Chapter 1

Lighting and Material of *Halo 3*

Hao Chen[1]
Xinguo Liu[2]

Figure 1. *A still frame from* Halo 3 *showing the main character rendered with complex materials under global illumination.*

[1] haochen@bungie.com
[2] xgliu@cad.zju.edu.cn

1.1 Introduction

Lighting and material are very important aspects of the visual appearances of games and they present some of the hardest challenges in real time graphics today. For *Halo* and indeed many other games, keeping the players immersed in the virtual environment for long periods of time is a top priority of the graphics system, and good quality lighting and realistic materials are the fundamental building blocks for achieving the level of realism necessary to accomplish this goal.

Figure 2. *A still frame from* Halo 3 *in diffuse only mode that demonstrates the effect of global illumination.*

Except in special cases such as in a highly stylized environment, a good lighting scheme must be able to capture some form of global illumination, as can be seen here in Figure 2. Rendering global illumination involves solving the rendering equation first proposed in [IGC86][KAJIYA86]. If we assume that a surface is not self-emissive, then the rendering equation can be written as:

$$I(V) = \int_{\Omega} f(V, L)\cos(\theta)\ell(\omega)d\omega \quad , \qquad (1)$$

The equation states that the total reflected light energy I from a surface point along a viewing direction V is the product of the BRDF f, the incoming light ℓ, and the cosine of the angle θ between the view and the light direction (*V* and *L*, respectively), integrated over the hemisphere defined by the surface normal (See Table 1 for a list of symbols and their definitions used throughout the chapter).

Solving the equation directly is generally not feasible for a real time game, except for special cases such as when the lighting environment is made of only a small number of point light sources and when we only consider direct illumination.

Figure 3. *An outdoor scene made of different materials.*

Rendering complex materials under global illumination presents further challenges. A typical scene in *Halo 3* is made of many different materials, from dull concrete to shiny metal, as seen in Figure 3. Although simple BRDF models such as Phong have been used extensively in real time games, they are only trivial to evaluate interactively if the light source are point lights. For area light sources like the sky light in Figure 3, rendering the BRDF involves evaluating an expensive integral, as in Equation 1. The Phong model also does not capture important physical properties such as the Fresnel effect, which can contribute significantly to the realism of many materials.

For *Halo 3*, our goal is to be able to handle arbitrary light sources and to capture important global illumination effects, such as the soft lighting bounced from the floor towards the tree trunk as seen in Figure 2. We also want to render a large variety of materials under global and environment lighting, using a more physically accurate BRDF model.

In this chapter we describe the core lighting and material models used in *Halo 3*. We first describe the Cook Torrance BRDF model. Then we show how Equation 1 can be factored into separate components, which can each be rendered in real time using different techniques, and how the parts are combined together in the end for final shading. The shader code and further details are listed in the Appendix.

V	unit vector in the viewer direction
L	unit vector in the lighting direction
H	halfway vector: unit vector in $V + L$
m	material roughness parameter
k_d	diffuse reflectance coefficient
k_s	specular reflectance coefficient
R_d	diffuse reflectance
R_m	roughness function
F_0	Fresnel value at normal incidence angle
F	Fresnel term
G	geometrical attenuation factor in Cook-Torrance model
D	micro-facet distribution function in Cook-Torrance model

Table 1. *Symbols used in this work*

1.2 The Cook Torrance BRDF

The Cook-Torrance BRDF, first introduced in [COOKTORRANCE81], is based on the micro-facet theory, and is found to be more accurate than Blinn-Phong [BLINN77] model commonly used in games [NDM05]. We pick this model because it offers a good trade-off between BRDF expressiveness and computational complexity. Our methods can be extended to other models (See Appendix for an implementation of the Phong model). The Cook Torrance reflectance model is as follows:

$$f(V,L) = k_d R_d + k_s F R_m(V,L), \qquad (2)$$

where the first term is the diffuse component, which is dependent on the incoming light only, and the second term is the specular component, which is dependent on both the light and the viewing directions. Plugging the BRDF into the rendering equation in (1), we have:

$$I(V) = k_d R_d \int \cos(\theta)\ell(\omega)d\omega + k_s \int F R_m(V,L)\cos(\theta)\ell(\omega)d\omega \qquad (3)$$

Both the diffuse and specular components in Equation 3 involve an integral that is expensive to compute directly in real time. However, if the light sources are point lights only, the integral becomes a sum of ($N \cdot L$) terms for diffuse illumination. The specular reflectance can be evaluated in a similar fashion by evaluating the specular lobe directly for each point light and then summing. Games have used these simple forms of lighting and material models extensively, some to great effect. However, the restriction of point light sources (and direct illumination only) is one of the main reasons that many games

look unrealistic. To handle arbitrary light sources we need to find a way to evaluate the integrals in equation 3 efficiently, and for that we turn to spherical harmonics.

1.3 Spherical Harmonics Light Maps and Diffuse Reflectance

Spherical harmonics (SH) basis over the sphere is analogous to the Fourier basis over the line or the circle. It can be shown that for Lambertian surfaces, a small number of spherical harmonics terms are sufficient to approximate the original lighting to high accuracy [BRASRIJACOBS03]. The *Spherical Harmonics Irradiance Environment Maps*, introduced in [RAMAMOORTHIHANRAHAN01], encode the irradiance distribution function as a vector of SH coefficients. The un-shadowed diffuse transfer can then be computed in a shader using a quadratic polynomial approximation. For shadowed transfer and inter-reflections, the *Pre-computed Radiance Transfer* (PRT) method [SKS02] pre-computes the transfer function as a SH vector, which is then combined with incoming lighting using a SH dot product. Glossy materials are also possible in the SH representation [KSS02][RAMAMOORTHIHANRAHAN02][SHHS03], although none of the previous methods can meet the stringent performance and storage requirements of a real time game. We chose spherical harmonics basis for our lighting and material methods as well for the following reasons: SH is suitable for approximating smooth changing signals using a small number of coefficients which makes it ideal for lighting, transfer and material purposes; SH can be rotated easily in a shader; Finally, there are a number of algorithms such as PRT that are compatible with the SH representation which we use in our game.

The incoming lighting incident upon a point can be projected into spherical harmonics as follows:

$$\lambda_i = \int_{\Omega} \ell(\omega) Y_i(\omega) d\omega \qquad (4)$$

where Y_i are the spherical harmonics: Y_{00}, $Y_{1,-1}$, $Y_{1,0}$, $Y_{1,-1}$, $Y_{2,-2}$, $Y_{2,-1}$, $Y_{2,0}$, , $Y_{2,1}$, Y_{22}. Using SH basis, the diffuse reflectance in Equation 3 is simply:

$$I_d(V) = k_d R_d \int_{\Omega} \cos(\theta) \ell(\omega) d\omega = k_d R_d \sum_{i=0,2,6} \lambda_i A_i \qquad (5)$$

Here R_d is the Lambertian BRDF which is a constant, and A_i is the projection of the cosine lobe in SH. Since the cosine lobe is radially symmetric around the normal, the coefficients of its SH projection are non-zero only for the i = 0, 2, 6, … basis. The first three terms are given as (from [RAMAMOORTHIHANRAHAN01B]):

$$A_0 = \sqrt{\pi / 4}, \; A_1 = \sqrt{\pi / 3}, \; A_2 = \sqrt{5\pi / 64}$$

Notice equation 5 assumes that the incoming light is rotated into the local coordinate frame of the surface point.

Equation 4 encodes the incident radiance at a single point. This is insufficient for our purpose of lighting the whole scene since the incident radiance is spatially varying from point to point. A commonly used strategy to approximate such spatially varying functions is to sample the function at discrete sampling points, and then interpolate between the samples everywhere. There are several possible strategies we can use.

One such strategy is to spatially divide the scene into cells using a regular 3D grid, then store one or more samples per cell. This is the original scheme used in [GSHG98].

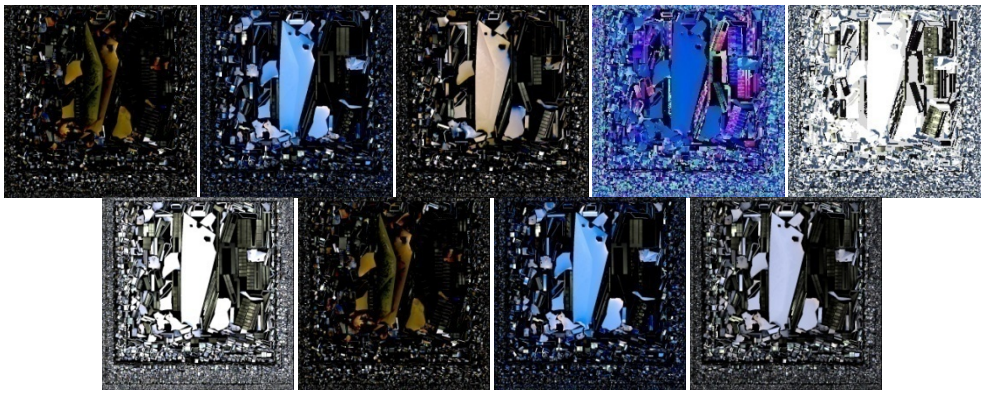

Figure 4. *Harmonics lightmap textures using quadratic SH.*

In the ATI *Ruby: Dangerous Curves* demo [OAT05], an adaptive subdivision method is proposed to reduce the number of samples needed while preserving important details.

Figure 5. *Hard shadow edges and bump mapping are hard problems for schemes based on discrete irradiance volumes.*

The ATI demo also uses spherical harmonics gradients to improve the interpolation between samples. The spatial subdivision scheme is well suited for rendering small dynamic objects, since the whole object can be rendered from one or a few lighting samples, obtained by interpolating between the samples closest to the object's location. For large static environment geometry however, the spatial subdivision scheme has several hard-to-overcome problems as shown here in Figure 5. First, rendering bump maps from such a scheme would require associating each pixel with one or more samples closest to the pixel location, and this is non-trivial and expensive to do in real time. Furthermore, dense samples must be stored along a shadow boundary, in order to preserve the appearance of sharp shadows, and this would require very final levels of subdivisions.

In *Halo 3*, we represent the lighting as spherical harmonics light maps, as shown in Figure 4. They can be viewed as analogous to traditional light maps, but instead of

storing the exit radiance as a single color, each pixel stores the incident radiance at the surface point, encoded in SH as in Equation 4. A similar representation is used in [GOODTAYLOR05] as an optimization for photon mapping.

SH light maps are textures parameterized over the surface which can be mapped to the geometry in real time. These textures can be generated using any offline global illumination solver. We use a distributed photon mapping algorithm running on a multi-node rendering farm. See [CHEN08][VILLEGASSEAN08] for further details.

Rendering from SH light maps is straightforward. A key benefit of the SH light maps is that bump maps can now be rendered directly. We now show a couple of examples of the diffuse reflectance of an environment rendered directly from the SH light map representation.

Figure 6. *An Outdoor scene from* Halo 3. *The sun and sky models are from [PSS99], they are the only light source in the scene. Global illumination is the key factor to the realism of this scene, so is rendering everything from a consistent lighting representation.*

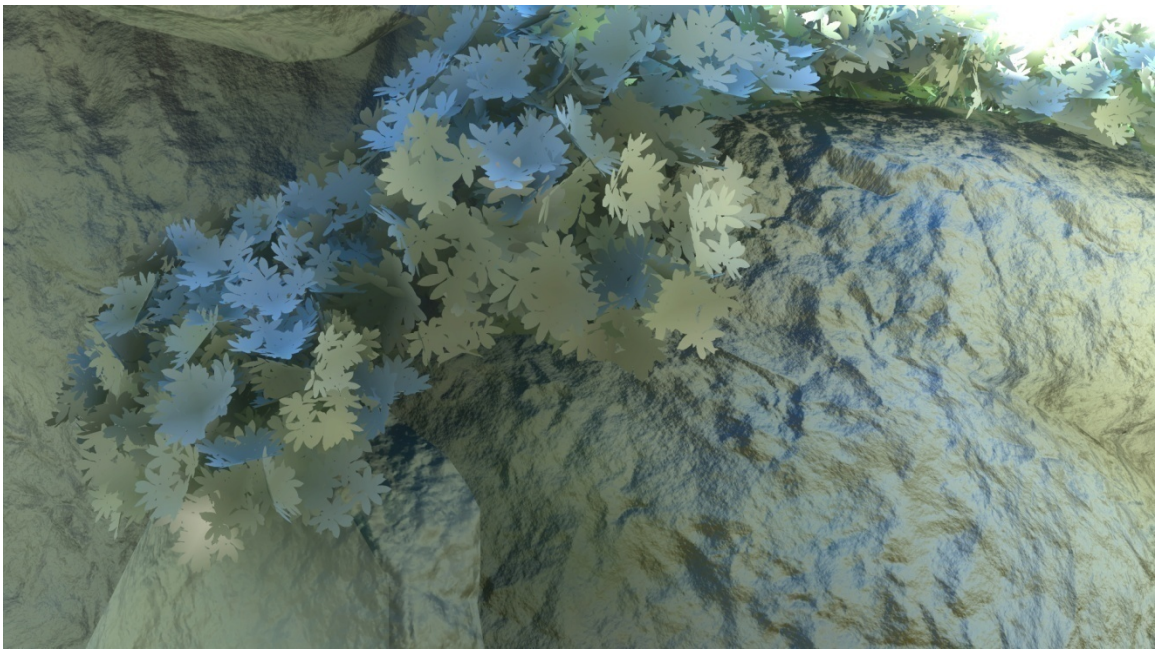

Figure 7. *Close-up view of a bump mapped surface. Notice the subtle shading of the bumped detail from reflected lighting below and the large, bluish sky light. This is very difficult to achieve with point lights, unless a large number of them are used.*

1.4 Specular Reflectance

Rendering glossy material under global and environment lighting is hard for real time games. There are several reasons for this. First, the specular reflectance is a view dependent quantity which makes the specular part in Equation 3 much more difficult to evaluate than the diffuse part. Second, a glossy material typically has both high and low frequency parts and everything in-between, but low order spherical harmonics is effectively a low pass filtered signal of the original, and thus ill suited for capturing high frequency glossy appearance.

Our method is to breakdown the all-frequency specular reflectance further into three separate layers, responsible for high, mid and low frequency reflectance respectively. These separate layers are then computed using a different technique that is appropriate for the type of appearance they are responsible for. Figure 8 shows renderings of the Master Chief in each layer separately and then with all layers combined. The details of the layers are described as follows below.

1.4.1 Analytical Specular

This layer is rendered by evaluating the BRDF model directly in shader from the directional light coming from the dominant light direction (See Appendix A for a HLSL

listing). This layer is responsible for capturing the sharp specular highlights as seen in Figure 8 (a). The specular lobe of the Cook Torrance model is the following:

$$R_m(V,L) = \frac{DG}{\pi(N \cdot L)(N \cdot V)}\ , \qquad (6)$$

where *D* is the micro-facet distribution function, and *G* is the geometry attenuation factor, and *L* is the light direction. Interested readers could find how the micro-facet distribution function and the geometry attenuation factor are defined in [COOKTORRANCE81]. For this layer, we only consider the light from the dominant light direction, which can be approximated by fitting a directional light to the quadratic SH light vector. For the direction vector, we use the optimal linear direction (the SH linear coefficients normalized). This direction is well behaved and smoothly varying. Alternatively, we can store a separate texture that encodes the dominant light direction. However, care must be taken to ensure that the directions are filterable since these textures are subject to various filtering operations in the texturing hardware. We can find the dominant light intensity by minimizing the squared error E between the SH light and the SH version of the "directional" light as follows:

$$E = \sum_{i=0,\ldots 8} (\lambda_i - cY_i(d))^2 .$$

Let the derivative with respect to d be zero, $E' = 0$, then the intensity of the directional light is given by

$$c = \sum_{i=0,\ldots 8} (\lambda_i Y_i(d)) / \sum_{i=0,\ldots 8} Y_i(d)^2$$

.

We can pre-compute and store the intensity c as a separate HDR texture.

1.4.2 Environment Map

For the middle frequency glossy reflectance, we use environment cube-maps that are pre-generated at discrete locations throughout the scene. A rendering of the Master Chief with environment maps only can be seen in Figure 8 (b). Since the analytical specular already handles the sharpest highlights, we render the environment maps with the high frequency filtered out. This also allows us to use smaller cube-map sizes (128 × 128 × 6). Unlike analytical specular, the environment map can capture reflectance from the whole lighting environment.

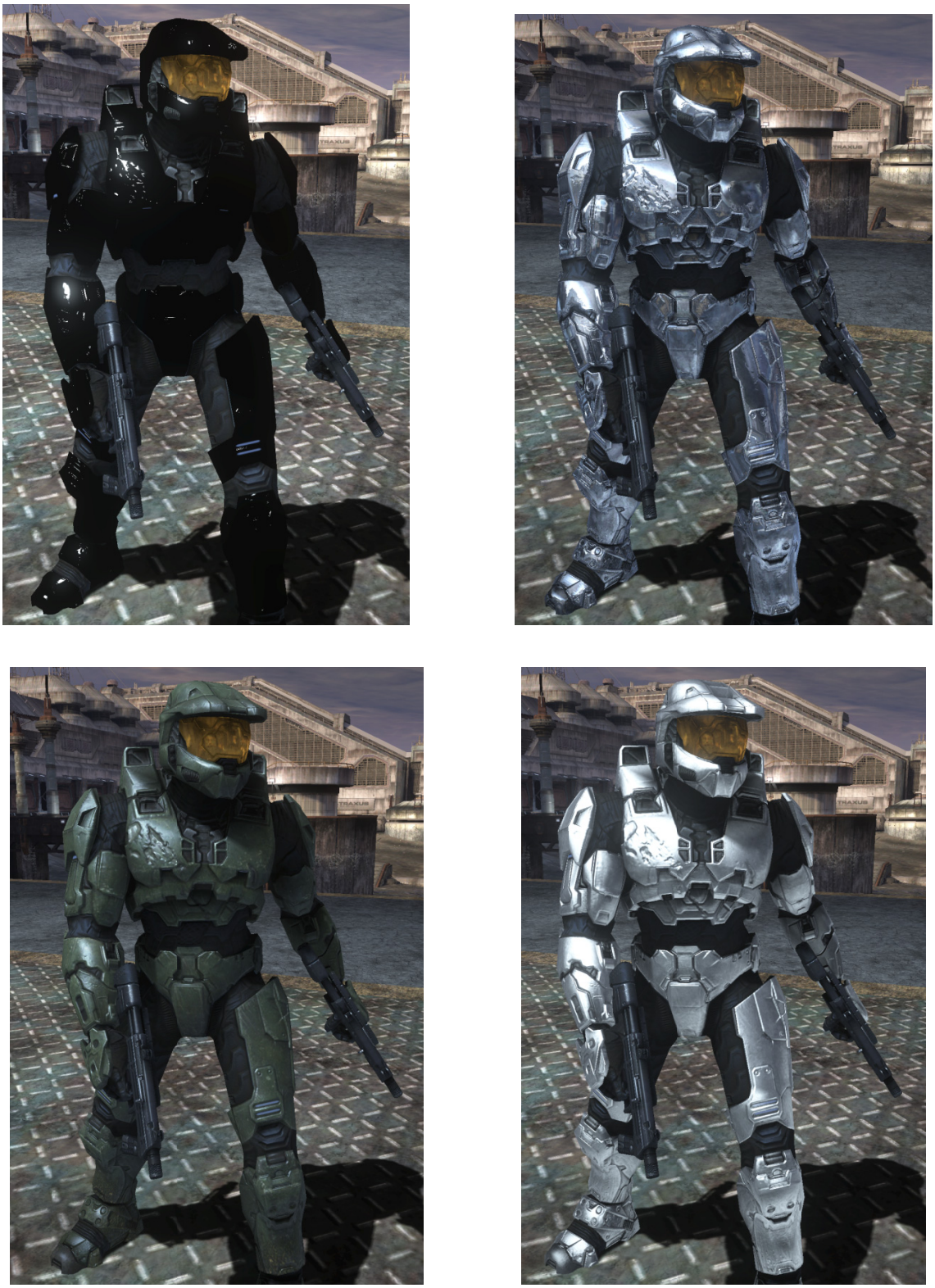

Figure 8. *Rendering of the* Master Chief *showing separate specular layers. Clockwise from top left: (a) Analytical specular only. (b) Environment map only. (c) Area specular only. (d) All combined + diffuse.*

Since we are storing the cube-maps at discrete locations instead of per-pixel on the surface, the environment map sampling suffers from the same limitation mentioned earlier, i.e. the inability to handle sharp transitions along the shadow boundaries. To overcome this, we pre-divide the environment maps by the area specular (described below), and in the shader, we multiply back in the area specular. This simple scheme allows us to darken or brighten the environment maps in and out of shadows.

1.4.3 Area Specular

We call the low frequency layer of the glossy reflectance the area specular term. This can be seen in Figure 8 (c). We developed a novel method to parameterize the Cook Torrance BRDF model in SH, which can then be rendered directly from the SH light map representation or any arbitrary lighting that can be represented in the SH basis.

Our method differs from others in that we parameterize the whole BRDF model instead of each material. Therefore we can freely change any parameters of the BRDF model in real time and generate different materials. Spatially varying BRDF can be achieved by storing these parameters in a texture. Our model requires only 3K of storage and can be computed efficiently in real time. Our method can be extended to other BRDF models, and the Phong BRDF is shown in the Appendix as an example.

1.4.3.1 *Light Integration*

Recall equation 3, the specular component of the Cook Torrance BRDF is:

$$I_s(V) = k_s \oint FR_m(V,L)\cos(\theta)\ell(\omega)d\omega$$

If we project both the BRDF and the cosine term in SH, then by the convolution theorem the integral becomes a SH dot product. We define the SH projection of the BRDF as $B_{m,i}(V)$. We also divide the Fresnel term by F_0 whose purpose will be explained later.

$$B_{m,i}(V) = \oint \frac{F}{F_0} R_m(V,L)\cos(\theta)Y_i(\omega)d\omega \quad (7)$$

$$I_s(V) = k_s F_0 \sum_{i=0}^{8} \lambda_i B_{m,i}(V) \quad (8)$$

1.4.3.2 *Fresnel Approximation*

We first introduce two terms for convenience:

$$C_{m,i}(V) = \oint R_m(V,L)\cos(\theta)Y_i(\omega)d\omega$$
$$D_{m,i}(V) = \oint \left(1-(L\cdot H)^5\right)R_m(V,L)\cos(\theta)Y_i(\omega)d\omega$$

The first term denotes the integral of the specular reflection function with the SH basis functions, while the second term is defined based on the Fresnel approximation method by Schlick [SCHLICK94].

For constant Fresnel, $F \approx F_0$, then $B_{m,i}(V) = C_{m,i}(V)$, and the Equation 8 becomes:

$$I_s(V) = k_s F_0 \sum_{i=0}^{8} \lambda_i C_{m,i}(V)$$

For non-constant Fresnel, we can use the Schlick approximation [SCHLICK94]:

$$F \approx F_0 + (1-F_0)\left(1-(L\cdot H)^5\right).$$

Then

$$B_{m,i}(V) = C_{m,i}(V) + \tfrac{1-F_0}{F_0} D_{m,i}(V),$$

and

$$I_s(V) = k_s F_0 \sum_{i=0}^{8} \lambda_i \left(C_{m,i}(V) + \tfrac{1-F_0}{F_0} D_{m,i}(V)\right)$$

In the following we show how $C_{m,i}(V)$ and $D_{m,i}(V)$ can be pre-integrated offline.

1.4.3.3 *Pre-Integration*

By the isotropic property of the Cook Torrance reflectance model, we can always integrate the shading result of Equation 3 in a local frame such that the view direction *V* lies in the *X-Z* plane of the local frame. Therefore we only need to pre-integrate $C_{m,i}(V)$ and $D_{m,i}(V)$ for some view directions in the *X-Z* plane.

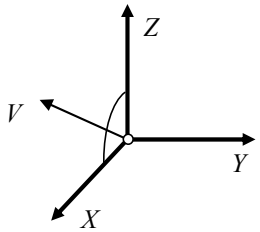

Figure 9. *The local coordinate frame where Z points along the normal, and V lies in the X-Z plane.*

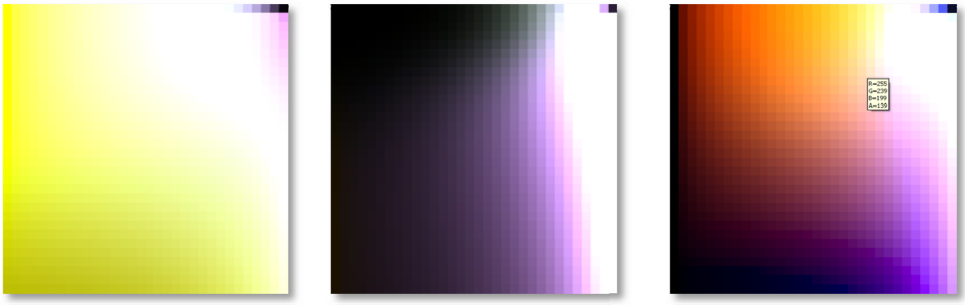

Figure 10. *Pre-integrated C and D textures, from left to right, C(0,2,3,6), D(0,2,3,6)and CD(7,8). Horizontal axis represents viewing changes and the vertical axis roughness changes.*

By the reflective symmetry property of the Cook Torrance reflectance model, we know that $C_{m,i}(V)$ and $D_{m,i}(V)$ are all zero for $i = 1,4,5$, and the variation in the textures are very smooth in the direction of both *m* and *V*. Therefore, we pre-integrate $C_{m,i}(V)$ and $D_{m,i}(V)$ for 16 roughness (m) values in (0, 1) and 8 viewing directions in the *X-Z* plane, yielding 12 2*D* textures. The texture values are all in (-2, 2), so we quantize them by 16 bits, or store them in 16-bits *float* format. The total storage is about 16*8*6*2*2 = 3 KB.

Figure 3 shows the integrated C and D textures. With these lookup tables, we can now render any Cook Torrance BRDF in real time and the only storage required is the parameters themselves.

To render area specular, we first construct a local frame as in Figure 9. The lighting SH vector is then rotated into this local frame. To compute the glossy reflectance, we look up the pre-integrated C and D textures, which we then integrate with the lighting through an SH dot product. The HLSL code is listed in the appendix.

1.5 Conclusion

In this chapter we have described the core lighting and material models used in Halo 3. The SH light map representation is a natural extension to an existing light mapping pipeline. By storing incident radiance as a SH vector, instead of exit radiance as a color, we can capture sharp shadows just like light maps, and bump maps can be rendered directly from it. SH light maps require much larger storage than traditional light maps, however, lighting data can be heavily compressed with minimal loss of perceptible quality ([HuWang08] describes our multi-stage compression algorithm for SH light maps).

By separating the all frequency reflectance into layers, each of which can then be rendered using different real time techniques, we can preserve the realism of a real word material while keeping the computational and storage costs to manageable levels for a real time game.

We introduced a novel way of parameterizing the Cook Torrance BRDF model in spherical harmonic basis, which can then be rendered under global and environment lighting or any lighting that can be represented in SH. Using small 2D textures as look up tables, all parameters of the BRDF models are free parameters, and we can render any Cook Torrance material efficiently and without per material storage other than the parameters themselves. Spatially varying materials are also possible by putting the parameters in a texture.

Finally, we show that advanced lighting and material models are feasible in the current generation of real time hardware, and with the projected increase of computational power of future generations, lighting and material improvements in games will continue and will result in significant improvements in the realism of games.

1.6 Acknowledgements

The Halo 3 graphics engine was developed by a team of dedicated engineers and researchers, they are:

Bungie Graphics Team: Hao Chen, Ben Wallace, Chris Tchou, David Cook, Xi Wang. And Microsoft Research Asia: Xinguo Liu, Yaohua Hu, Zhipeng Hu, Kun Zhou, Minmin Gong.

The authors would like to thank the following people for their contribution to the Halo 3 graphics engine: Peter Pike Sloan, Baining Guo, Harry Shum, Kutta Srinivasan, Matt Lee, Mikey Wetzel.

1.7 References:

[BASRIJACOBS03] BASRI, R., AND JACOBS, D. W. 2003. Lambertian reflectance and linear subspaces. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 2, pp. 218–233.

[BLINN77] BLINN, J. F. 1977. Models of light reflection for computer synthesized pictures. *ACM SIGGRAPH Comput. Graph. 11*, 2, pp. 192–198.

[CHEN08] CHEN, H. Lighting and materials of *Halo 3*. Game Developers Conference, 2008.

[COOKTORRANCE81] COOK, R. L., AND TORRANCE, K. E. 1981. A reflectance model for computer graphics. In *Proceedings of ACM SIGGRAPH 1981*, pp. 307–316.

[GOODTAYLOR05] GOOD, O., AND TAYLOR, Z. 2005. Optimized photon tracing using spherical harmonic light maps. In *Proceedings of ACM SIGGRAPH 2005, Technical Sketches*, p. 53.

[GSHG98] GREGER, G., SHIRLEY, P., HUBBARD, P. M., AND GREENBERG, D. P. 1998. The irradiance volume. *IEEE Comput. Graph. Appl. 18*, 2, pp. 32–43.

[HUWANG08] HU, Y., AND WANG, X. Lightmap compression in Halo 3. Game Developers Conference, 2008.

[ICG86] IMMEL, D. S., COHEN, M. F., AND GREENBERG, D. P. 1986. A radiosity method for non-diffuse environments. *ACM SIGGRAPH Comput. Graph. 20*, 4, pp. 133–142.

[KAJIYA86] KAJIYA, J. T. 1986. The rendering equation. In *Proceedings of ACM SIGGRAPH 1986*, pp. 143–150.

[KSS02] KAUTZ, J., SLOAN, P.-P., AND SNYDER, J. 2002. Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In *Proceedings of the 13th Eurographics workshop on Rendering 2002*, pp. 291–296.

[NDM05] NGAN, A., DURAND, F., AND MATUSIK, W. 2005. Experimental analysis of brdf models. In *Proceedings of the Eurographics Symposium on Rendering 2005*, pp. 117–226.

[OAT05] OAT, C. Irradiance Volumes for Games, Game Developers Conference, 2005. http://ati.amd.com/developer/gdc/GDC2005_PracticalPRT.pdf

[PSS99] PREETHAM, A.J., SHIRLEY, P. AND SMITS, B. 1999. A Practical Analytic Model for Daylight, In Proceedings of Siggraph 1999, pp. 91 – 100, Los Angeles, CA.

[RAMAMOORTHIHANRAHAN01] RAMAMOORTHI, R., AND HANRAHAN, P. 2001. An efficient representation for irradiance environment maps. In *Proceedings of ACM SIGGRAPH 2001*, pp. 497–500.

[RAMAMOORTHIHANRAHAN01B] RAMAMOORTHI, R., AND HANRAHAN, P. 2001. On the relationship between radiance and irradiance: Determining the illumination from images of a convex Lambertian object. *Journal of the Optical Society of America, Vol. 18*, 10, pp. 2448–2459.

[RAMAMOORTHIHANRAHAN02] RAMAMOORTHI, R., AND HANRAHAN, P. 2002. Frequency space environment map rendering. In *Proceedings of ACM SIGGRAPH 2002*, 517–526.

[SCHLICK94] SCHLICK, C. 1994. An inexpensive BRDF model for physically-based rendering. *Computer Graphics Forums. 13*, (3), 233–246.

[SLOANSNYDER02] SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low frequency lighting environments. *ACM Trans. Graph. 21*, 3, 527–536.

[SHHS03] SLOAN, P.-P., HALL, J., HART, J., AND SNYDER, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22*, 3, 382–391.

[VILLEGASSEAN08] VILLEGAS, L., AND SEAN S. Life on the Bungie Farm: Fun Things to Do with 180 Servers . Game Developers Conference, 2008.

Appendix A. Shader Code Listings:

```
float3 diffuse_reflectance(float3 normal, float4 lighting_constants[10])
{
  float c1 = 0.429043f;
  float c2 = 0.511664f;
  float c4 = 0.886227f;
  float3 x1, x2, x3;

  //linear
  x1.r = dot( normal, lighting constants[1].rgb);
  x1.g = dot( normal, lighting constants[2].rgb);
  x1.b = dot( normal, lighting constants[3].rgb);

  //quadratic
  float3 a = normal.xyz*normal.yzx;
  x2.r = dot( a.xyz, lighting constants[4].rgb);
  x2.g = dot( a.xyz, lighting constants[5].rgb);
  x2.b = dot( a.xyz, lighting_constants[6].rgb);

  float4 b = float4( normal.xyz*normal.xyz, 1.f/3.f );
  x3.r = dot( b.xyzw, lighting constants[7].rgba);
  x3.g = dot( b.xyzw, lighting constants[8].rgba);
  x3.b = dot( b.xyzw, lighting constants[9].rgba);

  float3 lightprobe_color=
     c4 * lighting_constants[0] + (-2.f*c2) * x1 + (-2.f*c1)*x2 - c1 * x3;O)

  return lightprobe color/3.1415926535f;
}

void pack_constants(in float3 sh[9], out float4 lc[10])
{
  lc[0]= float4(sh[0], 0);
  lc[1]= float4(sh[3].r, sh[1].r, -sh[2].r, 0);
  lc[2]= float4(sh[3].g, sh[1].g, -sh[2].g, 0);
  lc[3]= float4(sh[3].b, sh[1].b, -sh[2].b, 0);
  lc[4]= float4(-sh[4].r,sh[5].r, sh[7].r, 0);
  lc[5]= float4(-sh[4].g,sh[5].g, sh[7].g, 0);
  lc[6]= float4(-sh[4].b,sh[5].b, sh[7].b, 0);
  lc[7]= float4(-sh[8].r, sh[8].r,-sh[6].r*1.7320508f,
                sh[6].r*1.7320508f);
  lc[8]= float4(-sh[8].g, sh[8].g,-sh[6].g*1.7320508f,
                sh[6].g*1.7320508f);
  lc[9]= float4(-sh[8].b, sh[8].b,-sh[6].b*1.7320508f,
                sh[6].b*1.7320508f);
}
```

Listing 1. *HLSL shader code for rendering diffuse reflectance from SH Light map. Notice a quadratic SH vector is pre-packed into 10* `float4` *constants.*

```
void calc_material_analytic_specular_cook_torrance_ps(
  in float3 view dir,
  in float3 normal dir,
  in float3 reflect dir,
  in float3 light dir,
  in float3 light_intensity,
  in float3 c_fresnel_f0,
  in float  c toughness,
  out float3 analytic specular)
{
  float n dot l = dot( normal dir, light dir );
  float n_dot_v = dot( normal_dir, view_dir );
  float min_dot = min( n_dot_l, n_dot_v );
  if ( min dot > 0)
  {
    // geometric attenuation
    float3 half vector = normalize( view dir + light dir );
    float n_dot_h = dot( normal_dir, half_vector );
    float v_dot_h = dot( view_dir, half_vector);
    float G = 2 * n dot h * min dot / (saturate(v dot h));

    //calculate fresnel term
    float3 f0= c fresnel f0;
    float3 sqrt_f0 = sqrt( f0 );
    float3 n = ( 1.f + sqrt_f0 )/( 1.0 - sqrt_f0 );
    float3 g = sqrt( n*n + v dot h*v dot h - 1.f );
    float3 gpc = g + v dot h;
    float3 gmc = g - v dot h;
    float3 r =(v dot h*gpc-1.f)/(v dot h*gmc+1.f);
    float3 F= (0.5f*( (gmc*gmc)/(gpc*gpc+0.00001f))*( 1.f+r*r));

    //calculate the distribution term
    float t roughness= c toughness;
    float m squared= t roughness*t roughness;
    float cosine alpha squared = n dot h * n dot h;
    float D;
    D= exp((cosine_alpha_squared-1)/
           (m squared*cosine alpha squared))/
           (m squared*cosine alpha squared*cosine alpha squared);

    //puting it all together
    analytic_specular= D*saturate(G)/(3.14159265 * n_dot_v)*F;
    analytic_specular*= light_intensity;
  }
  else
  {
    analytic specular= 0.0f;
  }
}
```

Listing 2. *Cook Torrance BRDF evaluated directly in shader from a point light source for the analytical specular. The dominant light is a directional light fitted from the quadratic SH coefficients.*

```
void area_specular_cook_torrance(
  in float3 view_dir,
  in float3 rotate z,
  in float4 sh 0,
  in float4 sh 312[3],
  in float4 sh 457[3],
  in float4 sh_8866[3],
  in float roughness,
  in float r dot l,
  out float3 area specular)
{
  float3 specular part;
  float3 schlick_part;
  //build the local frame
  float3 rotate x= normalize(view dir-dot(view dir,rotate z)*rotate z);
  float3 rotate y= cross(rotate z, rotate x);

  //calculate the texure coord for lookup
  float2 view_lookup= float2(dot(view_dir,rotate_x), roughness);

  // bases: 0,2,3,6
  float4 c value= tex2D(g sampler cc0236, view lookup);
  float4 d value= tex2D(g sampler dd0236, view lookup);

  //rotate lighting basis 0,2,3,6 into local frame
  float4 quadratic_a, quadratic_b, sh_local;
  quadratic a.xyz= rotate z.yzx * rotate z.xyz *(-SQRT3);
  quadratic b= float4(rotate z.xyz*rotate z.xyz,1.0f/3.0f)*0.5f*(-SQRT3);

  //red
  sh_local.xyz= sh_rotate_023( 0,
                               rotate_x,
                               rotate z,
                               sh 0,
                               sh 312);
  sh local.w= dot(quadratic a.xyz,
         sh_457[0].xyz)+dot(quadratic_b.xyzw, sh_8866[0].xyzw);

  //dot with C and D look up
  sh local*= float4(1.0f, r dot l, r dot l, r dot l);
  specular part.r= dot(c value, sh local );
  schlick_part.r= dot(d_value, sh_local );

  //repeat for green and blue
      …
```

Listing 3 Part 1. *Shader code for area specular calculation, continues next page*

```
  // basis - 7
  c value= tex2D( g sampler c78d78, view lookup ).SWIZZLE;
  quadratic a.xyz = rotate x.xyz * rotate z.yzx + rotate x.yzx *
                    rotate z.xyz;
  quadratic b.xyz = rotate x.xyz * rotate z.xyz;
  sh_local.rgb= float3(dot(quadratic_a.xyz, sh_457[0].xyz) +
                       dot(quadratic_b.xyz, sh_8866[0].xyz),
                       dot(quadratic a.xyz,sh 457[1].xyz) +
                       dot(quadratic b.xyz, sh 8866[1].xyz),
                       dot(quadratic a.xyz, sh 457[2].xyz) +
                       dot(quadratic b.xyz, sh 8866[2].xyz));


  sh local*= r dot l;
  //c7 * L7
  specular part.rgb+= c value.x*sh local.rgb;
  //d7 * L7
  schlick_part.rgb+= c_value.z*sh_local.rgb;

  //basis - 8
  quadratic a.xyz = rotate x.xyz * rotate x.yzx - rotate y.yzx *
                    rotate y.xyz;
  quadratic b.xyz = 0.5f*(rotate x.xyz * rotate x.xyz - rotate y.xyz *
                    rotate_y.xyz);
  sh_local.rgb= float3(-dot(quadratic_a.xyz, sh_457[0].xyz) -
                        dot(quadratic b.xyz, sh 8866[0].xyz),
                       -dot(quadratic a.xyz, sh 457[1].xyz) -
                        dot(quadratic b.xyz, sh 8866[1].xyz),
                       -dot(quadratic a.xyz, sh 457[2].xyz) -
                        dot(quadratic_b.xyz, sh_8866[2].xyz));
  sh_local*= r_dot_l;

  //c8 * L8
  specular part.rgb+= c value.y*sh local.rgb;
  //d8 * L8
  schlick_part.rgb+= c_value.w*sh_local.rgb;
  schlick_part= schlick_part * 0.01f;

  area specular= specular part*k f0 + (1 - k f0)*schlick part;

}

float3 sh_rotate_023(int irgb, float3 rotate_x, float3 rotate_z,
                     float4 sh 0,
                     float4 sh 312[3])
{
  float3 result = float3( sh 0[irgb],
                         -dot(rotate_z.xyz, sh_312[irgb].xyz),
                          dot(rotate_x.xyz, sh_312[irgb].xyz));

  return result;
}
```

Listing 3 Part 2. *Shader code for area specular calculation*

Appendix B. Phong BRDF

Let $\hat{V}$ be the reflection direction of the viewing direction around the surface normal. We define a glossy reflection distribution function as follows:

$$R_m(V,L) = \frac{1}{\cos\theta_V}\frac{1}{\pi m^2 \cos^3\alpha}\exp\left\{-\frac{\tan^2\alpha}{m^2}\right\},$$

where α is the angle between the reflection direction $\hat{V}$ and the incident light direction L. Assume that the effective incoming light has almost the same incident angle as θ_V, the total energy is conserved, since

$$\oint \frac{1}{\pi m^2 \cos^3 \alpha} \exp\left\{-\frac{\tan^2 \alpha}{m^2}\right\} d\omega = 1.$$

Under an environmental lighting $L(\omega)$, the illumination result can be computed by:

$$I(V) = \oint \frac{1}{\cos \theta_V} \frac{1}{\pi m^2 \cos^3 \alpha} \exp\left\{-\frac{\tan^2 \alpha}{m^2}\right\} \ell(\omega) \cos(\theta) d\omega$$

Again assume that the effective incoming light has almost the same incident angle as θ_V (i.e. the material is high glossy), we have

$$I(V) = \oint \frac{1}{\cos \theta_V} \frac{1}{\pi m^2 \cos^3 \alpha} \exp\left\{-\frac{\tan^2 \alpha}{m^2}\right\} \ell(\omega) \cos(\theta_V) d\omega$$

$$= \oint \frac{1}{\pi m^2 \cos^3 \alpha} \exp\left\{-\frac{\tan^2 \alpha}{m^2}\right\} \ell(\omega) d\omega = \sum_i \lambda_i E_{m,i}(V)$$

where

$$E_{m,i}(V) = \oint \frac{1}{\pi m^2 \cos^3 \alpha} \exp\left\{-\frac{\tan^2 \alpha}{m^2}\right\} Y_i(\omega) d\omega$$

(note that angle α has dependence on V.)

Appendix C. Additional Screen Captures.