


Sunflow Wiki PDF



navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help

search

toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link

page | discussion | view source | history

Main Page

****NOTICE**** As Sunflow has been in a frozen state for many years (with even the forum no longer functioning), the need for a wiki to capture dynamic changes and additions to Sunflow knowledge is no longer necessary. A static pdf user guide would certainly fit the current need and I intend to convert the wiki into a pdf guide available for download and subsequently retire the wiki. If you would like the wiki database and images in it, I can certainly provide them to anyone who wishes so that they may replicate it elsewhere.

Welcome to the unofficial Sunflow wiki. This is my attempt to really start understanding Sunflow by having all our collective Sunflow knowledge in one place. But what, you might ask, is Sunflow? Sunflow is an open source rendering system for photo-realistic image synthesis. It is written in Java and built around a flexible ray tracing core and an extensible object-oriented design.

Due to lots of vandalism, not the least of which came from "research" by [Brown University](#), I'm turning off the registration feature. If you would like to edit the pages [contact me](#) directly and I'll set you up an account. Once an account is set up feel free to log in and go for it, but be sure to check out the [wiki style guide](#) first!

Sunflow User Documentation

Scene File (.sc) Syntax

- [Notes On the .sc Format](#)
- [Image Settings](#)
- [Shaders](#)
- [Cameras](#)
- [Lights](#)
- [Global Illumination](#)
- [Mesh Objects](#)
- [File Meshes](#)
- [Primitives](#)
- [Instances](#)
- [Modifiers](#)

Scene File (.sca/.scb) Syntax

- [Waiting For Release](#)

Support

- [Sunflow Home Page](#)
- [Sunflow Forum](#)

How To's

- [Installing Sunflow](#)
- [Compiling Sunflow](#)
- [Getting Resolutions Greater Than 16K](#)
- [Command Line Flags](#)
- [Rendering An Animation](#)
- [Lightmap Baking](#)
- [Output Different Image Types](#)
- [Alpha Maps](#)
- [Sunflow FAQ](#)

Logo Pack

- [Logo Pack \(.zip\)](#)

Exporters

- [Exporters](#)

Special builds

- [Windows: 0.7.3 Compiled W. Java Native Jet Compiler](#)

Beware!

- The file `sunflow.sh` in the 0.7.2 release has errors. Instead, use [this one](#)

Integrations

- [Helios: Distributed Rendering](#)
- [Moonlight3D](#)
- [P5Sunflow \(Processing library\)](#)
- [Wirefusion](#)
- [B-flow - Direct Blender Integration](#)

The Sunflow wiki was fantastic. I enjoyed building and hosting it and always like collecting information so I only have to go to one place to get it. Since Sunflow stopped being developed years ago, the wiki has never been updated, and so its dynamic editing nature is not needed. Additionally, maintaining the Wikimedia backend to ensure security patches are in place is time consuming. Still, there is traffic to the Sunflow wiki so rather than making all the information and effort put into it disappear; I have converted all the pages into a pdf. Where there are code blocks that extended beyond the page and the cut-off code is vital to its meaning or there are important files that were once linked to in the wiki, I have added these as attachments at the end of the document. There are bookmarks in the pdf to help navigate to the content desired.

Main Page

From Sunflow Wiki

Welcome to the unofficial Sunflow wiki. This is my attempt to really start understanding Sunflow by having all our collective Sunflow knowledge in one place. But what, you might ask, is Sunflow? Sunflow is an open source rendering system for photo-realistic image synthesis. It is written in Java and built around a flexible ray tracing core and an extensible object-oriented design.

Due to lots of vandalism, not the least of which has come from Brown University (<http://graffiti.cs.brown.edu/info/>) , I'm turning off the registration feature. If you would like to edit the pages contact me (<http://www.geneome.net/contact/>) directly and I'll set you up an account. Once an account is set up feel free to log in and go for it, but be sure to check out the wiki style guide first!

Sunflow User Documentation

Scene File (.sc) Syntax

- Notes On the .sc Format
- Image Settings
- Shaders
- Cameras
- Lights
- Global Illumination
- Mesh Objects
- File Meshes
- Primitives
- Instances
- Modifiers

Scene File (.sca/.scb) Syntax

- Waiting For Release

Support

- Sunflow Home Page (<http://sunflow.sourceforge.net/>)
- Sunflow Forum (<http://sunflow.sourceforge.net/phpbb2/>)

How To's

- Installing Sunflow (<http://sunflow.sourceforge.net/index.php?pg=docs&doc=1>)
- Compiling Sunflow

- Getting Resolutions Greater Than 16K
- Command Line Flags
- Rendering An Animation
- Lightmap Baking
- Output Different Image Types
- Alpha Maps
- Sunflow FAQ (<http://home.comcast.net/~gamma-ray/sf/sunflow-faq.htm>)

Logo Pack

- Logo Pack (.zip) (http://www.geneome.net/other/otherfiles/SunFlow_Logo_2007.zip)

Exporters

- Exporters

Special builds

- Windows: 0.7.3 Compiled W. Java Native Jet Compiler (<http://www.polyquark.com/opensource/>)

Beware!

- The file `sunflow.sh` in the 0.7.2 release has errors. Instead, use this one (<http://www.geneome.net/drawer/sunflow/sunflow.sh>)

Integrations

- Helios: Distributed Rendering (<http://sfgrid.geneome.net/>)
- Moonlight3D (<http://www.moonlight3d.eu/cms/>)
- P5Sunflow (Processing library) (<http://hipstersinc.com/p5sunflow/>)
- Wirefusion (<http://nemoinfo.free.fr/stage/truc.html>)
- B-flow - Birect Blender Integration (<http://code.google.com/p/b-flow/>)

Sunflow Developer Documentation

Projects

- Sunflow Projects
- 0.07.2 Java Docs (<http://sunflow.sourceforge.net/docs/javadoc/>)
- 0.07.3 Java Docs (SVN 12/9/07) (<http://www.geneome.net/other/otherfiles/SunflowSVNDocs/>)
- Sunflow Change Log

Lessons

- API Tips

- Matrix Notation
- Benchmarks

Wiki Style Guide

- Wiki Style Guide

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Main_Page"

- This page was last modified on 7 July 2012, at 01:44.

SC Notes

From Sunflow Wiki

Contents

- 1 Order Of Code Blocks
- 2 Include
- 3 Commenting Out Lines
 - 3.1 Scene File
 - 3.2 Java File Or Janino

Order Of Code Blocks

It's important to keep track of the order of the code blocks in cases of shaders and instances because the scene file parser reads these code blocks in order. If the order of the blocks isn't right, for example an object with a certain shader is read before the shader block, the object won't get the shader because when Sunflow read the object's shader it didn't know that shader since it hadn't seen it yet. Similarly with instances, the object needs to be read first before the instance of that object is defined. Here's the order I suggest:

- Image Settings
- Lights
- Shaders
- Modifiers
- Objects
- Instances

Include

You can include other .sc files to be used along with your original scene file by adding the following line to the original scene file:

```
include myOtherScene.sc
```

This helps when you have some big files that are hard to manage (mesh files) or if you have scene settings that you know will always be constant and you don't want to accidentally change.

Commenting Out Lines

If you want to not have something in your scene to work or show up you can always comment out sections of the scene file rather than delete blocks of code.

Scene File

There are 2 ways to do this in the scene file:

```
%Comment out a single line with a "%" sign.
```

```
/* Comment out a single line  
or many lines  
with a "/*" and "*/"  
at the beginning and end  
of your code */
```

Java File Or Janino

In java, There are 2 ways to do this:

```
//Comment out a single line with "//".
```

```
/* Comment out a single line  
or many lines  
with a "/*" and "*/"  
at the beginning and end  
of your code */
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=SC_Notes"

- This page was last modified on 9 January 2008, at 18:46.

Image

From Sunflow Wiki

Contents

- 1 Image Block
 - 1.1 Anti-Aliasing
 - 1.2 Samples
 - 1.3 Contrast Threshold
 - 1.4 Filters
 - 1.5 Jitter
- 2 Bucket Size/Order
- 3 Alpha Channels
- 4 Banding
- 5 Tone Mapping

Image Block

The image block dictates the resolution of the image as well as a few other image options. The samples and jitter lines are optional and if they are left out samples will equal 1, contrast will be 0.1, and jitter will be false will be used as the default.

```
image {  
  resolution 800 600  
  aa 0 2  
  samples 4  
  contrast 0.1  
  filter gaussian  
  jitter false  
}
```

Anti-Aliasing

The aa line is the settings for the adaptive anti-aliasing. These control the under/over-sampling of the image. The image will be first sampled at the rate prescribed by minimum aa value (the first value). Then, based on color and surface normal differences, the image will be refined up to the rate prescribed by the maximum aa value (the second value). Taken from the Sunflow ReadMe file:

A value of 0 corresponds to 1 sample per pixel. A value of -1 corresponds to 1 sample every 2 pixels (1 per 2x2 block) A value of -2 corresponds to 1 sample every 4 pixels (1 per 4x4 block) A value of -3 corresponds to 1 sample every 8 pixels (1 per 8x8 block) ... A value of 1 corresponds to 4 samples per pixel (2x2 subpixel grid) A value of 2 corresponds to 16 samples per pixel (4x4 subpixel grid) A value of 3 corresponds to 64 samples per pixel (8x8 subpixel grid) ...

Examples:

```
- quick undersampled preview:      -2 0
- preview with some edge refinement: 0 1
- final rendering:                  1 2
```

You can turn adaptive anti-aliasing off by setting the min and max values to the same number. For example, an aa of 0 0 will be 1 sample per pixel with no subpixels.

You can see a video on explaining what adaptive anti-aliasing is here (.zip)
(<http://www.geneome.net/other/videotutorials/AdaptiveSamplingInSunflow-XviD.zip>).

Samples

Samples are the number of samples. Surprised? When used they indirectly affect many aspects of the scene but directly affects DoF and camera/object motion blur.

Contrast Threshold

There is a line in the image block in which you can change the default contrast threshold. This affects the point at which the renderer decides to adaptively refine between four pixels when doing adaptive anti-aliasing. This line isn't required and the default is usually the right setting. I personally don't see a change in the render with different values.

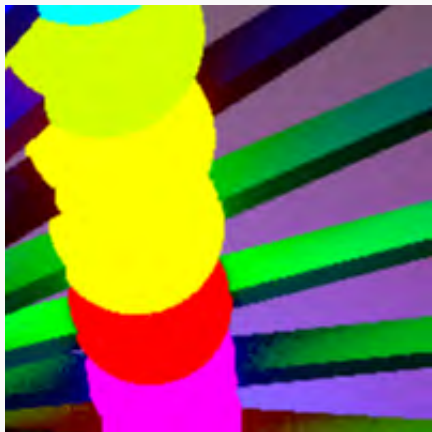
Filters

If you are oversampling the image (i.e., having the min and max aa positive numbers) you will likely want to use more advanced filters to control the look of the image. The available filters are:

```
box (filter size = 1)
triangle (filter size = 2)
gaussian (filter size = 3)
mitchell (filter size = 4)
catmull-rom (filter size = 4)
blackman-harris (filter size = 4)
sinc (filter size = 4)
lanczos (filter size = 4)
bspline (in the 0.07.3 SVN) (filter size = 4)
```

Check this page (http://gardengnomesoftware.com/dll_patch.php) out for a good overview of filters. Triangle and box are better for previews since they are faster. The other filters are recommended for final image rendering (my personal favorite is mitchell).

Sunflow Filter Examples



box



triangle



gaussian



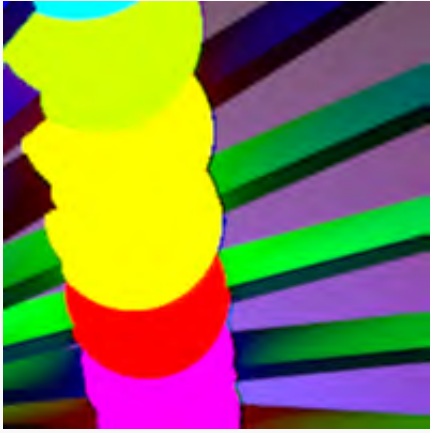
mitchell



catmull-rom



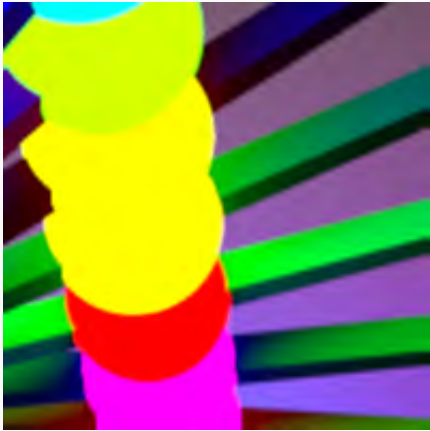
blackman-harris



sinc



lanczos



bspline

Jitter

The jitter line (either true or false) turns jitter on or off. Jitter moves the rays randomly a small amount to reduce aliasing that might still be present even when anti-aliasing is turned on. So jittering these pixels makes the aliasing issues become less perceptible. Jitter should be turned off (false) when doing animations because the randomness of jitter makes it very obvious it's on when moving from frame to frame.

Bucket Size/Order

You change the bucket order in the scene file just like you can in the command line. In the scene file, if you want to change the order from the default hilbert (for which you don't need to add any line) you would type the following as it's own line in the .sc file:

```
bucket 64 column
```

If you want a reverse order it would look like this:

```
bucket 48 "reverse spiral"
```

A larger bucket size means more RAM usage and less time rendering. Usually, a bucket size 64 is a good default - especially if you are using a wide pixel filter like gaussian or mitchell. There are six bucket order types available: hilbert (default), spiral, column, row, diagonal, and random. You can also use these ordered in reverse by adding "reverse" in front of the name. To use reverse, you'll need to use quotes around the reverse order so that the bucket order is still parsed as one token. The number in the line is the pixel size of the each bucket. There is some clamping done internally to make sure the bucket size isn't too small or too big.

Alpha Channels

The SVN 0.07.3 version of Sunflow create alpha channels when exporting to png or tga. Alphas will be recognized in textures in future releases.

Banding

In some cases, and rendered image with a somewhat uniform color will appear to have a banding affect and not be a smooth gradient. The issue is due to the 8 bits per channel used in the rendered image causing "mach banding" (an effect of the human eye) that happens just when values jump from one integer to the next after staying constant for several pixels. In this case the only solution found so far is to use dithering (which only seems to be found in Photoshop) when converting to 24 bit RGB.

Tone Mapping

If you find yourself looking to tone map or you intend to tone map your image before it's even rendered, you may want to consider this tip. Render a bigger image (at least 2x in each direction), then you can reduce the number of samples since the resize (to your originally intended size) will blur the noise. This will avoid having the problem of the high intensity pixels filtering too far out when Sunflow does its filtering. Ideally you would render with aa samples set to 0 0 (fixed at 1 sample per pixel) and a really big image. Then do your tonemapping on that big image and then you can resize to final resolution. The OpenEXR output driver is probably the better choice (vs .hdr) if you want to do this since it can handle arbitrary image sizes (it writes a bucket at a time).

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Image>"

- This page was last modified on 7 March 2009, at 21:21.

Shaders

From Sunflow Wiki

Contents

- 1 Introduction
- 2 Color Spaces
- 3 Shader Override
- 4 Samples
- 5 Shaders
 - 5.1 Constant Shader
 - 5.2 Diffuse Shader
 - 5.2.1 Textured Diffuse Shader
 - 5.3 Phong Shader
 - 5.3.1 Textured Phong Shader
 - 5.4 Shiny Shader
 - 5.4.1 Textured Shiny Shader
 - 5.5 Glass Shader
 - 5.5.1 Absorption
 - 5.5.2 Trace Depths
 - 5.5.3 Caustics
 - 5.6 Mirror Shader
 - 5.7 Ward Shader
 - 5.7.1 Textured Ward Shader
 - 5.8 Ambient Occlusion Shader
 - 5.8.1 Textured Ambient Occlusion Shader
 - 5.9 Uber Shader
 - 5.10 Command Line Shaders
- 6 Janino Shaders
 - 6.1 Mix Shader
 - 6.2 Fresnel Shader
 - 6.3 Stained Glass
 - 6.4 Shiny Shader with Reflection Mapping
 - 6.5 Simple SSS
 - 6.6 Specular Pass
 - 6.7 Translucent Shader
 - 6.8 Wire Shader
- 7 Source Code Shaders
 - 7.1 True SSS
 - 7.2 Textured SSS

Introduction

After a post asking for help regarding texturing shaders, I thought I would go over each of the available shaders in Sunflow 0.07.2.

For the bump, normal, and perlin (in the 0.07.3 SVN) modifiers, these aren't a part of shaders but rather have their own modifier syntax. For examples of how to use modifiers, see the modifiers page.

Color Spaces

Sunflow has 6 color spaces available to use in your shaders. On a few of these (internal, sRGB, and XYZ) you can increase the color values beyond 1.0 if you really want to saturate the effect.

- internal - requires 3 values
- sRGB nonlinear - requires 3 values
- sRGB linear - requires 3 values
- XYZ - requires 3 values

SVN 0.07.3 only

- blackbody - requires 1 value (temperature in Kelvins). See this page (<http://en.wikipedia.org/wiki/Blackbody>) for a good image of the color range.
- spectrum [min] [max] - any number of values (must be >0), [min] and [max] is the range over which the spectrum is defined in nanometers. This defines a spectrum from wavelengths X nm to Y nm with a number of regularly spaced values. Note that this spectrum will simply get converted to RGB, there's no spectral rendering in Sunflow at the moment.

Here are examples of each using the diffuse shader's diff line:

```
diff 0.8 0.8 0.2
diff { "sRGB nonlinear" 0.8 0.8 0.2 }
diff { "sRGB linear" 0.8 0.8 0.2 }
diff { "XYZ" 0.8 0.8 0.2 }
diff { "blackbody" 1200 }
diff { "spectrum 1 500" 3 6 9 190 }
```

Keep in mind that for colors the syntax for sRGB nonlinear color space (http://en.wikipedia.org/wiki/SRGB_color_space) is used in the shader examples below. These values are what most of us are used to.

Shader Override

The name of these shaders are variable, so you can name them whatever you like. Names of the shaders are also used in "shader overriding." You can use any shader listed in your scene file (it doesn't have to be applied to an object) and with a single line make everything in your scene use that shader. To override, add this line to the scene file:

```
override shaderName true
```

If you want to scene to then be rendered as usual, simply comment out the line:

```
%override shaderName true
```

Samples

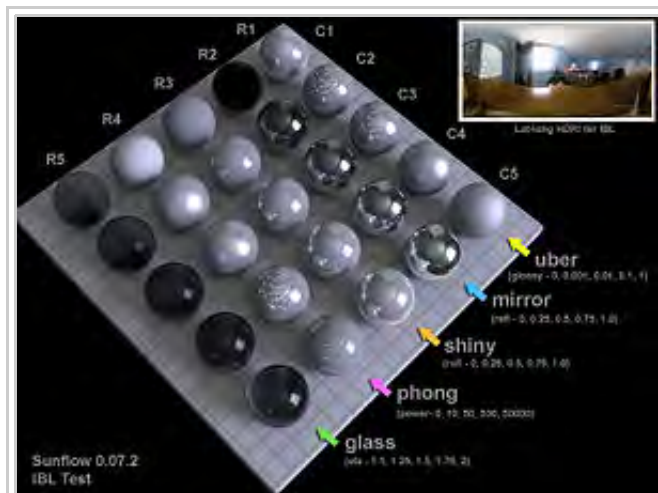
Samples, samples, samples. Samples are key for those shaders whose reflection quality are sample driven. If they are too low, the reflections will look bad. If they are too high, your render will take forever. So it's important to experiment to get the right setting for your scene. I suggest starting with low samples, and working your way up till you reach just the right amount. I usually like a sample number of 4 for most of the shaders.

Shaders

It's important to note that not all shaders can use textures, and those that can only use textures for their diffuse channel with the exception of the uber shader. The shaders that can handle textures are diffuse, phong, shiny, ward, ambocc, and uber. The image types that are recognized are tga, png, jpg, bmp, hdr, and igi (in the SVN). For textures, the objects must have UVs mapped. Also note that textures can also be in relative paths:

```
texture texturepath/mybump.jpg
```

If you want to see a good overview of some of the shaders and varying settings, I'll show Kirk's IBL test image:

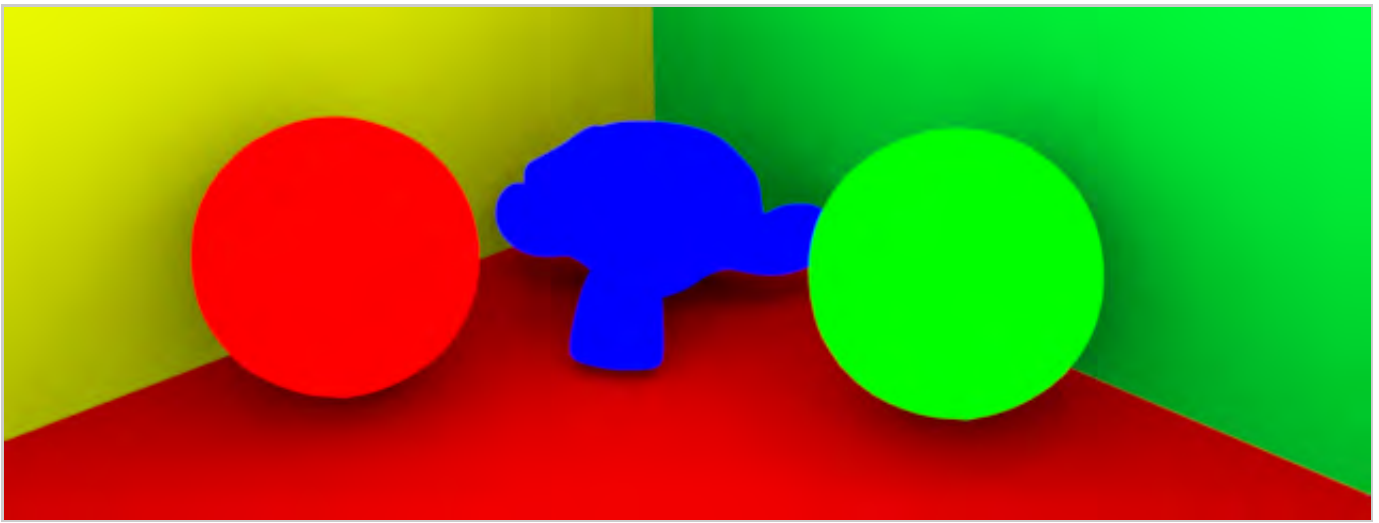


IBL importance sampling turned off (lock true) for a variety of shaders with various settings.

Example scene for the images below provided by olivS (<http://feeblemind.tuxfamily.org/dotclear/index.php>) with some shader settings changed and ambocc gi used.

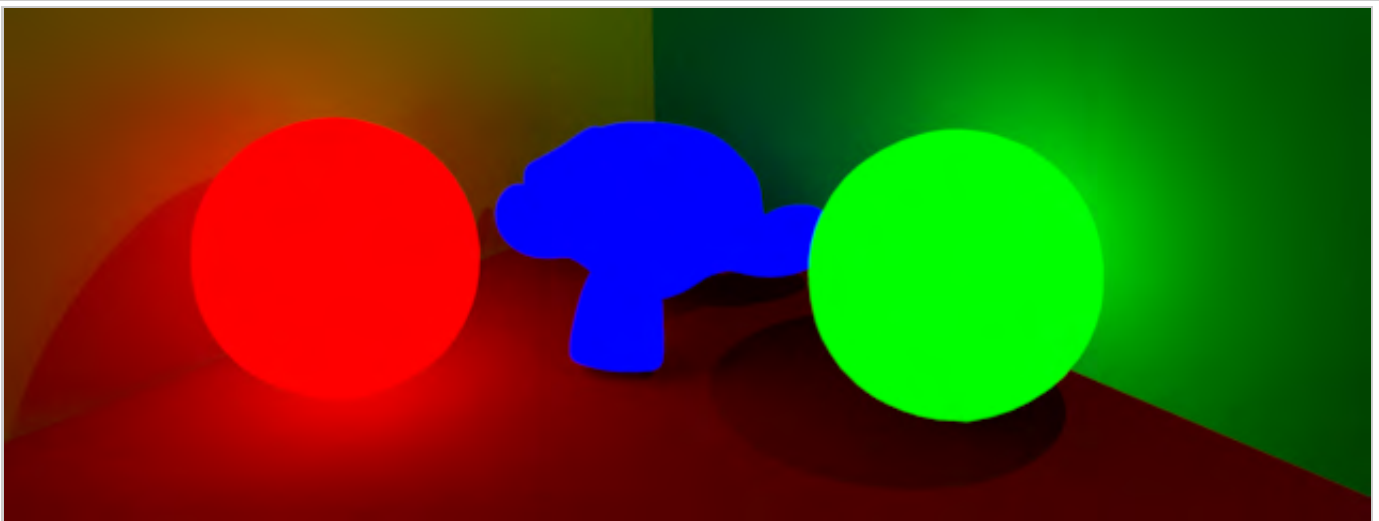
Constant Shader

```
shader {
    name sfcon.shader
    type constant
    color { "sRGB nonlinear" 0.800 0.800 0.800 }
}
```



Various colors showing the constant shader in action.

Aside from being used as a flat shader, the constant shader can also be used to fake lighting when path tracing is used by increasing the value of the color beyond 1.0.



This is the constant shader with color values above 1.0 with path tracing added to the scene.

It can also be used in conjunction with global illumination to get some interesting results (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=149>) .

Diffuse Shader

```
shader {
    name sfdif.shader
    type diffuse
    diff { "sRGB nonlinear" 0.800 0.800 0.800 }
}
```



Various diffuse shader colors. The walls are also the diffuse shader.

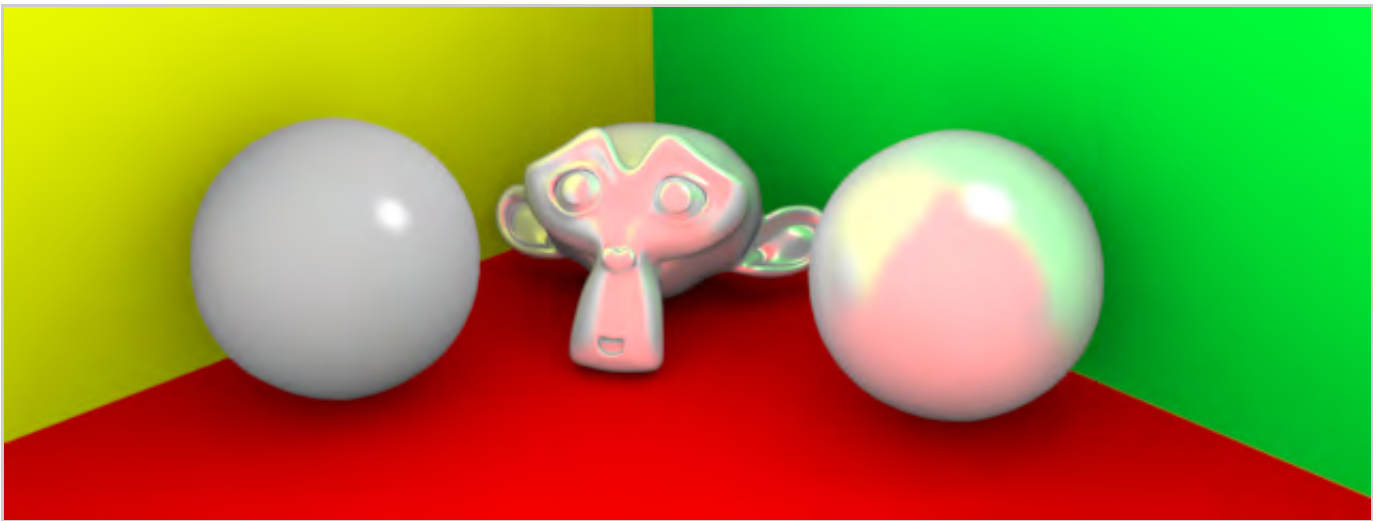
Textured Diffuse Shader

```
shader {  
    name sfdif.shader  
    type diffuse  
    texture "C:\mypath\image.png"  
}
```

Phong Shader

```
shader {  
    name sfpho.shader  
    type phong  
    diff { "sRGB nonlinear" 0.800 0.800 0.800 }  
    spec { "sRGB nonlinear" 1.0 1.0 1.0 } 50  
    samples 4  
}
```

The number after the spec color values is the "power" or hardness of the specularity. You can crank it pretty high (e.g. 50000), so start with low values like 50 and work your way up or down from there. If you set the samples to 0, you'll turn off the indirect glossy reflections. If you set the samples to anything greater than 1, you'll get blurry reflections (with higher samples giving better reflections).



The sphere on the left has samples set to 0 (so no blurry reflections) whereas two right objects have samples >0 (so have blurry reflections).

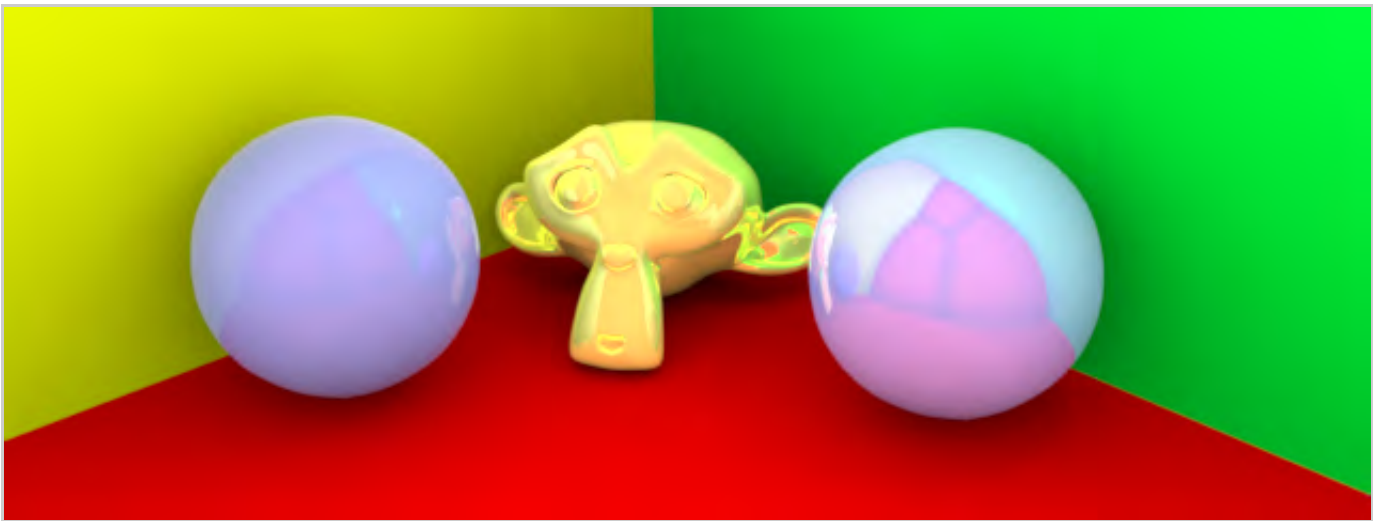
Textured Phong Shader

```
shader {  
    name sfpho.shader  
    type phong  
    texture "C:\mypath\image.png"  
    spec { "sRGB nonlinear" 1.0 1.0 1.0 } 50  
    samples 4  
}
```

Shiny Shader

```
shader {  
    name sfshi.shader  
    type shiny  
    diff { "sRGB nonlinear" 0.800 0.800 0.800 }  
    refl 0.5  
}
```

Remember that you can go beyond 1.0 for the reflection value to really kick it up a notch - bam.



The sphere on the left has a lower reflection setting than the two objects on the right.

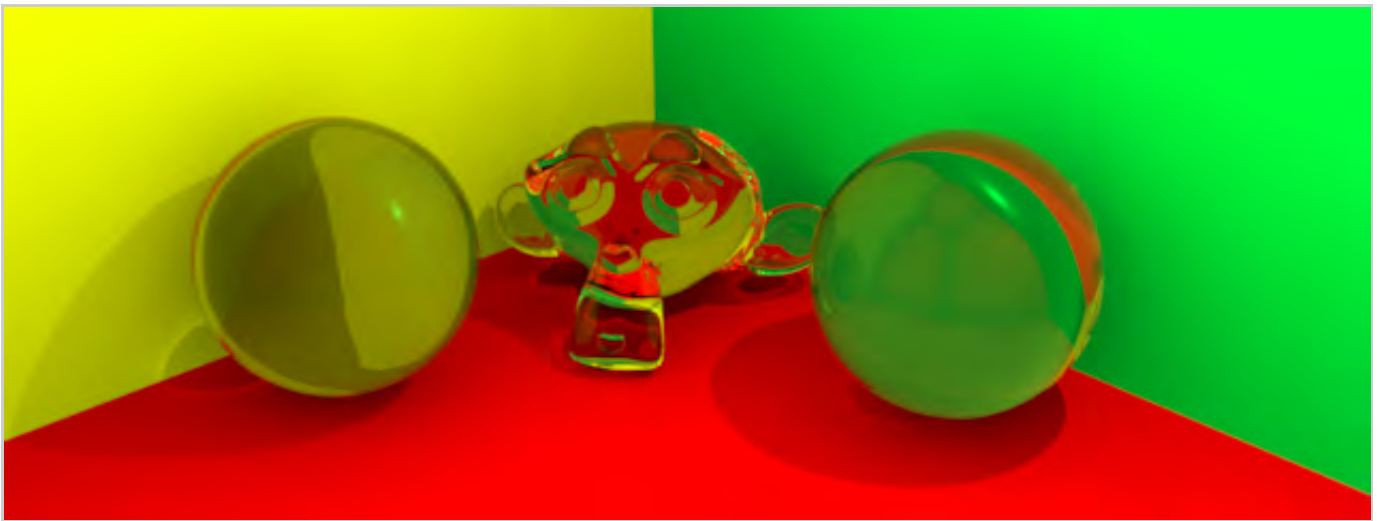
Textured Shiny Shader

```
shader {  
    name sfshi.shader  
    type shiny  
    texture "C:\mypath\image.png"  
    refl 0.5  
}
```

Glass Shader

```
shader {  
    name sfgla.shader  
    type glass  
    eta 1.0  
    color { "sRGB nonlinear" 0.800 0.800 0.800 }  
    absorbtion.distance 5.0  
    absorbtion.color { "sRGB nonlinear" 1.0 1.0 1.0 }  
}
```

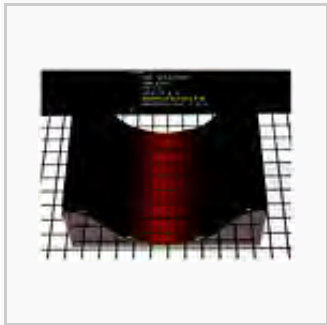
The "eta" is more commonly known as "ior" or index of refraction. It's important to note that without caustics turned on, the glass shader will cast shadows like any other object.



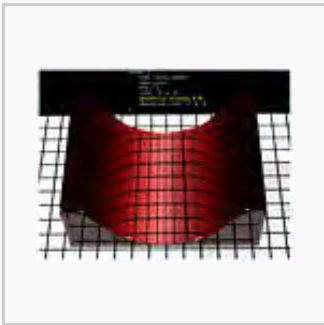
The sphere on the left has an eta of 1.33 (water) and the right two objects have an eta of 1.5 (glass).

Absorption

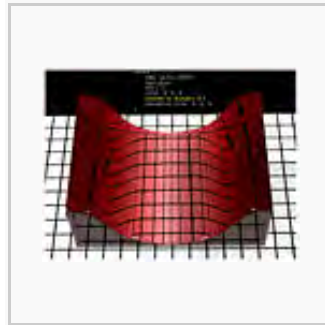
It's important to note that for the glass shader, the absorption lines are optional. Kirk did some tests using the absorption lines which I have added here:



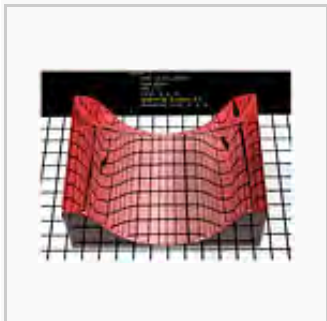
Absorption Distance =
0.01



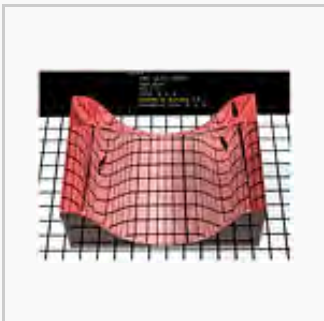
Absorption Distance =
0.05



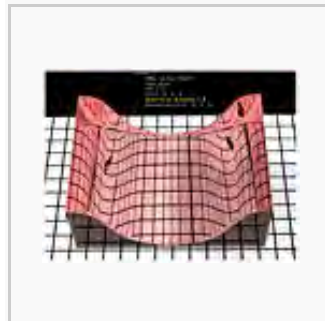
Absorption Distance =
0.1



Absorption Distance =
0.5



Absorption Distance =
1.0



Absorption Distance =
5.0

And for those savvy readers, I'll mention that the syntax's spelling of absorbtion isn't a typo. This spelling error has been fixed in the SVN so that Sunflow will recognize this spelling and the correct one.

Trace Depths

You can also manipulate the trace depths for glass by adding:

```
trace-depths {  
    diff 1  
    refl 4  
    refr 4  
}
```

If you don't add this to the scene file, the defaults of diff 1, refl 4, and refr 4 will be used. For glass the two important ones are refl and refr. For a look at various trace-depths in action see the below tests by Sielan:



Diff=1, Refl=1, Refr=1



Diff=2, Refl=2, Refr=2



Diff=5, Refl=5, Refr=5



Diff=8, Refl=8, Refr=8



Diff=12, Refl=12,
Refr=12

Caustics

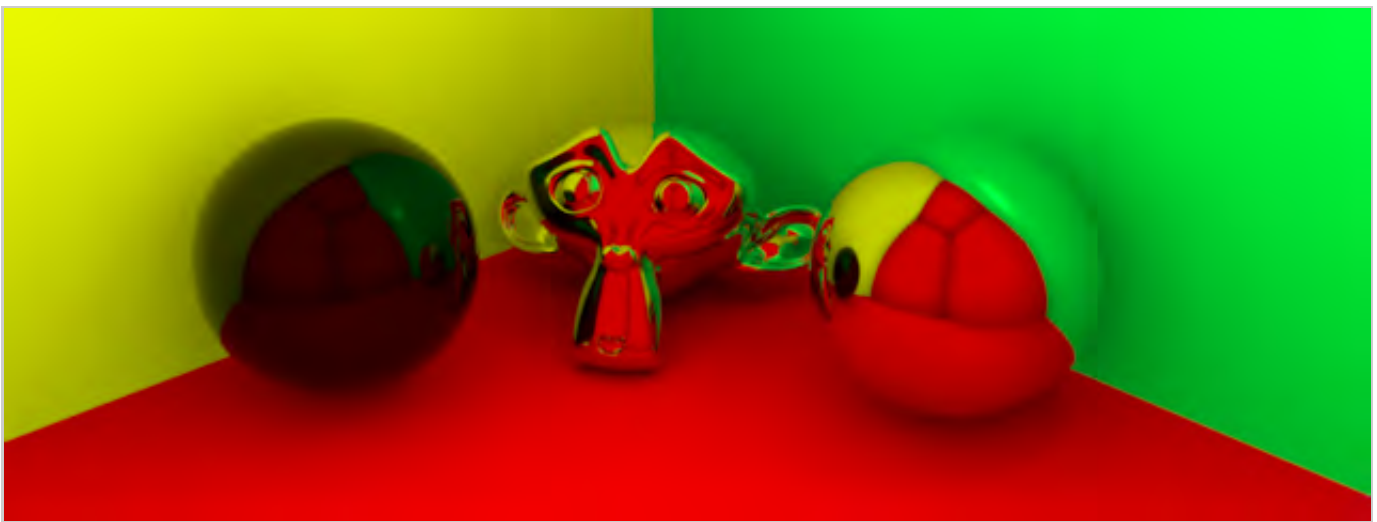
To enable caustics for glass you would add this to the scene:

```
photons {  
    caustics 1000000 kd 100 0.5  
}
```

For casutics you set the number of photons you want to use. Currently only kd mapping is allowed. Then you need to define the estimate and radius (in the example above the estimate is 100 and the radius is 0.5). These values are used at a single secondary bounced photon (of the many photons used). At that point, sphere with a radius expands outward to encompass a certain number of other photons (estimate) to be used to determine the caustics at that point. For estimates, typically 30 to 200 photons are used.

Mirror Shader

```
shader {
  name sfmir.shader
  type mirror
  refl { "sRGB nonlinear" 0.800 0.800 0.800 }
}
```

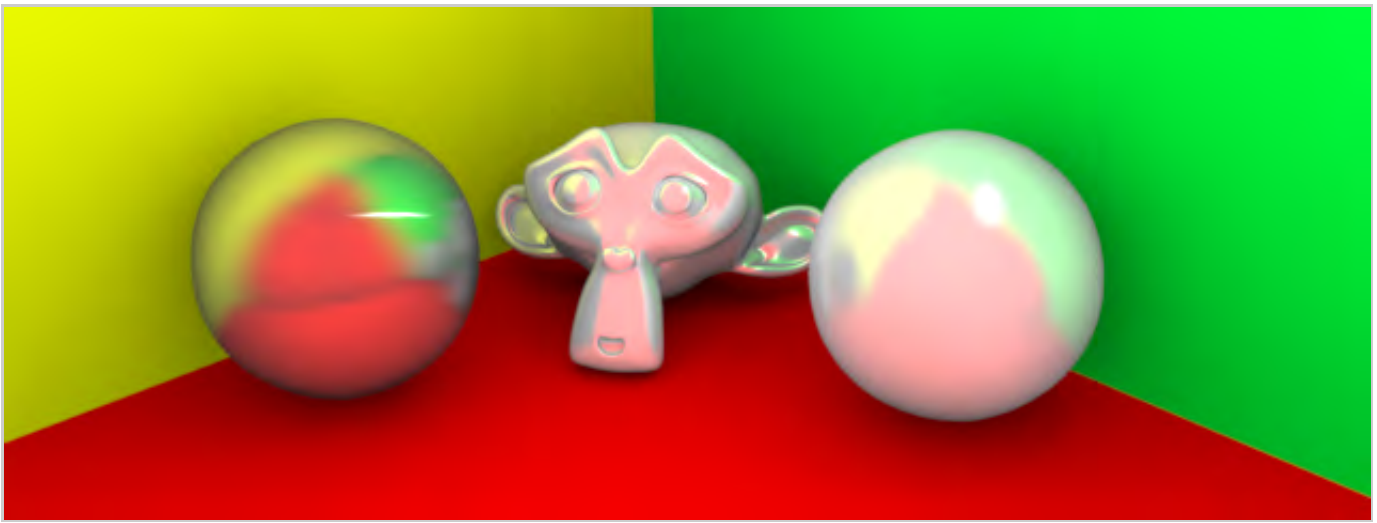


The sphere on the left has a darker color, the right two objects have a pure white color.

Ward Shader

```
shader {
  name sfwar.shader
  type ward
  diff { "sRGB nonlinear" .80 1 .80 }
  spec { "sRGB nonlinear" 1 1 1 }
  rough .07 .1
  samples 4
}
```

The x and y rough values are the amount of blurriness in the u and v tangent directions at the surface. The x and y rough values correspond to u and v tangent directions, so for the ward shader, you'll need to have uv coordinates defined on the object to get a proper result. If you set the samples to 0, you'll turn off the indirect glossy reflections. If you set the samples to anything greater than 1, you'll get blurry reflections (with higher samples giving better reflections).



The sphere on the left has the roughness and color different from the right two objects.

Some users of the ward shader are surprised that the result doesn't look like a similar shader in their other favorite application. That look having a metallic surface texture. This shader is a true ward shader, whereas other apps may add other effects to get the metal bump.

Note that the ward shader in 0.07.2 may cause a NaN error (black dots). This bug has been fixed in the the SVN 0.07.3 version.

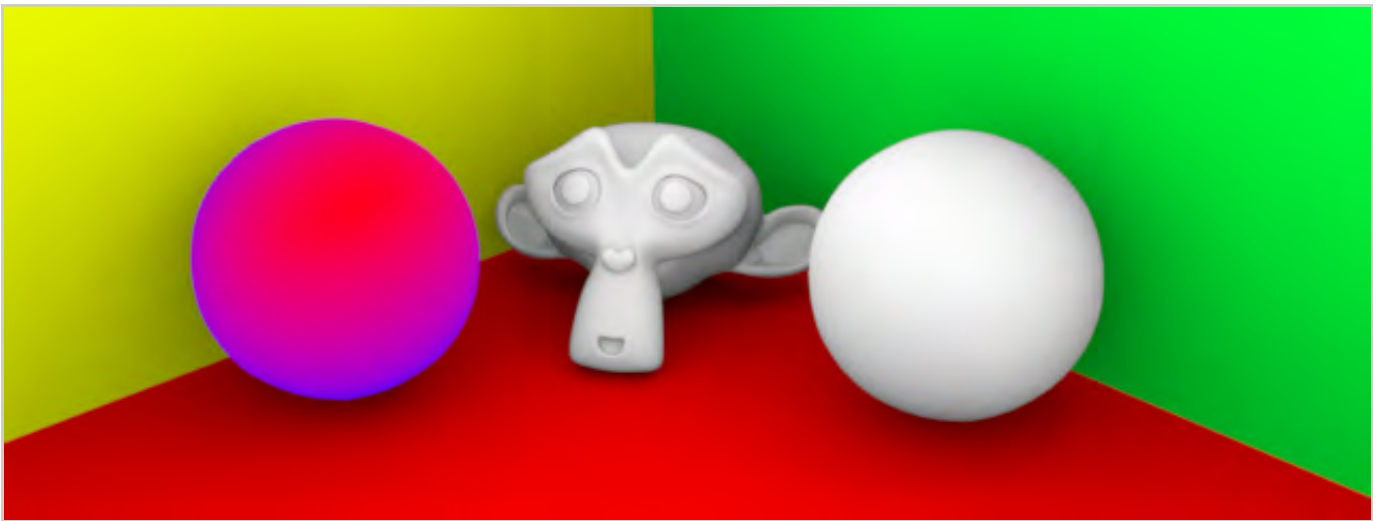
Textured Ward Shader

```
shader {
    name sfwar.shader
    type ward
    texture "C:\mypath\image.png"
    spec { "sRGB nonlinear" 1 1 1 }
    rough .07 .1
    samples 4
}
```

Ambient Occlusion Shader

```
shader {
    name sfamb.shader
    type amb-occ
    bright { "sRGB nonlinear" 0 0 0 }
    dark { "sRGB nonlinear" 1 1 1 }
    samples 32
    dist 3.0
}
```

Ambient occlusion shades based on geometric proximity - it totally ignores lights. The ambient occlusion gi engine does recognize lights.



The sphere on the left a different light and dark color, the right two objects have the classic light = white and dark = black color.

Textured Ambient Occlusion Shader

```
shader {
    name sfamb.shader
    type amb-occ
    texture "C:\mypath\image.png"
    dark { "sRGB nonlinear" 1.0 1.0 1.0 }
    samples 32
    dist 3.0
}
```

The amb-occ textured shader has been tricky for me in that I've never had much success with it. Maybe someone will post to this thread how they've used it.

Uber Shader

The uber shader is a mix of several shaders to give you more options in controlling the look with the added bonus of being able to use a specular texture map. Though typically used with textures it can be used without both or on of the textures by removing the texture line(s).

```
shader {
    name sfuber.shader
    type uber
    diff { "sRGB nonlinear" 0.8 0.8 0.8 }
    diff.texture "C:\mypath\diffimage.png"
    diff.blend 1.0
    spec { "sRGB nonlinear" 1 1 1 }
    spec.texture "C:\mypath\specimage.png"
    spec.blend 0.1
    glossy .1
    samples 4
}
```

The blend values control the blending between the color and the texture. A blend value of 1 will make the texture the sole contributor, and a value of 0 will make only the color used. Any value in between will mix the texture and the color to produce the final result. The glossy setting, which gives you the shiny shader aspect, uses very large step values to achieve the result. So a value of zero is shiny, and a value of 1 has no glossyness. When playing with the settings try the following values: 0, 0.001, 0.01, 0.1, and 1. Also, if you don't need a diffuse or specular texture (for whatever reason) you can omit the `diff.texture` and/or `spec.texture` lines.

In the uber shader, setting the spec color to 0 0 0 and `spec.blend` to 0 will allow transparency, though this won't translate to transparent shadows at the moment.

Command Line Shaders

Wireframe Shader/Normal Shader/UV Shader/ID Shader/Gray Shader/Prims Shader/Quick AO Shader. These shaders are actually accessed via the command line. I go over how to use these shader flags [here](#).

Janino Shaders

You might ask the question "What is Janino?" Think of Janino as a Java compiler that only compiles when a janino shader is encountered. So if you write the shader in Java using Sunflow classes, you can come up with creative shaders without having to edit the source code directly. Some simple testing have shown that the janino shaders compile just as fast as if they were in the source. The syntax for this shader looks like this:

```
shader {  
  name myJaninoShader  
  type janino  
  <code>  
  Your code goes here  
  </code>  
}
```

There are several Janino shaders written by forum members:

Mix Shader

Mark Thorpe wrote this mix shader which blends two shaders you define over an object. It was later found that running the shader on a multi-threaded machine caused this shader to return odd results which lead to the re-write/re-format by Don Casteel.

Fresnel Shader

Another mix shader, the fresnel shader uses the blend function in the api to mix two shaders you define based on the cosine between the shading normal and the ray. What's really cool is that you can change the blending just by defining your own value for `c`.

Stained Glass

Another elegant janino shader by Mark Thorpe, the stained glass shader gives a stained glass look by applying a texture to a glass shader. You need caustics turned on in your scene for this to work as intended.

Shiny Shader with Reflection Mapping

This shader is the shiny shader, except instead of setting a globally induced reflection value, I made it so you can use a texture to control the reflection value over the object. That way, you can have different reflection values in different places.

Simple SSS

Prior to the monumental undertaking that is the True SSS shader, Don tried a simple solution to fake subsurface scattering.

Specular Pass

This was a shader that was more of a proof of concept. Someone on the forum asked how we could derive passes from Sunflow even though Sunflow doesn't yet have that ability. I suggested you could use a janino shader to pull out the pass you want and gave this example which is the phong shader with the spec color of white and no diffuse shading.

Translucent Shader

A cool translucent shader from Mark Thorpe based off Don's True SSS shader.

Wire Shader

This shader doesn't work in 0.07.2 since there have been changes to the source code since it was created. I've included it here for reference. Plus, there is a command line wire frame shader that makes this shader superfluous.

Source Code Shaders

Shaders so awesome, they needed modifications to the source code to make them happen.

True SSS

This subsurface scattering shader is currently under development, but it's still amazing.

Textured SSS

Don's work in developing a procedural marble texture and volume shader that compliments his True SSS shader.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Shaders>"

- This page was last modified on 15 January 2008, at 18:33.

Shaders/Janino Mix

From Sunflow Wiki

< Shaders

Author: Don Casteel based on Mark Thorpe's version of the shader.



```
shader {  
    name "mix.shader"  
    type janino  
    <code>  
    import org.sunflow.core.ShadingState;  
    import org.sunflow.image.Color;  
    import org.sunflow.core.ParameterList;  
    import org.sunflow.SunflowAPI;  
    import org.sunflow.math.Point3;  
    import org.sunflow.core.shader.*;  
  
    DiffuseShader      ds = new DiffuseShader();  
    GlassShader        gs = new GlassShader();  
    MirrorShader       ms = new MirrorShader();  
    boolean            b1, b2, b3, initflag, updfalg = false;  
    Color              color = new Color(0f,0f,0f);  
}
```

```

public Color getRadiance(ShadingState state) {
    Color rColor = new Color(0f,0f,0f);
    float gap = 1f; // % size of the blending region
    float offset = -0.25f; // % offset from object centre of the blend position
    float extenty = state.getInstance().getBounds().getExtents().y;
    float centery = state.getInstance().getBounds().getCenter().y;
    centery += offset * extenty;
    float bot = centery - extenty * gap/2f; // start gap
    float top = centery + extenty * gap/2f; // end gap
    float fac = top - bot;
    float hit = state.getPoint().y;
    if (hit < bot)
    { // below the gap
        rColor.set(gs.getRadiance(state));
        return rColor;
    }
    if (hit > top)
    { // above the gap
        rColor.set(ms.getRadiance(state));
        return rColor;
    }

    // in the gap
    Point3 p = new Point3(state.getPoint()); // save point
    Color m = new Color( ms.getRadiance(state) );
    state.getPoint().set(p); // restore point
    Color g = new Color( gs.getRadiance(state) );
    //rColor.set(Color.blend(m, g, 0.1f ));
    rColor.set(Color.blend(m, g, (hit - bot) / fac ));
    return rColor;
}

public boolean update(ParameterList pl, SunflowAPI api) {
    if (!updflag) { updflag = true; // one shot
    api.parameter( "diffuse", new Color(0.5f,0.43f,0.2f) );
    b3 = ds.update(pl, api); // update diffuse shader
    api.parameter( "color", new Color(0.5f,0.43f,0.2f) );
    b2 = ms.update(pl, api); // update mirror shader
    api.parameter( "color", new Color(1f,1f,1f) );
    api.parameter( "eta", 1.5f );
    api.parameter( "absorbtion.distance", 0.5f );
    api.parameter( "absorbtion.color", new Color(0.3f,0.3f,0.3f) );
    b1 = gs.update(pl, api); // update glass shader
    return (b1 && b2 && b3);
    }
    return true;
}
</code>
}

```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shaders/Janino_Mix"

- This page was last modified on 11 January 2008, at 01:57.

Shaders/Janino Fresnel

From Sunflow Wiki

< Shaders

Author: MrFoo

```
shader {
    name "fresnel"
    type janino
    <code>
    import org.sunflow.core.ShadingState;
    import org.sunflow.image.Color;
    import org.sunflow.core.ParameterList;
    import org.sunflow.SunflowAPI;
    import org.sunflow.core.shader.*;
    DiffuseShader light=new DiffuseShader();
    DiffuseShader dark=new DiffuseShader();

    public Color getRadiance(ShadingState state) {
        state.faceforward();
        float c=state.getCosND();
        Color l = new Color( light.getRadiance(state) );
        Color d = new Color( dark.getRadiance(state) );
        return Color.blend(l, d, c);
    }

    public void scatterPhoton(ShadingState state, Color power) {
        light.scatterPhoton(state, power); //do we need to do this?
    }

    public boolean update(ParameterList pl, SunflowAPI api) {
        pl.addColor( "diffuse", new Color(1.0f,0.0f,0.0f) );
        boolean b2 = dark.update(pl, api);
        pl.addColor( "diffuse", new Color(0.0f,1.0f,0.0f) );
        boolean b1 = light.update(pl, api);
        return (b1 && b2);
    }
    </code>
}
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shaders/Janino_Fresnel"

- This page was last modified on 7 January 2008, at 01:59.

Shaders/Janino Stained Glass

From Sunflow Wiki

< Shaders

Author: Mark Thorpe



```
shader {
    name "stained_glass"
    type janino
    <code>
    /* Free to use for any purpose. Just do my PR a favour and leave this message here :-) */
    /* usage: Insert the stained glass texture in the update() method block. MPT 13/04/07. */

    import org.sunflow.core.ShadingState;
    import org.sunflow.image.Color;
    import org.sunflow.core.ParameterList;
    import org.sunflow.SunflowAPI;
    import org.sunflow.core.shader.*;

    GlassShader s_1 = new GlassShader();
    TexturedDiffuseShader s_2 = new TexturedDiffuseShader();
    boolean b1, b2, b3 = false;
    SunflowAPI myapi = new SunflowAPI();
```

```
public Color getDiffuse( ShadingState state ) {
    return s_2.getDiffuse( state );
}

public Color getRadiance( ShadingState state ) {
    ParameterList mypl = new ParameterList();
    mypl.addColor( "color", s_2.getDiffuse( state ) );
    b3 = s_1.update( mypl, myapi );
    return s_1.getRadiance( state );
}

public void scatterPhoton( ShadingState state, Color power ) {
    ParameterList mypl = new ParameterList();
    mypl.addColor( "color", s_2.getDiffuse( state ).mul(3f) );
    b3 = s_1.update( mypl, myapi );
    s_1.scatterPhoton( state, power );
}

public boolean update( ParameterList pl, SunflowAPI api ) {
    pl.addString( "texture", "./textures/sg2.jpg" );
    b2 = s_2.update( pl, api );
    pl.addFloat( "eta", 1.000000001f );
    b1 = s_1.update( pl, api );
    return ( b1 && b2 );
}
</code>
}
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shaders/Janino_Stained_Glass"

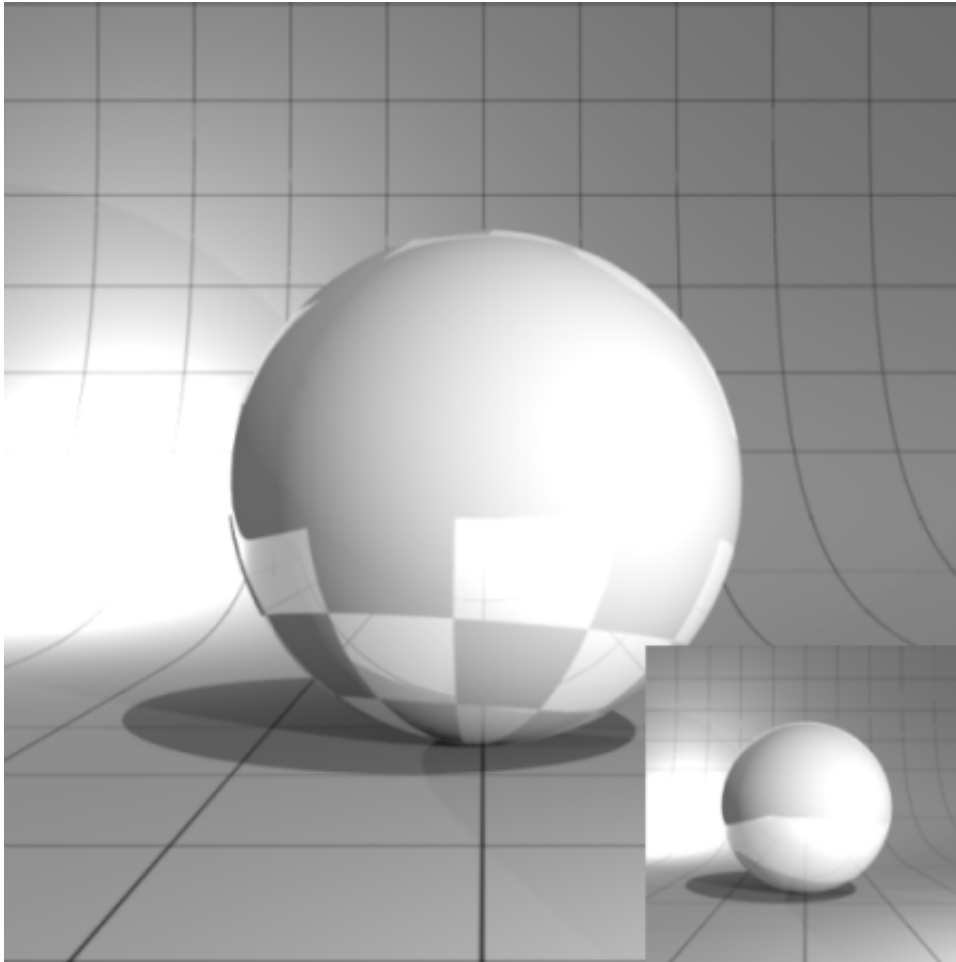
- This page was last modified on 7 December 2007, at 18:18.

Shaders/Janino Shiny Reflection Maps

From Sunflow Wiki

< Shaders

Author: Eugene Reilly



```
shader {  
    name "shinyReflectionMap"  
    type janino  
    <code>  
    import org.sunflow.SunflowAPI;  
    import org.sunflow.core.ParameterList;  
    import org.sunflow.core.Ray;  
    import org.sunflow.core.ShadingState;  
    import org.sunflow.core.Texture;  
    import org.sunflow.core.TextureCache;  
    import org.sunflow.image.Color;  
    import org.sunflow.math.Vector3;  
  
    Color diff;  
    //Texture texd;  
    Texture texr;  
  
    public boolean update(ParameterList pl, SunflowAPI api) {  
        /***Objects must have UVs mapped***  

```

```

//texd = TextureCache.getTexture("C:\\myPath\\diffuseMap.jpg", true);
//Refelction map must be grayscale
texr = TextureCache.getTexture("C:\\myPath\\reflectionMap.png", true);
diff = new Color(1.0f,1.0f,1.0f);
return true;
}

public Color getDiffuse(ShadingState state) {
    return diff;
    //return texd.getPixel(state.getUV().x, state.getUV().y);
}

public Color getRadiance(ShadingState state) {
    Color rColor = texr.getPixel(state.getUV().x, state.getUV().y);
    float refl = rColor.getAverage();
    // make sure we are on the right side of the material
    state.faceforward();
    // direct lighting
    state.initLightSamples();
    state.initCausticSamples();
    Color d = getDiffuse(state);
    Color lr = state.diffuse(d);
    if (!state.includeSpecular())
        return lr;
    float cos = state.getCosND();
    float dn = 2 * cos;
    Vector3 refDir = new Vector3();
    refDir.x = (dn * state.getNormal().x) + state.getRay().getDirection().x;
    refDir.y = (dn * state.getNormal().y) + state.getRay().getDirection().y;
    refDir.z = (dn * state.getNormal().z) + state.getRay().getDirection().z;
    Ray refRay = new Ray(state.getPoint(), refDir);
    // compute Fresnel term
    cos = 1 - cos;
    float cos2 = cos * cos;
    float cos5 = cos2 * cos2 * cos;

    Color w = Color.white();
    Color ret = w.copy().mul(refl);
    Color r = d.copy().mul(refl);
    ret.sub(r);
    ret.mul(cos5);
    ret.add(r);
    return lr.add(ret.mul(state.traceReflection(refRay, 0)));
}
}
</code>
}

```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shaders/Janino_Shiny_Reflection_Maps"

- This page was last modified on 2 January 2008, at 19:56.

Shaders/Janino Simple SSS

From Sunflow Wiki

< Shaders

Author: Don Casteel

```
shader {
    name "simple_sss"
    type janino
    <code>
    import org.sunflow.SunflowAPI;
    import org.sunflow.core.ParameterList;
    import org.sunflow.core.Ray;
    import org.sunflow.core.Shader;
    import org.sunflow.core.ShadingState;
    import org.sunflow.image.Color;
    import org.sunflow.math.OrthoNormalBasis;
    import org.sunflow.math.Vector3;
    import org.sunflow.math.Point3;

    Color diff = new Color(0.3f,1.0f,0.6f);
    Color spec = new Color(1.0f,1.0f,1.0f);
    Color bright = new Color(1f,1f,1f);
    Color dark = new Color(0f,0f,0f);
    Color color = new Color(0.6f,0.8f,0.8f);
    Color absorbtionColor = new Color(0.6f,0.5f,0.3f).opposite();
    int numRays = 5;
    float power = 100;
    float reflectiveness = 0.8f;
    float hardness = 0.15f;
    float depth = 0.15f;
    float spread = 1f;
    float glossyness = 0.8f;
    float absorbtionValue = 0.5f;

    public boolean update(ParameterList pl, SunflowAPI api) {
        return true;
    }

    protected Color getDiffuse(ShadingState state) {
        return diff;
    }

    public Color getRadiance(ShadingState state) {
        state.faceforward();
        state.initLightSamples();
        state.initCausticSamples();

        // execute shader
        float cos = state.getCosND();
        float dn = 2 * cos;
        Vector3 refDir = new Vector3();
        refDir.x = (dn * state.getNormal().x) + state.getRay().getDirection().x;
        refDir.y = (dn * state.getNormal().y) + state.getRay().getDirection().y;
        refDir.z = (dn * state.getNormal().z) + state.getRay().getDirection().z;
        Ray refRay = new Ray(state.getPoint(), refDir);

        Color reflections = Color.WHITE;
```

```

Color highlights = Color.WHITE;
reflections = state.traceReflection(refRay, 0);
highlights = state.diffuse(getDiffuse(state)).add(state.specularPhong(spec, power, numRays));
reflections = Color.blend(highlights, reflections, reflectiveness);

Color sColor = state.diffuse(getDiffuse(state));
sColor = Color.blend(sColor, reflections, glossyness);
Vector3 norm = state.getNormal();

Point3 pt = state.getPoint();
pt.x += norm.x*spread;
pt.y += norm.y*spread;
pt.z += norm.z*spread;

state.getPoint().set(pt);
Color tColor = state.getIrradiance(sColor);
pt.x -= norm.x*(spread+depth);
pt.y -= norm.y*(spread+depth);
pt.z -= norm.z*(spread+depth);

state.getPoint().set(pt);
state.getNormal().set(state.getRay().getDirection());

Color sssColor = Color.add(diff, tColor.mul(state.occlusion(16, absorbtionValue, bright, dark)).mul(absorbtionColor));
return Color.blend(sssColor, sColor, hardness);
}

public void scatterPhoton(ShadingState state, Color power) {
    //just a copy of the scatter method in PhongShader.....
    // make sure we are on the right side of the material
    state.faceforward();
    Color d = getDiffuse(state);
    state.storePhoton(state.getRay().getDirection(), power, d);
    float avgD = d.getAverage();
    float avgS = spec.getAverage();
    double rnd = state.getRandom(0, 0, 1);
    if (rnd < avgD) {
        // photon is scattered diffusely
        power.mul(d).mul(1.0f / avgD);
        OrthoNormalBasis onb = state.getBasis();
        double u = 2 * Math.PI * rnd / avgD;
        double v = state.getRandom(0, 1, 1);
        float s = (float) Math.sqrt(v);
        float s1 = (float) Math.sqrt(1.0f - v);
        Vector3 w = new Vector3((float) Math.cos(u) * s, (float) Math.sin(u) * s, s1);
        w = onb.transform(w, new Vector3());
        state.traceDiffusePhoton(new Ray(state.getPoint(), w), power);
    }
    else if (rnd < avgD + avgS) {
        // photon is scattered specularly
        float dn = 2.0f * state.getCosND();
        // reflected direction
        Vector3 refDir = new Vector3();
        refDir.x = (dn * state.getNormal().x) + state.getRay().dx;
        refDir.y = (dn * state.getNormal().y) + state.getRay().dy;
        refDir.z = (dn * state.getNormal().z) + state.getRay().dz;
        power.mul(spec).mul(1.0f / avgS);
        OrthoNormalBasis onb = state.getBasis();
        double u = 2 * Math.PI * (rnd - avgD) / avgS;
        double v = state.getRandom(0, 1, 1);
        float s = (float) Math.pow(v, 1 / (this.power + 1));
        float s1 = (float) Math.sqrt(1 - s * s);
        Vector3 w = new Vector3((float) Math.cos(u) * s1, (float) Math.sin(u) * s1, s);
        w = onb.transform(w, new Vector3());
    }
}

```

```
        state.traceReflectionPhoton(new Ray(state.getPoint(), w), power);
    }
}

public void init(ShadingState state) {
}
</code>
}
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shaders/Janino_Simple_SSS"

- This page was last modified on 2 December 2007, at 19:44.

Shaders/Janino Specular Pass

From Sunflow Wiki

< Shaders

Author: Eugene Reilly

```
shader {
    name specPass
    type janino
    <code>
    import org.sunflow.SunflowAPI;
    import org.sunflow.core.ParameterList;
    import org.sunflow.core.ShadingState;
    import org.sunflow.image.Color;

    private Color spec = Color.WHITE;
    private float power = 20;
    private int numRays = 4;

    public Color getRadiance(ShadingState state) {
        // make sure we are on the right side of the material
        state.faceforward();
        // setup lighting
        state.initLightSamples();
        state.initCausticSamples();
        // execute shader
        return state.specularPhong(spec, power, numRays);
    }

    public boolean update(ParameterList pl, SunflowAPI api) {
        spec = pl.getColor("specular", spec);
        numRays = pl.getInt("samples", numRays);
        return true;
    }
    </code>
}
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shaders/Janino_Specular_Pass"

- This page was last modified on 2 December 2007, at 19:45.

Shaders/Janino Translucent

From Sunflow Wiki

< Shaders

Author: Mark Thorpe



- For good results, use 128 rays per sample and aa 0 1. For mind-blowing results use 256 rps and aa 0 4. For testing use 16 rps and aa 0 0. These are all set in the sc file.
- DON'T use less than 2 rps, as this produces inexplicable garbage results.
- Also, the use of the 'glob' global might screw things up if running as more than a single thread. I couldn't find a way around this unfortunately.
- The shader produces the effect of translucency. It doesn't yet have the capability to include textures.

```

/* TRANSLUCENT - WITH STORAGE AND REFLECTION OF PHOTONS */
shader {
    name "translucent_sr"
    type janino
    <code>
    import org.sunflow.SunflowAPI;
    import org.sunflow.core.ParameterList;
    import org.sunflow.core.Ray;
    import org.sunflow.core.Shader;
    import org.sunflow.core.ShadingState;
    import org.sunflow.image.Color;
    import org.sunflow.math.Vector3;
    import org.sunflow.math.Point3;
    import org.sunflow.math.OrthoNormalBasis;
    // object color
    public Color color = Color.WHITE;
    // object absorption color
    //public Color absorptionColor = Color.RED;
    public Color absorptionColor = Color.BLUE;
    // inverse of absorption color
    public Color transmittanceColor = absorptionColor.copy().opposite();
    // global color-saving variable
    /* FIXME!?? - globals are not good */
    public Color glob = Color.black();
    // phong specular color
    public Color pcolor = Color.BLACK;
    // object absorption distance
    public float absorptionDistance = 0.25f;
    // depth correction parameter
    public float thickness = 0.002f;
    // phong specular power
    public float ppower = 85f;
    // phong specular samples
    public int psamples = 1;
    // phong flag
    public boolean phong = false;

    public boolean update(ParameterList pl, SunflowAPI api) {
        color = pl.getColor("color", color);
        if (absorptionDistance == 0f) {
            absorptionDistance+= 0.0000001f;
        }
        if (!pcolor.isBlack()) {
            phong = true;
        }
        return true;
    }

    public Color getRadiance(ShadingState state) {
        Color ret = Color.black();
        Color absorbtion = Color.white();
        glob.set(Color.black());
        state.faceforward();
        state.initLightSamples();
        state.initCausticSamples();
        if (state.getRefractionDepth() == 0) {
            ret.set(state.diffuse(color).mul(0.5f));
            bury(state,thickness);
        } else {
            absorbtion = Color.mul(-state.getRay().getMax() / absorptionDistance, transmittanceCo
        }
        state.traceRefraction(new Ray(state.getPoint(), randomVector()), 0);
        glob.add(state.diffuse(color));
        glob.mul(absorbtion);
    }
}

```

```
        if (state.getRefractionDepth() == 0 && phong) {
            bury(state, -thickness);
            glob.add(state.specularPhong(pcolor, ppower, psamples));
        }
        return glob;
    }

    public void bury(ShadingState state, float th) {
        Point3 pt = state.getPoint();
        Vector3 norm = state.getNormal();
        pt.x = pt.x - norm.x * th;
        pt.y = pt.y - norm.y * th;
        pt.z = pt.z - norm.z * th;
    }

    public Vector3 randomVector() {
        return new Vector3(
            (float) (2f*Math.random()-1f),
            (float) (2f*Math.random()-1f),
            (float) (2f*Math.random()-1f)
        ).normalize();
    }

    public Color getDiffuse(ShadingState state) {
        return color;
    }

    public void scatterPhoton(ShadingState state, Color power) {
        Color diffuse = getDiffuse(state);
        state.storePhoton(state.getRay().getDirection(), power, diffuse);
        state.traceReflectionPhoton(new Ray(state.getPoint(), randomVector()), power.mul(diffuse));
    }
}
</code>
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shaders/Janino_Translucent"

- This page was last modified on 31 December 2007, at 03:28.

Shaders/Janino Wire

From Sunflow Wiki

< Shaders

Author: Chris Kulla

```
shader {
  name triangle_wire
  type janino
  <code>
  import org.sunflow.core.RenderState;
  import org.sunflow.image.Color;
  import org.sunflow.math.Vector3;

  private Color lineColor = new Color(0.05f, 0.05f, 0.05f);
  private Color fillColor = new Color(0.95f, 0.95f, 0.95f);
  private float width = 0.02f;

  public Color getRadiance(RenderState state) {
    float cos = 1 - (float) Math.pow(1 - Math.abs(Vector3.dot(state.getNormal(), state.getRay().d),
    float u = state.getU();
    float v = state.getV();
    float w = 1 - u - v;
    return ((u < width || v < width || w < width) ? lineColor : fillColor).copy().mul(cos);
  }

  public void scatterPhoton(RenderState state, Color power) {
  }
  </code>
}
```

override triangle_wire true

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shaders/Janino_Wire"

- This page was last modified on 2 December 2007, at 19:38.

True SSS

From Sunflow Wiki

Author: Don Casteel

Contents

- 1 Source Code Changes
- 2 .sc Syntax
- 3 Diff
- 4 Tests

Source Code Changes

In order to correctly render the subsurface scattering, a modification was needed in the source code and therefore needs a special build of Sunflow. The problem was that the surface normals have to be controlled carefully to keep them from pointing inward from the surface even if the ray being traced is the inside heading out. For this reason, the shader requires that you use a special build.

.sc Syntax

Since we need to have a special build, Don also took the time to incorporate the shader into the .sc file format whose syntax looks like this:

```
shader {  
    name "sssShdr"  
    type sss  
    diff 0 1 1  
    //texture "../mypath/image.png"  
    absorptionDistance 0.001  
    absorptionPower    1  
    thickness          0.005  
    sssRays            4  
    specular           0.2 0.2 0.2 80  
    phongRays          6  
}
```

Diff

If you would like to build your own or see the changes that were made to the source, I have the diff from the 0.07.3 SVN build I did:

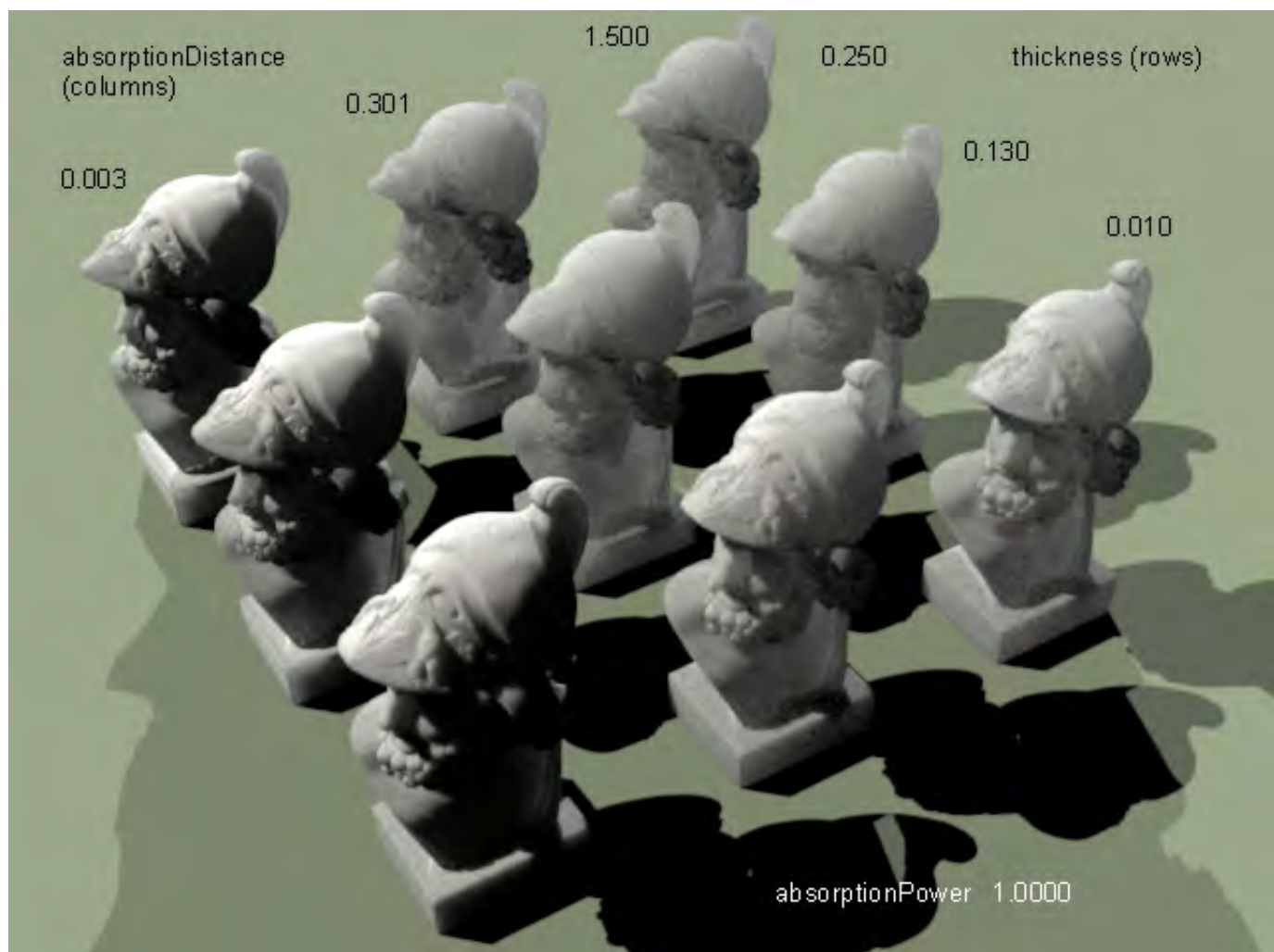
- 0.07.3 SVN Java 1.6 diff (<http://www.geneome.net/drawer/sunflow/J16-SSS-SVN-sunflow.diff>)
- 0.07.3 SVN Java 1.5 diff (<http://www.geneome.net/drawer/sunflow/J15-SSS-SVN-sunflow.diff>)

Tests

Don recently posted this to the thread (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?p=4037#4037>) which I find very helpful: "I'm working on understanding the relationships between the shader parameters and the appearance. The general size of the model affects what parameters you'll want to use, as well as the thinnest cross sections. I haven't checked yet, but assume scaleu will not affect the appearance since rays get the inverse transform applied when checking for intersection.

The ratio between absorptionDist and thickness seems to make a difference, you can achieve the same level of translucence with different ratios but starting with a higher thickness will help soften the apparent geometry details, where lower values preserve them.

I've got a lot more playing to do, but thought I'd share this render:



Retrieved from "http://sfwiki.geneome.net/index.php5?title=True_SSS"

- This page was last modified on 4 August 2009, at 20:14.

Procedurally Textured SSS

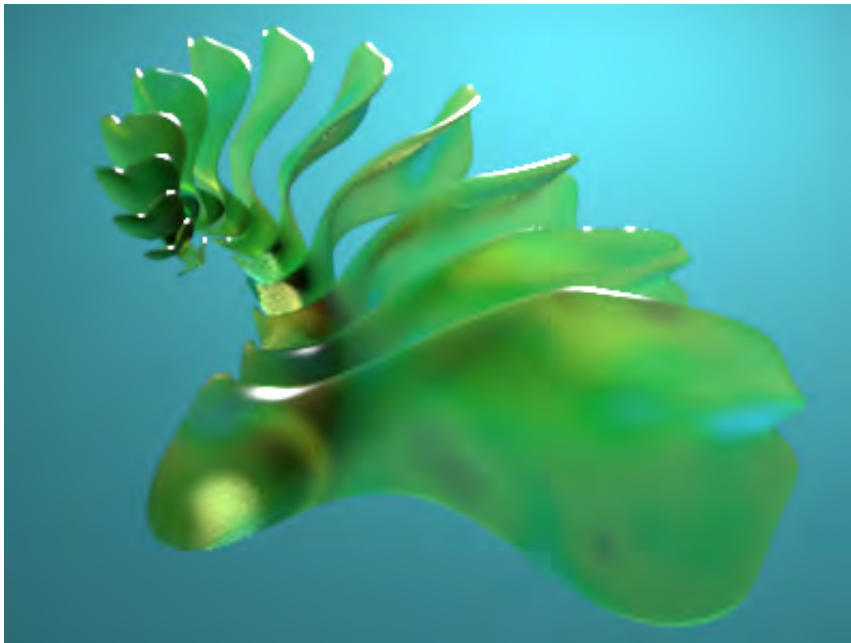
From Sunflow Wiki

Author: Don Casteel

The first procedural texture for Sunflow, this project was created by Don to work in conjunction with his true SSS shader.

Contents

- 1 Docs
- 2 .sc Syntax
 - 2.1 Solid Texture
 - 2.2 Volume Texture
- 3 Diff



Docs

Don has written up some documentation for the shader which you can find here (.doc)
(https://fracture.dev.java.net/files/documents/6137/91100/SSS_Solid_Shader_documentation.doc) .

.sc Syntax

Obviously it's slower, but only as you raise the sample level. Each sub-ray samples the texture by the number of samples, so the sampling is exponential... 4 rays samples 4 times = 16 texture samples, 5 rays 5 times = 25 samples etc.... PLUS n-samples along the reverse normal to the thickness, so actually if samples=5 there will be 30 texture samples.

You can use either "fill" and "veins" OR "map" but there is no syntax check, so if something is wrong, it will probably crash Sunflow. The value after "map" is the number of color values you are using. Each color line has a value for red, green, and blue, plus a value between 0.0 and 1.0 as it's location in the map.

Some notes from Don:

- There are still a few problems with this current version of the shader, particularly with "diffusion" values above 0.5 there starts to be visible banding that I just can't figure out.
- "diffusion" is a value >0 and <1 that controls how much the random rays diverge from the regular refraction vector. Values near zero will look a lot like a "Glass" shader, and values near 1.0 should have more of a typical subsurface scattering look.
- The "maxOpacity" parameter cuts off the calculation once the opacity exceeds the parameter value. The purpose is mainly to shorten render times in preliminary scene tests.
- "sampDist" is again to help with render times, I was using the thickness value to increment through the volume, but very shallow thicknesses with low opacities were taking forever, so you can set this parameter higher than the thickness value to reduce the samples along each scattering ray.
- specular is ignored if the power is set to zero.
- absorptionPower now is an opacity value per the thickness distance
- opacity is cumulative and nonlinear
- for now to get the volumetric shader you MUST use "volume_texture" with a colormap as shown below
- for a solid color use a colormap where all colors are the same
- the "environment" parameter is used to control how much the exit rays contribute to the final hit color

Solid Texture

The solid texture shader is the normal sss shader and can use map files as the volume shader can below.

```
shader {
  name "sssShdr2"
  type sss
  solid_texture
  map 2
    0 1 1 0.0
    1 1 1 1.0
  function 0
  size 10
  scale 3.142
  opacity 0.0625
  thickness 0.0125
  sssRays 5
  specular 0.2 0.2 0.2 80
  phongRays 5
  maxOpacity 0.3
  diffusion 0.125
  sampDist 0.0625
  environment 0.75
}
```

Volume Texture

Although the quality is much better and more accurate, the volume shader is slower, so be very careful how high you set the "sssRays" parameter. I suggest starting with values of 3-5 until you get a feel for it. You also have to have the trace-depths for "diff" and "refr" set to a minimum of 2, but setting them any higher will slow things down a lot. Here's the syntax for using a Fractint pallet map:

```
shader {
    name "sssShader01"
    type sss
    volume_texture
        mapfile "C:\Documents and Settings\Don\My Documents\Apophysis\Apophysis-080318-201.map"
        function 0
        size 0.25
        scale 3.142
    opacity    0.12
    thickness  0.12
    sssRays    12
    specular   0.2   0.2   0.2   0
    phongRays  3
    maxOpacity 0.999
    diffusion  0.999
    sampDist   0.12
    environment 0.75
    maxSubRayLength 2.5
    sssTraceDepth 1
}
```

Here's the syntax for using your own map:

```
shader {
    name "sssShader01"
    type sss
    volume_texture
        map 7
        1.0 1.0 1.0 0.0
        1.0 1.0 1.0 0.35
        0.6 0.0 0.0 0.425
        1.0 1.0 1.0 0.5
        0.75 0.7 0.0 0.575
        1.0 0.0 0.5 0.65
        1.0 1.0 1.0 1.0
        function 0
        size 0.25
        scale 3.142
    opacity    0.12
    thickness  0.12
    sssRays    12
    specular   0.2   0.2   0.2   0
    phongRays  3
    maxOpacity 0.999
    diffusion  0.999
    sampDist   0.12
    environment 0.75
    maxSubRayLength 2.5
    sssTraceDepth 1
}
```

Diff

You can find the diff for this here (.diff)

(https://fractrace.dev.java.net/files/documents/6137/91229/sss_Solid_Shader_Mar24-08.diff) . The big ticket item for this code is the SolidTexture.java where the procedural magic happens. It's also important to note that the diff contains the fake ambient term modification which isn't required for the shader.

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Procedurally_Textured_SSS"

- This page was last modified on 5 August 2009, at 01:45.

Cameras

From Sunflow Wiki

Contents

- 1 Cameras
 - 1.1 Pinhole
 - 1.2 Thinlens
 - 1.2.1 Depth of Field
 - 1.2.2 Bokeh
 - 1.3 Spherical
 - 1.4 Fisheye
- 2 Camera Motion Blur
- 3 Camera Shutter

Cameras

Sunflow has four camera types: Pinhole, thinlens, spherical, and fisheye. I'll go into the syntax used for each and a few points that need to be made.

A main aspects of all the cameras is the eye, target, and up values. Though obvious, I'll go over what they are. Eye is where in the world space the eye of the camera is, the target is where is the workspace the camera is looking at, and the up value dictates in what direction (in terms of the workspace) the camera sees as up (basically how the camera is rotated). Cameras can also use matrix transforms in place of the eye, target, and up values such as

```
transform col myMatrixReadByColumn
transform row myMatrixReadByRow
```

Cameras can also have motion blur added.

Pinhole

Probably what people would consider the "standard" perspective camera.

```
camera {
  type pinhole
  eye 7.0 -6.0 5.0
  target 6.0 -5.0 4.0
  up -0.30 0.30 0.90
  fov 49.134
  aspect 1.333
}
```


The SVN (0.07.3) has a neat feature added to the pinhole, which is camera shifting. It basically takes the perspective shot you have and shifts the view in the x or y without ruining the perspective. I would start with small values for shift. It looks like this:

```
camera {  
    type pinhole  
    eye 7.0 -6.0 5.0  
    target 6.0 -5.0 4.0  
    up -0.30 0.30 0.90  
    fov 49.134  
    aspect 1.333  
    shift 0.1 0.2  
}
```

Thinlens

Depth of Field

The thinlens camera is our depth of field (dof) camera, which is also capable of doing bokeh effects. If you activate dof it's better to "lock" the AA (e.g. an AA of 2/2) avoiding adaptive sampling. For a thinlens without bokeh you use:

```
camera {  
    type thinlens  
    eye 7.0 -6.0 5.0  
    target 6.0 -5.0 4.0  
    up -0.30 0.30 0.90  
    fov 49.134  
    aspect 1.333  
    fdist 30.0  
    lensr 1.0  
}
```

Depth of field is also one of the three things (the others being motion blur and object motion blur) that are directly affected by samples in the image block:

```
image {  
    resolution 400 225  
    aa 0 0  
    samples 3  
    filter gaussian  
}
```

Bokeh

If you want to use bokeh, you would add two attributes to the end (the number of sides and the rotation of the effect):

```
camera {
```

```
}
    type thinlens
    eye 7.0 -6.0 5.0
    target 6.0 -5.0 4.0
    up -0.30 0.30 0.90
    fov 49.134
    aspect 1.333
    fdist 30.0
    lensr 1.0
    sides 6
    rotation 36.0
}
```

As with the pinhole camera, shifting has been added as an option. It looks like this:

```
camera {
    type thinlens
    eye 7.0 -6.0 5.0
    target 6.0 -5.0 4.0
    up -0.30 0.30 0.90
    fov 49.134
    aspect 1.333
    shift 0.2 0.2
    fdist 30.0
    lensr 1.0
    sides 6
    rotation 36.0
}
```

Spherical

The spherical camera produces a longitude/latitude environment map.

```
camera {
    type spherical
    eye 7.0 -6.0 5.0
    target 6.0 -5.0 4.0
    up -0.30 0.30 0.90
}
```

Fisheye

A classic lens.

```
camera {
    type fisheye
    eye 7.0 -6.0 5.0
    target 6.0 -5.0 4.0
    up -0.30 0.30 0.90
}
```

Camera Motion Blur

Camera motion blur is available in the current 0.07.2 release and you can see some example images below in the second post. Object motion blur has been added to the SVN (0.07.3) and if you are looking to use it, you'll need to compile it from the source.

Camera motion blur is also one of the three things (the others being dof and object motion blur) that are directly affected by samples in the image block, so if it's not there, you'll want to add it or the default of 1 is used. For example:

```
image {
    resolution 400 225
    aa 0 2
    samples 3
    filter gaussian
}
```

So if your're not getting a smooth results you might want to try increasing the samples or reducing the step increases (as described below).

Camera motion blur consists of different steps of the transform and those steps differing is aspects of the transform. For example, here is a camera with a 3 step motion blur with the x up vector changing:

```
camera {
    type pinhole
    steps 3
    {
        eye 1.3 -0.9 1.1
        target 0.6 -0.4 0.7
        up 0 0 1
    }
    {
        eye 1.3 -0.9 1.1
        target 0.6 -0.4 0.7
        up 0.1 0 1
    }
    {
        eye 1.3 -0.9 1.1
        target 0.6 -0.4 0.7
        up 0.2 0 1
    }
    fov 49.1343426412
    aspect 1.33333333333
}
```

You may use this with the other camera lens types as well. Eye, target and up vector are free to vary at each step. You can also specify transformation matrices directly using:

```
steps <n>
    transform row myMatrixReadByRow
    transform row myOtherMatrixReadByRow
    ...
```

or

```
steps <n>
  transform col myMatrixReadByColumn
  transform col myOtherMatrixReadByColumn
  ...
```

You can also specify transforms for the non-motion blurred case (when "steps n" is omitted). You can give the raw data in column/row major form, or via a series of translate/scale/rotate commands. Let me know if you need more details/examples on that part."

Camera Shutter

Sunflow 0.07.3 has added functionality to the camera, allowing you to change the shutter time (the times over which the moving camera is defined (uniform spacing is assumed)) which in 0.07.2 is clamped to [0,1]. For object motion blur to work the camera in the scene needs the shutter line added:

```
camera {
  type pinhole
  shutter 0 1
  eye -18.19 8.97 -0.93
  target -0.690 0.97 -0.93
  up 0 1 0
  fov 30
  aspect 1.77
}
```

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Cameras>"

- This page was last modified on 22 December 2007, at 20:05.

Lights

From Sunflow Wiki

Contents

- 1 Lights
 - 1.1 Samples
- 2 Attenuated Lights
 - 2.1 Point Light
 - 2.2 Meshlight/Area Light
- 3 Non-Attenuated Lights
 - 3.1 Spherical Lights
 - 3.2 Directional Lights
- 4 Infinitely Far Away Lights
 - 4.1 Image Based Light
 - 4.2 Sunsky
- 5 Showboating Lights
 - 5.1 Cornell Box (0.07.3 Only)

Lights

Keep in mind that for colors the syntax sRGB nonlinear is used in these examples. These values are the color space most of us are used to. If you prefer, you can use other color spaces which I outline in the shader overview.

It's also important to note that at this time IBL and Sunsky do not emit photons, therefore, no caustics are viewable when using these lights. All other lights can emit photons.

Samples

Samples, samples, samples. Samples are key for the lights that use them. If they are too low, your image will look bad. If they are too high, your render will take forever. So it's important to experiment to get the right setting for your scene. I suggest starting with low samples, and working your way up till you reach just the right amount.

Attenuated Lights

The point and area lights are attenuated, meaning that the farther away you go away from the light, the less power/influence on the scene it has. Also remember that you can use the power/radiance in negative numbers to remove light from the scene.

Point Light

```
light {
    type point
    color { "sRGB nonlinear" 1.000 1.000 1.000 }
    power 100.0
    p 1.0 3.0 6.0
}
```

For the point light, power is measured in watts.

Meshlight/Area Light

```
light {
    type meshlight
    name meshLamp
    emit { "sRGB nonlinear" 1.000 1.000 1.000 }
    radiance 100.0
    samples 16
    points 4
        0.6 0.1 6.0
        0.3 1.0 6.0
        1.0 1.0 5.5
        1.0 0.3 5.5
    triangles 2
        0 1 2
        0 2 3
}
```

Any mesh can become a light. For mesh lights you specify the radiance (which is watts/sr/m²), and the total power is radiance*area*pi (in watts if your area was m²). The example above is a simple square. Area lights/mesh lights automatically create soft shadows and you control the quality of shadows by changing the samples value. An important thing to keep in mind is that the light samples are per face (per triangle), so the more complicated the mesh the more it will take to render. For this reason a simple two triangle quad is probably the way to go. See this thread (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=128>) for more information.

For large area lights, it's much better to let them be sampled by the indirect rays (diffuse and glossy) than to try and use the light sampling to get a good result. In these cases in 0.07.2, you can set the meshlight samples to 0 (so they won't be counted by direct lighting). The 0.07.3 SVN version of Sunflow will not count the lights at all if you do this. But with a simple change to the SVN source you can get the lights to be counted by the indirect rays. Change the getRadiance method in the area light to this and compile a new Sunflow:

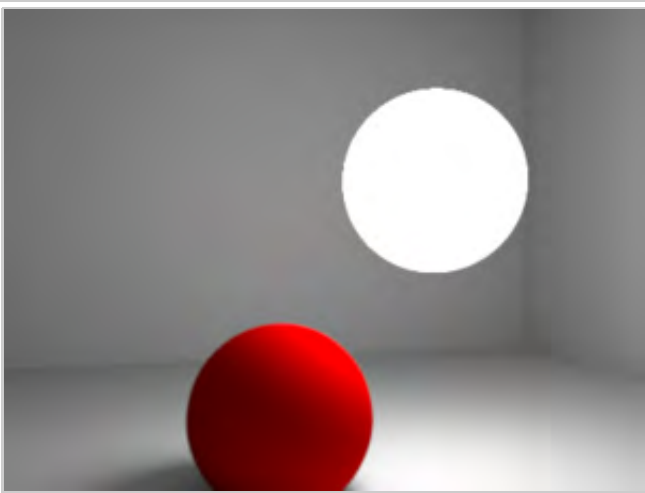
```
public Color getRadiance(ShadingState state) {
    if (numSamples > 0 && !state.includeLights())
        return Color.BLACK;
    state.faceforward();
    // emit constant radiance
    return state.isBehind() ? Color.BLACK : radiance;
}
```

Non-Attenuated Lights

Spherical Lights

```
light {  
    type spherical  
    color { "sRGB nonlinear" 1.000 1.000 1.000 }  
    radiance 100.0  
    center 5 -1.5 6  
    radius 30  
    samples 16  
}
```

The appearance of sphere lights is greatly influenced by the anti-aliasing filter you use. See the Image Settings page for examples.



Example of a spherical light, using 16 samples, radius=1

Directional Lights

```
light {  
    type directional  
    source 4 1 6  
    target 3.5 0.8 5  
    radius 23  
    emit { "sRGB nonlinear" 1.000 1.000 1.000 }  
    intensity 100  
}
```

Inifinitely Far Away Lights

For IBL/Sunsky power, the exact power is measured from the content of the map or the sun position. For IBL it will also depend on a correctly calibrated image.

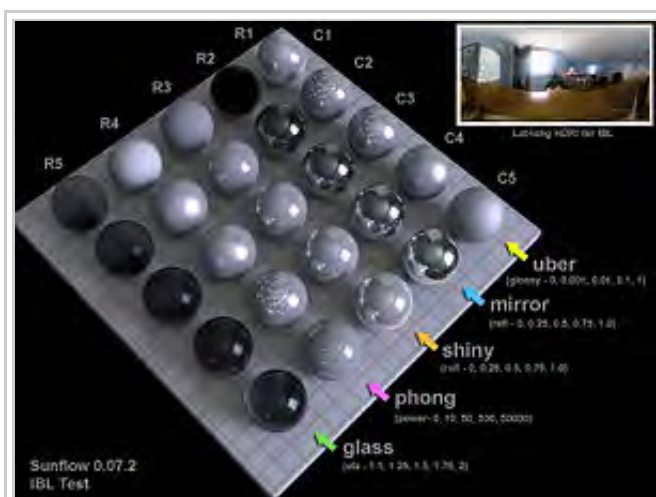
Image Based Light

```
light {  
    type ibl  
    image "C:\mypath\myimage.hdr"  
    center 0 -1 0  
}
```

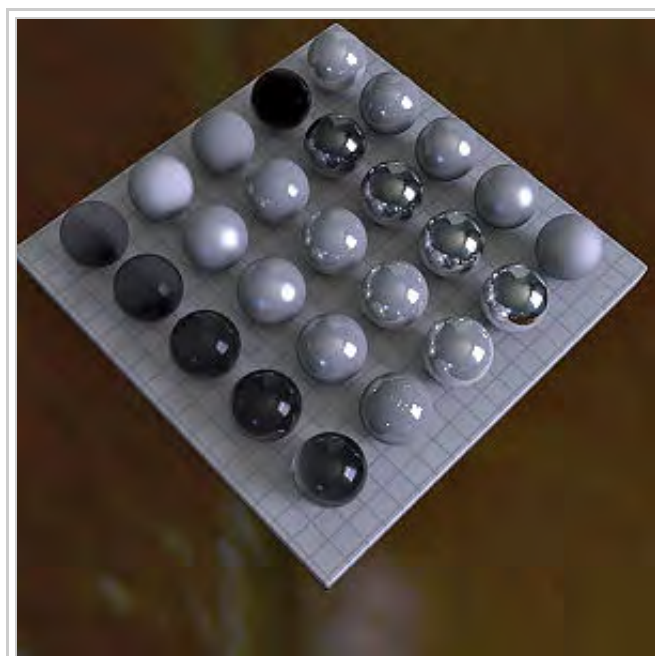
```
up 0 0 1
lock true
samples 200
}
```

The image based light is designed for use with high dynamic range images. The "center" vector is a world space direction that points towards the center pixel in the map, letting you rotate the map however you want. The "up" orients the map relative to that axis. This is to be able to support multiple 3d applications: some like to have Y pointing up, others Z.

An important feature of this light is that we can turn importance sampling on or off. Using lock false (importance sampling) forces use of unique samples per shading point instead of fixing them over the whole image. Which basically means that when lock false is set, the points of the image that will affect the light result the most (i.e. more "important") will be sampled more.



Importance sampling turned off (lock true) for a variety of shaders with various settings.



Importance sampling turned on (lock false).

You can find the files that were used to create the above image here (.zip)
(<http://www.geneome.net/other/otherfiles/IBLTest.zip>) .

So a conclusion one could draw from these tests is that when using the phong and uber shaders with high power and glossy values respectively, using importance sampling can reduce the light points from the ibl. Increasing samples does not rid the phong and uber shaders of, what Kirk referred to as, "constellations."

You don't have explicit control over the handedness of the rotation. If it looks like your image is coming in flipped, just do that in your favorite hdr image editor.

Sunflow only supports lon/lat HDR maps at the moment. So if you have your images in another format (spherical probe or cube-map) you'll need to do some kind of conversion in another program (like HDRShop).

Sunsky


```
light {
    type sunsky
    up 0 0 1
    east 0 1 0
    sundir 0.5 0.2 0.8
    turbidity 6.0
    samples 128
}
```

There isn't a setting in the syntax that controls sun intensity, but you can instead control the suns direction in terms of angle to the object. So if the Sunsky direction is at a near 0 degree angle with the object (the sun on the horizon) it will be dark and the sky will be more a sunset color. If the direction is more high in the sky at around 80 degrees it will be bright with the sky being white/blue. Changing the up and east values can also change the look, but these are more used to change how the Sunsky is interpreted in different world spaces which might be required in different applications. The up and east values in the above example usually work for everyone.

The Sunsky light has a set horizon where the sky stops and the blackness of the world shows up. Normally an infinite plane is the work-around. Future versions of Sunflow might have a control to extend the sky, but you can also modify the source and compile Sunflow yourself so the sky extends on its own. In src.org.sunflow.core.light.SunSkyLight.java go to the line that says

```
groundExtendSky = false;
```

Change it to

```
groundExtendSky = true;
```

Compile Sunflow and the sky will no longer terminate at the horizon.

Showboating Lights

The Cornell Box isn't really a light that you would use in a typical scene but it is useful to illuminate your models. Plus, in the SVN source's plugin registry it's listed under lights now so there you have it, it must be a light ;). The code block specifies that it's an object in 0.07.2 (as the example show below) but in 0.07.3 and up it will be a light so you would use "light {" rather than "object {".

Cornell Box (0.07.3 Only)

In 0.07.3 and up, the cornell box is a light. Prior to this, it was considered a primitive.

```
light {
    type cornellbox
    corner0 -60 -60 0
    corner1 60 60 100
    left 0.80 0.25 0.25
    right 0.25 0.25 0.80
    top 0.70 0.70 0.70
    bottom 0.70 0.70 0.70
    back 0.70 0.70 0.70
}
```

```
emit      15 15 15  
samples 32
```

Let's do a quick run through. The size of the box is defined by `corner0` and `corner1`. `Corner0`'s `x y z` values corresponds to position of the lower left corner of the box closest to us. `Corner1` is the back top right corner. In the example above, the center of the world (0,0) is in the center of the floor. The color of the sides of the box are then defined. The `emit` and `samples` values are just like the `meshlight` (above).

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Lights>"

- This page was last modified on 7 March 2009, at 21:29.

GI

From Sunflow Wiki

Contents

- 1 Global Illumination
 - 1.1 Instant GI
 - 1.2 Irradiance Caching (Final Gathering)
 - 1.3 Path Tracing
 - 1.4 Ambient Occlusion
 - 1.5 Fake Ambient Term
- 2 Overriding

Global Illumination

Sunflow supports 5 main global illumination (gi) types: Instant GI, Irradiance Caching (aka Final Gathering), Path Tracing, Ambient Occlusion, and Fake Ambient Term. Remember that only one gi type at a time can be used in the scene.

Instant GI

This type of gi is based on this paper (pdf) (<http://graphics.uni-ulm.de/Singularity.pdf>) (page 6 really explains everything in a nice way with pictures). Hopefully, my understanding of the method is okay enough that I can adequately interpret the settings. What happens is that random points are added to the scene then a ray is traced from a position in the scene to those points to determine the radiance of that position. If rays coming from that point don't meet a particular criteria (set by the bias), a ray is scattered until it does meet the criteria.

```
gi {  
  type igi  
  samples 64  
  sets 1  
  b 0.01  
  bias-samples 0  
}
```

The samples values is the number of samples (virtual photons) used per set. Increasing the samples gives smoother results, but increases render time.

The number of sets seems to be the number of sets of virtual photons emitted. The more sets, the more points, and the more points added to the calculation, the more noisy less illuminated areas become.

The % bias, or *b*, is used for the estimate of direct illumination. If this value is too high, you'll see spots of light where the illumination is pooling, but if it's too low you'll lose indirect illumination contributions. This value should be above zero, but usually low numbers do the trick. Try starting with 1 and work your way down in big steps.

If the % bias is too high, you'll see spots of light in your scene, most often in the corners or other nooks and crannies. Decrease the bias to get rid of them. This then can begin to reduce the amount of light in those corners since you are effectively reducing the amount of rays finding their way to the corners. With bias-samples, you can increase the amount of samples of those bias-diverted rays to get some of that light back into the corners. A value of 0 will not increase the sampling so you'll keep the slightly darker corners, but increasing the value to 1 or greater will give unbiased sampling results. To me, this is one of the key features of the paper, so definitely experiment with this value. Be forewarned, the larger you make this number the slower your scene will render.

You can also manipulate the number of bounces using the diffuse trace depths by adding:

```
trace-depths {  
    diff 1  
    refl 4  
    refr 4  
}
```

A diff of 1 means 1 bounce, a diff of 2 means two bounces, and so on. The more bounces, the longer your render will take. The refl and refr trace-depths are used for glass shaders.

Irradiance Caching (Final Gathering)

In a final gather, a hemisphere above a point is sampled by shooting rays at it. These rays bounce off the hemisphere. Secondary bounces are then calculated with either path tracing rays or global photon maps. Here is what the syntax for irradiance caching looks like when using path tracing for secondary bounces:

```
gi {  
    type irr-cache  
    samples 512  
    tolerance 0.01  
    spacing 0.05 5.0  
}
```

The samples values is the number of samples (virtual rays) used to calculate the irradiance.

The tolerance option indicates how much error you will allow in the calculation. If the tolerance is set high, the calculation will be faster, but the result will not be as good. If the tolerance is set low, the calculation will be slower, but the result will be better.

The spacing option is used to determine a minimum and maximum distance from the sampled point that will be used to look for other initial ray bounces. The points found within this range will be used to determine what the irradiance intensity should be at the initial point.

Using the above syntax, secondary bounces from the point will be calculated using path tracing, which can be pretty slow. To speed it up, you can instead use a global photon map using the following syntax:

```
gi {
    type irr-cache
    samples 512
    tolerance 0.01
    spacing 0.05 5.0
    global 1000000 grid 100 0.75
}
```

The only difference is the addition of the global line. The increase in speed from using global photons is based on pre-computing local irradiance values at the many photon positions. In Sunflow, you set the number of global photons you want to use and how you want the photons mapped, either in a grid or a kd tree. The grid is a better way to go, but at least you know the kd option is available (though using kd might throw an exception and cause the scene to stop rendering in some cases). Then you need to define the global estimate and radius (in the example above the estimate is 100 and the radius is 0.75). These values are used at a single secondary bounced photon (of the many photons used). At that point, a sphere with a radius (global radius) expands outward to encompass a certain number of other photons (global estimate) to be used to determine the irradiance at that point. For global estimates, typically 30 to 200 photons are used.

Path Tracing

Probably the most recognized and classic way of doing true global illumination. Sunflow's implementation only handles diffuse inter-reflection and will not produce any caustics. There's not much to say about it except that is probably the most accurate, but slowest, gi method. It usually gives out a noisy image unless your samples are really high. Its implementation is very straight forward:

```
gi {
    type path
    samples 32
}
```

To calculate how many samples to use, the [Sunflow FAQ (<http://home.comcast.net/~gamma-ray/sf/sunflow-faq.htm>)] recommends setting the anti-aliasing to 0 0 in the image block, then adjusting the gi samples until the noise disappears. Finally, increase the anti-aliasing, avoiding adaptive anti-aliasing, and divide the samples in the gi block according to the amount of over-sampling you are doing (4, 16, etc.)

You can also manipulate the number of bounces using the diffuse trace depths by adding:

```
trace-depths {
    diff 1
    refl 4
    refr 4
}
```

So a diff of 1 means 1 bounce, a diff of 2 means two bounces, and so on. The more bounces you have the longer your render will take. The refl and refr trace-depths are used for glass shaders.

Ambient Occlusion

What more can I say than it's ambient occlusion. The settings are pretty straight forward:

```
gi {
    type ambocc
    bright { "sRGB nonlinear" 1 1 1 }
    dark { "sRGB nonlinear" 0 0 0 }
    samples 32
    maxdist 3.0
}
```

Fake Ambient Term

Get some quick ambient light in your scene using this one. You can find the reference for this here (<http://www.cs.utah.edu/~shirley/papers/rtrt/node7.html>) .

```
gi {
    type fake
    up 0 1 0
    sky { "sRGB nonlinear" 0 0 0 }
    ground { "sRGB nonlinear" 1 1 1 }
}
```

Overriding

Like shader overrides, you can also override the global photons and global illumination to render only these feature's contribution to the scene (so you can fine tune your settings). To view global photons you would use:

```
shader {
    name debug_globals
    type view-global
}

override debug_globals false
```

To view the gi in the scene, you would use:

```
shader {
    name debug_gi
    type view-irradiance
}

override debug_gi false
```

To render the scene normally all you would need to do is comment out the override line:

```
% override debug_gi false
```

or

```
% override debug_globals false
```

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=GI>"

- This page was last modified on 8 March 2009, at 18:21.

Mesh

From Sunflow Wiki

Contents

- 1 Triangle Meshes
 - 1.1 Syntax
 - 1.2 Transforms
 - 1.3 Face Shaders/Modifiers
 - 1.4 Object Motion Blur
- 2 Bezier Patches
 - 2.1 Bezier Format Example
 - 2.2 Transforms
- 3 Quad Meshes

Triangle Meshes

Syntax

I'll list the general format below, then give examples of each variation. It's important to understand that in these cases, a point is equal to a vertex.

```
object {  
  shader default  
  type generic-mesh  
  name meshName  
  points X  
    x y z  
    ...  
  triangles X  
    A B C  
    ...  
  normals none/vertex/facevarying  
  uvs none/vertex/facevarying  
}
```

For triangles, it is important to note that the triangle indices are listed using the point numbers starting with 0 as the first point. So if you have three points/vertices defined, the triangle values would look like 0 1 2. Given this way of indexing the triangles based on the number of points, there cannot be a triangle index number above ((the number of points) – 1). So in the below examples, you can't have a triangle with an index of 3.

For normals and uvs, you have the option of using none, vertex, or facevarying coordinates. For none, no normals/uvs will be used. The normals and uvs don't have to use the same type to work, so you can have "normals vertex" and "uvs facevarying" or "normals facevarying" and "uvs none". A key point is that if you are using the

vertex type you need the same number of values that there are points. For facevarying, you need the same number of values equal to the number of triangles (see below for examples).

For the vertex type, each point in the mesh will need its own normal or uv coordinate:

```
...
points 3
  x1 y1 z1
  x2 y2 z2
  x3 y3 z3
triangles 1
  0 1 2
normals vertex
  d1 e1 f1
  d2 e2 f2
  d3 e3 f3
uvs vertex
  u1 v1
  u2 v2
  u3 v3
}
```

For facevarying you would use this format:

```
...
points 3
  x1 y1 z1
  x2 y2 z2
  x3 y3 z3
triangles 1
  0 1 2
normals facevarying
  d1 e1 f1 d2 e2 f2 d3 e3 f3
uvs facevarying
  u1 v1 u2 v2 u3 v3
}
```

Note that I don't define the number of vertex normals or uvs like I do for the points and triangles since point and triangle numbers will determine how many normal and uv coordinates I will need.

Transforms

Want to rotate your object around a particular normal? Then in your mesh that has vertex normals, after the shader/modifier call use the syntax:

```
transform {
  rotate x y z d
}
```

Where x, y, and z are the normal coordinates and d is the degrees of rotation about that normal.

What about if you want to transform a vertex based object without having to re-export the object. No problem:

```
object {
  shader default
  transform {
    rotatex 60.0
    translate 3.2 1.2 0.8
    ...
  }
  type generic-mesh
```

For transform, you can use or not use the transform options: translate, rotatex, rotatey, rotatez, scalex, scaley, scalez, and scaleu. You could also use a transform matrix by row (transform row) or column (transform col).

```
object {
  shader default
  transform col my4x4MatrixReadByColumn
  type generic-mesh
```

Keep in mind that the values are relative to the original position of the object.

Face Shaders/Modifiers

If you want to assign multiple shaders or modifiers to different faces on the same mesh, you can do that like so:

```
object {
shaders 2
  shaderName0
  shaderName1
modifiers 2
  bumpName0
  "None"
type generic-mesh
name meshName
points 6
  x1 y1 z1
  x2 y2 z2
  x3 y3 z3
  x4 y4 z4
  x5 y5 z5
  x6 y6 z6
triangles 2
  0 1 2
  1 2 3
normals none
uvs facevarying
  u1 v1 u2 v2 u3 v3
  u4 v4 u5 v5 u6 v6
face_shaders
  0
  1
}
```

In the face shader section you are assigning shader 0 (the first shader in the shader list - shaderName0) to the first triangle in the triangle list, shader 1 to the second triangle list, etc. It's the same with modifiers, but remember that for modifiers with textures (bump/normal map) you need uvs assigned. Also, if you don't have a modifier for a face, just use "None" in the list.

Object Motion Blur

Sunflow 0.07.3 has added functionality to the camera, allowing you to change the shutter time which in 0.07.2 is clamped to [0,1]. For object motion blur to work the camera in the scene needs the shutter line added:

```
camera {
  type pinhole
  shutter 0 1
  eye -18.19 8.97 -0.93
  target -0.690 0.97 -0.93
  up 0 1 0
  fov 30
  aspect 1.77
}
```

Once that's enabled, you would blur the object with the following syntax (similar to camera motion blur), but with the added "times" line, which is the time over which the motion is defined. In the below example, I am blurring the object to simulate it traveling some distance (translation).

```
object {
  shader someShader
  modifier bumpMap
  transform
  steps 3
  times 0 1
  {
    rotatex -90
    scaleu 0.018
    rotatey 245
    translate 1.5 0 -1
  }
  {
    rotatex -90
    scaleu 0.018
    rotatey 245
    translate 1.5 0 -1.5
  }
  {
    rotatex -90
    scaleu 0.018
    rotatey 245
    translate 1.5 0 -2
  }
  type teapot
  name myTeapot
}
```

Note that you can use row (transform row) or column (transform col) transform matrices as well.

```
object {
  shader someShader
  modifier bumpMap
  transform
  steps 3
  times 0 1
  transform col my4x4MatrixReadByColumn1
  transform col my4x4MatrixReadByColumn2
  transform col my4x4MatrixReadByColumn3
}
```

```
type teapot
name myTeapot
```

Motion blur is also one of the three things (the others being dof and camera motion blur) that are directly affected by samples in the image block, so if it's not there, you'll want to add it or the default of 1 is used. For example:

```
image {
  resolution 400 225
  aa 0 2
  samples 3
  filter gaussian
}
```

Bezier Patches

```
object {
  shader shaderName
  type bezier-mesh
  n X Y
  wrap false false
  points x y z x y z...
```

Where n is the number of cv's in u and v, wrap is equivalent to renderman's uwrap/vwrap option (optional and defaults to false), and points are the cv data and should be in the same order as in the renderman spec and there should be exactly $3*(u*v)$ values.

Bezier Format Example

```
object {
  shader myShader
  type bezier-mesh
  n 4 7
  wrap false false
  points 3.2 0 2.25 3.2 -0.15 2.25 2.8 -0.15 2.25 2.8 0 2.25 3.45 0 2.3625 3.45 -0.15
  2.3625 2.9 -0.25 2.325 2.9 0 2.325 3.525 0 2.34375 3.525 -0.25 2.34375 2.8 -0.25 2.325
  2.8 0 2.325 3.3 0 2.25 3.3 -0.25 2.25 2.7 -0.25 2.25 2.7 0 2.25 2.4 0 1.875 2.4 -0.25
  1.875 2.3 -0.25 1.95 2.3 0 1.95 3.1 0 0.675 3.1 -0.66 0.675 2.6 -0.66 1.275 2.6 0
  1.275 1.7 0 0.45 1.7 -0.66 0.45 1.7 -0.66 1.275 1.7 0 1.275
```

Transforms

As with other objects you can use transforms on a bezier object just as described in the file-mesh transforms section. For transforming, you can use or not use the transform options: translate (as in translate x y z), rotatex, rotatey, rotatex, scalex, scaley, scalez, scaleu. You could also use a transform matrix by row (transform row) or column (transform col).

Quad Meshes

Not in the .sc file format, but is in the source and is available in the .sca format.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Mesh>"

- This page was last modified on 23 December 2007, at 01:00.

FileMesh

From Sunflow Wiki

Contents

- 1 File-Meshes
- 2 Transforms
- 3 Object Motion Blur

File-Meshes

You can import an obj, stl, or ra3 file automatically into Sunflow without having to convert it to Sunflow's file format:

```
object {  
  shader myShader  
  type file-mesh  
  name objectName  
  filename test.obj  
  smooth_normals true  
}
```

The filename can also be an absolute path.

The smooth_normals line is optional, it defaults to false (which is best if you have millions of triangles. Note that the stl format does not store mesh connectivity so it isn't possible to smooth their normals.

Sunflow doesn't accept an obj file if there are texture coordinates and normals stored in it. For example, a line like f 1/1/1 2/2/2/ 3/3/3 in the obj file doesn't work. Certain exporters will allow you to not have this information exported to the file. This limitation can be overcome using NeuroWorld's obj to .sc converter (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=356>) .

Transforms

It's important to note that objects (including file meshes) can also be transformed. The transform line goes under the shader declaration (or geometry declaration if it is an instance) in the object syntax. For transforms, you can use or not use the following transform options:

```
translate (translate x y z)  
rotate x  
rotate y  
rotate z
```

```
scalex  
scaley  
scalez  
scaleu
```

You could also use a transform matrix by row (row) or column (col). For example, here is one using some of the above transforms:

```
object {  
    shader myShader  
    transform {  
        translate 1.0 4.6 0.3  
        scaleu 0.5  
    }  
    type file-mesh  
    name objectName  
    filename myOBJ.obj  
    smooth_normals true  
}
```

And one using matrices:

```
object {  
    name nameOfObject  
    shader shaderForObject  
    transform col my4x4MatrixReadByColumn  
    ...  
}
```

Object Motion Blur

As with generic meshes, in the 0.07.3 SVN version of Sunflow file meshes can be object blurred.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=FileMesh>"

- This page was last modified on 13 December 2007, at 21:24.

Primitives

From Sunflow Wiki

Contents

- 1 Transforms
- 2 Object Motion Blur
- 3 Gumbo
- 4 Teapot
- 5 Background
- 6 Plane
 - 6.1 Infinite Plane I
 - 6.2 Infinite Plane II
- 7 Sphere
- 8 Hair
- 9 Julia
- 10 Particle
 - 10.1 In 0.07.2
 - 10.2 In 0.07.3
 - 10.3 Other Method
- 11 Banchoff Surface
- 12 Torus
- 13 Cube Grid
- 14 Box
- 15 Janino Primitive
- 16 Cornell Box (0.07.2)
- 17 Cylinder (0.07.3)
- 18 Sphere Flake (0.07.3)

Transforms

As with other objects you can use transforms just as described in the file-mesh transforms section. For transforming, you can use or not use a lot of the transform options: translate (as in translate x y z), rotatex, rotatey, rotatex, scalex, scaley, scalez, scaleu. You could also use a transform matrix by row (row) or column (col).

Object Motion Blur

As with generic meshes, in the 0.07.3 SVN version of Sunflow primitives can be object blurred.

Gumbo

```
object {  
  shader simple_green  
  transform {  
    rotatex -90  
    scaleu 0.1  
    rotatey -25  
    translate 1.5 0 +1.5  
  }  
  type gumbo  
  name gumbo_3  
  subdivs 6  
  smooth false  
}
```

Teapot

```
object {  
  shader simple_yellow  
  transform {  
    rotatex -90  
    scaleu 0.008  
    rotatey 245  
    translate -1.5 0 -3  
  }  
  type teapot  
  name teapot_1  
  subdivs 4  
  smooth false  
}
```

Background

```
background {  
  color { "sRGB nonlinear" 0.057 0.221 0.400 }  
}
```

Plane

You can define an infinite plane in two ways.

Infinite Plane I

```
object {  
  shader floor  
  type plane  
  p 0 0 0  
  p 4 0 3  
  p -3 0 4  
}
```

The first p is the center of the plane, and the following two points on the plane. If you use 3 points to define a plane instead of a single point and a normal (as described below) you will get texture coordinates on the infinite plane.

Infinite Plane II

```
object {  
    shader floor  
    type plane  
    p 0 0 0  
    n 0 1 0  
}
```

The p is the center of the plane, and the n is the direction of the normal.

Sphere

```
object {  
    shader Mirror  
    type sphere  
    name mirror  
    c -30 30 20  
    r 20  
}
```

Hair

```
object {  
    shader blue  
    type hair  
    segments 3  
    width .1  
    points 12 0.0 0.0 0.0 0.0 0.5 0.0 0.0 1.0 0.0 0.0 1.5 0.0  
}
```

You put hair strands one after the other. For example, if you have 3 segments, this means each strand will need 4 vertices. So if you specify 20 points, you should end up with 5 distinct hair strands.

Julia

```
object {  
    shader simple1  
    transform { scalex 2 rotatey 45 rotatex -55 }  
    type julia  
    name left  
    q -0.125 -0.256 0.847 0.0895  
    iterations 8  
    epsilon 0.001  
}
```

The line `q` is the main julia set parameter and defines its shape. The iterations and epsilon affect the speed and accuracy of the calculation but are not required. If you comment these two lines out you will use the high quality defaults.

Particle

In 0.07.2

```
object {
    shader grey
    type particles
    little_endian
    filename "c:/your_folder/particles.dat"
    radius 0.03
}
```

`particles.dat` should be a binary file, ie: not human readable. The file should be exactly $4 \times 3 \times n$ bytes long, where n is the number of particles. The `little_endian` line is optional and will make the byte order little endian rather than the default big endian.

In 0.07.3

```
object {
    shader grey
    type particles
    points 5
        0.5 0.2 0.3
        0.6 0.2 0.3
        0.7 0.2 0.3
        0.8 0.2 0.3
        0.9 0.2 0.3
    radius 0.03
}
```

The filename method still works obviously, it will be more efficient for large amounts of particles (the files can now be specified relative to the include path).

Other Method

Here is a workaround if you're using 0.07.2 and don't have a dat file handy. Create a `.java` file alongside your `.sc` file with the following contents:

```
import org.sunflow.core.primitive.ParticleSurface;
import org.sunflow.system.Parser;

public void build() {
    parse("your_scene.sc"); // the name of your scene file goes here
    try {
        Parser p = new Parser(resolveIncludeFilename("your_ascii_filename.dat")); // the name of your parti
        int n = p.getNextInt();
        float[] data = new float[3 * n];
    }
```

```

for (int i = 0; i < data.length; i++)
data[i] = p.getNextFloat();
p.close();
parameter("particles", "point", "vertex", data);
parameter("num", data.length / 3);
parameter("radius", 0.1f); // the radius of the particles goes here
geometry("particle_object_name", new ParticleSurface());
parameter("shaders", "shader_name"); // replace with the shader name you want to use
instance("particle_object_name.instance", "particle_object_name");
} catch (Exception e) {
e.printStackTrace();
}
}
}

```

Just render this .java file from the command line instead of the main .sc file. All the paths can be relative, you only need to specify the filename as long as they are all in the same folder... I also assumed that your ascii file contains the number of points in the first line.

Banchoff Surface

```

object {
  shader default-shader
  transform {
    rotatex -90
    scaleu 5
    rotatey 25
    translate 0 0 0
  }
  type banchoff
  name myShape
}

```

A banchoff surface in the .sc format is a bit different because it is pre-defined so it has no settings. This is overcome by using transforms as in the above example. I've also had shading issues with it but haven't investigated/asked about it.

Torus

```

object {
  shader myShader
  type torus
  r 2 4
}

```

Where the first r value is the inner radius and the second is the outer radius.

Cube Grid

The cube grid primitive is a unit block in the creation of unique shapes. You need to use some clever java to really use it, so take a look at the menger sponge example and the isosurface example.

Box

The box primitive isn't accessible in the .sc format but can be created in java like any other primitive. I've added it myself to the sc file format in 0.07.3 by using this diff (.diff) (<http://www.geneome.net/drawer/sunflow/boxSCparser.diff>) . This method creates a box which you can manipulate with transforms.

Janino Primitive

Like shaders, you can define a primitive via janino.

```
object {
  name myJaninoObject
  type janino-tesselatable
  <code>
  Your code goes here
  </code>
}
```

Cornell Box (0.07.2)

In 0.07.3 and up the cornell box is considered a light. In 0.07.2, it is considered an object and so you would use the syntax below.

```
object {
  shader none
  type cornellbox
  corner0 -60 -60 0
  corner1 60 60 100
  left 0.80 0.25 0.25
  right 0.25 0.25 0.80
  top 0.70 0.70 0.70
  bottom 0.70 0.70 0.70
  back 0.70 0.70 0.70
  emit 15 15 15
  samples 32
}
```

Let's do a quick run through. This "object" doesn't really have/need a shader so it's set to none. Setting a shader won't do anything. The size of the box is defined by corner0 and corner1. Corner0's x y z values corresponds to position of the lower left corner of the box closest to us. Corner1 is the back top right corner. In the example above, the center of the world (0,0) is in the center of the floor. The color of the sides of the box are then defined. The emit and samples values are just like the meshlight.

Cylinder (0.07.3)

```
object {
  shader default-shader
  transform {
```

```
    rotatex -90
    scaleu 5
    rotatey 25
    translate 0 0 0
}
type cylinder
name myShape
}
```

A cylinder in the .sc format is a bit different because it is pre-defined so it has no settings. This is overcome by using transforms as in the above example.

Sphere Flake (0.07.3)

```
object {
  shader metal
  type sphereflake
  name left
  level 7
  axis 2 3 4
  radius 0.5
}
```

Level, axis, and radius are optional. When defined, the axis line is a vector.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Primitives>"

- This page was last modified on 2 January 2008, at 17:28.

Instances

From Sunflow Wiki

Contents

- 1 Instances
- 2 Transforms
- 3 Object Motion Blur

Instances

Instancing is when you use the geometry of another object in a scene without having to duplicate the object's mesh data. Instancing an object is useful in reducing the memory overhead, scene file size, and giving control over the objects since you change the main object and those mesh changes will be propagated to all the instances. Instances in Sunflow are great because you can change the location, rotation, scale, shader, and modifier of each instance. In Sunflow 0.07.2, you would use the following to instance the geometry:

```
instance {  
  name nameOfInstance  
  geometry theOriginalObjectName  
  transform {  
    rotatex -90  
    scaleu 1.0  
    translate -1.0 3.0 -1.0  
  }  
  shader shaderForInstance  
  modifier modForInstance  
}
```

Of course, the modifier line is optional.

Transforms

As with other objects you can use transforms just as described in the file-mesh transforms section. For transforming, you can use or not use a lot of the transform options: translate (as in translate x y z), rotatex, rotatey, rotatez, scalex, scaley, scalez, scaleu. You could also use a transform matrix by row (row) or column (col).

If you are using Blender, the exporter can take duplgrouped objects in the scene and export them out as instances. I go over this in the Blender exporter usage page here.

Object Motion Blur

As with generic meshes, in the 0.07.3 SVN version of Sunflow instances can be object blurred.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Instances>"

- This page was last modified on 13 December 2007, at 21:12.

Modifiers

From Sunflow Wiki

Contents

- 1 Modifiers
 - 1.1 Bump Map
 - 1.2 Normal Map
 - 1.3 Perlin Noise (SVN 0.07.3 only)

Modifiers

There are 3 object modifiers in Sunflow. A bump, normal, and perlin. At this time, only one modifier type can be applied to an object at a time.

The modifiers are applied to objects by adding a line to the object (or instance) below the shader declaration. Here is an example of a sphere with a modifier:

```
object {  
  shader default  
  modifier modName  
  type sphere  
  c -30 30 20  
  r 20  
}
```

Also note that textures can also be in relative paths:

```
texture texturepath/mybump.jpg
```

Bump Map

```
modifier {  
  name bumpName  
  type bump  
  texture "C:\texturepath\mybump.jpg"  
  scale -0.02  
}
```

In 0.07.2 the bump scale is set for very small values and actually is in the negative scale. In the SVN 0.07.3 version of Sunflow, the scale has been fixed making it positive, but also magnifying the effect of the values by a thousand. So a 0.07.2 value of -0.02 would be equivalent to 20 in 0.07.3. This magnification was due to finding

(<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=233>) that for a bump scale in most images, a value of around 0.001 was usually needed. So in 0.07.3, this value is now the more reasonable 1 value.

Normal Map

```
modifier {  
    name normalName  
    type normalmap  
    texture "C:\texturepath\mynormal.jpg"  
}
```

Perlin Noise (SVN 0.07.3 only)

```
modifier {  
    name perlinName  
    type perlin  
    function 0  
    size 1  
    scale 0.5  
}
```

The "size" parameter affects as you might guess how spread-out or tight the bumps are. However larger values make a tighter texture (smaller, coarser bumps), and smaller values spread it out more (bigger shallower bumps).

The scale parameter affects the height of the bumps.

A function parameter of 0 results in a generally uniform noise function. A value of 1 gives a striped function along the x axis. Any other value for the function parameter performs a turbulence function.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Modifiers>"

- This page was last modified on 27 December 2007, at 17:14.

Waiting

From Sunflow Wiki

You might have read on the forum or seen in the 0.07.3 source that there is a new file format in the upcoming versions. This is indeed the case. There will be an ascii (.sca) and binary (.scb) format that is more in line with the way parameters are written in the source, opening up new possibilities (like defining width of hair objects over the length of the strand), and making way for future feature integration (like different scene settings being turned off and on via the command line).

Here is the likely .sca syntax format based on Chris's examples on the forum for Sunflow versions above 0.07.3.. Keep in mind that this syntax will not be recognized in the current SVN version of Sunflow and these pages will likely get an overhaul later since most of this is my best guess of what is to come.

- Image Settings
- Shaders
- Cameras
- Lights
- Global Illumination
- Mesh Objects
- File Meshes
- Primitives
- Instances
- Modifiers

I've also started working on a Blender exporter for the new version.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Waiting>"

- This page was last modified on 12 January 2008, at 02:25.

Image sca

From Sunflow Wiki

Contents

- 1 Image Block
 - 1.1 Anti-Aliasing
 - 1.2 Samples
 - 1.3 Contrast Threshold
 - 1.4 Filters
 - 1.5 Jitter
- 2 Bucket Size/Order
- 3 Banding
- 4 Tone Mapping

Image Block

The image block dictates the resolution of the image as well as a few other image options. The samples and jitter lines are optional and if they are left out samples will equal 1, contrast will be 0.1, and jitter will be false will be used as the default.

```
options ::options {  
    param resolutionX int 800  
    param resolutionY int 600  
    param aa.min int 0  
    param aa.max int 2  
    param aa.samples int 4  
    param aa.contrast float 0.1  
    param filter string gaussian  
    param aa.jitter bool false  
}
```

Anti-Aliasing

The aa line is the settings for the adaptive anti-aliasing. These control the under/over-sampling of the image. The image will be first sampled at the rate prescribed by minimum aa value (the first value). Then, based on color and surface normal differences, the image will be refined up to the rate prescribed by the maximum aa value (the second value). Taken from the Sunflow ReadMe file:

A value of 0 corresponds to 1 sample per pixel. A value of -1 corresponds to 1 sample every 2 pixels (1 per 2x2 block) A value of -2 corresponds to 1 sample every 4 pixels (1 per 4x4 block) A value of -3 corresponds to 1 sample every 8 pixels (1 per 8x8 block) ... A value of 1 corresponds to 4 samples per pixel (2x2 subpixel grid) A value of 2 corresponds to 16 samples per pixel (4x4 subpixel grid) A value of 3 corresponds to 64 samples per pixel (8x8 subpixel grid) ...

Examples:

```
- quick undersampled preview:      -2 0
- preview with some edge refinement: 0 1
- final rendering:                  1 2
```

You can turn adaptive anti-aliasing off by setting the min and max values to the same number. For example, an aa of 0 0 will be 1 sample per pixel with no subpixels.

You can see a video on explaining what adaptive anti-aliasing is here (XviD .avi) (<http://www.geneome.net/other/videotutorials/AdaptiveSamplingInSunflow-XviD.avi>) .

Samples

Samples are the number of samples. Surprised? When used they indirectly affect many aspects of the scene but directly affects DoF and camera/object motion blur.

Contrast Threshold

There is a line in the image block in which you can change the default contrast threshold. This affects the point at which the renderer decides to adaptively refine between four pixels when doing adaptive anti-aliasing. This line isn't required and the default is usually the right setting. I personally don't see a change in the render with different values.

Filters

If you are oversampling the image (i.e., having the min and max aa positive numbers) you will likely want to use more advanced filters to control the look of the image. The available filters are:

```
box (filter size = 1)
triangle (filter size = 2)
gaussian (filter size = 3)
mitchell (filter size = 4)
catmull-rom (filter size = 4)
blackman-harris (filter size = 4)
sinc (filter size = 4)
lanczos (filter size = 4)
bspline (filter size = 4)
```

Check this page (http://gardengnomesoftware.com/dll_patch.php) out for a good overview of filters. Triangle and box are better for previews since they are faster. The other filters are recommended for final image rendering (my personal favorite is mitchell).

Jitter

The jitter line (either true or false) turns jitter on or off. Jitter moves the rays randomly a small amount to reduce aliasing that might still be present even when anti-aliasing is turned on. So jittering these pixels makes the aliasing issues become less perceptible. Jitter should be turned off (false) when doing animations because the randomness of jitter makes it very obvious it's on when moving from frame to frame.

Bucket Size/Order

You change the bucket order in the scene file just like you can in the command line. In the scene file, if you want to change the order from the default hilbert (for which you don't need to add any line) you would type the following as it's own line in the .sc file:

```
options ::options {  
    param bucket.size int 64  
    param bucket.order string column  
}
```

If you want a reverse order it would look like this:

```
options ::options {  
    param bucket.size int 64  
    param bucket.order string "reverse spiral"  
}
```

A larger bucket size means more RAM usage and less time rendering. Usually, a bucket size 64 is a good default - especially if you are using a wide pixel filter like gaussian or mitchell. There are six bucket order types available: hilbert (default), spiral, column, row, diagonal, and random. You can also use these ordered in reverse by adding "reverse" in front of the name. To use reverse, you'll need to use quotes around the reverse order so that the bucket order is still parsed as one token. The number in the line is the pixel size of the each bucket. There is some clamping done internally to make sure the bucket size isn't too small or too big.

Banding

Banding (or "posterization") is the effect where, instead of having a smooth color gradient, a portion of the image shows distinct steps or bands of colors. This is particularly evident in backgrounds with large areas of a single, slowly varying color.

This effect occurs because the 8 bits per channel in the rendered image are insufficient to accurately represent the continuous gradation of color, and are instead rounded (or "clipped") to a nearby color value. The result is that values jump from one integer to the next after staying constant for several pixels. Unfortunately, the human eye is particularly sensitive to edge transitions such as these.

In this case the only solution found so far is to apply dithering (which only seems to be found in Photoshop) when converting to 24 bit RGB. This essentially applies random noise to the image, which makes it harder to see the transitions.

Tone Mapping

If you find yourself looking to tone map or you intend to tone map your image before it's even rendered, you may want to consider this tip. Render a bigger image (at least 2x in each direction), then you can reduce the number of samples since the resize (to your originally intended size) will blur the noise. This will avoid having the problem of the high intensity pixels filtering too far out when Sunflow does its filtering. Ideally you would render with aa samples set to 0 0 (fixed at 1 sample per pixel) and a really big image. Then do your tonemapping on that big image and then you can resize to final resolution. The OpenEXR output driver is probably the better choice (vs .hdr) if you want to do this since it can handle arbitrary image sizes (it writes a bucket at a time).

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Image_sca"

- This page was last modified on 7 January 2008, at 03:58.

Shaders sca

From Sunflow Wiki

Contents

- 1 Introduction
- 2 Color Spaces
- 3 Shader Override
- 4 Samples
- 5 Linking
- 6 Shaders
 - 6.1 Textures
 - 6.2 Constant Shader
 - 6.3 Diffuse Shader
 - 6.3.1 Textured Diffuse Shader
 - 6.4 Phong Shader
 - 6.4.1 Textured Phong Shader
 - 6.5 Shiny Shader
 - 6.5.1 Textured Shiny Shader
 - 6.6 Glass Shader
 - 6.6.1 Absorption
 - 6.6.2 Trace Depths
 - 6.7 Mirror Shader
 - 6.8 Ward Shader
 - 6.8.1 Textured Ward Shader
 - 6.9 Ambient Occlusion Shader
 - 6.9.1 Textured Ambient Occlusion Shader
 - 6.10 Thin Glass
 - 6.10.1 Texture Thin Glass
 - 6.10.2 Trace Depths
 - 6.11 Uber Shader
 - 6.12 Wire Frame Shader
 - 6.13 Command Line Shaders
- 7 Janino Shaders
 - 7.1 Mix Shader
 - 7.2 Fresnel Shader
 - 7.3 Stained Glass
 - 7.4 Shiny Shader with Reflection Mapping
 - 7.5 Simple SSS
 - 7.6 Specular Pass
 - 7.7 Translucent Shader
 - 7.8 Wire Shader

- 8 Source Code Shaders
 - 8.1 True SSS
 - 8.2 Textured SSS

Introduction

After a post asking for help regarding texturing shaders, I thought I would go over each of the available shaders in Sunflow 0.07.2.

For the bump, normal, and perlin modifiers, these aren't a part of shaders but rather have their own modifier syntax. For examples of how to use modifiers, see the modifiers page.

Color Spaces

Sunflow has 6 color spaces available to use in your shaders. On a few of these (internal, sRGB, and XYZ) you can increase the color values beyond 1.0 if you really want to saturate the effect.

- internal - requires 3 values
- sRGB nonlinear - requires 3 values
- sRGB linear - requires 3 values
- XYZ - requires 3 values
- blackbody - requires 1 value (temperature in Kelvins). See this page (<http://en.wikipedia.org/wiki/Blackbody>) for a good image of the color range.
- spectrum [min] [max] - any number of values (must be >0), [min] and [max] is the range over which the spectrum is defined in nanometers. This defines a spectrum from wavelengths X nm to Y nm with a number of regularly spaced values. Note that this spectrum will simply get converted to RGB, there's no spectral rendering in Sunflow at the moment.

Here are examples of each using the diffuse shader's diff line:

```
0.8 0.8 0.2
"sRGB nonlinear" 0.8 0.8 0.2
"sRGB linear" 0.8 0.8 0.2
"XYZ" 0.8 0.8 0.2
"blackbody" 1200
"spectrum 1 500" 3 6 9 190
```

Keep in mind that for colors the syntax for sRGB linear color space is used in the shader examples below. These values are what most of us are used to.

Shader Override

The name of these shaders are variable, so you can name them whatever you like. Names of the shaders are also used in "shader overriding." You can use any shader listed in your scene file (it doesn't have to be applied to an object) and with a single line make everything in your scene use that shader. To override, add this to the scene file:

```
options ::options {
    param override.shader string shaderName
}
```

If you want to scene to then be rendered as usual, simply comment out the block:

```
/*
options ::options {
    param override.shader string shaderName
}
*/
```

Samples

Samples, samples, samples. Samples are key for those shaders whose reflection quality are sample driven. If they are too low, the reflections will look bad. If they are too high, you're render will take forever. So it's important to experiment to get the right setting for your scene. I suggest starting with low samples, and working your way up till you reach just the right amount. I usually like a sample number of 4 for most of the shaders.

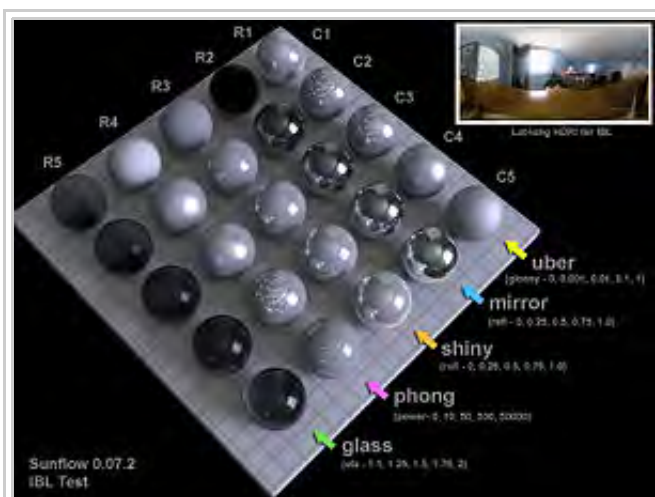
Linking

Linking allows us to take a shader or texture and link it into another shader or texture.

@@@See wireframe shader@@@

Shaders

If you want to see a good overview of some of the shaders and varying settings, I'll show Kirk's IBL test image:



IBL importance sampling turned off (lock true) for a variety of shaders with various settings.

Example scene for the shader examples below provided by olivS

(<http://feeblemind.tuxfamily.org/dotclear/index.php>) with some shader settings changed and ambocc gi used.

Textures

It's important to note that not all shaders can use textures, and those that can only use textures for their diffuse channel with the exception of the uber shader. The shaders that can handle textures are diffuse, phong, shiny, ward, ambocc, and uber. The image types that are recognized are tga, png, jpg, bmp, hdr, and igr. For textures, the objects must have UVs mapped. Also note that textures can also be in relative paths. Textures are initialized as a texture object:

```
texture myTex {  
    type color_texture_lookup  
    param filename string C:\myImage.jpg  
}
```

```
texture myTex {  
    type alpha_texture_lookup  
    param filename string C:\myImage.jpg  
}
```

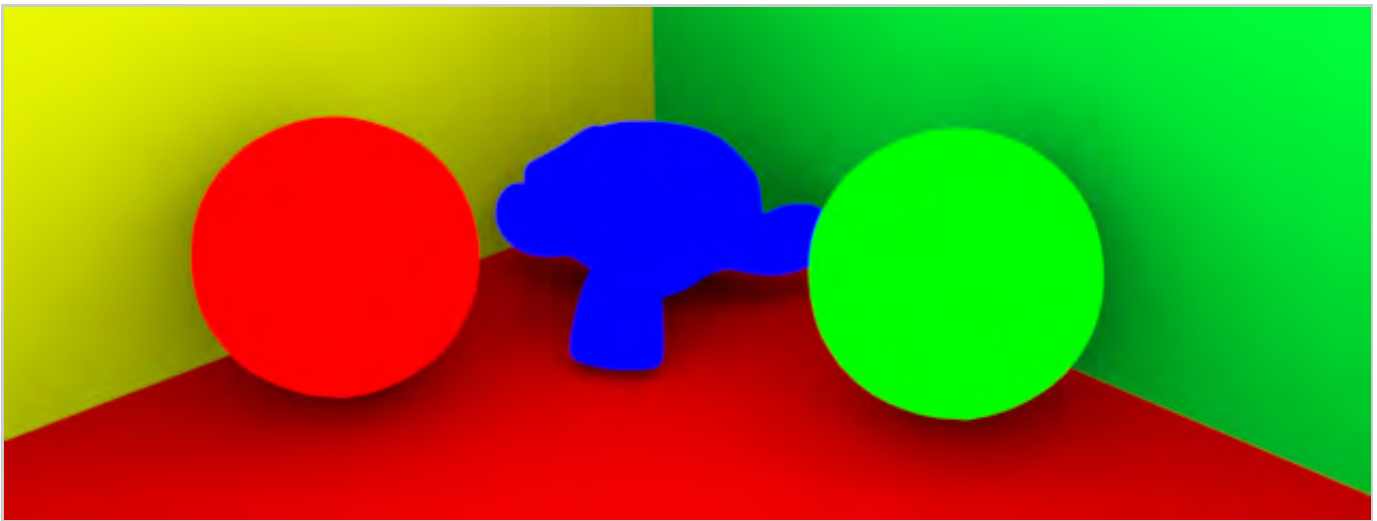
There is a "place2d" texture which lets you modify the scale/rotate/translate of a texture in uv space. This is useful for repeating textures or decals.

```
texture warp_stained_tex {  
    type place2d  
    param repeatU float 3.0  
    param repeatV float 2.0  
    param rotate float 32.0  
    param shiftU float 0.0  
    param shiftV float 0.0  
    param repeatBorderU bool false  
    param repeatBorderV bool true  
    param input string stained_tex  
    param outside color "sRGB linear" 0.2 0.3 0.4  
}
```

These texture objects can then be called in later shaders.

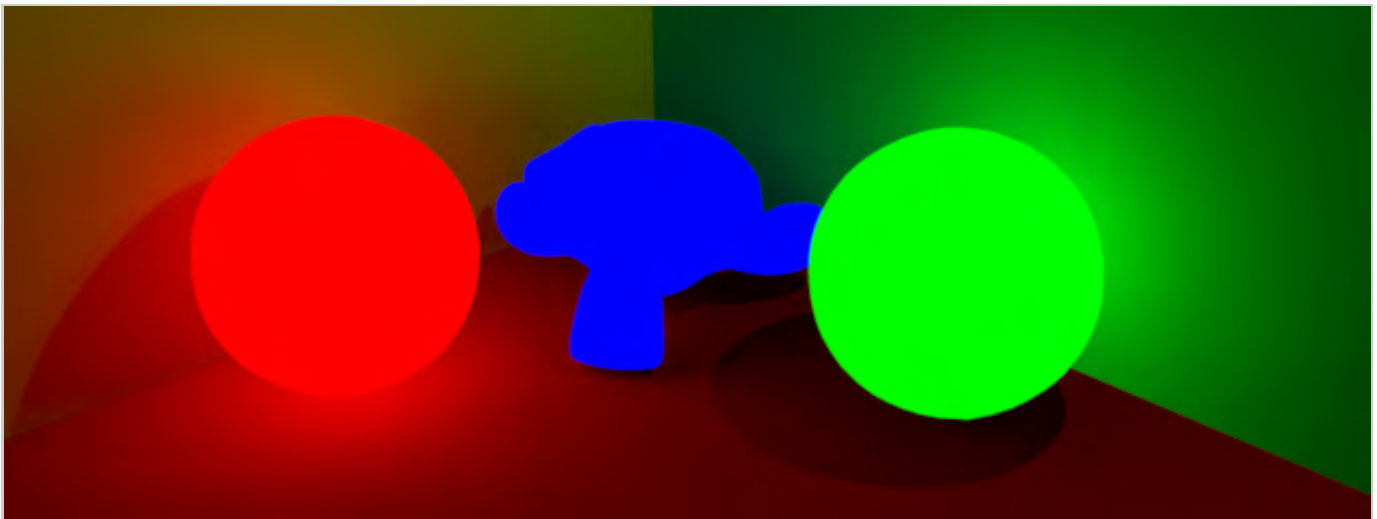
Constant Shader

```
shader sfcon.shader {  
    type constant  
    param color color "sRGB linear" 0.604 0.604 0.604  
}
```



Various colors showing the constant shader in action.

Aside from being used as a flat shader, the constant shader can also be used to fake lighting when path tracing is used by increasing the value of the color beyond 1.0.



This is the constant shader with color values above 1.0 with path tracing added to the scene.

It can also be used in conjunction with global illumination to get some interesting results (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=149>) .

Diffuse Shader

```
shader sfdif.shader {  
    type diffuse  
    param diffuse color "sRGB linear" 0.604 0.604 0.604  
}
```



Various diffuse shader colors. The walls are also the diffuse shader.

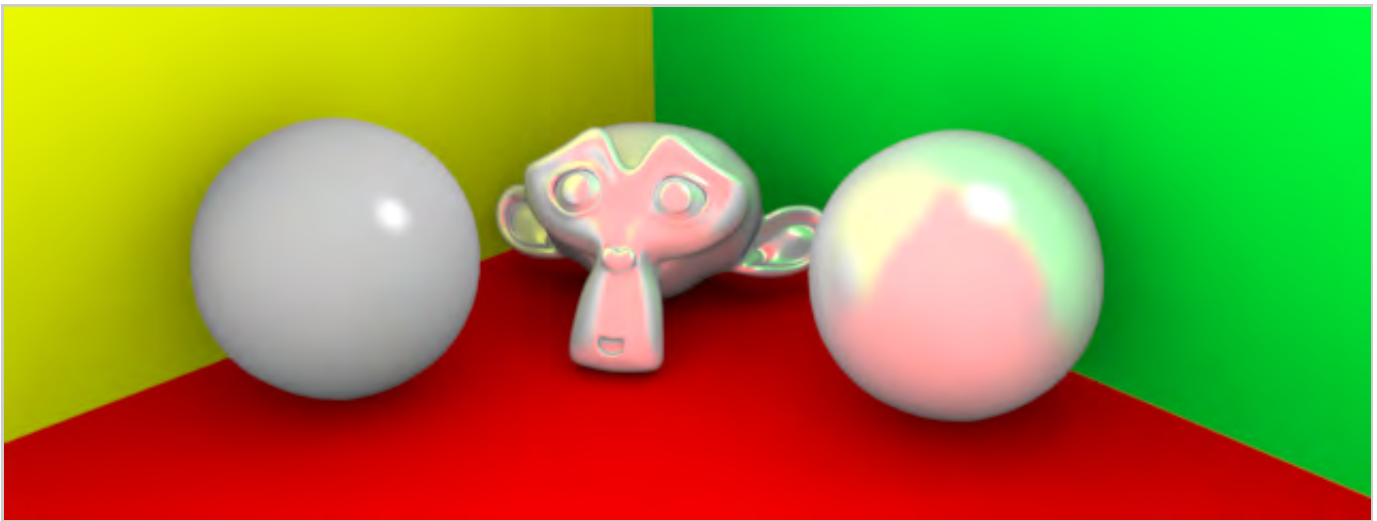
Textured Diffuse Shader

```
shader sfdif.shader {  
    type diffuse  
    param diffuse string myTex  
}
```

Phong Shader

```
shader sfpho.shader {  
    type phong  
    param diffuse color "sRGB linear" 0.604 0.604 0.604  
    param specular color "sRGB linear" 1.0 1.0 1.0  
    param power float 50.0  
    param samples int 4  
}
```

The number after the spec color values is the "power" or hardness of the specularity. You can crank it pretty high (e.g. 50000), so start with low values like 50 and work your way up or down from there. If you set the samples to 0, you'll turn off the indirect glossy reflections. If you set the samples to anything greater than 1, you'll get blurry reflections (with higher samples giving better reflections).



The sphere on the left has samples set to 0 (so no blurry reflections) whereas two right objects have samples >0 (so have blurry reflections).

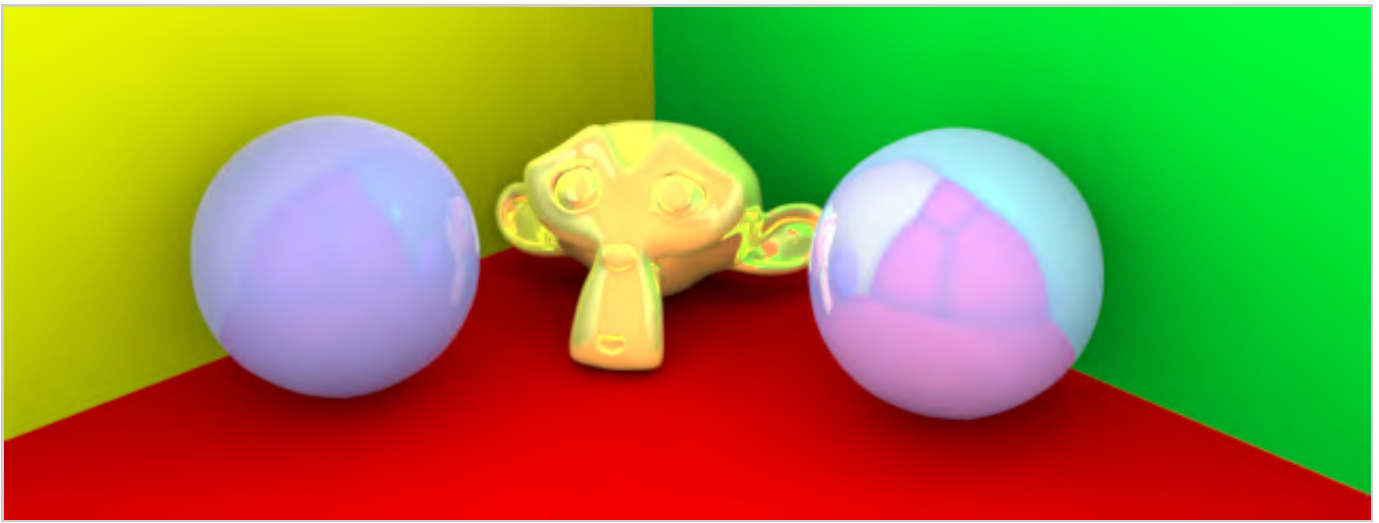
Textured Phong Shader

```
shader sfpho.shader {  
    type phong  
    param diffuse string myTex  
    param specular color "sRGB linear" 1.0 1.0 1.0  
    param power float 50.0  
    param samples int 4  
}
```

Shiny Shader

```
shader sfshi.shader {  
    type shiny_diffuse  
    param diffuse color "sRGB linear" 0.604 0.604 0.604  
    param shiny float 0.5  
}
```

Remember that you can go beyond 1.0 for the reflection value to really kick it up a notch - bam.



The sphere on the left has a lower reflection setting than the two objects on the right.

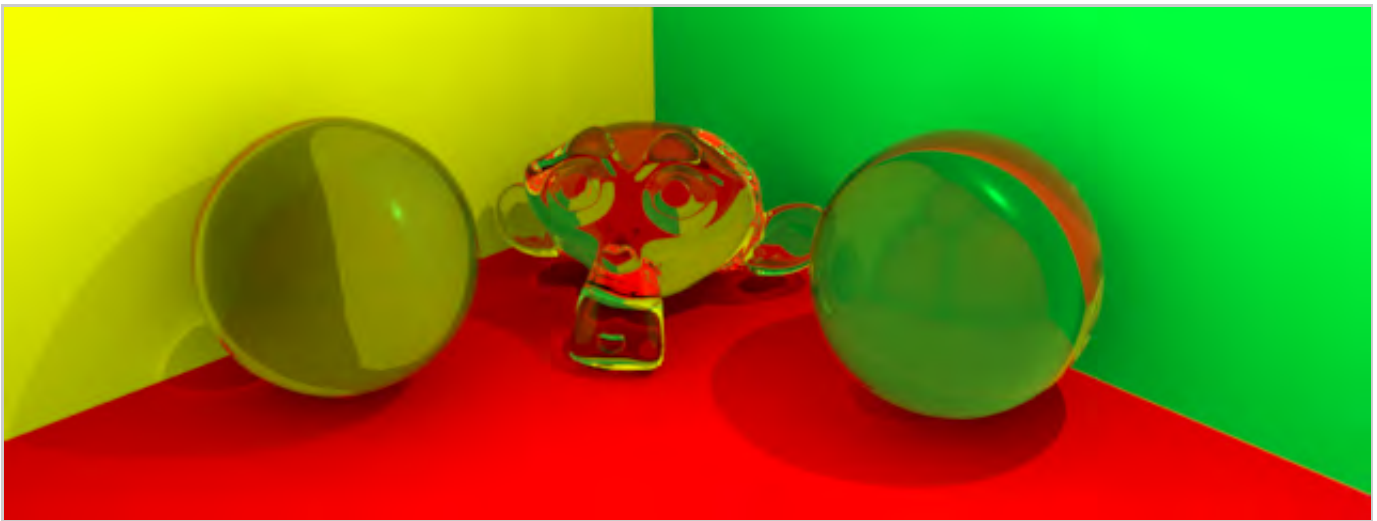
Textured Shiny Shader

```
shader sfshi.shader {  
    type shiny_diffuse  
    param diffuse string myTex  
    param shiny float 0.5  
}
```

Glass Shader

```
shader sfgla.shader {  
    type glass  
    param eta float 1.0  
    param color color "sRGB linear" 0.604 0.604 0.604  
    param absorption.distance float 5.0  
    param absorption.color color "sRGB linear" 1.0 1.0 1.0  
}
```

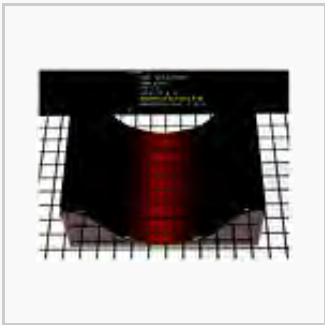
The "eta" is more commonly known as "ior" or index of refraction. It's important to note that without caustics, the glass shader will cast shadows like any other object.



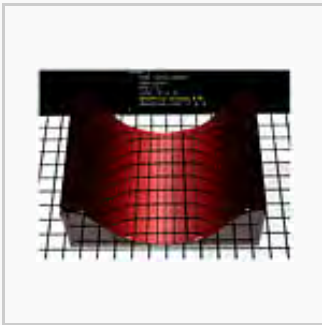
The sphere on the left has an eta of 1.33 (water) and the right two objects have an eta of 1.5 (glass).

Absorption

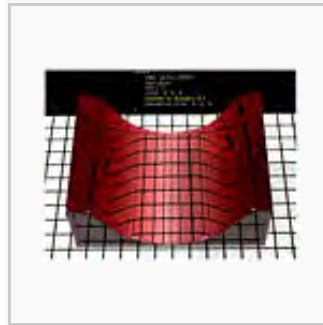
It's important to note that for the glass shader, the absorption lines are optional. Kirk did some tests using the absorption lines which I have added here:



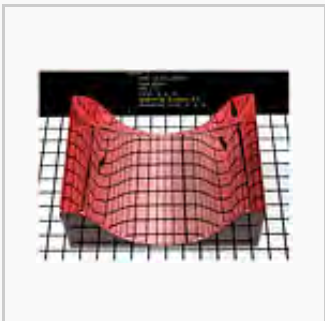
Absorption Distance =
0.01



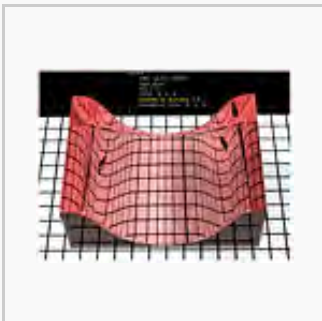
Absorption Distance =
0.05



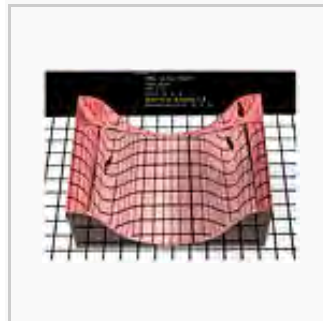
Absorption Distance =
0.1



Absorption Distance =
0.5



Absorption Distance =
1.0



Absorption Distance =
5.0

And for those savvy readers, I'll mention that the syntax's spelling of absorbtion isn't a typo. This spelling error has been fixed in the SVN so that Sunflow will recognize this spelling and the correct one.

Trace Depths

You can also manipulate the trace depths for glass by adding:

```
options ::options {
    param depths.diffuse int 1
    param depths.reflection int 4
    param depths.refraction int 4
    param depths.transparency int 10
}
```

If you don't add this to the scene file, the defaults of diff 1, refl 4, refr 4, and trans 10 will be used. For glass the two important ones are refl and refr. For a look at various glass related trace-depths in action see the below tests by Sielan:



Diff=1, Refl=1, Refr=1



Diff=2, Refl=2, Refr=2



Diff=5, Refl=5, Refr=5



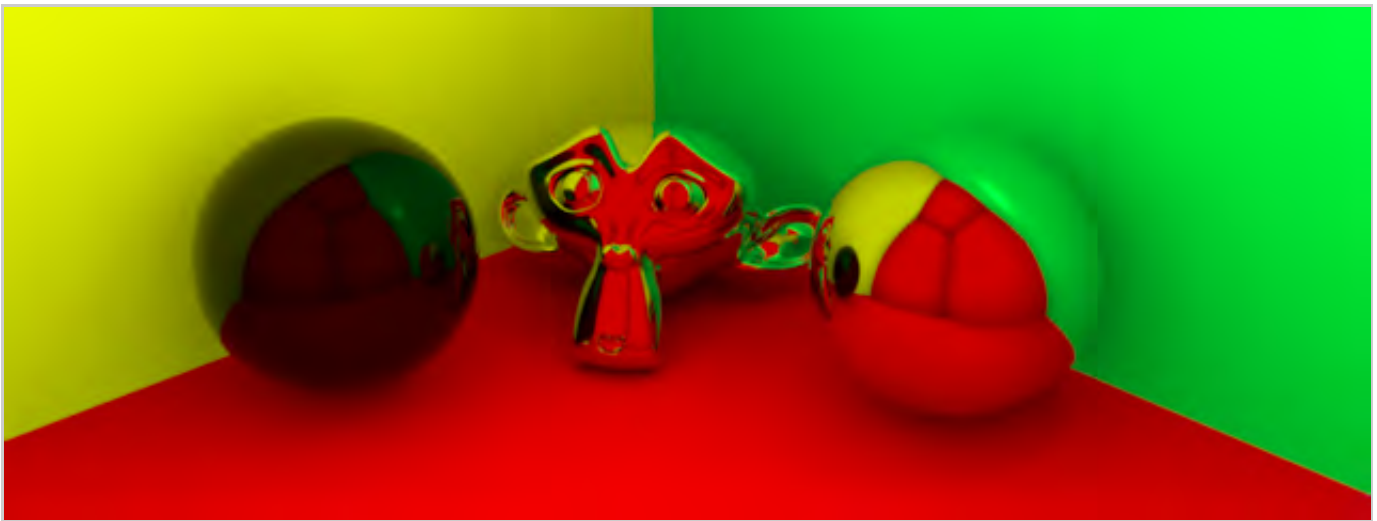
Diff=8, Refl=8, Refr=8



Diff=12, Refl=12,
Refr=12

Mirror Shader

```
shader sfmir.shader {
    type mirror
    param color color "sRGB linear" 0.604 0.604 0.604
}
```

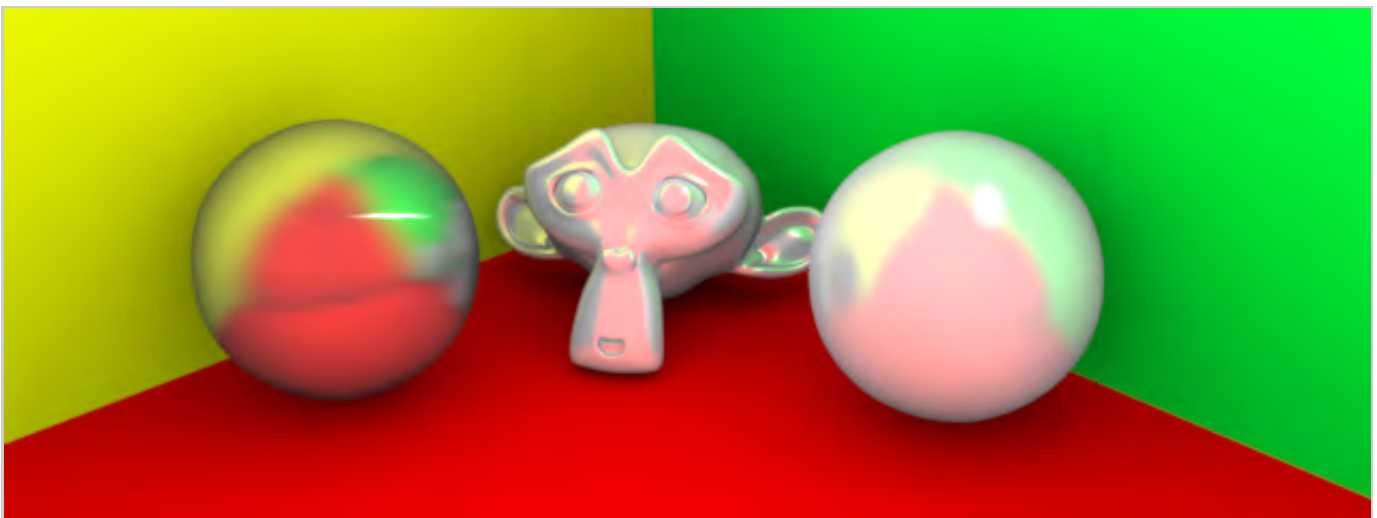


The sphere on the left has a darker color, the right two objects have a pure white color.

Ward Shader

```
shader sfwar.shader {  
    type ward  
    param diffuse color "sRGB linear" 0.604 1.0 0.604  
    param specular color "sRGB linear" 1.0 1.0 1.0  
    param roughnessX float 0.07  
    param roughnessY float 0.1  
    param samples int 4  
}
```

The x and y rough values are the amount of blurriness in the u and v tangent directions at the surface. The x and y rough values correspond to u and v tangent directions, so for the ward shader, you'll need to have uv coordinates defined on the object to get a proper result. If you set the samples to 0, you'll turn off the indirect glossy reflections. If you set the samples to anything greater than 1, you'll get blurry reflections (with higher samples giving better reflections).



The sphere on the left has the roughness and color different from the right two objects.

Some users of the ward shader are surprised that the result doesn't look like a similar shader in their other favorite application. That look having a metallic surface texture. This shader is a true ward shader, whereas other apps may add other effects to get the metal bump.

Note that the ward shader in 0.07.2 may cause a NaN error (black dots). This bug has been fixed in the the SVN 0.07.3 version.

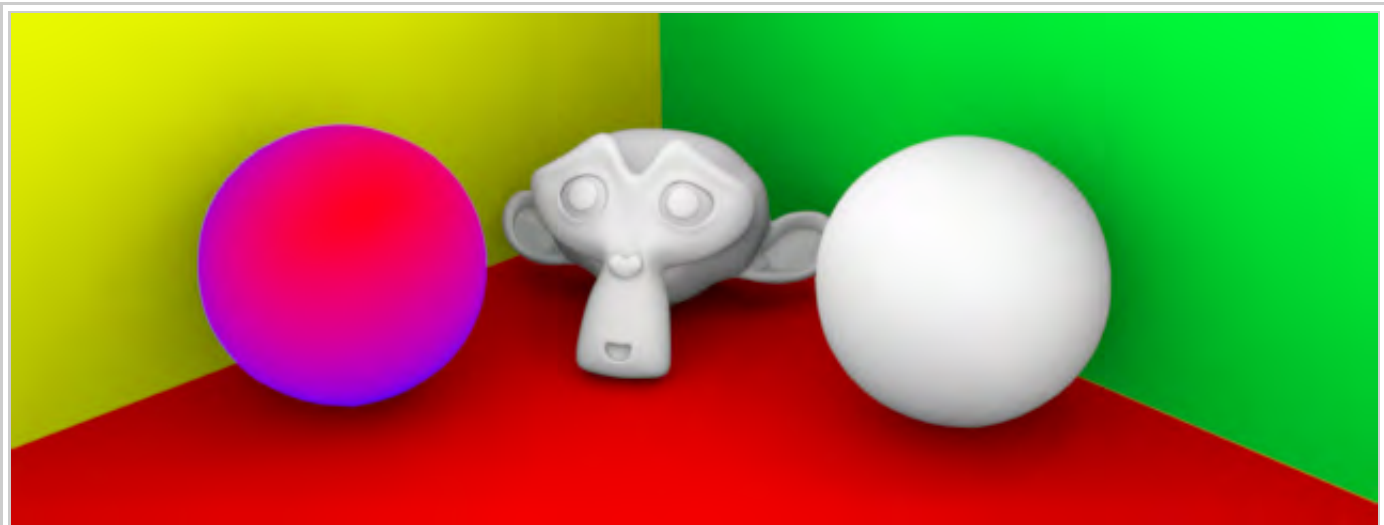
Textured Ward Shader

```
shader sfwar.shader {  
    type ward  
    param diffuse string myTex  
    param specular color "sRGB linear" 1.0 1.0 1.0  
    param roughnessX float 0.07  
    param roughnessY float 0.1  
    param samples int 4  
}
```

Ambient Occlusion Shader

```
shader sfamb.shader {  
    type ambient_occlusion  
    param bright color "sRGB linear" 0.0 0.0 0.0  
    param dark color "sRGB linear" 1.0 1.0 1.0  
    param samples int 32  
    param maxdist float 3.0  
}
```

Ambient occlusion shades based on geometric proximity - it totally ignores lights. The ambient occlusion gi engine does recognize lights.



The sphere on the left a different light and dark color, the right two objects have the classic light = white and dark = black color.

Textured Ambient Occlusion Shader

```
shader sfamb.shader {
    type ambient_occlusion
    param bright string myTex
    param dark color "sRGB linear" 1.0 1.0 1.0
    param samples int 32
    param maxdist float 3.0
}
```

The amb-occ textured shader has been tricky for me in that I've never had much success with it. Maybe someone will post to this thread how they've used it.

Thin Glass

```
shader glass {
    type thinglass
    param diffuse color "sRGB linear" 0.604 1.0 0.604
    param eta float 1.1
}
```

Texture Thin Glass

```
shader glass {
    type thinglass
    param color string myTex
    param eta float 1.1
}
```

Trace Depths

You can also manipulate the transparency trace depths for thinglass by adding:

```
options ::options {
    param depths.diffuse int 1
    param depths.reflection int 4
    param depths.refraction int 4
    param depths.transparency int 10
}
```

If you don't add this to the scene file, the defaults of diff 1, refl 4, refr 4, and trans 10 will be used. For glass the two important ones are refl and refr.

Uber Shader

The uber shader is a mix of several shaders to give you more options in controlling the look with the added bonus of being able to use a specular texture map. Though typically used with textures it can be used without both or on of the textures by removing the texture line(s).

```
shader sfuber.shader {
```

```

    type uber
    param diffuse color "sRGB linear" 0.604 0.604 0.604
    param diffuse.texture string myTex
    param diffuse.blend float 1.0
    param specular color "sRGB linear" 1.0 1.0 1.0
    param specular.texture string mySpecTex
    param specular.blend float 0.1
    param glossyness float 0.1
    param samples int 4
}

```

The blend values control the blending between the color and the texture. A blend value of 1 will make the texture the sole contributor, and a value of 0 will make only the color used. Any value in between will mix the texture and the color to produce the final result. The glossy setting, which gives you the shiny shader aspect, uses very large step values to achieve the result. So a value of zero is shiny, and a value of 1 has no glossyness. When playing with the settings try the following values: 0, 0.001, 0.01, 0.1, and 1. Also, if you don't need a diffuse or specular texture (for whatever reason) you can omit the diff.texture and/or spec.texture lines.

In the uber shader, setting the spec color to 0 0 0 and spec.blend to 0 will allow transparency, though this won't translate to transparent shadows at the moment.

Wire Frame Shader

The wireframe shader is now one of the "texture" shaders as well as a command line shader (which ust is the tessellated wireframe). It can be linked up to an ambient occlusion shader, diffuse shader, or any other shader you like. You can even plug one wireframe shader into another through linking. Here's an example using a black line and ambient occlusion where patches = true is a non tessellated wireframe.

```

texture myWireLine {
    type wireframe
    param line color "sRGB linear" 0.5 0.5 0.5
}

```

```

texture myWireFill {
    type wireframe
    param fill link myWireLine
    param patches bool true
}

```

```

shader sfamb.shader {
    type ambient_occlusion
    param bright link myWireFill
    param dark color "sRGB linear" 0.0 0.0 0.0
    param samples int 32
    param maxdist float 3.0
}

```

The first wireframe shader shows the tessellated edges, the next shows the patch edges, and the result of that is used at the "bright" color for the ambocc shader.

Command Line Shaders

Wireframe Shader/Normal Shader/UV Shader/ID Shader/Gray Shader/Prims Shader/Quick AO Shader. These shaders are actually accessed via the command line. I go over how to use these shader flags here.

Janino Shaders

You might ask the question "What is Janino?" Think of Janino as a Java compiler that only compiles when a janino shader is encountered. So if you write the shader in Java using Sunflow classes, you can come up with creative shaders without having to edit the source code directly. Some simple testing have shown that the janino shaders compile just as fast as if they were in the source. The syntax for this shader looks like this:

```
@@@
```

There are several Janino shaders written by forum members:

Mix Shader

Mark Thorpe wrote this mix shader which blends two shaders you define over an object. It was later found that running the shader on a multi-threaded machine caused this shader to return odd results which lead to the re-write/re-format by Don Casteel.

Fresnel Shader

Another mix shader, the fresnel shader uses the blend function in the api to mix two shaders you define based on the cosine between the shading normal and the ray. What's really cool is that you can change the blending just by defining your own value for c.

Stained Glass

Another elegant janino shader by Mark Thorpe, the stained glass shader gives a stained glass look by applying a texture to a glass shader. You need caustics turned on in your scene for this to work as intended.

Shiny Shader with Reflection Mapping

This shader is the shiny shader, except instead of setting a globally induced reflection value, I made it so you can use a texture to control the reflection value over the object. That way, you can have different reflection values in different places.

Simple SSS

Prior to the monumental undertaking that is the True SSS shader, Don tried a simple solution to fake subsurface scattering.

Specular Pass

This was a shader that was more of a proof of concept. Someone on the forum asked how we could derive passes from Sunflow even though Sunflow doesn't yet have that ability. I suggested you could use a janino shader to pull out the pass you want and gave this example which is the phong shader with the spec color of white and no diffuse shading.

Translucent Shader

A cool translucent shader from Mark Thorpe based off Don's True SSS shader.

Wire Shader

This shader doesn't work in 0.07.2 since there have been changes to the source code since it was created. I've included it here for reference. Plus, there is a command line wire frame shader that makes this shader superfluous.

Source Code Shaders

Shaders so awesome, they needed modifications to the source code to make them happen.

True SSS

This subsurface scattering shader is currently under development, but it's still amazing.

Textured SSS

Don's work in developing a procedural marble texture and volume shader that compliments his True SSS shader.

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shaders_sca"

- This page was last modified on 21 March 2008, at 16:26.

Cameras sca

From Sunflow Wiki

Contents

- 1 Cameras
 - 1.1 Pinhole
 - 1.2 Thinlens
 - 1.2.1 Depth of Field
 - 1.2.2 Bokeh
 - 1.3 Spherical
 - 1.4 Fisheye
- 2 Camera Motion Blur
- 3 Camera Shutter

Cameras

Sunflow has four camera types: Pinhole, thinlens, spherical, and fisheye. I'll go into the syntax used for each and a few points that need to be made.

Cameras use matrix transforms such as

```
param transform matrix col myMatrixReadByColumn
param transform matrix row myMatrixReadByRow
```

Cameras can also have motion blur added.

Pinhole

Probably what people would consider the "standard" perspective camera.

```
camera myCamera {
  type pinhole
  param transform matrix row 0.0 0.416 -0.909 -18.19 0.0 0.909 0.416 8.97 1.0 0.0 0.0 -0.93 0.0 0.
  param fov float 30.0
  param aspect float 1.77
  param shift.x float 0.0
  param shift.y float 0.0
}

options ::options {
  param camera string myCamera
}
```


You can also camera shift (which are optional lines), which basically takes the perspective shot you have and shifts the view in the x or y without ruining the perspective. I would start with small values for shift.

Thinlens

Depth of Field

The thinlens camera is our depth of field (dof) camera, which is also capable of doing bokeh effects. For a thinlens without bokeh you use:

```
camera myCamera {
  type thinlens
  param transform matrix row 0.707 -0.408 0.577 7.0 0.707 0.408 -0.577 -6.0 0.0 0.816 0.577 5.0 0.
  param fov float 49.134
  param aspect float 1.333
  param focus.distance float 30.0
  param lens.radius float 1.0
}

options ::options {
  param camera string myCamera
}
```

Depth of field is also one of the three things (the others being motion blur and object motion blur) that are directly affected by samples in the image block.

Bokeh

If you activate dof it's better to "lock" the AA (e.g. an AA of 2/2) avoiding adaptive sampling. If you want to use bokeh, you would add two attributes to the end (the number of sides and the rotation of the effect):

```
camera myCamera {
  type thinlens
  param transform matrix row 0.707 -0.408 0.577 7.0 0.707 0.408 -0.577 -6.0 0.0 0.816 0.577 5.0 0.
  param fov float 49.134
  param aspect float 1.333
  param focus.distance float 30.0
  param lens.radius float 1.0
  param lens.sides int 6
  param lens.rotation float 36.0
}

options ::options {
  param camera string myCamera
}
```

As with the pinhole camera, shifting has been added as an option. It looks like this:

```
camera myCamera {  
  type thinlens  
  param transform matrix row 0.707 -0.408 0.577 7.0 0.707 0.408 -0.577 -6.0 0.0 0.816 0.577 5.0 0.  
  param fov float 49.134  
  param aspect float 1.333  
  param shift.x float 0.0  
  param shift.y float 0.0  
  param focus.distance float 30.0  
  param lens.radius float 1.0  
  param lens.sides int 6  
  param lens.rotation float 36.0  
}  
  
options ::options {  
  param camera string myCamera  
}
```

Spherical

The spherical camera produces a longitude/latitude environment map.

```
camera myCamera {  
  type spherical  
  param transform matrix row 0.707 -0.408 0.577 7.0 0.707 0.408 -0.577 -6.0 0.0 0.816 0.577 5.0 0.  
}  
  
options ::options {  
  param camera string myCamera  
}
```

Fisheye

A classic lens.

```
camera myCamera {  
  type fisheye  
  param transform matrix row 0.707 -0.408 0.577 7.0 0.707 0.408 -0.577 -6.0 0.0 0.816 0.577 5.0 0.  
}  
  
options ::options {  
  param camera string myCamera  
}
```

Camera Motion Blur

Camera motion blur is available in the current release. Object motion blur has also been added.

Camera motion blur is also one of the three things (the others being dof and object motion blur) that are directly affected by samples in the image block, so if it's not there, you'll want to add it or the default of 1 is used. So if you're not getting a smooth results you might want to try increasing the samples or reducing the step increases (as described below).

Camera motion blur consists of different steps of the transform (in either row or column major format) and those steps differing is aspects of the transform. For example, here is a camera with a 3 step motion blur with the x up vector changing:

```
camera myCamera {
  type pinhole
  param shutter.open float 0.0
  param shutter.close float 1.0
  param transform.steps int 3
  param transform.times float[] none 2 0.0 1.0
  param transform[0] matrix row 0.581 -0.343 0.738 1.3 0.814 0.245 -0.527 -0.9 0.0 0.907 0.422 1.1
  param transform[1] matrix row 0.603 -0.304 0.738 1.3 0.796 0.299 -0.527 -0.9 -0.060 0.905 0.422
  param transform[2] matrix row 0.623 -0.260 0.738 1.3 0.772 0.355 -0.527 -0.9 -0.125 0.898 0.422
  param fov float 49.134
  param aspect float 1.33
}

options ::options {
  param camera string myCamera
}
```

You may use this with the other camera lens types as well.

Camera Shutter

Sunflow has added functionality for the pinhole and thinlens cameras, allowing you to change the shutter time (the times over which the moving camera is defined (uniform spacing is assumed)) which used to be clamped to [0,1]. For object motion blur to work the camera in the scene needs the shutter line added:

```
camera myCamera {
  type pinhole
  param shutter.open float 0.0
  param shutter.close float 1.0
  param transform matrix row 0.0 0.416 -0.909 -18.19 0.0 0.909 0.416 8.97 1.0 0.0 0.0 -0.93 0.0 0.
  param fov float 30.0
  param aspect float 1.77
  param shift.x float 0.0
  param shift.y float 0.0
}

options ::options {
  param camera string myCamera
}
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Cameras_sca"

- This page was last modified on 23 December 2007, at 02:24.

Lights sca

From Sunflow Wiki

Contents

- 1 Lights
 - 1.1 Samples
- 2 Attenuated Lights
 - 2.1 Point Light
 - 2.2 Meshlight/Area Light
- 3 Non-Attenuated Lights
 - 3.1 Spherical Lights
 - 3.2 Directional Lights
- 4 Inifinitely Far Away Lights
 - 4.1 Image Based Light
 - 4.2 Sunsky
- 5 Showboating Lights
 - 5.1 Cornell Box

Lights

Keep in mind that for colors the syntax sRGB nonlinear is used in these examples. These values are the color space most of us are used to. If you prefer, you can use other color spaces which I outline in the shader overview.

It's also important to note that at this time IBL and Sunsky do not emit photons, therefore, no caustics are viewable when using these lights. All other lights can emit photons.

Samples

Samples, samples, samples. Samples are key for the lights that use them. If they are too low, your image will look bad. If they are too high, your render will take forever. So it's important to experiment to get the right setting for your scene. I suggest starting with low samples, and working your way up till you reach just the right amount.

Attenuated Lights

The point and area lights are attenuated, meaning that the farther away you go away from the light, the less power/influence on the scene it has. Also remember that you can use the power/radiance in negative numbers to remove light from the scene.

Point Light

```
light myPointLight {
    type point
    param center point 1.0 3.0 6.0
    param power color "sRGB linear" 100.0 100.0 100.0
}
```

For the point light, power is measured in watts.

Meshlight/Area Light

```
light meshLamp {
    type triangle_mesh
    param radiance color "sRGB linear" 100.0 100.0 100.0
    param samples int 16
    param points point[] vertex 4
        0.6 0.1 6.0
        0.3 1.0 6.0
        1.0 1.0 5.5
        1.0 0.3 5.5
    param triangles int[] 6
        0 1 2 0 2 3
}
```

Any mesh can become a light. For mesh lights you specify the radiance (which is watts/sr/m²), and the total power is radiance*area*pi (in watts if your area was m²). The example above is a simple square. Area lights/mesh lights automatically create soft shadows and you control the quality of shadows by changing the samples value. An important thing to keep in mind is that the light samples are per face (per triangle), so the more complicated the mesh the more it will take to render. For this reason a simple two triangle quad is probably the way to go. See this thread (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=128>) for more information.

Non-Attenuated Lights

Spherical Lights

```
light myspherelight {
    type sphere
    param radiance color "sRGB linear" 100.0 100.0 100.0
    param center point 5.0 -1.5 6.0
    param radius float 30.0
    param samples int 16
}
```

Directional Lights

```
light mydirectionallight {
    type directional
    param source point 4.0 1.0 6.0
    param dir vector -0.5 -0.2 -1.0
    param radius float 23.0
    param radiance color "sRGB linear" 1.0 1.0 1.0
}
```

Intensity is controlled by the radiance value. So for a bigger intensity you could use something like:

```
param radiance color "sRGB linear" 100.0 100.0 100.0
```

Inifinitely Far Away Lights

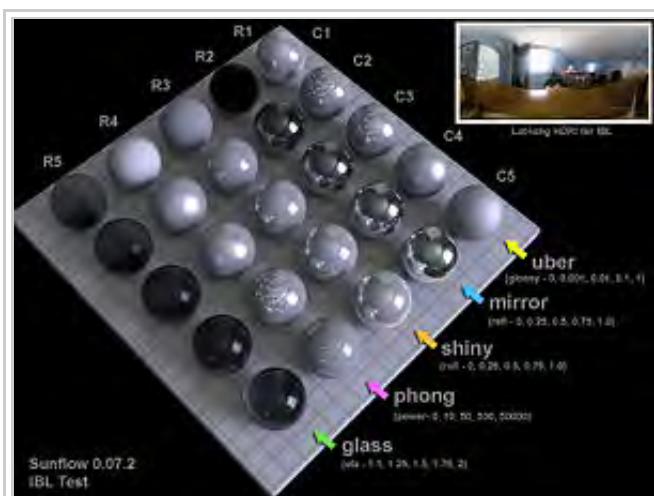
For IBL/Sunsky power, the exact power is measured from the content of the map or the sun position. For IBL it will also depend on a correctly calibrated image.

Image Based Light

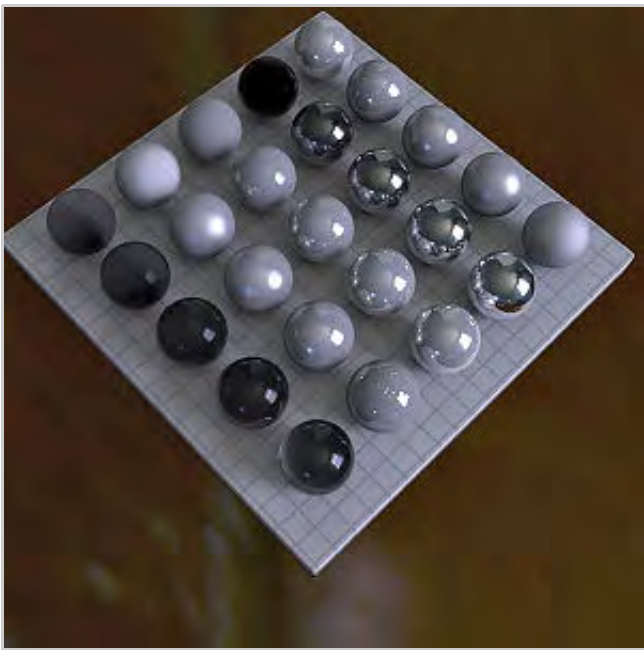
```
light myibl {
    type ibl
    param texture string C:\mypath\myimage.hdr
    param center vector 0.0 -1.0 0.0
    param up vector 0.0 0.0 1.0
    param fixed bool true
    param samples int 200
    param lowsamples int 200
}
```

You can use both low and high dynamic range images whose samples are controlled separately. The "center" vector is a world space direction that points towards the center pixel in the map, letting you rotate the map however you want. The "up" orients the map relative to that axis. This is to be able to support multiple 3d applications: some like to have Y pointing up, others Z.

An important feature of this light is that we can turn importance sampling on or off. Using fixed bool false (importance sampling) forces use of unique samples per shading point instead of fixing them over the whole image. Which basically means that when lock false is set, the points of the image that will affect the light result the most (i.e. more "important") will be sampled more.



Importance sampling turned off (lock true) for a variety of shaders with various settings.



Importance sampling turned on (lock false).

You can find the files that were used to create the above image here (.zip) (<http://www.geneome.net/other/otherfiles/IBLTest.zip>) .

So a conclusion one could draw from these tests is that when using the phong and uber shaders with high power and glossy values respectively, using importance sampling can reduce the light points from the ibl. Increasing samples does not rid the phong and uber shaders of, what Kirk referred to as, “constellations.”

You don't have explicit control over the handedness of the rotation. If it looks like your image is coming in flipped, just do that in your favorite hdr image editor.

Sunflow only supports lon/lat image maps at the moment. So if you have your images in another format (spherical probe or cube-map) you'll need to do some kind of conversion in another program (like HDRShop). You set the samples of different types of images, for high dynamic range images, you would use "param samples int" to set the samples, whereas you would use "param lowsamples int" for low dynamic range images (e.g. png files).

Sunsky

```
light mysunsky {
    type sunsky
    param up vector 0.0 0.0 1.0
    param east vector 0.0 1.0 0.0
    param sundir vector 0.5 0.2 0.8
    param turbidity float 6.0
    param samples int 16
}
```

There isn't a setting in the syntax that controls sun intensity, but you can instead control the suns direction in terms of angle to the object. So if the Sunsky direction is at a near 0 degree angle with the object (the sun on the horizon) it will be dark and the sky will be more a sunset color. If the direction is more high in the sky at around 80 degrees it

will be bright with the sky being white/blue. Changing the up and east values can also change the look, but these are more used to change how the Sunsky is interpreted in different world spaces which might be required in different applications. The up and east values in the above example usually work for everyone.

The Sunsky light has a set horizon where the sky stops and the blackness of the world shows up. Normally an infinite plane is the work-around. Future versions of Sunflow might have a control to extend the sky, but you can also modify the source and compile Sunflow yourself so the sky extends on its own. In `src.org.sunflow.core.light.SunSkyLight.java` go to the line that says

```
groundExtendSky = false;
```

Change it to

```
groundExtendSky = true;
```

Compile Sunflow and the sky will no longer terminate at the horizon.

Showboating Lights

The Cornell Box isn't really a light that you would use in a typical scene but it is useful to illuminate your models.

Cornell Box

```
light nameofcornellbox {
    type cornell_box
    param corner0 point -60.0 -60.0 0.0
    param corner1 point 60.0 60.0 100.0
    param leftColor color "sRGB linear" 0.8 0.25 0.25
    param rightColor color "sRGB linear" 0.25 0.25 0.8
    param topColor color "sRGB linear" 0.7 0.7 0.7
    param bottomColor color "sRGB linear" 0.7 0.7 0.7
    param backColor color "sRGB linear" 0.7 0.7 0.7
    param radiance color "sRGB linear" 15.0 15.0 15.0
    param samples int 32
}
```

Let's do a quick run through. The size of the box is defined by corner0 and corner1. Corner0's x y z values corresponds to position of the lower left corner of the box closest to us. Corner1 is the back top right corner. In the example above, the center of the world (0,0) is in the center of the floor. The color of the sides of the box are then defined. The emit and samples values are just like the meshlight (above).

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Lights_sca"

- This page was last modified on 21 February 2008, at 03:25.

GI sca

From Sunflow Wiki

Contents

- 1 Global Illumination
 - 1.1 Path Tracing
 - 1.2 Ambient Occlusion
 - 1.3 Fake Ambient Term
- 2 Overriding

Global Illumination

Sunflow supports 3 main global illumination (gi) types: Path Tracing, Ambient Occlusion, and Fake Ambient Term. Remember that only one gi type at a time can be used in the scene.

Path Tracing

Probably the most recognized and classic way of doing true global illumination. Sunflow's implementation only handles diffuse inter-reflection. There's not much to say about it except that is probably the most accurate, but slowest, gi method. It usually gives out a noisy image unless your samples are really high. Its implementation is very straight forward:

```
options ::options {
  param gi.engine string path
  param gi.path.samples int 32
}
```

You can also manipulate the number of bounces using the diffuse trace depths by adding:

```
options ::options {
  param depths.diffuse int 1
  param depths.reflection int 4
  param depths.refraction int 4
  param depths.transparency int 10
}
```

So a param depths.diffuse int of 1 means 1 bounce, a param depths.diffuse int of 2 means two bounces, and so on. The more bounces you have the longer your render will take. The other trace-depths are used for glass or thin glass shaders.

Ambient Occlusion

What more can I say than it's ambient occlusion. The settings are pretty straight forward:

```
options ::options {  
    param gi.engine string ambocc  
    param gi.ambocc.bright color "sRGB linear" 1.0 1.0 1.0  
    param gi.ambocc.dark color "sRGB linear" 0.0 0.0 0.0  
    param gi.ambocc.samples int 32  
    param gi.ambocc.maxdist float 3.0  
}
```

Fake Ambient Term

Get some quick ambient light in your scene using this one. You can find the reference for this here (<http://www.cs.utah.edu/~shirley/papers/rtrt/node7.html>) .

```
options ::options {  
    param gi.engine string fake  
    param gi.fake.up vector 0.0 1.0 0.0  
    param gi.fake.sky color "sRGB linear" 0.0 0.0 0.0  
    param gi.fake.ground color "sRGB linear" 1.0 1.0 1.0  
}
```

Overriding

Overriding, or seeing the contribution of the global illumination is done via the command line. Specifically the -view_gi flag.

Retrieved from "http://sfwiki.geneome.net/index.php5?title=GI_sca"

- This page was last modified on 20 March 2008, at 23:57.

Mesh sca

From Sunflow Wiki

Contents

- 1 Triangle Meshes
 - 1.1 Syntax
 - 1.2 Transforms
 - 1.3 Face Shaders/Modifiers
 - 1.4 Object Motion Blur
- 2 Bezier Patches
 - 2.1 Bezier Format Example
- 3 Quad Meshes
- 4 Subdivision Surfaces

Triangle Meshes

Syntax

I'll list the general format below, then give examples of each variation. It's important to understand that in these cases, a point is equal to a vertex.

```

geometry objectName {
    type triangle_mesh
    param points point[] vertex X
        x y z
    ...
    param triangles int[] X
        A B C
    ...
    param normals vector[] none/vertex/facevarying
    param uvs texcoord[] none/vertex/facevarying
}

instance objectName.instance {
    param geometry string objectName
    param shaders string[] 1
        myShader
}

```

For triangles, it is important to note that the triangle indices are listed using the point numbers starting with 0 as the first point. So if you have three points/vertices defined, the triangle values would look like 0 1 2. Given this way of indexing the triangles based on the number of points, there cannot be a triangle index number above ((the number of points) – 1). So in the below examples, you can't have a triangle with an index of 3.

For normals and uvs, you have the option of using none, vertex, or facevarying coordinates. For none, no normals/uvs will be used. The normals and uvs don't have to use the same type to work, so you can have normals vertex and uvs facevarying or normals facevarying and uvs none. A key point is that if you are using the vertex type you need the same number of values that there are points. For facevarying, you need the same number of values equal to the number of triangles (see below for examples).

For the vertex type, each point in the mesh will need its own normal or uv coordinate:

```
...
param points point[] vertex 3
    x1 y1 z1
    x2 y2 z2
    x3 y3 z3
param triangles int[] 3
    0 1 2
param normals vector[] vertex 3
    d1 e1 f1
    d2 e2 f2
    d3 e3 f3
param uvs texcoord[] vertex 3
    u1 v1
    u2 v2
    u3 v3
}
```

For facevarying you would use this:

```
...
param points point[] vertex 3
    x1 y1 z1
    x2 y2 z2
    x3 y3 z3
param triangles int[] 3
    0 1 2
param normals vector[] facevarying 3
    d1 e1 f1
    d2 e2 f2
    d3 e3 f3
param uvs texcoord[] facevarying 3
    u1 v1
    u2 v2
    u3 v3
}
```

Note that I don't define the number of vertex normals or uvs like I do for the points and triangles since point and triangle numbers will determine how many normal and uv coordinates I will need.

Transforms

What about if you want to transform a vertex based object without having to re-export the object. No problem, just transform the instance of the geometry:

```
instance objectName.instance {
    param geometry string objectName
    param shaders string[] 1
}
```

```

    myShader
    param transform matrix row 0.091 0.042 0.0 1.5 0.0 0.0 0.1 0.0 0.042 -0.091 0.0 1.5 0.0 0.0 0.0
}

```

Keep in mind that the transforms are relative to the original position of the geometry.

Face Shaders/Modifiers

If you want to assign multiple shaders or modifiers to different faces on the same mesh, you can do that like so:

```

<pre>geometry objectName {
    type triangle_mesh
    param points point[] vertex X
        x y z
    ...
    param triangles int[] X
        A1 B1 C1
        A2 B2 C2
    ...
    param normals vector[] none/vertex/facevarying
    param uvs texcoord[] none/vertex/facevarying
    param faceshaders int[] 2
        0
        1
}

instance objectName.instance {
    param geometry string objectName
    param shaders string[] 2
        myShader0
        myShader1
    param modifiers string[] 2
        myModifier0
        none
}

```

In the face shader section you are assigning shader 0 (the first shader in the shader list - myShader0) to the first triangle in the triangle list, shader 1 to the second triangle list, etc. It's the same with modifiers, but remember that for modifiers with textures (bump/normal map) you need uvs assigned. Also, if you don't have a modifier for a face, just use "None" in the list.

Object Motion Blur

Sunflow allows you to change the shutter time of the camera which by default clamped to [0,1]. For object motion blur to work the camera (pinhole or thinlens) in the scene needs the shutter lines added:

```

camera myCamera {
    type pinhole
    param shutter.open float 0.0
    param shutter.close float 1.0
    param transform matrix row 0.0 0.416 -0.909 -18.19 0.0 0.909 0.416 8.97 1.0 0.0 0.0 -0.93 0.0 0.
    param fov float 30.0
    param aspect float 1.77
}

```

```
options ::options {
    param camera string myCamera
}
```

Once that's enabled, you would blur the object with the following syntax (similar to camera motion blur), but with the added "times" line, which is the time over which the motion is defined (uniform spacing is assumed). In the below example, I am blurring the object to simulate it traveling some distance (translation).

```
geometry myTeapot {
    type teapot
}

instance myTeapot.instance {
    param geometry string myTeapot
    param shaders string[] 1
        someShader
    param modifiers string[] 1
        bumpMap
    param transform.steps int 3
    param transform.times float[] none 2 0.0 1.0
    param transform[0] matrix row -0.006 0.016 0.0 1.5 0.0 0.0 0.018 0.0 0.016 0.008 0.0 -1.0 0.0 0.
    param transform[1] matrix row -0.008 0.016 0.0 1.5 0.0 0.0 0.018 0.0 0.016 0.008 0.0 -1.5 0.0 0.
    param transform[2] matrix row -0.008 0.016 0.0 1.5 0.0 0.0 0.018 0.0 0.016 0.008 0.0 -2.0 0.0 0.
}
```

Motion blur is also one of the three things (the others being dof and camera motion blur) that are directly affected by samples in the image block, so if it's not there, you'll want to add it or the default of 1 is used.

Bezier Patches

```
geometry myObject {
    type bezier_mesh
    param nu int X
    param nv int X
    param unwrap bool false
    param vwrap bool false
    param points point[] vertex X
        x y z
        ...
}

instance myObject.instance {
    param geometry string myObject
    param shaders string[] 1
        myShader
    param transform matrix row 0.091 0.042 0.0 1.5 0.0 0.0 0.1 0.0 0.042 -0.091 0.0 1.5 0.0 0.0 0.0
}
```

Where X is the number of cv's in u and v, wrap is equivalent to renderman's unwrap/vwrap option (optional and defaults to false), and points are the cv data and should be in the same order as in the renderman spec and there should be exactly $3*(u*v)$ values.

Bezier Format Example

```

geometry myObject {
    type bezier_mesh
    param nu int 4
    param nv int 7
    param unwrap bool false
    param vwrap bool false
    param points point[] vertex 28
        3.2 0.0 2.25
        3.2 -0.15 2.25
        2.8 -0.15 2.25
        2.8 0.0 2.25
        3.45 0.0 2.3625
        3.45 -0.15 2.3625
        2.9 -0.25 2.325
        2.9 0.0 2.325
        3.525 0.0 2.34375
        3.525 -0.25 2.34375
        2.8 -0.25 2.325
        2.8 0.0 2.325
        3.3 0.0 2.25
        3.3 -0.25 2.25
        2.7 -0.25 2.25
        2.7 0.0 2.25
        2.4 0.0 1.875
        2.4 -0.25 1.875
        2.3 -0.25 1.95
        2.3 0.0 1.95
        3.1 0.0 0.675
        3.1 -0.66 0.675
        2.6 -0.66 1.275
        2.6 0.0 1.275
        1.7 0.0 0.45
        1.7 -0.66 0.45
        1.7 -0.66 1.275
        1.7 0.0 1.275
    }
}

instance myObject.instance {
    param geometry string myObject
    param shaders string[] 1
        myShader
    param transform matrix row 0.091 0.042 0.0 1.5 0.0 0.0 0.1 0.0 0.042 -0.091 0.0 1.5 0.0 0.0 0.0
}

```

Quad Meshes

@@@

Subdivision Surfaces

Implements Catmull-Clark type subdivisions from raw mesh data. Currently there is a constant level via the number of iterations.

Syntax needed.

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Mesh_sca"

- This page was last modified on 5 January 2008, at 20:21.

FileMesh sca

From Sunflow Wiki

Contents

- 1 File-Meshes
- 2 Transforms
- 3 Object Motion Blur

File-Meshes

You can import an obj, stl, or ra3 file automatically into Sunflow without having to convert it to Sunflow's file format:

```
geometry objectName {
    type file_mesh
    param filename string test.obj
    param smooth_normals bool true
}

instance objectName.instance {
    param geometry string objectName
    param shaders string[] 1
    myShader
}
```

The filename can also be an absolute path.

The smooth_normals line is optional, it defaults to false (which is best if you have millions of triangles. Note that the stl format does not store mesh connectivity so it isn't possible to smooth their normals.

Sunflow doesn't accept an obj file if there are texture coordinates and normals stored in it. For example, a line like f 1/1/1 2/2/2/ 3/3/3 in the obj file doesn't work. Certain exporters will allow you to not have this information exported to the file. This limitation can be overcome using NeuroWorld's obj to .sc converter (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=356>) .

Transforms

It's important to note that objects (including file meshes) can also be transformed. The transform line goes under the shader declaration (or geometry declaration if it is an instance) in the instance syntax:

```
geometry objectName {
    type file_mesh
```

```
param filename string test.obj
param smooth_normals bool true
}

instance objectName.instance {
  param geometry string objectName
  param shaders string[] 1
    myShader
  param transform matrix row 0.707 0.0 0.707 0.0 -0.579 0.574 0.579 0.0 -0.406 -0.819 0.406 0.0 0.0
}
```

Object Motion Blur

As with generic meshes, file meshes can be object blurred.

Retrieved from "http://sfwiki.geneome.net/index.php5?title=FileMesh_sca"

- This page was last modified on 22 December 2007, at 05:11.

Primitives sca

From Sunflow Wiki

Contents

- 1 Transforms
- 2 Object Motion Blur
- 3 Gumbo
- 4 Teapot
- 5 Background
- 6 Plane
 - 6.1 Infinite Plane I
 - 6.2 Infinite Plane II
- 7 Sphere
- 8 Hair
 - 8.1 Fixed Widths
 - 8.2 Varying Widths
- 9 Julia
- 10 Particle
 - 10.1 Other Method
- 11 Banchoff Surface
- 12 Torus
- 13 Cube Grid
- 14 Box
- 15 Cylinder
- 16 Sphere Flake
- 17 Implicit Surface
- 18 Meta Balls
- 19 Janino Primitive

Transforms

As with other objects you can use transforms just as described in the file-mesh transforms section. For transforming, you can use or not use a lot of the transform options: `translate` (as in `translate x y z`), `rotatex`, `rotatey`, `rotatex`, `scalex`, `scaley`, `scalez`, `scalex`. You could also use a transform matrix by row (row) or column (col).

Object Motion Blur

As with generic meshes, in the 0.07.3 SVN version of Sunflow primitives can be object blurred.

Gumbo

```
geometry myGumbo {  
    type gumbo  
    param subdivs int 6  
    param smooth bool false  
}  
  
instance myGumbo.instance {  
    param geometry myGumbo  
    param shaders string[] 1  
        myShader  
    param transform matrix row 0.091 0.042 0.0 1.5 0.0 0.0 0.1 0.0 0.042 -0.091 0.0 1.5 0.0 0.0 0.0  
}
```

Teapot

```
geometry myTeapot {  
    type teapot  
    param subdivs int 6  
    param smooth bool false  
}  
  
instance myTeapot.instance {  
    param geometry myTeapot  
    param shaders string[] 1  
        myShader  
    param transform matrix row 0.091 0.042 0.0 1.5 0.0 0.0 0.1 0.0 0.042 -0.091 0.0 1.5 0.0 0.0 0.0  
}
```

Background

```
shader background.shader {  
    type constant  
    param color color "sRGB linear" 0.005 0.040 0.133  
}  
  
geometry background {  
    type background  
}  
  
instance backgrond.instance {  
    param geometry string background  
    param shaders string background.shader  
}
```

Plane

You can define an infinite plane in two ways.

Infinite Plane I

```

geometry myPlane {
    type plane
    param center point 0.0 0.0 0.0
    param point1 point 4.0 0.0 3.0
    param point2 point -3.0 0.0 4.0
}

instance myPlane.instance {
    param geometry string myPlane
    param shaders string[] 1
    myShader
}

```

The first p is the center of the plane, and the following two points on the plane. If you use 3 points to define a plane instead of a single point and a normal (as described below) you will get texture coordinates on the infinite plane.

Infinite Plane II

```

geometry myPlane {
    type plane
    param center point 0.0 0.0 0.0
    param normal vector 0.0 1.0 0.0
}

instance myPlane.instance {
    param geometry string myPlane
    param shaders string[] 1
    myShader
}

```

The p is the center of the plane, and the n is the direction of the normal.

Sphere

```

geometry mySphere {
    type sphere
}

instance mySphere.instance {
    param geometry string mySphere
    param transform matrix row 20.0 0.0 0.0 -30.0 0.0 20.0 0.0 30.0 0.0 0.0 20.0 20.0 0.0 0.0 0.0 1.
    param shaders string[] 1
    myShader
}

```

Hair

Fixed Widths

```

geometry myHair {
    type hair
    param segments int 3
    param widths float 0.1
    param points point[] vertex 4
        0.0 0.0 0.0
        0.0 0.5 0.0
        0.0 1.0 0.0
        0.0 1.5 0.0
}

instance myHair.instance {
    param geometry string myHair
    param shaders string[] 1
        myShader
}

```

For example, if you have 3 segments, this means each strand will need 4 vertices. So if you specify 20 points, you should end up with 5 distinct hair strands.

Varying Widths

```

geometry myHair {
    type hair
    param segments int 3
    param widths float[] vertex 4 0.2 0.1 0.05 0.03
    param points point[] vertex 4
        0.0 0.0 0.0
        0.0 0.5 0.0
        0.0 1.0 0.0
        0.0 1.5 0.0
}

instance myHair.instance {
    param geometry string myHair
    param shaders string[] 1
        myShader
}

```

You specify one width per vertex.

Julia

```

geometry myJulia {
    type julia
    param cw float -0.125
    param cx float -0.256
    param cy float 0.847
    param cz float 0.0895
    param iterations int 8
    param epsilon float 0.0010
}

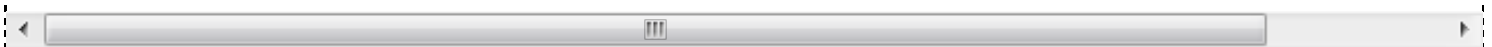
instance myJulia.instance {
    param geometry string myJulia
    param shaders string[] 1
}

```

```

    myShader
    param transform matrix row 0.707 0.0 0.707 0.0 -0.579 0.574 0.579 0.0 -0.406 -0.819 0.406 0.0 0.0
}

```



The line c values are the main julia set parameters and defines its shape. The iterations and epsilon affect the speed and accuracy of the calculation but are not required. If you comment these two lines out you will use the high quality defaults.

Particle

```

geometry myParticles {
    type particles
    param particles point[] vertex 5
        0.5 0.2 0.3
        0.6 0.2 0.3
        0.7 0.2 0.3
        0.8 0.2 0.3
        0.9 0.2 0.3
    param num int 5
    param radius float 0.03
}

instance myParticles.instance {
    param geometry string myParticles
    param shaders string[] 1
        myShader
}

```

Other Method

Here is an workaround when 0.07.2 was being used. Create a .java file alongside your .sca file with the following contents (might not work due to source code changes since 0.07.3:

```

import org.sunflow.core.primitive.ParticleSurface;
import org.sunflow.system.Parser;

public void build() {
    include("your_scene.sc"); // the name of your scene file goes here
    try {
        Parser p = new Parser(resolveIncludeFilename("your_ascii_filename.dat")); // the name of your particle file goes here
        int n = p.getNextInt();
        float[] data = new float[3 * n];
        for (int i = 0; i < data.length; i++)
            data[i] = p.getNextFloat();
        p.close();
        parameter("particles", "point", "vertex", data);
        parameter("num", data.length / 3);
        parameter("radius", 0.1f); // the radius of the particles goes here
        geometry("particle_object_name", new ParticleSurface());
        parameter("shaders", "shader_name"); // replace with the shader name you want to use
        instance("particle_object_name.instance", "particle_object_name");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```


Just render this .java file from the command line instead of the main .sc file. All the paths can be relative, you only need to specify the filename as long as they are all in the same folder... I also assumed that your ascii file contains the number of points in the first line.

Banchoff Surface

```
geometry myBanchoff {  
    type banchoff  
}  
  
instance myBanchoff.instance {  
    param geometry string myBanchoff  
    param shaders string[] 1  
        myShader  
    param transform matrix row 0.707 0.0 0.707 0.0 -0.579 0.574 0.579 0.0 -0.406 -0.819 0.406 0.0 0.0  
}
```

Torus

```
geometry myTorus {  
    type torus  
    param radiusInner float 2.0  
    param radiusOuter float 4.0  
}  
  
instance myTorus.instance {  
    param geometry string myTorus  
    param shaders string[] 1  
        myShader  
    param transform matrix row 0.707 0.0 0.707 0.0 -0.579 0.574 0.579 0.0 -0.406 -0.819 0.406 0.0 0.0  
}
```

Cube Grid

The cube grid primitive is a unit block in the creation of unique shapes. You need to use some clever java to really use it, so take a look at the menger sponge example and the isosurface example.

Box

```
geometry myBox {  
    type box  
}  
  
instance myBox.instance {  
    param geometry string myBox  
    param shaders string[] 1  
        myShader  
}
```

```

    param transform matrix row 0.707 0.0 0.707 0.0 -0.579 0.574 0.579 0.0 -0.406 -0.819 0.406 0.0 0.0
}

```

Cylinder

```

<pre>geometry myCyl {
    type cylinder
}

instance myCyl.instance {
    param geometry string myCyl
    param shaders string[] 1
        myShader
    param transform matrix row 0.707 0.0 0.707 0.0 -0.579 0.574 0.579 0.0 -0.406 -0.819 0.406 0.0 0.0
}

```

Sphere Flake

```

geometry mySF {
    type sphereflake
    param level int 7
    param axis vector 2.0 3.0 4.0
    param radius float 0.5
}

instance mySF.instance {
    param geometry string mySF
    param shaders string[] 1
        myShader
    param transform matrix row 0.707 0.0 0.707 0.0 -0.579 0.574 0.579 0.0 -0.406 -0.819 0.406 0.0 0.0
}

```

The level, axis, and radius lines are optional.

Implicit Surface

@@@

Meta Balls

@@@

Janino Primitive

Like shaders, you can define a primitive via janino.

@@@

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Primitives_sca"

- This page was last modified on 2 January 2008, at 17:29.

Instances sca

From Sunflow Wiki

Contents

- 1 Instances
- 2 Object Motion Blur

Instances

Instancing is when you use the geometry of another object in a scene without having to duplicate the object's mesh data. Instancing an object is useful in reducing the memory overhead, scene file size, and giving control over the objects since you change the main object and those mesh changes will be propagated to all the instances. Instances in Sunflow are great because you can change the location, rotation, scale, shader, and modifier of each instance. What's great about the .sca file format is that you declare geometry and at least one instance of that geometry so it shows up in the scene. So really, adding another instance is quite trivial. You would use the following to instance the geometry:

```
instance nameOfInstance {  
    param geometry string theOriginalObjectName  
    param transform matrix row 1.0 0.0 0.0 -1.0 0.0 0.0 1.0 3.0 0.0 -1.0 0.0 -1.0 0.0 0.0 0.0 1.0  
    param shaders string[] 1  
        shaderForInstance  
    param modifiers string[] 1  
        modifierForInstance  
}
```

Of course, the modifier lines are optional.

Object Motion Blur

As with generic meshes, instances can be object blurred.

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Instances_sca"

- This page was last modified on 23 December 2007, at 02:19.

Modifiers sca

From Sunflow Wiki

Contents

- 1 Modifiers
 - 1.1 Bump Map
 - 1.2 Normal Map
 - 1.3 Perlin Noise

Modifiers

There are 3 object modifiers in Sunflow. A bump, normal, and perlin. At this time, only one modifier type can be applied to an object at a time.

The modifiers are applied to objects by adding a line to the object (or instance) below the shader declaration. Here is an example of a sphere with a modifier:

```
geometry myTeapot {
    type teapot
}

instance myTeapot.instance {
    param geometry string myTeapot
    param shaders string[] 1
        someShader
    param modifiers string[] 1
        bumpMap
    param transform[2] matrix row -0.008 0.016 0.00 1.5 0.0 0.00 0.018 0.0 0.016 0.008 0.00 -2.0 0.0
}
```

The image types that are recognized are tga, png, jpg, bmp, hdr, and igi. For textures, the objects must have UVs mapped. Also note that textures can also be in relative paths. Textures are initialized as a texture object:

```
texture myTex {
    type color_texture_lookup
    param filename string C:\myImage.jpg
}
```

This object can then be called in later shaders.

Bump Map

```
modifier bumpName {  
    type bump_map  
    param color string myTex  
    param scale float 1.3  
}
```

Normal Map

```
modifier normalName {  
    type normal_map  
    param color string myTex  
}
```

Perlin Noise

```
modifier perlinName {  
    type perlin  
    param function int 0  
    param size float 1.0  
    param scale float 0.5  
}
```

The "size" parameter affects as you might guess how spread-out or tight the bumps are. However larger values make a tighter texture (smaller, coarser bumps), and smaller values spread it out more (bigger shallower bumps).

The scale parameter affects the height of the bumps.

A function parameter of 0 results in a generally uniform noise function. A value of 1 gives a striped function along the x axis. Any other value for the function parameter performs a turbulence function.

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Modifiers_sca"

- This page was last modified on 27 December 2007, at 17:13.

Compiling

From Sunflow Wiki

Contents

- 1 Compiling Intro
- 2 Compiling Sunflow for Java 1.6
- 3 Compiling Sunflow for Java 1.5

Compiling Intro

Yes, Sunflow is open source! And you can check out the latest features being added to the SVN by getting the source and compiling it yourself. The method described below is just one way to do it since there is also an ant build file in the source. So if you are using an IDE (for example, Eclipse) you can use that instead. Keep in mind that the Sunflow SVN needs Java 1.6.0 and up to work! For Mac users that have Java 1.5, see the end of this post for how to build Sunflow for Java 1.5.

If you have Java 1.6 and up, you can download my build of the SVN (0.07.3) here (zip) (<http://www.geneome.net/other/otherfiles/SunflowSVN.zip>) .

Compiling Sunflow for Java 1.6

1) First, you'll need the source. The source uses SVN so you'll have to pick the client of choice in your OS to get the sources. As a Windows user I use TortoiseSVN (<http://tortoisesvn.net>) . It's crazy easy since it's a shell program and all you have to do is right click in a folder, select SVN checkout, and use the following location: <https://sunflow.svn.sourceforge.net/svnroot/sunflow>

for command line folks, you would use:

```
svn co https://sunflow.svn.sourceforge.net/svnroot/sunflow sunflow
```

2) Once you have the sources you'll see three folders: branches, tags, and trunks. In tags, you can find older versions or the current version, but it's trunk where you will find the latest additions not found in the final release. In this folder you will need to create a folder named "classes." You don't need to put anything in it.

3) To compile you'll need the JDK installed and have it available in your path. For Windows users you do that by adding the java path (C:\...\bin) to the environment variables under "path." Windows users could also use a .bat, similar the sunflow.bat that indicates the location of java, and the following command to compile it, and unix users can use the shell script.

If your compiling from the path, type javac to make sure it's there (or if the path is not set, just include the explicit path to javac ("c:\program files\java\...\bin\javac") before the following:

Assuming you are in the trunk folder, the command line to compile Sunflow is:

```
javac -classpath janino.jar -sourcepath src -d classes -g:none -O src/*.java
```

If that works, you can run Sunflow from the compiled code using:

```
java -cp .;classes;janino.jar -server -Xmx1024M SunflowGUI
```

Or you can build your own sunflow.jar using the command below:

```
jar cvfm sunflow.jar yourmanifestname.mf -C classes\ .
```

The manifest file (the yourmanifestname.mf) is simply a file you create that states the main class and where it is located. If you have the manifest in the trunk folder this is the text it should include:

```
Manifest-Version: 1.0
Main-Class: SunflowGUI
Class-Path: janino.jar
```

Be sure to end the last line with a return (blank line).

Compiling Sunflow for Java 1.5

It's the same instructions as above, except that you'll need to apply a patch to the source code. You can see the thread discussing this here (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=357>) , but for completeness sake I will add phihag's patch (.diff) (<http://www.geneome.net/drawer/sunflow/J15-ImagePanel.diff>)

If you're looking for already made Java 1.5 SVN builds, check out phihag's site (<http://phihag.de/sunflow/#sunflow-inofficial>) .

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Compiling>"

- This page was last modified on 2 January 2008, at 14:36.

Getting Resolutions Greater Than 16K

From Sunflow Wiki

The code (up to 0.7.3) limits resolution to 16k. If you're looking to create an image with resolution dimensions greater than 16384 - no problem! You can recompile the source with a subtle change to a single file. In `\trunk\src\org\sunflow\core\Scene.java` (starting at line 309 in 0.7.3) you will see this:

```
// limit resolution to 16k
imageWidth = MathUtils.clamp(imageWidth, 1, 1 << 14);
imageHeight = MathUtils.clamp(imageHeight, 1, 1 << 14);
```

A quick bit shift change to the following and a recompile will open up resolutions beyond 16K:

```
// do not limit resolution to 16k
imageWidth = MathUtils.clamp(imageWidth, 1, 1 << 15);
imageHeight = MathUtils.clamp(imageHeight, 1, 1 << 15);
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Getting_Resolutions_Greater_Than_16K"

- This page was last modified on 7 July 2012, at 01:46.

Commandline

From Sunflow Wiki

Contents

- 1 Command Line Flags
 - 1.1 List Of Flags
 - 1.2 Flag Descriptions
 - 1.3 Command Line Shaders
 - 1.4 For SVN 0.07.3 only

Command Line Flags

Sunflow's GUI is great for getting your scene rendered to a png file, but to really unlock Sunflow, there are several command line options available to control the render result. There are several ways to access Sunflow through the command line like editing the sunflow.bat/sunflow.sh file or navigating to the sunflow.jar location to type in the commands. Here is the general look of the command line with a few flags added:

```
%Sunflow Path%\java -Xmx1G -jar sunflow.jar -quick_uvs -nogui -o "C:\outputlocation\output.png" "C:
```

List Of Flags

For completeness sake I will list all the available command line flags but you can use the -help flag to get a list as well:

```
-resolution W H
-aa MIN MAX
-filter X
-nogui
-o "filename"
-bucket X Y
-threads X
-smallmesh
-nogi
-nocaustics
-ipr
-anim
-bake <object_name>.instance, -bakedir ortho/-bakedir view
-sampler type
-lopri
-hipri
-dumpkd
```

```

-buildonly
-showaa
-pathgi X
-bench
-rtbench
-frame X
-X verbosity
-h or -help
-quick_uvs
-quick_normals
-quick_id
-quick_prims
-quick_gray
-quick_wire -aa MIN MAX -filter X
-quick_ambocc X

```

Flag Descriptions

Here are some quick descriptions of the flags:

- -resolution W H : Sets the image resolution, overriding the one set in the scene file.
- -aa MIN MAX : You can set the anti-aliasing of the scene in the command line, overriding the one set in the scene file.
- -filter X : You can set the filter used for the scene, overriding the one set in the scene file. The available filters are box, gaussian, mitchell, triangle, catmull-rom, blackman-harris, sinc, lanczos, and bspline (in the 0.07.3 SVN). See the filter section of the image settings for more info.
- -nogui used to render images without opening the GUI. Required for all images except png files.
- -o "filename" : used to render images out to a file type.
- -bucket X Y : A larger bucket size means more RAM usage and less time rendering. Usually, a bucket size 64 is a good default - especially if you are using a wide pixel filter like gaussian or mitchell. There are six bucket order types available: hilbert (default), spiral, column, row, diagonal, and random. You can also use these ordered in reverse by adding "reverse" in front of the name. To use reverse, you'll need to use quotes around the reverse order (e.g. bucket 48 "reverse spiral") so that the bucket order is still parsed as one token. The number in the line is the pixel size of the each bucket. There is some clamping done internally to make sure the bucket size isn't too small or too big.
- -threads X : For forcing a certain number of threads. Sunflow automatically detects if you have a multi-threaded system so if you do, you should see the number of squares in the Sunflow window that you have threads during rendering. However, there was a report that if you force more threads you *might* see a performance gain - though in theory, you shouldn't.
- -smallmesh : Load triangle meshes using triangles optimized for memory use.
- -nogi : Turns off all global illumination in the scene.
- -nocautics : Turns off any caustics in the scene.
- -ipr : Used to render a scene to the GUI as IPR rather than a final render.
- -anim for animating with Sunflow.
- -bake <object_name>.instance, -bakedir ortho (for diffuse maps), and -bakedir view (for reflections/specular maps). See lightmap baking for more info.
- -sampler type : Render using the specified algorithm.
- -lopri : Set thread priority to low (default).
- -hipri : Set thread priority to high.

- -dumpkd : Dump KDTTree to an obj file for visualization.
- -buildonly : Do not call render method after loading the scene.
- -showaa : Display sampling levels per pixel for bucket renderer.
- -pathgi X : Use path tracing with n samples to render global illumination.
- -bench : Run several built-in scenes for benchmark purposes.
- -rtbench : Run realtime ray-tracing benchmark (cannot be used with -nogui)
- -frame X : Set frame number to the specified value.
- -X verbosity : Set the verbosity level: 0=none,1=errors,2=warnings,3=info,4=detailed.
- -h or -help : Prints all commands.

Command Line Shaders

- -quick_uv : Renders the UVs of objects.
- -quick_normals : Renders the normals of objects.
- -quick_id : Renders using a unique color for each instance.
- -quick_prims : Renders using a unique color for each primitive.
- -quick_gray : Renders the all the objects in the scene gray diffuse, overriding all shaders - great for lighting tests.
- -quick_wire -aa MIN MAX -filter X : Renders objects as wireframe. You set the aa and filter to be used here because without it, the wire doesn't look great.
- -quick_ambocc X : Renders a scene in ambient occlusion mode with a certain distance (X).

Only one command line shader at a time can be used in the command line.

For SVN 0.07.3 only

```
-translate
```

Translate the old file format of the scene file to the new file format in 0.07.3 in either ascii or binary format:

```
...sunflow.bat -translate new_scene.sca my_old_scene.sc
```

Where .sca is the file extension for ascii scenes, and .scb is for binary scenes (not human readable unless you have a hexadecimal editor handy).

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Commandline>"

- This page was last modified on 17 December 2007, at 02:13.

Animation

From Sunflow Wiki

Contents

- 1 Rendering An Animation
 - 1.1 Using 0.07.2
 - 1.2 Using 0.07.3
 - 1.3 Command Line
- 2 Image Sequences

Rendering An Animation

In order to render an animation in Sunflow you'll have to use a command line - not the GUI. The short of it is using an exporter of your choice to output individual scene files for each frame of your animation (yourBaseFileName.001, yourBaseFileName.002, etc.) and a .java file which parses the information.

Using 0.07.2

For the 0.07.2 version of Sunflow, here's an example parsing file that has a non-animated global settings file and the animated frames:

```
public void build() {  
    parse("yourBaseFileName" + ".settings.sc");  
    parse("yourBaseFileName" + "." + getCurrentFrame() + ".sc");  
}
```

Where yourBaseFileName is the name of your scene file without extensions added on.

Using 0.07.3

For the SVN 0.07.3 version of Sunflow, the above file won't work. You'll need to replace the word "parse" with the word "include", and replace "getCurrentFrame()" with "currentFrame()" so it will look like this:

```
public void build() {  
    include("yourBaseFileName" + ".settings.sc");  
    include("yourBaseFileName" + "." + currentFrame() + ".sc");  
}
```

Command Line

So what's the command line to send an animation to Sunflow?

```
-anim 1 10 -o output.#.png scenename.java
```

You can also use paths:

```
-anim 1 10 -o myFolder\output.#.png "C:\My Animations\scenename.java"
```

What this is saying is send frame 1 through 10 and output each frame as a png (or hdr, tga, exr, igr), using the files identified in scenename.java.

For us windows users who are using the sunflow.bat, this is how I run it from a windows command line:

```
c:\sunflow\sunflow.bat -anim 1 10 -o output.#.png scene.java
```

OR

add "-anim 1 10 -o outputfiles\output.#.png scenefiles\scene.java" after the sunflow.jar in the .bat file.

Image Sequences

This, of course, will render out an image sequence. You'll need to then take this sequence and render it out into a movie (if that's what you want to do) using software that can process said images into movie files. You can use Blender's video sequence editor to do this by adding an image and selecting all the images in the file browser. You can then render out the image sequence.

I did a little video (.zip) (<http://www.geneome.net/other/videotutorials/SunflowAnimationUsingBlender-XviD.zip>) on the subject for Blender users. Other users still might find it helpful when I use the command line to render.

Also, if you're using the Blender exporter, you can run an animation command from the script by exporting an animation, and with the animation button pressed, select the image type, then hit the Render button.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Animation>"

- This page was last modified on 17 December 2008, at 19:16.

Lightmap

From Sunflow Wiki

Lightmap Baking

- 1) Make sure the object you want to bake textures for has non-overlapping UV coordinates that fall in the $[0,1]$ square (i.e. all UVs need to be between 0 and 1).
- 2) Export your scene.
- 3) Render it to make sure you have the scene set up properly and the looks the way you want.
- 4) Launch a new render on the command line with the following extra arguments:

```
-bake <object_name>.instance
```

replacing `<object_name>` with the name of the mesh to bake. Then,

```
-bakedir ortho
```

for diffuse maps.

or

```
-bakedir view
```

for reflections/speculars as seen from the main camera which will only look right when viewed through the same camera viewpoint.

So the command line might look something like this for a diffuse map:

```
java -Xmx1G -jar sunflow.jar -nogui -o "C:\outputlocation\output.xxx" "C:\scenelocation\test.sc" -b
```

Where 1G is 1 gig of memory to be used (you can change the amount) and where xxx is the file type you want the map (e.g. hdr, png, exr, tga, or igi (if you use 0.07.3)).

The image that gets rendered will then be mapped back to your objects UVs.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Lightmap>"

- This page was last modified on 23 June 2011, at 18:28.

Output

From Sunflow Wiki

It is important to note that when using the gui you can only save out png files, so you'll need to use the command line to get the other file types (along with png).

From the command line, use the following commands:

```
-nogui -o output.hdr scenefile.sc  
-nogui -o output.exr scenefile.sc  
-nogui -o output.tga scenefile.sc  
-nogui -o output.png scenefile.sc
```

- When rendering really large images the exr output driver only needs to keep the active buckets in memory instead of keeping the entire image in memory (which is good).

In the SVN (Sunflow 0.07.3), Sunflow also has support for the IGI file type (Indigo Image format (.igi) HDR file type), so it would look like:

```
-nogui -o output.igi scenefile.sc
```

For windows users add "-nogui -o output.hdr scenefile.sc" after the sunflow.jar in the .bat file OR
c:\sunflow\sunflow.bat -nogui -o output.hdr folderinsunflow\scenefile.sc

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Output>"

- This page was last modified on 27 June 2011, at 19:51.

Alpha Maps

From Sunflow Wiki

Sunflow 0.07.2 doesn't produce RGBA images, but the 0.07.3 SVN version and up does, provided that there is no background primitive. Future releases will also be able to utilize the alpha in RGBA textures, but that isn't in the SVN yet.

If you're still using 0.07.2 and you're looking to produce an alpha map for your rendered image you can do this to get it:

Have your object(s) alone in a scene and have an ambocc shader (in override mode) in the scene have a distance of 0 and have the dark and light colors reversed (black is light, white is dark). Here is what the ambocc shader looks like in the scene file - note that you don't need any object to have this shader and the number of samples is irrelevant:

```
shader {  
  name amboccshader  
  type amb-occ2  
  bright { "sRGB nonlinear" 0.0 0.0 0.0 }  
  dark { "sRGB nonlinear" 1.0 1.0 1.0 }  
  samples 1  
  dist 0.0  
}
```

If you add under this the following line:

```
override amboccshader true
```

It will cause the whole scene to use the ambocc shader with a distance of 0, allowing you to create an alpha map - which is really an inverted flat ao pass.

Commenting this last line out will cause the scene to render as usual:

```
%override amboccshader true
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Alpha_Maps"

- This page was last modified on 31 December 2007, at 02:19.

Exporters

From Sunflow Wiki

Exporters

If any of the external links go down please contact me through my site (<http://www.geneome.net/index.php/contact/>) and I can link to my copies of them. I would rather not do that now since the sites usually have the howto's and other interesting info. Be aware that some of these script may be old and not work with current versions of the designated application. If you have updates or you know for certain the script doesn't work with certain versions of the application, please tell me and I'll note it here.

- Blender Exporter
- 3DS Max Exporter
- Maya Exporter
- SketchUp Exporter
- Lightwave (<http://www.kevinmacphail.com/6.html>)
- Cinema4D (.zip) (<http://www.geneome.net/other/otherfiles/sfexporters/C4DSunflowExporterSuite.zip>)
- XSI (<http://migaweb.de/downloads.php?id=2>)
- Houdini 8.1 (<http://forums.odforce.net/index.php?showtopic=5527>)
- Houdini 9.1 (bottom of page) - Not working, and looking for help to get it working (<http://forums.odforce.net/index.php?showtopic=5527&st=12>)

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Exporters>"

- This page was last modified on 2 March 2008, at 21:21.

Exporters/Blender

From Sunflow Wiki

< Exporters

This page deals with the script in the SVN. You can find the latest SVN script here (http://sunflow.svn.sourceforge.net/viewvc/sunflow/trunk/exporters/blender/sunflow_export.py?view=log) .

To install the script in Blender:

- 1) Make sure you have Python 2.5 installed (2.3 for Mac) for Blender 2.44.
- 2) Download the script (the download button above the most recent SVN addition). I suggest right clicking on the button and saving the file that way since some browsers open the script up in their own window.
- 3) Save the script to %your Blender folder%/.blend/scripts/.
 - 3a) If you have Blender open when you do this, you can go to the user preferences panel > file paths button and click the gear-looking button next to the Python entry (there doesn't have to be anything in the path) to re-evaluate your script folder.
- 4) In Blender, go to the Scripts panel > Export, and look for the Sunflow Exporter. Click it to initialize it.
[/*list*] To start, I will say this: Sunflow is incredibly powerful in the right hands. The Blender exporter script tries its best to help unlock Sunflow, but there are still several aspects of Sunflow that either Blender can't do or aren't in the script yet (lightmap baking, camera motion blur, etc.). So my suggestion, to really know Sunflow, is study the scene files and get to know what Sunflow can really do.

Okay, on to the show.

The Blender export script is constantly evolving, but I thought I would introduce the nuances of the script since there are no official instructions. The script has two types of values that are exported from Blender, those that are in Blender's interface and those that are exported from values you define in the script itself. Since the latter are obvious (they are in the script), I thought I would mention what values can be Blender derived.

Contents

- 1 HUGE Thing To Know
- 2 HUGE Thing To Know 2
- 3 Blender's Command Window
- 4 Layers
- 5 Scene Properties
- 6 Shader Properties (See the Shader Section of the Script for More Information)
- 7 World Properties
- 8 Light Properties

- 9 Textures
- 10 Camera Properties
- 11 Objects
- 12 Instancing/Dupligroups
- 13 Hair Objects
- 14 Particle Objects - FOR SVN 0.07.3 ONLY
- 15 Sunflow Command Line in the Blender Export Script
- 16 Regarding the "Configure SF" panel
- 17 Saving script settings to the .blend file

HUGE Thing To Know

Every object in your scene needs to have a material set in Blender (any material, not necessarily a Sunflow specific one) or you'll get a Nonetype object error. This is because the script looks for the material's name to define a Sunflow shader, but if the object doesn't have a material set, there isn't a name for the script to look at, so the export fails.

HUGE Thing To Know 2

While working in Blender and at the same time have the Sunflow Blender Exporter Python Script running, it "crashes" the script if you make a "undo (ctrl+z)" in Blender.

Blender's Command Window

It's important to remember that the script will print to the Blender command window. It will tell you if an export has been made successful (or if it's still in the process of exporting), how it's loading the Sunflow render window, etc. Since Blender's command window is usually behind Blender we miss these messages, so if the script seems to have frozen Blender, be sure to check the command window to see what's going on.

Layers

Firstly, The exporter will only export objects on active layers.

Scene Properties

1) Scene > Animation panel: The start and end frame are used to tell the script how many files the "Export As Animation" exports out.

2) Scene > Format panel: The image size (e.g. 800 x 600) is used to define your image size in Sunflow.

3) Scene > Render panel: The image size percentage buttons can be used to reduce the "real" image size, just like in Blender.

Shader Properties (See the Shader Section of the Script for More Information)

4) Shading > Material Buttons > Material panel: The "Col" and "Spe" RGB values are used for a variety of shaders.

5) Shading > Material Buttons > Mirror Transp panel: The RayMir value (when the Ray Mirror is pressed) is used in Sunflow's mirror shader and the IOR value (when the Ray Transp button is pressed) is used in Sunflow's glass shader. The RayMir value is also used for the shiny shader, but you don't need the button pressed. Side note: The SF glass shader has two absorption values that are hard coded in the script, I would like to use the limits value in Blender for one of them, however it isn't a part of the Blender Python API.

6) The script understands material indices. So you can assign different materials to different faces of an object and it will be understood by Sunflow.

World Properties

7) Shading > World Buttons > World panel: Unless you specify your own background in the script, the Horizon RGB values will be used as the background color in Sunflow.

Light Properties

8) Shading > lamp buttons > Preview panel/Lamp panel: Sunflow supports 4 light types that Blender has... sort of. The Point lamp, Sun, Area (aka meshlight), and Spot. Spot is the sort of. If you have spots in your Blender scene they are replaced by directional (cylindrical) lights in Sunflow. You'll need to adjust the distance (in the spot's lamp panel) as close as possible to the ground receiving the cone of light if you want radius as close as possible. Sun is another unique one since Sunflow's sun has turbidity. So in the background panel you'll find a button that says "import sun." With this pressed and a sun in the scene, you'll be able to export out a sun where you can select the samples and turbidity settings. Also you can export a Sunflow spherical lamp from Blender by using a hemi-lamp position as the lamps center, and the distance as the radius.

9) Shading > Lamp buttons > Lamp panel: You can specify the size of area lamps (square or rectangular) , the color of any lamp's RGB values, and any lamp's energy. Though for energy (which is by default 1.0), you can use the script to multiply the value since in Sunflow, the Blender values always show up dark.

10a) Meshlights (thanks to Heavily Tessellated for pointing this out in another thread): If you are looking to make a mesh a light source, begin the name of the object with "meshlight," and it will be interpreted as a meshlight. The meshes material settings will be ignored except for the diffuse color (which will be the light's color). You can control the meshlight power on the Light tab of the exporter. The important thing to understand is that the light samples are per face. So if, in HT's example, you were thinking of making a florescent light tube and you used a 16-sided cylinder for it in Blender, that works out to: 16 quads become 32 side face triangles, +16 end tris, +16 other end tris = 64 individual light sources, times whatever sample value you chose... say 16 (the exporter default) would make it 1024 light samples. That's pretty intense, but in the scheme of things it's not so bad.

10b) If an image texture name is "ibllight" (must be lowercase), then that image will be used as an image based light. It doesn't need to be applied to an object or a textured shader to work, so I would suggest adding the texture to the World textures to indicate that it's on its own. The image based light is infinitely far away, so if you have a

camera in a closed box, then the ibl won't show through and the image will render without a light. The script exports it with general center and up vectors, so you'll need to play with the settings to get the placement exactly the way you want it.

Textures

11) Shading > Texture buttons > Texture panel: This information is in the shader info panel of the script but I'll mention it here. If an image is in the first texture slot, it will be used in the diffuse channel of the shader (if the shader supports textures). If an image is in the second texture slot, it will be used as either a normal or bump map for the shader. You specify which by beginning the name texture with either "normal" or bump." The scale of the effect of the bump or normal map is dictated by having the Nor button of the Map To panel pressed for that texture and using the Nor slider to control the value. If there is a non-image texture in slot 4 and you begin the name with "perlin," the perlin modifier (in the SVN for 0.07.3) will be used. The Nor slider controls the scale, but the function and size are currently hard coded. If an image is in the third texture slot, it will be used in the specular channel of the shader (if the shader supports textures). Remember, image textures need to be UV mapped.

12) Any texture panel: If the image texture name is "ibllight" (must be lowercase), then that image will be used as an image based light.

Camera Properties

13) Editing > Camera panel: The lens value is used to determine the fov in Sunflow and the DOF distance value is used in the exporter as, you guessed it, the DOF distance. COMING SOON: In Sunflow's SVN there is support for camera shifting. If you're using an SVN Sunflow, the script has the shift values taken from Blender's script value for the pinhole camera, but it's commented out. If you want to use it, just un-comment that line.

Objects

14) The key objects Blender that can be exported are NURBS surfaces and mesh objects. NURBS surfaces render out a bit blocky, so I suggest converting to a mesh (ALT+C in Blender), then subsurfing. The exporter will take the Render Level of the subsurf modifier as well as the set smooth result (if used) for mesh objects. UV information (if any) is also taken from objects in Blender and is the key method of mapping with an image.

Instancing/Dupligrpups

The script will understand if an object is dupligrpued to an empty. The script will then create a Sunflow instance of that object in the empty's location using the empty's rotation and scale. To dupligrpup an object: Select your main object, the CTRL+G to add to a new group. You can change the group name in the object panel Once that's done, use Spacebar>Add>Group>GroupName. This will add an instanced object on an empty. This empty's location, rotation, and scale will be used as the instance's location, rotation, and scale in the exported Sunflow scene file. It's important to note that in Blender, the instances will all use the same material that the main object has. In Sunflow you can change the shader of the instances, or do different groups in Blender. The script assumes that the object and mesh names are the same, so if your instances aren't exporting, this might be the reason.

Hair Objects

Sunflow has a hair object type so I got Blender hair particle data to be able to be exported out to Sunflow. Just create a static particle system with "vect" enabled, set the "step" value to 1, export, and there you have it! The hair width is currently hard coded to a set value since the Blender Python API doesn't have access to the strand button. The Sunflow SVN can deal with changing widths over the length of the hair, but I haven't added that ability to the script for the same reason. I've found that hair usually needs a high AA (like 0,3) with a gaussian filter to look nice.

Particle Objects - FOR SVN 0.07.3 ONLY

If you happen to be using an SVN version of Sunflow you can export particle data, or the vertices of objects whose name begins with "particleob" as a Sunflow particle object. Like hair, the radius of the particles is hard coded. You might be thinking that 0.07.2 has particle objects so why is 0.07.3 needed. Answer: Only 0.07.3 allows the point data to be stored in the .sc file, whereas 0.07.2 requires an external binary .dat file.

Here is some more info regarding the script:

Sunflow Command Line in the Blender Export Script

olivS added the ability to use execute Sunflow command line functions through the exporter. All of these functions can be found in the Render Settings button and are enacted by ==first exporting the .sc file==, adding the flags you want (though you don't need to add any flags), then hit the "Render exported" button. It will render your image out to the location of the .sc file (with any flags you might have selected). I recently added on to this the ability to render out an animation sequence (after first being exported as .sc files). Just keep the "Export As Animation" button turned on and then hit the "Render exported" button. It will then render your sequence out in the same location the .sc files are in. You can also use the IPR button pressed to get a quick look at the progressive render. You can hit "esc" to stop the render.

Regarding the "Configure SF" panel

This panel has several settings that are necessary for the command line functions of the script to work (mentioned above). The settings are saved in a config file in your Blender folders .blend/scripts/bpydata/ folder as path2sf.cfg which you edit or delete. Since these settings are for use with the script's command line, you don't need to add/save these settings if all you are going to do is export a .sc file. IMPORTANT: If you want to use the render button (mentioned above in the command line paragraph) in the script to load Sunflow from the script, you MUST set your Sunflow and Java paths. The paths would be the folders in which Sunflow and Java reside (e.g. C:\Sunflow\ and C:\Java\jdk1.6.0_01\bin\). Also note that the two paths can't have white spaces (e.g. "Program files") - that's just the way Blender is when using its command line to run things.

==Samples in the AA panel== (from Chris) The samples value affects DOF and motion blur together. They are in the image block and not in the camera as they are properties of the image sampler, not the camera itself. What it does is super-sample the lens/time areas. It can have an effect on the quality of other elements (like shadows or GI) but only indirectly as a result of those (sub) pixels being sampled more often.

Saving script settings to the .blend file

There are enough options in the exporter that if you wanted to save the same buttons pressed and values changed for a specific scene, you would need to start writing it down somewhere. Enter ID properties. In the config panel of the script you can send the settings you have set to the .blend file so when you save the .blend, these settings will be saved with it. When you open the script again, those values will be auto imported back to the script.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Exporters/Blender>"

- This page was last modified on 10 October 2008, at 00:05.

Exporters/3DSMax

From Sunflow Wiki

< Exporters

The most recent version of the Max to Sunflow exporter is v0.24. It can be downloaded here (http://www.anidesign.de/index_sc.htm) .

In its current state, the script can export the following...

Contents

- 1 General Settings
- 2 Cameras
- 3 Lights
- 4 Geometry
- 5 Textures
- 6 Shaders
 - 6.1 Diffuse
 - 6.2 Constant
 - 6.3 Glass
 - 6.4 Mirror
 - 6.5 Phong
 - 6.6 Ward
 - 6.7 Shiny
 - 6.8 Uber
 - 6.9 Amb-Occ
 - 6.10 IMPORTANT
- 7 Other Stuff
- 8 Installation

General Settings

- Antialias settings
- Sample settings
- Filter settings
- Bucket settings
- GI settings

- Photon settings
- Background colours (if the background in Max is black, then it is ignored)

Cameras

- Both free and target cameras can be exported
- If you enable DOF, you can use the cameras 'Far Range' to focus
- Also with DOF, if the 'Lens Sides' setting is less than 3, a circular lens will be used

Lights

- Point (omni) and directional lights (free / target spotlights / directlights)
- IES_Sun, mr_Sun and Daylight systems
- IBL (image-based lighting) is automatic if an HDR or EXR image is used as an environment map (and is turned on). To get the same view of your HDR in SunFlow as in Max, you will need to set the U Tiling value to -1
- Meshlights: the object MUST have a name beginning with "meshlight", and a standard material with an output map in the diffuse channel. The diffuse color is multiplied by the output amount to get the radiance amount.

Geometry

- Primitives: sphere, torus, teapot, plane (planes with a render-scale of more than 1 get converted to an infinite ground-plane)
- Standard particle systems and PFlow systems
- Hair and Fur modifiers: the width setting is taken from the "Root Thick" setting in the modifier, the amount from the "Hair Count" setting, and the shader just uses the "Root Color" setting to create a diffuse shader
- Everything else gets turned into a generic mesh (including UVs and Material IDs)
- Object instances get exported as SunFlow instances

Textures

- Bitmaps in the Diffuse map channel get exported as part of the shader
- Bitmaps in the Bump map channel get exported as bump modifiers. Uses the 'Bump Amount' value from the maps' Output settings.

Shaders

Diffuse

- All objects with no material assigned get a diffuse shader with the object colour
- All objects with a material type not covered by the other shaders get a diffuse shader with the object colour
- Standard and raytrace materials with shader type OrenNayerBlinn get exported as a diffuse shader using the diffuse colour

Constant

- Any materials with a self-illumination value of more than 0 get exported as a constant shader
- Any materials with a self-illumination / luminosity colour more than 0,0,0 get exported as a constant shader

Glass

- Any materials with an opacity of less than 100 are exported as a glass shader using the diffuse colour and IOR set in Max

Mirror

- Raytrace materials with a reflection colour more than 0,0,0 get exported as a mirror shader using the reflect colour as the mirror colour
- Raytrace materials with a reflection amount of more than 0 get exported as a mirror shader using the reflect amount to set a greyscale for the mirror colour

Phong

- Standard and raytrace materials with shader type Blinn or Phong get exported as a phong shader using the diffuse and specular colours in Max
- The blur strength is calculated using the Soften setting in Max x 1000

Ward

- Standard and raytrace materials with shader type Anisotropic or Multi-Layer get exported as a ward shader using the diffuse and specular colours in Max.
- X and Y roughness is calculated using the glossiness setting (0 = very blurry, 100 = sharp) and the anisotropy setting (0 = X and Y equal, 50 = Y half the width of X, etc.)

Shiny

- Standard and raytrace materials with shader type Metal or Strauss get exported as a shiny shader using the diffuse colour

- Shininess is set using the glossiness amount

Uber

- any Shellac material exports the Base Material as an Uber shader
- diffuse color and texture are set using the Diffuse color and map slot in Max
- specular color and texture are set using the Specular color and map slot in Max
- diffuse and specular blend amounts are set using the Map Amount settings for the Diffuse and Specular maps divided by 100
- samples are set using the Specular Level amount in Max
- glossiness is set using the Soften amount in Max x 100 (I couldn't use the Glossiness setting in Max because it only handles integers)

Amb-Occ

- any Standard material with a Falloff map in the Diffuse slot exports as an amb-occ shader
- the Falloff front colour is used for the SunFlow bright colour
- the Falloff back colour is used for the SunFlow dark colour
- at the moment, the "samples" and "dist" values are fixed at 32 and 10. If I can think of a good way of setting them in Max, I'll change this

IMPORTANT

Shaders always get assigned in the following order:

1. Any standard material with a falloff map in the diffuse slot gets an amb-occ shader
2. Any material with self-illumination gets a constant shader
3. Any material with transparency gets a glass shader
4. Any raytrace material with reflection gets a mirror shader
5. After that the different shader types get assigned according to the material types listed above

Other Stuff

- Support for single frames and animations: uses the Render Scene Dialog to get the image size and still / animation / range settings
- Animations get written to multiple files (test0000.sc, test0001.sc, etc.)

- Particle systems get written to an extra .bin file which has the same filename as the .sc file.
- Particle systems get written to an extra *.part.sc file which has the same filename as the .sc file
- If the 'Separate geometry file' checkbox is checked, all geometry is exported to an extra file with the ending *.geo.sc
- The 'Skip' checkbox (only active if the 'Separate geometry file' is checked) is useful if you've already exported everything once, and just need to tweak materials and render settings. When this is on, the exporter will export everything except the geometry.
- If the 'Start Sunflow after exporting' checkbox is checked, the exported scene is automatically rendered (the first time you do this you need to set the paths to Sunflow and Java by clicking on the 'Set Paths' button)

Installation

The main script (max2sunflow024.mse) goes in the main Scripts folder (usually C:\Program Files\Autodesk\3ds Max 9\Scripts)

The macroscript (max2sunflow.mcr) goes in the macroscripts folder: C:\Program Files\Autodesk\3ds Max 9\ui\macroscripts

The 4 icons (*.bmp) go in the icons folder: C:\Program Files\Autodesk\3ds Max 9\ui\Icons

Now start Max, go to Customize -> Customize User Interface... and add the macroscript to a toolbar or menu of your choice (I always use the 'Extras' toolbar for my bits and pieces)

If you are going to be exporting a lot of particles, or very heavy scenes, it might be an idea to increase the memory allocation for MAXScript. You can do this by going into the 'Customize -> Preferences... -> MAXScript' settings, and increasing the 'Initial heap allocation' setting. I've got mine set to 64MB.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Exporters/3DSMax>"

- This page was last modified on 28 July 2008, at 20:01.

Exporters/Maya

From Sunflow Wiki

< Exporters

The exporter source files are in the SVN, however the compiled version has disappeared, so if you have the Maya API and have a build environment, please help us out and build it for us!.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Exporters/Maya>"

- This page was last modified on 30 December 2007, at 22:48.

Exporters/SketchUp

From Sunflow Wiki

< Exporters

You can find the SketchUp exporter in the SVN

(<http://sunflow.svn.sourceforge.net/viewvc/sunflow/trunk/exporters/sketchup/>) , however the user manual can be found bundled with it on Didier Bur's site (http://www.crai.archi.fr/RubyLibraryDepot/Ruby/em_fil_page.htm) in the su2sf_11.zip file. For those that have the plugin already installed I'll point to the manual here (.pdf) (http://www.geneome.net/other/otherfiles/sfexporters/su2sf_user_guide.pdf) .

Mac Installation

Place the su2sf.rb inside your home folder's Library/Application Support/Google SketchUp 6/PlugIns/. You may need to create the PlugIns directory there if it doesn't exist already.

Launch SketchUp and you'll see there is a PlugIns menu item which contains the Sunflow Exporter.

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Exporters/SketchUp>"

- This page was last modified on 8 March 2009, at 03:02.

Sunflow Projects

From Sunflow Wiki

Introduction

Most of the projects below are meant to be added to a .java file (though some are patches to the source). So for the java file projects all you would need to do is take a plain text file then rename it to anything.java. Using the GUI you can open this file like a scene file or use the command line to render the file out by putting the image type of your choice (e.g. -o output.hdr fractal.java). Some of the files below make use of outside scene files that you must have, but several are "stand-alone" and don't require you to do anything except load the java file.

- Menger Sponge
- Iterative Fractals
- File Mesh Allowing UV And Normal Data
- Isosurface using CubeGrid class
- Quadric Primitive (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=422>)
- Shader Blend Through The Iterations
- Modified Fake Ambient Term
- Filter Size Control
- Source Code Shaders
- IsoCube Primitive Using Version 0.06.3
- Isometric Lens

Note On The SVN Version

You might notice that running these files using the SVN version of Sunflow (0.07.3 and up) may throw errors. This may be due to several changes to the code, but I will list the top two things you should check/correct:

- "parse" has been replaced by "include"
- Calling objects is different in 0.07.3 thanks to the use of the plugin registry. As an example, lets look at how a camera is added to the .java file. The 0.07.2 way is to use the following:

```
camera("myCamera", new PinholeLens());
```

The 0.07.3 way is a bit different.

```
camera("myCamera", "pinhole");
```

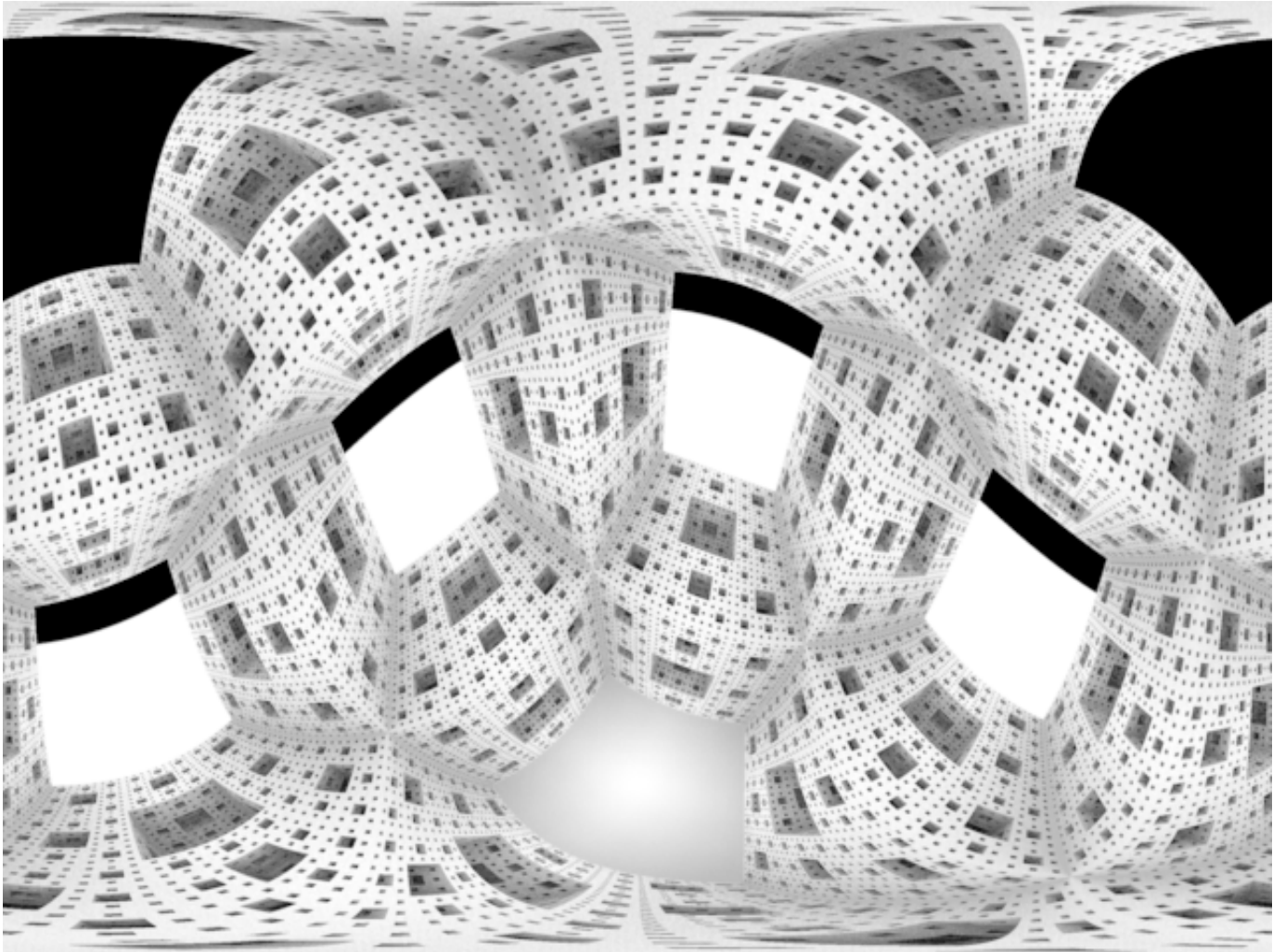
Retrieved from "http://sfwiki.geneome.net/index.php5?title=Sunflow_Projects"

- This page was last modified on 20 April 2008, at 17:31.

Menger Sponge

From Sunflow Wiki

Author: Chris Kulla



```
import org.sunflow.SunflowAPI;
import org.sunflow.core.*;
import org.sunflow.core.camera.*;
import org.sunflow.core.primitive.*;
import org.sunflow.core.shader.*;
import org.sunflow.image.Color;
import org.sunflow.math.*;

// Change settings here
int depth = 5;
boolean preview = false;

public void build() {
    parameter("eye", new Point3(2.0f, 2.0f, -5.0f));
    parameter("target", new Point3(0, 0, 0));
    parameter("up", new Vector3(0.0f, 1.0f, 0.0f));
    parameter("fov", 45.0f);
    parameter("aspect", 2.0f);
```

```

camera("camera_outside", new PinholeLens());

parameter("eye", new Point3(0, 0.2f, 0));
parameter("target", new Point3(-1.0f, -0.5f, 0.5f));
parameter("up", new Vector3(0.0f, 1.0f, 0.0f));
camera("camera_inside", new SphericalLens());

parameter("maxdist", 0.4f);
parameter("samples", 16);
shader("ao_sponge", new AmbientOcclusionShader());

parameter("maxdist", 0.4f);
parameter("samples", 128);
shader("ao_ground", new AmbientOcclusionShader());

geometry("sponge", new MengerSponge(depth));
// Matrix4 m = null;
// m = Matrix4.rotateX((float) Math.PI / 3);
// m = m.multiply(Matrix4.rotateZ((float) Math.PI / 3));
// parameter("transform", m);
parameter("shaders", "ao_sponge");
instance("sponge.instance", "sponge");

parameter("center", new Point3(0, -1.25f, 0.0f));
parameter("normal", new Vector3(0.0f, 1.0f, 0.0f));
geometry("ground", new Plane());
parameter("shaders", "ao_ground");
instance("ground.instance", "ground");

// rendering options
parameter("camera", "camera_inside");
// parameter("camera", "camera_outside");
parameter("resolutionX", 1024);
parameter("resolutionY", 768);
if (preview) {
    parameter("aa.min", 0);
    parameter("aa.max", 1);
    parameter("bucket.order", "spiral");
} else {
    parameter("aa.min", 1);
    parameter("aa.max", 2);
    parameter("bucket.order", "column");
    parameter("filter", "mitchell");
}
options(DEFAULT_OPTIONS);
}

private static class MengerSponge extends CubeGrid {
    private int depth;

    MengerSponge(int depth) {
        this.depth = depth;
    }

    public boolean update(ParameterList pl, SunflowAPI api) {
        int n = 1;
        for (int i = 0; i < depth; i++)
            n *= 3;
        pl.addInteger("resolutionX", n);
        pl.addInteger("resolutionY", n);
        pl.addInteger("resolutionZ", n);
        return super.update(pl, api);
    }

    protected boolean inside(int x, int y, int z) {

```

```
for (int i = 0; i < depth; i++) {  
    if ((x % 3) == 1 ? (y % 3) == 1 || (z % 3) == 1 : (y % 3) == 1 && (z % 3) == 1) return  
    x /= 3;  
    y /= 3;  
    z /= 3;  
}  
return true;  
}
```

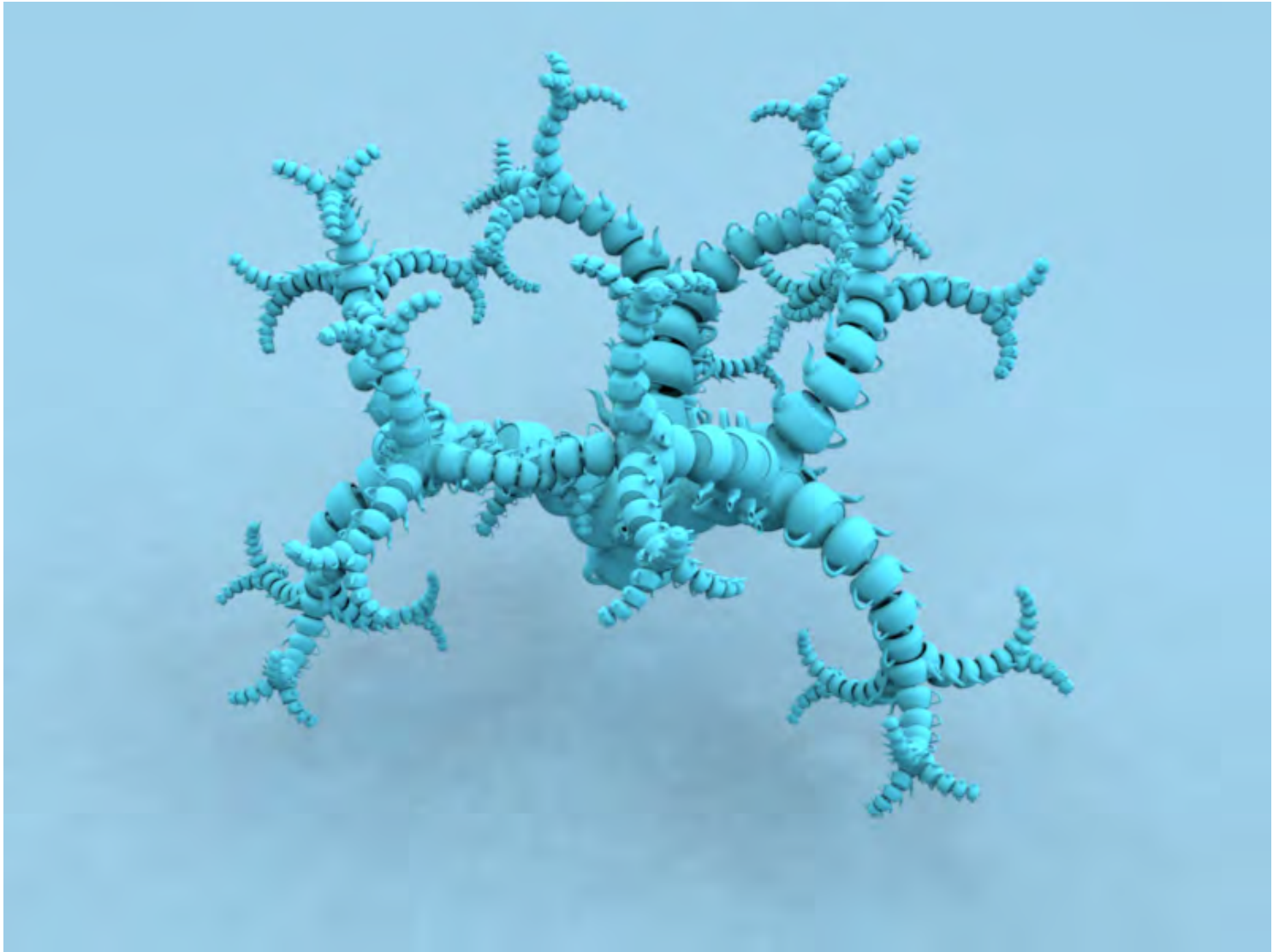
Retrieved from "http://sfwiki.geneome.net/index.php5?title=Menger_Sponge"

- This page was last modified on 12 December 2007, at 19:24.

Iterative Fractals

From Sunflow Wiki

Author: Don Casteel



Instructions: Load in Sunflow as a .java file with a scene called ComboJavaSC.sc with an object in that file named "item" that has a shader called "itemShader".

```
import org.sunflow.core.tesselatable;
import org.sunflow.core.tesselatable.*;
import org.sunflow.SunflowAPI;
import org.sunflow.core.*;
import org.sunflow.core.camera.*;
import org.sunflow.core.primitive.*;
import org.sunflow.core.shader.*;
import org.sunflow.image.Color;
import org.sunflow.math.*;
import org.sunflow.core.LightSource;
import org.sunflow.core.light.*;

// Change settings here
boolean preview = false;
int maxiteration = 22; // total iterations to generate (careful instances are exponential)
int branch = 5; // branch every ? iterations
```

```

//define globals
float geometryScale = 1.3f; //scale the initial object before iterating
int f = 0;
Matrix4 m = null;
Matrix4 m000 = null;
Matrix4 m001 = null;
Matrix4 m002 = null;

public void build()
{
    parse("ComboJavaSC.sc"); // create the scene

    parameter("diffuse", new Color(0.1f,0.75f,1f));
    shader("ds_objects", new DiffuseShader());

    m = Matrix4.scale(geometryScale);

    // this is the first instance at the starting point
    parameter("transform", m);
    parameter("shaders", "itemShader"); // itemShader is defined in "ComboJavaSC.sc"
    instance("item.instance_"+f, "item"); // "item" is defined in "ComboJavaSC.sc"
    f++;

    // this transform is used for every iteration even between branches
    m000 = Matrix4.scale(geometryScale);
    m000 = m000.rotate(1f,0f,0f,0.875f/4f);
    m000 = m000.multiply(Matrix4.translation(0f,0f,0.88f/geometryScale));
    m000 = m000.multiply(Matrix4.scale(0.90f));

    // branching transform
    m001 = Matrix4.scale(1f);
    m001 = m001.multiply(Matrix4.rotate(0f,0f,1f,2.094395102f));
    m001 = m001.multiply(m000);

    // branching transform
    m002 = Matrix4.scale(1f);
    m002 = m002.multiply(Matrix4.rotate(0f,0f,1f,-2.094395102f));
    m002 = m002.multiply(m000);

    Matrix4[] trnsfrms = {m000,m001,m002}; /* group the transforms to be passed to the iteration method */

    iterate(Matrix4.scale(1f),0,trnsfrms,0); // call the iteration method

    parameter("diffuse", new Color(0.2f,0.3f,0.1f));
    shader("ds_ground", new DiffuseShader());
}

public void iterate(Matrix4 matrix, int itn, Matrix4[] xforms, int fh)
{
    printMatrix(matrix,"\n--> in matrix f=" + f + ": ");
    printMatrix(xforms[0],"xforms[0] f=" + f + ": ");
    printMatrix(xforms[1],"xforms[1] f=" + f + ": ");
    printMatrix(xforms[2],"xforms[2] f=" + f + ": ");

    Matrix4 m41 = Matrix4.scale(1f);
    Matrix4 m42 = Matrix4.scale(1f);
    Matrix4 m43 = Matrix4.scale(1f);

    if(f==1)
    {
        f++;
        parameter("transform", m.multiply(m41));
        parameter("shaders", "itemShader");
        instance("item.instance_"+f, "item");
        iterate(m41,itn+1,xforms,fh+1);

        f++;
        m42=m41.multiply(xforms[1]);
        parameter("transform", m.multiply(m42));
        parameter("shaders", "itemShader");
        instance("item.instance_"+f, "item");
        iterate(m42,itn+1,xforms,fh+1);

        f++;
    }
}

```

```

        m43=m41.multiply(xforms[2]);
        parameter("transform", m.multiply(m43));
        parameter("shaders", "itemShader");
        instance("item.instance_"+f, "item");
        iterate(m43,itn+1,xforms,fh+1);
    }

    else

    {
        m41=m41.multiply(matrix);
        m42=m42.multiply(matrix);
        m43=m43.multiply(matrix);

        f++;
        m41=m41.multiply(xforms[0]);
        if(itn<=maxiteration)
        {
            parameter("transform", m.multiply(m41));
            parameter("shaders", "itemShader");
            instance("item.instance_"+f, "item");
            iterate(m41,itn+1,xforms,fh+1);
        };

        if(fh==branch)
        {
            f++;
            m41=m41.multiply(xforms[0]);
            if(itn<=maxiteration)
            {
                parameter("transform", m.multiply(m41));
                parameter("shaders", "itemShader");
                instance("item.instance_"+f, "item");
                iterate(m41,itn+1,xforms,0);
            };

            f++;
            m42=m42.multiply(xforms[1]);
            if(itn<=maxiteration)
            {
                parameter("transform", m.multiply(m42));
                parameter("shaders", "itemShader");
                instance("item.instance_"+f, "item");
                iterate(m42,itn+1,xforms,0);
            };

            f++;
            m43=m43.multiply(xforms[2]);
            if(itn<=maxiteration)
            {
                parameter("transform", m.multiply(m43));
                parameter("shaders", "itemShader");
                instance("item.instance_"+f, "item");
                iterate(m43,itn+1,xforms,0);
            };

        };
    };
};

```

```

public void printMatrix(Matrix4 matrix4, String s)
{
    float[] fa = matrix4.asRowMajor();
    for(int par=0;par<16;par++)
    {
        s = s + fa[par] + " ";
    }
    System.out.println(s);
}

```

```

/*
Work In Progress
//utility class
class TformBranch
{
    Transforms[] tforms;
    int branchFrequency=1;
}

```

```
void TformBranch(Transform[] tf, int bf)
{
    tforms = tf;
    branchFrequency = bf;
}

public void iterate2(TformBranch[] branches)
{
    int numBranches = branches.length()
}

*/
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Iterative_Fractals"

- This page was last modified on 3 December 2007, at 17:35.

File Mesh Tweak

From Sunflow Wiki

Author: MrFoo

"I was getting a bit annoyed with the lack of a file importer that actually accepts uv/normal data, so did a bit of a hack job on FileMesh.java adding some code I wrote for a different application into it. It now supports vertex/texture/uv info.. but now isn't happy if there's just vertex, or just vertex/uv info etc. so from one extreme to the other. However it may still be useful to someone. Oh, and it doesn't import the materials or anything, just the relevant co-ordinates for the faces/vertices loaded. So you can bake a lightmap with it, but not easily load a fully textured object."

The modified FileMesh.java:

```
package org.sunflow.core.tesselatable;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import java.nio.IntBuffer;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.util.ArrayList;

import org.sunflow.SunflowAPI;
import org.sunflow.core.ParameterList;
import org.sunflow.core.PrimitiveList;
import org.sunflow.core.Tesselatable;
import org.sunflow.core.ParameterList.InterpolationType;
import org.sunflow.core.primitive.TriangleMesh;
import org.sunflow.math.BoundingBox;
import org.sunflow.math.Matrix4;
import org.sunflow.math.Point3;
import org.sunflow.math.Vector3;
import org.sunflow.system.Memory;
import org.sunflow.system.UI;
import org.sunflow.system.UI.Module;
import org.sunflow.util.FloatArray;
import org.sunflow.util.IntArray;

public class FileMesh implements Tesselatable {
    public class triinfo
    {
        public int vert;
        public int norm;
        public int uv;
        triinfo(String data)
        {
            vert=0; //.obj is 1-indexed not 0-indexed
        }
    }
}
```



```

        norm=0;
        uv=0;

        String[] inf=data.split("/");
        if(inf.length==1)
        {
            vert=Integer.parseInt(inf[0]);
        }
        if(inf.length==2 && data.indexOf("//")<0) // "vert//normal"
        {
            vert=Integer.parseInt(inf[0]);
            norm=Integer.parseInt(inf[1]);
        }
        else // "vert/uv"
        {
            vert=Integer.parseInt(inf[0]);
            uv=Integer.parseInt(inf[1]);
        }
        if(inf.length==3)
        {
            vert=Integer.parseInt(inf[0]);
            uv=Integer.parseInt(inf[1]);
            norm=Integer.parseInt(inf[2]);
        }
        vert-=1;//"bad" now == -1 and we are 0-index not 1-index
        norm-=1;
        uv-=1;
    }

    }

    private String filename = null;
    private boolean smoothNormals = false;

    public BoundingBox getWorldBounds(Matrix4 o2w) {
        // world bounds can't be computed without reading file
        // return null so the mesh will be loaded right away
        return null;
    }

    public PrimitiveList tessellate() {
        if (filename.endsWith(".ra3")) {
            try {
                UI.println(Module.GEOM, "RA3 - Reading geometry: \"%s\" ...", filename);
                File file = new File(filename);
                FileInputStream stream = new FileInputStream(filename);
                MappedByteBuffer map = stream.getChannel().map(FileChannel.MapMode.READ_ONLY, 0, file.length());
                map.order(ByteOrder.LITTLE_ENDIAN);
                IntBuffer ints = map.asIntBuffer();
                FloatBuffer buffer = map.asFloatBuffer();
                int numVerts = ints.get(0);
                int numTris = ints.get(1);
                UI.println(Module.GEOM, "RA3 - * Reading %d vertices ...", numVerts);
                float[] verts = new float[3 * numVerts];
                for (int i = 0; i < verts.length; i++)
                    verts[i] = buffer.get(2 + i);
                UI.println(Module.GEOM, "RA3 - * Reading %d triangles ...", numTris);
                int[] tris = new int[3 * numTris];
                for (int i = 0; i < tris.length; i++)
                    tris[i] = ints.get(2 + verts.length + i);
                stream.close();
                UI.println(Module.GEOM, "RA3 - * Creating mesh ...");
                return generate(tris, verts, smoothNormals);
            } catch (FileNotFoundException e) {
                e.printStackTrace();
                UI.println(Module.GEOM, "Unable to read mesh file \"%s\" - file not found", filename);
            } catch (IOException e) {

```

```

        e.printStackTrace();
        UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - I/O error occurred", filename);
    }
} else if (filename.endsWith(".obj")) {
    int lineNumber = 1;
    try {
        UI.printInfo(Module.GEOM, "OBJ - Reading geometry: \"%s\" ...", filename);
        FloatArray verts = new FloatArray();
        FloatArray vertsnormal = new FloatArray();
        FloatArray vertsuvs = new FloatArray();
        //IntArray tris = new IntArray();
        ArrayList tris = new ArrayList();
        FileReader file = new FileReader(filename);
        BufferedReader bf = new BufferedReader(file);
        String line;
        while ((line = bf.readLine()) != null) {
            if (line.startsWith("v ")) {
                String[] v = line.split("\\s+");
                verts.add(Float.parseFloat(v[1]));
                verts.add(Float.parseFloat(v[2]));
                verts.add(Float.parseFloat(v[3]));
            }
            else if (line.startsWith("vn")) {
                String[] vn = line.split("\\s+");
                vertsnormal.add(Float.parseFloat(vn[1]));
                vertsnormal.add(Float.parseFloat(vn[2]));
                vertsnormal.add(Float.parseFloat(vn[3]));
            }
            else if (line.startsWith("vt")) {
                String[] vt = line.split("\\s+");
                vertsuvs.add(Float.parseFloat(vt[1]));
                vertsuvs.add(Float.parseFloat(vt[2]));
            }
        }
        else if (line.startsWith("f")) {
            String[] f = line.split("\\s+");
            if (f.length == 5) {
                triinfo v1 = new triinfo(f[1]);
                triinfo v2 = new triinfo(f[2]);
                triinfo v3 = new triinfo(f[3]);
                triinfo v4 = new triinfo(f[4]);
                tris.add(v1);
                tris.add(v2);
                tris.add(v3);
                tris.add(v1);
                tris.add(v3);
                tris.add(v4);
            }
            else if (f.length == 4) {
                triinfo v1 = new triinfo(f[1]);
                triinfo v2 = new triinfo(f[2]);
                triinfo v3 = new triinfo(f[3]);
                tris.add(v1);
                tris.add(v2);
                tris.add(v3);
            }
        }
        if (lineNumber % 100000 == 0)
            UI.printInfo(Module.GEOM, "OBJ - * Parsed %7d lines ...", lineNumber);
        lineNumber++;
    }
}

```

```

        file.close();
        UI.printInfo(Module.GEOM, "OBJ - * Creating mesh ...");
        return generateMine(tris, verts.trim(), vertsuv.trim(), vertsnormal.trim(), smooth);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - file not found", file);
    } catch (NumberFormatException e) {
        e.printStackTrace();
        UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - syntax error at line");
    } catch (IOException e) {
        e.printStackTrace();
        UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - I/O error occurred", file);
    }
} else if (filename.endsWith(".stl")) {
    try {
        UI.printInfo(Module.GEOM, "STL - Reading geometry: \"%s\" ...", filename);
        FileInputStream file = new FileInputStream(filename);
        DataInputStream stream = new DataInputStream(new BufferedInputStream(file));
        file.skip(80);
        int numTris = getLittleEndianInt(stream.readInt());
        UI.printInfo(Module.GEOM, "STL - * Reading %d triangles ...", numTris);
        long filesize = new File(filename).length();
        if (filesize != (84 + 50 * numTris)) {
            UI.printWarning(Module.GEOM, "STL - Size of file mismatch (expecting %s, found %s)", filesize, (84 + 50 * numTris));
            return null;
        }
        int[] tris = new int[3 * numTris];
        float[] verts = new float[9 * numTris];
        for (int i = 0, i3 = 0, index = 0; i < numTris; i++, i3 += 3) {
            // skip normal
            stream.readInt();
            stream.readInt();
            stream.readInt();
            for (int j = 0; j < 3; j++, index += 3) {
                tris[i3 + j] = i3 + j;
                // get xyz
                verts[index + 0] = getLittleEndianFloat(stream.readInt());
                verts[index + 1] = getLittleEndianFloat(stream.readInt());
                verts[index + 2] = getLittleEndianFloat(stream.readInt());
            }
            stream.readShort();
            if ((i + 1) % 100000 == 0)
                UI.printInfo(Module.GEOM, "STL - * Parsed %7d triangles ...", i + 1);
        }
        file.close();
        // create geometry
        UI.printInfo(Module.GEOM, "STL - * Creating mesh ...");
        if (smoothNormals)
            UI.printWarning(Module.GEOM, "STL - format does not support shared vertices - normal smoothing ignored");
        return generate(tris, verts, false);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - file not found", file);
    } catch (IOException e) {
        e.printStackTrace();
        UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - I/O error occurred", file);
    }
} else
    UI.printWarning(Module.GEOM, "Unable to read mesh file \"%s\" - unrecognized format", filename);
return null;
}

private TriangleMesh generateMine(ArrayList tris, float[] verts, float[] vertsuv, float[] vertsnormal, float[] smooth) {
    ParameterList pl = new ParameterList();

```

```

UI.printlnInfo(Module.GEOM, "mine - a");
int numfaces=tris.size()/3;
float[] objverts=new float[numfaces*3*3];
float[] objnorm=new float[numfaces*3*3];
float[] objtex=new float[numfaces*3*2];
int[] trisarr=new int[numfaces*3];
UI.printlnInfo(Module.GEOM, "mine - b");
for(int i=0;i<tris.size();i++)
{
    triinfo ti=(triinfo)tris.get(i);
    trisarr[i]=i;
    objverts[i*3]=verts[ti.vert*3];
    objverts[i*3+1]=verts[ti.vert*3+1];
    objverts[i*3+2]=verts[ti.vert*3+2];
    objnorm[i*3]=vertsnormal[ti.norm*3];
    objnorm[i*3+1]=vertsnormal[ti.norm*3+1];
    objnorm[i*3+2]=vertsnormal[ti.norm*3+2];
    objtex[i*2]=vertsuv[ti.uv*2];
    objtex[i*2+1]=vertsuv[ti.uv*2+1];
}

UI.printlnInfo(Module.GEOM, "mine - c");

pl.addIntegerArray("triangles", trisarr);
pl.addPoints("points", InterpolationType.VERTEX, objverts);
pl.addVectors("normals", InterpolationType.VERTEX, objnorm);
pl.addTexCoords("uvs", InterpolationType.VERTEX, objtex);
UI.printlnInfo(Module.GEOM, "mine - d");
TriangleMesh m = new TriangleMesh();
if (m.update(pl, null))
    return m;
    UI.printlnInfo(Module.GEOM, "mine - e");
return null;
}

```

```

private TriangleMesh generate(int[] tris, float[] verts, boolean smoothNormals) {
    ParameterList pl = new ParameterList();
    pl.addIntegerArray("triangles", tris);
    pl.addPoints("points", InterpolationType.VERTEX, verts);
    if (smoothNormals) {
        float[] normals = new float[verts.length]; // filled with 0's
        Point3 p0 = new Point3();
        Point3 p1 = new Point3();
        Point3 p2 = new Point3();
        Vector3 n = new Vector3();
        for (int i3 = 0; i3 < tris.length; i3 += 3) {
            int v0 = tris[i3 + 0];
            int v1 = tris[i3 + 1];
            int v2 = tris[i3 + 2];
            p0.set(verts[3 * v0 + 0], verts[3 * v0 + 1], verts[3 * v0 + 2]);
            p1.set(verts[3 * v1 + 0], verts[3 * v1 + 1], verts[3 * v1 + 2]);
            p2.set(verts[3 * v2 + 0], verts[3 * v2 + 1], verts[3 * v2 + 2]);
            Point3.normal(p0, p1, p2, n); // compute normal
            // add face normal to each vertex
            // note that these are not normalized so this in fact weights
            // each normal by the area of the triangle
            normals[3 * v0 + 0] += n.x;
            normals[3 * v0 + 1] += n.y;
            normals[3 * v0 + 2] += n.z;
            normals[3 * v1 + 0] += n.x;
            normals[3 * v1 + 1] += n.y;
            normals[3 * v1 + 2] += n.z;
            normals[3 * v2 + 0] += n.x;
            normals[3 * v2 + 1] += n.y;
            normals[3 * v2 + 2] += n.z;
        }
    }
}

```

```

        // normalize all the vectors
        for (int i3 = 0; i3 < normals.length; i3 += 3) {
            n.set(normals[i3 + 0], normals[i3 + 1], normals[i3 + 2]);
            n.normalize();
            normals[i3 + 0] = n.x;
            normals[i3 + 1] = n.y;
            normals[i3 + 2] = n.z;
        }
        pl.addVectors("normals", InterpolationType.VERTEX, normals);
    }
    TriangleMesh m = new TriangleMesh();
    if (m.update(pl, null))
        return m;
    // something failed in creating the mesh, the error message will be
    // printed by the mesh itself - no need to repeat it here
    return null;
}

public boolean update(ParameterList pl, SunflowAPI api) {
    String file = pl.getString("filename", null);
    if (file != null)
        filename = api.resolveIncludeFilename(file);
    smoothNormals = pl.getBoolean("smooth_normals", smoothNormals);
    return filename != null;
}

private int getLittleEndianInt(int i) {
    // input integer has its bytes in big endian byte order
    // swap them around
    return (i >>> 24) | ((i >>> 8) & 0xFF00) | ((i << 8) & 0xFF0000) | (i << 24);
}

private float getLittleEndianFloat(int i) {
    // input integer has its bytes in big endian byte order
    // swap them around and interpret data as floating point
    return Float.intBitsToFloat(getLittleEndianInt(i));
}
}
}

```

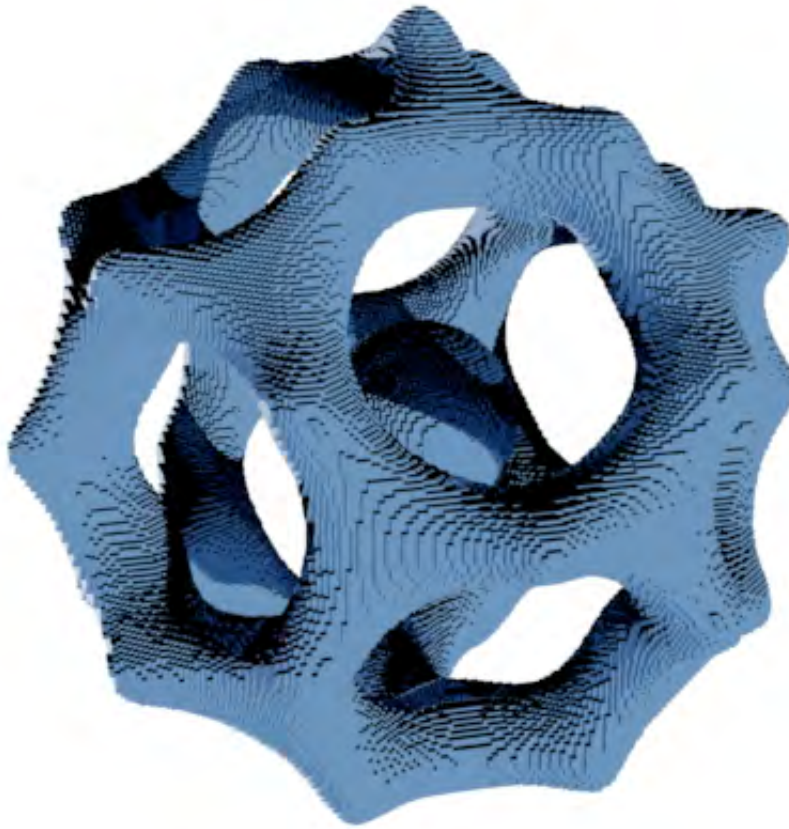
Retrieved from "http://sfwiki.geneome.net/index.php5?title=File_Mesh_Tweak"

- This page was last modified on 1 January 2008, at 16:44.

CubeGrid Isosurface

From Sunflow Wiki

Author: enkidu



Instructions: Change the function to get a different isosurface approximation.

```
import org.sunflow.SunflowAPI;
import org.sunflow.core.*;
import org.sunflow.core.camera.*;
import org.sunflow.core.primitive.*;
import org.sunflow.core.shader.*;
import org.sunflow.image.Color;
import org.sunflow.math.*;

// Change settings here
boolean preview = false;
int gridResolution = 200;

public void build() {
    parameter("eye", new Point3(1.3f, 1.5f, -4.5f));
    parameter("target", new Point3(0, 0, 0));
```

```

    parameter("up", new Vector3(0.0f, 1.0f, 0.0f));
    parameter("fov", 35.0f);
    parameter("aspect", 1.33f);
    camera("camera_outside", new PinholeLens());

    parameter("maxdist", 0.4f);
    parameter("samples", 128);
    shader("ao_ground", new AmbientOcclusionShader());

    geometry("isogrid", new CubeIsosurface( gridResolution ));

    parameter("diffuse", new Color( 0.3f, 0.6f, 1.0f ) );
    shader( "shiny", new ShinyDiffuseShader() );

    parameter("shaders", "shiny");
    instance("isogrid.instance", "isogrid");

    parameter("center", new Point3(0, -1.25f, 0.0f));
    parameter("normal", new Vector3(0.0f, 1.0f, 0.0f));
    geometry("ground", new Plane());
    parameter("shaders", "ao_ground");
    instance("ground.instance", "ground");

    // rendering options
    parameter("camera", "camera_outside");
    parameter("resolutionX", 800);
    parameter("resolutionY", 600);
    if (preview) {
        parameter("aa.min", 0);
        parameter("aa.max", 1);
        parameter("bucket.order", "spiral");
    } else {
        parameter("aa.min", 1);
        parameter("aa.max", 2);
        parameter("bucket.order", "spiral");
        parameter("filter", "mitchell");
    }
    options(DEFAULT_OPTIONS);
}

private static class CubeIsosurface extends CubeGrid {

    private int resolution;

    CubeIsosurface( int resolution ) {
        this.resolution = resolution;
    }

    public boolean update(ParameterList pl, SunflowAPI api) {
        int n = resolution;
        pl.addInteger("resolutionX", n);
        pl.addInteger("resolutionY", n);
        pl.addInteger("resolutionZ", n);
        return super.update(pl, api);
    }

    public double function( double x, double y, double z ){
        // the actual function that defines the isosurface
        // choose from the functions defined below, or define your own!
        return function2( x, y, z );
    }

    public double function1( double x, double y, double z ){
        // CSG example - cylinder subtracted from a sphere with surface turbulence

```

```

double f1 = (x*x + y*y + z*z) - 0.25;    // sphere
// add some displacement to the sphere
f1 += Math.sin( x * 20 + y * 10 + z * 15 ) * 0.02;
double f2 = (x*x + y*y) - 0.06;          // cylinder
return Math.max( f1, -f2 );              // sphere - cylinder
}

public double function2( double x, double y, double z ){
    // icosahedron function taken from k3dsurf

    // scale the function to fit grid range [-0.5, 0.5]
    x *= 11.0; y *= 11.0; z *= 11.0;

    double result = 1.0;
    if( x*x + y*y + z*z < 35 ){
        result = 2 - (Math.cos(x + (1+Math.sqrt(5))/2*y) + Math.cos(x - (1+Math.sqrt(5))/2*y) + Ma
    }

    return result;
}

public double function3( double x, double y, double z ){
    // gyroid function taken from k3dsurf

    // scale the function to fit grid range [-0.5, 0.5]
    x *= 8.0; y *= 8.0; z *= 8.0;

    double result = Math.cos(x) * Math.sin(y) + Math.cos(y) * Math.sin(z) + Math.cos(z) * Math.si

    result = -result + 0.5;

    return result;
}

protected boolean inside(int x, int y, int z) {
    return (function( ((double)x / resolution) - 0.5, ((double)y / resolution) - 0.5, ((double)z
}
}

```

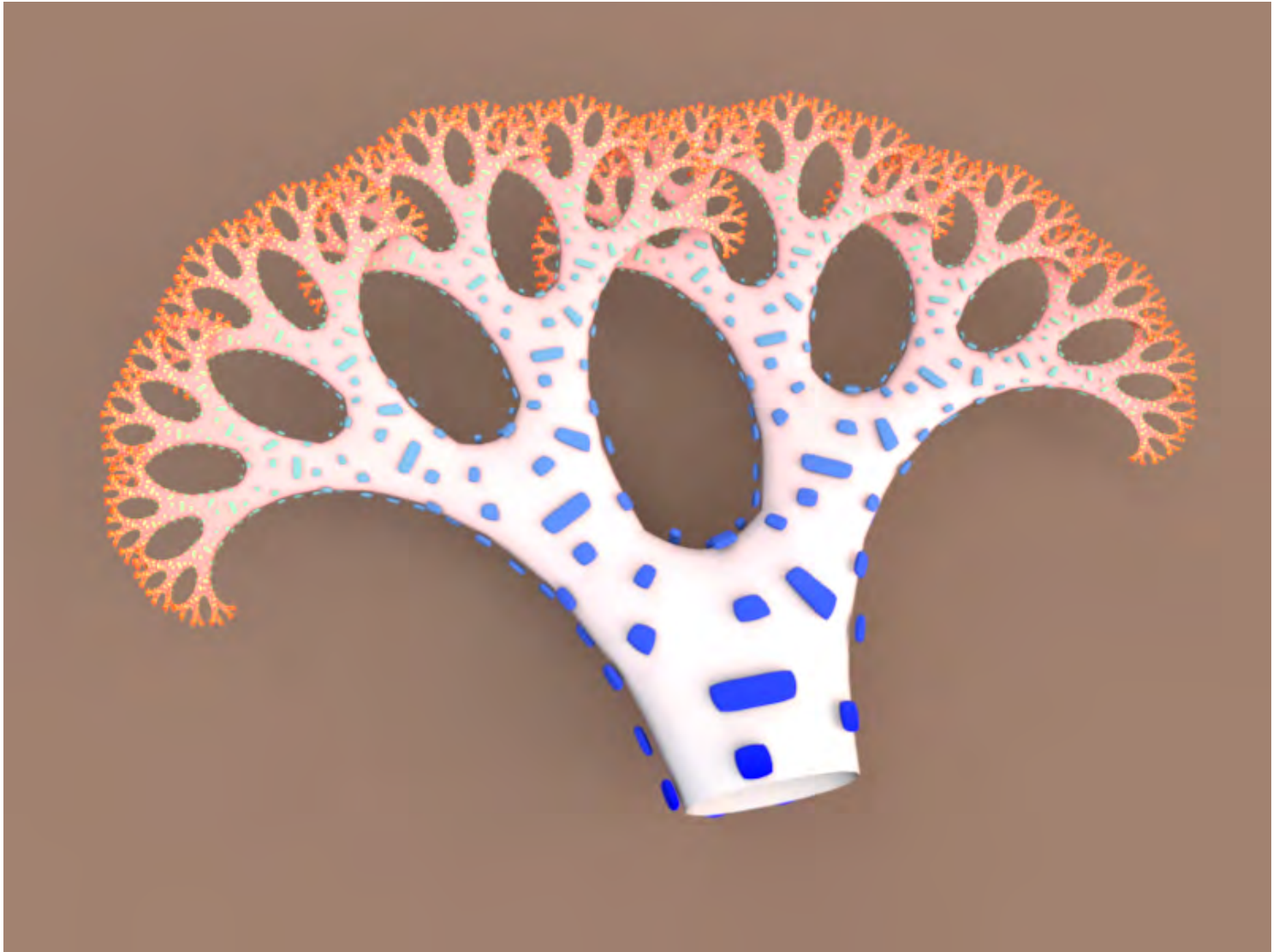
Retrieved from "http://sfwiki.geneome.net/index.php5?title=CubeGrid_Isosurface"

- This page was last modified on 4 December 2007, at 17:39.

Shader Blend Iterations

From Sunflow Wiki

Author: Don Casteel



```
import org.sunflow.core.tesselatable;
import org.sunflow.core.tesselatable.*;
import org.sunflow.SunflowAPI;
import org.sunflow.core.*;
import org.sunflow.core.camera.*;
import org.sunflow.core.primitive.*;
import org.sunflow.core.shader.*;
import org.sunflow.image.Color;
import org.sunflow.math.*;
import org.sunflow.core.LightSource;
import org.sunflow.core.light.*;

// Change settings here
int maxiteration = 10; // total iterations to generate (careful instances are exponential)
float geometryScale = 1f; //scale the initial object before iterating

//define globals
int f = 0;
Matrix4 m = null;
Matrix4 m000 = null;
```

```

Matrix4 m001 = null;
String[] shaderString0 = new String[2];
String[] shaderString1 = new String[2];
Point3 blendPoint0 = new Point3(0f,0f, -2f);
Point3 blendPoint1 = new Point3(0f, 0f,4.6f);

public void build()
{
    parse("prcdrl_FS_003.sc"); // create the scene
    m = Matrix4.scale(geometryScale);
    shaderString0[0] = "itemShader0";
    shaderString0[1] = "itemShader1";
    shaderString1[0] = "itemShader2";
    shaderString1[1] = "itemShader3";

    parameter("shaders", shaderString0);
    parameter("shaders", shaderString1);

    shader("blend0", new BlendShader("blend0", shaderString0, blendPoint0, blendPoint1, 0, maxiteration, Matrix4.sc);
    parameter("shaders", "blend0");

    shader("blend1", new BlendShader("blend1", shaderString1, blendPoint0, blendPoint1, 0, maxiteration, Matrix4.sc);
    parameter("shaders", "blend1");

    String[] shaderString = {"blend0", "blend1"};

    // this is the first instance at the starting point
    parameter("transform", m);
    parameter("shaders", shaderString);
    instance("item.instance_"+f, "item");
    f++;
    float scl = 0.6524214788f;
    float[] matr0 = {
        0.5831704041f, 0.01573119195f, 0.2978083435f, 0f,
        -0.02100930122f, 0.654630097f, 0.006560900709f, 0f,
        -0.2974825862f, -0.01539369208f, 0.5833456481f, 0f,
        -3.050048408f, 0.08159457636f, 5.050982106f, 1f
    };
    m000 = new Matrix4(matr0, false);
    float[] matr1 = {
        0.582074188f, -0.1000984229f, -0.2831871207f, 0f,
        0.03180928248f, 0.6346268361f, -0.1589400783f, 0f,
        0.2986683933f, 0.1274915082f, 0.5688304723f, 0f,
        3.032827956f, 0.2877690502f, 5.071354204f, 1f
    };
    m001 = new Matrix4(matr1, false);
    Matrix4[] trnsfrms = {m000,m001}; /* group the transforms to be passed to the iteration method */
    iterate(Matrix4.scale(1f),0,trnsfrms,0,shaderString0,shaderString1,blendPoint0,blendPoint1); // call the iterat
}

public void iterate(Matrix4 matrix, int itn, Matrix4[] xforms, int fh, String[] shaderString0, String[] shaderStri
{
    itn++;
    Matrix4 m41 = xforms[0];
    Matrix4 m42 = xforms[1];
    m41=m41.multiply(matrix);
    m42=m42.multiply(matrix);
    if(itn<maxiteration)
    {
        f++;
        if(itn<maxiteration)
        {
            m41 = m.multiply(m41);
            Point3 point1 = m41.transformP(pt1);
            Point3 point2 = m41.transformP(pt2);
            String iterShdr0 = "blend0_" + f;// create unique name for this shader
            String iterShdr1 = "blend1_" + f;// create unique name for this shader
            String[] iterShdr = new String[2];
            iterShdr[0]=iterShdr0;
            iterShdr[1]=iterShdr1;
            String inst = "item.instance_"+f;// create unique name for this instance

            shader(iterShdr0, new BlendShader(iterShdr0, shaderString0, new Point3(point1), new Point3(point2), itn,
            parameter("shaders", iterShdr0);

```

```

        shader(iterShdr1, new BlendShader(iterShdr1, shaderString1, new Point3(point1), new Point3(point2), itn,
        parameter("shaders", iterShdr1);

        parameter("transform", m41);
        parameter("shaders", iterShdr);
        instance(inst, "item");
        iterate(m41,itn,xforms,0,shaderString0,shaderString1,pt1,pt2);
    };

    f++;
    if(itn<maxiteration)
    {
        m42 = m.multiply(m42);
        Point3 point1 = m42.transformP(pt1);
        Point3 point2 = m42.transformP(pt2);
        String iterShdr0 = "blend0_" + f;// create unique name for this shader
        String iterShdr1 = "blend1_" + f;// create unique name for this shader
        String[] iterShdr = new String[2];
        iterShdr[0]=iterShdr0;
        iterShdr[1]=iterShdr1;
        String inst = "item.instance_"+f;// create unique name for this instance

        shader(iterShdr0, new BlendShader(iterShdr0, shaderString0, new Point3(point1), new Point3(point2), itn,
        parameter("shaders", iterShdr0);

        shader(iterShdr1, new BlendShader(iterShdr1, shaderString1, new Point3(point1), new Point3(point2), itn,
        parameter("shaders", iterShdr1);

        parameter("transform", m42);
        parameter("shaders", iterShdr);
        instance(inst, "item");
        iterate(m42,itn,xforms,0,shaderString0,shaderString1,pt1,pt2);
    };
};

};

public class BlendShader implements Shader // extends janino
{
    boolean          b1, b2, b3, initflag, updfalg = false;
    String[]          shaderString;
    Point3            point1,point2;
    float             maxiteration,iteration;
    ParameterList     pl;
    SunflowAPI        api;
    String            name;
    Matrix4           matrix;

    public BlendShader(String nm, String[] shaderStr, Point3 pt1, Point3 pt2, int itn, int mxit, Matrix4 mtx4)
    {
        point1 = new Point3(pt1);
        point2 = new Point3(pt2);
        iteration = (float)itn;
        maxiteration = (float)mxit;
        shaderString = shaderStr;
        name = nm;
        matrix = Matrix4.scale(1f);
        matrix = matrix.multiply(mtx4);
    }

    public boolean update(ParameterList p, SunflowAPI a)
    {
        pl=p;
        api=a;
        return true;
    }

    public Color getRadiance(ShadingState state)
    {
        Point3 p = state.getPoint();
        Color base = new Color( api.lookupShader(shaderString[0]).getRadiance(state) );
        state.getPoint().set(p); // restore point
        Color tip = new Color( api.lookupShader(shaderString[1]).getRadiance(state) );
        state.getPoint().set(p); // restore point
        //Color rColor = new Color(0f,0f,0f);

```

```
float iterationOffset = iteration/maxiteration;
float g = gradient(state);
float gradientSegment = g/maxiteration;
float gradientValue = gradientSegment+iterationOffset;
return Color.blend(base, tip, gradientValue);
}

private float gradient(ShadingState state)
{
    Point3 point3 = new Point3(state.getPoint().x,state.getPoint().y,state.getPoint().z);
    Vector3 gradVec = new Vector3(point2.x-point1.x, point2.y-point1.y, point2.z-point1.z);
    Vector3 ptVec = new Vector3(point3.x-point1.x, point3.y-point1.y, point3.z-point1.z);
    float radians = Vector3.dot(gradVec,ptVec);
    float dist = ptVec.length();
    float gradLen = gradVec.length();
    float t = radians/(gradLen*gradLen);
    // Clamp the value just in case
    if(t>1f)t=1f;
    if(t<0f)t=0f;
    return t;
}

// a helper method just in case
public void printMatrix(Matrix4 matrix4, String s)
{
    float[] fa = matrix4.asRowMajor();
    for(int par=0;par<16;par++)
    {
        s = s + fa[par] + " ";
    }
    System.out.println(s);
}

// a helper method just in case
public float radians(float degrees)
{
    return 3.1416f/180f*degrees;
};
```

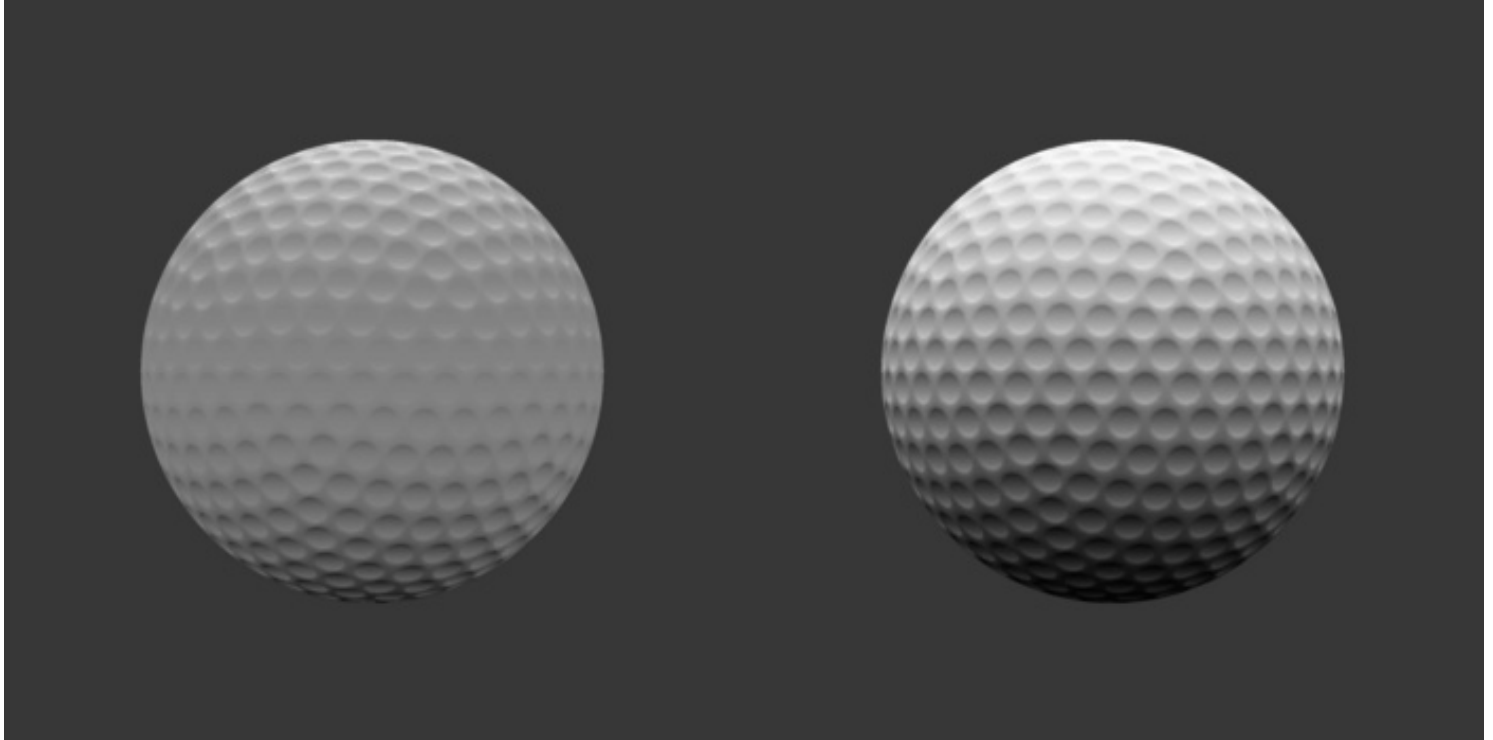
Retrieved from "http://sfwiki.geneome.net/index.php5?title=Shader_Blend_Iterations"

- This page was last modified on 3 December 2007, at 01:39.

Modified Fake Ambient Term

From Sunflow Wiki

Author: Anthony D'Agostino



Here's an improved fake-gi engine (aka, hemilight). The one on the left is the original, and the right is the new one. Use it in the gi block, but use "fake2" instead of "fake".

```
gi {
  type fake2
  up 0 0 1
  sky { "sRGB nonlinear" 1.0 1.0 1.0 }
  ground { "sRGB nonlinear" 0.0 0.0 0.0 }
}
```

The diff to merge into the source:

```
Index: src/org/sunflow/PluginRegistry.java
=====
--- src/org/sunflow/PluginRegistry.java (revision 388)
+++ src/org/sunflow/PluginRegistry.java (working copy)
@@ -39,6 +39,7 @@
 import org.sunflow.core.filter.TriangleFilter;
 import org.sunflow.core.gi.AmbientOcclusionGIEngine;
 import org.sunflow.core.gi.FakeGIEngine;
+import org.sunflow.core.gi.FakeGIEngine2;
 import org.sunflow.core.gi.InstantGI;
 import org.sunflow.core.gi.IrradianceCacheGIEngine;
 import org.sunflow.core.gi.PathTracingGIEngine;
@@ -267,6 +268,7 @@
```

```

// gi engines
giEnginePlugins.registerPlugin("ambocc", AmbientOcclusionGIEngine.class);
giEnginePlugins.registerPlugin("fake", FakeGIEngine.class);
+ giEnginePlugins.registerPlugin("fake2", FakeGIEngine2.class);
giEnginePlugins.registerPlugin("igi", InstantGI.class);
giEnginePlugins.registerPlugin("irr-cache", IrradianceCacheGIEngine.class);
giEnginePlugins.registerPlugin("path", PathTracingGIEngine.class);
@@ -319,4 +321,4 @@
    bitmapWriterPlugins.registerPlugin("exr", EXRBitmapWriter.class);
    bitmapWriterPlugins.registerPlugin("igi", IGIBitmapWriter.class);
}
-}
\ No newline at end of file
+}
Index: src/org/sunflow/core/parser/SCParser.java
=====
--- src/org/sunflow/core/parser/SCParser.java      (revision 388)
+++ src/org/sunflow/core/parser/SCParser.java      (working copy)
@@ -275,6 +275,14 @@
    api.parameter("gi.fake.sky", null, parseColor().getRGB());
    p.checkNextToken("ground");
    api.parameter("gi.fake.ground", null, parseColor().getRGB());
+   } else if (p.peekNextToken("fake2")) {
+       api.parameter("gi.engine", "fake2");
+       p.checkNextToken("up");
+       api.parameter("gi.fake2.up", parseVector());
+       p.checkNextToken("sky");
+       api.parameter("gi.fake2.sky", null, parseColor().getRGB());
+       p.checkNextToken("ground");
+       api.parameter("gi.fake2.ground", null, parseColor().getRGB());
+   } else if (p.peekNextToken("igi")) {
+       api.parameter("gi.engine", "igi");
+       p.checkNextToken("samples");
@@ -1281,4 +1289,4 @@
    return m;
}
-}
\ No newline at end of file
+}

```

Here's the necessary new fake ambient term java file which should be named FakeGIEngine2.java:

```

package org.sunflow.core.gi;

import org.sunflow.core.GIEngine;
import org.sunflow.core.Options;
import org.sunflow.core.Scene;
import org.sunflow.core.ShadingState;
import org.sunflow.image.Color;
import org.sunflow.math.Vector3;

/**
 * This is a quick way to get a bit of ambient lighting into your scene with
 * hardly any overhead. It's based on the formula found here:
 *
 * @link http://www.cs.utah.edu/~shirley/papers/rtrt/node7.html#SECTION00031100000000000000
 */
public class FakeGIEngine2 implements GIEngine {
    private Vector3 up;
    private Color sky;
    private Color ground;
}

```

```
public Color getIrradiance(ShadingState state, Color diffuseReflectance) {
    float cosTheta = Vector3.dot(up, state.getNormal());
    float a = 0.5f + 0.5f*cosTheta;
    return Color.blend(ground, sky, a);
}

public Color getGlobalRadiance(ShadingState state) {
    return Color.BLACK;
}

public boolean init(Options options, Scene scene) {
    up = options.getVector("gi.fake2.up", new Vector3(0, 1, 0)).normalize();
    sky = options.getColor("gi.fake2.sky", Color.WHITE).copy();
    ground = options.getColor("gi.fake2.ground", Color.BLACK).copy();
    sky.mul((float) Math.PI);
    ground.mul((float) Math.PI);
    return true;
}
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Modified_Fake_Ambient_Term"

- This page was last modified on 4 December 2007, at 17:43.

Filter Size Control

From Sunflow Wiki

Author: Eugene Reilly

I had a thought about the filters and how they are a set size depending on the filter you use (e.g. mitchell is always 4.0). That got me thinking about some other programs I use and how the filter size can be set. So I thought I would see the effects of varying filter sizes by hacking Sunflow's source. Not a big deal - but still, it works, and I thought I would share it if anyone else wanted to play. At the moment it doesn't work the right way with the gaussian filter since the offset method uses a value derived from getSize().

The way I set it is to have it defined in the image block like so:

```
image {
    resolution 640 480
    aa 0 1
    filter box
    filter.size 3
}
```

In this example, it's a box filter with a size of 3. The filter.size is optional and if it's not used the default filter size is used (in the box filter's case a size of 1 would be used).

The diff against the SVN 0.07.3 version of the source:

```
Index: src/org/sunflow/core/parser/SCParser.java
=====
--- src/org/sunflow/core/parser/SCParser.java    (revision 391)
+++ src/org/sunflow/core/parser/SCParser.java    (working copy)
@@ -173,6 +173,8 @@
     api.parameter("aa.contrast", p.getNextFloat());
     if (p.peekNextToken("filter"))
         api.parameter("filter", p.getNextToken());
+    if (p.peekNextToken("filter.size"))
+        api.parameter("filter.size", p.getNextFloat());
     if (p.peekNextToken("jitter"))
         api.parameter("aa.jitter", p.getNextBoolean());
     if (p.peekNextToken("show-aa")) {
Index: src/org/sunflow/core/renderer/BucketRenderer.java
=====
--- src/org/sunflow/core/renderer/BucketRenderer.java    (revision 391)
+++ src/org/sunflow/core/renderer/BucketRenderer.java    (working copy)
@@ -55,6 +55,7 @@
     // filtering
     private String filterName;
+    private float filterSize;
     private Filter filter;
     private int fs;
     private float fhs;
@@ -65,6 +66,7 @@
     displayAA = false;
     contrastThreshold = 0.1f;
```



```

        filterName = "box";
+        filterSize = 0.0f;
        jitter = false; // off by default
        dumpBuckets = false; // for debugging only - not user settable
    }
@@ -108,6 +110,7 @@
        thresh = contrastThreshold * (float) Math.pow(2.0f, minAADepth);
        // read filter settings from scene
        filterName = options.getString("filter", filterName);
+        filterSize = options.getFloat("filter.size", filterSize);
        filter = PluginRegistry.filterPlugins.createObject(filterName);
        // adjust filter
        if (filter == null) {
@@ -115,7 +118,10 @@
            filter = new BoxFilter();
            filterName = "box";
        }
-        fhs = filter.getSize() * 0.5f;
+        if (filterSize > 0.0f)
+            fhs = filterSize * 0.5f;
+        else
+            fhs = filter.getSize() * 0.5f;
        fs = (int) Math.ceil(subPixelSize * (fhs - 0.5f));

        // prepare QMC sampling
@@ -133,7 +139,10 @@
        UI.printInfo(Module.BCKT, " * Subpixel jitter:    %s", useJitter ? "on" : (jitter ? "auto
        UI.printInfo(Module.BCKT, " * Contrast threshold: %.2f", contrastThreshold);
        UI.printInfo(Module.BCKT, " * Filter type:      %s", filterName);
        UI.printInfo(Module.BCKT, " * Filter size:      %.2f pixels", filter.getSize());
+        if (filterSize > 0.0f)
+            UI.printInfo(Module.BCKT, " * Filter size:      %.2f pixels", filterSize
+        else
+            UI.printInfo(Module.BCKT, " * Filter size:      %.2f pixels", filter.get
        return true;
    }

```

Here's a Java 1.6 SVN 0.07.3 build here (.zip) (<http://www.geneome.net/drawer/sunflow/fs-sunflow.zip>)

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Filter_Size_Control"

- This page was last modified on 19 December 2007, at 15:52.

IsoCube Primitive Using 0.06.3

From Sunflow Wiki

Author: oodmd

From the forum (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=468>) : I'm not sure if anybody is interested, but i created an isocube primitive for version .06, (as one voxel in a whole isosurface). I don't have any idea how easily this can be updated to work with v .07, and I'm sorry for doing it in .06, but version 06 was easier for me to create a real time view port for what I'm really doing (creating a fluid dynamics simulator based on the paper by Foster and Fedkiw).

and btw, i really must thank the writer of Sunflow.

this code is completely unoptimized and currently relies on the jama package (<http://math.nist.gov/javanumerics/jama/#Package>)

here it is:

```
package org.sunflow.core.primitive;

import org.sunflow.core.BoundedPrimitive;
import org.sunflow.core.IntersectionState;
import org.sunflow.core.Ray;
import org.sunflow.core.Shader;
import org.sunflow.core.ShadingState;
import org.sunflow.math.BoundingBox;
import org.sunflow.math.MathUtils;
import org.sunflow.math.Matrix;
import org.sunflow.math.Matrix4f;
import org.sunflow.math.OrthoNormalBasis;
import org.sunflow.math.Point3;
import org.sunflow.math.Solvers;
import org.sunflow.math.Vector3;

/**
 * @author Matt
 *
 */
class Value4{
    float x, y, z, v;

    public Value4(float x, float y, float z, float v){
        this.v=v;
        this.x=x;
        this.y=y;
        this.z=z;
    }
}

public class IsoCube implements BoundedPrimitive {
    private float minX, minY, minZ;
    private float maxX, maxY, maxZ;

    private float a, b, c, d, e, f, g, h;
```

```

private BoundingBox lightBounds;

private Shader shader;

/**
 * @param t
 * @return
 */
public float getValue(Point3 t){
    return
        a*t.x*t.y*t.z +
        b*t.x*t.y +
        c*t.y*t.z +
        d*t.x*t.z +
        e*t.x +
        f*t.y+
        g*t.z +
        h;          //      return (( v_0 - v_1 )*(m.x-minX)/(maxX-minX))+v_1;
}

/**
 * @param m
 * @return
 */
public Vector3 getNormal(Point3 m){
    float vX=a*m.y*m.z + b*m.y + d*m.z + e;
    float vY=a*m.x*m.z + b*m.x + c*m.z + f;
    float vZ=a*m.y*m.x + c*m.y + d*m.x + g;
    return new Vector3(vX, vY, vZ).normalize();
}

/**
 * @param wheights
 */
public static Vector3 findCoefficients(
    Point3 m,
    Point3 corner0,
    Point3 corner1,
    float wheight000, float wheight001,
    float wheight010, float wheight011,
    float wheight100, float wheight101,
    float wheight110, float wheight111
){

    float minX = Math.min(corner0.x, corner1.x);
    float minY = Math.min(corner0.y, corner1.y);
    float minZ = Math.min(corner0.z, corner1.z);
    float maxX = Math.max(corner0.x, corner1.x);
    float maxY = Math.max(corner0.y, corner1.y);
    float maxZ = Math.max(corner0.z, corner1.z);

    Value4[] s={
        new Value4(minX, minY, minZ, wheight000),
        new Value4(maxX, minY, minZ, wheight100),
        new Value4(minX, maxY, minZ, wheight010),
        new Value4(minX, minY, maxZ, wheight001),
        new Value4(minX, maxY, maxZ, wheight011),
        new Value4(maxX, minY, maxZ, wheight101),
        new Value4(maxX, maxY, minZ, wheight110),
        new Value4(maxX, maxY, maxZ, wheight111),
    };
};

```

```

double[][] pig=new double[s.length][8];
for (int i=0;i<pig.length;i++){
    pig[i][0]=s[i].x*s[i].y*s[i].z;

    pig[i][1]=s[i].x*s[i].y;

    pig[i][2]=s[i].y*s[i].z;

    pig[i][3]=s[i].x*s[i].z;

    pig[i][4]=s[i].x;

    pig[i][5]=s[i].y;

    pig[i][6]=s[i].z;

    pig[i][7]=1;
}
double [][] bigwist=
{
    {s[0].v},
    {s[1].v},
    {s[2].v},
    {s[3].v},
    {s[4].v},
    {s[5].v},
    {s[6].v},
    {s[7].v},
};

Matrix mpm=new Matrix(pig);
bigwist=mpm.solve(new Matrix( bigwist)).getArray();

float a=(float)bigwist[0][0];
float b=(float)bigwist[1][0];
float c=(float)bigwist[2][0];
float d=(float)bigwist[3][0];
float e=(float)bigwist[4][0];
float f=(float)bigwist[5][0];
float g=(float)bigwist[6][0];
float h=(float)bigwist[7][0];

float vX=a*m.y*m.z + b*m.y + d*m.z + e;
float vY=a*m.x*m.z + b*m.x + c*m.z + f;
float vZ=a*m.y*m.x + c*m.y + d*m.x + g;
return new Vector3(vX, vY, vZ);
}

/**
 * @param wheights
 */
public void findCoefficients(Wheights wheights){
    Value4[] s={
        new Value4(minX, minY, minZ, wheights.get(0, 0, 0)),
        new Value4(maxX, minY, minZ, wheights.get(1, 0, 0)),
        new Value4(minX, maxY, minZ, wheights.get(0, 1, 0)),
        new Value4(minX, minY, maxZ, wheights.get(0, 0, 1)),
        new Value4(minX, maxY, maxZ, wheights.get(0, 1, 1)),
        new Value4(maxX, minY, maxZ, wheights.get(1, 0, 1)),
        new Value4(maxX, maxY, minZ, wheights.get(1, 1, 0)),
        new Value4(maxX, maxY, maxZ, wheights.get(1, 1, 1)),
    };
};

```

```

double[][] pig=new double[s.length][8];
for (int i=0;i<pig.length;i++){
    pig[i][0]=s[i].x*s[i].y*s[i].z;

    pig[i][1]=s[i].x*s[i].y;

    pig[i][2]=s[i].y*s[i].z;

    pig[i][3]=s[i].x*s[i].z;

    pig[i][4]=s[i].x;

    pig[i][5]=s[i].y;

    pig[i][6]=s[i].z;

    pig[i][7]=1;
}
double [][] bigwist=
{
    {s[0].v},
    {s[1].v},
    {s[2].v},
    {s[3].v},
    {s[4].v},
    {s[5].v},
    {s[6].v},
    {s[7].v},
};

Matrix m=new Matrix(pig);
bigwist=m.solve(new Matrix( bigwist)).getArray();

a=(float)bigwist[0][0];
b=(float)bigwist[1][0];
c=(float)bigwist[2][0];
d=(float)bigwist[3][0];
e=(float)bigwist[4][0];
f=(float)bigwist[5][0];
g=(float)bigwist[6][0];
h=(float)bigwist[7][0];
}

/**
 * @author Matt
 *
 */
private class Wheights{
    Float [] values=new Float[8];

    /**
     * @param values
     */
    public Wheights(float[] values){
        for (int i=8; --i>=0;){
            this.values[i]=new Float(values[i]);
        }
    }

    /**
     * @param values
     */
    public Wheights(Float[] values){
        this.values=values;
    }
}

```

```

/**
 * @param wheight000
 * @param wheight001
 * @param wheight010
 * @param wheight011
 * @param wheight100
 * @param wheight101
 * @param wheight110
 * @param wheight111
 */
public Wheights(
    float wheight000, float wheight001,
    float wheight010, float wheight011,
    float wheight100, float wheight101,
    float wheight110, float wheight111){

    set(wheight000, 0,0,0); set(wheight001, 0,0,1);
    set(wheight010, 0,1,0); set(wheight011, 0,1,1);
    set(wheight100, 1,0,0); set(wheight101, 1,0,1);
    set(wheight110, 1,1,0); set(wheight111, 1,1,1);
    return;
};

/**
 * @param wheight000
 * @param wheight001
 * @param wheight010
 * @param wheight011
 * @param wheight100
 * @param wheight101
 * @param wheight110
 * @param wheight111
 */
public Wheights(
    Float wheight000, Float wheight001,
    Float wheight010, Float wheight011,
    Float wheight100, Float wheight101,
    Float wheight110, Float wheight111){

    set(wheight000, 0,0,0); set(wheight001, 0,0,1);
    set(wheight010, 0,1,0); set(wheight011, 0,1,1);
    set(wheight100, 1,0,0); set(wheight101, 1,0,1);
    set(wheight110, 1,1,0); set(wheight111, 1,1,1);
    return;
};

/**
 * @param i
 * @param j
 * @param k
 * @return
 */
public int index(int i, int j, int k){ return (i<<2)+(j<<1)+k; };

/**
 * @param value
 * @param i
 * @param j
 * @param k
 */
public void set(Float value, int i, int j, int k){ values[index(i, j, k)]=value; re

/**
 * @param value

```

```

        * @param i
        */
        public void set(Float value, int i){ values[i]=value; return; };

        /**
        * @param i
        * @param j
        * @param k
        * @return
        */
        public float get( int i, int j, int k){ return values[index(i, j, k)]; };

        /**
        * @param i
        * @return
        */
        public float get( int i){ return values[i]; };
    }

    /**
    * @param shader
    * @param corner0
    * @param corner1
    * @param wheight000
    * @param wheight001
    * @param wheight010
    * @param wheight011
    * @param wheight100
    * @param wheight101
    * @param wheight110
    * @param wheight111
    */
    public IsoCube(
        Shader shader,
        Point3 corner0,
        Point3 corner1,

        Float wheight000, Float wheight001,
        Float wheight010, Float wheight011,
        Float wheight100, Float wheight101,
        Float wheight110, Float wheight111
    ) {

        // cube extents
        minX = Math.min(corner0.x, corner1.x);
        minY = Math.min(corner0.y, corner1.y);
        minZ = Math.min(corner0.z, corner1.z);
        maxX = Math.max(corner0.x, corner1.x);
        maxY = Math.max(corner0.y, corner1.y);
        maxZ = Math.max(corner0.z, corner1.z);

        lightBounds = new BoundingBox();
        lightBounds.include(new Point3(minX, minY, minZ));
        lightBounds.include(new Point3(maxX, maxY, maxZ));
        lightBounds.enlargeUlps();

        this.shader=shader;
        findCoefficients(new Wheights(
            wheight000, wheight001,
            wheight010, wheight011,
            wheight100, wheight101,
            wheight110, wheight111
        ));
    }

    /**

```

```

* @param shader
* @param corner0
* @param corner1
* @param wheight000
* @param wheight001
* @param wheight010
* @param wheight011
* @param wheight100
* @param wheight101
* @param wheight110
* @param wheight111
*/
public IsoCube(
    Shader shader,
    Point3 corner0,
    Point3 corner1,

    float wheight000, float wheight001,
    float wheight010, float wheight011,
    float wheight100, float wheight101,
    float wheight110, float wheight111
) {
    // cube extents
    minX = Math.min(corner0.x, corner1.x);
    minY = Math.min(corner0.y, corner1.y);
    minZ = Math.min(corner0.z, corner1.z);
    maxX = Math.max(corner0.x, corner1.x);
    maxY = Math.max(corner0.y, corner1.y);
    maxZ = Math.max(corner0.z, corner1.z);

    lightBounds = new BoundingBox();
    lightBounds.include(new Point3(minX, minY, minZ));
    lightBounds.include(new Point3(maxX, maxY, maxZ));
    lightBounds.enlargeUlps();

    this.shader=shader;

    findCoeficients(new Wheights(
        wheight000, wheight001,
        wheight010, wheight011,
        wheight100, wheight101,
        wheight110, wheight111
    ));
}

/**
* @param shader
* @param corner0
* @param corner1
* @param wheightvalues
*/
public IsoCube(
    Shader shader,
    Point3 corner0,
    Point3 corner1,
    float[] wheightvalues
) {
    // cube extents
    minX = Math.min(corner0.x, corner1.x);
    minY = Math.min(corner0.y, corner1.y);
    minZ = Math.min(corner0.z, corner1.z);
    maxX = Math.max(corner0.x, corner1.x);
    maxY = Math.max(corner0.y, corner1.y);
    maxZ = Math.max(corner0.z, corner1.z);

```



```

        lightBounds = new BoundingBox();
        lightBounds.include(new Point3(minX, minY, minZ));
        lightBounds.include(new Point3(maxX, maxY, maxZ));
        lightBounds.enlargeUlp();

        this.shader=shader;

        findCoefficients(new Wheights(wheightvalues));
    }

    /**
     * @param shader
     * @param corner0
     * @param corner1
     * @param wheightvalues
     */
    public IsoCube(
        Shader shader,
        Point3 corner0,
        Point3 corner1,
        Float[] wheightvalues
    ) {

        // cube extents
        minX = Math.min(corner0.x, corner1.x);
        minY = Math.min(corner0.y, corner1.y);
        minZ = Math.min(corner0.z, corner1.z);
        maxX = Math.max(corner0.x, corner1.x);
        maxY = Math.max(corner0.y, corner1.y);
        maxZ = Math.max(corner0.z, corner1.z);

        lightBounds = new BoundingBox();
        lightBounds.include(new Point3(minX, minY, minZ));
        lightBounds.include(new Point3(maxX, maxY, maxZ));
        lightBounds.enlargeUlp();

        this.shader=shader;

        findCoefficients(new Wheights(wheightvalues));
    }

    /* (non-Javadoc)
     * @see org.sunflow.core.BoundedPrimitive#getBounds()
     */
    public BoundingBox getBounds() {
        return lightBounds;
    }

    /* (non-Javadoc)
     * @see org.sunflow.core.BoundedPrimitive#intersects(org.sunflow.math.BoundingBox)
     */
    public boolean intersects(BoundingBox box) {
        // this could be optimized
        BoundingBox b = new BoundingBox();
        b.include(new Point3(minX, minY, minZ));
        b.include(new Point3(maxX, maxY, maxZ));
        if (b.intersects(box)) {
            // the box is overlapping or enclosed
            if (!b.contains(new Point3(box.getMinimum().x, box.getMinimum().y, box.getMinimum().z)))
                return true;
            if (!b.contains(new Point3(box.getMinimum().x, box.getMinimum().y, box.getMaximum().z)))
                return true;
            if (!b.contains(new Point3(box.getMaximum().x, box.getMaximum().y, box.getMaximum().z)))
                return true;
        }
    }

```

```

        if (!b.contains(new Point3(box.getMinimum().x, box.getMaximum().y, box.getMaximum().z))
            return true;
        if (!b.contains(new Point3(box.getMaximum().x, box.getMinimum().y, box.getMinimum().z))
            return true;
        if (!b.contains(new Point3(box.getMaximum().x, box.getMinimum().y, box.getMaximum().z))
            return true;
        if (!b.contains(new Point3(box.getMaximum().x, box.getMaximum().y, box.getMinimum().z))
            return true;
        if (!b.contains(new Point3(box.getMaximum().x, box.getMaximum().y, box.getMaximum().z))
            return true;
        // all vertices of the box are inside - the surface of the box is
        // not intersected
    }
    return false;
}

/* (non-Javadoc)
 * @see org.sunflow.core.Primitive#prepareShadingState(org.sunflow.core.ShadingState)
 */
public void prepareShadingState(ShadingState state) {
    state.init();
    state.getRay().getPoint(state.getPoint());
    state.getNormal().set(this.getNormal(state.getPoint()));
    state.getGeoNormal().set(state.getNormal());
    state.setBasis(OrthoNormalBasis.makeFromW(state.getNormal()));
    state.setShader(shader);
}

/* (non-Javadoc)
 * @see org.sunflow.core.Primitive#intersect(org.sunflow.core.Ray, org.sunflow.core.IntersectionState)
 */
public void intersect(Ray r, IntersectionState state) {
    float intervalMin = Float.NEGATIVE_INFINITY;
    float intervalMax = Float.POSITIVE_INFINITY;
    float orgX = r.ox;
    float invDirX = 1 / r.dx;
    float t1, t2;
    t1 = (minX - orgX) * invDirX;
    t2 = (maxX - orgX) * invDirX;
    if (invDirX > 0) {
        if (t1 > intervalMin) {
            intervalMin = t1;
        }
        if (t2 < intervalMax) {
            intervalMax = t2;
        }
    } else {
        if (t2 > intervalMin) {
            intervalMin = t2;
        }
        if (t1 < intervalMax) {
            intervalMax = t1;
        }
    }
    if (intervalMin > intervalMax) return;
    float orgY = r.oy;
    float invDirY = 1 / r.dy;
    t1 = (minY - orgY) * invDirY;
    t2 = (maxY - orgY) * invDirY;
    if (invDirY > 0) {
        if (t1 > intervalMin) {
            intervalMin = t1;
        }
    }
    if (t2 < intervalMax) {

```

```

        intervalMax = t2;
    }
} else {
    if (t2 > intervalMin) {
        intervalMin = t2;
    }
    if (t1 < intervalMax) {
        intervalMax = t1;
    }
}
if (intervalMin > intervalMax) return;
float orgZ = r.oz;
float invDirZ = 1 / r.dz;
t1 = (minZ - orgZ) * invDirZ; // no front wall
t2 = (maxZ - orgZ) * invDirZ;
if (invDirZ > 0) {
    if (t1 > intervalMin) {
        intervalMin = t1;
    }
    if (t2 < intervalMax) {
        intervalMax = t2;
    }
} else {
    if (t2 > intervalMin) {
        intervalMin = t2;
    }
    if (t1 < intervalMax) {
        intervalMax = t1;
    }
}
if (intervalMin > intervalMax) return;

if (!r.isInsideMin(intervalMax)) return;

float dxdy=r.dx*r.dy;
float dydz=r.dy*r.dz;
float dxdz=r.dx*r.dz;

float oxoy=r.ox*r.oy;
float oyoZ=r.oy*r.oz;
float oxoz=r.ox*r.oz;

float A=a*dxdy*r.dz;
float B=
    a*( dydz*r.ox + dxdz*r.oy + dxdy*r.oz)+
    b*dxdy + c*dydz + d*dxdz;
float C=
    a*(r.dz*oxoy + r.dx*oyoZ + r.dy*oxoz)+
    b*(r.dx*r.oy + r.dy*r.ox)+
    c*(r.dy*r.oz + r.dz*r.oy)+
    d*(r.dz*r.ox + r.dx*r.oz)+
    e*r.dx + f*r.dy + g*r.dz;
float D=
    a*oxoy*r.oz +
    b*oxoy +
    c*oyoZ +
    d*oxoz +
    e*r.ox +
    f*r.oy +
    g*r.oz +
    h;
double[] primeZero=Solvers.solveCubic(A,B,C,D);

if (primeZero.length==0) return;

```

```
double sZero=Float.POSITIVE_INFINITY;
for (double i: primeZero){
    if (i<sZero && r.isInside(i) && i<=intervalMax && i>=intervalMin )sZero=i;
}

if (!r.isInside(sZero)) return;
r.setMax((float)sZero);
state.setIntersection(this,0, 0);
return;

}

private static final double _2_3=2d/3d;
private static final double _1_3=1d/3d;

}
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=IsoCube_Primitive_Using_0.06.3"

- This page was last modified on 31 January 2008, at 01:27.

Isometric Lens

From Sunflow Wiki

Author: kickfish (as described on the forum (<http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=493>))

```
package org.sunflow.core.camera;

import org.sunflow.SunflowAPI;
import org.sunflow.core.CameraLens;
import org.sunflow.core.ParameterList;
import org.sunflow.core.Ray;
import org.sunflow.math.Point3;
import org.sunflow.math.Vector3;

public class IsometricLens implements CameraLens {
    private float au, av;
    private float fov, aspect;
    private Point3 eye;

    public boolean update(ParameterList pl, SunflowAPI api) {
        fov = pl.getFloat("fov", fov);
        aspect = pl.getFloat("aspect", aspect);
        eye = pl.getPoint("eye", eye);
        update();
        return true;
    }

    private void update() {
        au = (float) Math.tan(Math.toRadians(fov * 0.5f));
        av = au / aspect;
    }

    public Ray getRay(float x, float y, int imageWidth, int imageHeight, double lensX, double lensY) {
        float du = 0 - au + ((2.0f * au * x) / (imageWidth - 1.0f));
        float dv = 0 - av + ((2.0f * av * y) / (imageHeight - 1.0f));
        Vector3 point = new Vector3(du, dv, -1).normalize();
        float scalar = eye.z / (-1 * point.z);
        return new Ray(scalar * point.x, scalar * point.y, 0, 0, 0, -1);
    }
}
```

Chris went on to say that the use of `eye.z` isn't quite correct. As you would only need to generate the ray as if the camera were at the origin - looking down the negative Z-axis. Sunflow will transform it for you. With Chris's correction the lens would look like so:

```
package org.sunflow.core.camera;

import org.sunflow.SunflowAPI;
import org.sunflow.core.CameraLens;
import org.sunflow.core.ParameterList;
import org.sunflow.core.Ray;
import org.sunflow.math.Point3;
```

```
import org.sunflow.math.Vector3;

public class IsometricLens implements CameraLens {
    private float au, av;
    private float fov, aspect;
    private Point3 eye;

    public boolean update(ParameterList pl, SunflowAPI api) {
        fov = pl.getFloat("fov", fov);
        aspect = pl.getFloat("aspect", aspect);
        eye = pl.getPoint("eye", eye);
        update();
        return true;
    }

    private void update() {
        au = (float) Math.tan(Math.toRadians(fov * 0.5f));
        av = au / aspect;
    }

    public Ray getRay(float x, float y, int imageWidth, int imageHeight, double lensX, double lensY) {
        float du = -au + ((2.0f * au * x) / (imageWidth - 1.0f));
        float dv = -av + ((2.0f * av * y) / (imageHeight - 1.0f));
        return new Ray(du, dv, 0, 0, 0, -1);
    }
}
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Isometric_Lens"

- This page was last modified on 20 April 2008, at 17:35.

API Tips

From Sunflow Wiki

Contents

- 1 Lights
 - 1.1 Light Samples
- 2 Normals
 - 2.1 `math.OrthoNormalBasis.makeFromW`
 - 2.2 `state.getNormal()` and Modifiers
- 3 Rays
 - 3.1 `state.getRay()` and `.getMax()`
 - 3.2 Refraction Rays and Specular
- 4 Shading
 - 4.1 `Color.X` versus `Color.x`
 - 4.2 Loading A Texture
 - 4.3 UVs
- 5 Geometry
 - 5.1 API Methods
 - 5.2 Two-Sided Geometry
- 6 Thread Safe Code
 - 6.1 Example
- 7 Changes In 0.07.3

Lights

Light Samples

The only way to get a valid light sample is inside a light loop. If you are using java 5+, it looks like this:

```
state.initLightSamples();
for (LightSample ls : state) {
    // do stuff with ls
}
```

If you are going with Janino which doesn't yet have support for the extended for-loop syntax of Java 5 you have to do it manually:

```
state.initLightSamples();
for (Iterator it = state.iterator(); it.hasNext();) {
    LightSample ls = (LightSample) it.next();
```

```
// do stuff with ls  
}
```

Keep in mind that for the `Iterator` type you need to use `"import java.util.Iterator;"`.

Normals

`math.OrthoNormalBasis.makeFromW`

The orthonormal basis defines the local tangent frame at the hit point. This is needed for bump mapping or anisotropic reflections. If you're unsure how to fill it out, just make a dummy one from the surface normal like

```
OrthoNormalBasis.makeFromW(state.getNormal())
```

`state.getNormal()` and Modifiers

If you are doing bump mapping via modifiers, `ShadingState`'s `getNormal()` will contain the bumped normal when the shader starts running. There is no easy way to get back the original un-bumped smooth normal. Note that you can always do the bump mapping inside your shader if you prefer by copying the code from the bump modifier into your shader. Then you will be able to store the original normal before changing it. The reason for having the modifier system was so that you could switch shaders without affecting the bump.

Rays

`state.getRay()` and `.getMax()`

`state.getRay()` is the ray that generated the shading state you are currently shading. In other words, for the first hits (seen by the camera) `state.getRay().getMax()` would be the distance from the eye to the current point.

If you shoot your own reflection ray (or any other kind) you could look at `reflectedRay.getMax()` to get an idea of how far the first intersect surface is (after you've traced the ray of course). If nothing was hit, you will get infinity back.

Wouldn't the refracted ray have to be traced before absorption can be correctly calculated?

For the glass shader, the refracted ray is applied when leaving the glass object, which means the `state.getRay()` is the ray fired from the last refraction (or total internal reflection).

Refraction Rays and Specular

At the moment you will get specular highlights when you shoot refraction rays. A possible workaround would be to use a custom shader and manually intercept the rays that have refraction depth > 1 and skip their specular contribution.

Shading

Color.X versus Color.x

When changing a color X (for example, `Color.WHITE.mul(...)`), this is actually changing the value of the constant "WHITE". You should instead use a method that returns an object (for example, `Color.white().mul(..)`).

Chris notes: This could be argued to be a poor design of the Color class. Perhaps the methods should always return new objects - but I believe that would have a small speed penalty. This is one of the drawbacks of Java actually.

static final (in the case of Color.X) just means you can't point the reference to X (for example WHITE) to a different object. But it guarantees nothing about the contents of the object itself. Since the `mul()` method actually modifies objects in place (to avoid creating new ones) - this leads to the observed erratic behavior. Sadly Java doesn't have the concept of "const" as Sun felt it would complicate the language too much. The way you are supposed to design this is to create objects that cannot be mutated. This would mean creating a Color class with final members (they can only be assigned once). Then, each member method (`mul`, `add`, etc ...) would need to create a new object each time.

But this might have a small performance penalty unfortunately. It would be an interesting experiment to try it out under the latest JVMs. Similar caveats apply to the `Point3`, `Vector3` classes as well.

Loading A Texture

The specific method that deals with this is `TextureCache.getTexture(filename, isLinear)`. So you would define the texture like so:

```
Texture myTexture = TextureCache.getTexture("C:\\myPath\\myImage.jpg", true);
```

This will let you have several shaders share the same texture in memory if they all point to the same file. You should call this code during your `update()` method and then use the resulting texture object in `getRadiance()`.

UVs

There are two types of UV calculations: one for the intersection code, and the other for the shading preparation code. The intersection calculation gives you the u/v for free. Note that there is still a difference between the hitpoint (u,v) pair and the texture coordinates. One is for barycentric coords, the other is for actual texture lookups. Only the latter is important for shaders.

Geometry

API Methods

The api has two methods that relate to geometry:

`geometry(..)` `instance(...)`

The first one just lets you declare the kind of geometry you want: triangle mesh, sphere, hair, particles, etc. The second actually puts that geometry in the scene, attaches a shader to it, etc. If you never call the second method, you won't get anything in your scene.

Two-Sided Geometry

Geometry is always two-sided at the moment, there isn't an option for the opposite. If you want such an ability, you'd have to manually do that in a shader.

Thread Safe Code

The simple rule is: don't use global variables unless they are read-only.

Example

Here is code run in a non-thread safe way. Notice that `rColor` is defined globally in the unsafe code. This will lead to poor results (in this case, heavy speckling).

Unsafe:

```
Color rColor;
Texture texr;

public boolean update(ParameterList pl, SunflowAPI api) {
    texr = TextureCache.getTexture("C:\\myPath\\myImage.jpg", true);
    return true;
}

public Color getRadiance(ShadingState state) {
    rColor = texr.getPixel(state.getUV().x, state.getUV().y);
}
```

In the safe code I've defined `rColor` locally in the `getRadiance` method.

Thread Safe:

```
Texture texr;

public boolean update(ParameterList pl, SunflowAPI api) {
    texr = TextureCache.getTexture("C:\\myPath\\myImage.jpg", true);
    return true;
}

public Color getRadiance(ShadingState state) {
    Color rColor = texr.getPixel(state.getUV().x, state.getUV().y);
}
```

Changes In 0.07.3

In 0.07.3 and up:

- "parse" has been replaced by "include"
- "getCurrentFrame()" has been replaced by "currentFrame()"

Retrieved from "http://sfwiki.geneome.net/index.php5?title=API_Tips"

- This page was last modified on 16 December 2008, at 03:50.

Matrix Notation

From Sunflow Wiki

Contents

- 1 The Matrix In Sunflow
- 2 Matrix Notation
- 3 Matrices in Sunflow Transforms (Example)
- 4 Matrix Components (Right Handed Rotation)

The Matrix In Sunflow

You can find the 4x4 matrix class in `org.sunflow.math.Matrix4`.

In Sunflow rotation matrices assume a right-handed convention and is used to represent general affine transformations in 3D. The bottom row of the matrix is assumed to be `[0,0,0,1]`.

Matrix Notation

The 4x4 matrix:

```
m11 m12 m13 m14
m21 m22 m23 m24
m31 m32 m33 m34
m41 m42 m43 m44
```

In Sunflow, the 4x4 matrix can be written as a string in either row or column format:

Written in row format:

```
transform row m11 m12 m13 m14 m21 m22 m23 m24 m31 m32 m33 m34 m41 m42 m43 m44
```

Written in column format:

```
transform col m11 m21 m31 m41 m12 m22 m32 m42 m13 m23 m33 m43 m14 m24 m34 m44
```

Matrices in Sunflow Transforms (Example)

Here is a matrix of a cube in column major order, not rotated or scaled but translated to -4.6, 3.7, 5.8 in x y z world space (notice the first three 1.0 values are scale values):

```
transform col 1.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0 -4.6 3.7 5.8 1.0
```

then rotated 75 degrees about the z axis

```
transform col 0.2588 -0.9659 0.0 0.0 0.9659 0.2588 0.0 0.0 0.0 0.0 1.0 0.0 -4.6 3.7 5.8 1.0
```

then scaled in x y and z by 1.3x

```
transform col 0.3364 -1.2557 0.0 0.0 1.2557 0.3364 0.0 0.0 0.0 0.0 1.3000 0.0 -4.6 3.7 5.8 1.0
```

Notice that the first value (which is both scale in x and cos theta for rotation in y and z) after rotation is 0.2588 (the cosine of 75 degrees) and after scaling is multiplied by 1.3 which is 0.3364.

Matrix Components (Right Handed Rotation)

rotation in x =

```
1      0      0      0
0 cos(theta) -sin(theta) 0
0 sin(theta)  cos(theta) 0
0      0      0      1
```

rotation in y =

```
cos(theta)  0  sin(theta)  0
      0      1      0      0
-sin(theta)  0  cos(theta)  0
      0      0      0      1
```

rotation in z =

```
cos(theta)  -sin(theta)  0  0
sin(theta)   cos(theta)  0  0
      0      0      1  0
      0      0      0  1
```

translation in x, y, z =

```
1  0  0  x
0  1  0  y
0  0  1  z
0  0  0  1
```

scale in x, y, z =

```
x 0 0 0
0 y 0 0
0 0 z 0
0 0 0 1
```

Retrieved from "http://sfwiki.geneome.net/index.php5?title=Matrix_Notation"

- This page was last modified on 15 December 2007, at 04:27.

Benchmarks

From Sunflow Wiki

Contents

- 1 Java Virtual Machine (JVM): Client vs. Server
- 2 Benchmarks
 - 2.1 OS's Tested
 - 2.2 JVM's Tested
 - 2.3 Results
 - 2.4 Conclusions

Java Virtual Machine (JVM): Client vs. Server

Sunflow renders faster if you use a server JVM. If you load Sunflow and see a message stating that "API error : You do not appear to be running Sun's server JVM Performance may suffer. This means that you are running Sunflow using a client java virtual machine (though I've found Fedora 8's IcedTea Java will display this even though it is running a server JVM). You can instead use the server JVM by installed the Java JDK and pointing to it when loading Sunflow and adding the -server switch as in the below example:

```
java -Xmx2G -server -jar sunflow.jar
```

So what's the difference between server and client? Here's the excerpt from the Java VM FAQs (http://java.sun.com/docs/hotspot/HotSpotFAQ.html#compiler_types) :

"These two systems are different binaries. They are essentially two different compilers (JITs) interfacing to the same runtime system. The client system is optimal for applications which need fast startup times or small footprints, the server system is optimal for applications where the overall performance is most important. In general the client system is better suited for interactive applications such as GUIs. Some of the other differences include the compilation policy, heap defaults, and inlining policy."

Benchmarks

Author mikews99:

I've been investigating various renderers and found out about SunFlow. Curious as to its performance, especially since it was a java app, I ran some benchmarks on several different platforms to get an idea of its efficiency. After reading this article, I thought I would compare various OS'es JVM implementation on a single computer in hopes that I'd find the best setup for rendering.

The machine I selected was one (that I had) that I could get the OSes I tried installed properly with minimal variations:

AMD Athlon 64 3800+ (2.4GHz) single core socket 939, nVidia 6150 chipset, using the onboard video 2GB DDR400 memory Hitachi 80GB IDE hard drive on its own channel Yamaha IDE CD-RW on its own channel

OS's Tested

OS's Tested

OS	Notes
Ubuntu Linux 7.10	standard installation
Windows XP SP2	with current patches
Windows Vista	no notes
Windows 2000 SP4	hacked for AMD installation
Fedora 8	no notes
Ubuntu x64 Linux 7.10	standard installation
Windows Vista 64	no notes

JVM's Tested

Java VMs

VM	Notes
IBM JDK Java 6 server for Linux 32-bit	beta
Sun JDK Java 6 server for Linux 32-bit	installed from Ubuntu apt repository
OS X Java 32-bit for 10.4.10 (Tiger)	Apple developer preview 6
Sun JDK Java 6 server for Windows 32-bit	1.6.0_03
Sun JDK Java 6 server for Linux 32-bit	RPM for Fedora
JDK Java 1.7.0 for Fedora 8	IcedTea
Sun JDK Java 6 server for Linux 64-bit	installed from Ubuntu apt repository
Sun JDK Java 6 server for Windows 64-bit	1.6.0_03

The Windows installations were stripped of any unnecessary services and all background processes were killed. The *nix installs had all unnecessary processes killed.

Results

I used the Gumbo_and_teapot.sc example file as a test case. Since I found that the video output affected the times, I minimized the image window when testing. To even out the results, I ran the test four times, but only counted the last three runs so as to eliminate any effects of disk caching. These are the results averaged over three runs:

Java VMs

--

OS	VM	Time
Ubuntu Linux	IBM Java	5m:36s
Ubuntu Linux	Sun Java	4m:20s
OS X	Apple Java	4m:06s
Windows XP	Sun Java	3m:54s
Windows Vista	Sun Java	3m:50s
Windows 2000	Sun Java	3m:48s
Fedora 8	Sun Java	4m:26s
Fedora 8	IcedTea Java	4m:44s
Ubuntu x64 Linux	Sun Java	4m:04s
Windows Vista 64	Sun Java	3m:49s

Conclusions

It was quite surprising to see that Sun Java on Windows XP bested the *nix implementations by a such a large margin. I was even more surprised to see the IBM version fair so poorly. Granted it was the current beta version, so perhaps it hasn't been optimized, but for it to trail by a minute was still quite a lot. Ubuntu had a big improvement going from 32 to 64 bit. Windows Vista 64 bit showed no significant improvement.

Since this is only one file, it's imprudent to draw an absolute conclusion. Large, complex scenes may very well give different results. Scenes that require more memory than the 32 bit OSes can support will only be able to render on 64 bit based OSes. None the less, one can at least take this data to make a more informed choice as to OS and renderer.

As requested I used the Apple Java developer preview 6 on OS X which helped improve it 15 seconds (along with the image window being closed). Also, closing the image window on Ubuntu gave an 11 second improvement. Since the Ubuntu 32 and 64 bit tests were so far apart, I tried installing Fedora Core 8 32 bit and running the tests on both the preinstalled JVM and the Sun JVM to see if there was a problem with the Ubuntu packages. The Fedora times were 4:44 for the included JVM and 4:26 for the RPM downloaded from Sun.

I really can't think of a good reason why the Linux 32 bit implementations should be so much slower than either the 64 bit version or the 32 bit Windows versions. Either Sun didn't bother turning on optimizations for Linux, or they used the gcc compiler, which just doesn't cut it (they should have used the Intel compiler).

Retrieved from "<http://sfwiki.geneome.net/index.php5?title=Benchmarks>"

- This page was last modified on 2 January 2008, at 16:58.

```
/* TRANSLUCENT - WITH STORAGE AND REFLECTION OF PHOTONS */
```

```
shader {  
    name "translucent_sr"  
    type janino  
    <code>  
    import org.sunflow.SunflowAPI;  
    import org.sunflow.core.ParameterList;  
    import org.sunflow.core.Ray;  
    import org.sunflow.core.Shader;  
    import org.sunflow.core.ShadingState;  
    import org.sunflow.image.Color;  
    import org.sunflow.math.Vector3;  
    import org.sunflow.math.Point3;  
    import org.sunflow.math.OrthoNormalBasis;  
    // object color  
    public Color color = Color.WHITE;  
    // object absorption color  
    //public Color absorptionColor = Color.RED;  
    public Color absorptionColor = Color.BLUE;  
    // inverse of absorption color  
    public Color transmittanceColor = absorptionColor.copy().opposite();  
    // global color-saving variable  
    /* FIXME!?? - globals are not good */  
    public Color glob = Color.black();  
    // phong specular color  
    public Color pcolor = Color.BLACK;  
    // object absorption distance  
    public float absorptionDistance = 0.25f;  
    // depth correction parameter  
    public float thickness = 0.002f;  
    // phong specular power  
    public float ppower = 85f;  
    // phong specular samples  
    public int psamples = 1;  
    // phong flag  
    public boolean phong = false;  
  
    public boolean update(ParameterList pl, SunflowAPI api) {  
        color = pl.getColor("color", color);  
        if (absorptionDistance == 0f) {  
            absorptionDistance+= 0.0000001f;  
        }  
        if (!pcolor.isBlack()) {  
            phong = true;  
        }  
        return true;  
    }  
  
    public Color getRadiance(ShadingState state) {  
        Color ret = Color.black();  
        Color absorbtion = Color.white();  
        glob.set(Color.black());  
        state.faceforward();
```

```

state.initLightSamples();
state.initCausticSamples();
if (state.getRefractionDepth() == 0) {
    ret.set(state.diffuse(color).mul(0.5f));
    bury(state,thickness);
} else {
    absorbtion = Color.mul(-state.getRay().getMax() / absorptionDistance, transmittanceColor).exp();
}
state.traceRefraction(new Ray(state.getPoint(), randomVector()), 0);
glob.add(state.diffuse(color));
glob.mul(absorbtion);
if (state.getRefractionDepth() == 0 && phong) {
    bury(state,-thickness);
    glob.add(state.specularPhong(pcolor,ppower,psamples));
}
return glob;
}

```

```

public void bury(ShadingState state, float th) {
    Point3 pt = state.getPoint();
    Vector3 norm = state.getNormal();
    pt.x = pt.x - norm.x * th;
    pt.y = pt.y - norm.y * th;
    pt.z = pt.z - norm.z * th;
}

```

```

public Vector3 randomVector() {
    return new Vector3(
        (float)(2f*Math.random()-1f),
        (float)(2f*Math.random()-1f),
        (float)(2f*Math.random()-1f)
    ).normalize();
}

```

```

public Color getDiffuse(ShadingState state) {
    return color;
}

```

```

public void scatterPhoton(ShadingState state, Color power) {
    Color diffuse = getDiffuse(state);
    state.storePhoton(state.getRay().getDirection(), power, diffuse);
    state.traceReflectionPhoton(new Ray(state.getPoint(), randomVector()), power.mul(diffuse));
}

```

</code>

}

```

shader {
    name "simple_sss"
    type janino
    <code>
import org.sunflow.SunflowAPI;
import org.sunflow.core.ParameterList;
import org.sunflow.core.Ray;
import org.sunflow.core.Shader;
import org.sunflow.core.ShadingState;
import org.sunflow.image.Color;
import org.sunflow.math.OrthoNormalBasis;
import org.sunflow.math.Vector3;
import org.sunflow.math.Point3;

Color diff = new Color(0.3f,1.0f,0.6f);
Color spec = new Color(1.0f,1.0f,1.0f);
Color bright = new Color(1f,1f,1f);
Color dark = new Color(0f,0f,0f);
Color color = new Color(0.6f,0.8f,0.8f);
Color absorbtionColor = new Color(0.6f,0.5f,0.3f).opposite();
int numRays = 5;
float power = 100;
float reflectiveness = 0.8f;
float hardness = 0.15f;
float depth = 0.15f;
float spread = 1f;
float glossyness = 0.8f;
float absorbtionValue = 0.5f;

public boolean update(ParameterList pl, SunflowAPI api) {
    return true;
}

protected Color getDiffuse(ShadingState state) {
    return diff;
}

public Color getRadiance(ShadingState state) {
    state.faceforward();
    state.initLightSamples();
    state.initCausticSamples();

    // execute shader
    float cos = state.getCosND();
    float dn = 2 * cos;
    Vector3 refDir = new Vector3();
    refDir.x = (dn * state.getNormal().x) + state.getRay().getDirection().x;
    refDir.y = (dn * state.getNormal().y) + state.getRay().getDirection().y;
    refDir.z = (dn * state.getNormal().z) + state.getRay().getDirection().z;
    Ray refRay = new Ray(state.getPoint(), refDir);

    Color reflections = Color.WHITE;
    Color highlights = Color.WHITE;

```

```

reflections = state.traceReflection(refRay, 0);
highlights = state.diffuse(getDiffuse(state)).add(state.specularPhong(spec, power, numRays));
reflections = Color.blend(highlights, reflections, reflectiveness);

Color sColor = state.diffuse(getDiffuse(state));
sColor = Color.blend(sColor, reflections, glossiness);
Vector3 norm = state.getNormal();

Point3 pt = state.getPoint();
pt.x += norm.x * spread;
pt.y += norm.y * spread;
pt.z += norm.z * spread;

state.getPoint().set(pt);
Color tColor = state.getIrradiance(sColor);
pt.x -= norm.x * (spread + depth);
pt.y -= norm.y * (spread + depth);
pt.z -= norm.z * (spread + depth);

state.getPoint().set(pt);
state.getNormal().set(state.getRay().getDirection());

Color sssColor = Color.add(diff, tColor.mul(state.occlusion(16, absorbtionValue, bright,
dark).opposite().mul(absorbtionColor)));
sssColor.mul(absorbtionColor);
return Color.blend(sssColor, sColor, hardness);
}

public void scatterPhoton(ShadingState state, Color power) {
    //just a copy of the scatter method in PhongShader.....
    // make sure we are on the right side of the material
    state.faceforward();
    Color d = getDiffuse(state);
    state.storePhoton(state.getRay().getDirection(), power, d);
    float avgD = d.getAverage();
    float avgS = spec.getAverage();
    double rnd = state.getRandom(0, 0, 1);
    if (rnd < avgD) {
        // photon is scattered diffusely
        power.mul(d).mul(1.0f / avgD);
        OrthoNormalBasis onb = state.getBasis();
        double u = 2 * Math.PI * rnd / avgD;
        double v = state.getRandom(0, 1, 1);
        float s = (float) Math.sqrt(v);
        float s1 = (float) Math.sqrt(1.0f - v);
        Vector3 w = new Vector3((float) Math.cos(u) * s, (float) Math.sin(u) * s, s1);
        w = onb.transform(w, new Vector3());
        state.traceDiffusePhoton(new Ray(state.getPoint(), w), power);
    }
    else if (rnd < avgD + avgS) {
        // photon is scattered specularly
        float dn = 2.0f * state.getCosND();
        // reflected direction
        Vector3 refDir = new Vector3();

```

```

        refDir.x = (dn * state.getNormal().x) + state.getRay().dx;
        refDir.y = (dn * state.getNormal().y) + state.getRay().dy;
        refDir.z = (dn * state.getNormal().z) + state.getRay().dz;
        power.mul(spec).mul(1.0f / avgS);
        OrthoNormalBasis onb = state.getBasis();
        double u = 2 * Math.PI * (rnd - avgD) / avgS;
        double v = state.getRandom(0, 1, 1);
        float s = (float) Math.pow(v, 1 / (this.power + 1));
        float s1 = (float) Math.sqrt(1 - s * s);
        Vector3 w = new Vector3((float) Math.cos(u) * s1, (float) Math.sin(u) * s1, s);
        w = onb.transform(w, new Vector3());
        state.traceReflectionPhoton(new Ray(state.getPoint(), w), power);
    }
}

public void init(ShadingState state) {
}
</code>
}

```

```
/* TRANSLUCENT - WITH STORAGE AND REFLECTION OF PHOTONS */
```

```
shader {  
    name "translucent_sr"  
    type janino  
    <code>  
    import org.sunflow.SunflowAPI;  
    import org.sunflow.core.ParameterList;  
    import org.sunflow.core.Ray;  
    import org.sunflow.core.Shader;  
    import org.sunflow.core.ShadingState;  
    import org.sunflow.image.Color;  
    import org.sunflow.math.Vector3;  
    import org.sunflow.math.Point3;  
    import org.sunflow.math.OrthoNormalBasis;  
    // object color  
    public Color color = Color.WHITE;  
    // object absorption color  
    //public Color absorptionColor = Color.RED;  
    public Color absorptionColor = Color.BLUE;  
    // inverse of absorption color  
    public Color transmittanceColor = absorptionColor.copy().opposite();  
    // global color-saving variable  
    /* FIXME!?? - globals are not good */  
    public Color glob = Color.black();  
    // phong specular color  
    public Color pcolor = Color.BLACK;  
    // object absorption distance  
    public float absorptionDistance = 0.25f;  
    // depth correction parameter  
    public float thickness = 0.002f;  
    // phong specular power  
    public float ppower = 85f;  
    // phong specular samples  
    public int psamples = 1;  
    // phong flag  
    public boolean phong = false;  
  
    public boolean update(ParameterList pl, SunflowAPI api) {  
        color = pl.getColor("color", color);  
        if (absorptionDistance == 0f) {  
            absorptionDistance+= 0.0000001f;  
        }  
        if (!pcolor.isBlack()) {  
            phong = true;  
        }  
        return true;  
    }  
  
    public Color getRadiance(ShadingState state) {  
        Color ret = Color.black();  
        Color absorbtion = Color.white();  
        glob.set(Color.black());  
        state.faceforward();
```

```

state.initLightSamples();
state.initCausticSamples();
if (state.getRefractionDepth() == 0) {
    ret.set(state.diffuse(color).mul(0.5f));
    bury(state,thickness);
} else {
    absorbtion = Color.mul(-state.getRay().getMax() / absorptionDistance, transmittanceColor).exp();
}
state.traceRefraction(new Ray(state.getPoint(), randomVector()), 0);
glob.add(state.diffuse(color));
glob.mul(absorbtion);
if (state.getRefractionDepth() == 0 && phong) {
    bury(state,-thickness);
    glob.add(state.specularPhong(pcolor,ppower,psamples));
}
return glob;
}

```

```

public void bury(ShadingState state, float th) {
    Point3 pt = state.getPoint();
    Vector3 norm = state.getNormal();
    pt.x = pt.x - norm.x * th;
    pt.y = pt.y - norm.y * th;
    pt.z = pt.z - norm.z * th;
}

```

```

public Vector3 randomVector() {
    return new Vector3(
        (float)(2f*Math.random()-1f),
        (float)(2f*Math.random()-1f),
        (float)(2f*Math.random()-1f)
    ).normalize();
}

```

```

public Color getDiffuse(ShadingState state) {
    return color;
}

```

```

public void scatterPhoton(ShadingState state, Color power) {
    Color diffuse = getDiffuse(state);
    state.storePhoton(state.getRay().getDirection(), power, diffuse);
    state.traceReflectionPhoton(new Ray(state.getPoint(), randomVector()), power.mul(diffuse));
}

```

</code>

}

Index: src/org/sunflow/core/parser/SCParser.java

```
-----
--- src/org/sunflow/core/parser/SCParser.java (revision 396)
+++ src/org/sunflow/core/parser/SCParser.java (working copy)
@@ -492,6 +492,31 @@
     api.shader(name, "textured_phong");
     else
         api.shader(name, "phong");
+ } else if (p.peekNextToken("sss")) {
+     String tex = null;
+     if (p.peekNextToken("texture"))
+         api.parameter("texture", tex = p.getNextToken());
+     else {
+         p.checkNextToken("diff");
+         api.parameter("diffuse", null, parseColor().getRGB());
+     }
+     if (p.peekNextToken("absorptionDistance"))
+         api.parameter("absorptionDistance", p.getNextFloat());
+     if (p.peekNextToken("absorptionPower"))
+         api.parameter("absorptionPower", p.getNextFloat());
+     if (p.peekNextToken("thickness"))
+         api.parameter("thickness", p.getNextFloat());
+     if (p.peekNextToken("sssRays"))
+         api.parameter("sssRays", p.getNextInt());
+     if (p.peekNextToken("specular"))
+         api.parameter("specular", null, parseColor().getRGB());
+         api.parameter("power", p.getNextFloat());
+     if (p.peekNextToken("phongRays"))
+         api.parameter("phongRays", p.getNextInt());
+     if (tex != null)
+         api.shader(name, "textured_sss");
+     else
+         api.shader(name, "sss");
+ } else if (p.peekNextToken("amb-occ") || p.peekNextToken("amb-occ2")) {
+     String tex = null;
+     if (p.peekNextToken("diff") || p.peekNextToken("bright"))
```

Index: src/org/sunflow/core/shader/SSS_Shader.java

```
-----
--- src/org/sunflow/core/shader/SSS_Shader.java (revision 0)
+++ src/org/sunflow/core/shader/SSS_Shader.java (revision 0)
@@ -0,0 +1,258 @@
+package org.sunflow.core.shader;
+
+import java.io.*;
+import org.sunflow.core.tesselatable.*;
+import org.sunflow.SunflowAPI;
+import org.sunflow.core.*;
+import org.sunflow.core.camera.*;
+import org.sunflow.core.primitive.*;
+import org.sunflow.core.shader.*;
+import org.sunflow.image.Color;
+import org.sunflow.math.*;
+import org.sunflow.core.LightSource;
```

```

+import org.sunflow.core.light.*;
+import org.sunflow.core.LightSample;
+import org.sunflow.core.ParameterList;
+import org.sunflow.core.Ray;
+import org.sunflow.core.Shader;
+import org.sunflow.core.ShadingState;
+import org.sunflow.core.primitive.TriangleMesh;
+import org.sunflow.math.MathUtils;
+import org.sunflow.math.OrthoNormalBasis;
+import org.sunflow.math.Point3;
+import org.sunflow.math.Vector3;
+
+public class SSS_Shader implements Shader
+ {
+     private float thickness;
+     private float absorptionDistance;
+     private float absorptionPower;
+     private int samples;
+     private Color diff;
+     private Color specular;
+     private float power;
+     private int phongRays;
+
+     public SSS_Shader() {
+         thickness = 0.001f;
+         absorptionDistance = 0.001f;
+         absorptionPower = 1f;
+         samples = 10;
+         diff = new Color(0.3f,0.3f,0.3f);;
+         specular = new Color(0.3f,0.3f,0.3f);;
+         power = 100f;
+         phongRays = 10;
+     }
+
+     public boolean update(ParameterList pl, SunflowAPI api)
+     {
+         // the color of the material if not textured
+         diff = pl.getColor("diffuse", diff);
+         // controls how much the length of the sub-rays affect their contribution
+         absorptionDistance = pl.getFloat("absorptionDistance",absorptionDistance);
+         // controls the falloff curve
+         absorptionPower = pl.getFloat("absorptionPower",absorptionPower);
+         // sets the distance below the surface for sampling
+         thickness = pl.getFloat("thickness", thickness);
+         // number of SSS samples
+         samples = pl.getInt("sssRays", samples);
+         // phong specular color (set to 0,0,0 for no highlights)
+         specular = pl.getColor("spec", specular);
+         // phong power
+         power = pl.getFloat("power", power);
+         // phong specular samples
+         phongRays = pl.getInt("phongRays", phongRays);
+
+         return true;
+     };
+

```

```

+
+ public Color getDiffuse(ShadingState state)
+ {
+     return diff;
+ }
+
+ public Color getRadiance(ShadingState state)
+ {
+     state.faceforward();
+     state.initLightSamples();
+     if(state.getDiffuseDepth()>0)return state.diffuse(getDiffuse(state));
+     Color rColor = getSubRadiance(state,getDiffuse(state));
+     if(power>0f)
+     {
+         state.initLightSamples();
+         rColor = rColor.add(state.specularPhong(specular, power, phongRays));
+     }
+     return rColor;
+ }
+
+ public Color getSubRadiance(ShadingState state, Color dColor)
+ {
+     Point3 pt = state.getPoint();
+     Vector3 dir = state.getRay().getDirection();
+     Vector3 norm = state.getNormal();
+     Vector3 rNorm = new Vector3(-norm.x,-norm.y,-norm.z);
+     float fsamp = 1f;// (float)Math.max((float)Math.sqrt((double)samples),1f);
+     float totalabsorbtion = fsamp;//1f;// <----- do not change this value
+     float subdist = 0f;
+     Color c1 = state.diffuse(getDiffuse(state));
+     float[] rgbA = c1.getRGB();
+
+     +//     float red = rgbA[0];
+     +//     float green = rgbA[1];
+     +//     float blue = rgbA[2];
+
+     float red = rgbA[0]*fsamp;
+     float green = rgbA[1]*fsamp;
+     float blue = rgbA[2]*fsamp;
+
+     Point3 pt4 = new Point3(pt.x,pt.y,pt.z);
+     pt4.x = pt.x-norm.x*0.002f;
+     pt4.y = pt.y-norm.y*0.002f;
+     pt4.z = pt.z-norm.z*0.002f;
+
+     Ray thkCheckRay = new Ray(pt4,rNorm);
+     ShadingState thkchk = state.traceFinalGather(thkCheckRay,5);
+     thkchk.init();
+     thkchk.setBasis(state.getBasis());
+     Color na = thkchk.shade();
+     thkchk.faceforward();
+     thkchk.initLightSamples();
+     na = thkchk.diffuse(getDiffuse(thkchk));
+     float thk = thkchk.getRay().getMax()/2f;

```

```

+     if(thickness<thk)
+     {
+         thk=thickness;
+     }
+     pt4.x = pt4.x-norm.x*thk;
+     pt4.y = pt4.y-norm.y*thk;
+     pt4.z = pt4.z-norm.z*thk;
+
+     for(int s=0; s<samples;s++)
+     {
+         Ray sssRay = new Ray(pt4,randomVector(state, s, samples));
+         ShadingState sss = state.traceFinalGather(sssRay,s);
+         sss.init();
+         sss.setBasis(state.getBasis());
+         na = sss.shade();
+         sss.faceAway();
+         sss.initLightSamples();
+         Color c2 = sss.diffuse(getDiffuse(sss));
+         float sssRayDistance = sss.getRay().getMax();
+         if (!c2.isNan() && !c2.isInf())
+         {
+             subdist = (float)
Math.pow(absorptionDistance/(absorptionDistance+(sssRayDistance*sssRayDistance)),absorptionPower);
+             totalabsorbtion += subdist;
+             float[] rgb = c2.getRGB();
+             red   += rgb[0]*subdist;
+             green += rgb[1]*subdist;
+             blue  += rgb[2]*subdist;
+             //System.out.println("sssRayDistance:" + sssRayDistance + "\tsubray color: " + rgb[0] + "\t" + rgb[1] + "\t"
+ rgb[2]);
+         }
+     }
+
+     return totalabsorbtion>0f ? new Color(red/totalabsorbtion,green/totalabsorbtion,blue/totalabsorbtion) : new
Color(0f,1f,0f);
+ }
+
+ private Vector3 randomVector(ShadingState st, int s, int samp)
+ {
+     double r1 = st.getRandom(s, 0, samp);
+     double r2 = st.getRandom(s, 1, samp);
+     return new Vector3
+     (
+         (float) (2 * Math.cos(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+         (float) (2 * Math.sin(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+         (float) (1 - 2 * r2)
+     );
+ }
+
+ /*----- copied from the standard phong shader -----*/
+ public void scatterPhoton(ShadingState state, Color power) {
+     // make sure we are on the right side of the material
+     state.faceforward();
+     Color d = getDiffuse(state);

```

```

+ state.storePhoton(state.getRay().getDirection(), power, d);
+ float avgD = d.getAverage();
+ float avgS = specular.getAverage();
+ double rnd = state.getRandom(0, 0, 1);
+ if (rnd < avgD) {
+     // photon is scattered diffusely
+     power.mul(d).mul(1.0f / avgD);
+     OrthoNormalBasis onb = state.getBasis();
+     double u = 2 * Math.PI * rnd / avgD;
+     double v = state.getRandom(0, 1, 1);
+     float s = (float) Math.sqrt(v);
+     float s1 = (float) Math.sqrt(1.0f - v);
+     Vector3 w = new Vector3((float) Math.cos(u) * s, (float) Math.sin(u) * s, s1);
+     w = onb.transform(w, new Vector3());
+     state.traceDiffusePhoton(new Ray(state.getPoint(), w), power);
+ } else if (rnd < avgD + avgS) {
+     // photon is scattered specularly
+     float dn = 2.0f * state.getCosND();
+     // reflected direction
+     Vector3 refDir = new Vector3();
+     refDir.x = (dn * state.getNormal().x) + state.getRay().dx;
+     refDir.y = (dn * state.getNormal().y) + state.getRay().dy;
+     refDir.z = (dn * state.getNormal().z) + state.getRay().dz;
+     power.mul(specular).mul(1.0f / avgS);
+     OrthoNormalBasis onb = state.getBasis();
+     double u = 2 * Math.PI * (rnd - avgD) / avgS;
+     double v = state.getRandom(0, 1, 1);
+     float s = (float) Math.pow(v, 1 / (this.power + 1));
+     float s1 = (float) Math.sqrt(1 - s * s);
+     Vector3 w = new Vector3((float) Math.cos(u) * s1, (float) Math.sin(u) * s1, s);
+     w = onb.transform(w, new Vector3());
+     state.traceReflectionPhoton(new Ray(state.getPoint(), w), power);
+ }
+ }
+
+/*
+ public void scatterPhoton(ShadingState state, Color power)
+ {
+     Point3 pt = state.getPoint();
+     Vector3 dir = state.getRay().getDirection();
+     Vector3 norm = state.getNormal();
+     Vector3 rNorm = new Vector3(-norm.x, -norm.y, -norm.z);
+     float fsamp = 1f; // (float) Math.max((float) Math.sqrt((double) samples), 1f);
+     float totalabsorbtion = fsamp; // 1f; // <----- do not change this value
+     float subdist = 0f;
+     Color c1 = state.diffuse(getDiffuse(state));
+
+     Point3 pt4 = new Point3(pt.x, pt.y, pt.z);
+     pt4.x = pt.x - norm.x * 0.002f;
+     pt4.y = pt.y - norm.y * 0.002f;
+     pt4.z = pt.z - norm.z * 0.002f;
+
+     Ray thkCheckRay = new Ray(pt4, rNorm);
+     ShadingState thkchk = state.traceFinalGather(thkCheckRay, 5);

```

```

+   thkchk.init();
+   thkchk.setBasis(state.getBasis());
+   Color na = thkchk.shade();
+   thkchk.faceforward();
+   float thk = thkchk.getRay().getMax()/2f;
+   if(thickness<thk)
+   {
+       thk=thickness;
+   }
+   pt4.x = pt4.x-norm.x*thk;
+   pt4.y = pt4.y-norm.y*thk;
+   pt4.z = pt4.z-norm.z*thk;
+
+   for(int s=0; s<samples;s++)
+   {
+       Vector3 normSamp = state.getNormal();
+       Vector3 rNormSamp = new Vector3(-normSamp.x,-normSamp.y,-normSamp.z);
+       Ray sssRay = new Ray(pt4,randomVector(state, s, samples));
+       ShadingState sss = state.createSubPhotonState(sssRay, s);
+       //ShadingState sss = state.traceFinalGather(sssRay,s);
+       sss.init();
+       sss.setBasis(state.getBasis());
+       na = sss.shade();
+       sss.faceAway();
+       float sssRayDistance = sss.getRay().getMax();
+       subdist = (float)
Math.pow(absorptionDistance/(absorptionDistance+(sssRayDistance*sssRayDistance)),absorptionPower);
+       sss.storePhoton(rNormSamp, power.mul(subdist), sss.diffuse(getDiffuse(sss)));
+//       sss.storePhoton(sss.getRay().getDirection(), power, sss.diffuse(getDiffuse(sss)));
+//       sss.traceDiffusePhoton(new Ray(state.getPoint(), rNormSamp), power.mul(subdist));
+   }
+ }
+ */
+ }

```

Index: src/org/sunflow/core/shader/TexturedSSS_Shader.java

```

=====
--- src/org/sunflow/core/shader/TexturedSSS_Shader.java (revision 0)
+++ src/org/sunflow/core/shader/TexturedSSS_Shader.java (revision 0)
@@ -0,0 +1,29 @@
+package org.sunflow.core.shader;
+
+import org.sunflow.SunflowAPI;
+import org.sunflow.core.ParameterList;
+import org.sunflow.core.ShadingState;
+import org.sunflow.core.Texture;
+import org.sunflow.core.TextureCache;
+import org.sunflow.image.Color;
+
+public class TexturedSSS_Shader extends SSS_Shader
+ {
+   private Texture tex;
+
+   public TexturedSSS_Shader() {
+       tex = null;

```

```

+ }
+
+ public boolean update(ParameterList pl, SunflowAPI api) {
+     String filename = pl.getString("texture", null);
+     if (filename != null)
+         tex = TextureCache.getTexture(api.resolveTextureFilename(filename), false);
+     return tex != null && super.update(pl, api);
+ }
+
+ @Override
+ public Color getDiffuse(ShadingState state) {
+     return tex.getPixel(state.getUV().x, state.getUV().y);
+ }
+ }
+ }

```

Index: src/org/sunflow/core/ShadingState.java

```

=====
--- src/org/sunflow/core/ShadingState.java    (revision 396)
+++ src/org/sunflow/core/ShadingState.java    (working copy)
@@ -98,6 +98,16 @@
     return finalGatherState;
 }

+ static ShadingState createSubSurfaceScatterState(Shader shdr, ShadingState state, Ray r, int i, LightServer ls) {
+     ShadingState subSurfaceScatter = new ShadingState(state, state.istate, r, i, 2);
+     subSurfaceScatter.diffuseDepth = 10;
+     subSurfaceScatter.includeLights = false;
+     subSurfaceScatter.includeSpecular = false;
+     subSurfaceScatter.setShader(shdr);
+     subSurfaceScatter.instance = state.getInstance();
+     return subSurfaceScatter;
+ }
+
+ private ShadingState(ShadingState previous, IntersectionState istate, Ray r, int i, int d) {
+     this.r = r;
+     this.istate = istate;
@@ -200,6 +210,29 @@
     p.z += bias * ng.z;
 }

+ public final void faceAway() {
+     // make sure we are on the right side of the material
+     if(ng==null)System.out.println("geo Normal = null");
+     if(n==null)System.out.println("Normal = null");
+     if(basis==null)System.out.println("basis = null");
+
+     if (r.dot(ng) > 0) {
+     } else {
+         // this ensure the ray and the geomtric normal are pointing in the
+         // same direction
+         ng.negate();
+         n.negate();
+         basis.flipW();
+         behind = true;
+     }
+ }

```

```

+   cosND = Math.max(-r.dot(n), 0); // can't be negative
+   // offset the shaded point away from the surface to prevent
+   // self-intersection errors
+   p.x += 0.001f * ng.x;
+   p.y += 0.001f * ng.y;
+   p.z += 0.001f * ng.z;
+ }
+
+ /**
+  * Get x coordinate of the pixel being shaded.
+  *
+  @@ -779,7 +812,67 @@
+   public final ShadingState traceFinalGather(Ray r, int i) {
+       return server.traceFinalGather(this, r, i);
+   }
+ /**
+  public Color shadeSSS(Color dColor, Shader shader,float thickness,float absorptionDistance,float
absorptionPower,int samp)
+   {
+       Point3 origPt = getPoint();
+       Ray origRay = getRay();
+       Vector3 origNormal = getNormal();
+       Vector3 origGeoNormal = getGeoNormal();
+
+
+       diffuseDepth++;
+       Point3 pt = getPoint();
+       Vector3 dir = getRay().getDirection();
+       float totalabsorbtion = 0f; // do not change this value
+       Color c1 = dColor.copy();
+
+       float red = 0f;
+       float green = 0f;
+       float blue = 0f;
+
+       Point3 pt4 = new Point3(pt.x,pt.y,pt.z);
+       pt4.x = pt.x-n.x*thickness;
+       pt4.y = pt.y-n.y*thickness;
+       pt4.z = pt.z-n.z*thickness;
+       for(int s=0; s<samp;s++)
+       {
+           r = new Ray(pt4,randomVector(s, samp));
+           Color na = shade();
+           //faceAway();
+           initLightSamples();
+           c1 = diffuse(dColor);
+           float subdist = 10000f;
+
+           float sssRayDistance = getRay().getMax();
+           if (!c1.isNan() && !c1.isInf())
+           {
+               subdist = (float)
Math.pow(absorptionDistance/(absorptionDistance+(sssRayDistance*sssRayDistance)),absorptionPower);
+               totalabsorbtion += subdist;

```



```

+         float[] rgb = c1.getRGB();
+
+         red   += rgb[0]*subdist;
+         green += rgb[1]*subdist;
+         blue  += rgb[2]*subdist;
+     }
+ }
+ p=origPt = new Point3(p);
+ r= origRay;
+ n= origNormal;
+ ng= origGeoNormal;
+ return new Color(red/totalabsorbtion,green/totalabsorbtion,blue/totalabsorbtion);
+ }
+
+ private Vector3 randomVector(int s, int samp)
+ {
+     double r1 = getRandom(s, 0, samp);
+     double r2 = getRandom(s, 1, samp);
+     return new Vector3(
+         (float) (2 * Math.cos(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+         (float) (2 * Math.sin(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+         (float) (1 - 2 * r2));
+ }
+ */
+ /**
+  * Simple black and white ambient occlusion.
+  *

```

Index: src/org/sunflow/PluginRegistry.java

```

=====
--- src/org/sunflow/PluginRegistry.java (revision 396)
+++ src/org/sunflow/PluginRegistry.java (working copy)
@@ -90,11 +90,13 @@
import org.sunflow.core.shader.PrimIDShader;
import org.sunflow.core.shader.QuickGrayShader;
import org.sunflow.core.shader.ShinyDiffuseShader;
+import org.sunflow.core.shader.SSS_Shader;
import org.sunflow.core.shader.SimpleShader;
import org.sunflow.core.shader.TexturedAmbientOcclusionShader;
import org.sunflow.core.shader.TexturedDiffuseShader;
import org.sunflow.core.shader.TexturedPhongShader;
import org.sunflow.core.shader.TexturedShinyDiffuseShader;
+import org.sunflow.core.shader.TexturedSSS_Shader;
import org.sunflow.core.shader.TexturedWardShader;
import org.sunflow.core.shader.UVShader;
import org.sunflow.core.shader.UberShader;
@@ -183,6 +185,7 @@
    shaderPlugins.registerPlugin("mirror", MirrorShader.class);
    shaderPlugins.registerPlugin("phong", PhongShader.class);
    shaderPlugins.registerPlugin("shiny_diffuse", ShinyDiffuseShader.class);
+
+    shaderPlugins.registerPlugin("sss", SSS_Shader.class);
    shaderPlugins.registerPlugin("uber", UberShader.class);
    shaderPlugins.registerPlugin("ward", AnisotropicWardShader.class);
    shaderPlugins.registerPlugin("wireframe", WireframeShader.class);
@@ -192,6 +195,7 @@

```

```

        shaderPlugins.registerPlugin("textured_diffuse", TexturedDiffuseShader.class);
        shaderPlugins.registerPlugin("textured_phong", TexturedPhongShader.class);
        shaderPlugins.registerPlugin("textured_shiny_diffuse", TexturedShinyDiffuseShader.class);
+       shaderPlugins.registerPlugin("textured_sss", TexturedSSS_Shader.class);
        shaderPlugins.registerPlugin("textured_ward", TexturedWardShader.class);

        // preview shaders
Index: src/org/sunflow/system/ImagePanel.java
=====
--- src/org/sunflow/system/ImagePanel.java      (revision 396)
+++ src/org/sunflow/system/ImagePanel.java      (working copy)
@@ -5,6 +5,7 @@
import java.awt.event.InputEvent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseWheelEvent;
+import java.awt.event.MouseWheelListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
@@ -26,7 +27,7 @@
    private float w, h;
    private long repaintCounter;

-    private class ScrollZoomListener extends MouseInputAdapter {
+    private class ScrollZoomListener extends MouseInputAdapter implements MouseWheelListener {
        int mx;
        int my;
        boolean dragging;
@@ -78,7 +79,6 @@
        mouseDragged(e);
    }

-    @Override
    public void mouseWheelMoved(MouseWheelEvent e) {
        zoom(-20 * e.getWheelRotation(), 0);
    }

```

Index: src/org/sunflow/core/parser/SCParser.java

```
-----
--- src/org/sunflow/core/parser/SCParser.java (revision 396)
+++ src/org/sunflow/core/parser/SCParser.java (working copy)
@@ -492,6 +492,31 @@
     api.shader(name, "textured_phong");
     else
         api.shader(name, "phong");
+ } else if (p.peekNextToken("sss")) {
+     String tex = null;
+     if (p.peekNextToken("texture"))
+         api.parameter("texture", tex = p.getNextToken());
+     else {
+         p.checkNextToken("diff");
+         api.parameter("diffuse", null, parseColor().getRGB());
+     }
+     if (p.peekNextToken("absorptionDistance"))
+         api.parameter("absorptionDistance", p.getNextFloat());
+     if (p.peekNextToken("absorptionPower"))
+         api.parameter("absorptionPower", p.getNextFloat());
+     if (p.peekNextToken("thickness"))
+         api.parameter("thickness", p.getNextFloat());
+     if (p.peekNextToken("sssRays"))
+         api.parameter("sssRays", p.getNextInt());
+     if (p.peekNextToken("specular"))
+         api.parameter("specular", null, parseColor().getRGB());
+         api.parameter("power", p.getNextFloat());
+     if (p.peekNextToken("phongRays"))
+         api.parameter("phongRays", p.getNextInt());
+     if (tex != null)
+         api.shader(name, "textured_sss");
+     else
+         api.shader(name, "sss");
+ } else if (p.peekNextToken("amb-occ") || p.peekNextToken("amb-occ2")) {
+     String tex = null;
+     if (p.peekNextToken("diff") || p.peekNextToken("bright"))
```

Index: src/org/sunflow/core/shader/SSS_Shader.java

```
-----
--- src/org/sunflow/core/shader/SSS_Shader.java (revision 0)
+++ src/org/sunflow/core/shader/SSS_Shader.java (revision 0)
@@ -0,0 +1,258 @@
+package org.sunflow.core.shader;
+
+import java.io.*;
+import org.sunflow.core.tesselatable.*;
+import org.sunflow.SunflowAPI;
+import org.sunflow.core.*;
+import org.sunflow.core.camera.*;
+import org.sunflow.core.primitive.*;
+import org.sunflow.core.shader.*;
+import org.sunflow.image.Color;
+import org.sunflow.math.*;
+import org.sunflow.core.LightSource;
```

```

+import org.sunflow.core.light.*;
+import org.sunflow.core.LightSample;
+import org.sunflow.core.ParameterList;
+import org.sunflow.core.Ray;
+import org.sunflow.core.Shader;
+import org.sunflow.core.ShadingState;
+import org.sunflow.core.primitive.TriangleMesh;
+import org.sunflow.math.MathUtils;
+import org.sunflow.math.OrthoNormalBasis;
+import org.sunflow.math.Point3;
+import org.sunflow.math.Vector3;
+
+public class SSS_Shader implements Shader
+ {
+     private float thickness;
+     private float absorptionDistance;
+     private float absorptionPower;
+     private int samples;
+     private Color diff;
+     private Color specular;
+     private float power;
+     private int phongRays;
+
+     public SSS_Shader() {
+         thickness = 0.001f;
+         absorptionDistance = 0.001f;
+         absorptionPower = 1f;
+         samples = 10;
+         diff = new Color(0.3f,0.3f,0.3f);;
+         specular = new Color(0.3f,0.3f,0.3f);;
+         power = 100f;
+         phongRays = 10;
+     }
+
+     public boolean update(ParameterList pl, SunflowAPI api)
+     {
+         // the color of the material if not textured
+         diff = pl.getColor("diffuse", diff);
+         // controls how much the length of the sub-rays affect their contribution
+         absorptionDistance = pl.getFloat("absorptionDistance",absorptionDistance);
+         // controls the falloff curve
+         absorptionPower = pl.getFloat("absorptionPower",absorptionPower);
+         // sets the distance below the surface for sampling
+         thickness = pl.getFloat("thickness", thickness);
+         // number of SSS samples
+         samples = pl.getInt("sssRays", samples);
+         // phong specular color (set to 0,0,0 for no highlights)
+         specular = pl.getColor("spec", specular);
+         // phong power
+         power = pl.getFloat("power", power);
+         // phong specular samples
+         phongRays = pl.getInt("phongRays", phongRays);
+
+         return true;
+     };
+

```

```

+
+ public Color getDiffuse(ShadingState state)
+ {
+     return diff;
+ }
+
+ public Color getRadiance(ShadingState state)
+ {
+     state.faceforward();
+     state.initLightSamples();
+     if(state.getDiffuseDepth()>0)return state.diffuse(getDiffuse(state));
+     Color rColor = getSubRadiance(state,getDiffuse(state));
+     if(power>0f)
+     {
+         state.initLightSamples();
+         rColor = rColor.add(state.specularPhong(specular, power, phongRays));
+     }
+     return rColor;
+ }
+
+ public Color getSubRadiance(ShadingState state, Color dColor)
+ {
+     Point3 pt = state.getPoint();
+     Vector3 dir = state.getRay().getDirection();
+     Vector3 norm = state.getNormal();
+     Vector3 rNorm = new Vector3(-norm.x,-norm.y,-norm.z);
+     float fsamp = 1f;// (float)Math.max((float)Math.sqrt((double)samples),1f);
+     float totalabsorbtion = fsamp;//1f;// <----- do not change this value
+     float subdist = 0f;
+     Color c1 = state.diffuse(getDiffuse(state));
+     float[] rgbA = c1.getRGB();
+
+     +//     float red = rgbA[0];
+     +//     float green = rgbA[1];
+     +//     float blue = rgbA[2];
+
+     float red = rgbA[0]*fsamp;
+     float green = rgbA[1]*fsamp;
+     float blue = rgbA[2]*fsamp;
+
+     Point3 pt4 = new Point3(pt.x,pt.y,pt.z);
+     pt4.x = pt.x-norm.x*0.002f;
+     pt4.y = pt.y-norm.y*0.002f;
+     pt4.z = pt.z-norm.z*0.002f;
+
+     Ray thkCheckRay = new Ray(pt4,rNorm);
+     ShadingState thkchk = state.traceFinalGather(thkCheckRay,5);
+     thkchk.init();
+     thkchk.setBasis(state.getBasis());
+     Color na = thkchk.shade();
+     thkchk.faceforward();
+     thkchk.initLightSamples();
+     na = thkchk.diffuse(getDiffuse(thkchk));
+     float thk = thkchk.getRay().getMax()/2f;

```

```

+     if(thickness<thk)
+     {
+         thk=thickness;
+     }
+     pt4.x = pt4.x-norm.x*thk;
+     pt4.y = pt4.y-norm.y*thk;
+     pt4.z = pt4.z-norm.z*thk;
+
+     for(int s=0; s<samples;s++)
+     {
+         Ray sssRay = new Ray(pt4,randomVector(state, s, samples));
+         ShadingState sss = state.traceFinalGather(sssRay,s);
+         sss.init();
+         sss.setBasis(state.getBasis());
+         na = sss.shade();
+         sss.faceAway();
+         sss.initLightSamples();
+         Color c2 = sss.diffuse(getDiffuse(sss));
+         float sssRayDistance = sss.getRay().getMax();
+         if (!c2.isNan() && !c2.isInf())
+         {
+             subdist = (float)
Math.pow(absorptionDistance/(absorptionDistance+(sssRayDistance*sssRayDistance)),absorptionPower);
+             totalabsorbtion += subdist;
+             float[] rgb = c2.getRGB();
+             red   += rgb[0]*subdist;
+             green += rgb[1]*subdist;
+             blue  += rgb[2]*subdist;
+             //System.out.println("sssRayDistance:" + sssRayDistance + "\tsubray color: " + rgb[0] + "\t" + rgb[1] + "\t"
+ rgb[2]);
+         }
+     }
+
+     return totalabsorbtion>0f ? new Color(red/totalabsorbtion,green/totalabsorbtion,blue/totalabsorbtion) : new
Color(0f,1f,0f);
+ }
+
+ private Vector3 randomVector(ShadingState st, int s, int samp)
+ {
+     double r1 = st.getRandom(s, 0, samp);
+     double r2 = st.getRandom(s, 1, samp);
+     return new Vector3
+     (
+         (float) (2 * Math.cos(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+         (float) (2 * Math.sin(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+         (float) (1 - 2 * r2)
+     );
+ }
+
+ /*----- copied from the standard phong shader -----*/
+ public void scatterPhoton(ShadingState state, Color power) {
+     // make sure we are on the right side of the material
+     state.faceforward();
+     Color d = getDiffuse(state);

```

```

+ state.storePhoton(state.getRay().getDirection(), power, d);
+ float avgD = d.getAverage();
+ float avgS = specular.getAverage();
+ double rnd = state.getRandom(0, 0, 1);
+ if (rnd < avgD) {
+     // photon is scattered diffusely
+     power.mul(d).mul(1.0f / avgD);
+     OrthoNormalBasis onb = state.getBasis();
+     double u = 2 * Math.PI * rnd / avgD;
+     double v = state.getRandom(0, 1, 1);
+     float s = (float) Math.sqrt(v);
+     float s1 = (float) Math.sqrt(1.0f - v);
+     Vector3 w = new Vector3((float) Math.cos(u) * s, (float) Math.sin(u) * s, s1);
+     w = onb.transform(w, new Vector3());
+     state.traceDiffusePhoton(new Ray(state.getPoint(), w), power);
+ } else if (rnd < avgD + avgS) {
+     // photon is scattered specularly
+     float dn = 2.0f * state.getCosND();
+     // reflected direction
+     Vector3 refDir = new Vector3();
+     refDir.x = (dn * state.getNormal().x) + state.getRay().dx;
+     refDir.y = (dn * state.getNormal().y) + state.getRay().dy;
+     refDir.z = (dn * state.getNormal().z) + state.getRay().dz;
+     power.mul(specular).mul(1.0f / avgS);
+     OrthoNormalBasis onb = state.getBasis();
+     double u = 2 * Math.PI * (rnd - avgD) / avgS;
+     double v = state.getRandom(0, 1, 1);
+     float s = (float) Math.pow(v, 1 / (this.power + 1));
+     float s1 = (float) Math.sqrt(1 - s * s);
+     Vector3 w = new Vector3((float) Math.cos(u) * s1, (float) Math.sin(u) * s1, s);
+     w = onb.transform(w, new Vector3());
+     state.traceReflectionPhoton(new Ray(state.getPoint(), w), power);
+ }
+ }
+
+/*
+ public void scatterPhoton(ShadingState state, Color power)
+ {
+     Point3 pt = state.getPoint();
+     Vector3 dir = state.getRay().getDirection();
+     Vector3 norm = state.getNormal();
+     Vector3 rNorm = new Vector3(-norm.x, -norm.y, -norm.z);
+     float fsamp = 1f; // (float) Math.max((float) Math.sqrt((double) samples), 1f);
+     float totalabsorbtion = fsamp; // 1f; // <----- do not change this value
+     float subdist = 0f;
+     Color c1 = state.diffuse(getDiffuse(state));
+
+     Point3 pt4 = new Point3(pt.x, pt.y, pt.z);
+     pt4.x = pt.x - norm.x * 0.002f;
+     pt4.y = pt.y - norm.y * 0.002f;
+     pt4.z = pt.z - norm.z * 0.002f;
+
+     Ray thkCheckRay = new Ray(pt4, rNorm);
+     ShadingState thkchk = state.traceFinalGather(thkCheckRay, 5);

```

```

+   thkchk.init();
+   thkchk.setBasis(state.getBasis());
+   Color na = thkchk.shade();
+   thkchk.faceforward();
+   float thk = thkchk.getRay().getMax()/2f;
+   if(thickness<thk)
+   {
+       thk=thickness;
+   }
+   pt4.x = pt4.x-norm.x*thk;
+   pt4.y = pt4.y-norm.y*thk;
+   pt4.z = pt4.z-norm.z*thk;
+
+   for(int s=0; s<samples;s++)
+   {
+       Vector3 normSamp = state.getNormal();
+       Vector3 rNormSamp = new Vector3(-normSamp.x,-normSamp.y,-normSamp.z);
+       Ray sssRay = new Ray(pt4,randomVector(state, s, samples));
+       ShadingState sss = state.createSubPhotonState(sssRay, s);
+       //ShadingState sss = state.traceFinalGather(sssRay,s);
+       sss.init();
+       sss.setBasis(state.getBasis());
+       na = sss.shade();
+       sss.faceAway();
+       float sssRayDistance = sss.getRay().getMax();
+       subdist = (float)
Math.pow(absorptionDistance/(absorptionDistance+(sssRayDistance*sssRayDistance)),absorptionPower);
+       sss.storePhoton(rNormSamp, power.mul(subdist), sss.diffuse(getDiffuse(sss)));
+//       sss.storePhoton(sss.getRay().getDirection(), power, sss.diffuse(getDiffuse(sss)));
+//       sss.traceDiffusePhoton(new Ray(state.getPoint(), rNormSamp), power.mul(subdist));
+   }
+ }
+*/
+}

```

Index: src/org/sunflow/core/shader/TexturedSSS_Shader.java

```

=====
--- src/org/sunflow/core/shader/TexturedSSS_Shader.java (revision 0)
+++ src/org/sunflow/core/shader/TexturedSSS_Shader.java (revision 0)
@@ -0,0 +1,29 @@
+package org.sunflow.core.shader;
+
+import org.sunflow.SunflowAPI;
+import org.sunflow.core.ParameterList;
+import org.sunflow.core.ShadingState;
+import org.sunflow.core.Texture;
+import org.sunflow.core.TextureCache;
+import org.sunflow.image.Color;
+
+public class TexturedSSS_Shader extends SSS_Shader
+ {
+     private Texture tex;
+
+     public TexturedSSS_Shader() {
+         tex = null;
+     }
+ }

```



```

+ }
+
+ public boolean update(ParameterList pl, SunflowAPI api) {
+     String filename = pl.getString("texture", null);
+     if (filename != null)
+         tex = TextureCache.getTexture(api.resolveTextureFilename(filename), false);
+     return tex != null && super.update(pl, api);
+ }
+
+ @Override
+ public Color getDiffuse(ShadingState state) {
+     return tex.getPixel(state.getUV().x, state.getUV().y);
+ }
+ }
+ }

```

Index: src/org/sunflow/core/ShadingState.java

```

=====
--- src/org/sunflow/core/ShadingState.java    (revision 396)
+++ src/org/sunflow/core/ShadingState.java    (working copy)
@@ -98,6 +98,16 @@
     return finalGatherState;
 }

+ static ShadingState createSubSurfaceScatterState(Shader shdr, ShadingState state, Ray r, int i, LightServer ls) {
+     ShadingState subSurfaceScatter = new ShadingState(state, state.istate, r, i, 2);
+     subSurfaceScatter.diffuseDepth = 10;
+     subSurfaceScatter.includeLights = false;
+     subSurfaceScatter.includeSpecular = false;
+     subSurfaceScatter.setShader(shdr);
+     subSurfaceScatter.instance = state.getInstance();
+     return subSurfaceScatter;
+ }
+
+ private ShadingState(ShadingState previous, IntersectionState istate, Ray r, int i, int d) {
+     this.r = r;
+     this.istate = istate;
@@ -200,6 +210,29 @@
     p.z += bias * ng.z;
 }

+ public final void faceAway() {
+     // make sure we are on the right side of the material
+     if(ng==null)System.out.println("geo Normal = null");
+     if(n==null)System.out.println("Normal = null");
+     if(basis==null)System.out.println("basis = null");
+
+     if (r.dot(ng) > 0) {
+     } else {
+         // this ensure the ray and the geomtric normal are pointing in the
+         // same direction
+         ng.negate();
+         n.negate();
+         basis.flipW();
+         behind = true;
+     }
+ }

```

```

+   cosND = Math.max(-r.dot(n), 0); // can't be negative
+   // offset the shaded point away from the surface to prevent
+   // self-intersection errors
+   p.x += 0.001f * ng.x;
+   p.y += 0.001f * ng.y;
+   p.z += 0.001f * ng.z;
+ }
+
+ /**
+  * Get x coordinate of the pixel being shaded.
+  *
+  @@ -779,7 +812,67 @@
+   public final ShadingState traceFinalGather(Ray r, int i) {
+       return server.traceFinalGather(this, r, i);
+   }
+ /**
+  public Color shadeSSS(Color dColor, Shader shader,float thickness,float absorptionDistance,float
absorptionPower,int samp)
+   {
+       Point3 origPt = getPoint();
+       Ray origRay = getRay();
+       Vector3 origNormal = getNormal();
+       Vector3 origGeoNormal = getGeoNormal();
+
+
+       diffuseDepth++;
+       Point3 pt = getPoint();
+       Vector3 dir = getRay().getDirection();
+       float totalabsorbtion = 0f;// do not change this value
+       Color c1 = dColor.copy();
+
+
+       float red = 0f;
+       float green = 0f;
+       float blue = 0f;
+
+
+       Point3 pt4 = new Point3(pt.x,pt.y,pt.z);
+       pt4.x = pt.x-n.x*thickness;
+       pt4.y = pt.y-n.y*thickness;
+       pt4.z = pt.z-n.z*thickness;
+       for(int s=0; s<samp;s++)
+       {
+           r = new Ray(pt4,randomVector(s, samp));
+           Color na = shade();
+           //faceAway();
+           initLightSamples();
+           c1 = diffuse(dColor);
+           float subdist = 10000f;
+
+
+           float sssRayDistance = getRay().getMax();
+           if (!c1.isNan() && !c1.isInf())
+           {
+               subdist = (float)
Math.pow(absorptionDistance/(absorptionDistance+(sssRayDistance*sssRayDistance)),absorptionPower);
+               totalabsorbtion += subdist;

```

```

+         float[] rgb = c1.getRGB();
+
+         red   += rgb[0]*subdist;
+         green += rgb[1]*subdist;
+         blue  += rgb[2]*subdist;
+     }
+ }
+ p=origPt = new Point3(p);
+ r= origRay;
+ n= origNormal;
+ ng= origGeoNormal;
+ return new Color(red/totalabsorbtion,green/totalabsorbtion,blue/totalabsorbtion);
+ }
+
+ private Vector3 randomVector(int s, int samp)
+ {
+     double r1 = getRandom(s, 0, samp);
+     double r2 = getRandom(s, 1, samp);
+     return new Vector3(
+         (float) (2 * Math.cos(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+         (float) (2 * Math.sin(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+         (float) (1 - 2 * r2));
+ }
+ */
+ /**
+  * Simple black and white ambient occlusion.
+  *

```

Index: src/org/sunflow/PluginRegistry.java

```

=====
--- src/org/sunflow/PluginRegistry.java (revision 396)
+++ src/org/sunflow/PluginRegistry.java (working copy)
@@ -90,11 +90,13 @@
import org.sunflow.core.shader.PrimIDShader;
import org.sunflow.core.shader.QuickGrayShader;
import org.sunflow.core.shader.ShinyDiffuseShader;
+import org.sunflow.core.shader.SSS_Shader;
import org.sunflow.core.shader.SimpleShader;
import org.sunflow.core.shader.TexturedAmbientOcclusionShader;
import org.sunflow.core.shader.TexturedDiffuseShader;
import org.sunflow.core.shader.TexturedPhongShader;
import org.sunflow.core.shader.TexturedShinyDiffuseShader;
+import org.sunflow.core.shader.TexturedSSS_Shader;
import org.sunflow.core.shader.TexturedWardShader;
import org.sunflow.core.shader.UVShader;
import org.sunflow.core.shader.UberShader;
@@ -183,6 +185,7 @@
    shaderPlugins.registerPlugin("mirror", MirrorShader.class);
    shaderPlugins.registerPlugin("phong", PhongShader.class);
    shaderPlugins.registerPlugin("shiny_diffuse", ShinyDiffuseShader.class);
+
+    shaderPlugins.registerPlugin("sss", SSS_Shader.class);
    shaderPlugins.registerPlugin("uber", UberShader.class);
    shaderPlugins.registerPlugin("ward", AnisotropicWardShader.class);
    shaderPlugins.registerPlugin("wireframe", WireframeShader.class);
@@ -192,6 +195,7 @@

```

```
+ shaderPlugins.registerPlugin("textured_diffuse", TexturedDiffuseShader.class);
  shaderPlugins.registerPlugin("textured_phong", TexturedPhongShader.class);
  shaderPlugins.registerPlugin("textured_shiny_diffuse", TexturedShinyDiffuseShader.class);
  shaderPlugins.registerPlugin("textured_sss", TexturedSSS_Shader.class);
  shaderPlugins.registerPlugin("textured_ward", TexturedWardShader.class);

// preview shaders
```

Index: src/org/sunflow/core/gi/FakeGIEngine2.java

```
-----
--- src/org/sunflow/core/gi/FakeGIEngine2.java (revision 0)
+++ src/org/sunflow/core/gi/FakeGIEngine2.java (revision 0)
@@ -0,0 +1,39 @@
+package org.sunflow.core.gi;
+
+import org.sunflow.core.GIEngine;
+import org.sunflow.core.Options;
+import org.sunflow.core.Scene;
+import org.sunflow.core.ShadingState;
+import org.sunflow.image.Color;
+import org.sunflow.math.Vector3;
+
+/**
+ * This is a quick way to get a bit of ambient lighting into your scene with
+ * hardly any overhead. It's based on the formula found here:
+ *
+ * @link http://www.cs.utah.edu/~shirley/papers/rtrt/node7.html#SECTION00031100000000000000
+ */
+public class FakeGIEngine2 implements GIEngine {
+    private Vector3 up;
+    private Color sky;
+    private Color ground;
+
+    public Color getIrradiance(ShadingState state, Color diffuseReflectance) {
+        float cosTheta = Vector3.dot(up, state.getNormal());
+        float a = 0.5f + 0.5f*cosTheta;
+        return Color.blend(ground, sky, a);
+    }
+
+    public Color getGlobalRadiance(ShadingState state) {
+        return Color.BLACK;
+    }
+
+    public boolean init(Options options, Scene scene) {
+        up = options.getVector("gi.fake2.up", new Vector3(0, 1, 0)).normalize();
+        sky = options.getColor("gi.fake2.sky", Color.WHITE).copy();
+        ground = options.getColor("gi.fake2.ground", Color.BLACK).copy();
+        sky.mul((float) Math.PI);
+        ground.mul((float) Math.PI);
+        return true;
+    }
+}
+}
```

Index: src/org/sunflow/core/parser/SCParser.java

```
-----
--- src/org/sunflow/core/parser/SCParser.java (revision 391)
+++ src/org/sunflow/core/parser/SCParser.java (working copy)
@@ -25,6 +25,7 @@
import org.sunflow.system.UI;
import org.sunflow.system.Parser.ParserException;
import org.sunflow.system.UI.Module;
+import org.sunflow.core.shader.MapReader;
```

```

/**
 * This class provides a static method for loading files in the Sunflow scene
 @@ -275,6 +276,14 @@
     api.parameter("gi.fake.sky", null, parseColor().getRGB());
     p.checkNextToken("ground");
     api.parameter("gi.fake.ground", null, parseColor().getRGB());
+ } else if (p.peekNextToken("fake2")) {
+     api.parameter("gi.engine", "fake2");
+     p.checkNextToken("up");
+     api.parameter("gi.fake2.up", parseVector());
+     p.checkNextToken("sky");
+     api.parameter("gi.fake2.sky", null, parseColor().getRGB());
+     p.checkNextToken("ground");
+     api.parameter("gi.fake2.ground", null, parseColor().getRGB());
+ } else if (p.peekNextToken("igi")) {
+     api.parameter("gi.engine", "igi");
+     p.checkNextToken("samples");
@@ -467,17 +476,145 @@
    UI.printlnInfo(Module.API, "Reading shader: %s ...", name);
    p.checkNextToken("type");
    if (p.peekNextToken("diffuse")) {
-     if (p.peekNextToken("diff")) {
+     String tex = null;
+     String mapfile = null;
+     if (p.peekNextToken("solid_texture"))
+     {
+         if(p.peekNextToken("fill"))
+         {
+             api.parameter("fill",null, parseColor().getRGB());
+             p.checkNextToken("veins");
+             api.parameter("veins",null, parseColor().getRGB());
+         }
+         else if(p.peekNextToken("mapfile"))
+         {
+             mapfile = p.getNextToken();
+             MapReader mapreader = new MapReader(mapfile);
+             int[] intColormap = mapreader.getStaticMap();
+             int numColors = 256;
+             //api.parameter("numColors", numColors);
+             float[] colormap = new float[numColors*3];
+             float[] cmapvalues = new float[numColors];

+             for(int remap=0; remap<intColormap.length; remap++)
+             {
+                 colormap[remap] = ((float)intColormap[remap])/256f;
+                 cmapvalues[remap] = ((float)remap)/256f;
+             }

+             for(int mapColors=0;mapColors<numColors;mapColors++)
+             {
+                 float[] mapcolor = parseColor().getRGB();
+                 colormap[mapColors*3] = mapcolor[0];
+                 colormap[mapColors*3+1] = mapcolor[1];

```

```

+         colormap[mapColors*3+2] = mapcolor[2];
+         cmapvalues[mapColors] = p.getNextFloat();
+     }
+     api.parameter("colormap", "float", "NONE", colormap);
+     api.parameter("cmapvalues", "float", "NONE", cmapvalues);
+ }
+ else if(p.peekNextToken("map"))
+ {
+     int numColors = p.getNextInt();
+     //api.parameter("numColors", numColors);
+     float[] colormap = new float[numColors*3];
+     float[] cmapvalues = new float[numColors];
+
+     for(int mapColors=0;mapColors<numColors;mapColors++)
+     {
+         float[] mapcolor = parseColor().getRGB();
+         colormap[mapColors*3] = mapcolor[0];
+         colormap[mapColors*3+1] = mapcolor[1];
+         colormap[mapColors*3+2] = mapcolor[2];
+         cmapvalues[mapColors] = p.getNextFloat();
+     }
+     api.parameter("colormap", "float", "NONE", colormap);
+     api.parameter("cmapvalues", "float", "NONE", cmapvalues);
+ }
+ p.checkNextToken("function");
+ api.parameter("function", p.getNextInt());
+ p.checkNextToken("size");
+ api.parameter("size", p.getNextFloat());
+ p.checkNextToken("scale");
+ api.parameter("scale", p.getNextFloat());
+ api.shader(name, "textured_diffuse");
+ }
+ else if (p.peekNextToken("diff")) {
+     api.parameter("diffuse", null, parseColor().getRGB());
+     api.shader(name, "diffuse");
+ } else if (p.peekNextToken("texture")) {
+     api.parameter("texture", p.getNextToken());
+     api.shader(name, "textured_diffuse");
- } else
+ }
+ else
+     UI.printWarning(Module.API, "Unrecognized option in diffuse shader block: %s", p.getNextToken());
+ } else if (p.peekNextToken("phong")) {
+     String tex = null;
-     if (p.peekNextToken("texture"))
+     String mapfile = null;
+     boolean solid_texture=false;
+     if (p.peekNextToken("solid_texture"))
+     {
+         solid_texture=true;
+         if(p.peekNextToken("fill"))
+         {
+             api.parameter("fill", null, parseColor().getRGB());
+             p.checkNextToken("veins");

```

```

+         api.parameter("veins",null, parseColor().getRGB());
+     }
+ else if(p.peekNextToken("mapfile"))
+     {
+         mapfile = p.getNextToken();
+         MapReader mapreader = new MapReader(mapfile);
+         int[] intColormap = mapreader.getStaticMap();
+         int numColors = 256;
+         //api.parameter("numColors", numColors);
+         float[] colormap = new float[numColors*3];
+         float[] cmapvalues = new float[numColors];
+
+         for(int remap=0; remap<intColormap.length; remap++)
+         {
+             colormap[remap] = ((float)intColormap[remap])/256f;
+             cmapvalues[remap] = ((float)remap)/256f;
+         }
+
+         for(int mapColors=0;mapColors<numColors;mapColors++)
+         {
+             float[] mapcolor = parseColor().getRGB();
+             colormap[mapColors*3] = mapcolor[0];
+             colormap[mapColors*3+1] = mapcolor[1];
+             colormap[mapColors*3+2] = mapcolor[2];
+             cmapvalues[mapColors] = p.getNextFloat();
+         }
+         api.parameter("colormap","float","NONE",colormap);
+         api.parameter("cmapvalues","float","NONE",cmapvalues);
+     }
+ else if(p.peekNextToken("map"))
+     {
+         int numColors = p.getNextInt();
+         //api.parameter("numColors", numColors);
+         float[] colormap = new float[numColors*3];
+         float[] cmapvalues = new float[numColors];
+
+         for(int mapColors=0;mapColors<numColors;mapColors++)
+         {
+             float[] mapcolor = parseColor().getRGB();
+             colormap[mapColors*3] = mapcolor[0];
+             colormap[mapColors*3+1] = mapcolor[1];
+             colormap[mapColors*3+2] = mapcolor[2];
+             cmapvalues[mapColors] = p.getNextFloat();
+         }
+         api.parameter("colormap","float","NONE",colormap);
+         api.parameter("cmapvalues","float","NONE",cmapvalues);
+     }
+ p.checkNextToken("function");
+ api.parameter("function",p.getNextInt());
+ p.checkNextToken("size");
+ api.parameter("size", p.getNextFloat());
+ p.checkNextToken("scale");
+ api.parameter("scale", p.getNextFloat());
+ api.shader(name, "textured_phong");

```



```

+     }
+     else if (p.peekNextToken("texture"))
+         api.parameter("texture", tex = p.getNextToken());
+     else {
+         p.checkNextToken("diff");
@@ -490,8 +627,177 @@
+         api.parameter("samples", p.getNextInt());
+     if (tex != null)
+         api.shader(name, "textured_phong");
+     else if(!solid_texture)api.shader(name, "phong");
+     //else
+     //    UI.printWarning(Module.API, "Unrecognized option in phong shader block: %s", p.getNextToken());
+ } else if (p.peekNextToken("sss")) {
+     String tex = null;
+     String mapfile = null;
+     boolean solidTex = false;
+     boolean volumeTex = false;
+     boolean textureMap = false;
+     if (p.peekNextToken("solid_texture"))
+     {
+         if(p.peekNextToken("fill"))
+         {
+             api.parameter("fill",null, parseColor().getRGB());
+             p.checkNextToken("veins");
+             api.parameter("veins",null, parseColor().getRGB());
+         }
+         else if(p.peekNextToken("mapfile"))
+         {
+             mapfile = p.getNextToken();
+             MapReader mapreader = new MapReader(mapfile);
+             int[] intColormap = mapreader.getStaticMap();
+             int numColors = 256;
+             //api.parameter("numColors", numColors);
+             float[] colormap = new float[numColors*3];
+             float[] cmapvalues = new float[numColors];
+
+             for(int remap=0; remap<intColormap.length; remap++)
+             {
+                 colormap[remap] = ((float)intColormap[remap])/256f;
+                 cmapvalues[remap] = ((float)remap)/256f;
+             }
+
+             for(int mapColors=0;mapColors<numColors;mapColors++)
+             {
+                 float[] mapcolor = parseColor().getRGB();
+                 colormap[mapColors*3] = mapcolor[0];
+                 colormap[mapColors*3+1] = mapcolor[1];
+                 colormap[mapColors*3+2] = mapcolor[2];
+                 cmapvalues[mapColors] = p.getNextFloat();
+             }
+             api.parameter("colormap","float","NONE",colormap);
+             api.parameter("cmapvalues","float","NONE",cmapvalues);
+         }
+     else if(p.peekNextToken("map"))

```

```

+         {
+         int numColors = p.getNextInt();
+         //api.parameter("numColors", numColors);
+         float[] colormap = new float[numColors*3];
+         float[] cmapvalues = new float[numColors];
+
+         for(int mapColors=0;mapColors<numColors;mapColors++)
+         {
+             float[] mapcolor = parseColor().getRGB();
+             colormap[mapColors*3] = mapcolor[0];
+             colormap[mapColors*3+1] = mapcolor[1];
+             colormap[mapColors*3+2] = mapcolor[2];
+             cmapvalues[mapColors] = p.getNextFloat();
+         }
+         api.parameter("colormap","float","NONE",colormap);
+         api.parameter("cmapvalues","float","NONE",cmapvalues);
+     }
+     p.checkNextToken("function");
+     api.parameter("function",p.getNextInt());
+     p.checkNextToken("size");
+     api.parameter("size", p.getNextFloat());
+     p.checkNextToken("scale");
+     api.parameter("scale", p.getNextFloat());
+     solidTex = true;
+ }
else
-     api.shader(name, "phong");
+     if (p.peekNextToken("volume_texture"))
+     {
+         if(p.peekNextToken("fill"))
+         {
+             api.parameter("fill",null, parseColor().getRGB());
+             p.checkNextToken("veins");
+             api.parameter("veins",null, parseColor().getRGB());
+         }
+         else if(p.peekNextToken("mapfile"))
+         {
+             mapfile = p.getNextToken();
+             MapReader mapreader = new MapReader(mapfile);
+             int[] intColormap = mapreader.getStaticMap();
+             int numColors = intColormap.length/3;
+             //api.parameter("numColors", numColors);
+             float[] colormap = new float[numColors*3];
+             float[] cmapvalues = new float[numColors];
+
+             for(int remap=0; remap<intColormap.length; remap++)
+             {
+                 colormap[remap] = ((float)intColormap[remap])/256f;
+             }
+             for(int remap=0; remap<numColors; remap++)
+             {
+                 cmapvalues[remap] = ((float)remap)/256f;
+             }
+             api.parameter("colormap","float","NONE",colormap);

```

```

+         api.parameter("cmapvalues","float","NONE",cmapvalues);
+     }
+ else if(p.peekNextToken("map"))
+     {
+         int numColors = p.getNextInt();
+         //api.parameter("numColors", numColors);
+         float[] colormap = new float[numColors*3];
+         float[] cmapvalues = new float[numColors];
+
+         for(int mapColors=0;mapColors<numColors;mapColors++)
+         {
+             float[] mapcolor = parseColor().getRGB();
+             colormap[mapColors*3] = mapcolor[0];
+             colormap[mapColors*3+1] = mapcolor[1];
+             colormap[mapColors*3+2] = mapcolor[2];
+             cmapvalues[mapColors] = p.getNextFloat();
+         }
+         api.parameter("colormap","float","NONE",colormap);
+         api.parameter("cmapvalues","float","NONE",cmapvalues);
+     }
+ p.checkNextToken("function");
+ api.parameter("function",p.getNextInt());
+ p.checkNextToken("size");
+ api.parameter("size", p.getNextFloat());
+ p.checkNextToken("scale");
+ api.parameter("scale", p.getNextFloat());
+ volumeTex = true;
+ }
+ else if (p.peekNextToken("texture"))
+     api.parameter("texture", tex = p.getNextToken());
+ else {
+     p.checkNextToken("diff");
+     api.parameter("diffuse",null, parseColor().getRGB());
+ }
+ if (p.peekNextToken("opacity"))
+     api.parameter("opacity", p.getNextFloat());
+ if (p.peekNextToken("thickness"))
+     api.parameter("thickness", p.getNextFloat());
+ if (p.peekNextToken("sssRays"))
+     api.parameter("sssRays", p.getNextInt());
+ if (p.peekNextToken("specular"))
+     api.parameter("spec", null, parseColor().getRGB());
+     api.parameter("power", p.getNextFloat());
+ if (p.peekNextToken("phongRays"))
+     api.parameter("phongRays", p.getNextInt());
+ if (p.peekNextToken("maxOpacity"))
+     api.parameter("maxOpacity", p.getNextFloat());
+ if (p.peekNextToken("eta"))
+     api.parameter("eta", p.getNextFloat());
+ if (p.peekNextToken("diffusion"))
+     api.parameter("diffusion", p.getNextFloat());
+ if (p.peekNextToken("sampDist"))
+     api.parameter("sampDist", p.getNextFloat());
+ if (p.peekNextToken("environment"))

```

```

+         api.parameter("environment", p.getNextFloat());
+     if (p.peekNextToken("maxSubRayLength"))
+         api.parameter("maxSubRayLength", p.getNextFloat());
+     if (p.peekNextToken("sssTraceDepth"))
+         api.parameter("sssTraceDepth", p.getNextInt());
+     if (tex != null)
+         api.shader(name, "textured_sss");
+     else if(solidTex)
+     {
+         api.shader(name, "solid_textured_sss");
+         solidTex = false;
+     }
+     else if(volumeTex)
+     {
+         api.shader(name, "volume_textured_sss");
+         volumeTex = false;
+     }
+     else
+         api.shader(name, "sss");
+ } else if (p.peekNextToken("amb-occ") || p.peekNextToken("amb-occ2")) {
+     String tex = null;
+     if (p.peekNextToken("diff") || p.peekNextToken("bright"))

```

Index: src/org/sunflow/core/shader/MapReader.java

```

=====
--- src/org/sunflow/core/shader/MapReader.java (revision 0)
+++ src/org/sunflow/core/shader/MapReader.java (revision 0)

```

```

@@ -0,0 +1,138 @@
+package org.sunflow.core.shader;
+
+import java.lang.Float;
+import java.lang.reflect.Array;
+
+import java.io.*;
+//import java.net.*;
+//import java.applet.*;
+
+import java.awt.*;
+//import java.awt.BorderLayout;
+import java.awt.event.*;
+import java.util.*;
+//import java.beans.*;
+
+//import javax.media.j3d.*;
+//import javax.vecmath.*;
+
+public class MapReader extends Object
+ {
+     boolean appletFlag = true;
+     public static MapReader mapReader;
+     FileDialog fileDialog;
+     File file;
+     File directory;
+     Frame frame;
+     String mapFile;

```

```

+ String mapPath, mapString;
+ int numLines;
+ int nTokens1;
+ String fileType;
+ int[] map = new int[256*3];
+ static int[] staticMap;
+
+// MapReader(Frame f){frame=f;init();};
+// MapReader(){frame = new Frame("Read Fractal Flame File");init();};
+ public MapReader(String mapFile)
+ {
+     File f = new File(mapFile);
+     readFile(f);
+     staticMap = map;
+ }
+
+ public void readFile(File f)
+ {
+     file=f;
+     char[] mapData = new char[5120];
+     try
+     {
+         FileReader fr = new FileReader(file);
+         fr.read(mapData);
+         fr.close();
+     }
+     catch (Exception e){System.out.println(e);};
+     mapString = String.valueOf(mapData).trim();
+     doTokenizer(mapString);
+ }
+/*
+ public File getFilename()
+ {
+     if(file==null)readFile();
+     return file;
+ };
+*/
+ public void doTokenizer(String string)
+ {
+     int intcounter = 0;
+     String delim1 = "\n";
+     StringTokenizer tokenizer1 = new StringTokenizer(string,delim1);
+     nTokens1 = tokenizer1.countTokens();
+     String[] mapValues = new String[256*3];
+     String[] mapLines = new String[nTokens1];
+     for(int t=0;t<nTokens1;t++)
+     {
+         String nextLine = tokenizer1.nextToken();
+         nextLine.trim();
+         mapLines[t]=nextLine;
+     };
+     String delim2 = "\" =";
+     for(int line=0;line<nTokens1;line++)
+     {

```

```

+         StringTokenizer tokenizer2 = new StringTokenizer(mapLines[line],delim2);
+         int nTokens2 = tokenizer2.countTokens();
+         String[] lineElements = new String[nTokens2];
+         for(int el=0;el<3;el++)
+         {
+             String nextString = tokenizer2.nextToken();
+             nextString.trim();
+             mapValues[intcounter++]=nextString;
+         };
+     };
+     for(int c=0;c<256*3;c++)
+     {
+         String valueString = mapValues[c];
+         valueString.trim();
+         double d = Double.parseDouble(valueString);
+         int tempInt = (int)d;
+         map[c]= tempInt;
+     };
+ };
+
+ public int[] getMap()
+ {
+     return map;
+ };
+
+ public static int[] getStaticMap()
+ {
+     return staticMap;
+ };
+
+
+ static class killAdapter extends WindowAdapter
+ {
+     public void windowClosing(WindowEvent event)
+     {
+         System.exit(0);
+     }
+ };
+
+ /*
+ public static void main(String[] args)
+ {
+     mapReader = new MapReader();
+     mapReader.setAppletFlag(false);
+     mapReader.init();
+ };
+
+ public void setAppletFlag(boolean flag)
+ {
+     appletFlag = flag;
+ };
+
+ public void close()
+ {
+     frame.dispose();

```

```
+      };  
+*/  
+    }
```

\ No newline at end of file

Index: src/org/sunflow/core/shader/SSS_Shader.java

```
-----  
--- src/org/sunflow/core/shader/SSS_Shader.java (revision 0)  
+++ src/org/sunflow/core/shader/SSS_Shader.java (revision 0)  
@@ -0,0 +1,189 @@
```

```
+package org.sunflow.core.shader;  
+  
+import java.io.*;  
+import org.sunflow.core.tesselatable.*;  
+import org.sunflow.SunflowAPI;  
+import org.sunflow.core.*;  
+import org.sunflow.core.camera.*;  
+import org.sunflow.core.primitive.*;  
+import org.sunflow.core.shader.*;  
+import org.sunflow.image.Color;  
+import org.sunflow.math.*;  
+import org.sunflow.core.LightSource;  
+import org.sunflow.core.light.*;  
+import org.sunflow.core.LightSample;  
+import org.sunflow.core.ParameterList;  
+import org.sunflow.core.Ray;  
+import org.sunflow.core.Shader;  
+import org.sunflow.core.ShadingState;  
+import org.sunflow.core.primitive.TriangleMesh;  
+import org.sunflow.math.MathUtils;  
+import org.sunflow.math.OrthoNormalBasis;  
+import org.sunflow.math.Point3;  
+import org.sunflow.math.Vector3;  
+  
+public class SSS_Shader implements Shader  
+ {  
+     private float thickness;  
+     private float absorptionDistance;  
+     private float absorptionPower;  
+     private int samples;  
+     private Color diff;  
+     private Color specular;  
+     private float power;  
+     private int phongRays;  
+     boolean solidShader=false;  
+     boolean volumeShader=false;  
+  
+     public SSS_Shader() {  
+         thickness = 0.001f;  
+         absorptionDistance = 0.001f;  
+         absorptionPower = 1f;  
+         samples = 10;  
+         diff = new Color(0.3f,0.3f,0.3f);;  
+         specular = new Color(0.3f,0.3f,0.3f);;  
+         power = 100f;
```

```

+   phongRays = 10;
+ }
+ public boolean update(ParameterList pl, SunflowAPI api)
+ {
+     // the color of the material if not textured
+     diff      = pl.getColor("diffuse", diff);
+     // controls how much the length of the sub-rays affect their contribution
+     absorptionDistance = pl.getFloat("absorptionDistance",absorptionDistance);
+     // controls the falloff curve
+     absorptionPower  = pl.getFloat("absorptionPower",absorptionPower);
+     // sets the distance below the surface for sampling
+     thickness       = pl.getFloat("thickness", thickness);
+     // number of SSS samples
+     samples         = pl.getInt("sssRays", samples);
+     // phong specular color (set to 0,0,0 for no highlights)
+     specular        = pl.getColor("spec", specular);
+     // phong power
+     power           = pl.getFloat("power", power);
+     // phong specular samples
+     phongRays       = pl.getInt("phongRays", phongRays);
+
+     return true;
+ };
+
+ public Color getDiffuse(ShadingState state)
+ {
+     return diff;
+ }
+
+ public Color getRadiance(ShadingState state)
+ {
+     state.faceforward();
+     state.initLightSamples();
+     if(state.getDiffuseDepth()>0)return state.diffuse(getDiffuse(state));
+     Color rColor = getSubRadiance(state,getDiffuse(state));
+     if(power>0f)
+     {
+         state.initLightSamples();
+         rColor = rColor.add(state.specularPhong(specular, power, phongRays));
+     }
+     return rColor;
+ }
+
+ public Color getSubRadiance(ShadingState state, Color dColor)
+ {
+     // collect information about the hit point
+     Color SssPointColor = Color.white();
+     Point3 pt = state.getPoint();
+     Vector3 dir = state.getRay().getDirection();
+     Vector3 norm = state.getNormal();
+     Vector3 rNorm = new Vector3(-norm.x,-norm.y,-norm.z);
+     // set the influence of the hit point
+     // zero should be used under most conditions
+     float fsamp = 1f;

```



```

+   float totalabsorbtion = fsamp;
+   float subdist = 0f;
+   Color c1 = state.diffuse(getDiffuse(state));
+   float[] rgbA = c1.getRGB();
+   float red = rgbA[0]*fsamp;
+   float green = rgbA[1]*fsamp;
+   float blue = rgbA[2]*fsamp;
+   // create a point on the inside of the surface
+   Point3 pt4 = new Point3(pt.x,pt.y,pt.z);
+   pt4.x = pt.x-norm.x*0.0012f;
+   pt4.y = pt.y-norm.y*0.0012f;
+   pt4.z = pt.z-norm.z*0.0012f;
+   // check the thickness
+   Ray thkCheckRay = new Ray(pt4,rNorm);
+   ShadingState thkchk = state.traceFinalGather(thkCheckRay,5);
+   thkchk.init();
+   thkchk.setBasis(state.getBasis());
+   Color na = thkchk.shade();
+   thkchk.faceforward();
+   thkchk.initLightSamples();
+   na = thkchk.diffuse(getDiffuse(thkchk));
+   float thk = thkchk.getRay().getMax()/2f;
+   if(thickness<thk)
+   {
+       thk=thickness;
+   }
+   // move the point along the normal by the thickness
+   pt4.x = pt4.x-norm.x*thk;
+   pt4.y = pt4.y-norm.y*thk;
+   pt4.z = pt4.z-norm.z*thk;
+   // collect random samples
+   for(int s=0; s<samples;s++)
+   {
+       // create a random sampling ray
+       Ray sssRay = new Ray(pt4,randomVector(state, s, samples));
+       ShadingState sss = state.traceFinalGather(sssRay,s);
+       sss.init();
+       sss.setBasis(state.getBasis());
+       float[] colorAtSssPoint={0f,0f,0f};
+       // project the ray and find the intersection color and distance
+       na = sss.shade();
+       sss.faceAway();
+       sss.initLightSamples();
+       Color c2 = sss.diffuse(getDiffuse(sss));
+       float sssRayDistance = sss.getRay().getMax();
+       sssRayDistance = pt4.distanceTo(sss.getPoint());
+       float ssLength = pt.distanceTo(pt4);
+       float actualLength = pt.distanceTo(sss.getPoint());
+       // check to be sure everything is valid
+       if (!c2.isNan() && !c2.isInf())
+       {
+           // evaluate the influence of this point's color based on it's distance from the scattering point
+           subdist = (float)
+           Math.pow(absorptionDistance/(absorptionDistance+(sssRayDistance*sssRayDistance)),absorptionPower);

```

```

+//      subdist = (float)
Math.pow(absorptionDistance/(absorptionDistance+(actualLength*actualLength)),absorptionPower);
+      subdist = MathUtils.clamp((absorptionDistance-actualLength)/absorptionDistance,0.000001f,1f);
+      totalabsorbtion = totalabsorbtion + subdist;
+      float[] rgb = c2.copy().mul(subdist).getRGB();
+      red   = red + rgb[0];
+      green  = green + rgb[1];
+      blue   = blue + rgb[2];
+      }
+      // let us know if something went wrong, the color was either infinite or not a number
+      else System.out.println("failed sss check: if (!c2.isNan() && !c2.isInf())");
+      }
+      // return GREEN on failure, otherwise return the color
+      return totalabsorbtion>0f ? c1.add(new Color(red/totalabsorbtion,green/totalabsorbtion,blue/totalabsorbtion)) :
new Color(0f,1f,0f);
+      }
+
+      private Vector3 randomVector(ShadingState st, int s, int samp)
+      {
+      double r1 = st.getRandom(s, 0, samp);
+      double r2 = st.getRandom(s, 1, samp);
+      return new Vector3
+      (
+      (float) (2 * Math.cos(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+      (float) (2 * Math.sin(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+      (float) (1 - 2 * r2)
+      );
+      }
+
+      /*----- copied from the standard phong shader -----*/
+      public void scatterPhoton(ShadingState state, Color power) {
+      // photon scattering code by ma0mpt in Sunflow forum thread:
+      // http://sunflow.sourceforge.net/phpbb2/viewtopic.php?t=444&start=0&postdays=0&postorder=asc&highlight=
+      +      Color diffuse = getDiffuse(state);
+      +      state.storePhoton(state.getRay().getDirection(), power, diffuse);
+      +      state.traceReflectionPhoton(new Ray(state.getPoint(), randomVector(state,1,1)), power.mul(1f));
+      +      }
+      }

```

Index: src/org/sunflow/core/shader/SSS_Solid_Shader.java

```

=====
--- src/org/sunflow/core/shader/SSS_Solid_Shader.java (revision 0)
+++ src/org/sunflow/core/shader/SSS_Solid_Shader.java (revision 0)
@@ -0,0 +1,430 @@
+package org.sunflow.core.shader;
+
+import java.io.*;
+import org.sunflow.core.tesselatable.*;
+import org.sunflow.SunflowAPI;
+import org.sunflow.core.*;
+import org.sunflow.core.camera.*;
+import org.sunflow.core.primitive.*;
+import org.sunflow.core.shader.*;
+import org.sunflow.image.Color;

```

```

+import org.sunflow.math.*;
+import org.sunflow.core.LightSource;
+import org.sunflow.core.light.*;
+import org.sunflow.core.LightSample;
+import org.sunflow.core.ParameterList;
+import org.sunflow.core.Ray;
+import org.sunflow.core.Shader;
+import org.sunflow.core.ShadingState;
+import org.sunflow.core.primitive.TriangleMesh;
+import org.sunflow.math.MathUtils;
+import org.sunflow.math.OrthoNormalBasis;
+import org.sunflow.math.Point3;
+import org.sunflow.math.Vector3;
+import org.sunflow.system.UI;
+import org.sunflow.system.UI.Module;
+
+public class SSS_Solid_Shader implements Shader
+ {
+     private float thickness;
+     private float absorptionDistance;
+     private float opacity;
+     private float diffusion;
+     private float f0; // fresnel normal incidence
+     private int samples;
+     private Color diff;
+     private Color specular;
+     private float power;
+     private int phongRays;
+     boolean solidShader=false;
+     boolean volumeShader=false;
+     int diffTraceDepth;
+     private float maxOpacity;
+     int maxRefractionDepth;
+     float eta;
+     ParameterList parameterList;
+     boolean prntTraceNFO = false;
+     float sampDist = 1.0f;
+     int refrDepth = 0;
+     int difDepth = 0;
+     int reflDepth = 0;
+     int sssTraceDepth;
+     float environment;
+     float maxSubRayLength;
+     float amtTrshld;
+     float diffusionSign = 1.0f;
+
+     public SSS_Solid_Shader() {
+         thickness = 0.001f;
+         absorptionDistance = 0.001f;
+         opacity = 1f;
+         samples = 10;
+         diff = new Color(0.3f,0.3f,0.3f);;
+         specular = new Color(0.3f,0.3f,0.3f);;
+         power = 100f;

```

```

+ phongRays = 10;
+ diffTraceDepth = 0;
+ maxRefractionDepth = 0;
+ eta = 1.333f;
+ diffusion = 0.8f;
+ f0 = (1 - eta) / (1 + eta);
+ f0 = f0 * f0;
+ sampDist = 1f;
+ maxOpacity      = 0.998f;
+ environment = 0.01f;
+ maxSubRayLength = 0.9f;
+ sssTraceDepth = 1;
+
+ }
+ public boolean update(ParameterList pl, SunflowAPI api)
+ {
+     parameterList      = pl;
+     // the color of the material if not textured
+     diff                = pl.getColor("diffuse", diff);
+     // sets the amount of opacity per the thickness distance
+     opacity             = pl.getFloat("opacity", opacity);
+     // sets the distance below the surface for sampling
+     thickness           = pl.getFloat("thickness", thickness);
+     // number of SSS samples
+     samples             = pl.getInt("sssRays", samples);
+     // phong specular color (set to 0,0,0 for no highlights)
+     specular            = pl.getColor("spec", specular);
+     // phong power
+     power              = pl.getFloat("power", power);
+     // phong specular samples
+     phongRays          = pl.getInt("phongRays", phongRays);
+     //api.parameter("depths.diffuse", p.getNextInt());
+     maxOpacity          = pl.getFloat("maxOpacity", maxOpacity);
+
+     eta                = pl.getFloat("eta", eta);
+
+     environment         = pl.getFloat("environment", environment);
+
+     maxSubRayLength     = pl.getFloat("maxSubRayLength", maxSubRayLength);
+
+     diffusion           = pl.getFloat("diffusion", diffusion);
+     if(diffusion<0f){diffusionSign = -1.0f;};
+     diffusion = MathUtils.clamp(Math.abs(diffusion),0.00001f,0.99999f);
+
+     sampDist           = pl.getFloat("sampDist", sampDist);
+     sssTraceDepth      = pl.getInt("sssTraceDepth", sssTraceDepth);
+     amtTrshld          = thickness*maxSubRayLength*maxSubRayLength;
+
+     if(sampDist==0 || sampDist>=1f || sampDist<=-1f)
+     {
+         UI.printError(Module.API, "SSS error: sampDist==0 or sampDist>=1f or sampDist<=-1f");
+         UI.printError(Module.API, "SSS error: Setting sampDist to 0.999f");
+     }
+ }

```

```

+         sampDist = 0.999f;
+     }
+     //sampDist = MathUtils.clamp(sampDist, 0.0001f, 0.9999f);
+     //UI.printError(Module.API, "Unable to declare options \"%s\", name is already in use", name);
+     return true;
+ };
+
+ public Color getDiffuse(ShadingState state)
+ {
+     return diff;
+ }
+
+ public Color getPointColor(ShadingState state, Point3 pt )
+ {
+     return diff;
+ }
+
+ public Color getRadiance(ShadingState state)
+ {
+     if (!state.includeSpecular())
+     {
+         return new Color(0.3f,0.6f,1f);
+     }
+     if(state.getDepth()>=maxSubRayLength)
+     {
+         state.faceforward();
+         state.initLightSamples();
+         return state.diffuse(getDiffuse(state));
+     }
+     Vector3 reflDir = new Vector3();
+     Vector3 refractionDir = new Vector3();
+     Vector3 dir = state.getRay().getDirection();
+     Vector3 norm = state.getNormal();
+     Vector3 rNorm = new Vector3(-norm.x,-norm.y,-norm.z);
+
+     state.faceforward();
+     float cos = state.getCosND();
+     boolean inside = state.isBehind();
+     float neta = inside ? eta : 1.0f / eta;
+
+     float dn = 2 * cos;
+     reflDir.x = (dn * state.getNormal().x) + state.getRay().getDirection().x;
+     reflDir.y = (dn * state.getNormal().y) + state.getRay().getDirection().y;
+     reflDir.z = (dn * state.getNormal().z) + state.getRay().getDirection().z;
+
+     // refracted ray
+     float arg = 1 - (neta * neta * (1 - (cos * cos)));
+     boolean tir = arg < 0;
+     if (tir)
+     {
+         return state.diffuse(getDiffuse(state));
+     }
+     else {
+         float nK = (neta * cos) - (float) Math.sqrt(arg);

```

```

+     refractionDir.x = (neta * state.getRay().dx) + (nK * state.getNormal().x);
+     refractionDir.y = (neta * state.getRay().dy) + (nK * state.getNormal().y);
+     refractionDir.z = (neta * state.getRay().dz) + (nK * state.getNormal().z);
+ }
+ refractionDir.normalize();
+
+ Color rColor = getSubRadiance(state,refractionDir);
+ state.setLightsInclude();
+ if(power>0f)
+ {
+     state.initLightSamples();
+     rColor = rColor.add(state.specularPhong(specular, power, phongRays));
+ }
+
+ return rColor;
+ }
+
+ public Color getSubRadiance(ShadingState state, Vector3 rDir)
+ {
+     state.setLightsExclude();
+ // collect information about the hit point
+     Vector3 refrDir = new Vector3(rDir);
+     float workOpacity = opacity;
+     float sssRayDistance = 0f;
+     float amount = 0f;
+     float rayLength = 0f;
+     float inOpacity = 0f;
+     Point3 pt = state.getPoint();
+     Point3 curPt = new Point3(pt.x,pt.y,pt.z);
+     Point3 nxtPt = new Point3(pt.x,pt.y,pt.z) ;
+     Point3 ep = new Point3(pt.x,pt.y,pt.z);
+     Vector3 dir = state.getRay().getDirection();
+     Vector3 norm = state.getNormal();
+     Vector3 rNorm = new Vector3(-norm.x,-norm.y,-norm.z);
+     dir.normalize();
+     norm.normalize();
+     rNorm.normalize();
+
+     TransparencyInfo tinf;
+
+     Color na = Color.white();// use this color to perform operations where the return color is not needed
+     Color workingColor = Color.white();
+     Color startColor = state.diffuse(getPointColor(state,pt));
+     Color ptcCol = Color.white();
+     Color eColor = Color.white();
+     Color sssTrnspry = Color.white();
+
+     float fsamp = 0f;
+     float totalabsorbtion = fsamp;
+     float[] rgbA = startColor.getRGB();
+     float[] rgb = startColor.getRGB();
+     float red = rgbA[0]*fsamp;
+     float green = rgbA[1]*fsamp;

```

```

+     float blue = rgbA[2]*fsamp;
+
+ // create a point on the inside of the surface
+     Point3 pt4 = new Point3(pt.x,pt.y,pt.z);
+     pt4.x = pt.x+rNorm.x*0.002f;
+     pt4.y = pt.y+rNorm.y*0.002f;
+     pt4.z = pt.z+rNorm.z*0.002f;
+
+ //     pt4.x = pt.x+refrDir.x*0.002f;
+ //     pt4.y = pt.y+refrDir.y*0.002f;
+ //     pt4.z = pt.z+refrDir.z*0.002f;
+
+
+ // check the thickness
+     Ray thkCheckRay = new Ray(pt4,refrDir);
+     ShadingState thkchk = state.traceFinalGather(thkCheckRay,5);
+     thkchk.init();
+     thkchk.setBasis(state.getBasis());
+     na = thkchk.shade();
+     thkchk.faceforward();
+     thkchk.initLightSamples();
+     na = thkchk.diffuse(getPointColor(thkchk,pt));
+     float thk = thkchk.getRay().getMax()/2f;
+     if(thickness<thk)
+     {
+         thk=thickness;
+     }
+     else
+     {
+         workOpacity = opacity * thk/thickness;
+     }
+
+ // create a point along the reverse normal, offset from the surface by the thickness
+     pt4.x = pt4.x+refrDir.x*thk;
+     pt4.y = pt4.y+refrDir.y*thk;
+     pt4.z = pt4.z+refrDir.z*thk;
+ // offset the point along the sampling vector
+
+
+     float invdiffusion = 1f-diffusion;
+     Vector3 turVec = new Vector3(refrDir).mul(invdiffusion);
+     turVec.mul(diffusionSign);
+     // collect random samples
+     for(int s=0; s<samples;s++)
+     {
+         if(prntTraceNFO)System.out.print("\nvalue of s: " + s+"(");
+ // create a random sampling ray
+         Vector3 countVec; // = new Vector3(refrDir.x,refrDir.y,refrDir.z).mul(thk);
+         ShadingState sss;
+         Vector3 sssVec = randomVector(state, s, samples);
+         sssVec.normalize();
+         sssVec = sssVec.mul(diffusion);
+         sssVec.x = sssVec.x+turVec.x;
+         sssVec.y = sssVec.y+turVec.y;
+         sssVec.z = sssVec.z+turVec.z;
+         sssVec.normalize();

```

```

+ Ray sssRay = new Ray(pt4,sssVec);
+ sss = state.traceFinalGather(sssRay,s);
+ sss.init();
+ sss.setBasis(state.getBasis());
+ // project the ray and find the intersection color
+ sss.shade();
+ sss.faceAway();
+ sss.initLightSamples();
+ eColor = sss.diffuse(getPointColor(sss,sss.getPoint())) ;
+ // check to be sure everything is valid so far
+ if (!eColor.isNan() && !eColor.isInf())
+ {
+     ep = sss.getPoint();
+     ep.x += sssVec.x*0.002f;
+     ep.y += sssVec.y*0.002f;
+     ep.z += sssVec.z*0.002f;
+
+     sssRayDistance = pt4.distanceTo(ep);
+     amount = amtTrshld/(amtTrshld+sssRayDistance*sssRayDistance);
+
+     if(amount>thickness)
+     {
+         amount = (amount-thickness)/(1.0f-thickness);
+         if (!state.includeSpecular())return new Color(1f,0.3f,0.75f);
+         Ray trnspsyRay = new Ray(ep,sssVec);
+         sssTrnspsy = state.traceRefraction(trnspsyRay, 1).copy();
+         //sssTrnspsy.add(eColor);
+         sssTrnspsy.mul(getPointColor(sss,ep));
+
+         inOpacity = opacity;
+         workOpacity = opacity;
+         inOpacity = inOpacity*thk/thickness;
+         workOpacity = workOpacity*thk/thickness;
+         // inOpacity = inOpacity*sampDist/thickness;
+         // workOpacity = workOpacity*sampDist/thickness;
+         countVec = new Vector3(sssVec);
+         countVec.mul(sampDist);
+         //countVec = new Vector3(sssVec.x*thk,sssVec.y*thk,sssVec.z*thk);
+         curPt = new Point3(pt4.x,pt4.y,pt4.z);
+
+         state.faceforward();
+         state.initLightSamples();
+         workingColor=state.diffuse(getPointColor(state, curPt));
+
+         //tinf = calculateTransparency(startColor.copy(),workingColor.copy(),inOpacity,workOpacity);
+         //workingColor = tinf.returnColor.copy();
+         //inOpacity = tinf.returnOpacity;
+
+         float progress = pt4.distanceTo(curPt);
+
+         int counter = 0;
+         while(sssRayDistance-sampDist > progress && inOpacity<maxOpacity)// &&
counter<maxSamplesPerRay)
+         {

```



```

+         if(prntTraceNFO)System.out.print(counter++ + "/" + inOpacity + ", ");
+         //         float depthAbs = 1f-(curPt.distanceTo(ep)/sssRayDistance);
+         float depthAbs = (thk/curPt.distanceTo(ep));
+         ptcCol = getPointColor(state, curPt).mul(eColor.copy());
+         ptcCol.mul(depthAbs);
+         tinf = calculateTransparency(workingColor.copy(),ptcCol.copy(),inOpacity,workOpacity);
+         workingColor = tinf.returnColor.copy();
+         inOpacity = tinf.returnOpacity;
+
+         curPt.x += countVec.x;
+         curPt.y += countVec.y;
+         curPt.z += countVec.z;
+         progress = pt4.distanceTo(curPt);
+         if(prntTraceNFO)System.out.println("sssRayDistance= " + sssRayDistance + "\tprogress= " +
progress);
+         };
+
+         if(inOpacity<maxOpacity)
+         {
+             tinf = calculateTransparency(workingColor.copy(),eColor.copy(),inOpacity,workOpacity);
+             workingColor = tinf.returnColor.copy();
+             inOpacity = tinf.returnOpacity;
+             tinf = calculateTransparency(workingColor.copy(),sssTrnspry.copy(),inOpacity,environment);
+             workingColor = tinf.returnColor.copy();
+         }
+
+         if (!workingColor.isNan() && !workingColor.isInf())
+         {
+             rgb = workingColor.copy().getRGB();
+             //rgb = sssTrnspry.copy().getRGB();
+             //rgb = eColor.copy().getRGB();
+             totalabsorbtion += amount;
+             red += rgb[0]*amount;
+             green += rgb[1]*amount;
+             blue += rgb[2]*amount;
+         }
+         //else System.out.println("failed sss check: if (!workingColor.isNan() && !workingColor.isInf())");
+         }
+     }
+     // let us know if something went wrong, the color was either infinite or not a number
+     //else System.out.println("failed sss check: if (!eColor.isNan() && !eColor.isInf())");
+     if(prntTraceNFO)System.out.print("");
+     }
+     // divide the results by the number of valid samples for a useable Color
+     //return totalabsorbtion==0f ? new Color(red,green,blue) : new Color(red/totalabsorbtion, green/totalabsorbtion,
blue/totalabsorbtion);
+     return totalabsorbtion>0f ? new Color(red/totalabsorbtion, green/totalabsorbtion, blue/totalabsorbtion):
Color.black();
+     }
+
+     private Vector3 randomVector(ShadingState st, int s, int samp)
+     {
+         double r1 = st.getRandom(s, 0, samp);
+         double r2 = st.getRandom(s, 1, samp);

```

```

+   return new Vector3
+   (
+       (float) (2 * Math.cos(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+       (float) (2 * Math.sin(Math.PI * 2 * r1) * Math.sqrt(r2 * (1 - r2))),
+       (float) (1 - 2 * r2)
+   );
+ }
+
+ public void scatterPhoton(ShadingState state, Color power) {
+ }
+
+ /*
+  * I don't remember where I found this math, but it works
+  * so you probably don't want to mess with it
+  */
+ private TransparencyInfo calculateTransparency(Color inColor, Color voxelColor, float inOpacity, float
+ voxelOpacity)
+ {
+     float TIOpacityPrime=0f;
+     Color inColorPrime = new Color(0.2f,0.5f,0.1f);
+     Color voxColorPrime = new Color(0.1f,0.5f,0.3f);
+     Color returnColor = new Color(0.5f,0.3f,0.1f);
+     Color returnColorPrime = new Color(0.3f,0.1f,0.5f);
+
+     inColorPrime = inColor.mul(inOpacity);
+     voxColorPrime = voxelColor.mul(voxelOpacity);
+     TIOpacityPrime=inOpacity+voxelOpacity*(1f-inOpacity);
+
+     returnColorPrime = voxColorPrime.mul((1f-inOpacity));
+     returnColorPrime.add(inColorPrime);
+     returnColor = returnColorPrime.mul(1f/TIOpacityPrime);
+
+     return new TransparencyInfo(returnColor,TIOpacityPrime);
+ }
+ }
+
+ // Just a data class in case the colors used don't have an Alpha channel
+ class TransparencyInfo
+ {
+     public Color returnColor;
+     public float returnOpacity;
+     TransparencyInfo(Color rC, float rO)
+     {
+         returnColor = rC;
+         returnOpacity = rO;
+     }
+ }

```

Index: src/org/sunflow/core/shader/TexturedDiffuseShader.java

```

=====
--- src/org/sunflow/core/shader/TexturedDiffuseShader.java    (revision 391)
+++ src/org/sunflow/core/shader/TexturedDiffuseShader.java    (working copy)
@@ -2,13 +2,28 @@

```

```
import org.sunflow.SunflowAPI;
```

```

import org.sunflow.core.ParameterList;
+import org.sunflow.core.ParameterList.FloatParameter;
import org.sunflow.core.ShadingState;
import org.sunflow.core.Texture;
+import org.sunflow.core.SolidTexture;
import org.sunflow.core.TextureCache;
import org.sunflow.image.Color;
+import org.sunflow.math.Point3;
+import org.sunflow.math.Vector3;

public class TexturedDiffuseShader extends DiffuseShader {
    private Texture tex;
+   boolean solidTex;
+   private int function = 0;
+   private float scale = 50;
+   private float size = 1;
+   Color fill = Color.white();
+   Color veins = Color.black();
+   SolidTexture solidTexture = new SolidTexture();
+   int numColors;
+   float[] colormap;
+   float[] cmapvalues;
+   boolean isColorMapped = false;

    public TexturedDiffuseShader() {
        tex = null;
@@ -17,13 +32,35 @@
        @Override
        public boolean update(ParameterList pl, SunflowAPI api) {
            String filename = pl.getString("texture", null);
-            if (filename != null)
+            fill = pl.getColor("fill",Color.white());
+            veins = pl.getColor("veins",Color.black());
+            function = pl.getInt("function", function);
+            size = pl.getFloat("size", size);
+            scale = pl.getFloat("scale", scale);
+            FloatParameter cmp = pl.getFloatArray("colormap");
+            FloatParameter cmv = pl.getFloatArray("cmapvalues");
+            if(cmp != null && cmv != null)
+            {
+                colormap = cmp.data;
+                cmapvalues = cmv.data;
+                isColorMapped = true;
+                //UI.printlnInfo(Module.USER, " isColorMapped = true");
+            }
+            else if (filename != null)
+            {
                tex = TextureCache.getTexture(api.resolveTextureFilename(filename), false);
-            return tex != null && super.update(pl, api);
+            }
+            return super.update(pl, api);
        }
    }

    @Override

```

```

    public Color getDiffuse(ShadingState state) {
+       if(isColorMapped)return getPointColor(state,state.getPoint());
        return tex.getPixel(state.getUV().x, state.getUV().y);
    }
+
+   public Color getPointColor(ShadingState state, Point3 pt) {
+       if(isColorMapped)return solidTexture.getPointColor(state,colormap, cmapvalues,pt,function,size,scale);
+       return solidTexture.getColor(state,fill,veins,function,size,scale);
+   }
+ }
}

```

\ No newline at end of file

Index: src/org/sunflow/core/shader/TexturedPhongShader.java

```

=====
--- src/org/sunflow/core/shader/TexturedPhongShader.java      (revision 391)
+++ src/org/sunflow/core/shader/TexturedPhongShader.java      (working copy)
@@ -2,13 +2,28 @@

```

```

import org.sunflow.SunflowAPI;
import org.sunflow.core.ParameterList;
+import org.sunflow.core.ParameterList.FloatParameter;
import org.sunflow.core.ShadingState;
import org.sunflow.core.Texture;
+import org.sunflow.core.SolidTexture;
import org.sunflow.core.TextureCache;
import org.sunflow.image.Color;
+import org.sunflow.math.Point3;
+import org.sunflow.math.Vector3;

```

```

public class TexturedPhongShader extends PhongShader {
    private Texture tex;
+   boolean solidTex;
+   private int function = 0;
+   private float scale = 50;
+   private float size = 1;
+   Color fill = Color.white();
+   Color veins = Color.black();
+   SolidTexture solidTexture = new SolidTexture();
+   int numColors;
+   float[] colormap;
+   float[] cmapvalues;
+   boolean isColorMapped = false;

    public TexturedPhongShader() {
        tex = null;
@@ -17,13 +32,35 @@
        @Override
        public boolean update(ParameterList pl, SunflowAPI api) {
            String filename = pl.getString("texture", null);
-           if (filename != null)
+           fill = pl.getColor("fill",Color.white());
+           veins = pl.getColor("veins",Color.black());
+           function = pl.getInt("function", function);
+           size = pl.getFloat("size", size);
+           scale = pl.getFloat("scale", scale);

```

```

+   FloatParameter cmp = pl.getFloatArray("colormap");
+   FloatParameter cmv = pl.getFloatArray("cmapvalues");
+   if(cmp != null && cmv != null)
+   {
+       colormap = cmp.data;
+       cmapvalues = cmv.data;
+       isColorMapped = true;
+       //UI.printlnInfo(Module.USER, " isColorMapped = true");
+   }
+   else if (filename != null)
+   {
+       tex = TextureCache.getTexture(api.resolveTextureFilename(filename), false);
-   return tex != null && super.update(pl, api);
+   }
+   return super.update(pl, api);
+   }

@Override
public Color getDiffuse(ShadingState state) {
+   if(isColorMapped)return getPointColor(state,state.getPoint());
+   return tex.getPixel(state.getUV().x, state.getUV().y);
+   }

+
+   public Color getPointColor(ShadingState state, Point3 pt) {
+       if(isColorMapped)return solidTexture.getPointColor(state,colormap, cmapvalues,pt,function,size,scale);
+       return solidTexture.getColor(state,fill,veins,function,size,scale);
+   }
+   }

```

\ No newline at end of file

Index: src/org/sunflow/core/shader/TexturedProceduralSSS_Shader.java

```

-----
--- src/org/sunflow/core/shader/TexturedProceduralSSS_Shader.java    (revision 0)
+++ src/org/sunflow/core/shader/TexturedProceduralSSS_Shader.java    (revision 0)
@@ -0,0 +1,59 @@
+package org.sunflow.core.shader;
+
+import org.sunflow.SunflowAPI;
+import org.sunflow.core.ParameterList;
+import org.sunflow.core.ParameterList.FloatParameter;
+import org.sunflow.core.ShadingState;
+import org.sunflow.core.SolidTexture;
+import org.sunflow.core.TextureCache;
+import org.sunflow.image.Color;
+import org.sunflow.system.UI;
+import org.sunflow.system.UI.Module;
+
+public class TexturedProceduralSSS_Shader extends SSS_Shader
+ {
+     private int function = 0;
+     private float scale = 50;
+     private float size = 1;
+     Color fill = Color.white();
+     Color veins = Color.black();
+     SolidTexture solidTexture = new SolidTexture();

```

```

+   int numColors;
+   float[] colormap;
+   float[] cmapvalues;
+   float[] alphamap;
+   boolean isColorMapped = false;
+
+   public TexturedProceduralSSS_Shader() {
+       super.solidShader=true;// only needed for the sss shader
+   }
+
+   public boolean update(ParameterList pl, SunflowAPI api) {
+       fill = pl.getColor("fill",Color.white());
+       veins = pl.getColor("veins",Color.black());
+       function = pl.getInt("function", function);
+       size = pl.getFloat("size", size);
+       scale = pl.getFloat("scale", scale);
+       FloatParameter cmp = pl.getFloatArray("colormap");
+       FloatParameter cmv = pl.getFloatArray("cmapvalues");
+       FloatParameter amp = pl.getFloatArray("alphamap");
+       if(cmp != null && cmv != null)
+       {
+           colormap = cmp.data;
+           cmapvalues = cmv.data;
+           isColorMapped = true;
+           //UI.println(Module.USER, " isColorMapped = true");
+           if(amp != null)
+           {
+               alphamap = amp.data;
+               UI.println(Module.USER, " isAlphaMapped = true");
+           }
+       }
+       return super.update(pl, api);
+   }
+   @Override
+   public Color getDiffuse(ShadingState state) {
+       if(isColorMapped)return solidTexture.getColor(state,colormap, cmapvalues,function,size,scale);
+       return solidTexture.getColor(state,fill,veins,function,size,scale);
+   }
+}

```

Index: src/org/sunflow/core/shader/TexturedSSS_Shader.java

```

=====
--- src/org/sunflow/core/shader/TexturedSSS_Shader.java (revision 0)
+++ src/org/sunflow/core/shader/TexturedSSS_Shader.java (revision 0)
@@ -0,0 +1,29 @@
+package org.sunflow.core.shader;
+
+import org.sunflow.SunflowAPI;
+import org.sunflow.core.ParameterList;
+import org.sunflow.core.ShadingState;
+import org.sunflow.core.Texture;
+import org.sunflow.core.TextureCache;
+import org.sunflow.image.Color;
+
+public class TexturedSSS_Shader extends SSS_Shader

```

```

+ {
+   private Texture tex;
+
+   public TexturedSSS_Shader() {
+       tex = null;
+   }
+
+   public boolean update(ParameterList pl, SunflowAPI api) {
+       String filename = pl.getString("texture", null);
+       if (filename != null)
+           tex = TextureCache.getTexture(api.resolveTextureFilename(filename), false);
+       return tex != null && super.update(pl, api);
+   }
+
+   @Override
+   public Color getDiffuse(ShadingState state) {
+       return tex.getPixel(state.getUV().x, state.getUV().y);
+   }
+}

```

Index: src/org/sunflow/core/shader/TexturedVolumetricSSS_Shader.java

```

-----
--- src/org/sunflow/core/shader/TexturedVolumetricSSS_Shader.java    (revision 0)
+++ src/org/sunflow/core/shader/TexturedVolumetricSSS_Shader.java    (revision 0)
@@ -0,0 +1,59 @@
+package org.sunflow.core.shader;
+
+import org.sunflow.SunflowAPI;
+import org.sunflow.core.ParameterList;
+import org.sunflow.core.ParameterList.FloatParameter;
+import org.sunflow.core.ShadingState;
+import org.sunflow.core.SolidTexture;
+import org.sunflow.core.TextureCache;
+import org.sunflow.image.Color;
+import org.sunflow.math.Point3;
+import org.sunflow.math.Vector3;
+import org.sunflow.system.UI;
+import org.sunflow.system.UI.Module;
+
+public class TexturedVolumetricSSS_Shader extends SSS_Solid_Shader
+ {
+   private int function = 0;
+   private float scale = 50;
+   private float size = 1;
+   Color fill = Color.white();
+   Color veins = Color.black();
+   SolidTexture solidTexture = new SolidTexture();
+   int numColors;
+   float[] colormap;
+   float[] cmapvalues;
+   boolean isColorMapped = false;
+
+   public TexturedVolumetricSSS_Shader() {
+       super.volumeShader=true;// only needed for the sss shader
+   }

```

```

+
+ public boolean update(ParameterList pl, SunflowAPI api) {
+     fill = pl.getColor("fill",Color.white());
+     veins = pl.getColor("veins",Color.black());
+     function = pl.getInt("function", function);
+     size = pl.getFloat("size", size);
+     scale = pl.getFloat("scale", scale);
+     FloatParameter cmp = pl.getFloatArray("colormap");
+     FloatParameter cmv = pl.getFloatArray("cmapvalues");
+     if(cmp != null && cmv != null)
+     {
+         colormap = cmp.data;
+         cmapvalues = cmv.data;
+         isColorMapped = true;
+         //UI.println(Module.USER, " isColorMapped = true");
+     }
+     return super.update(pl, api);
+ }
+ @Override
+ public Color getDiffuse(ShadingState state) {
+     if(isColorMapped)return getPointColor(state,state.getPoint() );
+     return solidTexture.getColor(state,fill,veins,function,size,scale);
+ }
+ @Override
+ public Color getPointColor(ShadingState state, Point3 pt) {
+     if(isColorMapped)return solidTexture.getPointColor(state,colormap, cmapvalues,pt,function,size,scale);
+     return solidTexture.getColor(state,fill,veins,function,size,scale);
+ }
+ }
+}

```

Index: src/org/sunflow/core/ShadingState.java

```

=====
--- src/org/sunflow/core/ShadingState.java    (revision 391)
+++ src/org/sunflow/core/ShadingState.java    (working copy)
@@ -200,6 +200,29 @@
     p.z += bias * ng.z;
 }

+ public final void faceAway() {
+     // make sure we are on the right side of the material
+     if(ng==null)System.out.println("geo Normal = null");
+     if(n==null)System.out.println("Normal = null");
+     if(basis==null)System.out.println("basis = null");
+
+     if (r.dot(ng) > 0) {
+     } else {
+         // this ensure the ray and the geomtric normal are pointing in the
+         // same direction
+         ng.negate();
+         n.negate();
+         basis.flipW();
+         behind = true;
+     }
+     cosND = Math.max(-r.dot(n), 0); // can't be negative
+     // offset the shaded point away from the surface to prevent

```



```

+ // self-intersection errors
+ p.x += 0.001f * ng.x;
+ p.y += 0.001f * ng.y;
+ p.z += 0.001f * ng.z;
+ }
+
+ /**
+  * Get x coordinate of the pixel being shaded.
+  *
+  @@ -430,6 +453,14 @@
+  return includeLights;
+  }
+
+ public final void setLightsExclude() {
+     includeLights=false;
+ }
+
+ public final void setLightsInclude() {
+     includeLights=true;
+ }
+
+ /**
+  * Checks to see if the shader should include specular terms.
+  *
+  @@ -482,6 +513,18 @@
+  return diffuseDepth + reflectionDepth + refractionDepth;
+  }
+
+ public void incrDiffuseDepth()
+ {
+     diffuseDepth++;
+ }
+ public void incrReflectionDepth()
+ {
+     reflectionDepth++;
+ }
+ public void incrRefractionDepth()
+ {
+     refractionDepth++;
+ }
+
+ /**
+  * Get the current diffuse tracing depth. This is the number of diffuse
+  * surfaces reflected from.
+  @@ -673,6 +716,7 @@
+  return server.traceRefraction(this, r, i);
+  }
+
+
+ /**
+  * Trace transparency, this is equivalent to tracing a refraction ray in the
+  * incoming ray direction.

```

Index: src/org/sunflow/core/SolidTexture.java

```

-----
--- src/org/sunflow/core/SolidTexture.java    (revision 0)

```

```

+++ src/org/sunflow/core/SolidTexture.java    (revision 0)
@@ -0,0 +1,179 @@
+package org.sunflow.core;
+
+import org.sunflow.SunflowAPI;
+import org.sunflow.image.Color;
+import org.sunflow.core.ParameterList;
+import org.sunflow.core.ShadingState;
+import org.sunflow.math.OrthonormalBasis;
+import org.sunflow.math.PerlinScalar;
+import org.sunflow.math.Point3;
+import org.sunflow.math.Vector3;
+import org.sunflow.math.MathUtils;
+import org.sunflow.system.UI;
+import org.sunflow.system.UI.Module;
+
+public class SolidTexture {
+    private int function = 0;
+    private float scale = 50;
+    private float size = 1;
+    float[] fill = {0f,1f,0f};
+    float[] veins = {1f,0f,0f};
+    float[] colormap;
+    float[] cmapvalues;
+    float[] alphamap;
+
+    public SolidTexture(){
+    }
+
+    public Color getColor(ShadingState state, float[] cmp, float[] cmv, int fn, float sz, float sc)
+    {
+        colormap=cmp;
+        cmapvalues=cmv;
+        function = fn;
+        size = sz;
+        scale = sc;
+        Color solidTexture = Color.white();
+        float[] evalColor = solidTexture.getRGB();
+        Point3 p = state.transformWorldToObject(state.getPoint());
+        p.x *= size;
+        p.y *= size;
+        p.z *= size;
+        double f0 = f(p.x, p.y, p.z);
+        double fx = f(p.x + .0001, p.y, p.z);
+        double fy = f(p.x, p.y + .0001, p.z);
+        double fz = f(p.x, p.y, p.z + .0001);
+
+        fx=(1.0+(double)scale *(fx - f0) / .0001)/2.0;
+        fy=(1.0+(double)scale *(fy - f0) / .0001)/2.0;
+        fz=(1.0+(double)scale *(fz - f0) / .0001)/2.0;
+
+        fx=Math.abs((double)scale *(fx - f0) / .00005);
+        fy=Math.abs((double)scale *(fy - f0) / .00005);

```

```

+//      fz=Math.abs((double)scale *(fz - f0) / .00005);
+
+      float value = (float)((fx+fy+fz)/3.0);
+      value = MathUtils.clamp(value, 0.0f, 1.0f);
+
+      return interpolateColor(value, colormap, cmapvalues);
+  }
+  public Color getPointColor(ShadingState state, float[] cmp, float[] cmv, Point3 pt, int fn, float sz, float sc)
+  {
+      colormap=cmp;
+      cmapvalues=cmv;
+      function = fn;
+      size = sz;
+      scale = sc;
+      Color solidTexture = Color.white();
+      float[] evalColor = solidTexture.getRGB();
+      Point3 p = state.transformWorldToObject(pt);
+      p.x *= size;
+      p.y *= size;
+      p.z *= size;
+      double f0 = f(p.x, p.y, p.z);
+      double fx = f(p.x + .0001, p.y, p.z);
+      double fy = f(p.x, p.y + .0001, p.z);
+      double fz = f(p.x, p.y, p.z + .0001);
+
+      fx=(1.0+(double)scale *(fx - f0) / .0001)/2.0;
+      fy=(1.0+(double)scale *(fy - f0) / .0001)/2.0;
+      fz=(1.0+(double)scale *(fz - f0) / .0001)/2.0;
+
+      float value = (float)((fx+fy+fz)/3.0);
+      value = MathUtils.clamp(value, 0.0f, 1.0f);
+
+      return interpolateColor(value, colormap, cmapvalues);
+  }
+
+  private Color interpolateColor(float v, float[] cmp, float[] cmv)
+  {
+      int numColors = cmv.length;
+      int start = 0;
+      int finish = 0;
+      float t = 0f;
+      for(int search=0; search<numColors-1;search++)
+      {
+          if(v>=cmv[search]&&v<cmv[search+1])
+          {
+              start = search;
+              finish = search+1;
+              t = cmv[finish]-cmv[start];
+              t = (v - cmv[start])/t;
+          }
+      }
+      //UI.println(Module.USER, "SolidTexture Color Interpolator t-value = " +t);
+      float red  = cmp[start*3]*(1f-t)+cmp[finish*3]*t ;
+      float green = cmp[start*3+1]*(1f-t)+cmp[finish*3+1]*t;

```

```

+     float blue = cmp[start*3+2]*(1f-t)+cmp[finish*3+2]*t;
+
+     return new Color(red,green,blue);
+ }
+
+ public Color getColor(ShadingState state,Color f, Color v, int fn, float sz, float sc)
+ {
+     fill = f.getRGB();
+     veins = v.getRGB();
+     function = fn;
+     size = sz;
+     scale = sc;
+     Color solidTexture = Color.white();
+     float[] evalColor = solidTexture.getRGB();
+     Point3 p = state.transformWorldToObject(state.getPoint());
+     p.x *= size;
+     p.y *= size;
+     p.z *= size;
+     double f0 = f(p.x, p.y, p.z);
+     double fx = f(p.x + .0001, p.y, p.z);
+     double fy = f(p.x, p.y + .0001, p.z);
+     double fz = f(p.x, p.y, p.z + .0001);
+
+     fx=Math.abs((double)scale *(fx - f0) / .0001);
+     fy=Math.abs((double)scale *(fy - f0) / .0001);
+     fz=Math.abs((double)scale *(fz - f0) / .0001);
+
+     float value = (float)((double)(fx+fy+fz)/3.0);
+     value = MathUtils.clamp(value, 0.0f, 1.0f);
+     //0.1/(0.1+(2*ABS(1-0.5-A5))^2)
+     value =1f-(float) (0.1/(0.1+Math.pow((2.0*Math.abs(1.0-0.5-(double)value)),2.0)));
+     //value = value*value*value;
+     //value = 1.0f-value;
+
+     float red = (float)((double)fill[0]*value+(double)veins[0]*(1.0-value));
+     float green = (float)((double)fill[1]*value+(double)veins[1]*(1.0-value));
+     float blue = (float)((double)fill[2]*value+(double)veins[2]*(1.0-value));
+
+     return new Color(red,green,blue).clamp(0f, 1f);
+ }
+
+ double f(double x, double y, double z) {
+     switch (function) {
+     case 0:
+         return .03 * noise(x, y, z, 8);
+     case 1:
+         return .01 * stripes(x + 2 * turbulence(x, y, z, 1), 1.6);
+     default:
+         return -.10 * turbulence(x, y, z, 1);
+     }
+ }
+
+ private static final double stripes(double x, double f) {
+     double t = .5 + .5 * Math.sin(f * 2 * Math.PI * x);

```

```

+     return t * t - .5;
+ }
+
+ private static final double turbulence(double x, double y, double z, double freq) {
+     double t = -.5;
+     for (; freq <= 300 / 12; freq *= 2)
+         t += Math.abs(noise(x, y, z, freq) / freq);
+     return t;
+ }
+
+ private static final double noise(double x, double y, double z, double freq) {
+     double x1, y1, z1;
+     x1 = .707 * x - .707 * z;
+     z1 = .707 * x + .707 * z;
+     y1 = .707 * x1 + .707 * y;
+     x1 = .707 * x1 - .707 * y;
+     return PerlinScalar.snoise((float) (freq * x1 + 100), (float) (freq * y1), (float) (freq * z1));
+ }
+}

```

\ No newline at end of file

Index: src/org/sunflow/image/Color.java

```

=====
--- src/org/sunflow/image/Color.java (revision 391)
+++ src/org/sunflow/image/Color.java (working copy)
@@ -135,7 +135,7 @@
     public final float[] getRGB() {
         return new float[] { r, g, b };
     }
-
+
     public final int toRGB() {
         int ir = (int) (r * 255 + 0.5);
         int ig = (int) (g * 255 + 0.5);

```

Index: src/org/sunflow/PluginRegistry.java

```

=====
--- src/org/sunflow/PluginRegistry.java (revision 391)
+++ src/org/sunflow/PluginRegistry.java (working copy)
@@ -40,6 +40,7 @@
import org.sunflow.core.gi.AmbientOcclusionGIEngine;
import org.sunflow.core.gi.FakeGIEngine;
import org.sunflow.core.gi.InstantGI;
+import org.sunflow.core.gi.FakeGIEngine2;
import org.sunflow.core.gi.IrradianceCacheGIEngine;
import org.sunflow.core.gi.PathTracingGIEngine;
import org.sunflow.core.light.Directionalspotlight;
@@ -91,6 +92,11 @@
import org.sunflow.core.shader.QuickGrayShader;
import org.sunflow.core.shader.ShinyDiffuseShader;
import org.sunflow.core.shader.SimpleShader;
+import org.sunflow.core.shader.SSS_Shader;
+import org.sunflow.core.shader.SSS_Solid_Shader;
+import org.sunflow.core.shader.TexturedSSS_Shader;
+import org.sunflow.core.shader.TexturedProceduralSSS_Shader;
+import org.sunflow.core.shader.TexturedVolumetricSSS_Shader;

```

```

import org.sunflow.core.shader.TexturedAmbientOcclusionShader;
import org.sunflow.core.shader.TexturedDiffuseShader;
import org.sunflow.core.shader.TexturedPhongShader;
@@ -186,6 +192,7 @@
    shaderPlugins.registerPlugin("uber", UberShader.class);
    shaderPlugins.registerPlugin("ward", AnisotropicWardShader.class);
    shaderPlugins.registerPlugin("wireframe", WireframeShader.class);
+    shaderPlugins.registerPlugin("sss", SSS_Shader.class);

    // textured shaders
    shaderPlugins.registerPlugin("textured_ambient_occlusion", TexturedAmbientOcclusionShader.class);
@@ -193,6 +200,11 @@
    shaderPlugins.registerPlugin("textured_phong", TexturedPhongShader.class);
    shaderPlugins.registerPlugin("textured_shiny_diffuse", TexturedShinyDiffuseShader.class);
    shaderPlugins.registerPlugin("textured_ward", TexturedWardShader.class);
+    shaderPlugins.registerPlugin("textured_sss", TexturedSSS_Shader.class);
+
+    // solid textured shaders
+    shaderPlugins.registerPlugin("solid_textured_sss", TexturedProceduralSSS_Shader.class);
+    shaderPlugins.registerPlugin("volume_textured_sss", TexturedVolumetricSSS_Shader.class);

    // preview shaders
    shaderPlugins.registerPlugin("quick_gray", QuickGrayShader.class);
@@ -267,6 +279,7 @@
    // gi engines
    giEnginePlugins.registerPlugin("ambocc", AmbientOcclusionGIEngine.class);
    giEnginePlugins.registerPlugin("fake", FakeGIEngine.class);
+    giEnginePlugins.registerPlugin("fake2", FakeGIEngine2.class);
    giEnginePlugins.registerPlugin("igi", InstantGI.class);
    giEnginePlugins.registerPlugin("irr-cache", IrradianceCacheGIEngine.class);
    giEnginePlugins.registerPlugin("path", PathTracingGIEngine.class);
@@ -308,7 +321,6 @@
    bitmapReaderPlugins.registerPlugin("png", PNGBitmapReader.class);
    bitmapReaderPlugins.registerPlugin("jpg", JPGBitmapReader.class);
    bitmapReaderPlugins.registerPlugin("bmp", BMPBitmapReader.class);
-    bitmapReaderPlugins.registerPlugin("igi", IGIBitmapReader.class);
}

static {

```

Index: src/org/sunflow/core/parser/SCParser.java

```
-----  
--- src/org/sunflow/core/parser/SCParser.java (revision 391)  
+++ src/org/sunflow/core/parser/SCParser.java (working copy)  
@@ -173,6 +173,8 @@
```

```
    api.parameter("aa.contrast", p.getNextFloat());  
    if (p.peekNextToken("filter"))  
        api.parameter("filter", p.getNextToken());  
+    if (p.peekNextToken("filter.size"))  
+        api.parameter("filter.size", p.getNextFloat());  
    if (p.peekNextToken("jitter"))  
        api.parameter("aa.jitter", p.getNextBoolean());  
    if (p.peekNextToken("show-aa")) {
```

Index: src/org/sunflow/core/renderer/BucketRenderer.java

```
-----  
--- src/org/sunflow/core/renderer/BucketRenderer.java (revision 391)  
+++ src/org/sunflow/core/renderer/BucketRenderer.java (working copy)  
@@ -55,6 +55,7 @@
```

```
    // filtering  
    private String filterName;  
+    private float filterSize;  
    private Filter filter;  
    private int fs;  
    private float fhs;  
@@ -65,6 +66,7 @@  
    displayAA = false;  
    contrastThreshold = 0.1f;  
    filterName = "box";  
+    filterSize = 0.0f;  
    jitter = false; // off by default  
    dumpBuckets = false; // for debugging only - not user settable  
}  
@@ -108,6 +110,7 @@  
    thresh = contrastThreshold * (float) Math.pow(2.0f, minAADepth);  
    // read filter settings from scene  
    filterName = options.getString("filter", filterName);  
+    filterSize = options.getFloat("filter.size", filterSize);  
    filter = PluginRegistry.filterPlugins.createObject(filterName);  
    // adjust filter  
    if (filter == null) {  
@@ -115,7 +118,10 @@  
        filter = new BoxFilter();  
        filterName = "box";  
    }  
-    fhs = filter.getSize() * 0.5f;  
+    if (filterSize > 0.0f)  
+        fhs = filterSize * 0.5f;  
+    else  
+        fhs = filter.getSize() * 0.5f;  
    fs = (int) Math.ceil(subPixelSize * (fhs - 0.5f));  
  
    // prepare QMC sampling
```

@@ -133,7 +139,10 @@

```
    UI.printlnInfo(Module.BCKT, " * Subpixel jitter:  %s", useJitter ? "on" : (jitter ? "auto-off" : "off"));
    UI.printlnInfo(Module.BCKT, " * Contrast threshold: %.2f", contrastThreshold);
    UI.printlnInfo(Module.BCKT, " * Filter type:      %s", filterName);
-    UI.printlnInfo(Module.BCKT, " * Filter size:      %.2f pixels", filter.getSize());
+    if (filterSize > 0.0f)
+        UI.printlnInfo(Module.BCKT, " * Filter size:      %.2f pixels", filterSize);
+    else
+        UI.printlnInfo(Module.BCKT, " * Filter size:      %.2f pixels", filter.getSize());
    return true;
}
```


Index: src/org/sunflow/core/parser/SCParser.java

```
=====
--- src/org/sunflow/core/parser/SCParser.java (revision 391)
+++ src/org/sunflow/core/parser/SCParser.java (working copy)
@@ -769,6 +769,9 @@
     } else if (type.equals("cylinder")) {
         UI.printInfo(Module.API, "Reading cylinder ...");
         api.geometry(name, "cylinder");
+    } else if (type.equals("box")) {
+        UI.printInfo(Module.API, "Reading box ...");
+        api.geometry(name, "box");
     } else if (type.equals("banchoff")) {
         UI.printInfo(Module.API, "Reading banchoff ...");
         api.geometry(name, "banchoff");
```

```

shader {
    name triangle_wire
    type janino
<code>
import org.sunflow.core.RenderState;
import org.sunflow.image.Color;
import org.sunflow.math.Vector3;

private Color lineColor = new Color(0.05f, 0.05f, 0.05f);
private Color fillColor = new Color(0.95f, 0.95f, 0.95f);
private float width = 0.02f;

public Color getRadiance(RenderState state) {
    float cos = 1 - (float) Math.pow(1 - Math.abs(Vector3.dot(state.getNormal(), state.getRay().getDirection())), 5);
    float u = state.getU();
    float v = state.getV();
    float w = 1 - u - v;
    return ((u < width || v < width || w < width) ? lineColor : fillColor).copy().mul(cos);
}

public void scatterPhoton(RenderState state, Color power) {
}
</code>
}

override triangle_wire true

```

Index: src/org/sunflow/system/ImagePanel.java

```
=====
--- src/org/sunflow/system/ImagePanel.java    (revision 391)
+++ src/org/sunflow/system/ImagePanel.java    (working copy)
@@ -5,6 +5,7 @@
import java.awt.event.InputEvent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseWheelEvent;
+import java.awt.event.MouseWheelListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
@@ -26,7 +27,7 @@
    private float w, h;
    private long repaintCounter;

-   private class ScrollZoomListener extends MouseInputAdapter {
+   private class ScrollZoomListener extends MouseInputAdapter implements MouseWheelListener {
        int mx;
        int my;
        boolean dragging;
@@ -78,7 +79,6 @@
        mouseDragged(e);
    }

-   @Override
    public void mouseWheelMoved(MouseWheelEvent e) {
        zoom(-20 * e.getWheelRotation(), 0);
    }
@@ -259,4 +259,4 @@
    g.drawLine(x0, y1, x0, y0);
    g.drawImage(image, x, y, iw, ih, java.awt.Color.BLACK, this);
}
-}
\ No newline at end of file
+}
```

```

import org.sunflow.core.tesselatable;
import org.sunflow.core.tesselatable.*;
import org.sunflow.SunflowAPI;
import org.sunflow.core.*;
import org.sunflow.core.camera.*;
import org.sunflow.core.primitive.*;
import org.sunflow.core.shader.*;
import org.sunflow.image.Color;
import org.sunflow.math.*;
import org.sunflow.core.LightSource;
import org.sunflow.core.light.*;

// Change settings here
int maxiteration = 10; // total iterations to generate (careful instances are exponential)
float geometryScale = 1f; //scale the initial object before iterating

//define globals
int f = 0;
Matrix4 m = null;
Matrix4 m000 = null;
Matrix4 m001 = null;
String[] shaderString0 = new String[2];
String[] shaderString1 = new String[2];
Point3 blendPoint0 = new Point3(0f,0f, -2f);
Point3 blendPoint1 = new Point3(0f, 0f,4.6f);

public void build()
{
    parse("prcdrl_FS_003.sc"); // create the scene
    m = Matrix4.scale(geometryScale);
        shaderString0[0] = "itemShader0";
        shaderString0[1] = "itemShader1";
        shaderString1[0] = "itemShader2";
        shaderString1[1] = "itemShader3";

    parameter("shaders", shaderString0);
    parameter("shaders", shaderString1);

    shader("blend0", new BlendShader("blend0", shaderString0, blendPoint0, blendPoint1, 0, maxiteration,
Matrix4.scale(1f)));
    parameter("shaders", "blend0");

    shader("blend1", new BlendShader("blend1", shaderString1, blendPoint0, blendPoint1, 0, maxiteration,
Matrix4.scale(1f)));
    parameter("shaders", "blend1");

    String[] shaderString = {"blend0", "blend1"};

    // this is the first instance at the starting point
    parameter("transform", m);
    parameter("shaders", shaderString);
    instance("item.instance_"+f, "item");
    f++;

```

```

float scl = 0.6524214788f;
float[] matr0 = {
    0.5831704041f, 0.01573119195f, 0.2978083435f, 0f,
    -0.02100930122f, 0.654630097f, 0.006560900709f, 0f,
    -0.2974825862f, -0.01539369208f, 0.5833456481f, 0f,
    -3.050048408f, 0.08159457636f, 5.050982106f, 1f
};
m000 = new Matrix4(matr0, false);
float[] matr1 = {
    0.582074188f, -0.1000984229f, -0.2831871207f, 0f,
    0.03180928248f, 0.6346268361f, -0.1589400783f, 0f,
    0.2986683933f, 0.1274915082f, 0.5688304723f, 0f,
    3.032827956f, 0.2877690502f, 5.071354204f, 1f
};
m001 = new Matrix4(matr1, false);
Matrix4[] trnsfrms = {m000,m001}; /* group the transforms to be passed to the iteration method */
iterate(Matrix4.scale(1f),0,trnsfrms,0,shaderString0,shaderString1,blendPoint0,blendPoint1); // call the iteration
method
}

public void iterate(Matrix4 matrix, int itn, Matrix4[] xforms, int fh, String[] shaderString0, String[] shaderString1,
Point3 pt1, Point3 pt2)
{
    itn++;
    Matrix4 m41 = xforms[0];
    Matrix4 m42 = xforms[1];
    m41=m41.multiply(matrix);
    m42=m42.multiply(matrix);
    if(itn<maxiteration)
    {
        f++;
        if(itn<maxiteration)
        {
            m41 = m.multiply(m41);
            Point3 point1 = m41.transformP(pt1);
            Point3 point2 = m41.transformP(pt2);
            String iterShdr0 = "blend0_" + f;// create unique name for this shader
            String iterShdr1 = "blend1_" + f;// create unique name for this shader
            String[] iterShdr = new String[2];
            iterShdr[0]=iterShdr0;
            iterShdr[1]=iterShdr1;
            String inst = "item.instance_" + f;// create unique name for this instance

            shader(iterShdr0, new BlendShader(iterShdr0, shaderString0, new Point3(point1), new Point3(point2), itn,
maxiteration, m41));
            parameter("shaders", iterShdr0);

            shader(iterShdr1, new BlendShader(iterShdr1, shaderString1, new Point3(point1), new Point3(point2), itn,
maxiteration, m41));
            parameter("shaders", iterShdr1);

            parameter("transform", m41);
            parameter("shaders", iterShdr);
            instance(inst, "item");

```

```

        iterate(m41,itn,xforms,0,shaderString0,shaderString1,pt1,pt2);
    };

    f++;
    if(itn<maxiteration)
    {
        m42 = m.multiply(m42);
        Point3 point1 = m42.transformP(pt1);
        Point3 point2 = m42.transformP(pt2);
        String iterShdr0 = "blend0_" + f;// create unique name for this shader
        String iterShdr1 = "blend1_" + f;// create unique name for this shader
        String[] iterShdr = new String[2];
        iterShdr[0]=iterShdr0;
        iterShdr[1]=iterShdr1;
        String inst = "item.instance_" + f;// create unique name for this instance

        shader(iterShdr0, new BlendShader(iterShdr0, shaderString0, new Point3(point1), new Point3(point2), itn,
maxiteration, m42));
        parameter("shaders", iterShdr0);

        shader(iterShdr1, new BlendShader(iterShdr1, shaderString1, new Point3(point1), new Point3(point2), itn,
maxiteration, m42));
        parameter("shaders", iterShdr1);

        parameter("transform", m42);
        parameter("shaders", iterShdr);
        instance(inst, "item");
        iterate(m42,itn,xforms,0,shaderString0,shaderString1,pt1,pt2);
    };
};

};

```

```

public class BlendShader implements Shader // extends janino

```

```

{
    boolean          b1, b2, b3, initflag, updfalg = false;
    String[]          shaderString;
    Point3            point1,point2;
    float             maxiteration,iteration;
    ParameterList      pl;
    SunflowAPI         api;
    String             name;
    Matrix4            matrix;

```

```

    public BlendShader(String nm, String[] shaderStr, Point3 pt1, Point3 pt2, int itn, int mxit, Matrix4 mtx4)
    {
        point1 = new Point3(pt1);
        point2 = new Point3(pt2);
        iteration = (float)itn;
        maxiteration = (float)mxit;
        shaderString = shaderStr;
        name = nm;
        matrix = Matrix4.scale(1f);
        matrix = matrix.multiply(mtx4);
    }

```

```

    }

public boolean update(ParameterList p, SunflowAPI a)
{
    pl=p;
    api=a;
    return true;
}

public Color getRadiance(ShadingState state)
{
    Point3 p = state.getPoint();
    Color base = new Color( api.lookupShader(shaderString[0]).getRadiance(state) );
    state.getPoint().set(p); // restore point
    Color tip = new Color( api.lookupShader(shaderString[1]).getRadiance(state) );
    state.getPoint().set(p); // restore point
    //Color rColor = new Color(0f,0f,0f);

    float iterationOffset = iteration/maxiteration;
    float g = gradient(state);
    float gradientSegment = g/maxiteration;
    float gradientValue = gradientSegment+iterationOffset;
    return Color.blend(base, tip, gradientValue);
}

private float gradient(ShadingState state)
{
    Point3 point3 = new Point3(state.getPoint().x,state.getPoint().y,state.getPoint().z);
    Vector3 gradVec = new Vector3(point2.x-point1.x, point2.y-point1.y, point2.z-point1.z);
    Vector3 ptVec = new Vector3(point3.x-point1.x, point3.y-point1.y, point3.z-point1.z);
    float radians = Vector3.dot(gradVec,ptVec);
    float dist = ptVec.length();
    float gradLen = gradVec.length();
    float t = radians/(gradLen*gradLen);
    // Clamp the value just in case
    if(t>1f)t=1f;
    if(t<0f)t=0f;
    return t;
}
}

// a helper method just in case
public void printMatrix(Matrix4 matrix4, String s)
{
    float[] fa = matrix4.asRowMajor();
    for(int par=0;par<16;par++)
    {
        s = s + fa[par] + " ";
    }
    System.out.println(s);
}

// a helper method just in case
public float radians(float degrees)

```

```
{  
return 3.1416f/180f*degrees;  
};
```



```

import org.sunflow.SunflowAPI;
import org.sunflow.core.*;
import org.sunflow.core.camera.*;
import org.sunflow.core.primitive.*;
import org.sunflow.core.shader.*;
import org.sunflow.image.Color;
import org.sunflow.math.*;

// Change settings here
int depth = 5;
boolean preview = false;

public void build() {
    parameter("eye", new Point3(2.0f, 2.0f, -5.0f));
    parameter("target", new Point3(0, 0, 0));
    parameter("up", new Vector3(0.0f, 1.0f, 0.0f));
    parameter("fov", 45.0f);
    parameter("aspect", 2.0f);
    camera("camera_outside", new PinholeLens());

    parameter("eye", new Point3(0, 0.2f, 0));
    parameter("target", new Point3(-1.0f, -0.5f, 0.5f));
    parameter("up", new Vector3(0.0f, 1.0f, 0.0f));
    camera("camera_inside", new SphericalLens());

    parameter("maxdist", 0.4f);
    parameter("samples", 16);
    shader("ao_sponge", new AmbientOcclusionShader());

    parameter("maxdist", 0.4f);
    parameter("samples", 128);
    shader("ao_ground", new AmbientOcclusionShader());

    geometry("sponge", new MengerSponge(depth));
    // Matrix4 m = null;
    // m = Matrix4.rotateX((float) Math.PI / 3);
    // m = m.multiply(Matrix4.rotateZ((float) Math.PI / 3));
    // parameter("transform", m);
    parameter("shaders", "ao_sponge");
    instance("sponge.instance", "sponge");

    parameter("center", new Point3(0, -1.25f, 0.0f));
    parameter("normal", new Vector3(0.0f, 1.0f, 0.0f));
    geometry("ground", new Plane());
    parameter("shaders", "ao_ground");
    instance("ground.instance", "ground");

    // rendering options
    parameter("camera", "camera_inside");
    // parameter("camera", "camera_outside");
    parameter("resolutionX", 1024);
    parameter("resolutionY", 768);
    if (preview) {

```

```

        parameter("aa.min", 0);
        parameter("aa.max", 1);
        parameter("bucket.order", "spiral");
    } else {
        parameter("aa.min", 1);
        parameter("aa.max", 2);
        parameter("bucket.order", "column");
        parameter("filter", "mitchell");
    }
    options(DEFAULT_OPTIONS);
}

```

```

private static class MengerSponge extends CubeGrid {
    private int depth;

```

```

    MengerSponge(int depth) {
        this.depth = depth;
    }

```

```

    public boolean update(ParameterList pl, SunflowAPI api) {
        int n = 1;
        for (int i = 0; i < depth; i++)
            n *= 3;
        pl.addInteger("resolutionX", n);
        pl.addInteger("resolutionY", n);
        pl.addInteger("resolutionZ", n);
        return super.update(pl, api);
    }

```

```

    protected boolean inside(int x, int y, int z) {
        for (int i = 0; i < depth; i++) {
            if ((x % 3) == 1 ? (y % 3) == 1 || (z % 3) == 1 : (y % 3) == 1 && (z % 3) == 1) return false;
            x /= 3;
            y /= 3;
            z /= 3;
        }
        return true;
    }
}

```

```

package org.sunflow.core.camera;

import org.sunflow.SunflowAPI;
import org.sunflow.core.CameraLens;
import org.sunflow.core.ParameterList;
import org.sunflow.core.Ray;
import org.sunflow.math.Point3;
import org.sunflow.math.Vector3;

public class IsometricLens implements CameraLens {
    private float au, av;
    private float fov, aspect;
    private Point3 eye;

    public boolean update(ParameterList pl, SunflowAPI api) {
        fov = pl.getFloat("fov", fov);
        aspect = pl.getFloat("aspect", aspect);
        eye = pl.getPoint("eye", eye);
        update();
        return true;
    }

    private void update() {
        au = (float) Math.tan(Math.toRadians(fov * 0.5f));
        av = au / aspect;
    }

    public Ray getRay(float x, float y, int imageWidth, int imageHeight, double lensX, double lensY, double time) {
        float du = -au + ((2.0f * au * x) / (imageWidth - 1.0f));
        float dv = -av + ((2.0f * av * y) / (imageHeight - 1.0f));
        return new Ray(du, dv, 0, 0, 0, -1);
    }
}

```

```

package org.sunflow.core.camera;

import org.sunflow.SunflowAPI;
import org.sunflow.core.CameraLens;
import org.sunflow.core.ParameterList;
import org.sunflow.core.Ray;
import org.sunflow.math.Point3;
import org.sunflow.math.Vector3;

public class IsometricLens implements CameraLens {
    private float au, av;
    private float fov, aspect;
    private Point3 eye;

    public boolean update(ParameterList pl, SunflowAPI api) {
        fov = pl.getFloat("fov", fov);
        aspect = pl.getFloat("aspect", aspect);
        eye = pl.getPoint("eye", eye);
        update();
        return true;
    }

    private void update() {
        au = (float) Math.tan(Math.toRadians(fov * 0.5f));
        av = au / aspect;
    }

    public Ray getRay(float x, float y, int imageWidth, int imageHeight, double lensX, double lensY, double time) {
        float du = 0 - au + ((2.0f * au * x) / (imageWidth - 1.0f));
        float dv = 0 - av + ((2.0f * av * y) / (imageHeight - 1.0f));
        Vector3 point = new Vector3(du, dv, -1).normalize();
        float scalar = eye.z / (-1 * point.z);
        return new Ray(scalar * point.x, scalar * point.y, 0, 0, 0, -1);
    }
}

```

```

package org.sunflow.core.primitive;

import org.sunflow.core.BoundedPrimitive;
import org.sunflow.core.IntersectionState;
import org.sunflow.core.Ray;
import org.sunflow.core.Shader;
import org.sunflow.core.ShadingState;
import org.sunflow.math.BoundingBox;
import org.sunflow.math.MathUtils;
import org.sunflow.math.Matrix;
import org.sunflow.math.Matrix4f;
import org.sunflow.math.OrthoNormalBasis;
import org.sunflow.math.Point3;
import org.sunflow.math.Solvers;
import org.sunflow.math.Vector3;

/**
 * @author Matt
 *
 */

class Value4{
    float x, y, z, v;

    public Value4(float x, float y, float z, float v){
        this.v=v;
        this.x=x;
        this.y=y;
        this.z=z;
    }
}

public class IsoCube implements BoundedPrimitive {
    private float minX, minY, minZ;
    private float maxX, maxY, maxZ;

    private float a, b, c, d, e, f, g, h;

    private BoundingBox lightBounds;

    private Shader shader;

    /**
     * @param t
     * @return
     */
    public float getValue(Point3 t){
        return
            a*t.x*t.y*t.z +
            b*t.x*t.y +
            c*t.y*t.z +
            d*t.x*t.z +

```

```

    e*t.x +
    f*t.y+
    g*t.z +
    h;    //    return (( v_0 - v_1 )*(m.x-minX)/(maxX-minX))+v_1;
}

```

```

/**
 * @param m
 * @return
 */
public Vector3 getNormal(Point3 m){
    float vX=a*m.y*m.z + b*m.y + d*m.z + e;
    float vY=a*m.x*m.z + b*m.x + c*m.z + f;
    float vZ=a*m.y*m.x + c*m.y + d*m.x + g;
    return new Vector3(vX, vY, vZ).normalize();
}

```

```

/**
 * @param wheights
 */
public static Vector3 findCoefficients(
    Point3 m,
    Point3 corner0,
    Point3 corner1,
    float wheight000, float wheight001,
    float wheight010, float wheight011,
    float wheight100, float wheight101,
    float wheight110, float wheight111
){

    float minX = Math.min(corner0.x, corner1.x);
    float minY = Math.min(corner0.y, corner1.y);
    float minZ = Math.min(corner0.z, corner1.z);
    float maxX = Math.max(corner0.x, corner1.x);
    float maxY = Math.max(corner0.y, corner1.y);
    float maxZ = Math.max(corner0.z, corner1.z);

    Value4[] s={
        new Value4(minX, minY, minZ, wheight000),
        new Value4(maxX, minY, minZ, wheight100),
        new Value4(minX, maxY, minZ, wheight010),
        new Value4(minX, minY, maxZ, wheight001),
        new Value4(minX, maxY, maxZ, wheight011),
        new Value4(maxX, minY, maxZ, wheight101),
        new Value4(maxX, maxY, minZ, wheight110),
        new Value4(maxX, maxY, maxZ, wheight111),
    };

```

```

double[][] pig=new double[s.length][8];
for (int i=0;i<pig.length;i++){
    pig[i][0]=s[i].x*s[i].y*s[i].z;

```

```

        pig[i][1]=s[i].x*s[i].y;

        pig[i][2]=s[i].y*s[i].z;

        pig[i][3]=s[i].x*s[i].z;

        pig[i][4]=s[i].x;

        pig[i][5]=s[i].y;

        pig[i][6]=s[i].z;

        pig[i][7]=1;
    }
    double [][] bigwist=
    {
        {s[0].v},
        {s[1].v},
        {s[2].v},
        {s[3].v},
        {s[4].v},
        {s[5].v},
        {s[6].v},
        {s[7].v},
    };

    Matrix mpm=new Matrix(pig);
    bigwist=mpm.solve(new Matrix( bigwist)).getArray();

    float a=(float)bigwist[0][0];
    float b=(float)bigwist[1][0];
    float c=(float)bigwist[2][0];
    float d=(float)bigwist[3][0];
    float e=(float)bigwist[4][0];
    float f=(float)bigwist[5][0];
    float g=(float)bigwist[6][0];
    float h=(float)bigwist[7][0];

    float vX=a*m.y*m.z + b*m.y + d*m.z + e;
    float vY=a*m.x*m.z + b*m.x + c*m.z + f;
    float vZ=a*m.y*m.x + c*m.y + d*m.x + g;
    return new Vector3(vX, vY, vZ);
}

/**
 * @param wheights
 */
public void findCoefficients(Wheights wheights){
    Value4[] s={
        new Value4(minX, minY, minZ, wheights.get(0, 0, 0)),
        new Value4(maxX, minY, minZ, wheights.get(1, 0, 0)),
        new Value4(minX, maxY, minZ, wheights.get(0, 1, 0)),
        new Value4(minX, minY, maxZ, wheights.get(0, 0, 1)),
    }
}

```

```

        new Value4(minX, maxY, maxZ, wheights.get(0, 1, 1)),
        new Value4(maxX, minY, maxZ, wheights.get(1, 0, 1)),
        new Value4(maxX, maxY, minZ, wheights.get(1, 1, 0)),
        new Value4(maxX, maxY, maxZ, wheights.get(1, 1, 1)),
};

```

```

double[][] pig=new double[s.length][8];

```

```

for (int i=0;i<pig.length;i++){
    pig[i][0]=s[i].x*s[i].y*s[i].z;

```

```

    pig[i][1]=s[i].x*s[i].y;

```

```

    pig[i][2]=s[i].y*s[i].z;

```

```

    pig[i][3]=s[i].x*s[i].z;

```

```

    pig[i][4]=s[i].x;

```

```

    pig[i][5]=s[i].y;

```

```

    pig[i][6]=s[i].z;

```

```

    pig[i][7]=1;

```

```

}

```

```

double [][] bigwist=

```

```

{

```

```

    {s[0].v},

```

```

    {s[1].v},

```

```

    {s[2].v},

```

```

    {s[3].v},

```

```

    {s[4].v},

```

```

    {s[5].v},

```

```

    {s[6].v},

```

```

    {s[7].v},

```

```

};

```

```

Matrix m=new Matrix(pig);

```

```

bigwist=m.solve(new Matrix( bigwist)).toArray();

```

```

a=(float)bigwist[0][0];

```

```

b=(float)bigwist[1][0];

```

```

c=(float)bigwist[2][0];

```

```

d=(float)bigwist[3][0];

```

```

e=(float)bigwist[4][0];

```

```

f=(float)bigwist[5][0];

```

```

g=(float)bigwist[6][0];

```

```

h=(float)bigwist[7][0];

```

```

}

```

```

/**

```

```

 * @author Matt

```

```

 *

```

```

 */

```

```

private class Wheights{

```



```

Float [] values=new Float[8];

/**
 * @param values
 */
public Wheights(float[] values){
    for (int i=8; --i>=0;){
        this.values[i]=new Float(values[i]);
    }
}

/**
 * @param values
 */
public Wheights(Float[] values){
    this.values=values;
}

/**
 * @param wheight000
 * @param wheight001
 * @param wheight010
 * @param wheight011
 * @param wheight100
 * @param wheight101
 * @param wheight110
 * @param wheight111
 */
public Wheights(
    float wheight000, float wheight001,
    float wheight010, float wheight011,
    float wheight100, float wheight101,
    float wheight110, float wheight111){

    set(wheight000, 0,0,0); set(wheight001, 0,0,1);
    set(wheight010, 0,1,0); set(wheight011, 0,1,1);
    set(wheight100, 1,0,0); set(wheight101, 1,0,1);
    set(wheight110, 1,1,0); set(wheight111, 1,1,1);
    return;
};

/**
 * @param wheight000
 * @param wheight001
 * @param wheight010
 * @param wheight011
 * @param wheight100
 * @param wheight101
 * @param wheight110
 * @param wheight111
 */
public Wheights(
    Float wheight000, Float wheight001,
    Float wheight010, Float wheight011,

```

```

        Float wheight100, Float wheight101,
        Float wheight110, Float wheight111){

    set(wheight000, 0,0,0); set(wheight001, 0,0,1);
    set(wheight010, 0,1,0); set(wheight011, 0,1,1);
    set(wheight100, 1,0,0); set(wheight101, 1,0,1);
    set(wheight110, 1,1,0); set(wheight111, 1,1,1);
    return;
};

/**
 * @param i
 * @param j
 * @param k
 * @return
 */
public int index(int i, int j, int k){ return (i<<2)+(j<<1)+k; };

/**
 * @param value
 * @param i
 * @param j
 * @param k
 */
public void set(Float value, int i, int j, int k){ values[index(i, j, k)]=value; return; };

/**
 * @param value
 * @param i
 */
public void set(Float value, int i){ values[i]=value; return; };

/**
 * @param i
 * @param j
 * @param k
 * @return
 */
public float get( int i, int j, int k){ return values[index(i, j, k)]; };

/**
 * @param i
 * @return
 */
public float get( int i){ return values[i]; };
}

/**
 * @param shader
 * @param corner0
 * @param corner1
 * @param wheight000
 * @param wheight001
 * @param wheight010

```

```

* @param wheight011
* @param wheight100
* @param wheight101
* @param wheight110
* @param wheight111
*/
public IsoCube(
    Shader shader,
    Point3 corner0,
    Point3 corner1,

    Float wheight000, Float wheight001,
    Float wheight010, Float wheight011,
    Float wheight100, Float wheight101,
    Float wheight110, Float wheight111
) {
    // cube extents
    minX = Math.min(corner0.x, corner1.x);
    minY = Math.min(corner0.y, corner1.y);
    minZ = Math.min(corner0.z, corner1.z);
    maxX = Math.max(corner0.x, corner1.x);
    maxY = Math.max(corner0.y, corner1.y);
    maxZ = Math.max(corner0.z, corner1.z);

    lightBounds = new BoundingBox();
    lightBounds.include(new Point3(minX, minY, minZ));
    lightBounds.include(new Point3(maxX, maxY, maxZ));
    lightBounds.enlargeUlps();

    this.shader=shader;
    findCoefficients(new Wheights(
        wheight000, wheight001,
        wheight010, wheight011,
        wheight100, wheight101,
        wheight110, wheight111
    ));
}

/**
* @param shader
* @param corner0
* @param corner1
* @param wheight000
* @param wheight001
* @param wheight010
* @param wheight011
* @param wheight100
* @param wheight101
* @param wheight110
* @param wheight111
*/
public IsoCube(
    Shader shader,
    Point3 corner0,

```

```

        Point3 corner1,

        float wheight000, float wheight001,
        float wheight010, float wheight011,
        float wheight100, float wheight101,
        float wheight110, float wheight111
    ) {
        // cube extents
        minX = Math.min(corner0.x, corner1.x);
        minY = Math.min(corner0.y, corner1.y);
        minZ = Math.min(corner0.z, corner1.z);
        maxX = Math.max(corner0.x, corner1.x);
        maxY = Math.max(corner0.y, corner1.y);
        maxZ = Math.max(corner0.z, corner1.z);

        lightBounds = new BoundingBox();
        lightBounds.include(new Point3(minX, minY, minZ));
        lightBounds.include(new Point3(maxX, maxY, maxZ));
        lightBounds.enlargeUlps();

        this.shader=shader;

        findCoefficients(new Wheights(
            wheight000, wheight001,
            wheight010, wheight011,
            wheight100, wheight101,
            wheight110, wheight111
        ));
    }

/**
 * @param shader
 * @param corner0
 * @param corner1
 * @param wheightvalues
 */
public IsoCube(
    Shader shader,
    Point3 corner0,
    Point3 corner1,
    float[] wheightvalues
) {
    // cube extents
    minX = Math.min(corner0.x, corner1.x);
    minY = Math.min(corner0.y, corner1.y);
    minZ = Math.min(corner0.z, corner1.z);
    maxX = Math.max(corner0.x, corner1.x);
    maxY = Math.max(corner0.y, corner1.y);
    maxZ = Math.max(corner0.z, corner1.z);

    lightBounds = new BoundingBox();
    lightBounds.include(new Point3(minX, minY, minZ));
    lightBounds.include(new Point3(maxX, maxY, maxZ));
    lightBounds.enlargeUlps();

```

```

        this.shader=shader;

        findCoefficients(new Wheights(wheightvalues));
    }

/**
 * @param shader
 * @param corner0
 * @param corner1
 * @param wheightvalues
 */
public IsoCube(
    Shader shader,
    Point3 corner0,
    Point3 corner1,
    Float[] wheightvalues
) {
    // cube extents
    minX = Math.min(corner0.x, corner1.x);
    minY = Math.min(corner0.y, corner1.y);
    minZ = Math.min(corner0.z, corner1.z);
    maxX = Math.max(corner0.x, corner1.x);
    maxY = Math.max(corner0.y, corner1.y);
    maxZ = Math.max(corner0.z, corner1.z);

    lightBounds = new BoundingBox();
    lightBounds.include(new Point3(minX, minY, minZ));
    lightBounds.include(new Point3(maxX, maxY, maxZ));
    lightBounds.enlargeUlps();

    this.shader=shader;

    findCoefficients(new Wheights(wheightvalues));
}

/* (non-Javadoc)
 * @see org.sunflow.core.BoundedPrimitive#getBounds()
 */
public BoundingBox getBounds() {
    return lightBounds;
}

/* (non-Javadoc)
 * @see org.sunflow.core.BoundedPrimitive#intersects(org.sunflow.math.BoundingBox)
 */
public boolean intersects(BoundingBox box) {
    // this could be optimized
    BoundingBox b = new BoundingBox();
    b.include(new Point3(minX, minY, minZ));
    b.include(new Point3(maxX, maxY, maxZ));
    if (b.intersects(box)) {

```

```

        // the box is overlapping or enclosed
        if (!b.contains(new Point3(box.getMinimum().x, box.getMinimum().y, box.getMinimum().z)))
            return true;
        if (!b.contains(new Point3(box.getMinimum().x, box.getMinimum().y, box.getMaximum().z)))
            return true;
        if (!b.contains(new Point3(box.getMinimum().x, box.getMaximum().y, box.getMinimum().z)))
            return true;
        if (!b.contains(new Point3(box.getMinimum().x, box.getMaximum().y, box.getMaximum().z)))
            return true;
        if (!b.contains(new Point3(box.getMaximum().x, box.getMinimum().y, box.getMinimum().z)))
            return true;
        if (!b.contains(new Point3(box.getMaximum().x, box.getMinimum().y, box.getMaximum().z)))
            return true;
        if (!b.contains(new Point3(box.getMaximum().x, box.getMaximum().y, box.getMinimum().z)))
            return true;
        if (!b.contains(new Point3(box.getMaximum().x, box.getMaximum().y, box.getMaximum().z)))
            return true;
        // all vertices of the box are inside - the surface of the box is
        // not intersected
    }
    return false;
}

/* (non-Javadoc)
 * @see org.sunflow.core.Primitive#prepareShadingState(org.sunflow.core.ShadingState)
 */
public void prepareShadingState(ShadingState state) {
    state.init();
    state.getRay().getPoint(state.getPoint());
    state.getNormal().set(this.getNormal(state.getPoint()));
    state.getGeoNormal().set(state.getNormal());
    state.setBasis(OrthoNormalBasis.makeFromW(state.getNormal()));
    state.setShader(shader);
}

/* (non-Javadoc)
 * @see org.sunflow.core.Primitive#intersect(org.sunflow.core.Ray, org.sunflow.core.IntersectionState)
 */
public void intersect(Ray r, IntersectionState state) {
    float intervalMin = Float.NEGATIVE_INFINITY;
    float intervalMax = Float.POSITIVE_INFINITY;
    float orgX = r.oX;
    float invDirX = 1 / r.dX;
    float t1, t2;
    t1 = (minX - orgX) * invDirX;
    t2 = (maxX - orgX) * invDirX;
    if (invDirX > 0) {
        if (t1 > intervalMin) {
            intervalMin = t1;
        }
        if (t2 < intervalMax) {
            intervalMax = t2;
        }
    }
}

```

```

} else {
    if (t2 > intervalMin) {
        intervalMin = t2;
    }
    if (t1 < intervalMax) {
        intervalMax = t1;
    }
}
if (intervalMin > intervalMax) return;
float orgY = r.oy;
float invDirY = 1 / r.dy;
t1 = (minY - orgY) * invDirY;
t2 = (maxY - orgY) * invDirY;
if (invDirY > 0) {
    if (t1 > intervalMin) {
        intervalMin = t1;
    }
    if (t2 < intervalMax) {
        intervalMax = t2;
    }
} else {
    if (t2 > intervalMin) {
        intervalMin = t2;
    }
    if (t1 < intervalMax) {
        intervalMax = t1;
    }
}
if (intervalMin > intervalMax) return;
float orgZ = r.oz;
float invDirZ = 1 / r.dz;
t1 = (minZ - orgZ) * invDirZ; // no front wall
t2 = (maxZ - orgZ) * invDirZ;
if (invDirZ > 0) {
    if (t1 > intervalMin) {
        intervalMin = t1;
    }
    if (t2 < intervalMax) {
        intervalMax = t2;
    }
} else {
    if (t2 > intervalMin) {
        intervalMin = t2;
    }
    if (t1 < intervalMax) {
        intervalMax = t1;
    }
}
if (intervalMin > intervalMax) return;

if (!r.isInsideMin(intervalMax))return;

float dxdy=r.dx*r.dy;
float dydz=r.dy*r.dz;

```

```
float dxdz=r.dx*r.dz;
```

```
float oxoy=r.ox*r.oy;
```

```
float oyoZ=r.oy*r.oZ;
```

```
float oxoz=r.ox*r.oZ;
```

```
float A=a*dxdy*r.dz;
```

```
float B=
```

```
    a*( dydz*r.ox + dxdz*r.oy + dxdy*r.oZ)+
```

```
    b*dxdy + c*dydz + d*dxdz;
```

```
float C=
```

```
    a*(r.dz*oxoy + r.dx*oyoZ + r.dy*oxoz)+
```

```
    b*(r.dx*r.oy + r.dy*r.ox)+
```

```
    c*(r.dy*r.oZ + r.dz*r.oy)+
```

```
    d*(r.dz*r.ox + r.dx*r.oZ)+
```

```
    e*r.dx + f*r.dy + g*r.dz;
```

```
float D=
```

```
    a*oxoy*r.oZ +
```

```
    b*oxoy +
```

```
    c*oyoZ +
```

```
    d*oxoz +
```

```
    e*r.ox +
```

```
    f*r.oy +
```

```
    g*r.oZ +
```

```
    h;
```

```
double[] primeZero=Solvers.solveCubic(A,B,C,D);
```

```
if (primeZero.length==0) return;
```

```
double sZero=Float.POSITIVE_INFINITY;
```

```
for (double i: primeZero){
```

```
    if (i<sZero && r.isInside(i) && i<=intervalMax && i>=intervalMin )sZero=i;
```

```
}
```

```
if (!r.isInside(sZero)) return;
```

```
r.setMax((float)sZero);
```

```
state.setIntersection(this,0, 0);
```

```
return;
```

```
}
```

```
private static final double _2_3=2d/3d;
```

```
private static final double _1_3=1d/3d;
```

```
}
```



```

package org.sunflow.core.tesselatable;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import java.nio.IntBuffer;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.util.ArrayList;

import org.sunflow.SunflowAPI;
import org.sunflow.core.ParameterList;
import org.sunflow.core.PrimitiveList;
import org.sunflow.core.Tesselatable;
import org.sunflow.core.ParameterList.InterpolationType;
import org.sunflow.core.primitive.TriangleMesh;
import org.sunflow.math.BoundingBox;
import org.sunflow.math.Matrix4;
import org.sunflow.math.Point3;
import org.sunflow.math.Vector3;
import org.sunflow.system.Memory;
import org.sunflow.system.UI;
import org.sunflow.system.UI.Module;
import org.sunflow.util.FloatArray;
import org.sunflow.util.IntArray;

public class FileMesh implements Tesselatable {
    public class triinfo
    {
        public int vert;
        public int norm;
        public int uv;
        triinfo(String data)
        {
            vert=0; //obj is 1-indexed not 0-indexed
            norm=0;
            uv=0;

            String[] inf=data.split("/");
            if(inf.length==1)
            {
                vert=Integer.parseInt(inf[0]);
            }
            if(inf.length==2 && data.indexOf("/")<0) // "vert//normal"
            {
                vert=Integer.parseInt(inf[0]);
            }
        }
    }

```

```

        norm=Integer.parseInt(inf[1]);
    }
    else // "vert/uv"
    {
        vert=Integer.parseInt(inf[0]);
        uv=Integer.parseInt(inf[1]);
    }
    if(inf.length==3)
    {
        vert=Integer.parseInt(inf[0]);
        uv=Integer.parseInt(inf[1]);
        norm=Integer.parseInt(inf[2]);
    }
    vert-=1;//"bad" now == -1 and we are 0-index not 1-index
    norm-=1;
    uv-=1;
}

private String filename = null;
private boolean smoothNormals = false;

public BoundingBox getWorldBounds(Matrix4 o2w) {
    // world bounds can't be computed without reading file
    // return null so the mesh will be loaded right away
    return null;
}

public PrimitiveList tessellate() {
    if (filename.endsWith(".ra3")) {
        try {
            UI.printInfo(Module.GEOM, "RA3 - Reading geometry: \"%s\" ...", filename);
            File file = new File(filename);
            FileInputStream stream = new FileInputStream(filename);
            MappedByteBuffer map = stream.getChannel().map(FileChannel.MapMode.READ_ONLY, 0, file.length());
            map.order(ByteOrder.LITTLE_ENDIAN);
            IntBuffer ints = map.asIntBuffer();
            FloatBuffer buffer = map.asFloatBuffer();
            int numVerts = ints.get(0);
            int numTris = ints.get(1);
            UI.printInfo(Module.GEOM, "RA3 - * Reading %d vertices ...", numVerts);
            float[] verts = new float[3 * numVerts];
            for (int i = 0; i < verts.length; i++)
                verts[i] = buffer.get(2 + i);
            UI.printInfo(Module.GEOM, "RA3 - * Reading %d triangles ...", numTris);
            int[] tris = new int[3 * numTris];
            for (int i = 0; i < tris.length; i++)
                tris[i] = ints.get(2 + verts.length + i);
            stream.close();
            UI.printInfo(Module.GEOM, "RA3 - * Creating mesh ...");
            return generate(tris, verts, smoothNormals);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - file not found", filename);
        } catch (IOException e) {

```

```

        e.printStackTrace();
        UI.printError(Module.GEOM, "Unable to read mesh file \"%.s\" - I/O error occurred", filename);
    }
} else if (filename.endsWith(".obj")) {
    int lineNumber = 1;
    try {
        UI.printInfo(Module.GEOM, "OBJ - Reading geometry: \"%.s\" ...", filename);
        FloatArray verts = new FloatArray();
        FloatArray vertsnormal = new FloatArray();
        FloatArray vertsuvs = new FloatArray();
        //IntArray tris = new IntArray();
        ArrayList tris=new ArrayList();
        FileReader file = new FileReader(filename);
        BufferedReader bf = new BufferedReader(file);
        String line;
        while ((line = bf.readLine()) != null) {
            if (line.startsWith("v ")) {
                String[] v = line.split("\\s+");
                verts.add(Float.parseFloat(v[1]));
                verts.add(Float.parseFloat(v[2]));
                verts.add(Float.parseFloat(v[3]));
            }
            else if (line.startsWith("vn"))
            {
                String[] vn=line.split("\\s+");
                vertsnormal.add(Float.parseFloat(vn[1]));
                vertsnormal.add(Float.parseFloat(vn[2]));
                vertsnormal.add(Float.parseFloat(vn[3]));
            }
            else if (line.startsWith("vt"))
            {
                String[] vt=line.split("\\s+");
                vertsuvs.add(Float.parseFloat(vt[1]));
                vertsuvs.add(Float.parseFloat(vt[2]));
            }
        }
        else if (line.startsWith("f")) {
            String[] f = line.split("\\s+");
            if (f.length == 5)
            {
                triinfo v1=new triinfo(f[1]);
                triinfo v2=new triinfo(f[2]);
                triinfo v3=new triinfo(f[3]);
                triinfo v4=new triinfo(f[4]);
                tris.add(v1);
                tris.add(v2);
                tris.add(v3);
                tris.add(v1);
                tris.add(v3);
                tris.add(v4);
            }
            else if (f.length == 4)
            {
                triinfo v1=new triinfo(f[1]);
                triinfo v2=new triinfo(f[2]);
            }
        }
    }
}

```

```

        triinfo v3=new triinfo(f[3]);

        tris.add(v1);
        tris.add(v2);
        tris.add(v3);
    }
}
if (lineNumber % 100000 == 0)
    UI.printInfo(Module.GEOM, "OBJ - * Parsed %7d lines ...", lineNumber);
lineNumber++;
}
file.close();
UI.printInfo(Module.GEOM, "OBJ - * Creating mesh ...");
return generateMine(tris, verts.trim(), vertsuv.trim(), vertsnormal.trim(), smoothNormals);
} catch (FileNotFoundException e) {
    e.printStackTrace();
    UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - file not found", filename);
} catch (NumberFormatException e) {
    e.printStackTrace();
    UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - syntax error at line %d", lineNumber);
} catch (IOException e) {
    e.printStackTrace();
    UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - I/O error occurred", filename);
}
} else if (filename.endsWith(".stl")) {
    try {
        UI.printInfo(Module.GEOM, "STL - Reading geometry: \"%s\" ...", filename);
        FileInputStream file = new FileInputStream(filename);
        DataInputStream stream = new DataInputStream(new BufferedInputStream(file));
        file.skip(80);
        int numTris = getLittleEndianInt(stream.readInt());
        UI.printInfo(Module.GEOM, "STL - * Reading %d triangles ...", numTris);
        long filesize = new File(filename).length();
        if (filesize != (84 + 50 * numTris)) {
            UI.printWarning(Module.GEOM, "STL - Size of file mismatch (expecting %s, found %s)",
Memory.bytesToString(84 + 14 * numTris), Memory.bytesToString(filesize));
            return null;
        }
        int[] tris = new int[3 * numTris];
        float[] verts = new float[9 * numTris];
        for (int i = 0, i3 = 0, index = 0; i < numTris; i++, i3 += 3) {
            // skip normal
            stream.readInt();
            stream.readInt();
            stream.readInt();
            for (int j = 0; j < 3; j++, index += 3) {
                tris[i3 + j] = i3 + j;
                // get xyz
                verts[index + 0] = getLittleEndianFloat(stream.readInt());
                verts[index + 1] = getLittleEndianFloat(stream.readInt());
                verts[index + 2] = getLittleEndianFloat(stream.readInt());
            }
            stream.readShort();
            if ((i + 1) % 100000 == 0)

```

```

        UI.printInfo(Module.GEOM, "STL - * Parsed %7d triangles ...", i + 1);
    }
    file.close();
    // create geometry
    UI.printInfo(Module.GEOM, "STL - * Creating mesh ...");
    if (smoothNormals)
        UI.printWarning(Module.GEOM, "STL - format does not support shared vertices - normal smoothing disabled");
    return generate(tris, verts, false);
} catch (FileNotFoundException e) {
    e.printStackTrace();
    UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - file not found", filename);
} catch (IOException e) {
    e.printStackTrace();
    UI.printError(Module.GEOM, "Unable to read mesh file \"%s\" - I/O error occurred", filename);
}
} else
    UI.printWarning(Module.GEOM, "Unable to read mesh file \"%s\" - unrecognized format", filename);
return null;
}

```

private TriangleMesh generateMine(ArrayList tris, float[] verts, float[] vertsuv, float[] vertsnormal, boolean smoothNormals)

```

{
    ParameterList pl = new ParameterList();
    UI.printInfo(Module.GEOM, "mine - a");
    int numfaces=tris.size()/3;
    float[] objverts=new float[numfaces*3*3];
    float[] objnorm=new float[numfaces*3*3];
    float[] objtex=new float[numfaces*3*2];
    int[] trisarr=new int[numfaces*3];
    UI.printInfo(Module.GEOM, "mine - b");
    for(int i=0;i<tris.size();i++)
    {
        triinfo ti=(triinfo)tris.get(i);
        trisarr[i]=i;
        objverts[i*3]=verts[ti.vert*3];
        objverts[i*3+1]=verts[ti.vert*3+1];
        objverts[i*3+2]=verts[ti.vert*3+2];
        objnorm[i*3]=vertnormal[ti.norm*3];
        objnorm[i*3+1]=vertnormal[ti.norm*3+1];
        objnorm[i*3+2]=vertnormal[ti.norm*3+2];
        objtex[i*2]=vertsuv[ti.uv*2];
        objtex[i*2+1]=vertsuv[ti.uv*2+1];
    }

    UI.printInfo(Module.GEOM, "mine - c");

    pl.addIntegerArray("triangles", trisarr);
    pl.addPoints("points", InterpolationType.VERTEX, objverts);
    pl.addVectors("normals", InterpolationType.VERTEX, objnorm);
    pl.addTexCoords("uvs", InterpolationType.VERTEX, objtex);
    UI.printInfo(Module.GEOM, "mine - d");
    TriangleMesh m = new TriangleMesh();
    if (m.update(pl, null))

```

```

    return m;
    UI.printInfo(Module.GEOM, "mine - e");
return null;
}

private TriangleMesh generate(int[] tris, float[] verts, boolean smoothNormals) {
    ParameterList pl = new ParameterList();
    pl.addIntegerArray("triangles", tris);
    pl.addPoints("points", InterpolationType.VERTEX, verts);
    if (smoothNormals) {
        float[] normals = new float[verts.length]; // filled with 0's
        Point3 p0 = new Point3();
        Point3 p1 = new Point3();
        Point3 p2 = new Point3();
        Vector3 n = new Vector3();
        for (int i3 = 0; i3 < tris.length; i3 += 3) {
            int v0 = tris[i3 + 0];
            int v1 = tris[i3 + 1];
            int v2 = tris[i3 + 2];
            p0.set(verts[3 * v0 + 0], verts[3 * v0 + 1], verts[3 * v0 + 2]);
            p1.set(verts[3 * v1 + 0], verts[3 * v1 + 1], verts[3 * v1 + 2]);
            p2.set(verts[3 * v2 + 0], verts[3 * v2 + 1], verts[3 * v2 + 2]);
            Point3.normal(p0, p1, p2, n); // compute normal
            // add face normal to each vertex
            // note that these are not normalized so this in fact weights
            // each normal by the area of the triangle
            normals[3 * v0 + 0] += n.x;
            normals[3 * v0 + 1] += n.y;
            normals[3 * v0 + 2] += n.z;
            normals[3 * v1 + 0] += n.x;
            normals[3 * v1 + 1] += n.y;
            normals[3 * v1 + 2] += n.z;
            normals[3 * v2 + 0] += n.x;
            normals[3 * v2 + 1] += n.y;
            normals[3 * v2 + 2] += n.z;
        }
        // normalize all the vectors
        for (int i3 = 0; i3 < normals.length; i3 += 3) {
            n.set(normals[i3 + 0], normals[i3 + 1], normals[i3 + 2]);
            n.normalize();
            normals[i3 + 0] = n.x;
            normals[i3 + 1] = n.y;
            normals[i3 + 2] = n.z;
        }
        pl.addVectors("normals", InterpolationType.VERTEX, normals);
    }
    TriangleMesh m = new TriangleMesh();
    if (m.update(pl, null))
        return m;
    // something failed in creating the mesh, the error message will be
    // printed by the mesh itself - no need to repeat it here
    return null;
}

```

```

public boolean update(ParameterList pl, SunflowAPI api) {
    String file = pl.getString("filename", null);
    if (file != null)
        filename = api.resolveIncludeFilename(file);
    smoothNormals = pl.getBoolean("smooth_normals", smoothNormals);
    return filename != null;
}

private int getLittleEndianInt(int i) {
    // input integer has its bytes in big endian byte order
    // swap them around
    return (i >>> 24) | ((i >>> 8) & 0xFF00) | ((i << 8) & 0xFF0000) | (i << 24);
}

private float getLittleEndianFloat(int i) {
    // input integer has its bytes in big endian byte order
    // swap them around and interpret data as floating point
    return Float.intBitsToFloat(getLittleEndianInt(i));
}
}

```

```

import org.sunflow.SunflowAPI;
import org.sunflow.core.*;
import org.sunflow.core.camera.*;
import org.sunflow.core.primitive.*;
import org.sunflow.core.shader.*;
import org.sunflow.image.Color;
import org.sunflow.math.*;

// Change settings here
boolean preview = false;
int gridResolution = 200;

public void build() {
    parameter("eye", new Point3(1.3f, 1.5f, -4.5f));
    parameter("target", new Point3(0, 0, 0));
    parameter("up", new Vector3(0.0f, 1.0f, 0.0f));
    parameter("fov", 35.0f);
    parameter("aspect", 1.33f);
    camera("camera_outside", new PinholeLens());

    parameter("maxdist", 0.4f);
    parameter("samples", 128);
    shader("ao_ground", new AmbientOcclusionShader());

    geometry("isogrid", new CubeIsosurface( gridResolution ));

    parameter("diffuse", new Color( 0.3f, 0.6f, 1.0f ) );
    shader( "shiny", new ShinyDiffuseShader() );

    parameter("shaders", "shiny");
    instance("isogrid.instance", "isogrid");

    parameter("center", new Point3(0, -1.25f, 0.0f));
    parameter("normal", new Vector3(0.0f, 1.0f, 0.0f));
    geometry("ground", new Plane());
    parameter("shaders", "ao_ground");
    instance("ground.instance", "ground");

    // rendering options
    parameter("camera", "camera_outside");
    parameter("resolutionX", 800);
    parameter("resolutionY", 600);
    if (preview) {
        parameter("aa.min", 0);
        parameter("aa.max", 1);
        parameter("bucket.order", "spiral");
    } else {
        parameter("aa.min", 1);
        parameter("aa.max", 2);
        parameter("bucket.order", "spiral");
        parameter("filter", "mitchell");
    }
    options(DEFAULT_OPTIONS);
}

```



```
}
```

```
private static class CubeIsosurface extends CubeGrid {
```

```
    private int resolution;
```

```
    CubeIsosurface( int resolution ) {
        this.resolution = resolution;
    }
}
```

```
    public boolean update(ParameterList pl, SunflowAPI api) {
        int n = resolution;
        pl.addInteger("resolutionX", n);
        pl.addInteger("resolutionY", n);
        pl.addInteger("resolutionZ", n);
        return super.update(pl, api);
    }
}
```

```
    public double function( double x, double y, double z ){
        // the actual function that defines the isosurface
        // choose from the functions defined below, or define your own!
        return function2( x, y, z );
    }
}
```

```
    public double function1( double x, double y, double z ){
        // CSG example - cylinder subtracted from a sphere with surface turbulence

        double f1 = (x*x + y*y + z*z) - 0.25; // sphere
        // add some displacement to the sphere
        f1 += Math.sin( x * 20 + y * 10 + z * 15 ) * 0.02;
        double f2 = (x*x + y*y) - 0.06; // cylinder
        return Math.max( f1, -f2 ); // sphere - cylinder
    }
}
```

```
    public double function2( double x, double y, double z ){
        // icosahedron function taken from k3dsurf

        // scale the function to fit grid range [-0.5, 0.5]
        x *= 11.0; y *= 11.0; z *= 11.0;

        double result = 1.0;
        if( x*x + y*y + z*z < 35 ){
            result = 2 - (Math.cos(x + (1+Math.sqrt(5))/2*y) + Math.cos(x - (1+Math.sqrt(5))/2*y) + Math.cos(y +
(1+Math.sqrt(5))/2*z) + Math.cos(y - (1+Math.sqrt(5))/2*z) + Math.cos(z - (1+Math.sqrt(5))/2*x) + Math.cos(z +
(1+Math.sqrt(5))/2*x));
        }

        return result;
    }
}
```

```
    public double function3( double x, double y, double z ){
        // gyroid function taken from k3dsurf
    }
}
```

```
// scale the function to fit grid range [-0.5, 0.5]
x *= 8.0; y *= 8.0; z *= 8.0;

double result = Math.cos(x) * Math.sin(y) + Math.cos(y) * Math.sin(z) + Math.cos(z) * Math.sin(x);

result = -result + 0.5;

return result;
}

protected boolean inside(int x, int y, int z) {
    return (function( ((double)x / resolution) - 0.5, ((double)y / resolution) - 0.5, ((double)z / resolution) - 0.5 ) < 0.0);
}
}
```

```
import org.sunflow.core.primitive.ParticleSurface;
import org.sunflow.system.Parser;

public void build() {
    parse("your_scene.sc"); // the name of your scene file goes here
    try {
        Parser p = new Parser(resolveIncludeFilename("your_ascii_filename.dat")); // the name of your particle file goes here
        int n = p.getNextInt();
        float[] data = new float[3 * n];
        for (int i = 0; i < data.length; i++)
            data[i] = p.getNextFloat();
        p.close();
        parameter("particles", "point", "vertex", data);
        parameter("num", data.length / 3);
        parameter("radius", 0.1f); // the radius of the particles goes here
        geometry("particle_object_name", new ParticleSurface());
        parameter("shaders", "shader_name"); // replace with the shader name you want to use
        instance("particle_object_name.instance", "particle_object_name");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```