blenderseed Documentation

Release 2.0.0-beta

blenderseed Manual

Jun 10, 2020

Contents

appleseed
open source production rendering

appleseed for **Blender**
DOCUMENTATION

COVER IMAGE **WHITE KITCHEN**
BY **Juan Carlos Gutiérrez Rodríguez**

appleseedhq.net
blender.org

CHAPTER 1

Features

1.1 New in 1.0

- Completely redesigned export and render stage. It now uses the appleseed.python bindings (no more random export files in odd places)
- Interactive rendering
- Adaptive image sampling
- Full OSL materials
- Area lamps
- Support for linked objects and groups
- Archive assemblies
- Post processing stages
- The appleseed renderer is now bundled directly with blenderseed (no more configuration woes)
- Adjustable number of motion segments for camera, object, and deformation blur

1.2 Supported Features

- Pinhole, thin lens (supports physically correct depth of field), orthographic and spherical camera models
- Camera, transformation and deformation motion blur
- OSL shading
- BCD denoiser integration
- Integrated .tx texture converter
- Render results directly into Blender or export scene files (including animations) for later rendering

- AOVs
- Alpha mapping (object based)
- Point, directional, area and sun lamps
- Spot lights (with optional texturing)
- Physical sun/sky model
- Gradient, constant color, mirror ball and latitude-longitude map environment models
- Path tracing and SPPM lighting engines

1.3 Planned Features

- Dynamic OSL script node
- OSL volume rendering (once appleseed supports it)

CHAPTER 2

Installation

Download Download the .zip file of the latest blenderseed release for your platform. appleseed itself is bundled with the addon, so no additional downloads are needed. Note that as of 0.8.0 you must use Blender 2.79 or higher.

Blender 2.8 is not currently supported. We will add support for it once the Python API has been finalized.

Install From within Blender, open the User Preferences (usual hotkey is Ctrl+Alt+U) and navigate to the Addons tab. Click the button that says "Install From File". Using the file dialog, select the .zip file you downloaded, and click "Install From File…"

Alternative manual installation

Extract the blenderseed folder from the .zip file. Move or copy the blenderseed folder to your Blender installation's /scripts/addons directory. The addon relies on being able to find the blenderseed folder in one of a few conspicuous places, so be sure to install the folder under addons or addons_contrib.

Configure If the addon was installed successfully, you will see it among your addons under the "Render" category. Enable the addon, and click the small triangle to the left of the words "Render: appleseed".

Save your user preferences.

Select "appleseed" from the render dropdown selector.

Using Development Versions of blenderseed If you want access to cutting edge features, you can also download directly from the master branch (or any other visible branches). Any downloads will have a suffix of *'-branch name'* that needs to be removed before it will work. Be aware that new or under development features may require an up to date build of appleseed itself, and this is not included with direct branch downloads.[1][2]

Using Development Versions of appleseed While appleseed is packaged with all official releases, external versions of it may be used with blenderseed. To do so, set the following environment variables before launching Blender (Windows only):

- **APPLESEED_PYTHON_PATH:** Set to the Python27 directory in your appleseed build.
- **APPPLESEED_BIN_DIR:** Set to the /bin folder of your appleseed build.

[1] If you are compiling applessed for use with blenderseed, you will need to compile appleseed with the Python 3 bindings enabled. Please see the build instructions.

[2] You must also compile with the same version of Python 3 that is used by your Blender install. Blender 2.79 uses Python 3.5. Current development snapshots and the 2.8 branch use Python 3.7.

Footnotes:

CHAPTER 3

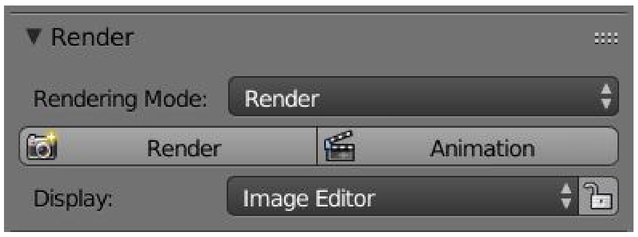
Reference

3.1 Panels

3.1.1 Render Panel

The render panel contains the controls for the rendering process, including sampling, image outputs, denoising and scene export.

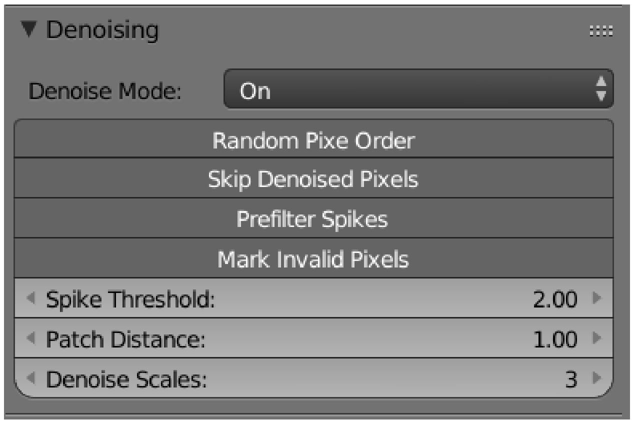
Render



- **Rendering Mode:** Allows you to either render the scene or export it as a series of files for later rendering.
- **Render:** Renders the current frame.
- **Render Animation:** Exports and renders all frames in between the start and end points defined in the dimensions panel.

- **Display:** This defines how the in-progress render is displayed within Blender.

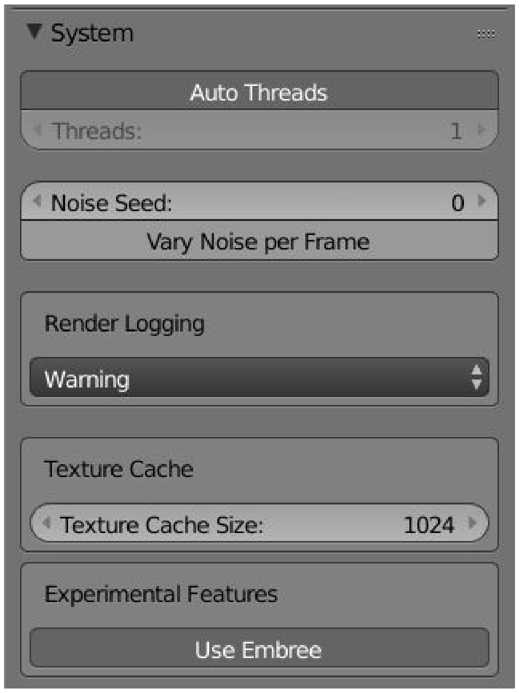
Denoiser



Denoise Mode: On
Spike Threshold: 2.00
Patch Distance: 1.00
Denoise Scales: 3

- **Denoise Mode:**
    - **On:** Denoising will run after a render completes
    - **Write Outputs:** The render will write out several multilayer EXR files that can then be denoised at a later time using the denoise utility. The render itself will not be denoised
- **Random Pixel Order:** TODO
- **Skip Denoised Pixels:** TODO
- **Prefilter Spikes:** This option tries to filter out overly bright pixels (firflies) before denoising
- **Mark Invalid Pixels:** TODO
- **Spike Threshold:** How bright a pixel has to be compared to its neighbors to be considered a spike
- **Patch Distance:** This controls the overall level of denoising that will be applied. Raising this too high can lead to a blurry image
- **Denoise Scales:** This sets how many scale levels are used to remove low frequency noise from the image

General


System

Auto Threads

Threads: 1

Noise Seed: 0

Vary Noise per Frame

Render Logging

Warning

Texture Cache

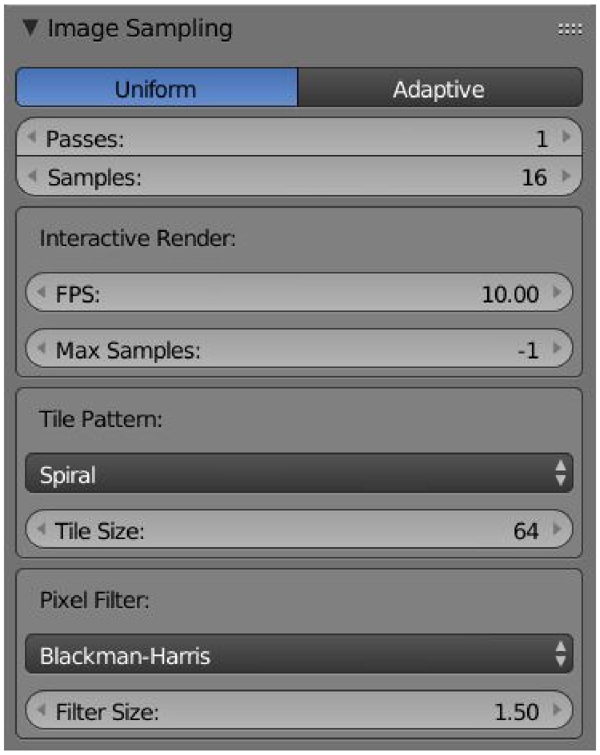
Texture Cache Size: 1024

Experimental Features

Use Embree

- **Auto Threads** Select this option to have appleseed automatically determine the number of rendering threads to use. Unselect it to choose manually.
- **Noise Seed:** This is used to initialize the number generator used for sampling. Changing it will cause different noise patterns in the rendered image.
- **Vary Noise per Frame:** This offsets the noise seed by the current frame number. In animations this causes the noise pattern to vary between frames, mimicking the appearance of film grain.
- **Render Logging** Selects the level of feedback from appleseed during rendering.
- **Texture Cache** Sets the size of the cache used for storing textures. Raising this will increase memory usage but may help speed up rendering.
- **Experimental Features** These features are active in appleseed, but maybe not quite ready for production. Use at your own risk.
  - **Use Embree** Use Intel's Embree raytracing library.

Sampling



Passes: 1
Samples: 16
Interactive Render:
FPS: 10.00
Max Samples: -1
Tile Pattern:
Spiral
Tile Size: 64
Pixel Filter:
Blackman-Harris
Filter Size: 1.50

- **Sampler**
  - Uniform: Every pixel is sampled the same number of times.
  - Adaptive: Sampling is based on whether or not a tile has reached a certain noise threshold.
- **Passes** The number of times the sampler will loop over all the pixels in the image.[1][2]
- **Uniform Sampler**
  - Samples: The number of samples taken per pixel per pass.
- **Adaptive Sampler**
  - Batch Size: The number of pixel samples taken in between noise evaluations.
  - Max Samples: The total number of samples that can be taken per pixel per pass.
  - Noise Threshold: The level of noise that is acceptable in the final image. A pixel will render until it hits this limit or the max samples.
  - Uniform Samples: The number of uniform samples that are taken before adaptive sampling kicks in. This is necessary to ake sure fine details are captured in the image before noise evaluations start.

[1] Setting the passes higher than 1 and lowering the number of samples enables a featured called 'progressive rendering'. The image will appear in its entirety after one pass and noise will be subsequently reduced with each additional pass.

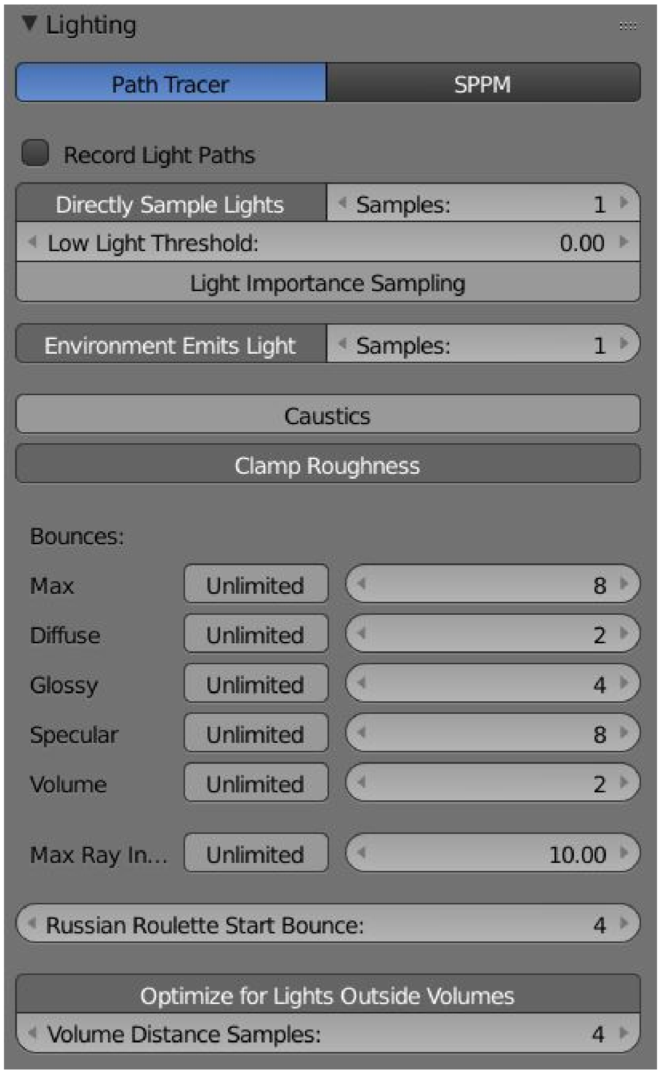
[2] Progressive rendering is slower than single pass rendering, however it does have its advantages. If you are not certain how many samples will be needed to create a clean image, you can set the samples to a low number and the passes number to a higher one. When the image has reached an acceptable level of quality the render can be aborted.

- **Interactive Render**
    - FPS: The maximum framerate of the interactive render session.
    - Max Samples: The number of samples taken before rendering halts in interactive mode.
- **Tile Pattern**
    - Pattern: The order in which tiles are selected during rendering. Pick anything other that random.
    - Tile Size: This is the size of the tiles that the image is broken into for rendering.
- **Pixel Filter:**
    - Filter: This is the type of filter used for image reconstruction.
    - Filter Size: The size of the filter kernel.

Lighting

- **Lighting:** This sets the lighting engine used during the render.

Path Tracing Engine



▼ Lighting

Path Tracer | SPPM

Record Light Paths

Directly Sample Lights | Samples: 1

Low Light Threshold: 0.00

Light Importance Sampling

Environment Emits Light | Samples: 1

Caustics

Clamp Roughness

Bounces:

Max	Unlimited	8
Diffuse	Unlimited	2
Glossy	Unlimited	4
Specular	Unlimited	8
Volume	Unlimited	2
Max Ray In...	Unlimited	10.00

Russian Roulette Start Bounce: 4

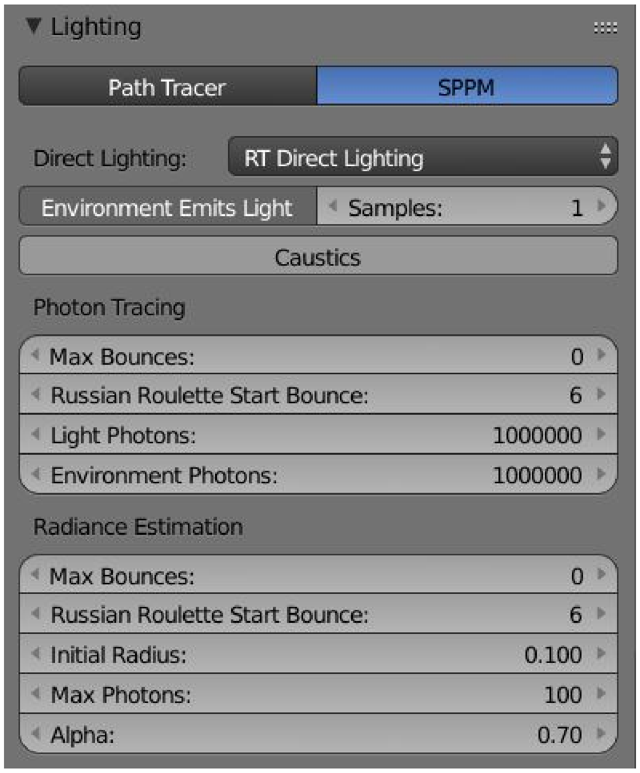
Optimize for Lights Outside Volumes

Volume Distance Samples: 4

- **Record Light Paths:** This instructs appleseed to store information on the path that a light ray travels. It can then be retrieved and visualized in appleseed.studio. See here for a demonstration of this feature.
- **Directly Sample Lights:** Whether or not discreet light sources (point, spotlight and Sun) are directly sampled at each surface hit point.
- **Samples:** How many light paths are sent to the light at every surface hit. Raising this can lower the variance on large area lamps.
- **Low Light Threshold:** This prevents shadow rays from being traced to lights that do not contribute significant illumination to the surface hit point. Higher numbers can lead to a decrease in lighting quality but an increase in convergence speed.
- **Light Importance Sampling:** This control activates importance sampling on rays traced directly to lights. In some cases it may help to reduce noise.
- **Environment Emits Light:** Whether or not an environment map can contribute lighting to the scene.
- **Samples:** How many samples are taken of the environment light at every surface hit.
- **Caustics:** Whether or not caustic effects are rendered.

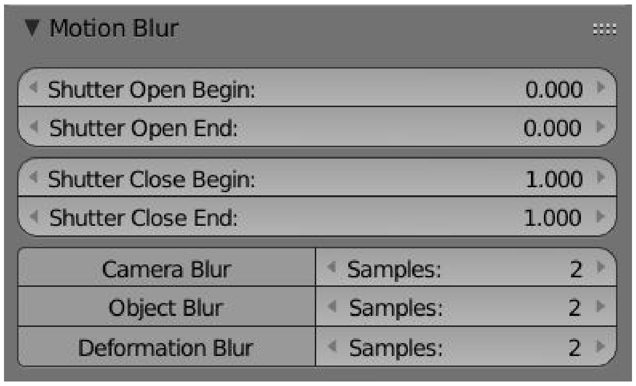
- **Clamp Roughness:** Raises the roughness of materials on hits by indirect rays. This can lower variance for caustics and strong specular reflections.
- **Bounces:**
  - Global: Sets a maximum number of bounces for a light ray.
  - Diffuse: Sets a maximum number of diffuse bounces a ray can make.
  - Glossy: Sets the number of glossy bounces a ray can make.
  - Specular: Sets the number of specular bounces a ray can make.
  - Volume: Sets the maximum number of volume scattering events.
  - Max Ray Intensity: Sets the maximum brightness an indirect ray can contribute to the scene. Lowering this can remove fireflies at the expense of accurate lighting.
- **Russian Roulette Start Bounce:** Sets the bounce after which Russian Roulette will be used to terminate rays that are not expected to contribute much to the final image. Lowering this can increase noise but speed up rendering.
- **Optimize for Lights Outside Volumes:** Use this if lights sources are outside of a volume.

SPPM Engine



Direct Lighting: RT Direct Lighting
Environment Emits Light
Samples: 1
Max Bounces: 0
Russian Roulette Start Bounce: 6
Light Photons: 1000000
Environment Photons: 1000000
Max Bounces: 0
Russian Roulette Start Bounce: 6
Initial Radius: 0.100
Max Photons: 100
Alpha: 0.70

Motion Blur



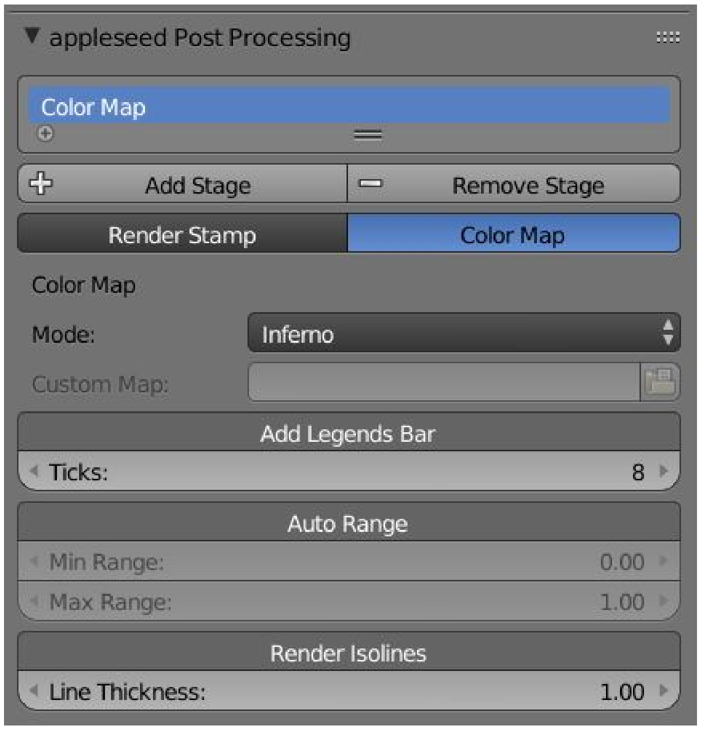
Shutter Open Begin: 0.000
Shutter Open End: 0.000
Shutter Close Begin: 1.000
Shutter Close End: 1.000
Camera Blur | Samples: 2
Object Blur | Samples: 2
Deformation Blur | Samples: 2

- **Shutter Open Begin** This is when the shutter begins to open in relation to the current frame. ‘0’ opens on the beginning of the frame.
- **Shutter Open End** This is when the shutter is fully open
- **Shutter Close Begin** This is when the shutter begins to close in relation to the current frame.[1]
- **Shutter Close End** This is when the shutter is fully closed.
- **Camera Blur** Enables blur caused by motion of the camera.
- **Object Blur** Enables blur caused by the motion (translation, rotation, or scaling) of an object.
- **Deformation Blur** Enables blur caused by an object changing shape (deforming)

Footnotes:

[1] For realistic blur the shutter should close before the end of the frame. ‘0.5’ is equivalent to a 180 degree shutter, which is a common shutter type used in motion picture cameras.

Post Processing

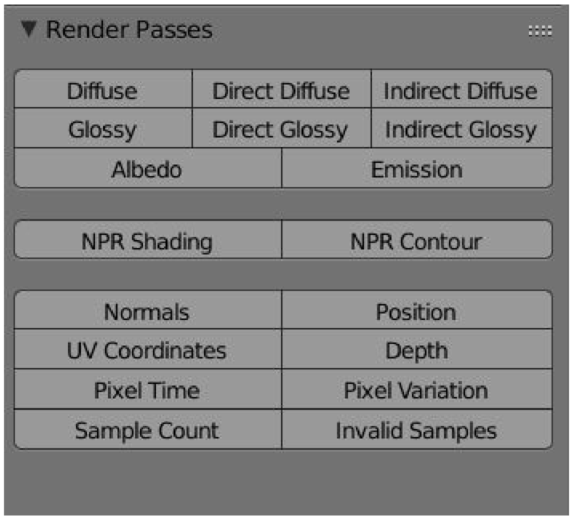


Mode: Inferno
Custom Map:
Ticks: 8
Min Range: 0.00
Max Range: 1.00
Line Thickness: 1.00

- **Add Stage:** This adds a post processing stage to your render. The order that the stages are applied follows the order of this list.
- **Remove Stage:** Removes the selected post process stage from the list.
- **Stage Parameters:** The parameters displayed here will vary depending on the post process stage that is selected.

3.1.2 Render Layers Panel

Render Passes



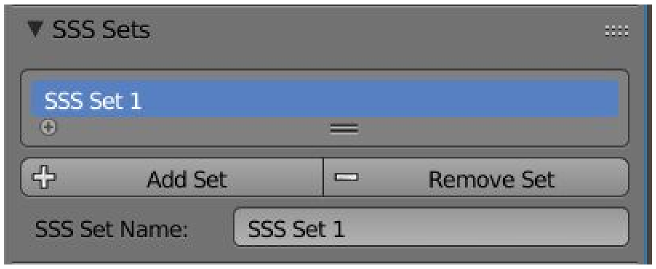
Render passes allow the lighting and geometric data of the scene to be sorted by reflection type and placed into independent images.

The passes are available to view during rendering and are also available in the compositor.

3.1.3 World Panel

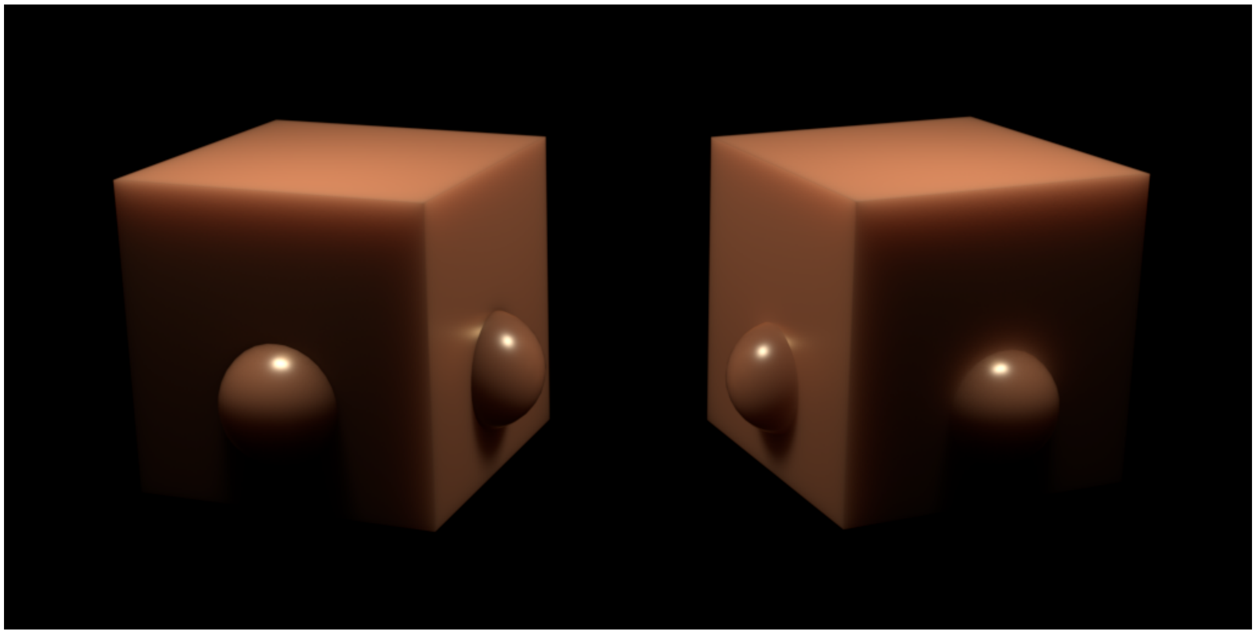
The world panel contains settings that affect the entire "world" of the 3D scene.

SSS Sets



SSS Set Name: SSS Set 1

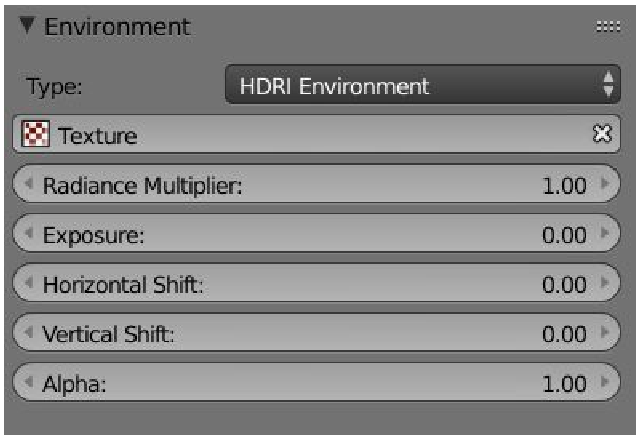
SSS sets are a way for appleseed to treat separate meshes as a single object when doing SSS calculations. This can prevent artifacts where meshes intersect.



Objects on the left are in separate SSS sets; objects on the right are in the same SSS set.

- **Add Set**
  - This adds an SSS set to the current scene.
- **Remove Set**
  - This removes the currently selected SSS set from the scene.
- **SSS Set Name**
  - This allows you to set the name of the SSS set.

Environment



▼ Environment

Type: HDRI Environment

Texture

Radiance Multiplier: 1.00

Exposure: 0.00

Horizontal Shift: 0.00

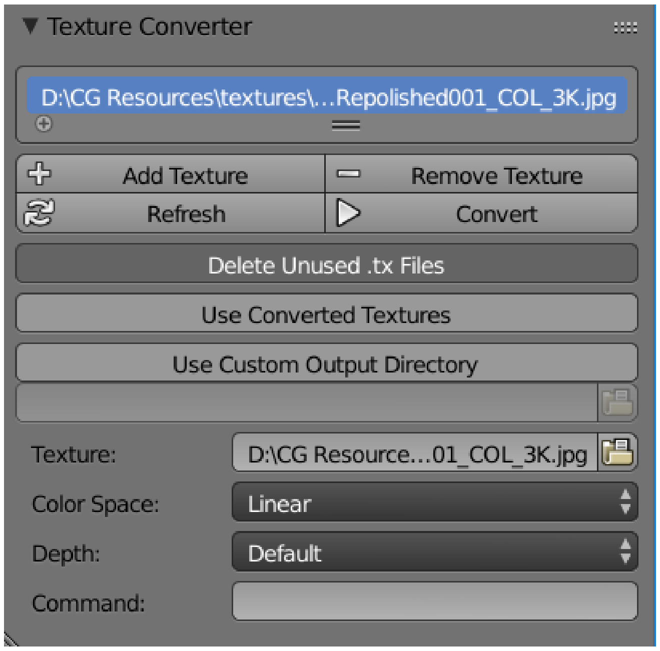
Vertical Shift: 0.00

Alpha: 1.00

- **Type:** This selects the type of environment that will surround the scene. Options include a constant color, a physically correct sky light, an HDRI environment map, and several other options. Properties for each type are covered in the shader reference section.

Texture Converter

The texture converter is a utility for converting traditional image formats (JPEG, PNG, TIFF, etc…) into tiled, mipmapped .tx files for use with appleseed's OSL shading system.



Texture: D:\CG Resource...01_COL_3K.jpg
Color Space: Linear
Depth: Default
Command:

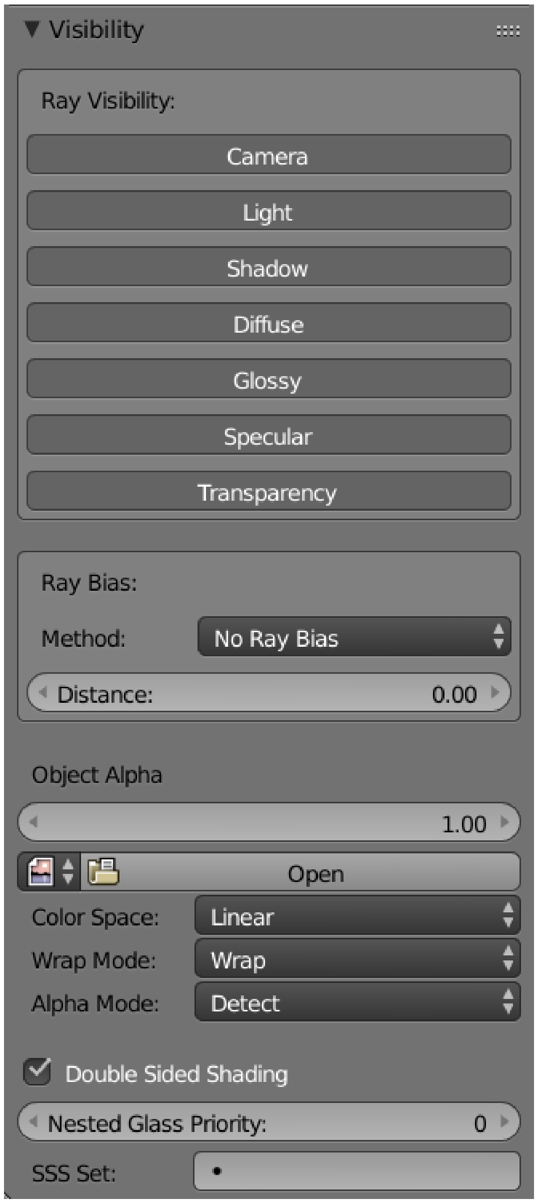
- **Texture List:** This shows the list of textures in the scene.
- **Add Textures:** This manually adds a texture entry to the list.
- **Remove Textures:** This manually removes the selected texture slot from the list.
- **Refresh Textures:** This scans the texture files that are used in the scene and adds them to the list. Entries will not be duplicated if they already exists. Entries will also be removed if they are no longer being used in the scene.
- **Convert Textures:** This launches maketx and converts all the textures in the list to .tx versions (if they haven't already been converted).
- **Delete Unused .tx Files:** This will delete any converted .tx files that are no longer being used in the scene (if a node is deleted for instance).
- **Use Converted Textures:** This tells blenderseed to substitute texture paths with the .tx version during export and rendering.
- **Use Custom Output Directory:** This allows you to put the converter .tx textures into a different directory than the original texture.

- **Output Directory:** This is where the .tx files will be placed when ‘Use Custom Output Directory’ is checked.
- **Texture:** This is the filepath of the selected entry in the texture list. Clicking on the filepath button will open a file selection window.
- **Input Color Space:** This parameter tells maketx what color space the original file is in. If it is anything other than ‘linear’ the colorspace will be converted during the maketx process. RGB textures such as color or specular color are usually sRGB. Bump maps, normal maps, and other grayscale value maps (roughness, metallness) are usually linear, but this may vary depending on the texture source.
- **Output Bit Depth:** This allows you to manually specify the bit depth of the .tx file. Leaving it at ‘default’ will set the bit depth to the same size as the input.
- **Additional Commands:** This allows you to specify any additional command variables that should be used during the maketx process.

3.1.4 Object Panel

These panels contain settings that can be applied to each object in the 3D scene.

Visibility



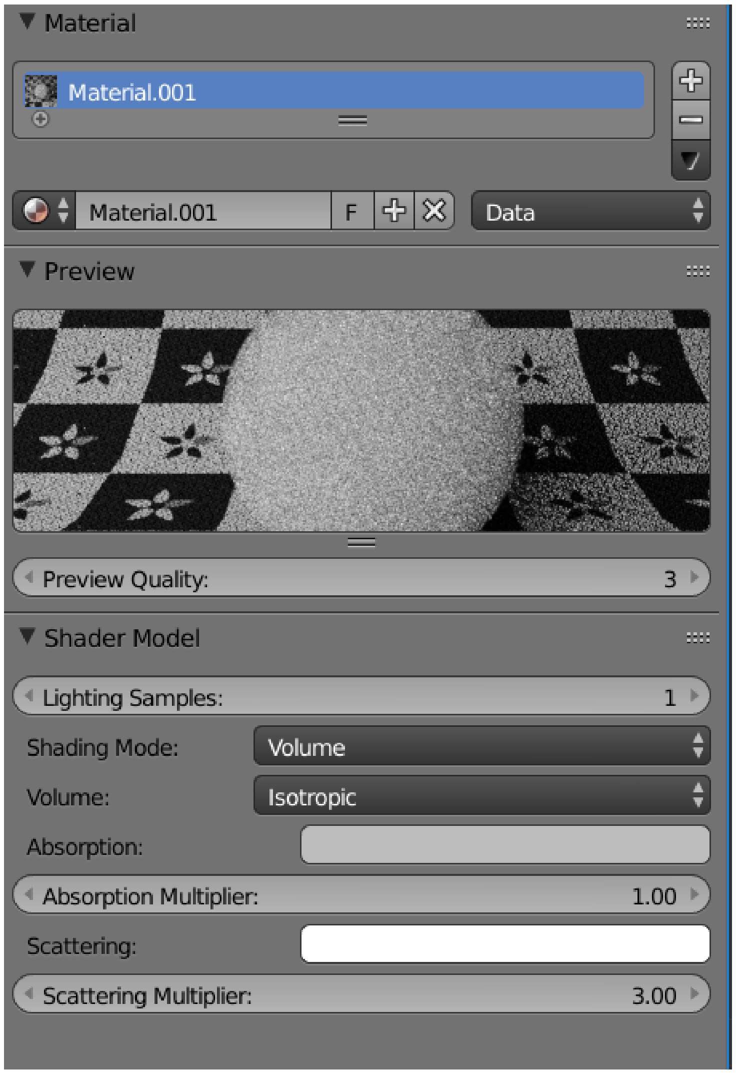
Method: No Ray Bias
Distance: 0.00
1.00
Color Space: Linear
Wrap Mode: Wrap
Alpha Mode: Detect
☑ Double Sided Shading
Nested Glass Priority: 0
SSS Set: •

- **Visibility Flags:** These flags determine if an object is visible to specific ray types during rendering.
- **Ray Bias:** This is used to offset the starting point of a ray from the surface of an object.
- **Object Alpha:** This allows a user to determine the transparency of an object. The parameter can also be driven by a texture map, allowing for cutout effects like leaves.
- **Double Sided Shading:** This tells the renderer to apply the object material to both sides of the polygon. This is required for glass materials to render properly.
- **Nested Glass Priority:** This allows you to set the priority for this object where it intersects with another object. A higher number equals a higher priority (i.e. 2 takes precedence over 1). This allows you to correctly

render intersecting media like fluid in a glass.

- **SSS Set:** A user can assign this object to an existing SSS set by selecting it here.

3.1.5 Material Panel


Material

Material.001

Material.001 F Data

Preview

Preview Quality: 3

Shader Model

Lighting Samples: 1

Shading Mode: Volume

Volume: Isotropic

Absorption:

Absorption Multiplier: 1.00

Scattering:

Scattering Multiplier: 3.00

- **Material Slot:** This window is where you create and name materials.
- **Preview:** You can see a preview of the material here.
- **Preview Quality:** This allows a user to set the sample count of the preview window. Higher numbers will have longer render times.
- **Lighting Samples:** Raising this number may resolve render noise by raising the number of direct lighting samples.
- **Shading Mode:** Select either OSL surface shading or volume shading.

- **Volume:** Selects the volume shading mode to be used.

3.2 OSL

3.2.1 Open Shading Language

What is it?

Open Shading Language (OSL) is a small, customized computer language intended specifically for writing shaders in a physically plausible rendering engine. While it was originally designed at Sony Pictures Imageworks for their Arnold renderer, it has since been integrated into appleseed and several other renderer engines (such as RenderMan and Cycles).

How does it work with appleseed?

Since the beginning appleseed has had its own native shading system (henceforth the 'internal' system). While this system works well enough, it was limited to defining material surfaces and optionally assigning UV mapped textures to them. That was it. It couldn't do any kind of procedural patterns, coordinate manipulation, fancy BSDF mixing, or any of the other utility functions that were needed for a production renderer. Instead of expanding the BIS system with these features, the decision was made to integrate OSL instead, as it is fully capable of all these features and is developed and hosted by a major visual effects company. As of the current release (1.0), the internal shading system is no longer exposed in blenderseed, with the exception of the volumetric shader (as this is not yet supported by OSL).

How do I use it in blenderseed?

OSL is available in conjunction with the node editor in Blender. When a material is created, by default it will add an OSL node tree and link to it. If you have an open node editor it will automatically update to show the active material tree. You can add nodes to the editor using Shift+A. You can add nodes from the different categories to build up your material, but always make sure the final node is an as_closure2surface node, as that is required to export the material properly.

Where does the OSL shading system get the nodes from?

The OSL shaders are found in the "shaders" directory of the appleseed folder. When blenderseed starts up, it scans this directory and builds the nodes dynamically based on the parameters contained inside the .oso files. When the scene is rendered the output file will point to those shaders and describe the connections between them. The shaders themselves are not copied or moved unless the scene is exported. In this case the shaders are copied to the export directory.

Textures with OSL

OSL can directly load most image formats (JPEG, PNG, TIFF). However, this is not the best practice for using it. Full sized textures take up a lot of memory in the render, and in many cases objects will not be sufficiently large enough in the output frame to justify the memory space used for high resolution, detailed images. The solution is to create a .tx file using maketx, a utility that ships with appleseed. Maketx will take an input image and produce a tiled, mipmapped version that is far more efficient when used with OSL. This is due to a few different features: for one the tiling allows appleseed to only load the sections of the image that are visible. Second the mipmapping allows appleseed to load in the appropriate resolution of the texture depending on how large the object is in the output frame. These two features not only reduce the memory requirement for textures, they also speed up the render.

How do I convert textures?

Textures can be converted to .tx files using the *texture converter* panel in the material tab.

Can I use OSL shaders from third parties?

Yes, with a few exceptions. OSL shaders transfer their results to the host renderer BSDF's by using an item called a 'closure'. While OSL suggests a limited set of generic closures, many renderers don't implement all of them and many add their own closures to the list. OSL shaders that use closure calls may or may not work with a different renderer. If the OSL shader is being used for pattern generation or processing, it will usually work just fine with another renderer. The only drawback is that third party shaders may not have the metadata that blenderseed uses to construct its node UI.

Can I write my own shaders?

Absolutely. OSL shaders can be written in any text editor and compiled into .oso files using the oslc utility that's included with appleseed. Once you've compiled the shader, you can put it into the "shaders" directory or add the folder name to the APPLESEED_SEARCHPATH environment variable. If you choose to write your own OSL shader, there are several formatting rules and *metadata* tags that should be used in order to properly build the node's UI and category.

3.2.2 OSL Metadata

When blenderseed scans an OSL shader it looks for several specific metadata tags that can be attached to a shader or parameter. This metadata is used to populate the node interface in Blender with buttons, sockets, labels, and the range a value can be. While this metadata is not essential to the shader execution itself, it is essential for making the node usable.

- **Shader Metadata: This metadata is used to describe the shader as a whole.**
  - **string node_name** This is the name of the node. If it is not used the name of the OSL file will be used.
  - **string classification**

    This tells blenderseed what the shader is. Options are:
    - surface
    - shader
    - utility
    - texture/2d
    - texture/3d
- **Parameter Metadata: This is metadata that is used on each parameter.**
  - **string label** This defines the name of the parameter that will appear in the UI.
  - **int as_blender_input_socket** This tells the UI whether it should hide the socket connection for this parameter. You would use this for any parameter that you do not want to be controllable through a texture. '0' hides the socket.

  - **string help** This is a tooltip that will appear whenever a parameter is hovered over.
  - **string option** This is used to define the menu options of a drop down list. Entries should be formatted with a ‘|’ separating each option.
  - **string widget** This is used for defining two things: if a string parameter needs a file selector option, or if the direct control of a parameter should be hidden (so only the socket is visible). Options are:
    * filename (tells blenderseed to add a file selection button to a string property)
    * null (tells blenderseed to hide the direct control for a parameter)
    * checkBox (tells blenderseed that the value should be expressed as a checkbox. Requires an integer value and int as_blender_input_socket to be set to 0)

- **Max/Min limits: Integer and float properties can be set with one of four different options that control the range of adjustm**
  - max
  - min
  - softmax
  - softmin
- **Examples:**
  - string help = “This is a help string”
  - int softmin = 5
  - string node_name = “asClosure2Surface”

3.3 General Tips

3.3.1 Path Tracing Tips

i.e. how do all these crazy parameters fit together?

What is Path Tracing, and why should I care? Path tracing is one of the dominant (if not *the* dominant) rendering technique in use today, and the images it produces can be almost indistinguishable from reality. It does this by using raytracing to accurately trace the path a beam of light takes as it bounces, reflects and refracts its way to the camera lens. While computationally expensive, path tracing has the advantage of naturally capturing the way light moves through a scene. Effects like global illumination (illumination created by light reflecting off objects) occur naturally with path tracing. With older methods such as REYES, these effects were laborious to create.

Path Tracing explained in thirty seconds… Path tracing works by taking a pixel from the output image frame and sending a virtual ray from the camera, through that pixel, and out into the scene. The ray will bounce off objects until it runs out of energy, hits a light source, or reaches a preset number of reflections (bounces). When that happens, the ray is traced back to the camera through all the objects it has hit and the final 'color' of that pixel sample is determined.

What are you skipping? Quite a bit. What was described above is the path that an individual ray takes, a single sample per pixel would account for only a minute fraction of the possible paths light could take through the scene to that pixel. This lack of accuracy is visible as noise in the image. Path tracers get around this limitation by shooting dozens or even hundreds of rays through each pixel. As each ray follows a slightly different path, the color of the pixel will 'converge', that is, gradually get closer to the correct value. As this happens the noise in the image also decreases. While a path tracer can never perfectly converge a pixel or image (the samples needed is almost infinitely high), eventually it gets close enough that the remaining noise is not objectionable to the human eye.

What else? Plain path tracing can produce beautiful images, eventually. One challenge that is frequently encountered is that light sources are not found often enough when rays bounce around randomly (especially small sources). To solve this issue, when a surface is hit by a ray, a secondary path (or paths) will be traced from that point directly to a light source. This is referred to as 'next event estimation'. One of the major features that distinguish one path tracer from another is how they perform this direct light sampling in scenes with dozens (or hundreds) of lights.

Okay, but path tracing is soooooooo slow Path tracing by its very nature is (and always will be) slower than legacy methods. While improvements to speed are continually being made, a path tracer simply has to crunch more numbers than older methods.

What are the settings and how do they connect? Get to the point… For appleseed, the settings that control the quality of the render (and the time spent to do it) are found in the render settings panel.

First off, some terminology:

- **Direct lighting:** This is any lighting that results when rays from a light source strike a surface without being blocked.
- **Indirect lighting:** This is lighting that results when a ray bounces around through the scene at least once (i.e it doesn't come directly from a light source).
- **Caustics:** These are distinctive patterns that occur when light rays are focused together because of reflection or refraction. Light patterns on the bottom of a pool are a good example. Basic path tracing is notoriously bad at resolving caustics, as they are the result of very specific reflection and refraction light bounces. When the ray is traced 'backwards' (i.e. from the camera to a light) the odds of these specific paths being found is relatively small. Appleseed has an SPPM (Stochastic Progressive Photon Mapping) integrator that is much more effective for rendering caustics.
- **Convergence:** This is a term used to indicate how close an image is to being 'correct', or finished. When the render first starts each pixel is quite far off from where it should be, and as it accumulates more samples it gets closer, or 'converges', to the correct color.
- **Firefly:** A specific kind of lighting artifact that affects path tracers. It is a pixel that is unusually bright compared to the neighboring pixels, appearing much like a firefly in the dark. They typically happen when a ray bounces off a surface and randomly hits a very bright, small light source. The small size of the light means it isn't hit very often, but when it is, it contributes a large amount of energy to a pixel.

Some of the notable controls are:

- **Samples:** This is the main quality control. The higher this is, the more rays are sent through each pixel, the cleaner the image will be, and the longer it will take to render.
- **Passes:** A pass is one round through all the pixels in the output image. Appleseed can perform what is called 'progressive rendering', which is when the image is converged through multiple passes over the image. The benefit of this is that you can see a rough version of the entire image that cleans up with each pass until the maximum number of passes has been reached. The downside is that it slows down

the rendering process somewhat, so there are tradeoffs to using it. If you decide to render in multiple passes, the ‘Samples’ control will determine how many pixel samples are taken per pass.

Path Tracer settings:

- **Directly Sample Lights:** This is the control that determines if light sources are directly sampled at surface hit points. There’s really no reason why you’d turn this off except if you have no discreet light sources in your scene.

  There are two additional controls that affect direct light sampling:

  - ‘Samples’ sets how many rays are traced to light sources for each surface hit point, so you may want to raise this if you have a scene with a lot of direct lighting.
  - ‘Low Light Threshold’ is a setting that you can use to speed up render convergence at the expense of lighting quality. It does this by not directly sampling a light if that light is determined to have less illumination on that point than the threshold setting. It’s a tradeoff, less light rays mean faster convergence, but too high of a threshold can cause the lighting of the entire scene to darken.
- **Image-Based Lighting:** This lets you control whether HDRI backgrounds can contribute light to the scene. The Samples parameter next to it has the same purpose as the direct lighting samples.
- **Caustics:** This is a scene wide control that enables or disable refractive caustics. Caustic patterns are very difficult to render properly with a path tracer (due to the unlikelihood of the light source being hit properly) so it’s usually best to disable them entirely.
- **Bounces:** These controls allow you to limit the number of times a ray can bounce. While you can allow the ray to bounce up to 99 times (basically unlimited) virtually all useful lighting information is gained after six bounces or so. All that extra ray tracing time basically gets you nothing. The bounce limits are also settable by the type of bounce. For instance, you can tell a ray to terminate after it hits a diffuse surface twice, regardless of what the global setting is at. Keep in mind that some situations may require high bounce counts in one category or another. For example, glass will require high specular bounce levels to look correct.
- **Max Ray intensity:** This is a bit of a cheat in that it alters the intensity of light bounces, but it benefits the final render by reducing fireflies. It does this by putting a limit on how bright an indirect light ray can be. Lowering this too far can cause indirect lighting to appear dull or washed out. Once again, it’s a compromise between faster convergence and lighting accuracy.
- **Russian Roulette Start Bounce:** This is another optimization that attempts to reduce the number of light rays traced in the scene. After a ray has reached the same bounce count as the setting, it stands a chance of being randomly terminated. While this does a good job of reducing the number of rays that need to be traced all the way to the bounce limit, setting it too low can hurt the convergence of the image by stopping paths too soon.

What settings should I use? It depends on the image you’re trying to render, honestly. If you are rendering an outdoor image lit by an HDRI sky and you have a few shiny objects that are directly lit, you could feasibly get a converged image with less than 100 samples and only a few bounces. If, on the other hand, you’re rendering an indoor scene with highly diffuse objects that are largely lit by indirect lighting, it will take considerably more samples (maybe even over 1,000) and a higher bounce limit. Trial and error are the keys. Use render regions if possible to isolate difficult areas of illumination.

What is the adaptive sampler? Is it better? The adaptive sampler adds an extra step to the rendering process. After a set number of samples, it will evaluate the remaining noise in the tile it is working on. If that noise is below a certain threshold, it will stop rendering that tile. The advantages of this process are that more difficult parts of the image will receive more samples for the same amount of render time, leading to an overall cleaner image. While the differences between adaptive and uniform rendering can often be subtle, the noise distribution and potential time savings of the adaptive sampler are often preferred.

Adaptive sampling controls:

- **Noise threshold:** This determines the acceptable level of noise for a tile to be considered done. Lowering the number lowers this level, hence the tile will render for longer.
- **Max samples:** This is the upper limit for how many samples can be taken per pixel. If a pixel hits this level and still hasn't reached the noise threshold, it will stop sampling anyway.
- **Uniform samples:** This is how many samples each pixel will receive before adaptive sampling begins. This step is necessary to resolve fine details in the image. If it is set too low there may be noise or other artifacts in the image that never clear up even with high max sample levels.
- **Step size:** This is how many samples are added to a pixel in between noise evaluations. This noise evaluation does take some processing time, so it may be tempting to raise this number. However, if it is set too high you may be wasting samples. For instance if you set it to 64 samples and a tile only needs 75 samples to converge, it will still have to take the remaining 53 samples to reach 128, which is the next time the noise evaluation would run.

What about denoising? One of the biggest disadvantages of path tracing is the image noise of an incomplete render. This is compounded by the fact that as the image continues to render, additional samples make less and less of an impact. This means it can often take a huge amount of time to remove the last bits of noise. To eliminate this time sink, most path tracers have some form of denoising that can be used on the image instead. Appleseed uses the BCD denoiser. While denoising can speed up the render process, incorrect settings or too low convergence will cause blurry textures and other image artifacts.

Anything else? High resolution HDRI's are difficult to sample and may lead to slow convergence. You are better off using a low-resolution image for the lighting itself and then compositing in a high resolution background afterwards.

3.4 Shaders

- Shader Reference

CHAPTER 4

About blenderseed

The original appleseed integration into blender was written by François Beaune when appleseed was in an alpha state. It was a primitive plugin, matching an equally primitive (at the time) renderer. Over the next several years appleseed grew and improved in leaps and bounds, and blenderseed has grown along with it thanks to the dedication and efforts of many volunteer authors:

- **Contributing Authors:**
    - François Beaune
    - Joel Daniels
    - Jonathan Dent
    - Petra Gospodnetic
    - Luke Kliber
    - Jasper van Nieuwenhuizen
    - Esteban Tovagliari

CHAPTER 5

Tutorials

Blender Diplom: Rendering Caustics in Blender with appleseed