Shading – the basics

The purpose of this section is to introduce the basics of rendering objects photo-realistically. It is intended to be a practical introduction and as such it does not address the theory of rendering.

To produce a realistic image a renderer must be provided with information about the nature of the light sources and the material and geometric attributes of the objects in a synthetic scene. In addition, the renderer also requires information about the virtual camera that is being used to view the scene. For the sake of simplicity the examples in this section use a camera with a standard focal length lens, that provides full coloured low resolution images, eg.

```
Projection "perspective" "fov" 40
Display "untitled" "framebuffer" "rgb"
Format 300 200 1
```

A simple object from the RenderMan library of quadric surfaces, eg.

```
Sphere  1  -1  1  360
```

is used throughout this section so that any undue complications that might arise as a result of modelling a complex object(s) can be avoided.

If computer graphics can be likened to “painting by numbers”, then the process of rendering a 3D scene can be thought of as “painting with light”. Therefore, a photo-realistic system such as RenderMan, whose purpose is to create imagery that is indistinguishable from ‘real’ photography, is useful only to the extent that it enables users to make subtle changes to the way light interacts with each part of a scene.

At each stage in the rendering process the way that light is changed as a result of these interactions is called *shading*, and RenderMan provides a mechanism for controlling the outcome of each interaction through the use of what it describes as *shaders*. For example, the interaction of light with the surface of an object is controlled by a *surface shader*; while the characteristics of the light produced by a source of illumination are governed by a *light source shader*. In addition to these two types of shaders this section also introduces a shader that controls the way light reacts with an object whose surface has been displaced to form small bumps and pits – a *displacement shader*.

Although RenderMan provides seven different types of shaders, this section only introduces three of them. The following RIB scripts have been carefully selected to act as templates for your own experiments in the creative use of Surface, Lightsource and Displacement shaders.

Using lights and materials

The following scene consists of a coloured 'plastic' sphere that is illuminated by two lights.

RIB

```
Projection "perspective" "fov" 40
Display "untitled" "framebuffer" "rgb"
Format 300 200 1

Translate  0 0 5
Rotate  -120  1 0 0
Rotate  25  0 0 1

WorldBegin
    LightSource "ambientlight" 1 "intensity" 0.1
    LightSource "distantlight" 2 "intensity" 1.5 "from" [0 0 4] "to" [0 0 0]
    Color  1 0 0
    Surface  "plastic"
    Sphere  1  -1  1  360
WorldEnd
```

Of the two light sources used to illuminate the scene the first provides a very small amount of background, or ambient lighting. The second light is positioned directly 'above' the sphere and is pointing toward the origin of the coordinate system and hence toward the centre of the sphere. Both "ambient-light" and "distantlight" are provided by RenderMan as part of a basic library of four types of light sources, the others are "pointlight" and "spotlight".

The surface of the sphere has been assigned the reflective properties of plastic eg.

```
Surface  "plastic"
Sphere  1  -1  1  360
```

(Using a shiny material makes it somewhat easier to see how the high-lights respond to any changes that are made to the position of the light sources.) RenderMan supplies a small number of materials eg. plastic, wood, granite, carpet etc., that can be used with the Surface statement. Each material has its own way of being "tuned" to the individual requirements of a scene. For example, the sphere could have been given the properties of a very matte and non-reflective roughened plastic with the following statement,

```
Surface  "plastic" "Ks" 0.1 "roughness" 0.5
```

The surface shader "plastic" can use five parameters (the default values for these are given in parentheses), namely,

"Ka"	response to ambient light (1.0),
"Kd"	diffuse reflections (0.5),
"Ks"	specular reflections (0.5),
"roughness"	graininess of the surface (0.1), and finally,
"specularcolor"	the colour of the high-lights ([1 1 1]).

Applying an image to an object

RIB

```
Projection "perspective" "fov" 40
Display "untitled" "framebuffer" "rgb"
Format 300 200 1

Translate  0 0 5
Rotate  -120  1 0 0
Rotate  25  0 0 1

MakeTexture "your picture.tiff" "your picture.tx" "periodic" "periodic" "gaussian" 2 2

WorldBegin
    LightSource "ambientlight" 1 "intensity" 0.1
    LightSource "distantlight" 2 "intensity" 1.5 "from" [0 0 4] "to" [0 0 0]
    Surface  "texmap" "texname" ["your picture.tx"] "maptype" 2
    Sphere  1  -1  1  360
WorldEnd
```

The scene used in this example is very similiar to the first except that a surface shader called "texmap" is used to ‘wrap’ a 2D image around the sphere – a technique known as *texture mapping*,

```
Surface  "texmap" "texname" ["your picture.tx"] "maptype" 2
```

The term texture in the context of 3D computer graphics is a little misleading because it only refers to variations of the colour of a surface, it does not imply anything about its structure or roughness. Since there are a number of ways an image can be wrapped around an object, "texmap" must be told to use "maptype" 2 ie. spherical mapping. Before "texmap" can apply an image to a surface, the image must first be used to generate an intermediate *texture file*. The following statement does the necessay conversion,

```
MakeTexture "your picture.tiff" "your picture.tx" "periodic" "periodic"
"gaussian" 2 2
```

The main thing to note is that the image to be used as the source for the texture file, which in this instance is called "your picture.tiff", is located in the same folder as the RIB file itself – otherwise the renderer has no way of knowing where to find the appropriate file. Generally, picture files are either created or modified using PhotoShop. It is essential they are stored as RGB files rather than, say, gray scale images. Once an image has been used to create a texture file the MakeTexture statement can be ‘commented-out’.

The purpose of the last part of the MakeTexture statement ie. "periodic" "periodic" "gaussian" 2 2, is to allow the texture to be repeatedly tiled over the sphere should that be necessary and to ensure the resulting texture map has a smooth, or anti-aliased, appearance. Like many of the area’s touched upon by this section, discussions about the finer details of texture mapping are dealt with elsewhere.

Preparing an image for texture mapping

Step 1 Scan and / or modify a graphic using PhotoShop, save it as either a TIFF or a PhotoShop 2.5 file. Even if it is a monochrome image be sure to manipulate it in RGB mode within PhotoShop.

Step 2 Reduce the graphic to a square format by choosing “Image Size…” from the “Image” menu item. The Image Size dialog box will allow the graphic to be resized to a square aspect ratio, say 800 x 800 pixels.

Step 3 “Select All” using the Select menu item then rotate the graphic 180 degrees with the “Rotate” command under the Image menu.

Step 4 Use “Save As…” to store the graphic as a TIFF file in the same folder as the RIB file that will use it for texture mapping.

DO NOT MODIFY THE ORIGINAL GRAPHIC – ALWAYS WORK ON A COPY.

Although the Macintosh operating system does not require file extensions, make sure the image file is named with a “.tiff” extension eg. me.tiff. Naming image files and RIB scripts with “.tiff” and “.rib” extensions makes them very easy to identify on the desktop. It is advisable to compress the file using LZW compression.

Step 5 Within the RIB script convert the graphics file to a texture file with the statement,

```
MakeTexture  "me.tiff" "me.tx" "periodic" "periodic" "gaussian" 2 2
```

Once the RIB script has been used successfully the MakeTexture statement can be commented-out ie.

```
#MakeTexture  "me.tiff" "me.tx" "periodic" "periodic" "gaussian" 2 2
```

This will ensure that subsequent renderings will be completed as quickly as possible. Of course if the original graphics file is altered then a new texture file must be produced, in which case the comment (ie. #) must be removed.

Step 6 Use the surface shader “texmap” to wrap the texture file around the sphere.

```
Surface "texmap" "texname" ["me.tx"] "maptype" 2
Sphere 1  -1  1  360
```

The “texmap” surface shader allows the same parameters as the “plastic” shader (ie. Ka, Kd, Ks, roughness and specularcolor) to be used to control the way light reflects from the surface of the texture map.

Using an image to displace a surface

RIB

```
Projection "perspective" "fov" 40
Display "untitled" "framebuffer" "rgb"
Format 300 200 1

Translate  0 0 5
Rotate  -120  1 0 0
Rotate  25  0 0 1

MakeTexture "your picture.tiff" "your picture.tx" "periodic" "periodic" "gaussian" 2 2

WorldBegin
    LightSource "ambientlight" 1 "intensity" 0.1
    LightSource "distantlight" 2 "intensity" 1.5 "from" [0 0 4] "to" [0 0 0]
    Displacement  "emboss" "texname" ["your picture.tx"] "Km" 0.03
    Sphere  1  -1  1  360
WorldEnd
```

In this example an image is used to upset or displace the surface of a sphere – a technique known as *displacement mapping.* Similiar to the surface shader “texmap”, the “emboss” displacement shader requires an image file to be converted to an intermediate *texture file* ie.

```
MakeTexture "your picture.tiff" "your picture.tx" "periodic" "periodic"
"gaussian" 2 2
```

Again you must ensure the image to be used as the source for the displacement map is located in the same folder as the RIB file itself. A tiff file to be used as a source image for displacement mapping can be saved in either PhotoShop’s gray-scale or rgb mode. However, gray scale images give a more pronounced embossing effect. Once an image has been used to create a texture file the MakeTexture statement can be ‘commented-out’ eg.

```
#MakeTexture "your picture.tiff" "your picture.tx" "periodic" "periodic"
"gaussian" 2 2
```

Unlike a technique called “bump mapping”, displacement mapping really does make changes to the geometry of the surface to which it is applied. It is, therefore, superior to bump mapping which only makes a surface **appear** to be bumpy. The “emboss” displacement shader must be given the name of a texture file that it will use for embossing ie.

```
Displacement "emboss" ["your picture.tx"] "Km" 0.03
```

It should also be given a reasonable value for the parameter “Km” which sets the magnitude of the embossing. The “emboss” shader responds to the gray values in the texture file – lighter parts of the image are ‘pressed’ deeper into the surface. The default value for “Km” is 0.03.

Avoiding rendering errors and improving performance

Bounding boxes

The renderer uses each objects bounding box to quickly determine where their surfaces are located in a scene. In this way it avoids trying to render "empty space". However, when the surface of an object is shifted as a result of displacement mapping, PRMAN may make severe rendering errors. The renderer literally ignores those parts of the object that have been displaced outside their bounding box. RenderMan provides a mechanism by which the renderer can be warned about such displacements eg.

```
Attribute "bound" "displacement" [0.2]
Displacement "emboss" ["your picture.tx"] "Km" 0.03
 Shere 1  -1  1  360
```

As long as it appears before the name of the object, Attribute can be inserted before or after the Displacement statement. Unfortunately, even though the displacement magnitude is set with the Km factor their is no way of knowing the exact Attribute value to use. In the example shown above it is set to 0.2 but often 0.1 is enough to prevent rendering errors from occuring.

Memory and speed

Rendering operations that involve texture files require more memory than those that don't. To ensure it can operate on computers with modest amounts of memory, RenderMan only sets aside a small amount of memory for working with textures. To improve performance an option may be set to inform the renderer to work with larger "chunks" of texture ie.

```
Option "limits" "texturememory" [4096]
```

The value "4096" specifies the number of Kbytes (4 MB) to set aside for memory to be used to store information read from a texture file. If you wish to use this option place it at the beginning of the script – options effect the whole scene and must be set before the camera and world are described.