Image-Based Lighting for Games

Dan Gelb, Tom Malzbender
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2004-225
December 7, 2004*

E-mail: dan.gelb@hp.com, tom.malzbender@hp.com

computer graphics, texture mapping, image-based rendering

Image-based rendering techniques are used extensively in interactive electronic games. Most frequently image-based rendering appears in the form of texture mapping when a photograph or image is used to represent a complex object's appearance. Texture mapping has several significant limitations, the primary one being that only a single lighting condition is captured. If dynamic lighting is needed, texture mapping alone is insufficient. This report describes some existing techniques to solve this problem as well as a technique for image-based lighting developed at HP Labs. Our method, Polynomial Texture Mapping (PTM) uses biquadtatic polynomials at every texel to reconstruct surface appearances under varying lighting conditions. Unlike previous methods, PTMs can capture complex illumination effects such as self-shadowing and interreflections. Our method can be efficiently implemented on programmable graphics hardware which is common to most current and future electronic gaming platforms.

* Internal Accession Date Only
Approved for External Publication

Image-Based Lighting for Games

Dan Gelb, Tom Malzbender

Mobile and Media Systems Lab, HP Laboratories

dan.gelb@hp.com, tom.malzbender@hp.com

http://www.hpl.hp.com/research/ptm/

Abstract

Image-based rendering techniques are used extensively in interactive electronic games. Most frequently image-based rendering appears in the form of texture mapping when a photograph or image is used to represent a complex object's appearance. Texture mapping has several significant limitations, the primary one being that only a single lighting condition is captured. If dynamic lighting is needed, texture mapping alone is insufficient. This report describes some existing techniques to solve this problem as well as a technique for image-based lighting developed at HP Labs. Our method, Polynomial Texture Mapping (PTM) uses biquadtatic polynomials at every texel to reconstruct surface appearances under varying lighting conditions. Unlike previous methods, PTMs can capture complex illumination effects such as self-shadowing and interreflections.Our method can be efficiently implemented on programmable graphics hardware which is common to most current and future electronic gaming platforms.

Introduction

Image-based rendering is one of the most widely used techniques in games, even though many game developers may not think of their game engines as being heavily image-based. Texture mapping, used throughout 3d graphics, is an image-based rendering technique where images are used in place of complex geometry and material properties. The images used for texturing can be generated by artists, from photographs of real objects, from synthetic renderings, or a combination of multiple sources. However, if realistic renderings are a primary objective, then beginning with photographs is quick and straightforward way to achieve photorealism. By starting with a photograph from the real world it is possible to capture complex light interactions such as interreflections, self shadowing, and subsurface scattering without modeling and simulation.

Perhaps the main disadvantage of using a photograph as your rendering source is that the photograph captures the appearance of your object from a single viewpoint, under a static lighting condition, at a single point in time. If the desired rendered environment in your game does not exactly match the acquired conditions then your texture will not appear realistic. While you may be able closely match the in game environment with your source environment, you probably want a dynamically changing scene, where the lighting can vary, as well as the user's viewpoint. For an image-based technique to be dynamic a single static image is obviously insufficient.

Bump mapping and related techniques

Bump mapping, normal mapping, and other related techniques have been developed to extend texture mapping to reproduce such light and view dependent dynamic effects. As with texture mapping, these methods store spatially variant information for rendering. However, instead of storing a texture image that already contains some existing illumination effects as in typical texture mapping, these methods store parameters that allow a computational lighting model to be varied spatially. Since these methods are often used with simple synthetic reflectance models such as Phong lighting they achieve dynamic illumination at the expense of photorealism.

Bump maps and other similar methods typically use a map that has been generated by an artist or a procedural method, rather than one captured from real world objects. Creating a bump map from photographs of real objects is difficult, and is generally not done for games or other realistic renderings. Research techniques have been developed to calculate bump maps automatically from multiple images of an object under known light directions [Rushmeier 97]. Unfortunately these methods typically have difficulty generating bump maps from objects that contain self-shadowing and intra-object interreflections. Even if methods could be derived that could generate bump maps in the presence of such illumination effects, the resulting renderings would not recreate them properly due to the limitations of these rendering techniques.

What game developers need is a technique that is image-based which can still be integrated realistically in dynamic, synthetic game environments. Image-based methods yield photorealistic renderings that are based on data from simple acquisition methods (photographs). Dynamic behavior in response to changes in the game environment has been lacking in image-based techniques however, and is desired for realistic renderings in games. In addition, there are additional requirements for usefulness, including real time performance, compact representation, and simple acquisition.

Light-Dependent Texture Mapping and Beyond

As we've described, the conventional texture mapping representation of a single RGB triple per texel is insufficient to represent a surface where the appearance may change as a function of light direction, view direction, or some other parameter. Clearly a higher-dimensional representation is needed per texel to allow the object to be rendered differently as the desired rendering parameters are changed. View-dependent texture mapping [Debevec 96] extends texture mapping to allow the appearance of each texel to change as a function of the viewer's position. The technique was intended for reproducing real objects with only approximate geometry but multiple images. As originally presented, view-dependent texture mapping stores multiple images captured from different viewpoints and projects them onto the desired geometry. The multiple projected textures are combined based on weights calculated from the source viewpoints and output viewpoint. This technique can greatly increase the amount of texture storage that is required, possibly multiplying it by the number of source images if a simple representation is used. More compact representations are possible, however the view-dependent reflectance per texel may be difficult to represent as it can be a very non-linear function. This is because view-dependent effects can change rapidly due to occlusions and other interactions while illumination effects tend to have less high-frequency transitions.

The Bidirectional Texture Function (BTF) [Dana 99] represents a four dimensional function across the texture. The BTF is a function of view direction and incident light direction, and is represented as a database of images. Even when the database sparsely samples the incident and view directions this results in a very large dataset, not suitable for interactive games. Also due to the database representation it is expensive to evaluate the BTF, as each texel requires a lookup into possibly different source images. Finally, capturing a large number of different view and light directions while maintaining accurate calibration and registration is difficult and time consuming.

By eliminating the two dimensions representing view direction from the BTF the resulting data represents a light dependent texture map (also known as a reflectance map). This reduced dimensionality dataset is much smaller, and very simple to acquire as a single fixed view direction is sufficient. Lack of acquisition of view-dependent illumination is not a problem for a many classes of materials and these effects can often be reintroduced using existing methods when necessary. This is illustrated by the many uses of standard texture mapping methods in representing diffuse and near-diffuse objects. Objects with view dependent effects, such as specular materials, are difficult to capture even with view-dependent image-based rendering due to the high-frequency nature of mirror like reflections. As we describe in more detail later, since reflected light typically varies smoothly and has less non-linearities than view-dependence compact approximations can be used to model the appearance. The polynomial texture maps (PTM) is one light-dependent method with a compact and efficient polynomial function at every texel that is evaluated based on the desired incident light direction.

Capture / Generation

Capturing a light-dependent texture map simply requires multiple images of the desired object under multiple known incident light directions. Since the view direction remains fixed, extrinsic camera calibration is generally unnecessary. In fact, if a digital camera is used and care is taken to ensure that the object does not move relative to the camera during acquisition then image registration is not required.

There are many options for constructing devices to capture images of objects under multiple illumination directions. Rigs have been created that have multiple stationary light sources at known positions that can be triggered independently. Other devices use a small number of light sources, allowing the light to be positioned around the object. Figure 1 shows two devices for capturing polynomial texture maps. There are also techniques where a light source is moved around manually and the illuminant direction is not known a priori. By placing an object or objects of known shape and possibly reflectance in the scene along with the target object the incident

illuminant direction can be estimated. For example, shadows cast by the known objects can be used to estimate the illuminant direction. Alternatively, if the reflectance is known, inverse rendering can be used to determine the source.

Figure 1: Dome and arm PTM capture devices

Light-dependent texture maps can also be generated synthetically, rather than from real objects. If a developer is interested in reproducing a complex rendering that cannot be done in real time it is possible to pre-render multiple images and use those to generate an image-based model. For example, a developer can render multiple images of an object with very complex shading under a set of multiple light directions and then generate a PTM from the rendered images. Skin is one such material where a full simulation may be too expensive but where a PTM can effectively render the appearance. For some types of objects it may be possible to algorithmically convert directly from the geometry and material properties to the light-dependent texture map representation. It is also possible for artists to generate and edit maps directly in a paint program or similar software. This may be difficult or non-intuitive however depending on the representation.

PTM Representation

Instead of representing light-dependent texture maps as a large number of source images, a compact representation that can be rapidly evaluated is necessary. The representation used in Polynomial Texture Maps is especially well suited for real-time games. For most materials other than mirrored surfaces the reflected light varies smoothly as a function of incident light direction. This is because materials with rough microstructures scatter light across a range of directions rather than a single reflected direction. As a result, in the illuminant domain the reflected light can be reproduced with low frequency representations. Any high-frequency effects that do exist are blurred in light space and become smoother. Shadows cast by point light sources result in high frequency changes in light space. Modeling with low frequency representations simply softens the hard shadows into what would be caused by area light sources. None of the softening in light space results in perceptually objectionable changes as it results in a plausible rendered appearance. Figure 2 shows two examples of PTMs rendered from different illumination directions.

The representation used in Polynomial Texture Maps is a biquadratic polynomial, which has sufficient degrees of freedom to represent many different types of materials and objects while requiring a small number of parameters per texel. The function used in PTMs is:

$$L(u,v;l_u,l_v) = a_0(u,v)l_u^2 + a_1(u,v)l_v^2 + a_2(u,v)l_u l_v + a_3(u,v)l_u + a_4(u,v)l_v + a_5(u,v)$$

The l_u and l_v input values are the projection of the desired incident light direction onto the tangent and binormal in the local texture coordinate system. The tangent and binormal are calculated as in other tangent space shading methods based on the geometry and texture coordinates. The PTM representation has six coefficients $a_0..a_5$ that are fit to the captured source images for every texel. Six scale and bias values are stored for each PTM to allow the calculated coefficients to be stored at 8-bit precision while reproducing the range parameters. Typical materials do not change their color significantly when the illumination direction is changed. For those materials, a single polynomial per texel can be used to represent the texel luminance function. A separate normalized RGB

image stores the underlying color of the material after factoring out illumination changes. Separate polynomials per color channel can be used for materials that exhibit color changes. For more details see [Malzbender 2001].

Figure 2: PTM renderings of two different PTM images from multiple light directions

One of the advantages of this polynomial representation is that the output luminance depends linearly on the PTM coefficients. This linearity means that applying mip-mapping and texture filtering directly on the coefficients results in the correct appearance from the filtered coefficients. This is an improvement over bump mapping techniques where naively filtering the bump map results in smoothing of the bump surface and incorrect renderings.

When a luminance polynomial and RGB image is used for light-dependent texture mapping the storage requirements are 6 8-bit coefficients and 3 8-bit color values per texel. While this is more data than standard texture maps, this is much less data than the original source images that the PTMs reproduce. Compression algorithms can be used to further reduce the size of the PTM. If compression is needed for offline storage than a variant of JPEG can be used. By using JPEG compression on planes of coefficients for some of the coefficients, and predicting other planes from the intracoded planes significant compression can be achieved with little perceptual artifacts. For compression that can be processed by the GPU other techniques can be investigated. For example, vector quantization can be implemented directly on the GPU and should achieve good compression for PTM datasets.

The PTM technique is also useful for modeling other appearance effects that map well onto the polynomial representation. For example, focus variation can be captured from photographs of a scene at multiple focus depths. The polynomial per texel then models the appearance of a pixel as a function of focal depth rather than light direction. The polynomial function can be reduced to a simpler 1D function as necessary depending on the number of degrees of freedom. Other simple 1D and 2D effects that can be modeled with the low-frequency representation can also be reproduced. High-frequency lighting such as specular highlights can also be rendered by combining PTMs with existing rendering techniques. Most synthetic lighting models require the surface normal for rendering. It is simple to estimate the surface normal for every texel of a PTM representing a diffuse object. Since diffuse objects are brightest when the incident light direction aligns with the surface normal we can solve for the light direction that maximizes the polynomial function to estimate the normal. During rendering the PTM is evaluated as usual to recreate the captured material and then combined with the synthetic specular reflectance.

Texture Synthesis

In order for any texture mapping technique to be useful in games it is usually a requirement that textures can be made tileable. For real world textures this is usually done using texture synthesis algorithms while painted textures this is typically done by the artist. Texture synthesis algorithms for standard texture maps are well known

and there are many methods to choose from. For BTFs and other textures with multidimensional per texel functions there are very few known methods. Those methods that do exist typically involve converting from the image-based BTF representation to an intermediate representation such as a height field before mapping back to the images. These methods rely on some type of 3D reconstruction which results in inaccuracies and a deviation from the image-based representation. However, for the PTM representation it is actually possible to do texture synthesis directly in the space of reflectance functions without 3D reconstruction [Hel-Or 03]

Rendering using Programmable Graphics Hardware

As a result of the simple polynomial model used in PTMs it has been possible to implement the technique on programmable graphics hardware for several generations of GPU. Any hardware with DirectX 8 level or better programmability, including the NVIDIA's Geforce3, ATI's Radeon 8500, and Microsoft's Xbox can implement PTMs and other similar light-dependent texture mapping techniques fully on the GPU using DirectX or OpenGL. Figure 3 shows a screenshot from an environment where every surface is textured with PTMs. As the user moves a light in the environment all surfaces respond realistically as a result of the light-dependent texture maps.

Figure 3: Real-time walkthrough running on programmable graphics hardware

See Code Listing 1 for a sample implementation in NVIDIA's Cg language which can be compiled for DirectX or OpenGL interfaces. The vertex program calculates the projection of the incident light direction onto vertices tangent and binormal and outputs the projections to the fragment shader via the vertices color output. The sample code is for a directional (infinite) light source, but implementing a local light source version is straightforward. The vertex program requires the normal and tangent as input data and calculates the binormal internally. This improves performance on some hardware by reducing memory traffic at the expense of vertex computation.

The fragment shader receives the l_u, l_v values from the vertex shader and the PTM coefficients via two textures. The shader simply evaluates the polynomial to determine the texel luminance and then combines the luminance with the texel's RGB color from a third texture. A similar shader that is straightforward to implement is used in the case where a different polynomial is needed for each color channel. The included shader has a hard coded scale of 4 and bias of 0.5, but this need not be hard coded on current hardware.

There are some workarounds for limitations of early GPU hardware present in the programs that can be replaced for current GPUs. For example, due to intermediate precision issues in the fragment shader it was useful to split the a_5 coefficient and store half in the first a_0-a_2 texture and half in the a_3-a_5 texture. This helped ensure that intermediate computations in the fragment shader didn't get clamped or lose precision. Additionally a static 1-D texture was used to cause the luminance to drop-off as the light vector becomes back-facing to prevent a sharp transition. This texture look-up could be replaced with a simple computation in current hardware.

This technique has excellent performance on all recent hardware due to the simplicity of the algorithm. The vertex shader computation is similar to what is used in many techniques that make use of tangent space operations. The fragment shader is also inexpensive as it only requires fetching a few coefficients from textures and then evaluating a simple polynomial. Because of the greater flexibility of new programmable graphics hardware more complex representations than the biquadratic polynomial used by PTMs are possible while maintaining good performance. For example, if it is observed that there are high frequency lighting effects that are not being captured by the existing polynomial a more complex function could be fit to the source images and evaluated in the fragment shader. High frequency effects such as specular highlights can also be added by estimating per texel normal directions from the PTM functions and relighting using the calculated normals. Algorithms that depend on other parameters, including the view direction, are also possible.

Conclusion

The principle advantage of image-based rendering techniques is that photorealism can be achieved without accurate modeling of complex real-world physical interactions. You can capture and reproduce interesting lighting effects directly from how they appear in reality. Since they are captured in photographs, complex interactions like self-shadowing, interreflections, and sub-surface scattering can be reproduced automatically. A related advantage is that the rendering cost is independent of the complexity of the scene and the surface properties of objects in the scene. The complexity of image-based methods usually depends on the number of images or on the representation rather than the complexity of the scene. Ease of acquisition is another advantage of image-based rendering. Rather than requiring complex modeling or measurements, simple photographs are all that is required.

One common disadvantage of image-based rendering methods is the amount of data storage required. As a result of using a database of images as the representation rather than compact mathematical models storage sizes can be very large. Some techniques, such as the PTM method just described, avoid some of the data storage by approximating the images with a simple but powerful model. Another common limitation of image-based methods arises out of sampling issues. It can be difficult to capture high-frequency effects accurately without extremely dense (and time consuming) sampling. Editing can also be problematic for these methods, depending on the representation.

Despite these limitations image-based rendering and relighting can be very useful for games because of their significant advantages. The recent great leaps in capabilities of graphics hardware have made many more image-based techniques useful for a wide range of games and other applications.

References

[Dana 99] Dana, K., Van Ginneken, B., Nayar, S., Koenderink, J., "Reflectance and Texture of Real-World Surfaces", ACM Transactions on Graphics, Vol. 18, No. 1, January 1999.

[Debevec 96] Paul Debevec, Camillo J Taylor & Jitendra Malik, "Modeling and Rendering Architecture from Photographs: A hybrid geometry and image based approach", Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, SIGGRAPH, August 1996.

[Hel-Or 03] Yacov Hel-Or, Tom Malzbender, Dan Gelb, "Synthesis of Reflectance Function Textures from Examples", Texture 2003 - 3rd International Workshop on Texture Analysis and Synthesis, Nice, France, October 17, 2003.

[Malzbender 01]Tom Malzbender, Dan Gelb, Hans Wolters, "Polynomial Texture Maps", Proceedings of the 28th annual conference on Computer graphics and interactive techniques, SIGGRAPH, August 2001.

[Rushmeier 97] Holly Rushmeier, Gabriel Taubin, Andre Gueziec, "Applying Shape from Lighting Variation to Bump Map Capture", Eurographics Rendering Workshop Proceedings, 1997

Code Listing 1

```
void vertexProgram_directional(float4 position    : POSITION,
                               float3 normal      : NORMAL,
                               float3 tangent     : COLOR,
                               float2 texCoord    : TEXCOORD0,

                               out float4 oPosition   : POSITION,
                               out float2 oTexCoord0  : TEXCOORD0,
                               out float2 oTexCoord1  : TEXCOORD1,
                               out float2 oTexCoord2  : TEXCOORD2,
                               out float1 oTexCoord3  : TEXCOORD3,
                               out float4 oUVVec      : COLOR,

                               uniform float3   objectSpaceLightPos,
                               uniform float4x4 modelViewProj,
                               uniform float4x4 modelViewIT)
{
   oPosition = mul(modelViewProj, position);

   // ABC, DEF, RGB textures use same texture coordinates
   oTexCoord0 = texCoord;
   oTexCoord1 = texCoord;
   oTexCoord2 = texCoord;

   // Compute PTM lu,lv  parameters

   // lu PTM component = light . tangent
   // Store u,1,v,u in output color channel

   oUVVec.xw = dot(objectSpaceLightPos, tangent);
   oUVVec.y = 1.0;

   // Compute the binormal from the tangent and normal
   float3 binormal;
   binormal = cross(normal, tangent);

   // compute lv
   oUVVec.z = dot(objectSpaceLightPos, binormal);

   // Convert range [-1.0..1.0] -> [0..1.0]
   oUVVec = 0.5 * oUVVec + 0.5;

   // Texture 3 used for light color/ and n dot l drop off
   // n dot l used as the texture coordinates
   // lz PTM component = light . normal

   oTexCoord3.x = dot(objectSpaceLightPos, normal);
   // Convert range [-1.0..1.0] -> [0..1.0]
   oTexCoord3.x = 0.5 * oTexCoord3.x + 0.5;

}
```

```
void fragmentProgramLRGB(float2 texCoord0    : TEXCOORD0,
                         float2 texCoord1    : TEXCOORD1,
                         float2 texCoord2    : TEXCOORD2,
                         float  texCoord3    : TEXCOORD3,
                         float4 uvVec        : COLOR,

                         out float4 color    : COLOR,

                         uniform sampler2D texAFBC       : TEXUNIT0,
                         uniform sampler2D texDFE        : TEXUNIT1,
                         uniform sampler2D texRGB        : TEXUNIT2,
                         uniform sampler1D texLightColor : TEXUNIT3)
{
   uvVec = expand(uvVec);

        // Calculate lu^2, 1, lv^2
   float3 uvVecSq = uvVec.xyz * uvVec.xyz;

   float3 polyEval;

        // The a5 coefficient (F) is stored as F/2 in both textures
        // This is to avoid range issues on older hardware

        // The expand and subsequent multiply by 2 gives scale of 4 and bias of 0.5
   float4 afbc = expand(tex2D(texAFBC, texCoord0).xyzw);
   float3 dfe = expand(tex2D(texDFE, texCoord1).xyz);

   // Do Au^2 + Bv^2 + Du + F + Ev
   polyEval = uvVecSq.xyz*afbc.xyz +  uvVec.xyz*dfe;

   float lum = polyEval.x + polyEval.y + polyEval.z + afbc.w*uvVec.x*uvVec.z;

   color = 2*lum.xxxx *tex2D(texRGB, texCoord2)*tex1D(texLightColor, texCoord3);
}
```