HyperMan: Seamless integration of Maya® & RenderMan®

Avinash Sunnasy

National Centre for Computer Animation at Bournemouth University

ABSTRACT

In this paper we present a method for combing the modelling system Maya with the rendering system RenderMan, to provide users with a functional, flexible modelling rendering package that brings the user the advanced features of RenderMan without having to learn anything new.
We propose a method for converting scene lighting and shading models into RenderMan shading language that is then parsed to the interpreter at render time.

1 Introduction

RenderMan is used in many production houses and continues to produce work and imagery of a notable standard, so much so that it has won an Academy award. Many hobbyists and an increasing number of students are looking to produce work of an industry standard. We find many in this category are intimidated by the idea of learning something completely different. HyperMan provides users with a solution; it is designed with ease of use in mind, whilst retaining the important features of RenderMan; speed, stability and quality. With HyperMan there is nothing new to learn, simply press render to see the awesome results that RenderMan produces.
In section one we discuss the mathematically principals behind the lighting and shading techniques which are used in the HyperMan system.
Section two describes the shading language structure used in the HyperMan system. The implementation principles are explored in section three. We demonstrate the capabilities of HyperMan in section four. Finally section 5 contains the conclusion.

Email b1421960@bmth.ac.uk

1.1 The BRDF

The Bi-directional Reflectance Distribution Function (BRDF) gives the reflectance of a target as a function of illumination. This depends on the wavelength and is determined by the structural and optical properties of the surface. Properties of a surface include shadow casting, multiple scattering, mutual shadowing, transmission, reflection, absorption and emission by surface elements, facet orientation distribution and facet density.

It should be noted that the BRDF describes what we all observe every day, i.e. that objects look differently when view from different directions and illuminated from different directions. In computer graphics today the BRDF needs to be taken into account, to produce visually aesthetic work. However, we often find that the BRDF is changed in such a way that it is no longer a physically accurate representation, but more a method to create images with the appropriate emphasis and impact. This is often called non-physical optics and lighting, almost trickery that provides an approximate "trick." Jim Blinn once called these tricks "The ancient art of Chi-Ting" [1].

> "In trying to improve the quality of the synthetic images, we do not expect to be able to display the object exactly as it would appear in reality, with texture, overcast shadows, etc. We hope only to display an image that approximates the real object closely enough to provide a certain degree of realism."[2]

[1] (Blinn, 1985)
[2] (Phong, 1975)

1.2 Lighting & Shading Models

Often lighting and shading is confused. By lighting we mean the interaction between materials and light sources and shading designates the colour of the surface point.

1.2.1 Ambient Light Reflection

Ambient light comes from a variety of light source it is reflected off various surfaces and eventually reaches the surface of interest. To simplify the model we make the assumption that light comes equally from all directions, this means that the ambient light has a constant contribution and does not vary from different viewing directions. The model for ambient light reflection is expressed as follows. [3]

$$Ia = La \,.\, Ka$$

Where
Ia is the intensity of the reflected ambient light,
La is the intensity of the incident ambient light, i.e. the ambient light reaching the surface and
Ka is the coefficient of ambient reflection (i.e. the fraction of ambient light that is reflected off the surface).

When lighting with ambient lights we get a flat, constant appearance. The standard RenderMan function "ambient()" contains no illuminance loop i.e. it does not need to reference the lights in the scene and the light direction vector L is set to zero, this indicated to the renderer that there is no directionality to the light.

1.2.2 Diffuse Light Reflection

Surfaces that appear matte are composed of randomly distributed micro-facets. When parallel rays hit a surface they are reflected in a random fashion and so scattering in all directions. These surfaces are equally bright when viewed from any direction this means that the viewing angle is insignificant. This is known as a diffuse or Lambertian reflection.

[3] Cominos (2003, pg175).

"Lamberts cosine law states that the reflected or transmitted *luminous intensity* in any direction from an element of a perfectly diffusing surface varies as the cosine of the angle between that direction and the *normal vector* of the surface. As a consequence, the *luminance* of that surface is the same regardless of the viewing angle. " [4]

The model for diffuse light reflection can be expressed as follows

$$Id = Lp \,.\, kd \,.\, cos(i) = Lp \,.\, kd \,.\, (\boldsymbol{L \bullet N})$$

Where,
Id is the intensity of the reflected diffuse light,
Lp is the intensity of the light of a point source incident on the surface,
L is the unit vector in the direction of the light source,
N is the unit normal vector, and
Kd is the coefficient of the diffuse reflection (i.e. the fraction of light that is reflected diffusely off the surface). Cominos (2003, pg176).

The standard RenderMan "diffuse()" call expands as follows.

```
color
diffuse( normal N )
{
        color C = 0;
        illuminance( P, N, PI/2 )
            C += Cl *
normalize(L).N;
        return C;
} [4]
```

Where N is the surface normal, P is the point on the surface and L is the light direction vector. It is clear that the diffuse model in RenderMan reflects the Lambert shading model, the "illuminance" statement simply loops through the lights in the scene.

[4] Mischler(2003, online)
[5] Pixar(2004, online)

1.2.3 Specular Light Reflection

Glossy surfaces all exhibit smooth surface finishes; examples are mirrors, and car paint. The angle of reflection is equal to the angle of incidence, these are called *r, I,* respectively. Glossy surfaces are never a perfect mirror thus highlights in a mirror-like surface will appear to fall-off. The halfway vector between the light source direction vector *L* and the viewing direction vector *E is H*. The reflection model for specular light reflection can be expressed as follows: [6]

$$Is = Lp \,.\, ks \,.\, Fs(\boldsymbol{N, L, E, n})$$

Where,
Is is the intensity of the reflected specular light,
Lp is the intensity of the light of a point source incident on the surface,
L is the unit vector in the direction of the light source,
E is the unit vector in the direction of the viewer,
N is surface unit normal vector,
n is the specular sharpness, i.e. the fraction of specular light that is reflected off the surface).
Fs is the speculat reflection function, and
Ks is the coefficient of specular reflection (i.e. the fraction of specular light that is reflected off the surface). Cominos (2003, pg177).

The specular reflection function Fs() can be calculated by Blinn's specular method.

1.2.3.1 Blinn Shading

Blinn's specular method was based on Horn's with furthermore simplification.
In Blinn's method the specular reflection function is given by $Fs(N, L, E, n) = \cos(g)^n$

Where ***N*** is known and ***H*** is the halfway vector between ***E*** and ***L***. So

$$\boldsymbol{H = (E + L) / |E + L|}$$

This means that the specular function can be computed as

$$\boldsymbol{Fs\ (N, L, E, n) = (N \cdot H)^n}$$

[6] Cominos (2003, pg177).

The standard RenderMan "specular()" call is as follows

```
color
specular( normal N; vector V;
float roughness )
{
        color C = 0;
        illuminance( P, N, PI/2 )
            C += Cl *
specularbrdf(normalize(L), N, V,
roughness);
        return C;
}
```

Which expands furthermore to ,

```
color specularbrdf(vector L, N,
V; float roughness)
{
    vector H = normalize(L+V);
    return pow(max(0, N.H),
1/roughness);
}[7]
```

Again we find that the RenderMan function implements Blinn's specular method, ***H*** is the halfway vector, ***L*** is the light direction vector, and ***N*** is the surface normal. However Maya uses the Torrance - Sparrow specular method that was described by Blinn is his 1978 paper. Torrance - Sparrow originally suggested the theoretically approach for calculating the specular model in 1967. This method attempts to provide a more physical model for specular reflections from real surfaces. It considers that intensity of specular highlights is dependent on the incident direction relative to normal. The Torrance & Sparrow method takes into account of micro-faceted surfaces instead of considering each surface to be completely smooth as with previous specular methods. As in the Lambertian model it was assumed that the diffuse component comes from multiple reflections between facets and from internal scattering. It is also assumed that the specular component comes from the facets orientation in the direction ***H*** (the half way vector).

[7] Pixar(2004, online)

The Torrance – Sparrow methods is as follows.

$I_S = DGF / (N \cdot V)$ [8]

D is the distribution function of the micro facet directions on the surface.
G is the amount that facets shadow and mask each other.
F is the Fresnel reflection law.
N is the surface normal.
V is the reflecting vector.
The distribution function ***D*** expands as follows, this is a simple Gaussian distribution function

$D = e^{-(sa)^2}$
a is the deviation angle from halfway vector, *H*.
s is the standard deviation.
e is the eccentricity

The values that are outputted form a shiny surface if the value is small and dull surfaces if the value is big.
We divide ***DGF*** by ***(N.V)*** to account for the fact that the observer see's more when the surface is tilted and so it is the intensity proportional to the number of facets in ***H*** direction.

The geometric attenuation factor ***G*** was described by Blinn in his paper. It basically accounts for three cases when light rays interfere with the mirco-facets, these are

a) no interference,
b) b) partial interference of reflected light and
c) c) partial interception of incident light. The geometric attenuation factor expands as follows

$G = \min \{ 1, B, C \}$

Where

$$B = \frac{2(N.H)(N.V)}{(V.H)}$$

$$C = \frac{2(N.H)(N.L)}{(V.H)}$$

[8] Owen (1999, online)

The Fresnel term ***F*** accounts for changes in color to the specular highlight.
The fresnel function ***F*** expands as follows

$$F = \frac{1}{2}\frac{\sin^2(\theta i - \theta t)\{1 + \cos^2(\theta i + \theta t)\}}{\sin^2(\theta i + \theta t)\{1 + \cos^2(\theta i - \theta t)\}}$$

$\theta i = 1/\cos(L.H) = 1/\cos(V.H)$
$si\Theta t = sin\Theta/\eta$

Where Θt is the angle of refraction and η *is the refractive index of the surface material.*

However, processing sin and cos is a costly computation therefore within the Blinn shader we use a Fresnel approximation.

```
Fn = pow((1 - VH), 3);
Ff = Fn + (1 - Fn) *
specularRollOff;
```

Where ***VH*** = ***V.H,*** and ***V*** is the reflecting vector and ***H*** is the halfway vector between the surface normal ***N*** and ***V***.

1.2.3.2 Phong Lighting

The phong illumination model has no physical basis but it able to compute specular falloff. The phong illumination model is defined by,

$Lr = Lra + Lrd + Lrs$

$$Lr = kaLi,a + kd\sum Ll(I \,.\, N) + ks \sum_{\kappa e} L(R(I).V)$$

Where
ke is the eccentricity,
Lr, La, Ld, Ls are the lights reflection, ambient, diffuse, and specular components respectively.
ka, kd, ks are the surface's ambient, diffuse, and specular components respectively.
I is the incident vector.
R is the reflection vector.

[9] Cominos (2003, pg178).

This function simply says for each light source perform the Phong lighting model. Now let us examine the RenderMan phong implementation.

```
color
phong( point N, V; float size )
{
 color C = 0;
 point Ln, R;
 R=reflect(-normalize(V),
normalize(N) );
illuminance( P, N, PI/2 )
{
Ln = normalize(L);
C += Cl * pow(max(0.0,R.Ln),
size);  }
        return C;
}[10]
```

Again we find that the implementations are the same, note that the illuminance loop will iterate through all the lights, at point P, using the surface normal N and in the direction of PI/2.

1.2.3 Ward's Anisotropic Model

In 1992, Ward proposed a method for the simulation of anisotropic surfaces. An anisotropic surface will reflect light is a directional manner due to fine detail on the surface that is too fine to render, examples of anisotropic surfaces are vinyl records and compact disks. Maya supports anisotropic shading models and we implement Wards illumination model. The Ward illumination model is an extension of Pierre Poulin [11], and Alain Fourier's model for anisotropic surfaces. The Ward[12] illumination model is as follows.

$$Fr = Pd/\pi + Ps \frac{1}{\exp[-\tan^2\delta(\cos^2\phi/ax^2 + \sin^2\phi/ay^2)]} \cdot \frac{}{\sqrt{\cos\theta i \cos\theta r}} \cdot \frac{}{4\pi\, ax\, ay}$$

Where ***Pd*** is the diffuse reflectance
Ps is the specular reflectance
ax is the standard deviation of the slope in the x direction.
ay is the standard deviation of the slope in the y direction.

[10] Pixar (2004, online)
[11] Poulin and Fourier (1990, 273 -282)
[12 Ward(1992, 265 –272)

δ is the angle between the halfway vector and the surface normal
ϕ is the azimuth angle of the half vector projected onto the surface plane

Again we have to eliminate the cosine and sine operators for performance, so we substitute in the dot product and the cross product. The following equation is used within our RenderMan representation of Ward's illumination model.

$$Fr = Pd/\pi + Ps \frac{1}{\exp(-\tan^2 < (H,N)/\sigma^2)} \cdot \frac{}{\sqrt{(L.N)(V.N)}} \cdot \frac{}{4\pi\sigma^2}$$

Where
Pd is the diffuse reflectance
Ps is the specular reflectance
σ^2 $(\sigma x, \sigma y)$ is the standard deviation of the slope in the x direction and y direction respectively.
L is the light direction vector
N is the surface normal
V is the reflecting vector
H is the halfway vector

2 Shade Trees & RenderMan Shading Language

Previously we have been using a static equation to describe the lighting and shading appearances. These models mean that all surfaces are grouped into specific models.
In his 1984 paper, Cook[13] developed the idea of shade trees. The paper introduced a flexible tree like structure that can represent many surface appearances. This is a very powerful model that allows the user to develop complex shading models using a "shading language" without having to delve into the actual framework of the rendering program. Perlin[14] implemented the shade tree language with a flow of control in 1985. A shade tree consists of a variety of nodes that are organised in a tree structure. The evaluation of a shade tree provides the resultant color of a point on a surface.

[13] Cook (1984, 223 - 231)
[14] Perlin (1985, 287 - 296)

However, to describe a complex surface appearance using a shade tree becomes quite unwieldy. Shade tree's were soon developed into a programmable shading language and this most noticible in RenderMan Shading Language, developed by Pat Hanrahan[15] in the RenderMan specification in 1989, later Hanrahan and Lawson developed the shading language compiler. An example shade tree for a plastic shader might look like the following.

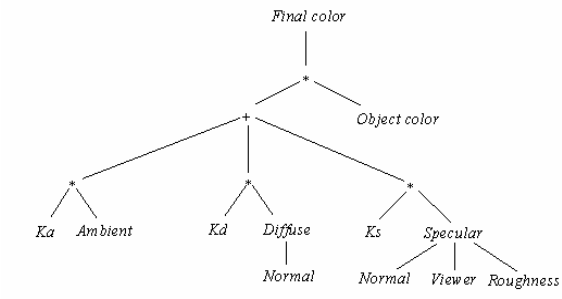

Figure 1 *Plastic Shading Model*

In RenderMan, this provides a small set of high level data types, with a full set of control structures as well as mathematical and shading functions. These set of functions are built-in to the shading laguage compiler, the shader interpreter implements these shaders at run-time. These functions are sometimes called shadeops. The interpreter executes as a virtual SIMD (single instruction, multiple data) vector math unit. The shader instructions are interpreted one at a time on all vertices. This is far more effiecient than running the entire shader on each vertex separately. However, to ensure that conditional instructions are evaluated properly the SIMD controller has run flags that flag the vertices as active or inactive. Another advantage is that information on the neighbourhood is available for most vertices. This allows differentials to be computed. Those vertices with less than four neighbours are estimated [16].

[15] Hanrahan (1989)
[16] Apodaca and Gritz (1999, pg 139)

3 Implementing the HyperMan system

The HyperMan system allows for the automated translation of Maya materials into RenderMan shaders. Maya's highly programmable and customisable format provides an excellent method to extract appearances from Maya and translate these into RenderMan via the node structure built into Maya (HyperGraph Nodes).
More information is available in the technical documentation.

3.1 Converting Textures

When a material is constructed in Maya a set of nodes are generated to form a structure that is able to descibe the surface appearance. Using the DAG (Directed Acyclic Graph), we are able to view the entire scene construction. However it is clear that only shading nodes need to be considered.

Converting the textures into a shading language format is simple. Using the shading models outlined in section one we construct the RenderMan shading language equivalent. The high functionality on the RenderMan shading language allows us to "plug" anything into the equation to give a variety of results.

In order to simplify the shading model we use texture maps to convert child nodes in a Maya shading tree. This makes the solution very simple, as we don't have to create a shading function equivalent to generate maps, allowing us to skip over the need to calculate computations based on attributes and node type. The basic process for converting a Maya shading group into a RenderMan shader is as follows.

1. For each surface appearance (i.e Blinn, Lambert etc) create a base RenderMan equivalent.
2. For each Maya attribute create the equivalent processing function e.g. if we want to map the translucency in Maya to a ramp, we have an equivalent function we takes data and processes it in the same way.

3. Within Maya iterate through the first set of children connected to a shader, determine the UV placement, and then convert the node into a texture map.
4. We then convert the outputted map into a RenderMan texture file.
5. We pass the file names of the RenderMan textures to RenderMan base shader equivalent.
6. By determining the placement we can imitate Maya's 2D and 3D placement nodes using the shading language call texture() or environment()
7. Render the scene via MTOR.

The philosophy behind the system is essentially very basic, but it provides a efficient conversion method via the use of texture calls.

3.2 Advanced Features

The HyperMan system also incorporates many advanced features that exploit the functionality of RenderMan to provide users with full customisation. The system incorporates will automate the process of generating complex features which are unavailable to Maya's software renderer. It also matches the Mental Ray features, such as Final Gather. The below table provides a comparison between the HyperMan system, Maya software, and Mental Ray.

Feature	Maya Software	Mental Ray	HyperMan
Anti Aliasing	Supported	Supported	Supported
Raytracing	Supported	Supported	Supported
Motion Blur	Supported	Supported	Supported
Multi Processing	Supported	Supported	Supported
Layers	Supported	Supported	Supported
Caustics	Unsupported	Supported	Supported
Global Illumination	Unsupported	Supported	Supported
Final Gather	Unsupported	Supported	Supported
Ambient Occlusion	Unsupported	Supported	Supported
Irradiance	Unsupported	Supported	Supported
Image Based Illumination	Unsupported	Supported	Supported
Custom Layer	N/A	Unsupported	Supported
Custom shader	N/A	Unsupported	Supported
Caching	Supported	Supported	Supported
Animation Caching	Unsupported	Unsupported	Supported

Figure 2 *Comparing the systems*

3.2.1 Custom Layers/Shaders

The HyperMan system allows users to create a shader using RenderMan shading language and integrates it into the scene. This expands the functionality of the system and will allow for new features available in RenderMan to be incorporated effortlessly.

Using RenderMan AOV's (arbitrary output variables), we can create a shader and create a custom Layer. For example if we want to output the surface normal, we create a shader in RenderMan shading language. By giving HyperMan the absolute path to the compiled shader we, and a selection set which holds the objects that the shader is to be attached to, we can generate a custom pass.
However the limitations are that the HyperMan shaders will not support the AOV, i.e. any surface that isn't attached to the custom shader wont generate a new pass.

3.2.2 Animation Caching

RenderMan now supports many-advanced ray tracing features, to exploit this we allow the use of cache files. Mental Ray also supports cache file however although they are camera independent the process needs to be recalculated if an object moves.
This is where HyperMan provides an improved method for caching. A cache is generated for an animation x (user specified) amount of times, we then combine these cache files into one file and force the
shader to lookup the cache. This initial process can be be slow but then rendering the animation is fast as no ray tracing calculations are made.

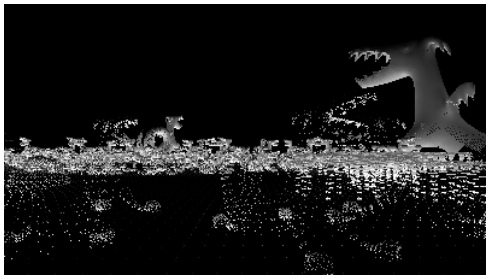

Fig 3 *Occlusion for keyframe 1 (dragons1.ocf)*

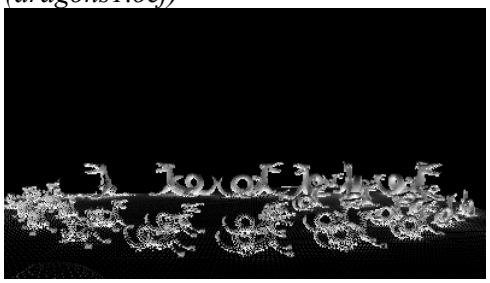

Fig 4 *Occlusion for keyframe 2 (dragons2.ocf)*

Fig 5 *Occlusion for keyframe 3 (dragons3.ocf)*

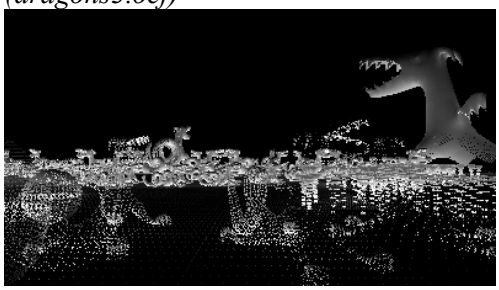

Fig 6 *Combined occlusion data* (dragonsall.ocf)[17]

[17] Pixar(2004, online)

4 Results from HyperMan

Below are a few examples of HyperMan's capability in comparison to the Maya Software Renderer and Mental Ray.

Fig 7 Maya Software Raytrace Production

Fig 8 Mental Ray Final Gather

Fig 9 HyperMan Average Quality

5 Conclusion & Improvements

The system is far off perfect, and needs to more efficient method of extracting data from Maya, the obvious solution is to use the Maya API that allows iterating through the DAG, without having to query connections. In this implementation for each shader all connections need to be queried, by using the API we can use a MFNiterator which gives us the ability to move down the DAG heirachy without having to perform any processing. However we would still need to use MEL to convert the nodes into texture files.

MayaMan a system developed by Animal Logic will create new shaders on the fly in correspondance to the shading network. This is a better approach although due to time restrictions I was unable to recreate each node in shading language.

Pixar are producing a system that integrates Maya with RenderMan. HyperMan integrates into Maya in seamlessly and from a user point of view (in terms of interaction) there are hardly any differences between the two systems. HyperMan still uses MTOR, and this is where HyperMan has added functionality, as it uses the Alfred to render, which allows us to utilize the network rendering capabilities of RenderMan.

The system has limitations, to system is not portable to RenderMan compliant renderers and so it requires Maya, MTOR (maya to renderman plug-in), and RenderMan. The solution would be to redevelop the system to write shaders on the fly and then parse the rib looking for each surface's identity attribute and then replacing the surface attribute with our new shader.

Release paper available
User documentation available
Technical documentation available

6 References

Apodaca, A. A. AND Gritz , L., 1999 Advanced RenderMan. San Francisco: Morgan Kaufmann.

Blinn, J., 1977, Models of Light Reflection for Computer Synthesized Pictures, SIGGRAPH 77, pp 192-198.

Cominos, P., 2003, Mathematical and Computer Programming Techniques for Computer Graphics. Bournemouth: Bournemouth University, England

Cook, R. L., 1984 , Shade Trees SIGGRAPH 84, pp 223- 231

Hanrahan, P. 1989, RenderMan Interface Specification

Hart, J. C. 2003, Advanced Topics in Computer Graphics. Illinois: University of Illinois

Miscler, G. 1998, Lighting Knowledge Database [online] München, Germany Available from http://www.schorsch.com/kbase/glossary/lambertian_surface.html [Accessed 25 Feburary 2005]

Owen, S. G. 1999, Jim Blinn Model For Specular Reflection [online] Georgia: Georgia State University Available from http://www.siggraph.org/education/materials/HyperGraph/illumin/specular_highlights/blinn_model_for_specular_reflect_1.htm [Accessed 26 February 2005]

Perlin, K. 1895 An Image Synthesizer ACM Computer Graphics, New York: New York University Media Research Labs

Phong, B. T. 1973 Illumination for Computer Generated Images, Utah: Utah Computer Science Dept.

Pixar 2004 [online] RenderMan Documentation Available from https://**renderman**.pixar.com/products/**ri**spec/**ri**spec_pdf/RISpec3_1.pdf [Acessed 26 February 2005]

Poulin, P. and Fourier, A. 1990, A Model For Anisotropic Reflection, Siggraph 90, pp 273 – 282

Seidel, H. P. and Myszkowski, K. 2004, Computer Graphics: Light Transport 2, Saarbrucken, Germany [online] Available from http://www.mpi-sb.mpg.de/units/ag4/teaching/uebung/lecture14.pdf [Accessed 26 February 2005]

Stephenson, I. 2003, Essential RenderMan Fast London: Springer

Ward, G. 1992, Measuring and Modelling Anisotropic Reflections. Siggraph 92 pp 3-7

Wynn, C. An Introduction to BRDF lighting [online] Available from http://download.nvidia.com/developer/presentations/2004/Eurographics/EG_04_TutorialNotes.pdf [Accessed 1 March 2005]