients. We would like to store these as 8 bit integers for evaluation speed, and so that all the polynomial coefficients may be stored in two 24-bit images. This is a non-trivial problem since there are typically several orders of magnitude difference between the high and low order coefficients. In addition, different textures may have very different coefficient ranges. To eliminate this problem we store three shift values (high, medium, and low order coefficient shifts) for each texture. These shift values specify how many bits the coefficients have been shifted before they where stored in the coefficient files, and correspondingly how many bits to shift by in the polynomial evaluation to undo the shift. For example, a high order coefficient may have a value of 0.0128. We examine the ranges of all high order coefficients and determine that a left shift of 13 bits places them all in the range 0 to 256. 0.0128 is multiplied by 2^{13}, and stored as 105. After polynomial evaluation we right shift the result by 13 bits to calculate the proper result. In order to allow for negative coefficients, we also add a scalar, typically 127, to each coefficient before storage, and subtract it when reading in the coefficients. These operations are very simple and can be implemented as integer shifts and adds in hardware.

The polynomial we used is specifically designed to be integrated into multi-texture capable graphics hardware. The six coefficients are all eight bit values so that the polynomial can be evaluated by using multi-texturing to combine two 24 bit textures. The only hardware changes that are needed are eight bit multipliers to multiply the coefficients by the appropriate v_x and v_y values, and eight bit adders to combine the output multi-texture channels into the output color Y.

If fitting where performed in a straightforward manner directly on the original input images then three independent polynomials would have to be calculated per texel, one for each color channel. Evaluating three different polynomials per texel would be computationally expensive, and would not be possible to integrate easily into current multitexturing hardware. Our method must require only one polynomial evaluation per texel to be practical. One possible solution would be to convert the RGB input images into color-indexed images, and then fit one polynomial to the indices. This would require additional look-up table hardware to convert back into RGB color after the polynomial evaluation. The primary difficulty with a color-indexed method is constructing a small color table (256 entries) that spans the color space of the input images satisfactorily, and results in accurate color transitions with small changes in the index. As the light direction changes the output index from the polynomial evaluation will change correspondingly. The color values generated from the look-up must not change wildly as the index changes. For example, on most surfaces the output hue must not fluctuate if the light elevation increases slightly. Constructing a single look-up table that covers the needed regions of the color space, and varies smoothly enough so as not to cause visible artifacts may be difficult or impossible for different textures. This technique may still be worth investigating however, as color tables exist in current hardware and are very efficient.

The solution we chose was to convert the input images into a color space that separates the luminance from the chromaticity, and fitting only the luminance. Chromaticity is assumed to stay almost constant as light direction changes, and is stored separately. The luminance changes as a function of light direction are fitted with a single polynomial per texel. Rendering requires evaluating the polynomial, then performing a 3x3 matrix multiply to convert back into RGB color space. This would require additional color matrix hardware to transform the texture map output. This capability does exist in some current graphics hardware [OpenGL 97]. If implementing the

color matrix in hardware is too costly, it may be possible to approximate the color space conversion using a sufficiently large color look-up table. Our current software implementation uses a 3x3 floating point matrix multiply. Slight color variation from the input images may occur due to storing a constant chromaticity per texel, but the variations were not very noticeable or disturbing in our experiments. Figure 6 shows an example image where the top half is rendered using three polynomials per texel, while the bottom half is a single luminance polynomial per texel.

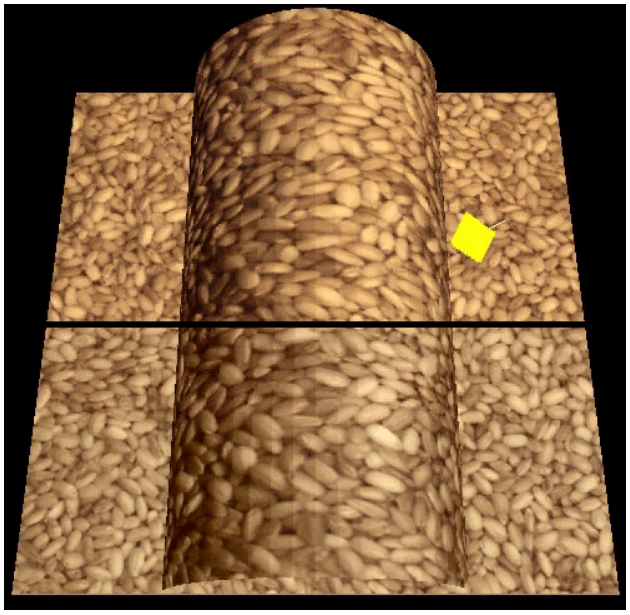

Figure 6: Top half: Light dependent texture mapping using three polynomials per texel. Bottom half: Light dependent texture mapping using a luminance polynomial per texel.

For our experiments, we used the YCBCR color space, fitting only the Y component. YCBCR is frequently used with digital images, and is used in the JPEG and MPEG compression schemes. It can also be converted to and from non-linear RGB using 3x3 matrixes, so it suits our needs well. YCBCR is however a non-linear color space, so it may not be a good choice for implementation in graphics hardware that does gamma correction after texture mapping. We used a non-linear color space because our input RGB images that were captured from a digital camera were non-linear. In addition the rendering library we used, an OpenGL like library called Mesa, did not do gamma correction.

The color space that is used in practice need not be fixed. It might be useful to examine the input images and determine a color conversion that gives one variant component, and two components that change very little as a function of light direction. The appropriate matrix to convert back to RGB would then need to be stored along with the polynomial coefficients. Calculating a suitable color space may be needed if luminance is not the dominant axis of color change, such as if the illuminant color is far from white.

To be useful in current rendering applications and to eliminate artifacts our method must support mip-mapping. For small changes in light direction our polynomial is effectively linear. As a result

we can mip-map the A-F polynomial coefficients, similarly to how the color values in traditional texture maps are mip-mapped. Without mip-mapping the texel polynomials would be point sampled in u,v space, resulting in disturbing aliasing artifacts.

4. Related Work and Extensions

Our method as we've described it could also be used as a view-dependent texture mapping technique. In view-dependent texture mapping, instead of having a polynomial that is a function of light direction it would be a function of the viewer's direction. This would allow the texture to reproduce view-dependent effects such as specular highlights, and possibly roughly capture depth and occlusion effects similar to a Light field [Levoy 96]. The input required is a set of input images taken from known view directions relative to the object. Lighting would need to be kept constant for all input images, which is a non-trivial task.

View-dependent texture mapping would not work flawlessly with our polynomial approximations however. If the texture object is very non-planar then a texel may fall on very different parts of the object as the view direction changes. A simple low order polynomial is unlikely to be able to capture the high frequency changes that may result. For example, if the object was a piece of fur, as the camera moves around the color of each texel may vary radically as different hairs fall under the pixel.

5. Conclusions

We have presented a method that requires only images to generate high quality photorealistic renderings of a textured surface. Our method captures light dependent effects, whether they are due to changes in the illuminant direction, or surface orientation of the texture mapped object. It can render changes in brightness due to shadows and indirect lighting that cannot be reproduced with existing bump mapping hardware, or Phong illuminated geometric objects. Our technique could be integrated into current graphics hardware with minor additional hardware requirements.

References

[Blinn 78] J. F. Blinn, Computer display of curved surfaces. Ph.D. Thesis, University of Utah, 1978.

[Dana 97] K. J. Dana, S. K. Nayar, B. van Ginneken, and J. J. Koenderink. Reflectance and texture of real-world surfaces. In *Proceedings of Computer Vision and Pattern Recognition 97*, pages 151-157, 1997.

[Debevec 96] P.E. Debevec, C.J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH 96)*, pages 11-20, 1996.

[Levoy 96] M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH 96)*, pages 31-42, 1996.

[Press 92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing*, Cambridge University Press, New York, N. Y., 1992.

[OpenGL 97] OpenGL 1.2 imaging extension specification, 1997.
http://www.opengl.org/Developers/Documentation/Version1.2/1.2specs/color_matrix.txt

[Rushmeier 97] H. Rushmeier, G. Taubin, A. Guéziec. Applying shape from lighting variation to bump map capture. In *Eurographics Rendering Workshop Proceedings 1997*, pages 35-44, 1997.