

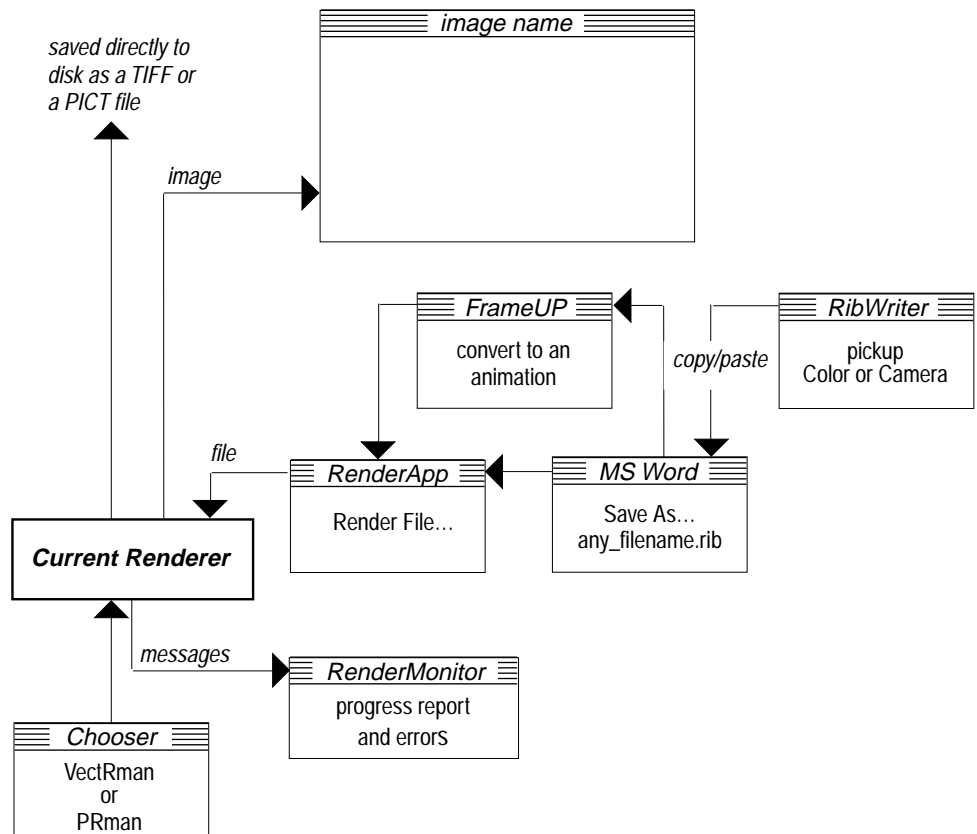
Appendix A – an overview of MacRenderMan

The diagram shown below represents the environment in which RIB scripts are written and rendered is shown below. Although it may appear to be complicated, working with MacRenderMan is reasonably straight forward. Initially, a RIB file is written using a word processor such as MicroSoft Word and is saved as “text only” with a .rib extension, for example, scene1.rib.

A small utility program called “RibWriter” can be used to write RIB scripts that correspond to a chosen camera position/angle, as well as a particular colour. The information from RibWriter can be copied and pasted into the RIB file being prepared in MS Word. “FrameUP” is described in the section dealing with Animation.

Either the vector (ie. line) or the photo-realistic renderer can be selected via “Chooser” in the same way as a network printer can be set-up for printing. The RIB file is sent to the chosen renderer via a small application called “RenderApp”. If the RIB file has been written so that the rendered scene is to be displayed immediately, rather than being saved as a TIFF or PICT file, RenderApp will open a window for the rendered image.

Information about the rendering process is automatically reported via a utility called “RenderMonitor” in much the same way as the status of a network printer is displayed by “PrintMonitor”.



Appendix B – RenderMan Quick Reference

The reference material contained in this section is based largely on PIXAR's PhotoRealistic RenderMan Application Notes¹ #1 and #8 published in May 1990. The original "Quick Reference" was intended to be used by programmers and, therefore, contained a great deal of information that is irrelevant to those who wish to write RIB scripts directly by hand. I have tailored this reference to meet the needs of the Vcdn301 course. Several RIB statements have been omitted² because they relate to very advanced capabilities of the RenderMan interface. The information in this reference is of necessity very terse and is really only intended to act as a 'memory jogger'.

RIB Summary

Shape – geometric primitives

Cone
Cylinder
Disk
Hyperboloid
Paraboloid
Sphere
Torus
GeneralPolygon
PointsGeneralPolygons
PointsPolygons
Polygon

Camera

Clipping
DepthOfField
Display
Exposure
Format
FrameAspectRatio
FrameBegin/End
MotionBegin/End
Perspective
Projection
Shutter

Bookkeeping

Declare
Option

Space – transformations and grouping

Rotate
Scale
Skew
Translate
AttributeBegin/End
ObjectBegin/End
ObjectInstance
Sides
SolidBegin/End
TransformBegin/End
WorldBegin/End

Shading

AreaLightSource
Atmosphere
Color
LightSource
MakeCubeFaceEnvironment
MakeLatLongEnvironment
MakeShadow
MakeTexture
Opacity
ShadingInterpolation
ShadingRate
Surface
TextureCoordinates

¹ PhotoRealistic RenderMan Application Note #1 "A Brief Introduction to the RenderMan Interface"; PhotoRealistic RenderMan Application Note #8 "RenderMan Quick Reference"

² These include references to splines, trim curves, patches and patch meshes; transformation matrices; user defined coordinate systems; levels of detail; geometric approximation; bump mapping (not supported on the Macintosh platform); error handling; image filtering, sampling and quantization.

Shape – geometric primitives

Cone Cone height radius thetamax parameters
Defines a partial or complete cone.
example

```
Cone 0.5 0.5 270 "Cs" [1 0 0 1 0 0 1 1 1 1 1 1]
```

"Cs" defines colours for the parameter space, which in this example provides the cone with a red base and a white apex.

Cylinder Cylinder radius zmin zmax thetamax parameters
Defines a partial or complete cylinder.
example

```
Cylinder 0.5 0.2 1 360 "Os" [0 0 0 0 0 0 1 1 1 1 1 1]
```

"Os" defines opacity for the parameter space, which in this example provides the cylinder with a fully transparent base (opacity = 0,0,0) and a fully opaque top (opacity = 1,1,1).

Disk Disk height radius thetamax parameters
Defines a partial or complete disk.
example

```
Disk 1.0 0.5 270 "Os" [0 0 0 0 0 0 1 1 1 1 1 1]
```

Opacity is used here to give the disk a fully transparent rim and a fully opaque centre.

Hyperboloid Hyperboloid x1 y1 z1 x2 y2 z2 thetamax parameters
Defines a partial or complete hyperboloid.
example

```
Hyperboloid 1.0 -1.0 -1.0 1.0 1.0 1.0 360
```

Paraboloid Paraboloid rmax zmin zmax thetamax parameters
Defines a partial or complete paraboloid.
example

```
Paraboloid 0.5 0.2 0.7 270
```

Sphere Sphere radius zmin zmax thetamax parameters
Defines a partial or complete sphere.
example

```
Sphere 0.5 0.0 0.5 360  
"Cs" [1 0 0 1 0 0 0 0 1 0 0 1]  
"Os" [0.7 0 0 0.7 0 0 1 1 1 1 1 1]
```

Both opacity and colour are used for the parameter space, which in this example provides the sphere with a semi-transparent red "base" and an opaque blue "top".

Torus	<p>Torus rmajor rmin phimin phimax thetamax parameters <i>Defines a partial or complete torus.</i> <i>example</i> Torus 3.5 0.25 0.0 180 300</p>
GeneralPolygon	<p>GeneralPolygon nloops nvertices parameters <i>Defines a single convex or concave (general) planar polygon, with optional holes.</i> <i>example</i> GeneralPolygon [3 3] "P" [-1.0 -1.0 0.0 -1.0 1.0 0.0 1.0 -1.0 0.0 -0.5 -0.5 0.0 0.0 0.5 0.0 0.5 -0.5 0.0]</p>
PointsGeneralPolygons	<p>PointsGeneralPolygons numLoops numVertices listVertices parameters <i>Defines several planar general polygons, with optional holes, that share vertices.</i> <i>example</i> PointsGeneralPolygons [2 2][4 3 4 3][0 1 3 4 6 7 8 1 2 5 4 9 10 11] "P" [0 0 1 0 1 1 0 2 1 0 0 0 0 1 0 0 2 0 0 0 2 0.5 0 0 7 0.7 0 1 7 0.2 0 1 2 0.5 0 1 7 0.7 0 1 7 0.2]</p>
PointsPolygons	<p>PointsPolygon numVertices listVertices parameters <i>Defines several non-concave polygons, without holes, that share vertices.</i> <i>example</i> PointsPolygons [3 3 3][0 3 2 0 1 3 1 4 3] "P" [0 1 1 0 3 1 0 0 0 0 2 0 0 4 0]</p>
Polygon	<p>Polygon listVertices parameters <i>Defines a single non-concave polygon with optional list of parameters that supplies information about vertex normals, colour, opacity and/or texture coordinates.</i> <i>examples</i> –no additional parameters Polygon "P" [0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0] –plus additional parameters relating to vertex colours Polygon "P" [0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0] "Cs" [1 0 0 0 1 0 0 0 1 1 1 1] –plus additional parameters relating to vertex opacity Polygon "P" [0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0] "Os" [1 1 1 0 0 0 0.5 0.5 0.5 0.2 0.5 0.8]</p>

Shape – transformations and grouping

Rotate	<p>Rotate angle dx dy dz <i>Turns the object space so that it is rotated by "angle" degrees around the given axis prior to a shape being defined.</i> <i>example</i> Rotate 90 0 1 0</p>
Scale	<p>Scale sx sy sz <i>Stretches or compresses the object space so that it is scaled along the x, y and z axes prior to a shape being defined.</i> <i>example</i> Scale 0.5 1.0 1.0</p>
Skew	<p>Skew angle dx1 dy1 dz1 dx2 dy2 dz2 <i>Shears the object space so that it is skewed by "angle" degrees along the x, y and z axes prior to a shape being defined.</i> <i>example</i> Skew 45 01 0 1 0 0</p>
Translate	<p>Translate dx dy dz <i>Shifts the origin of the object space so that it is translated along the x, y and z axes prior to a shape being defined.</i> <i>example</i> Translate 0.0 1.5 0.0</p>
AttributeBegin/End	<p>AttributeBegin/AttributeEnd <i>Forms a group of shapes, transformations and surface attributes.</i> <i>example</i> AttributeBegin some attributes such as color 1 0 0 AttributeEnd</p>
ObjectBegin/End	<p>ObjectBegin identifier/ObjectEnd <i>Defines a collection of shapes as a "retained" object that can be inserted, or instanced, within a scene.</i> <i>example</i> ObjectBegin 4 object 1 object 2... ObjectEnd</p>
ObjectInstance	<p>ObjectInstance identifier <i>Inserts, or instancies, a previously retained collection of shapes as a single object.</i> <i>example</i> ObjectInstance 4</p>

Sides	<p>Sides sides</p> <p><i>Defines subsequent shapes as single sided or double sided.</i></p> <p><i>example</i></p> <p>Sides 2</p>
SolidBegin/End	<p>SolidBegin operation/SolidEnd</p> <p><i>Defines a collection of shapes as a "solid" object according to the rules of constructive solid modelling ie. union, intersection and difference.</i></p> <p><i>example</i></p> <p>SolidBegin "union"</p> <p> define two or more objects to be joined together as a single object</p> <p>SolidEnd</p>
TransformBegin/End	<p>TransformBegin/TransformEnd</p> <p><i>Forms a group of shapes and transformations but IGNORES surface attributes.</i></p> <p><i>example</i></p> <p>TransformBegin</p> <p> some transformations ex. Rotate</p> <p> some shapes</p> <p>TransformEnd</p>
WorldBegin/End	<p>WorldBegin/WorldEnd</p> <p><i>Freezes the characteristics of the camera and marks the beginning of a world description.</i></p> <p><i>example</i></p> <p>WorldBegin</p> <p> scene description</p> <p>WorldEnd</p>

Camera

Clipping	Clipping near far <i>Sets the near and far clipping planes along the direction of view.</i> <i>example</i> Clipping 0.1 1000
DepthOfField	DepthOfField fstop focallength focaldistance <i>Parameters to simulate the depth of field.</i> <i>example</i> DepthOfField 22 1 26.7
Display	Display name type mode parameters <i>Chooses a display by name and sets the type of output being generated.</i> <i>examples</i> Display "filename" "file" "rgba" Display "filename" "zfile" "z" Display "windowname" "framebuffer" "rgba"
Exposure	Exposure gain gamma <i>Controls the sensitivity and non-linearity of the exposure process.</i> <i>example</i> Exposure 1.5 2.3
Format	Format xresolution yresolution pixelaspectratio <i>Sets the horizontal and vertical resolution in pixels of the image to be rendered.</i> <i>example</i> Format 400 300 1
FrameAspectRatio	FrameAspectRatio ratio <i>Ratio sets the ratio of the width to height of the desired image.</i> <i>example</i> FrameAspectRatio 1.333
FrameBegin/End	FrameBegin/FrameEnd <i>Marks the beginning and end of a frame of animation.</i> <i>example</i> FrameBegin 1 scene description for this frame FrameEnd

MotionBegin/End

MotionBegin t0 t1...tn-1/MotionEnd
Marks the beginning and end of motion
example

```
MotionBegin [0 1]
    transformation information at time 0
    transformation information at time 1
MotionEnd
```

Perspective

Perspective fov
Sets the camera to give a perspective view.
example

```
Perspective 90
```

Projection

Projection name parameters
Sets the type of projection and activates the camera coordinate system ie. the world coordinate system is only active between WorldBegin and WorldEnd.
example

```
Projection "perspective" "fov" 40
```

Shutter

Shutter opentime closetime
Sets the times at which the shutter opens and closes.
example

```
Shutter 0 1
```

Shading

AreaLightSource AreaLightSource name int parameters
Creates an area light and makes it the current light source. Each subsequent object is added to the list of surfaces that define the area light.

example

```
AreaLightSource "finitelight" 1 "decayexponent" 0.5
AreaLightSource "glowlight" 2 "color" [0.5 0 0] "intensity" 0.6
```

Atmosphere Atmosphere name parameters
Sets the currently active atmosphere shader.

examples

```
Atmosphere "fog" "background" [0.2 0.2 0.3] "distance" 39.4
Atmosphere "depthcue" "background" [0.2 0.2 0.3] "mindistance" ?
"maxdistance" ?
```

Color Color red green blue
Sets the colour that will be applied to subsequent objects.

example

```
Color 0.2 0.3 0.9
```

LightSource LightSource name sequencenumber parameters
Creates a non-area ie. infinitely small, light source, turns it on, and adds it to any other lights previously created.

example

```
LightSource "ambient" 2 "intensity" 10
```

MakeCubeFaceEnvironment

MakeCubeFaceEnvironment px nx py ny pz nz texturename fov filter swidth twidth parameters

Converts six images in a standard picture file (for example a TIFF file) representing six viewing directions into an environment map.

example

```
MakeCubeFaceEnvironment "foo.x" "foo.nx" "foo.y" "foo.ny" "foo.z" "foo.nz"
"foo.env" 95 "gaussian" 2.0 2.0
```

MakeLatLongEnvironment

MakeLatLongEnvironment picturename texturename filter swidth twidth parameters

Converts an image in a standard picture file (for example a TIFF file) representing a latitude-longitude map whose name is picturename into an environment map called texturename.

example

```
MakeLatLongEnvironment "long.tiff" "long.tx" "gaussian" 2 2
```

MakeShadow

MakeShadow picturename texturename parameters

Converts a depth image file into a shadow map.

example

```
MakeShadow "shadow.tiff" "shadow.tx"
```

MakeTexture

MakeTexture picturename texturename swrap twrap filter swidth twidth

Converts an image in a standard picture file (eg. TIFF) into a texture file.

examples

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

```
MakeTexture "globe.tiff" "globe.tx" "black" "black" "gaussian" 2 2
```

```
MakeTexture "globe.tiff" "globe.tx" "clamp" "clamp" "gaussian" 2 2
```

In the first example the image will if necessary repeat horizontally and vertically. In the second example the image will be mapped once and will be surrounded by black. While in the last example, the colour of the pixels image at the extreme edge of the image will be "smeared" outward if there is enough space available on the object being texture mapped.

Opacity

Opacity c1 c2 c3

Sets the opacity to the colour channels c1, c2, c3, like the use of Color, subsequent objects are set to these levels of opacity.

example

```
Opacity 0.5 1.0 1.0
```

ShadingInterpolation

ShadingInterpolation type

Controls how the values are interpolated ie. estimated, between shading samples.

examples

```
ShadingInterpolation "constant"
```

```
ShadingInterpolation "smooth"
```

ShadingRate

ShadingRate size

Sets the number screen pixels, across and down the image, the renderer will skip between making its shading calculations – large numbers give fast but coarse images. The skipped pixels are either shaded with constant or "smoothed" colour – see above.

example

```
ShadingRate 10
```

Surface

Surface name parameters

Sets the current surface shader, subsequent surfaces acquire the 'look' of the chosen surface.

example

```
Surface "wood" "roughness" 0.3 "Kd" 1.0
```

TextureCoordinates

TextureCoordinates s1 t1 s2 t2 s3 t3 s4 t4

Sets the current set of texture coordinates.

example

```
TextureCoordinates 0.0 0.0 2.0 -0.5 -0.5 1.75 3.0 3.0
```

Bookkeeping

any text upto the **end of a line** is a comment
Enables notes to be included in a RIB file and ensures these will be ignored by the renderer.

examples

```
# this is a comment
```

```
Color 1 0 0 #this is another comment
```

Declare Declare name declaration
Declares a non-standard parameter.

example

```
Declare "centrepoint" "uniform float"
```

Option Option name parameterslist
Allows any pre-set option to be set from within a RIB file.

example

```
Option "limits" "bucketsize" [24 24]
```

```
Option "limits" "texturememory" [1024]
```

Appendix C – Shaders Reference

This reference provides information about the basic shaders that support RenderMan. The data has been compiled from “The RenderMan Companion” by Steve Upstill and the PIXAR document, “MacRenderMan Shaders” published (August 1990) as part of “MacRenderMan Developers Stuff”.

Like ‘plug-ins’ for the Adobe image processing software PhotoShop, an almost unlimited number of (potential) shaders can be added to a RenderMan environment – this document only describes the essential ones.

Shader Summary

LightSource

ambientlight
distantlight
pointlight
spotlight
pointnofalloff
shadowdistant
shadowpoint
shadowspot

Atmosphere

depthcue
fog

Displacement

cloth
dented
diaknurl
droop
emboss
filament
sinknurl
threads

Surface

blue_marble
carpet
checker
cmarble
glass
glassbal
glow
matte
metal
plastic
rmarble
rsmetal
rubber
screen
show_st
show_xyz
sinknurl
spatter
stippled
stone
texmap
txtplastic
wood
transparent_texture
eroded

Surface Shaders

blue_marble

```
"blue_marble" "Ks" "Kd" "Ka" "roughness" "txtscale" "specularcolor"
```

```
"Ks" 0.4 "Kd" 0.6 "Ka" 0.1  
"roughness" 0.1  
"txtscale" 1  
"specularcolor" [1 1 1]
```

This shader gives a surface a very delicate marble-like appearance. It gives the visual impression of turbulent fluid flow, as if the marble had been formed by molten coloured rocks. The txtscale parameter scales the turbulence.

carpet

```
"carpet" "Ka" "Kd" "scuff" "nap"
```

```
"Ka" 0.1 "Kd" 0.6  
"scuff" 1 "nap" 5 "swirl" 1
```

This shader produces a carpeted surface, complete with scuff-marks. The scuff parameter controls the “amount of scuff”, or the relative frequency of intensity variations. Higher values produce more frequent scuffing. nap describes the “shagginess” of the carpet. Higher values make a more coarse-looking carpet.

The carpet shader makes a reasonable stab at anti-aliasing, so the actual grain of the carpet fades away with distance.

There are no specular reflections from real carpet (at least on a macroscopic scale), so the only lighting parameters are Ka and Kd, which have the usual meanings of ambient and diffuse reflective intensities, respectively.

This way anti-aliasing is performed can cause linear artifacts in some cases.

checher

```
"checker" "Kd" "Ka" "frequency" "blackcolor"
```

```
"Kd" 0.5 "Ka" 0.1  
"frequency" 10  
"blackcolor" [0 0 0]
```

This shader imposes a checker-board pattern over a surface. The frequency parameter sets how many times the pattern is to repeat itself within the texture space of the surface. Blackcolor sets the colour to be used for the pattern.

cmarble

```
"cmarble" "Ka" "Ks" "Kd" "roughness" "specularcolor" "veining"
```

```
"Ka" 0.1 "Ks" 0.4 "Kd" 0.6
```

```
"roughness" 0.1
```

```
"specularcolor" [1 1 1]
```

```
"veining" 1
```

This shader produces a marble that consists of coloured veins on a white background with some greyish mottling. The vein colour is determined by the current surface colour. The parameter `veining` controls the frequency of the veins in the marble; higher values produce more and narrower veins.

The parameters `Ka`, `Ks` and `Kd` have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. `Roughness` and `specularcolor` control the sharpness and colour of the specular highlight.

glass

```
"glass" "Ka" "Ks" "Kd" "roughness" "specularcolor" "envname" "Kr"
```

glassbal

```
"glassbal" "Ka" "Ks" "Kd" "roughness" "specularcolor" "envname" "Kr" "eta"
```

```
"Ka" 0 "Ks" 0.6 "Kd" 0
```

```
"roughness" 0.025 (for glass) or 0.2 (for glassbal)
```

```
"specularcolor" [1 1 1]
```

```
"envname" "your environment map"
```

```
"Kr" 0.5 "eta" 0.6
```

The glass shader makes an object appear to be made out of transparent and possibly coloured glass. No refraction is attempted, so the glass appears to be thin, but reflections are simulated using the environment map given with `envname`. The environment map's reflective intensity `Kr` can be controlled to fine-tune the appearance; a lower-intensity reflection may look better on very dark glass. Although you can use the shader even if you don't have an environment map (just ignore `envname`), it will look more like transparent plastic than glass.

You can use the shader for either clear glass or coloured glass by setting the surface colour (clear glass has color [1 1 1]). The transparency of the glass is controlled only by the surface colour; if you want to make the glass less transparent you should make the colour darker.

The `glassbal` shader can be used to make some objects look like solid glass. It simulates the refraction seen through a sphere turning objects seen through the glass upside-down and backwards. The shader needs the environment map given with `envname` to this refraction. It will not work without an environment map. Because the shader simulates refraction as if the object is a sphere, it works well on curvy objects like teapots (and spheres), but will not look correct on flat objects or cylinders.

The colour of the object can be set with the surface colour, just as with glass. The transparency is somewhat different here, however, because the camera is not really seeing "through" the object. Therefore, the surface opacity should always be set to [1 1 1], and the relative intensity of the "refraction" will again depend on the surface colour.

The parameter eta is the relative index of refraction of the atmosphere to the glass. By default, this is the standard value of air (1.0) relative to glass (1.66).

The parameters Ka, Ks and Kd have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. Roughness and specularcolor control the sharpness and colour of the specular highlight.

glassbal needs an environment map; glass looks bad without one.

glow

"glow" "attenuation"

"attenuation" 2

This shader imparts a glow to a surface. The glow is brightest when looking directly at the glowing object, and falls off rapidly nearby.

matte

"matte" "Ka" "Kd"

"Ka" 1 "Kd" 1

A matte surface exhibits only diffuse reflection, because it scatters light uniformly with no preferred direction. That makes the apparent brightness of such a surface independent of the direction from which it is viewed.

metal

"metal" "Ka" "Ks" "roughness"

"Ka" 1 "Ks" 1

"roughness" 0.25

Very similiar to the matte surface shader except that this surface allows specular reflections to occur ie. reflections are concentrated around the mirror direction. Roughness controls the concentration of the specular highlight, a high roughness value giving a more diffuse reflection.

plastic

"plastic" "Ka" "Ks" "Kd" "roughness" "specularcolor"

"Ka" 1.0 "Ks" 0.5 "Kd" 0.5

```
"roughness" 0.1  
"specularcolor" [1 1 1]
```

This shader models a plastic material as a solid medium with microscopic coloured particles suspended within it. The specular highlight is assumed to be reflected directly off the surface, and the surface colour is assumed to be due to light entering the medium, reflecting off the suspended particles, and re-emerging. This explains why the colour of the specular reflection is different from the surface.

rmarble

```
"rmarble" "Ka" "Ks" "Kd" "roughness" "specularcolor" "veining"
```

```
"Ka" 0.1 "Ks" 0.4 "Kd" 0.6  
"roughness" 0.1  
"specularcolor" [1 1 1]  
"veining" 1
```

This shader produces a marble that consists of red veins on a white background with some greyish mottling. The parameter `veining` controls the frequency of the veins in the marble; higher values produce more and narrower veins.

The parameters `Ka`, `Ks` and `Kd` have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. `Roughness` and `specularcolor` control the sharpness and colour of the specular highlight.

rsmetal

```
"rsmetal" "Ka" "Ks" "Kr" "roughness"
```

```
"Ka" 1.0 "Ks" 1.0 "Kr" 1.0  
"roughness" 0.1
```

This shader uses random data for the reflection instead of requiring an environment map. The parameter `Kr` controls the intensity of the reflection. This shader is not recommended for simple spheres, it produces a chrome-plated look on objects with more complex curvature.

rubber

```
"rubber" "Ka" "Kd" "txtscale"
```

```
"Ka" 1.0 "Kd" 1.0  
"txtscale" 1.5
```

This shader is similar to a matte shader except that it includes small amounts of white dust in the surface of the rubber. The amount of dust is controlled by the parameter `txtscale`.

screen "screen" "Ks" "Kd" "Ka" "roughness" "density" "frequency" "specularcolor"

```
"Ks" 0.5 "Kd" 0.5 "Ka" 0.1  
"roughness" 0.1  
"density" 0.25 "frequency" 20  
"specularcolor" [1 1 1]
```

This shader produces a wire-frame appearance. The frequency parameter controls how many grid lines there are in the surfaces texture space; the default produces 20 grid lines per surface. The density parameter controls the portion of the surface that is opaque. The default 0.25 means that the "wires" will cover 25% of the texture space.

show_st "show_st"
no parameters

For each point on a surface, this shader sets its red and green colour equal to the texture coordinates at that point. Transparency cannot be set with this shader - all surfaces are set to be opaque.

show_xyz "show_xyz" "xmin" "ymin" "zmin" "xmax" "ymax" "zmax"

```
"xmin" -1 "ymin" -1 "zmin" -1  
"xmax" 1 "ymax" 1 "zmax" 1
```

This shader converts points within a bounding box, given by the parameters, into red, green and blue values.

spatter "spatter" "Ka" "Ks" "Kd" "roughness" "specularcolor" "basecolor"
"spattercolor" "specksize" "sizes"

```
"Ka" 1 "Ks" 0.7 "Kd" 0.5 "roughness" 0.2  
"specularcolor" [1 1 1] "basecolor" [0.1 0.1 0.5] "spattercolor" [1 1 1]  
"specksize" 0.01 "sizes" 5
```

This shader makes objects look like blue camp cookware with white paint spatters. Actually, both the blue basecolor and the white spattercolor can be changed if you desire.

The paramter specksize controls the size of the paint specks as you would expect. However, there are a range of sizes of paint specks controlled by the parameter sizes. Lower (integer) values produce smaller and more uniform specks. Higher values produce some larger blotches and specks of many different sizes.

The parameters Ka, Ks and Kd, have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. roughness and specularcolor control the sharpness and colour of the specular highlight.

This shader can have problems with aliasing.

stippled

```
"stippled" "Ka" "Ks" "Kd" "roughness" "specularcolor" "grainsize" "stippling"
```

```
"Ka" 0.1 "Ks" 0.3 "Kd" 0.8  
"roughness" 0.3  
"specularcolor" [1 1 1]  
"grainsize" 0.01  
"stippling" 0.2
```

This shader makes objects appear to be made of plastic with lots of little bumps, as computer keyboards, camera surfaces, stucco and many other objects. This is done by making the surface appear to have intensity variation in small grains or granules. The parameter grainsize controls the size of these granules, and stippling controls the relative variation in intensity of the granules; larger values produce a rougher looking surface.

This shader makes a fairly good attempt at anti-aliasing itself, so the granules should appear to fade out with distance in a way similar to a 'real' stippled surface.

The parameters Ka, Ks and Kd have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. Roughness and specularcolor control the sharpness and colour of the specular highlight.

stone

```
"stone" "Ka" "Ks" "Kd" "roughness" "specularcolor" "scale" "nshades"  
"exponent" "graincolor"
```

```
"Ka" 0.2 "Ks" 0.9 "Kd" 0.8  
"roughness" 0.3  
"specularcolor" [1 1 1]  
"scale" 0.02 "nshades" 4  
"exponent" 2  
"graincolor" [0 0 0]
```

This shader makes objects look like they are carved out of grainular stone, like granite, by making "crystals" of varying intensity and colour. The parameter scale controls the size of the "crystals", or grains; larger values make larger grains. This is the only parameter that most users will want to change.

The parameter nshades is the number of unique intensity levels found in the

grains. Higher values of this will produce less "simplistic" looking stone. Setting nshades equal to 3 will produce stone that looks remarkably like the spattered-paint fake stone Zolatone. The exponent parameter controls the distribution of intensity levels; higher values push the intensities toward the darker end (more toward graincolor, as described below).

The "intensity" levels are actually levels of mixing two colours, the surface colour and the graincolor parameter. Since graincolor is black by default, the different colours normally are in fact different intensities of the surface colour. However, if you want red-and-green speckly stone for some reason, you could do this by setting the surface colour and graincolor appropriately, but you should probably set nshades to something pretty low to avoid getting lots of weird colours between red and green.

The parameters Ka, Ks and Kd have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. Roughness and specularcolor control the sharpness and colour of the specular highlight.

This shader can have problems with aliasing.

texmap

```
"texmap" "Ka" "Ks" "Kd" "roughness" "specularcolor" "texname" "maporigin"  
"xaxis" "yaxis" "zaxis" "maptype" "s1" "t1" "s2" "t2" "s3" "t3" "s4" "t4"  
  
"Ka" 1.0 "Ks" 0 "Kd" 1.0  
"roughness" 0.25  
"specularcolor" [1 1 1]  
"texname" "name of texture file"  
"maporigin" [0 0 0]  
"xaxis" [1 0 0] "yaxis" [0 1 0] "zaxis" [0 0 1]  
"maptype" 3 (ie. no projection)  
"s1" 0 "t1" 0 "s2" 1 "t2" 0  
"s3" 0 "t3" 1 "s4" 1 "t4" 1
```

This shader texture maps a surface. The name of the texture file is given by texname. The parameters maporigin, xaxis, yaxis, zaxis, maptype, s1-4 and t1-4 are passed directly

Note that because the t component of a texture map is displayed on a monitor as increasing downward, textures mapped onto surfaces can easily appear to be upside-down. You should be careful to orient your coordinate system correctly when using projections for texture mapping. For example, if you are using planar projection you may want to have the y axis of the projection plane pointing down.

The maptype parameter indicates the following types of projection:
0 planar,

- 1 cylindrical,
- 2 spherical,
- 3 no projection, and
- 4 automap.

In the case of spherical mapping the parameters `maporigin`, `xaxis`, `yaxis` and `zaxis` describe the coordinate system of the projection sphere. The `maporigin` is naturally the center point of the sphere. The `xaxis`, `yaxis` and `zaxis` are points describing the 3 coordinate axes relative to `maporigin`. The texture map wraps around the "equator" of the sphere such that the seam is located on the positive x-axis of the sphere.

The surface is texture-mapped as if it were painted with the image in the file. Normal shading techniques are then used to render the surface, as specified in the normal way.

The parameters `Ka`, `Ks` and `Kd` have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. Roughness and `specularcolor` control the sharpness and colour of the specular highlight.

txtplastic

```
"txtplastic" "Ks" "Kd" "Ka" "roughness" "specularcolor" "mapname"
```

```
"Ks" 0.5 "Kd" 0.5 "Ka" 1
"roughness" 0.1
"specularcolor" [1 1 1]
"mapname" "name of a texture file"
```

This shader is based on the "plastic" surface shader. The parameter `mapname` allows an image previously converted to a texture file to be mapped onto a surface.

wood

```
"wood" "Ka" "Ks" "Kd" "roughness" "specularcolor" "grain" "swirl" "swirlfreq"
"c0" "c1" "darkcolor"
```

```
"Ka" 1 "Ks" 0.4 "Kd" 0.6 "roughness" 0.2
"grain" 5 "swirl" 0.25 "swirlfreq" 1
"specularcolor" [1 1 1]
"darkcolor" [dependent on the surface colour]
"c0" [0 0 0] "c1" [0 0 1]
```

This shader creates a realistic-looking wood. The frequency of the wood grain can be changed with the `grain` parameter. The relative amount or amplitude of the turbulent swirl in the grain is controlled by the `swirl` parameter, and `swirlfreq` controls the frequency of this turbulence. Low values of `swirl` produce more uniform looking wood, while low values of `swirlfreq` make the wood

appear to be more knotty. Obviously these two parameters interact to a large extent. You should be careful not to set swirl too high or swirlfreq too low or the wood will become a jumbled mess.

The wood is simulated by creating a grain that is essentially composed of differently coloured concentric “cylinders” around a central axis defined by the two points c0 and c1. This axis is the z axis by default. Note that the orientation of this axis can be varied either by changing these two parameters or by doing some transformations between the call to the shader and the definition of the geometry. Either one of these approaches may make more intuitive sense in different applications.

The colour of the wood will normally consist of bands of different intensities of the surface colour. This is the most generally useful way of invoking the shader. However, for special appearances this can be changed by changing the darkcolor parameter, which controls the colour of the dark grain of the wood. The different intensity levels are actually levels of mixing between this colour and the surface colour, so setting the surface colour to red and darkcolor to white will produce red wood with white grain and various shades in between.

The parameters Ka, Ks and Kd have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. roughness and specularcolor control the sharpness and colour of the specular highlight. This shader can have problems with aliasing.

transparent_texture

```
"Ks" "Kd" "Ka" "roughness" "specularcolor" "texname" "traname"
```

```
"Ks" 0 "Kd" 1.0 "Ka" 1.0 "roughness" 0.1
```

```
"specularcolor" [1 1 1]
```

```
"texname" ["name of the texture file to be used for texturing"]
```

```
"traname" ["name of the texture file to be used for transparency"]
```

This shader uses two texture files. Like the shader texmap, the file associated with "texname" is used as a texture map – except that this shader provides no control over the type of projection used. The other texture file, associated with "traname", controls the transparency of the surface(s) to which this shader is assigned. The gray scale values of "traname" alter the level of transparency of the shaded surface. Black pixels of the image make the corresponding parts of a surface fully transparent while white pixels renders the surface opaque.

eroded

```
"eroded" "Ks" "Ka" "Km" "roughness"
```

```
"Ks" 0.4 "Ka" 0.1 "Km" 0.3
```

```
"roughness" 0.25
```

This shader erodes a "plastic-like" surface in such a way that parts of it are worn down to be transparent. The Km parameter controls the magnitude of the erosion.

LightSource Shaders

ambientlight "ambientlight" "intensity" "lightcolor"

```
"intensity" 1  
"lightcolor" [1 1 1]
```

An ambient light source supplies light of the same colour and intensity to all points on all surfaces

distantlight "distantlight" "intensity" "lightcolor" "from" "to"

```
"intensity" 1  
"lightcolor" [1 1 1]  
"from" [0 0 0] "to" [0 0 1]
```

Unlike an ambient source, a distant source casts its light in only the direction defined by the from and to parameters. Otherwise the output light is the same as an ambient light.

pointlight "pointlight" "intensity" "lightcolor" "from"

```
"intensity" 1  
"lightcolor" [1 1 1]  
"from" [0 0 0]
```

A point light source is the converse of a distant light. It radiates light in all directions, but from a single location. It has a from parameter, but no to.

spotlight "spotlight" "intensity" "lightcolor" "from" "to" "coneangle" "conedeltaangle" "beamdistribution"

```
"intensity" 1.0  
"lightcolor" [1 1 1]  
"from" [0 0 0] "to" [0 0 1]  
"coneangle" radians (30) "conedeltaangle" radians (5) "beamdistribution" 2
```

This shader reproduces the lighting effect of a spot light.

The intensity of the light varies from 0 (off) to any positive value (usually 1) representing the light at full intensity. The lightcolor parameter is an RGB triple representing the colour of light emitted by the source.

The from and to parameters specify the direction in which the light is shining. The coneangle and conedeltaangle parameters specify the distribution of the light as a cone-shaped beam, whose intensity falls off with the angle from the center to the cone. The falloff from the cone center is a "square-law" falloff (cosine of this angle raised to the power of 2) by default, but can be changed to a higher (or lower) power by setting the beamdistribution parameter.

pointnofalloff

```
"pointnofalloff" "intensity" "lightcolor" "from"
```

```
"intensity" 1.0  
"lightcolor" [1 1 1]  
"from" [0 0 0]
```

This is a point light shader without an inverse square intensity falloff. It is useful for lighting a scene when you don't want to go through the process of calculating the large intensity usually required to get a normal point light to look the way you want it. It is also useful for producing uniform lighting from objects inside a scene, without the "pooling" normally associated with point light sources.

In addition, some modelling systems, for example, work only with point light sources. In such a system some point lights may be placed far away from the scene to simulate distant lights, and this shader is a good choice in such circumstances.

The intensity of the light varies from 0 (off) to any positive value (usually 1) representing the light at full intensity. The lightcolor parameter is an RGB triple representing the colour of light emitted by the source.

The from parameter represents the position of the light source in space.

shadowdistant

```
"shadowdistant" "intensity" "lightcolor" "from" "to" "shadowname" "samples"  
"width"
```

```
"intensity" 1.0 "lightcolor" [1 1 1]  
"from" [0 0 0] "to" [0 0 1]  
"shadowname" (name of a shadow map)  
"samples" 16  
"width" 1
```

This is a normal distant light with an optional shadow map parameter given with shadowname. If a shadow map is not supplied the light behaves like a normal distant light source.

The parameter samples controls the sampling rate for filtering the shadow map. Higher values will produce less noisy-looking shadows, but will take

significantly longer. You can produce a (very noisy) test shadow very rapidly by setting samples to 1.

The width parameter controls "overfiltering" in the s and t directions. Higher values will give shadows more blurry edges, which can be used either as an effect or to hide the jagged edges or a low-resolution shadow map.

The intensity of the light varies from 0 (off) to any positive value (usually 1) representing the light at full intensity. The lightcolor parameter is an RGB triple representing the colour of light emitted by the source.

The from and to parameters specify the direction in which the light is shining.

shadowpoint

```
"shadowpoint" "intensity" "lightcolor" "from" "sfx" "sfy" "sfz" "sfnx" "sfny" "sfnz" "samples" "width" "shadowname"
```

```
"intensity" 1.0  
"lightcolor" [1 1 1]  
"from" [0 0 0]  
"sfx" "sfy" "sfz" "sfnx" "sfny" "sfnz" (6 shadow maps)  
"samples" 16  
"width" 1  
"shadowname"
```

This is a point light source that can cast shadows in all directions. To do this, you must supply 6 shadow maps, sfx, sfy, sfz and sfnx, sfny, sfnz for the positive and negative x, y and z directions respectively. This is very similar to the idea of creating an environment map from 6 cube-face images. If any of the cube faces are not supplied, the shader will behave as a normal point light in those directions. For best results, the field of view for shadow images should be greater than 90 degrees (95 recommended).

The parameter samples controls the sampling rate for filtering the shadow map. Higher values will produce less noisy-looking shadows, but will take significantly longer. You can produce a (very noisy) test shadow very rapidly by setting samples to 1.

The width parameter controls "overfiltering" in the s and t directions. Higher values will give shadows more blurry edges, which can be used either as an effect or to hide the jagged edges or a low-resolution shadow map.

The intensity of the light varies from 0 (off) to any positive value (usually 1) representing the light at full intensity. The lightcolor parameter is an RGB triple representing the colour of light emitted by the source.

The from parameter specifies the position of the light source in space.

shadowspot

"shadowspot" "intensity" "lightcolor" "from" "to" "coneangle" "conedeltaangle"
"beamdistribution" "shadowname" "samples" "width"

"intensity" 1.0

"lightcolor" [1 1 1]

"from" [0 0 0] "to" [0 0 1]

"coneangle" radians (30) "conedeltaangle" radians (5) "beamdistribution" 2

"shadowname" "name of the shadow file"

"samples" 16 "width" 1

This light is a spotlight with an optional shadow map parameter shadowname. If a shadow map is not used the light is a normal spotlight.

The parameter samples controls the sampling rate for filtering the shadow map. Higher values will produce less noisy-looking shadows, but will take significantly longer. You can produce a (very noisy) test shadow very rapidly by setting samples to 1.

The width parameter controls "overfiltering" in the s and t directions. Higher values will give shadows more blurry edges, which can be used either as an effect or to hide the jagged edges of a low-resolution shadow map.

The intensity of the light varies from 0 (off) to any positive value (usually 1) representing the light at full intensity. The lightcolor parameter is an RGB triple representing the colour of light emitted by the source.

The from and to parameters specify the direction in which the light is shining.

The coneangle and conedeltaangle parameters specify the distribution of the light as a cone-shaped beam, whose intensity falls off with the angle from the center to the cone. The falloff from the cone center is a "square-law" falloff (cosine of this angle raised to the power of 2) by default, but can be changed to a higher (or lower) power by setting the beamdistribution parameter.

Displacement Shaders

cloth

```
"cloth" "freq" "depth"
```

```
"freq" 500  
"depth" 0.02
```

This shader produces a cloth-like perpendicular weave pattern. The freq parameter changes the frequency of the "threads" (higher values mean the threads are closer together), and depth controls the height of the threads. The surface aliases pretty fiercely, but real cloth actually produces a somewhat similar effect, so it looks fairly realistic.

dented

```
"dented" "Km"
```

```
"Km" 1.0
```

This shader produces a dented surface. The amount of denting is controlled by Km.

diaknurl

```
"diaknurl" "maporigin" "xaxis" "yaxis" "zaxis" "freq" "depth" "width" "radius"  
"zmin" "dampzone"
```

```
"maporigin" [0 0 0]  
"xaxis" [1 0 0] "yaxis" [0 1 0] "zaxis" [0 0 1]  
"freq" 10 "depth" 0.25 "width" 0.05  
"radius" (refer to notes)  
"zmin" "zmax" (refer to notes)  
"dampzone" 0
```

This shader cuts a diamond knurl pattern into a cylindrical object. The parameters maporigin, xaxis, yaxis, and zaxis are used to do a cylindrical projection.

The freq parameter gives the number of grooves to cut per unit length along the z axis; higher values give closer grooves. The depth parameter controls the depth of the grooves, and the width parameter controls the width of the grooves. In order to render a correctly diamond-shaped pattern, the radius of the cylindrical object must be given with the radius parameter.

By default, the diamond knurl will be rendered along the entire length (along the z axis in shader space) of the cylinder. However, you can set minimum and maximum bounding z values with zmin and zmax parameters. The surface will not have knurl pattern cut outside these boundaries. In addition, you can make the knurl smoothly fade out instead of abruptly stopping at

these boundaries by setting the dampzone parameter. This parameter controls the width of the zone in which the depth of the grooves goes to zero. This zone is inside the zmin and zmax boundaries.

Remember to set the displacement bounds attribute when using this shader.

This shader can experience severe aliasing.

droop

```
"droop" "Km"
```

```
"Km"
```

This shader droops or sags a surface downward as if under the influence of gravity. 'Downward' for this shader means moving a surface in negative y.

emboss

```
"emboss" "Km" "texname"
```

```
"Km" 0.03
```

```
"texname" "name of a texture file that will control the embossing"
```

This shader embosses a surface according to an image given with the parameter texname. Pale areas of an image used for the texture file will push the surface "inward". The magnitude of the displacement is controlled by the parameter Km.

filament

```
"filament" "frequency" "phase" "width"
```

```
"frequency" 5.0
```

```
"phase" 0
```

```
"width" 0.3
```

This shader turns a cylinder into a spiral light-bulb filament. The filament can be brought to a point by applying the same shader to two cones at the ends of the cylinder. The frequency and phase parameters are identical to those of the "threads" shader.

sinknurl

```
"sinknurl" "maporigin" "xaxis" "yaxis" "zaxis" "freq" "depth" "zmin" "zmax"  
"dampzone"
```

```
"maporigin" [0 0 0]
```

```
"xaxis" [1 0 0] "yaxis" [0 1 0] "zaxis" [0 0 1]
```

```
"freq" 100 "depth" 0.005
```

```
"zmin" "zmax" (see notes)
```

```
"dampzone" 0
```

This shader cuts a sinusoidal knurl grooves along the length of a cylindrical object. The parameters `maporigin`, `xaxis`, `yaxis` and `zaxis` are used to do a cylindrical projection.

The `freq` parameter gives the number of grooves to cut around the circumference of the object. By default this number is quite high, but low numbers produce a shape like a classic Greek column. The `depth` parameter controls the depth of the grooves.

By default, the diamond knurl will be rendered along the entire length (along the `z` axis in shader space) of the cylinder. However, you can set minimum and maximum bounding `z` values with `zmin` and `zmax` parameters. The surface will not have knurl pattern cut outside these boundaries. In addition, you can make the knurl smoothly fade out instead of abruptly stopping at these boundaries by setting the `dampzone` parameter. This parameter controls the width of the zone in which the depth of the grooves goes to zero. This zone is inside the `zmin` and `zmax` boundaries.

Remember to set the displacement bounds attribute when using this shader.

This shader can have problems with aliasing.

threads

```
"threads" "maporigin" "frequency" "depth" "phase" "zmin" "zmax" "dampzone"  
  
"maporigin" [0 0 0]  
"frequency" 5 "depth" 0.1 "phase" 0  
"zmin" "zmax" (refer to notes)  
"dampzone" 0
```

This shader cuts a right-handed thread into a cylindrical object using the parameters `maporigin`, `xaxis`, `yaxis`, and `zaxis` to do a cylindrical projection.

The parameter `freq` gives the number of threads per unit length along the cylinder. The `depth` parameter controls the depth of the thread, and `phase` rotates the threads around the `z` axis of the cylinder. A value of 0 means no rotation and 1 means 360 degree rotation. This can be used to match threads from different cylinders.

By default, the threads will be rendered along the entire length (along the `z` axis in shader space) of the cylinder. However, you can set minimum and maximum bounding `z` values with `zmin` and `zmax` parameters. The surface will not have a thread pattern cut outside these boundaries. In addition, you can make the knurl smoothly fade out instead of abruptly stopping at these boundaries by setting the `dampzone` parameter. This parameter controls the width of the zone in which the depth of the grooves goes to zero. This zone is inside the `zmin` and `zmax` boundaries.

Remember to set the displacement bounds attribute when using this shader ie.

Attribute "bound" "displacement" [1.5]

This shader has problems with aliasing. The ShadingRate usually needs to be set to quite a low number.

Atmosphere Shaders

depthcue

```
"depthcue" "mindistance" "maxdistance" "background"
```

```
"mindistance" 0
```

```
"maxdistance" 1
```

```
"background" [0 0 0]
```

This atmospheric shader linearly adds the background colour according to the distance between the camera and a surface. No background colour is added if the surface is less than mindistance away. The background colour eliminates the surface colour entirely for points farther than maxdistance. In between, the two colours are mixed.

fog

```
"fog" "distance" "background"
```

```
"distance" 1
```

```
"background" [0 0 0]
```

This atmospheric shader is somewhat more realistic for emulating atmospheric absorption. It assumes that the attenuation of the surface colour in the fog is never complete, as it is in the depth-cue shader.

Appendix D

Project 1 Separating shape from shading

Overview

The purpose of this project is to explore the creative potential of displacement and texture maps; it is also intended to underscore the distinction between shape and shading – between the underlying geometry of an object and its outward visual appearance. Through this project you will learn how to

- create displacement and texture maps,
- control an abstraction called “parameter space”
- apply these maps to a quadric surface, and
- make a sphere as visually interesting as the surface of a **natural** object.

A simple object has been deliberately chosen for this project in order to focus your attention on controlling the process of shading, rather than shaping, an object.

You are to analyse two natural objects in terms of their surface attributes, these are to include such features as variations in colour, bumpiness, shininess (reflectivity), diffuseness and transparency. *In short, how they interact with light.* You will apply the characteristics of your chosen natural surfaces to two spheres. The natural objects, whose surfaces you are attempting to “re-create”, need not be spherical. Your images will be judged by the extent to which they portray spherical versions of the original objects – no matter how incongruent that might be, for example, a spherical banana, leaf or toe-nail!

Submission

Your submission will consist of two **sets** of files. Each rendered image must be accompanied by

- the RIB file you wrote to create the finished image, and the
- the original images used for the texture and displacement maps.

We will aim to create high resolution images suitable for recording to 35mm transparency film via our LFR film recorder. The fully rendered images should have an alpha channel and be saved in the TIFF file format with a resolution of 1166 by 800 pixels. All images should be LZW compressed; this can be done using RenderApp or PhotoShop.

It is left for you to decide the resolution of the TIFF images that you will use for the texture and displacement maps. In general each “source” image should not exceed 5MB in size; again these should be compressed using LZW. Do **NOT** submit the texture files.

The files relating to each sphere must be located in their own folder ie. at least three files in total. These should be placed in a folder clearly identified with your name and should be copied to the archive server.

Project 2 Combining the 'real' and the 'imaginary'

Overview

In this project you will combine a computer generated scene with a photographic image of the interior of a building or other architectural space. The computer generated scene should consist of a simple object or objects viewed by a virtual camera set to match the characteristics of the camera that recorded the photograph ie. position, orientation, focal length and f-stop.

The project will be completed in two separate phases. In the first, the objects in the synthetic scene will be modelled, viewed and the resulting image composited with the photograph. At this stage NO particular attention will be given to lighting the synthetic scene or assigning realistic surface attributes to the models. Emphasis will be on

- matching the viewpoint of the virtual and 'real' camera,
- matching the scale, placement and orientation of the models with the scene portrayed in the photograph.

In the second phase, light sources will be added to the synthetic scene in order to match the illumination of the models with the 'real' interior. Appropriate surfaces will be assigned to the models and the refined virtual scene will again be viewed and composited with the photograph. Compositing will be done using PhotoShop and the synthetic scenes will be created using hand written RIB files rendered with RenderMan.

The aesthetics underpinning the final composition can be anything from the literal to the sarcastic; the synthetic objects can be modelled to be entirely appropriate to their photographic "home" or may be startlingly incongruent with the context provided by the photographic image.

Background

The traditional separation between computer generated imagery and "live action" recording is now almost a thing of the past. The integration of imagery derived from the inner representations of a computer system and scenes captured through standard photographic techniques is challenging our notions of what is cinematically real and believable. What makes "special effects" special is often not so much what is added in the post-production process—a dinosaur here, or a dinosaur there—but by what is subtly altered or removed. This might be the judicious removal of items that were thought to be out of camera when the original footage was filmed. It might be the background, props or even the actors themselves that an editor deems superfluous to a scene.

Research is well underway on the generation of photo-realistic synthetic actors and more especially, as in the case of the research by Thalman et al, synthetic actresses.

Before the turn of the century movie stars, or indeed, any type of performer, may literally be even more pre-processed than the current crop of caricatured symbols emanating from the film, television and music industries.

With the advent of sophisticated VR systems, probably at the beginning of the next century, the propagation and wide acceptance of, perhaps even demand for, synthetic actors over 'real' ones will be paralleled on personal/home VR systems.

The spread and wide acceptance of synthetic actors compared to real ones may become driven not so much by what mass audiences wish to watch (passively) on film and video as by what they will become devoted to as a result of interacting with personal/home VR systems. In a 'media world' dominated by male fantasies, the enlightening promise of VR, like that of television before it, seems set to become subservient to financial returns and corporate market forces. For those in society who at one time or another are unable to 'test reality', the combination of seamless image synthesis and the immersive experiences of VR may lead to behaviour that will, by comparison, make the current debate about the link between criminality and explicit violence on the television and cinema seem like a trivial linguistic parlour game.

But how difficult is it to combine the real and the imaginary; the photographic and the synthetic? This project is designed to encourage you to explore the technical and aesthetic issues involved in synthesising imagery.

Readings

Nan-o-sex and Virtual Seduction
Siggraph Panel Session
Computer Graphics Proceedings of SIGGRAPH 93

Computer Graphics in Visual Effects – Course Notes Siggraph '93
Charles Gibson

Computer Graphics in Visual Effects, Cost Effective Special Effects
Section 4 – Course Notes Siggraph '93
Ricard Hollander

A Once and Future War
Jody Duncan
Cineflex issue 47, August 1991

An Integrated Control View of Synthetic Actors
D. Boisvert, N. Magnenat-Thalmann and D. Thalmann
New Advances in Computer Graphics
Proceedings of CG International 1989

Project 3 Three Dimensional Icons for a Graphical User Interface

Background

All modern interfaces use small pictograms called icons to help users interact with a computer system. The icons, generally much smaller than 100 pixels square, are used

- as a system of static **signage** to enable a users to orientate themselves within an imaginary electronic space, somewhat like highway signs and road markings,
- to dynamically provide a response, or feed-back, to support the users sense of **interaction**.

An icon as a *sign* can represent

- an **object**, such as a “recycler” for destroying files, fig 1,
- a **concept** or an abstraction – a body of knowledge such as an historical data base stored on a CD, fig 2,
- an **action**, for example a check box that controls colour printing, fig 3.

fig. 4

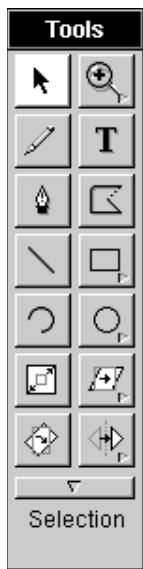


fig. 1



fig. 2



fig. 3 check box OFF

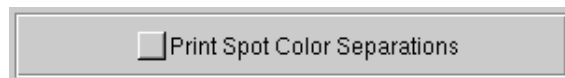


fig. 3 check box ON

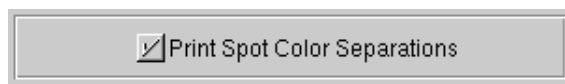


fig. 5 button UP

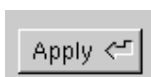
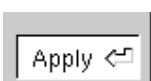


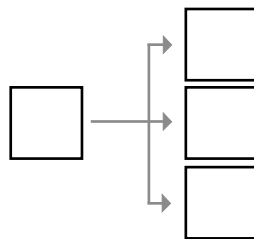
fig. 6 button DOWN



Icons are used extensively to represent storage devices, hard disks, diskettes and CD's, the directory structure of the file system and the files themselves. Often dozens of icons are used within a single application as 'covers' for buttons or the cells in tool palettes, fig 4. Such buttons and cells often have two graphics associated with them, one for the button's dormant state and the other to indicate to the user that some change has or is about to occur. For example, a button might normally have an icon that represents it's "UP" condition which is subsequently replaced by another icon to give the illusion of the button having a "DOWN" condition when a user presses it, figs 5 and 6.

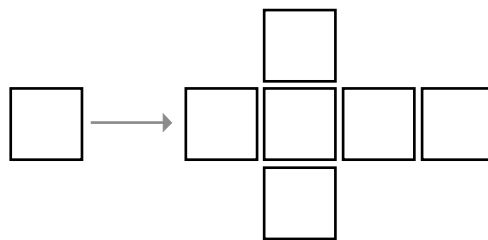
The majority of interfaces used at present are two dimensional even though, as can be seen by the figures on the previous page, there is often an implied depth. Interactive information systems, such as desktop computers, multimedia CD's, interactive television, information kiosks and "games" consoles will move away from using solely two dimensional elements in their user interface, such as 'flat' icons and buttons. Instead, three dimensional representations combined with various transformations (animations) will enliven, what many people now see as the bland uniformity that exists from one interface design to another.

The best designs will undoubtedly be those that use three dimensional graphics in a way that improves communication rather than those that have gratuitously applied 3D techniques. For example, an icon that represents some information on a hard disk might, when selected by a user, undergo a 3D transformation that conveys an animated "explanation" of what it represents, rather than, as is now the case, merely undergoing a colour inversion to show it has been selected. How often, when faced with the task of learning a new software package, have you wondered about the function a particular item in a tool palette? Using animated graphics, the icons in a palette might "act-out" their function, or divide or otherwise "decompose" into several more easily understandable 'sub-icons'.



Sub-division is one way in which an icon could decompose to provide more information to a user. For example, an icon representing the category "New Zealand" on a CD dealing with "Tourism in the Pacific" might divide into two or three sub icons representing, "sights and sounds", "adventure" and "relaxation", when selected by a user. The area of interface design posed several problems. For example, can 3D graphics help in sustaining an illusion that users are entering an informational space and not just browsing a computerised version of a travel agents notice board?

If the icon, “New Zealand”, was three dimensional it’s way of decomposing might be to “unwrap” as if it were a box being opened out flat. But should each of the new sub-icons be three dimensional? If so then what is to prevent the user becoming hopelessly lost? What conventions should be adopted for the icons – how many levels of decomposition can the user be expected to keep track of. How can a user be certain that an icon has no further sub-levels and what visual cues should be provided to enable a user back-track to a previous level? After all the icons only represent the information, they are not the information themselves.



Design Brief

This third and final project of the course is concerned with employing 3D computer graphics animation to some element of a real or imaginary graphical user interface. You are to design and animate a 3D icon(s). However, because the project is primarily concerned with 3D computer animation and not the issue of interface design as such, you are NOT required to design an interface in which the icon(s) could be used. However, you are strongly advised to be mindful of the questions raised in the previous paragraph.

There are no restrictions on the colour depth of your graphics ie. you may break the artificial colour barriers that are normally imposed on this work, and the content of what your icon, or system of icons represent is entirely your decision. However, the following restrictions apply,

- maximum screen space is 300x300 pixels – look upon this as your “stage” and the icons as the “actors”,
- maximum duration of any single animated segment such as a transformation, a transition or other effect is 50 frames.

Submissions,

- at least ONE “demo” animation of your 3D icon/button in the form of a QuickTime movie,
- all texture, transparency, displacement maps and rib files used in the production of the animation(s), and finally
- an A3 “concept board”, mounted or unmounted, that shows the development of your ideas.

More specific details about the time and place of the submission will be made available later.