Angel
V0.7.4

Ian Stephenson
DCT Sytems
www.dctsystems.freeserve.co.uk

October 1, 2003

Chapter 1

Introduction

Angel is a RenderMan[1] compatable renderer. While not intended as a production renderer, it includes many features typically found in top end production renderers, such as procedural shading, and support for complex surface types. In addition to these features Angel acts as a testbed for rendering, and includes many unique capabilities, like proceudural geometry, stereo rendering, and anti-aliased noise.

Except where noted Angel should behave as described in the RenderMan standard — it reads in RIB files and writes out images files. In this document we therefore restrict discussion to the idiosyncrases of Angel. For a more general introduction to RenderMan we recommend “Essential RenderMan Fast”.

You may use Angel for whatever purposes you choose, provided you notify DCT systems. Please let me know what you’re using Angel, and its related tools for. Bug reports and any images rendered with Angel are always greatfully received.

[1]RenderMan is a trademark of Pixar.

Chapter 2

Installation

Angel can be downloaded for a range of platforms from www.dctsystems.freeserve.co.uk. The standard download is simply a gziped binary for the platform you have selected. To install it simply gunzip it, rename it if necessary, make it executable (`chmod 755 angel`), and place it in your path.

If you have BMRT installed on your system then angel should now work correctly without further action, as Angel is intended to be BMRT compatable. However if BMRT is not installed you will need to install the necessary shaders. A tar file containing these is also on the DCT website. This should be unpacked. Having done this you need to set the environment variable SHADERS to point to that directory. You should now be able to render using the command `angel xxx.rib`.

To compile shaders you can either use BMRT's slc command, or download Angel's ”giles” shader compiler. Giles should be installed in the same fashion as angel. To operate Giles requires that you have a working version of cpp installed and in your path. On most unix machines, a version of cpp is installed by default. On win32, a version is supplied with BMRT which works well.

Chapter 3

Implementation Specific Information

The RenderMan standard leaves some implementation details unspecified. This section outlines how some of these features should be used with Angel

3.1 File Formats

Angel can read and write TIFF, JPEG and EXR files. The format used is automatically determined from the filename. These can be used for textures, environment maps and final output.

Most of the map making commands are not implemented, as standard file formats are used instead. Environment maps should be standard texture files, painted in a polar format.

The exception to this is shadow maps. These can be generated using either the MakeShadowMap command, or the ”shadow” output driver.

3.2 Motion Blur

Transformation Motion Blur is supported. However only two sample times are considered, which should co-incide with the shutter open and close times. Interpolation is linear. Deformation, and muli-segement blur are not supported.

Motion blur is implemented by a form of interlacing, Quality can therefore be improved by increasing the Y pixelsamples.

3.3 Light Shaders

Light shaders are not supported, but the standard shaders are supported, along with “arealight”, and “attenlight”.

3.4 Area Lights

Angel supports area lights on Parametric surfaces.

3.5 Depth of Field

Large Depth of Field blurs can take a horrendous amount of time to post-process. On the plus side, they look great!

3.6 Gamma Correction in Display Driver

When the “framebuffer” output driver is used, the output image will be gamma corrected prior to display. The amount of correction can be controlled by the GAMMA environment variable.

On Unix platforms, the default value is 1.0 — ie no correction, as most X servers allow gamma to be set using the “xgamma” command. However as most win32 systems do not allow the gamma of the display to be set, on win32 platforms the default value of Gamma is 2.0. Refer to the website for more information on gamma correction.

This is in addition to any correction which may have been applied by the exposure command.

Chapter 4

Features of Angel

As Angel has been developed, I’ve added a number of features and tricks that I thought were interesting. These are documented in this chapter.

4.1 Ray Tracing

Though primarily a z-buffer renderer, Angel supports tracing of secondary rays, allowing the easy creation of accurate reflections, and shadows. Ray tracing of reflections is accessed using the trace function from shading language.

To create ray traced shadows, simply set the shadow map name to be ”raytrace”, for any shadow casting light (shadowdistant, shadowspot, shadowpoint or arealight).

The visibility of objects to various kinds of rays is controlled by the visibility attribute:

```
Attribute "visibility" "camera" [1]
Attribute "visibility" "trace" [0]
Attribute "visibility" "photon" [0]
Attribute "visibility" "transmission" ["opaque"]
Attribute "visibility" "transmission" ["Os"]
Attribute "visibility" "transmission" ["shader"]
Attribute "visibility" "transmission" ["transparent"]
```

This is compatable with the ray tracing and GI elements of the PRMan 11 API. Note that by default objects are invisible to traced rays.

In the same way that shadow maps require a shadow bias, a ray tracing bias can be set with the arribute.

```
Attribute "trace" "bias" [0.01]
```

The BMRT fulltrace function, is partially implemememented (along with some other aspects of the BMRT ray tracing API), and most shaders which use it should produce some usefull results. However this is obsolete. Angel will in future implement the PRMan 11 ray tracing API

Improving Ray Tracing Performance

Remember that Angel is primarily *not* a ray tracer. Consider using environment or shadow maps — they're faster even in a fully optimised ray tracer.

Performance can be improved by carefull setting of the visibility attribute. Only objects which need to be seen in reflections should have their visibility to traced rays enabled. Turn off shadows from any objects which cannot or need not cast them. You should also limit the maximum ray depth using the command:

```
Option "trace" "maxdepth" [1]
```

You should avoid setting transmission to be ”shader” if at all possible, as this typically requires multiple shaders to be run for each shadow ray, and can spawn further rays. Using ”opaque” us *much* faster, and ”Os” is somewhere between the two.

If you must run shaders to calculate opacitiy for shadow rays, note that the shading engine only needs to extract Oi, and not Ci. If you can set Oi earlier in you shader then the shader need not be run to completion. It is particularly important that you assign Oi before interogating any lights and if possible before tracing any reflections, as these results in a cascade effect forcing the renderer to evaluate other shaders, which in turn run others.

Don't trace reflections from the backs of objects. Just as you might use backface culling to optimise scanline rendering, shaders can be optimised to ovoid having useless ray traces bouncing around inside objects. Using the code:

```
if(N.I<0)
Cr=trace(P,R);
else
Cr=0;
```

prevents the backs of objects having reflections which may not be visible. This is particularly important because of the aforementioned cascade effect. The ray you would have traced could have spawned many subsiquent rays.

Hackers Only: The shading engine actually shades a small area of the surface for each point. This is needed for derivative and area calculations. Unfortunatly all of the points within the area must be shaded, and can generate secondary rays. This is avoided by the SLC instruction ”return_area”. Unfortunatly the shading compiler isn't smart enough to generate these in the optimal place. By manually inserting a return_area intruction into the SLC file at an appropriate point (ideally before any trace or lighting commands) performance can be signifigantly improved.

Even with a more inteligent compiler speed ups might be possible by re-ordering the code to place derivative calculations at the begining of the SL code.

Though the shading engine is generally very efficient, it can be slowed down considerably when ray tracing surfaces whose shaders are stored on a remote server.This should be dramatically improved in 0.7.4 (non win32 versions).

4.2 Global Illumination

Angel implements global illumination through the use of Photon Maps. Two types of map are supported — Global and Caustic. Global includes all illumination, while Caustic only records specular paths from the light source. Both are generated by using the ”photon” Hider:

```
Hider "photon" "emit" [100000]
 "globalmap" ["global.pmap"]
 "causticmap" ["caustic.pmap"]
```

This is used in a seperate pass, which creates one or two maps. By default 100000 photons are created, though the number of photons in the final maps, can vary dramatically according to the scene used. Currently shaders are not used during this pass, rather a default shading model is used. However the shader parameters Kd, Ks, Kt and eta will be used if they are explicitly set in the RIB file (shader programmed defaults are not used).

Remember that by default objects are not visible to Photon rays. You use set the visibility attribute as described in the ray tracing section.

To access a map, simply use the shadeop:

```
indirectLight=photonmap("file.pmap",P,N);
```

Distant lights are currently not included in Photon maps.

4.3 Procedural Geometry

The RiGeometry call is implemented by calling a shader of type geometry - this can read u,v,du and dv and must calculate P and N. For example figure 4.1 shows how a superquad can be defined. To use this in a RIB file simply add the command `Geometry "superquad"`.

4.4 Procedural Projections

Projections are simply special cases of transformation shaders. These must take a point P in 3D camera space and transform them into 2D space. Figure 4.2 shows a standard perspective transform coded as a shader. A custom shader can be used as a parameter to the Projection command.

4.5 Stereo Rendering

If the Hider is set to ”stereo” then two images will be written at once, giving a left and right eye view. This has several advantages over rendering twice, as the image is shaded only once. The distance between the two views can be controlled by the hiders ”seperation” parameter.

Tools for combining these images into red/green stereograms are available on the DCT website.

```
geometry superquad( float east=1;
float north=1;
)
{
float uu,vv;
float cv,cu;
float sv,su;

uu=(u-0.5)*2*PI;
vv=(v-0.5)*PI;

cu=cos(uu);
cu=(cu<0)? -pow(-cu,east) : pow(cu,east);

cv=cos(vv);
cv=(cv<0)? -pow(-cv,north) : pow(cv,north);

su=sin(uu);
su=(su<0)? -pow(-su,east) : pow(su,east);

sv=sin(vv);
sv=(sv<0)? -pow(-sv,north) : pow(sv,north);

P=point (cu*cv,su*cv,sv);

N=calculatenormal(P);
}
```

Figure 4.1: A SuperQuad Shader

```
transformation myPerspective(float fov=90)
{
float fovFix=1/tan(fov/360*3.14159265);
P=point(
xcomp(P)*fovFix/zcomp(P),
ycomp(P)*fovFix/zcomp(P),
zcomp(P));
}
```

Figure 4.2: A Perspective Projection

4.6 Dump Meshes Hider

If the Hider is set to dumpmeshes, then rather than rendering an image, each shading grid will be written to a text file. The format of the file is the size of each grid, followed by the position, colour and opacity of each point in that grid.

This information can easily be used for deep shadow generation.

4.7 INoise

An Anti-aliased noise function is provided in the shading language. It is only available in 1 and 2 dimmensional versions, and returns a float.

`float Inoise(float); float Inoise(float,float);`

4.8 Illuminate

The illuminate command has been extended so that in rather than take a boolean value, which turns the light on or off, a float point value can be supplied to control the lights brightness on a per object basis.

4.9 OpenEXR Support

Angel on Unix platforms supports the OpenEXR file format. If can be used for both textures, and as an output format. To use it constructivly for output you should disable output quantisation by using the rib command: `Quantize "rgba" 0 0 0`

Chapter 5

Limitations of Angel

Though Angel supports many advanced features, it has a number of limitations. Most of these are minor, and are documented here so you can work around them. All of the issues detailed here could be considered bugs, and should really be fixed.

5.1 Clipping and Culling

Recent versions of Angel include more aggressive culling code. This should dramatically reduce render times for complex scenes. However as this is a new feature, it may still have problems. If you encounter problems which appear to relate to culling, then please report them.

5.2 Opacity

Angel is a Z-Buffer renderer. While this has many advantages, it handles transparant surfaces very poorly. Coloured Opacity is not supported at all.

The image can be improved by increasing PixelSamples. A better solution should be available in a furture release.

5.3 Varying Parameters

Varying parameters to surfaces are not supported. The exception to this (aside from P!), is the width parameter of particles which is supported.

5.4 UV's on Polygons

The UV coordinates of polygons are incorrect. While they should operate correctly for most purposes, they may not be continuous across a polyon, and are not implemented as defined in the RenderMan standard.

5.5 Blobby Objects

Angel has a limited implementation of Blobby Objects. However the results are poor, and not all instructions work correctly.

5.6 Subdivision Surface

Subdivision Surface are rendered as their control hull.

5.7 SL Variable Types

Support for Matrix and Array variable types is inconsitant. While some parts of the system have good support for these structures, others do not support them.

5.8 Giles

The Giles shader compiler does not support user defined functions.

5.9 Win32

Though a lot of angel users are on windows platforms, supporting some of the more advanced features of Angel on win32 is difficult or even impossible. Currently the following features are not supported on win32:

- RunProgram procedurals
- DynamicLoad procedurals
- OpenEXR file support

5.10 MacOS X

Support for MacOS X is new as of release 0.7.3. Though the unix core of Darwin enures that most of Angel should preform well under MacOSX, there are currently a few limitations:

- There is no display driver
- RunProgram procedurals seem to be unreliable

As a new platform, and feedback on the MacOS port would be greatly appreciated.