



The **YafaRay User's Guide** by www.yafaray.org is licensed under a [Creative Commons Attribution-Share Alike 3.0 Unported License](https://creativecommons.org/licenses/by-sa/3.0/). Permissions beyond the scope of this license may be available at www.yafaray.org.

Downloads for various platforms are available on our [Downloads Page](#). Please open a thread in the forum section if you have any question about the YafaRay installation.

Installation notes.

Table of Contents:

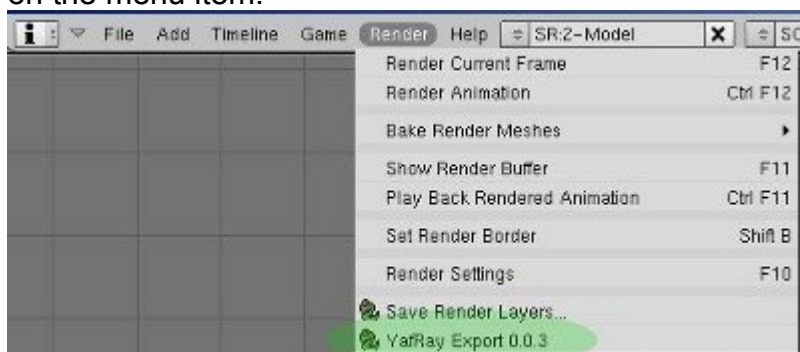
- [User Prerequisites.](#)
- [General Workflow.](#)
- [Windows Installer Notes.](#)
- [OSX Installation Notes for YafaRay 0.1.1.](#)
- [Debian packages for Ubuntu.](#)

User Prerequisites.

It is good to have some previous knowledge about Blender before taking on YafaRay. It will be particularly useful to know about Blender's interface arrangement, Blender lighting features, Blender texturing workflow and general scene handling. Some background on basic lighting & shading techniques will be very helpful to get good results with YafaRay as well.

General Workflow

- YafaRay works with any Blender official release from blender.org, as long as the binaries used are compiled with the same version of python than YafaRay.
- YafaRay uses a python-coded settings interface to set up most of lighting and shading parameters and all rendering settings. Once YafaRay is correctly installed, the settings interface can be launched by an item automatically added in the Blender **Render** menu, called **YafaRay Export**. Divide your 3Dwindows and click on the menu item.



- To launch rendering, you must use the **RENDER** buttons located in the settings interface. A separated window will pop up displaying the render progress as it was

with the old versions. In this separated interface you can view and save your images. Some more features are planned and if you have ideas, let us know (or send patches ...).



- You can try some example scenes from here: <http://www.yafaray.org/download/examples>
- The render GUI will block Blender as long as it's open. So you must close the GUI in order to continue with Blender. Remember to save your render in your hard disk before closing the render GUI. You can enable as well *Autosave* button in the *Settings* section, so the render is automatically saved in your hard disk. The render will be saved in the same folder than the Blender scene file.
- There is no way of "reopening" the GUI with the last rendered image. If you close the window, the rendered image is gone.
- More information about using YafaRay [here](#).

Windows Installer notes.

The installer has been tested and should work alright. If you experience any problems, please tell us right away!.

First of all, for everything to work correctly, [Python 2.6.2](#) must be installed. If Python has been correctly installed, you will get a message in Blender back terminal, saying '**Checking for installed Python, got it!**'

Blender should be already installed in your system and you should close Blender if opened, before installing YafaRay. The installation process is pretty straightforward, just follow on-screen instructions. If YafaRay has been correctly installed, you will get a new item in Blender *Render* menu, called *YafaRay Export*. Use it to launch the YafaRay settings panel, as explained in [General Workflow](#) section.

If you experience any problem, check out [this forum thread](#).

OSX installation notes for YafaRay 0.1.1

The install has been changed completely, to match the new structure of YafaRay:

- **Remove all old Python scripts before you install.**
- All YafaRay-Python-files are installed now in `/usr/local/share/yafaray/blender`. (all scripts and bindings, 10 files)
- YafaRay-libs are installed in: `/usr/local/lib` (as before)
- YafaRay-xml is installed in: `/usr/local/bin` (as before)
- QT-frameworks are installed in: `/library/frameworks` (as before)
- After the installation there will be only a symlinked **yafaray_ui.py** left either in Scripts-Folder or in Blender.app-package-scripts, depending on where you chose to install it. This script now controls the whole Exporter-functionality. **Don't worry it**

- shows only 0 byte to be installed, it is caused by being only a symbolic link.**
- The installer searches for Blender in Applications/Blender, only if found there, the "**Exporter->Blender**"-install is shown. If you have Blender in another location or renamed it, please revert this or choose "Exporter->PythonScripts"-install. If a PythonScripts-folder is not present, it will be created by the installer.
 - No need anymore to drag and drop anything.
 - **Just take your choice for exporter-location, thats all needed.**
 - Please read also the instructions whereas installing.
 - The renderparameter-overlay should now show: YafaRay 0.1.1 !!!
 - In terminal: yafaray-xml -v should too show: YafaRay 0.1.1 !!!
 - Install YafaRay-installer as usual. Used as an update, it only updates outdated components. You can skip QT if you already have QT 4.x or higher installed or want to hold another version. QT-4.5.1 is bundled with the install, but all QT-versions 4.x should work too.
 - If you never used external scripts before, make sure to set the pythonscripts-path in blender-preferences.
 - No further installs needed
 - Make sure you use a Blender compiled with Python 2.5 (called "custom" at blender.org/downloads)
 - Important: YafaRay is compiled against Python 2.5.1-Apple, the pre-installed one! Don't install any other Python-version. This could break functionality.
 - The tested configuration is using the preinstalled Python 2.5.1-Apple on OSX 10.5.x (resides in System/Library/Frameworks) and the Blender compiled with Python 2.5 "custom", too!
 - Other combinations may not work.

Greetz...Jens

Debian packages for Ubuntu.

Uninstall previous versions before installing YafaRay. If the export script doesn't appear on render menu, refresh scripts in the user preferences of Blender.
All dependencies should be set properly, so the installation may prompt you that additional packages from Ubuntu repository which will have to be installed.

What YafaRay is.

Table of Content:

- [What YafaRay is](#)
- [YafaRay Features](#)
- [Project History](#)

What YafaRay is.



YafaRay logo, created by [Javier Galán Rico](#) in 2005.

YafaRay is an open source raytracing render engine. As an open source project, users and developers have free access to the source code and it is free, as in free speech.

Ray tracing is a rendering technique for generating an image by tracing the path of light through the 3D scene. This technique tries to reproduce the natural behaviour of light and its particular effects on surfaces, such as reflection and refraction, caustics and indirect lighting.

YafaRay works as a render engine. An engine consists of a 'faceless' computer program that interacts with a host 3D application to provide very specific raytracing capabilities "on demand". [Blender 3D](#) is the host application of YafaRay.

By using python scripts, YafaRay takes advantage of the new ID Property mechanism of Blender to retrieve most of the scene data, without adding any custom code to Blender. Other specific parameters are set up by a python-coded settings interface used from Blender.

YafaRay Features.

YafaRay main features are:

Lights:

- point
- spot light

- area light (rectangular, or parallelogram actually)
- mesh light (uses a triangle mesh as light source)
- sphere light
- directional (with optional radius)
- sunlight (basically a directional with incoming direction sampled from a cone)
- environment light (background), with importance sampling for efficient image-based lighting (even without GI)

Materials:

- Multiple materials for a mesh.
- basic diffuse w. specular reflection, transparency and translucency, support for shader nodes on various properties
- diffuse+glossy material (Ashikhmin&Shirley), Blinn or anisotropic microfacet distribution w. fresnel effect
- shader nodes support on diffuse+glossy color, glossiness and bump
- coated version of the above mentioned, adding specular reflection with fresnel effect (dielectric)
- basic glass (dielectric) material, with fresnel, filter, absorption and dispersion.
- emit material
- export of blender's texture layers as shader nodes
- blend material, using a blend value or a texture map.

Mapping:

- Multiple textures for a shader.
- UV coordinates.
- Flat, cube, tube, sphere in global and relative coordinates.
- Blending modes.
- Stencil.

Textures:

- Basic image textures (tga, jpeg, png, exr, hdr),
- Procedural textures: cloud, marble, wood, voronoi, musgrave, distorted noise and "RGB-cube".

Backgrounds:

- Constant.
- Sunsky generator with sun light and sky light.
- Texture, Image-based Lighting.
- Simple gradient.

Cameras:

- Perspective camera with raytraced DoF.
- Architect with raytraced DOF.
- Orthographic camera.
- Angular camera.

Surface integrators:

- direct lighting with support for ambient occlusion and caustic photon maps
- path tracing
- photon mapping with final gather
- bi-directional path tracing
- Volume integrator

Volume rendering.

Antialiasing:

- adaptive (simple color threshold based)
- variable size reconstruction filters (box, gauss and mitchell-netravali currently)

Transparent shadows.

Multithreaded rendering passes and radiance map creation.

Basic XML writing and reading (using libXML) for scenes (see YafaRayXML for specifications).

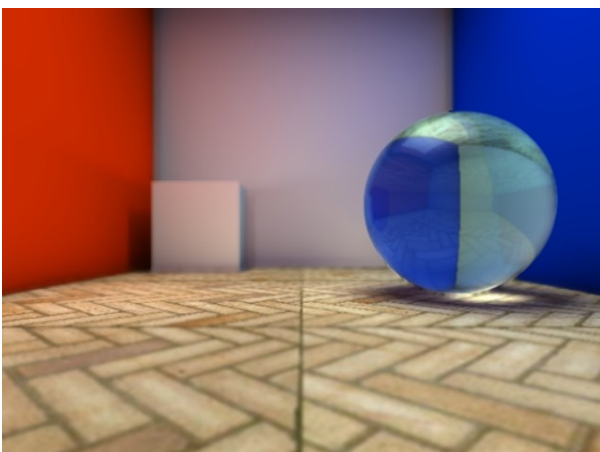
Plugin based.

Project History

The **YafRay** (Yet Another Free Raytracer) project was started in 2001 by Alejandro Conty Estévez, and the first public release was in July 2002. These are Jandro's recollections from those years:

<http://www.blender.org/documentation/html/c12235.html>

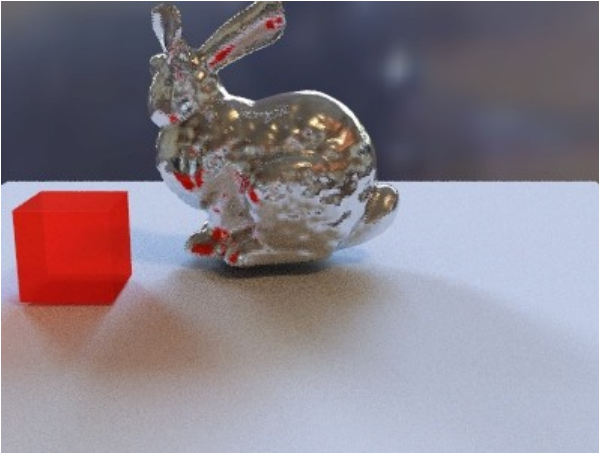
By request from the Blender Community, YafRay was added as a Blender plugin from the 2.34 release on, in August 2004. Alejandro de Greef (eeshlo) coded most of the features in the late YafRay releases. At that time it was already evident that a code rewriting was necessary in order to add new advanced features to YafRay. The last YafRay release was the 0.0.9 version in Summer 2006.



One of the first YafRay renders by **Jandro**, circa 2001.

YafaRay is the result of rewriting the YafRay source code from scratch. Mathias Wein (Lynx) started to work on the new engine in December 2005. As a result of the rewriting

and to make people aware that it was actually a completely new engine, the YafRay name was changed into YafaRay. Nobody knows for sure what the added 'a' stands for.



One of the first YafaRay renders by **Lyxn3D**, circa 2005.

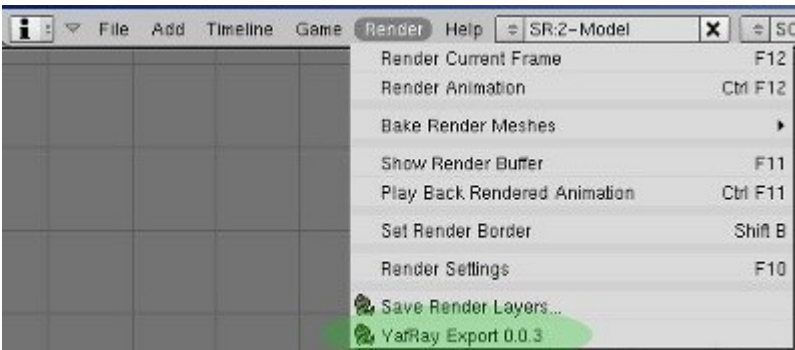
Render output.

Table of Contents:

- [YafaRay settings UI introduction and Render Buttons.](#)
- [Render Output Windows.](#)
 - [Blender settings supported.](#)
 - [Render Windows Settings.](#)
 - [Tone mapping.](#)

YafaRay settings UI introduction and Render buttons.

Once everything is installed, a special entry is added in the Blender Render menu, called *YafaRay Export 0.0.3*, which automatically executes the YafaRay User Interface (UI) inside a Blender 3DWindows. Split your 3DWindows and click on YafaRay Export 0.0.3



Note: [Here](#) you can learn how to split the Blender 3DWindows:

The YafaRay User Interface (UI) will be shown, which is divided into:

1. Three function buttons which are *Render*, *Render Anim* and *Render View*.
2. Three main sections of settings which are *Object/Light/Camera*, *Material*, *World* and *Settings*.



- **RENDER:** Launches the scene rendering in a separate Render Output Window on which the render progress can be seen. Use the Render Output Window settings to save your image. Once the render is finished or stopped, the Render Output Window must be closed before coming back to Blender. Don't forget to save your render.
- **Render anim:** renders the frames set in Blender, using the Blender active camera. Frames are saved in [Blender output dir](#). Use the first textbox ("Directory/name") in the F10 "Output" panel to determine where to output the animation frames. It also

uses the '#' character in the Blender input field to enumerate frames. If this box is empty, the Render directory in the Blender Preferences is used.

- *Render view*: renders the Blender active 3DWindows.
- *Objects*: The script takes the current Blender selected and active object and gives it custom YafaRay properties, depending on the object selected (camera, light or object). Lights and camera settings in the YafaRay UI script replace almost all Blender settings.
- *Material*: The script takes Blender defined materials and applies them custom YafaRay material properties.
- *World*: background settings and volumetric integrators.
- *Settings*: This section is used to choose a lighting method and to configure render general parameters and render anti-aliasing.

You can scroll the settings UI up and down by holding down the mouse scroll wheel while moving the mouse on the interface.

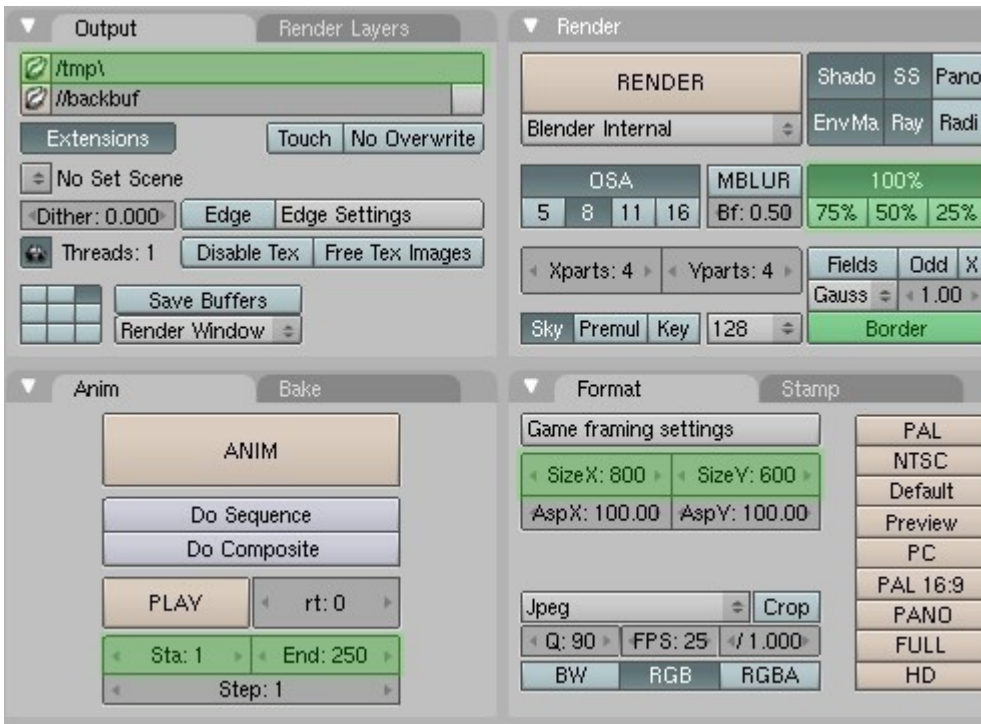
Render Output Window

The render output is handled in a separated window from Blender. This window appears when the rendering process is launched from the settings UI. It includes options and parameters to save renders. The render window will block Blender as long as it's open. You must close the it in order to continue working with Blender. Remember to save your render in your hard disk before closing.

Blender Render Settings Supported

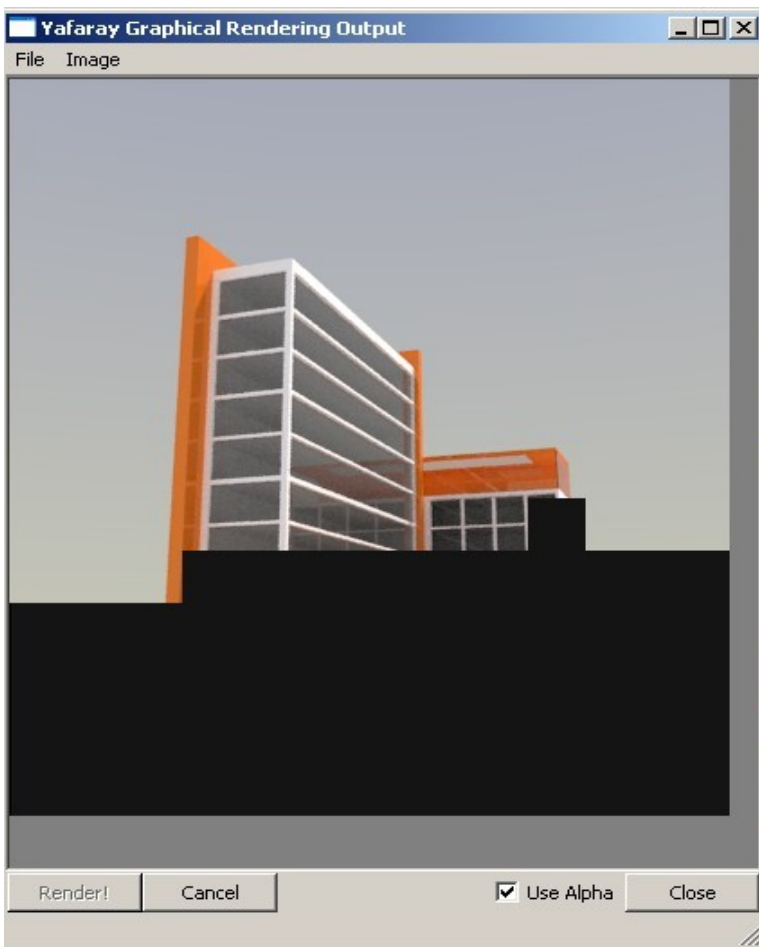
Render resolution is pre-configured in Blender, as well as the numbet of frames and frame output. These are the Blender render settings that will be taken into account when rendering with YafaRay (in green):

- *Ouput* panel: Button to set output directory for frames
- *Render* panel: Buttons to set a relative render size and Border button (Shift+B), for previews.
- *Anim* panel: start frame, end frame
- *Format* panel: Buttons to set image size



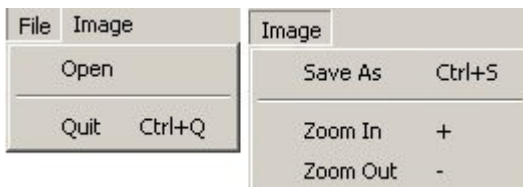
Render Output Window Settings.

The Render Output Window is an external application to Blender in which YafaRay renders are viewed and saved. Settings in this external application are:



- *Render!*: Renders the current scene again. Use this button to render a scene loaded in YafaRay XML format.
- *Cancel*: Cancels the render in progress.
- *Use Alpha*: When enabled, images are saved with RGB and alpha data instead of background, if the file format supports it (PNG).
- *Close*: Closes the Render Output Window and returns to Blender. When the the window is closed, the rendered image is gone unless it is saved.

Render output Window menus.



- *File, Open*: Loads a YafaRay scene in XML format. Scenes can be saved in the YafaRay XML format by using the *Output to XML* button in the 'Settings' section (UI).
- *File, Quit*: Quits the Render Output Window and returns to Blender.
- *Image, Save As*: Saves the current render as an image file, using the supported formats, which are BMP, PNG and EXR.
- *Image, Zoom In*: to zoom in details in a render. You can use + in your keyboard as well.
- *Image, Zoom Out*: zooms out for a full view. You can use - in your keyboard as well.

Tone mapping.

Renders can be saved (*save as*) using a high dynamic range format (EXR). The aim of this feature is to perform tone mapping operations on the High Dynamic Range (HDR) of the image. Some ideas about this option:

- YafaRay always works internally using high dynamic range, which means that it produces a range of colors and luminance more according with the human eye. The colors you see on display are only a limited part of the colors YafaRay has calculated for a given render. This is true for most raytracers.
- This extra range cannot be stored using 8-bits file formats (bmp, jpg, png).
- **EXR** is a [high dynamic range imaging](#) image file format, released as an open standard by [Industrial Light and Magic](#) (ILM) and released under a free software license. It is notable for supporting 16-bits-per-channel. **EXR** allows for a dynamic range of over thirty [stops](#) of exposure.
- Many common post processing operations like gamma correction, saturation or brightness & contrast are gamma-destructive, because results in such operations can be only approximated or the destination value does not exist because it is out of range. This is particularly true when editing 8-bits formats such as JPG.
- **Tone mapping** is a technique to approximate the appearance of high dynamic range images with a more limited dynamic range. With tone mapping, you can choose a mood for your renders from the full range of colors YafaRay is able to calculate.
- The *Exposure* control in the old **YafRay 009** was in fact a tone mapping control.

- It is a good idea saving always a EXR version of your render, along with the 8-bits version.
- By using HDR, you can perform these operations without destroying image data. Apart from exposure, tone mapping filters allow for other interesting operations, such as saturation control, brightness & contrast controls, chromatic and light adaptation, etc.

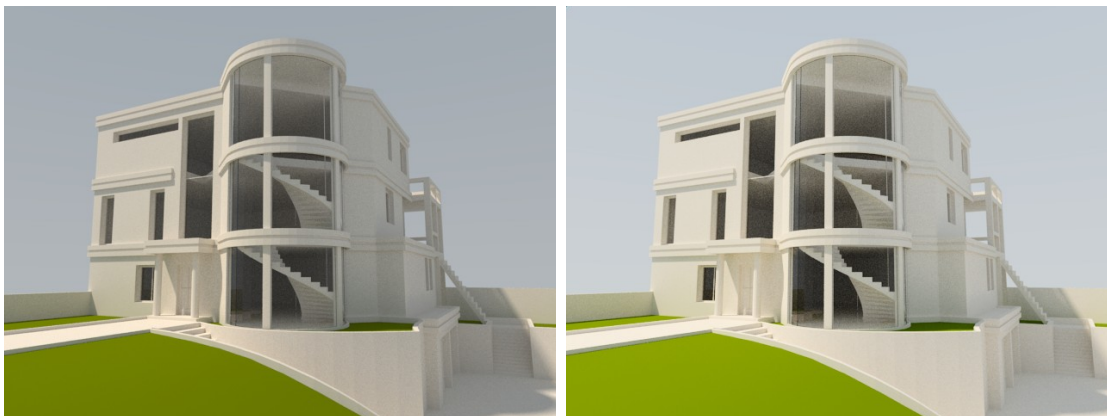
At the moment we don't have implemented any tonemapping filter in YafaRay, although it is a planned feature. in the meantime, we would like to recommend QTpfsgui for tonemapping operations on your HDR renders:

<http://qtpfsgui.sourceforge.net/>

The correct workflow in **qtpfsgui** is tone mapping on the linear (therefore darker) HDR image, which means that the **Pre Tone mapping gamma adjustment** should be = $1 / \text{YafaRay Output Gamma}$. This will produce more consistent colors and a better contrast. This is a table for tonemapping **Pre gamma** values:

YafaRay Output Gamma	qtpfsgui HDRI display	qtpfsgui Pre TM Gamma
1.8	Linear	0.5555
2.2	Linear	0.4545

This a comparison between a YafaRay original render (left), and its tone mapped versions (right), using Reinhard 2005. Scene by **Kronos**:



Objects, Lights and Camera settings.

Table of Contents:

- [Object, Lights, and Camera settings introduction.](#)
- [Object \(meshlight\).](#)
- [Lights.](#)
 - [Point and Sphere.](#)
 - [Directional and Sun.](#)
 - [Area.](#)
 - [Spot](#)
- [Camera settings.](#)
 - [Architect / DOF.](#)
 - [Angular.](#)
 - [Orthogonal.](#)
 - [Perspective / DOF.](#)

Object, Lights, and Camera settings.

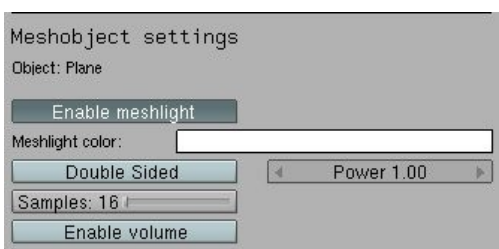
In general, settings under this section of the YafaRay UI are used to give specific YafaRay properties to light sources and cameras in the Blender 3D scene. Only a limited number of Blender settings are taken into account by YafaRay. The general workflow is:

1. Create or select an existing object (camera, light or mesh) in the 3D scene.
2. Give it YafaRay specific settings using the YafaRay settings UI.

Object (meshlight).

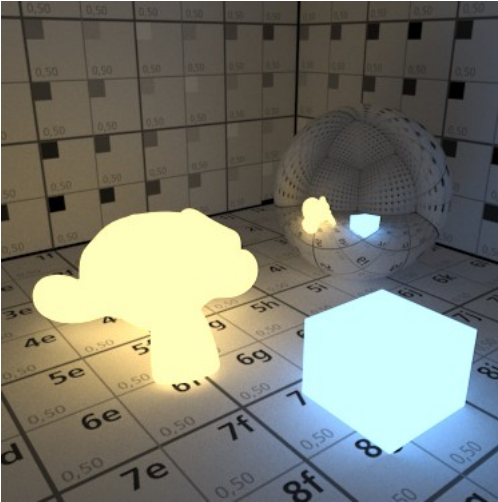
With this feature, objects in the scene can act as area light sources. The soft shadows produced by area lights need to be sampled several times and then interpolated to reduce noise. This type of light takes more time to render in contrast with point light types such as spot and point.

First of all, you must select the mesh you want it to act as a light source. Then you must click on the *Object/Light/Camera* main button. Finally you must activate the *Enable Meshlight* button. To change the light color, just click on the rectangle next to Meshlight color to open a Color Picker.



- *Meshlight color* rectangle: opens a Color Picker.
- *Power*: intensity multiplier for meshlight color.

- *Double Sided*: considers both sides of the mesh as arealight sources.
- *Samples*: defines the amount of samples taken to calculate the soft shadows. The more samples, the less noisy the shadows but the longer it will take to render. The total amount of light sampling depends as well on the anti-aliasing settings, as explained in [this section](#).



Example of Meshlights, rendered with Bidirectional.

Related articles:

- [First anti-aliasing pass.](#)
- [Glossy reflection sampling.](#)

Other area light types:

- [Spherelight.](#)
- [Arealight.](#)

Lights.

Lights settings are mostly controlled by the YafaRay UI. In Blender, the only actions required are placing lights and choosing a Blender light type. Arealight size and spot beam parameters are controlled in Blender panels though. In YafaRay, lighting power is controlled by a couple of settings (*Color* and *Power*) available in the python UI for every light type, therefore the Blender *Distance* and *Energy* buttons don't have any effect at all on YafaRay.

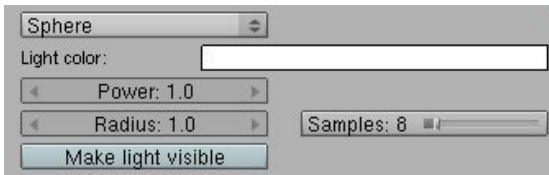
Once a light is placed and its type defined in Blender, select it and press *Object/Light/Camera* main button. The python UI will automatically change to show specific settings for the type of light selected. If you want to change a light to another type, you have to change it in Blender first.

Point and Sphere.

When you create or select a *Lamp* light in the Blender *3DWindows*, the python UI will let you choose between two options to do this kind of job, Point and Sphere. A Point light is a

typical omni directional point light source as in Blender Internal with hard shadows, while a Sphere light is a spherical area light source which can produce soft shadows.

The *Sphere* light settings are:



- *Light color* rectangle: opens a Color Picker.
- *Power*: intensity multiplier for light color.
- *Radius*: sets the radius of the Sphere light in Blender units.
- *Make light visible*: area light gets rendered visibly.
- *Samples*: defines the amount of samples taken to calculate the soft shadows. The more samples, the less noisy the shadows but the longer it will take to render. The total amount of light sampling depends as well on the anti-aliasing settings, as explained in [this section](#).



Comparison between point light (left) and spherelight (right). Notice the different shadows.

Related articles:

- [First anti-aliasing pass.](#)
- [Glossy reflection sampling.](#)

Other area light types:

- [Arealight.](#)
- [Meshlight.](#)

Directional and Sun.

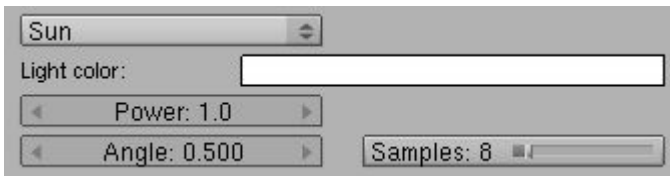
When you create or select a Sun light in the Blender 3DWindows, the YafaRay UI will let you choose between two kind of lights to do this kind of job, Directional and Sun.

Directional light is a traditional sun light model which produces parallel rays and hard-edged shadows.



- *Light color* rectangle: opens a Color Picker.
- *Power*: intensity multiplier for light color.
- *Infinite*: if enabled, area covered by the directional light is infinite. If disabled, light fills a semi-infinite cylinder.
- *Radius*: if infinite is disabled, radius of the semi-infinite cylinder for directional light.

The new YafaRay *Sun* light is a more advanced concept and will help us to get blurred-edged shadows when the shadow itself gets away from the casting object, as in real life. The angle button sets the visible area of the sun. Real Sun is visible in a cone angle of about $0,5^\circ$. A bigger angle mean a bigger sun, as well as softer shadows, which could be interesting for dawn or sunset scenes. A very big angle can be used to simulate sun light filtered by an overcast sky.



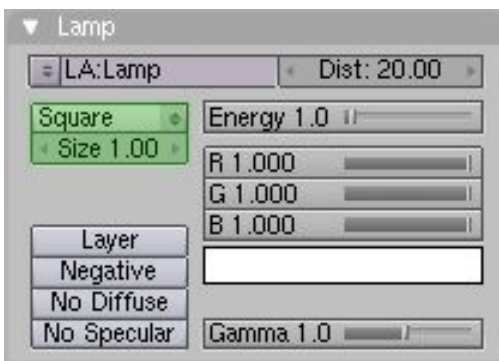
- *Light color* rectangle: opens a Color Picker.
- *Power*: intensity multiplier for light color.
- *Angle*: visible size of the sun light. Affects shadows.
- *Samples*: it defines the amount of samples taken to calculate the soft shadows. The more samples, the less noisy the shadows but the longer it will take to render. The total amount of light sampling depends as well on the anti-aliasing settings, as explained in [this section](#).



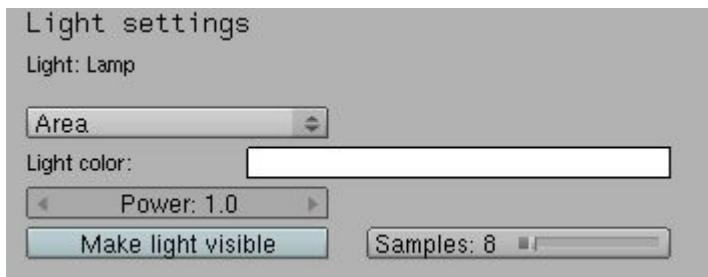
Comparison between Directional (left) and Sun (right). Notice how, with a Sun light, shadows get blurred as the distance with the casting object increases.

Area.

Arealight is a area light type that can produce soft shadows and its shape can be seen in reflective surfaces. The arealight shadows need to be sampled several times and interpolated to reduce noise. This type of light takes more time to be computed in contrast with point light types such as spot and point.



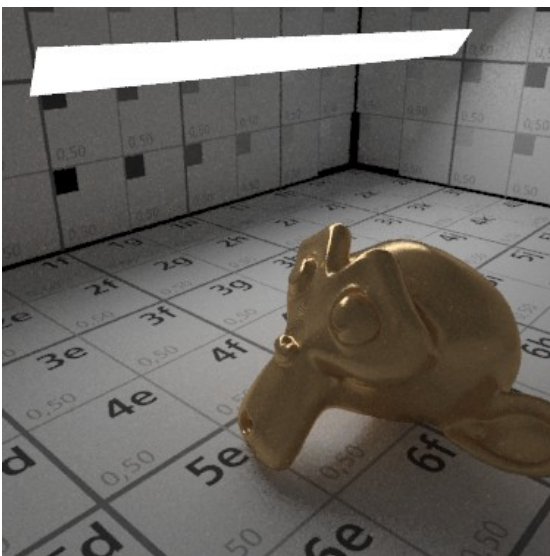
The area light size is controlled by the *Size* setting in the Blender Lamp panel. The rectangular shape option in that panel is not supported, although you can get a YafaRay rectangular arealight by *scaling on axis* in the *3DWindows*.



When the *Make light visible* button is enabled, a rectangle in the size of the area light is generated, so that the area light gets rendered visibly. More lighting power is added as well to the scene.

When photon mapping is enabled, arealights cast photons. The *Make light visible option* only makes the arealight visible from the camera and in reflective surfaces. When pathtracing is used, the *Make light visible* option also creates caustics paths, although there exist an option to not trace caustics paths with path tracing as they tend to be extremely noisy ('none' option in Pathtracing settings, see 5.1.1).

- *Light color* rectangle: opens a Color Picker.
- *Power*: intensity multiplier for arealight color.
- *Make light visible*: area light gets rendered visibly. In path tracing, it shoots caustic path rays.
- *Samples*: defines the amount of samples taken to calculate the soft shadows. The more samples, the less noisy the shadows but the longer it will take to render. The total amount of light sampling depends as well on the anti-aliasing settings, as explained in [this section](#).



Example of a rectangular visible arealight.

Related articles:

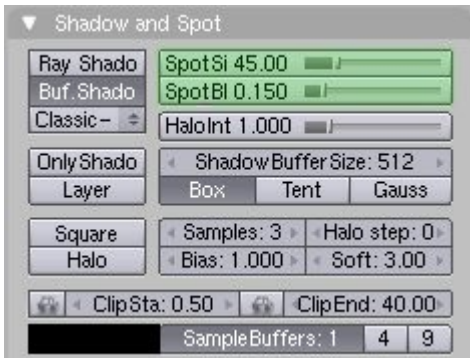
- [First anti-aliasing pass.](#)
- [Glossy reflection sampling.](#)

Other area light types:

- [Spherelight.](#)
- [Meshlight.](#)

Spot.

Spot is a common point light with directional properties. *Light color* and *Power* settings work like in the other light types. Beam properties are defined in Blender panels (light select. & F5) by *SpotSi* and *SpotBl* sliders:



Camera settings.

First a camera should be created or exist in your scene. Lens angle and ortho scale must be configured in the Blender *Camera* panel (F9). Then select the camera and press *Object/Light/Camera* main button. The python UI will automatically change to show specific camera settings. There are four camera types in YafaRay: Architect, Angular, Orthogonal and Perspective.

Architect/ DOF.

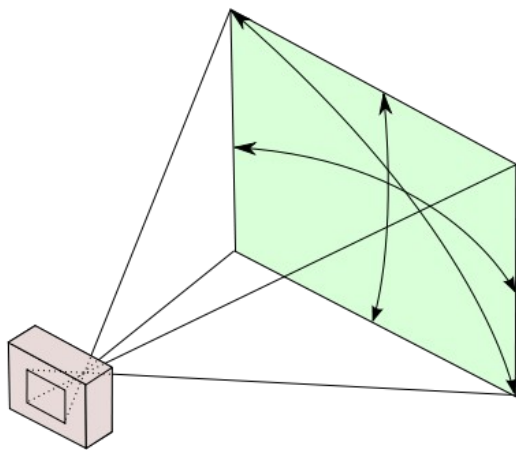
This camera type works like a Perspective camera type, the only difference is that the vertical component of the perspective effect is neglected, so scene vertical lines are not convergent. Depth of Field settings are available for this camera type as well. DOF settings are explained in the [Perspective/DOF](#) section.



Comparison between Perspective (left) and Architect camera (right). Notice the lack of vertical convergence in the Architect camera. Scene by Kronos

Angular.

A camera type useful to produce up to 180 degree panoramas, like the ones produced by fish eye cameras. Alternatively this camera type can be used to create a nice [perspective distortion](#) in your renders. Results with this camera type are controlled by two factors: distance to the subject and [angle of view](#).



Angle of view of a camera



- *Mirrored*: Mirrors the render in the x direction, as the reflections on a light probe.
- *Angle*: Horizontal opening angle of the camera, takes into account camera aspect ratio to calculate the vertical opening.
- *Circular*: Creates a circular render and shades areas outside.

- *Max Angle*: Diagonal opening of the circular render.



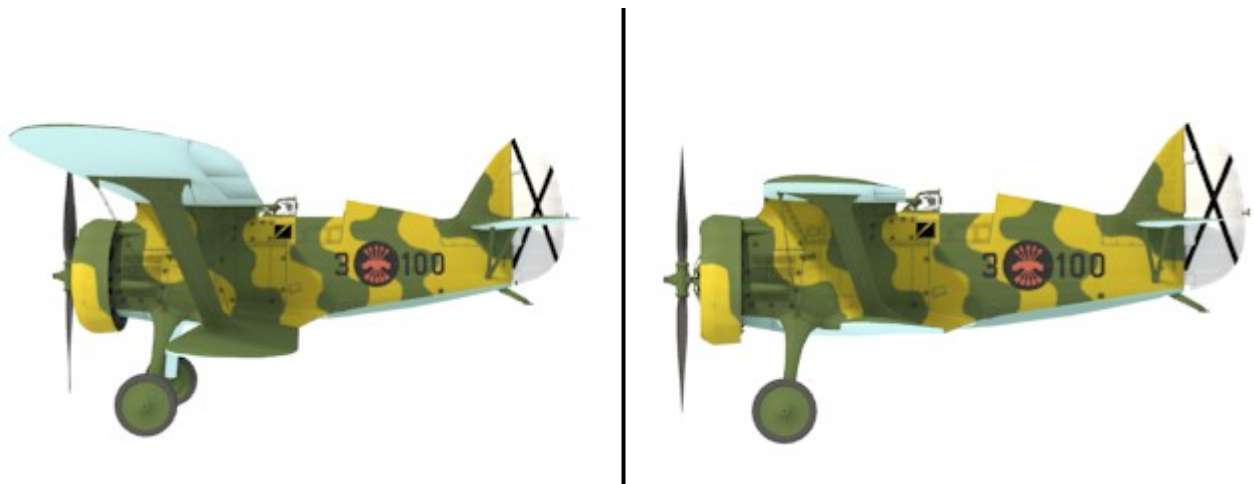
Example of an angular camera, used to create perspective distortion. Scene by **Gabich**.

Orthogonal.

A camera type that renders an orthographic (perpendicular) projection of the scene, without perspective effects. Camera only setting is:



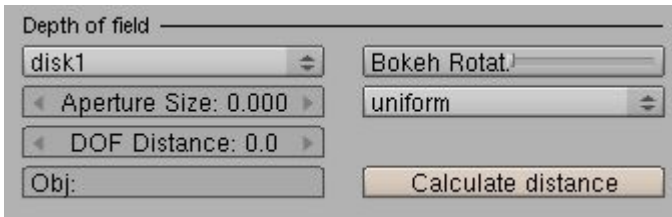
- **Scale**: Specify the scale of the ortho camera, to control camera zoom.



Comparison between perspective camera (left) and ortho camera (right).

Perspective / DOF.

Perspective is the standard camera mode that simulates a lenses photographic camera with perspective effects. All settings available for this camera type are used to enable and configure the depth of field (DOF) effect. The depth of field is the distance that objects appear in focus. DOF settings are:



- **Bokeh type:** controls the shape of out of focus points when rendering with depth of field enabled (blur disk). This is mostly visible on very out of focus highlights in the image. There are currently seven types to choose from.
- **Bokeh rotation:** rotation of the blur disk.
- **Aperture:** The size of the aperture determines how blurred the out-of-focus objects will be. A rule of thumb is to keep it between 0.100 and 0.500 (0 disables DOF).
- **Bokeh bias:** controls the accentuation of the blur disk. Three types available, uniform, center or edge, with uniform the default.
- **DOF distance:** set the focal point in which objects will be in focus.
- **Obj:** Enter the name of the Blender object that should be the focal point.
- **Calculate distance:** it calculates the distance between the object entered in Obj and the camera, and writes this value in the DOF distance button.

The **DOF** effect depends also on the render anti aliasing settings to get a nice blurred effect. First of all it is recommended to lower AA threshold a bit, but not set it to totally zero. Setting a high number of AA passes is also not really going to make all that much difference, the main smoothness factor that makes the most difference is really the amount of AA samples. A single pass with a high number of samples may be sufficient.

Texture Input.

Table of Contents:

- [Texture mapping \(introduction\)](#)
- [Image Texture Input](#)
- [Object Mapping](#)
- [Stencil](#)
- [Blending modes](#)
- [Procedural textures](#)

- [Noise Basis](#)
- [Clouds](#)
- [Marble](#)
- [Wood](#)
- [Musgrave](#)
- [Voronoi](#)
- [Distorted Noise](#)

Texture Mapping

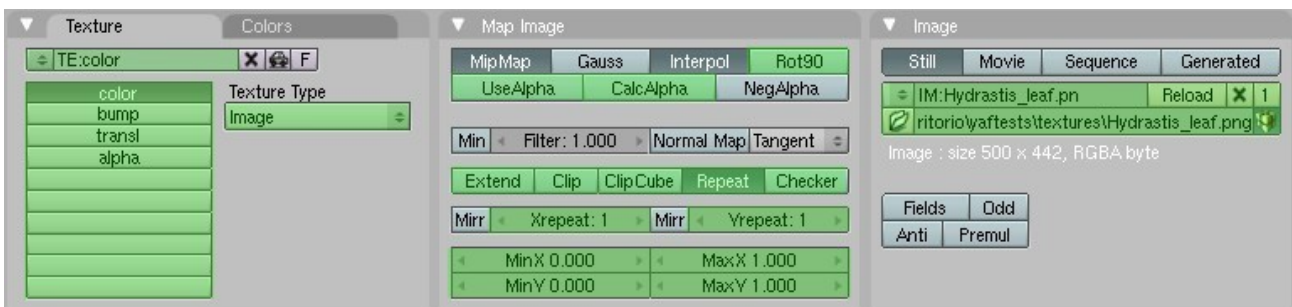
One of the main points of the old YafRay was the close support of Blender texturing features, and YafaRay keeps this tradition. YafaRay relies on Blender panels to setup texturing parameters. In general, YafaRay supports:

- Many of the settings in the Blender **Texture buttons** section (F6)
- Most of the settings in the 'Texture', 'Map input' and 'Map to' panels in the Blender **Material buttons** section (F5).
- Some of the Blender procedural texture types and their respective settings.
- Multiple texture channels for a material.

For a detailed explanation read on in the next sections.

Image Texture Input

Texture input is done by selecting *Image* as a Texture Type in the 'Texture' panel and loading the image in the 'Image' panel. You can use more than one texture channel in the 'Texture' panel, each of them doing different jobs. YafaRay supported formats are tga, jpeg, png, exr, hdr. This is an overview of supported settings in the Texture buttons section (F6), in green color:



- *Rot90*: rotates the Image 90 degrees counterclockwise.
- *UseAlpha*: Renders the texture alpha channel as transparent (if exists), white otherwise.
- *CalcAlpha*: transform RGB values into amount of transparency, for mapping transparency or translucency in the shinydiffuse material.
- *Extend*: the colour of the edge is extended outside the image.
- *Clip*: an alpha value of 0.0 is returned outside the image. This allows you to 'paste' a small logo on a large object.
- *Clipcube*: outside a cube-shaped area around the image, an alpha value of 0.0 is returned.
- *Repeat*: The image is repeated horizontally and vertically as often as set in Xrepeat and Yrepeat
- *Checker*: a checkerboard is made. Mortar governs the distance between the checkers.
- *Xrepeat, Yrepeat*: sets a repetition multiplier in the corresponding direction.
- *Mix X, Y - Max X, Y*: Use these to crop, or choose a portion of the texture.

Note: YafaRay can not load Blender encapsulated textures. If your Blender file has got encapsulated data, you must unpack first (File>External Data>Unpack). YafaRay will load textures from the created 'textures' folder.

During the rendering process, the Blender back console shows up messages when a texture image can't be loaded by YafaRay. Pay attention to them!

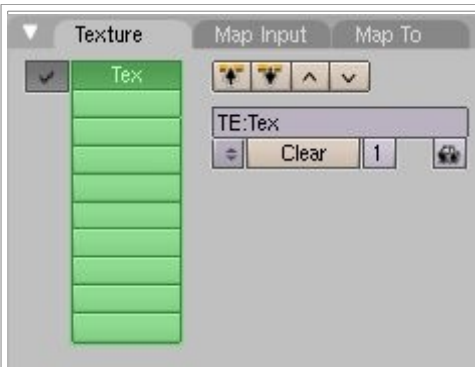
If your file is unpacked but YafaRay can't load textures, be sure that the path selected in the 'Image' panel points to the actual location of the file in your HD.

Object Mapping

In the Blender **Material buttons** section (F5), YafaRay supports only settings in the 'Texture', 'Map input' and 'Map to' panels. Settings in the other panels are irrelevant to YafaRay. In general, YafaRay supports:

- Multiple texture channels
- Some of the available coordinates for mapping.
- Flat, Cube, Tube and Sphere projection.
- Coordinates offset, scaling and transformation.
- Texture Modulation modes, depending on material type.
- Inverse Modulation of modes supported.
- Complementary modulation settings.
- Stencil.
- Blending modes.

Supported settings in each Blender panel are (in green color):



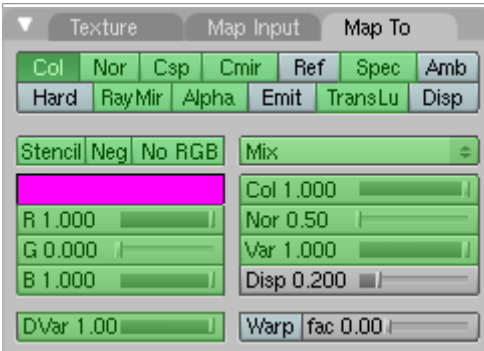
Texture channels: YafaRay supports multiple texture channels for a material. Blender principles are observed here: each channel can have its own individual mapping settings and the lower channel in the stack has got visibility priority over the upper one. For instance, a decal should be in a lower channel than a texture affecting the whole mesh. Also a Blending mode in a channel affects to channels above in the stack. More info about texture channels [here](#).



- *Global:* the texture uses the scene's Global 3D coordinates for mapping.
- *Object:* Uses a child Object's texture space as source of coordinates. The Object name must be specified in the text button on the right. Often used with Empty objects.
- *UV:* UV coordinates are used for texturing.
- *Orco:* the texture uses the object's local space.
- *Win:* the texture uses the window coordinates.
- *Flat, Cube, Tube, Sphere:* projection modes depending on the overall shape of the object. Use *Flat* for UV mapping.
- *ofsX, Y, Z:* moves the texture.
- *sizeX, Y, Z:* scales the texture.
- *X, Y, Z axis:* Re-orders the X, Y and Z coordinates.
- More info about map input [here](#).



The image above is an example of Orco mapping and **object** coordinates. Notice how the texture adapts to the objects position and size. Besides, each object is using a suitable projection mode, so each surface is perpendicularly mapped.



- *Modulation modes (Col, Nor, Csp, etc.):* the texture modulates the material property selected. Modes supported depend on the YafaRay material used (see it in the [materials modulation section](#)). Negative toggle-mode is supported. More than one modulation mode can be selected; in this case different material properties will be modulated with the same texture channel. Displacement mapping is only supported through a mesh modifier (more info [here](#)).
- *Blending modes:* How this channel interacts with other channels above it. Supported modes are mix, add, multiply, subtract, screen, difference, darken, lighten. More info [here](#).
- *Stencil:* the texture is used as a mask for all following textures. More info [here](#).
- *Neg:* produces a negative of the texture.
- *No RGB:* Converts a texture into a gradation, the two extremes of the gradation scale being Material main color and color set in *Color Swatch*.
- *Color Swatch:* A sort of a secondary color, to be mixed with Material diffuse color in procedural textures and in the *No RGB* feature.
- *Dvar:* default value for intensity-type textures.
- *Impact sliders (Col, Nor, Var, Disp):* the extent to which the texture affects the respective modulation mode.

Stencil

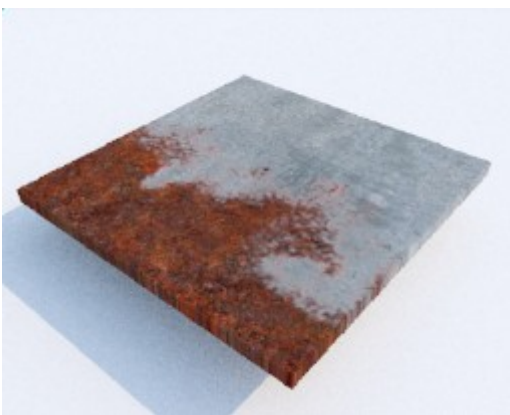
Stencil is a texturing feature that allows for mixing two texture channels in a controlled way, by using an intermediate mask texture. This feature can be used as well to set invisible parts of a texture, by using a stencil channel below. White value in the mask texture renders the above texture channel invisible.



The images above show an example of a stencil setup. On the left there are the textures used and on the right the texture channels arrangement.



The image above shows the 'Map to' panel of the **stencil** texture channel. Notice that the *Col*, *Stenci* and *No RGB* buttons are enabled.



The final result.

Related articles: [Material Modulation.](#)

Blending modes

Blending modes are used to determine how two texture channels are blended into each other. The default blend mode is Mix, which is used to hide the upper channel with whatever is present in the lower channel (remember that in Blender, textures have a higher visibility priority the lower they are in the channels stack). However, as each pixel in a texture has a numerical representation, a large number of mathematical ways to blend two channels is possible.

This feature can be used to add complexity and richness to your texturing works. Supported modes are mix, add, multiply, subtract, screen, difference, darken and lighten.



We are going to blend two textures from urbandirty.com, on the left is the **base** texture, on the right the **blend** texture. Please look at them carefully.



Base texture.



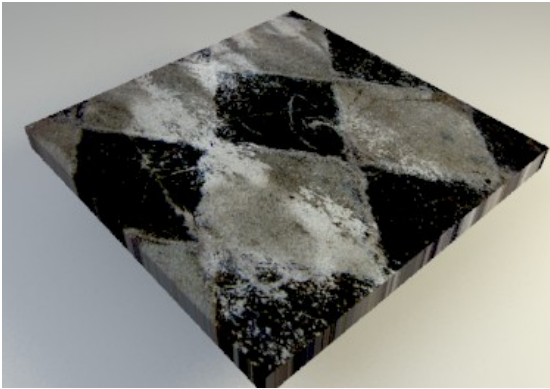
Blend texture.



Add

A very basic blending mode. It simply adds pixel values of one channel with the other ($a+b$). In case of values above 1, white is displayed. This mode is commutative (base and blend channels can be swapped)

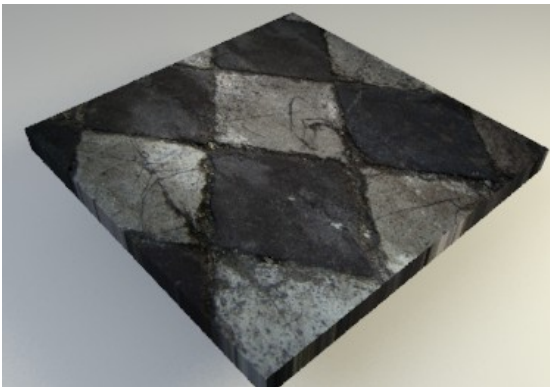
Notice how the black values in the blend texture don't affect the base texture, since $a+0=a$



Subtract

This blend mode simply subtracts pixel values of the base channel with the blend channel ($a-b$). In case of negative values, black is displayed.

Notice how black values in the blend texture don't affect the base texture since $a-0=a$, while white values produce a black result.

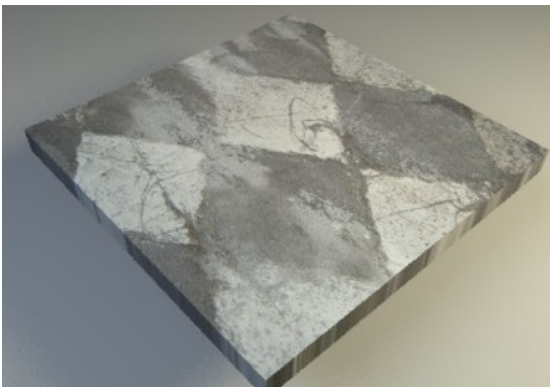


Multiply

Multiply is a basic blending mode for darkening areas of a texture.

Pixels from both channels are simply multiplied by each other. This returns a darker result than both input parameters, except if one of them equals to 1 (white). Completely white values do not change the base texture at all (and vice versa) - completely black values give a black result.

This mode is commutative (base and blend channels can be swapped)

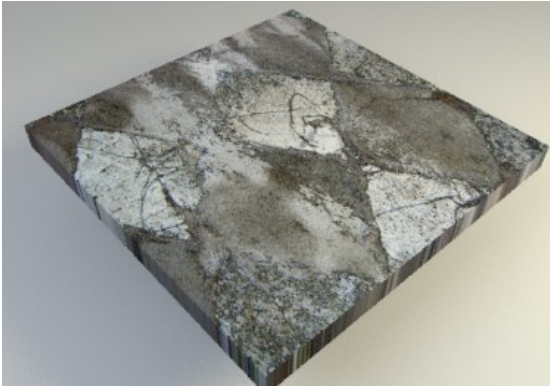


Screen

Screen is a basic blend mode for lightening areas of a texture.

It is the opposite of 'multiply' mode. It returns a brighter result than both input parameters in most cases, except if one of them equals 0 (black). Completely black values in the blend channel do not change the base at all (and vice versa) - completely white gives a white result.

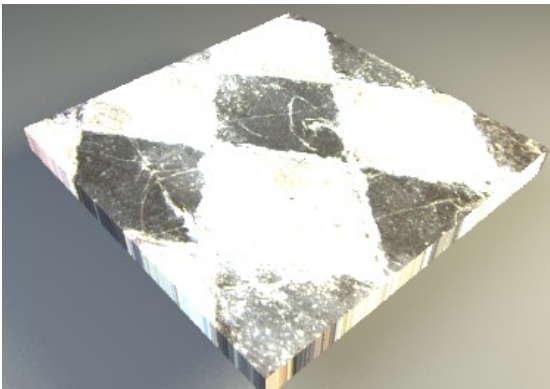
This mode is commutative, channels can be swapped.



Difference

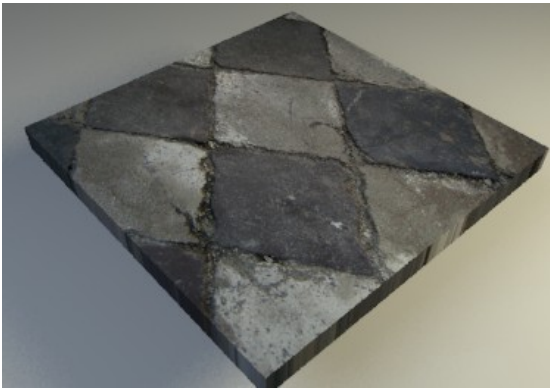
Subtracts either the blend texture from the base texture or the base texture from the blend texture, depending on which is brighter, to get always a positive result. Blending with black produces no change as $a-0=a$. Blending with white inverts the picture ($1-a$).

This kind of blending produces a lot of variation. This mode is commutative as well.



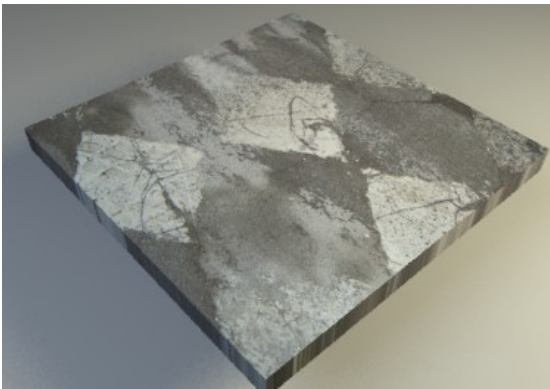
Divide

This mode simply divides pixel values of the base texture with the blend texture (a/b). The darker the blend texture is, the lighter the result. However, remember that it is not a linear function, so the result easily reaches the upper limit with regular factors, since $a/0=\infty$.



Darken

Pixels from both channels are compared to each other, the darkest one is taken.



Lighten

Pixels from both channels are compared to each other, the lightest one is taken.

Related articles: [Material Modulation.](#)

Procedural Textures

A **procedural texture** is a computer generated image produced by an algorithm, intended to create a realistic representation of natural elements such as wood, marble, granite, metal, stone, and others.

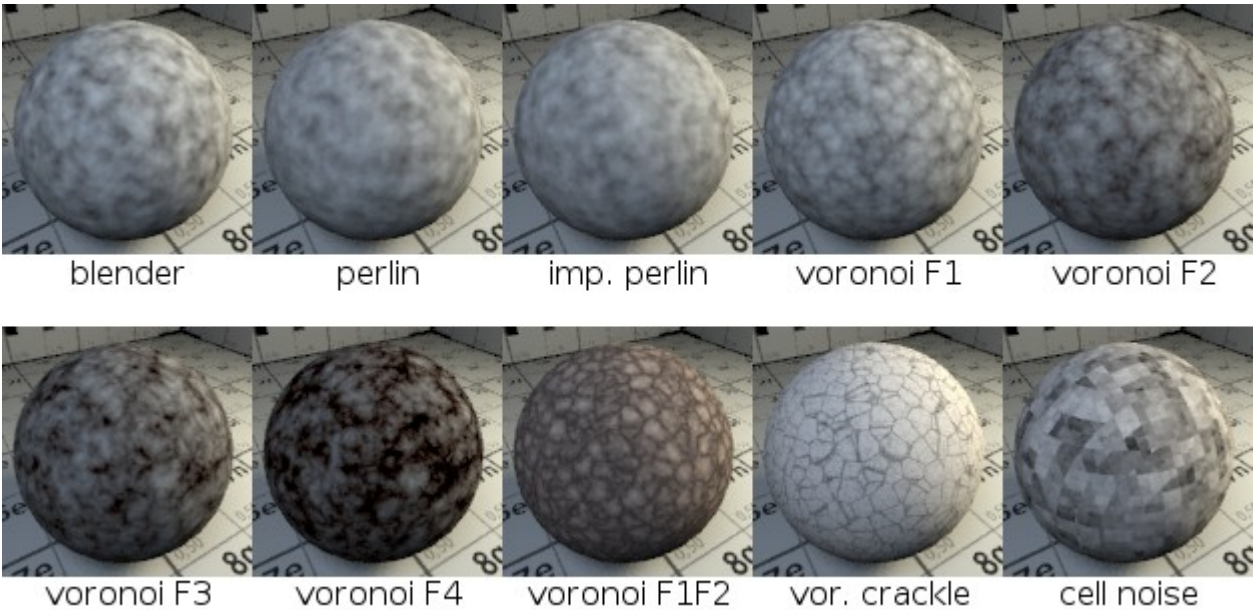
Procedurals can be used as a color texture or as an intensity texture in any of the mapping options available. Colors in a procedural texture are defined by two controls: *Color Swatch* in the Blender 'Map to' panel (F5) and material diffuse color in the YafaRay settings interface.

Procedural textures supported in the texture panel are: cloud, marble, wood, voronoi, musgrave and distorted noise. Besides, there is an additional "RGB-cube" procedural type only available by XML editing.

Texture Type
DistortedNoise
Voronoi
Musgrave
Plugin
Noise
Blend
Magic
Wood
Stucci
Marble
Clouds
EnvMap
Image
None

Noise Basis

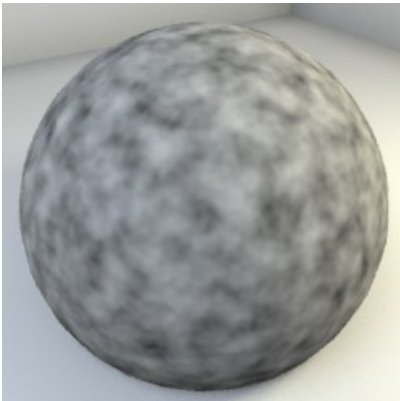
Noise Basis, which is a setting available for most procedural types, governs the structural appearance of the procedural texture. YafaRay supports all the *Noise Basis* types available in Blender, which are:



These are the settings supported in each procedural type:

Clouds

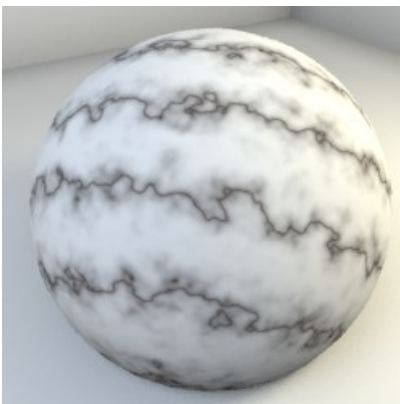
Useful for soft bump mapping of irregular surfaces.



- *Soft / Hard noise*: Noise strength.
- *Noise Size*: sets scaling for noise.
- *NoiseDepth*: Depth of calculation, the higher the more detail for noise.
- *Noise Basis*: basis used for turbulence, all modes supported

Marble

Useful for marble. ;)



- *Sharpness* (soft, sharp, sharper): Noise definition.
- *Soft / Hard noise*: Noise strength.
- *Shape* (sin, saw, tri): band type.
- *Noise Size*: sets scaling for noise.
- *Noise Depth*: Depth of calculation, the higher the more detail for noise.
- *Turbulence*: amount of turbulence for bands.
- *Noise Basis*: basis used for turbulence, all modes supported.

Wood

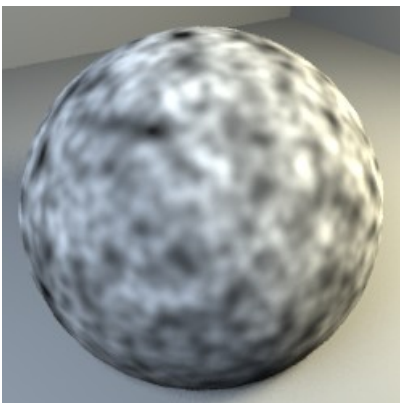
Useful for basic bands and noise with a structure.



- *Bands / BandNoise*: the procedural is arranged in bands.
- *Rings / RingNoise*: the procedural is arranged in rings.
- *Shape* (sin, saw, tri): band type.
- *Soft / Hard Noise*: Noise strength.
- *Noise Size*: sets scaling for noise.
- *Turbulence*: amount of turbulence for bands or rings.
- *Noise Basis*: basis used for turbulence, all modes supported.

Musgrave

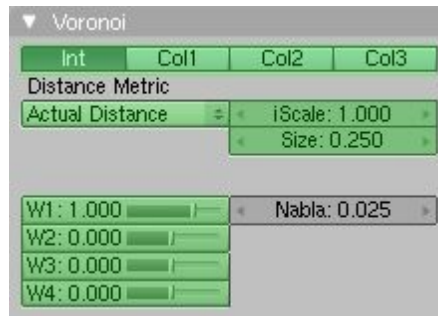
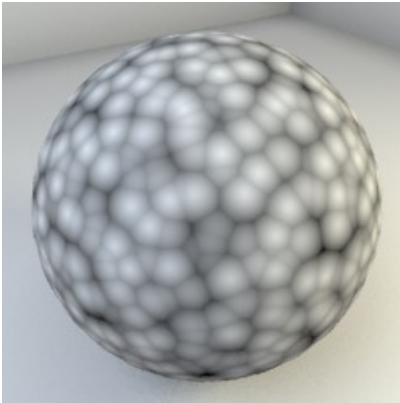
Very flexible. Useful for terrain and rock surfaces. The *Intensity Scale* control is useful to localise the result.



- *Musgrave type*: all modes supported except for *Hetero Terrain*.
- *H*: contrast between consecutive layers, for detail.
- *Lacunarity*: gap between layers.
- *Octaves*: layers of noise used.
- *Intensity Scale*: overall intensity of texture.
- *Noise Size*: sets scaling for noise.
- *Noise Basis*: basis used for turbulence, all modes supported.

Voronoi

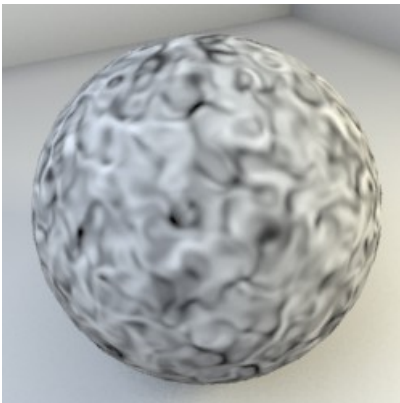
Voronoi is a procedural type with a cellular look. Very flexible. Useful for organic surfaces.



- *Cell type* (int, col1, col2, col3): color of the cells.
- *Distance Metric*: shape of the cells, all types supported.
- *Intensity Scale*: overall intensity of texture.
- *Size*: cell size.
- *Weight* (W1, W2, W3, W4): defines distances between each cell.

Distorted Noise

Complex and versatile.



- *Distance Amount*: amount of distortion.
- *Noise Size*: sets scaling for noise.
- *Distortion Noise*: basis used for distortion, all modes supported.
- *Noise Basis*: noise to distort, all modes supported.

Related articles:

- [Material Modulation.](#)
- [Volume types.](#)
- [The NoiseVolume Texture](#)

You can find some examples of different procedural combinations [here](#).

Materials.

Table of Content:

- [Multimaterial](#)
- [Material Settings](#)
- [Material Preview](#)
- [Glass](#)
- [Glossy](#)
- [Coated Glossy](#)
- [ShinyDiffuse](#)
 - [Diffuse reflection.](#)
 - [Mirror & Fresnel.](#)
 - [Transparency.](#)
 - [Translucency.](#)
 - [Emit.](#)
- [Blend Material](#)

Multimaterial



YafaRay supports Blender Multimaterial features, which means that you can assign different materials to different parts of a mesh, using sets of polygons. You must use Blender panels to set up 'Multimaterials' in YafaRay. You can find more information about multimaterials here:

http://wiki.blender.org/index.php/Doc:Manual/Materials/Multiple_Material

Alternatively you can use the *Blend material* to mix two materials in a mesh. In YafaRay, there are three ways to mix material properties in an object, which are:

- [Stencil](#), to mix two texture channels in a material, using a texture as a blending pattern.
- [Blend material](#), to mix two materials in a mesh, using either a blend factor or a texture.
- Multimaterial, to assign different materials to a mesh, using sets of polygons.

Material settings

To set up materials, the settings UI takes the Blender list of existing and assigned materials and applies YafaRay custom properties to them. The workflow is:

- Create & assign materials to objects using Blender panels (F5). Only texturing properties of materials will be taken into account.
- Assign YafaRay shading properties to the created materials, using the **'Material'** section in the YafaRay settings UI.

For instance, in the image below you can select from a list of four materials which has been previously created in Blender. *From active object* button will show YafaRay material settings for an object previously selected in the 3D scene.



Settings in Blender material panels (F5) are completely replaced by YafaRay material settings, even material colors. Blender Ramps are not supported. Multimaterial is supported.

There are four material types in YafaRay, with many possibilities for each of them to achieve advanced effects. They are glass, coated_glossy, glossy and shinydiffusemat. Also, there is a 'blend' type to mix two of them in a controlled way. This is a brief list of what YafaRay materials can be useful for:



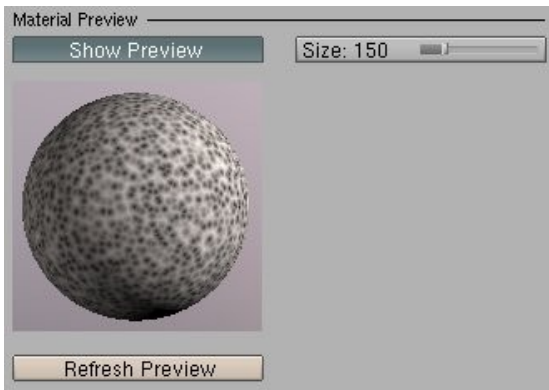
- Blend: blend two materials, using a blend factor or a texture map.
- Glass: glass, water, fake glass.
- Glossy: all kind of plastics, clean and polished metal, clean rough metal, car paint, finished wood, lacquered surfaces, painted surfaces, varnished wood, glaze and organic surfaces, materials with anisotropic reflections.
- Coated_glossy: car paint, lacquered surfaces.
- Shinydiffusemat: stone, rusted metal, concrete, fabric, paper, rough wood, curtains, emit surfaces, perfect mirror, materials with a basic transparency and alpha mapping with color-filtered shadows, translucent materials with color-filtered shadows, etc.

To acquire some background knowledge about the topics explained in the following sections, look at Neil Blevins's page:

http://www.neilblevins.com/cg_education/cg_education.htm

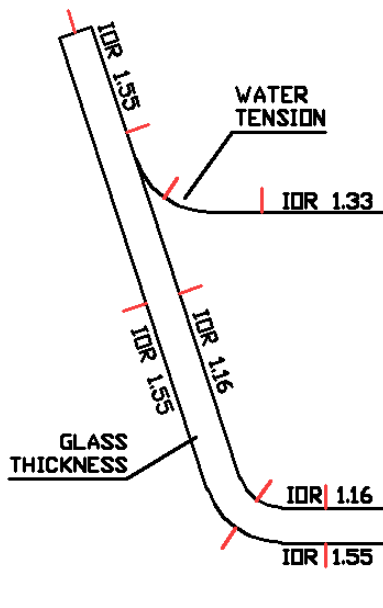
Material Preview

With this feature you can render a little preview of the material selected. The material preview will be rendered according with the material parameters chosen, and will take into account material mapped textures. The buttons available are:



- *Show Preview*: enables material preview.
- *Size*: Size of the material preview window, in pixels.
- *Refresh Preview*: Renders a material preview.

Glass



To render correct glass, it is important that glass objects follow realistic techniques for modelling, such as closed meshes with real thickness.

It is also important that your mesh normals point in the correct direction. Use your normal tools in Blender to control normal direction.

In real life, light refracts one time when passing from one medium to another, for instance from solid (glass) to liquid. In your 3D scene, use one mesh for each refractive event. That means: water meeting glass must be exactly one surface, and the relative refraction index of the "water meets glass" surface must be $\text{IOR_glass}/\text{IOR_water}$ (when normals are pointing into the water). Taking into account that glass IOR is 1.55 and water IOR is 1.33, a cross section of a glass of water should look like the glass cross section on the left (normals in red).

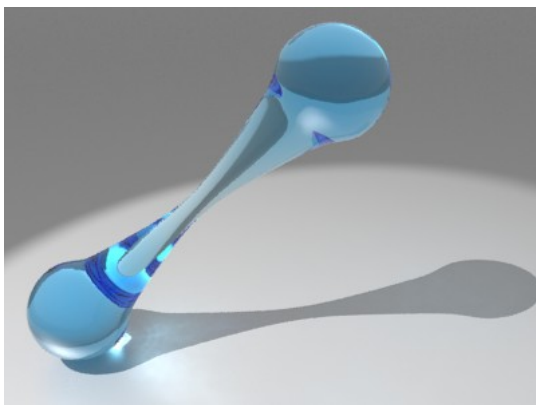
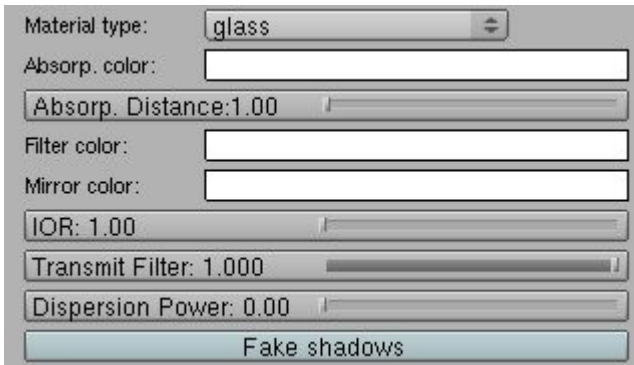
Rendering a number of consecutive transparent surfaces depends on recursive raydepth (General settings, *Settings* main section). If a transparent surface is rendered black, try increasing *Raydepth*.

Glass is basically refraction, reflection and absorption of incoming light. It means that the scene surrounding the glass has got an impact on the glass appearance, as well as the lighting setup. It means that glass will produce [caustics](#) if certain conditions are met, which are:

- There exist a light source and/or a background (IBL, Sunsky) which is shooting caustic rays.

- The glass material has got IOR>1.
- A global illumination method is used (pathtracing, photonmapping, bidirectional) or *Use caustics* is enabled in the *Direct Lighting* method.
- The caustic rays have enough depth to pass through all the transparent surfaces (Caustics depth).

Glass material mixes several independent concepts that can be used alone or combined with the other parameters to get different kinds of results. These concepts are:

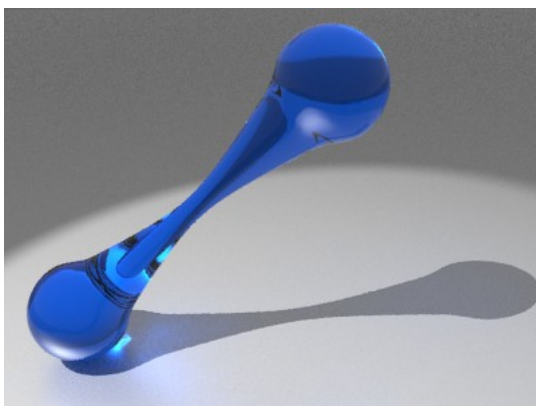


Absorption

Some of the incident light is absorbed by a transparent medium. The more distance light has to get through a medium, the more it gets absorbed. In a glass with different sections, the glass will get darker if the section is bigger.

Absorption also defines the color of the glass and the strength of the caustic effect. The more light is absorbed, the less light is transmitted. By using an absorption color, we also define the color of caustics. White disables absorption.

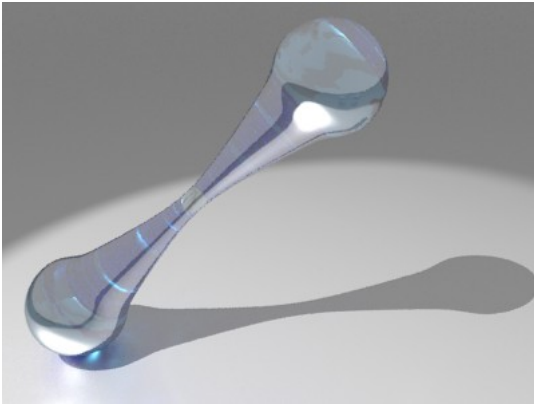
Filtering



Color is uniform regardless of glass section. The amount of transmitted light is also constant. You can use this setting instead of absorption if the glass section is uniform, for instance.

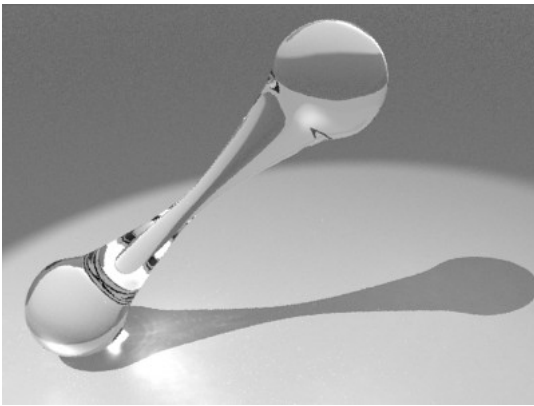
You need to use this setting to tint transparent shadows when *Fake Shadows* & *Transparent Shadows* are enabled.

Transmit Filter is a related setting which blends Absorption and Filtering. When *Transmit Filter* equals 1, Filtering is shown. When *Transmit Filter* equals 0, Absorption is shown (if enabled).



Reflection (Mirror)

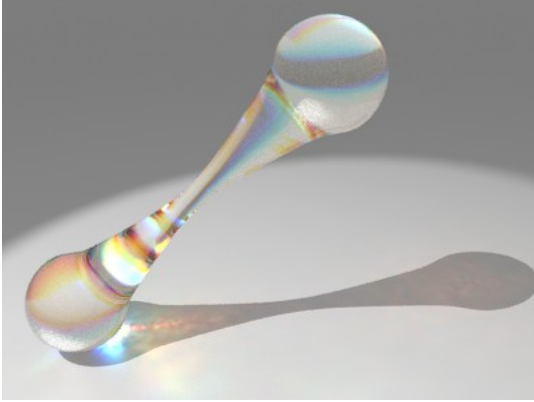
Some of the light is reflected. Amount of reflection depends on the index of refraction (IOR). The higher the index, the more reflective the glass is. It produces reflective caustics.



Refraction (IOR)

Refraction is the change in direction when light waves travel from a medium with a given refractive index to a medium with another. Refraction produces [caustics](#). Some materials index are:

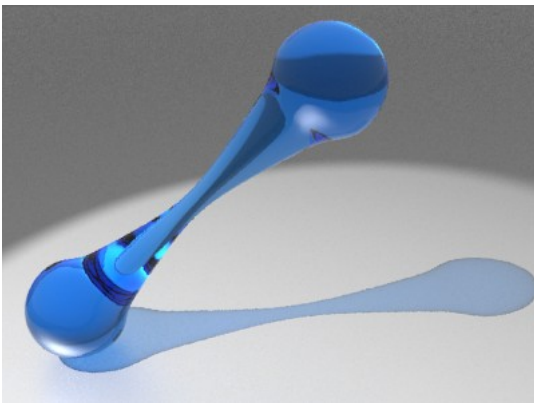
- Ice: 1.31
- Water: 1.33
- Clear plastic: 1.40
- Standard glass: 1.52
- Amber: 1.55
- Diamond: 2.42



Dispersion

Dispersion causes the spatial separation of a white light into components of different wavelengths (different colors).

When Path tracing is used, dispersion noise depends on path tracing samples, the more the samples the lesser the noise.



Fake Shadows

Not all light tracing methods are optimised to render caustics. This is the case of Direct lighting or Path tracing for instance. When fake glass is enabled, raytracing shadows rays get through this object when looking for light sources, and colored transparent shadows are calculated based on *Filtering* values. Notice the glass shadow, compared with the examples above.

The *Transparent Shadows* button in the *Settings* main section must be enabled too for this feature to work. *Filter color* controls color of the transparent shadows.

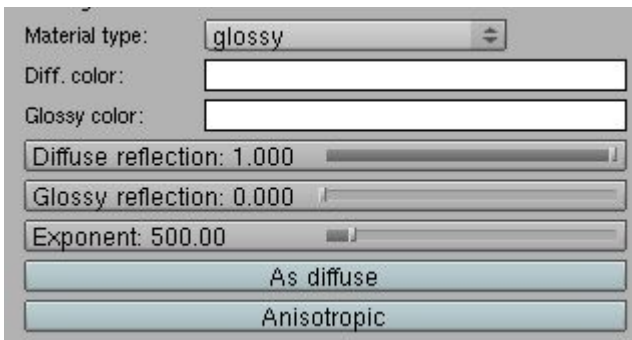
Related articles:

- [Ray Depth.](#)
- [Shadow Depth and Transparent shadows.](#)
- [Caustic Photon Map.](#)
- [Rendering Pathtracing caustic component.](#)
- [Photon Mapping.](#)

Glossy

A glossy reflection means that tiny random bumps on the surface of the material cause the reflection to be blurry. In fact there is a wide range of materials with such a reflection. YafaRay glossy material can be useful for all kinds of finished surfaces such as plastics, polished metal, car paint, finished wood, lacquered surfaces, painted surfaces, varnished wood, glaze, organic materials, etc. The glossy effect can be reinforced by using a fine bump map, or by mapping glossy reflection with a fine texture.

The main concepts of the glossy shader are:



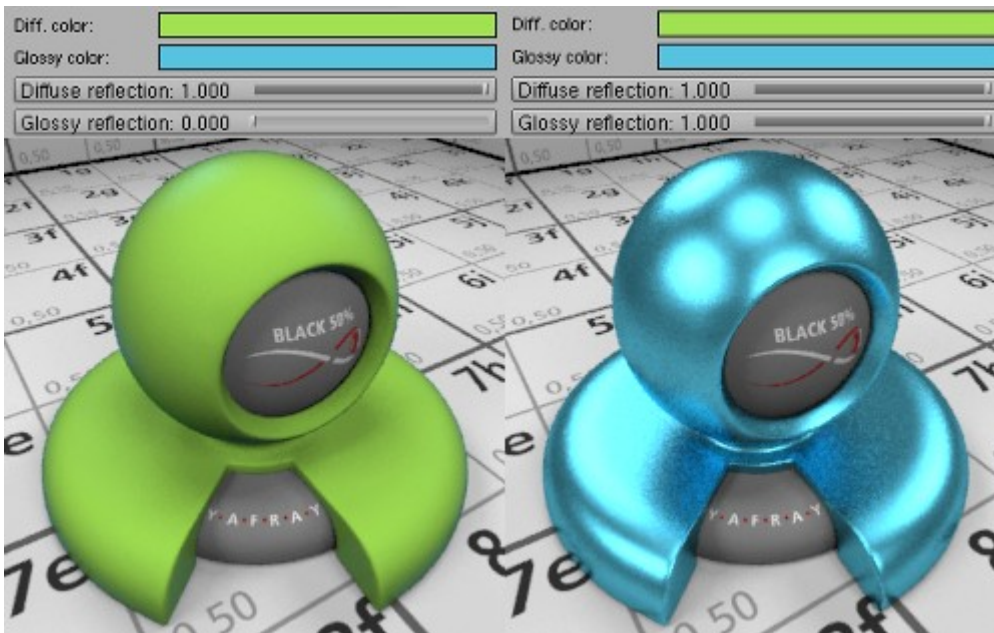
Diffuse and Glossy colors

A Glossy material has got two colors, diffuse and glossy. *Glossy reflection* parameter, apart from controlling the reflection strength, should be understood as a blend factor between the diffuse and the glossy color. The more reflective, the less diffuse. As stated in Siggraph 96 course notes book #30 Pixel Cinematography: A lighting approach for Computer Graphics...

"When light hits an object, the energy is reflected as one of two components; the specular component (the shiny highlight) and the diffuse (the color of the object). The relationship of these two components is what defines what kind of material the object is. These two kinds of energy make up the 100% of light reflected off an object. If 95% of it is diffuse energy, then the remaining 5% is specular energy. When the specularity increases, the diffuse component drops, and vice versa. A ping pong ball is considered to be a very diffuse object, with very little specularity and lots of diffuse, and a mirror is thought of as

having a very high specularity, and almost no diffuse."

When *Glossy reflection* is zero, you will see mostly the diffuse color with a bit of rim glossy color (image on the left). When *Glossy reflection* value is 1, you will see only the glossy color (image on the right). Notice **Glossy reflection** slider in both images:



Remember that coloured reflections is a particular feature of conductive materials (gold, copper), while non-conductive have got white colored reflections. So *Glossy color* of non-conductive materials like plastic should be white. **Diffuse Reflection** value is just a diffuse color multiplying factor.

Related articles:

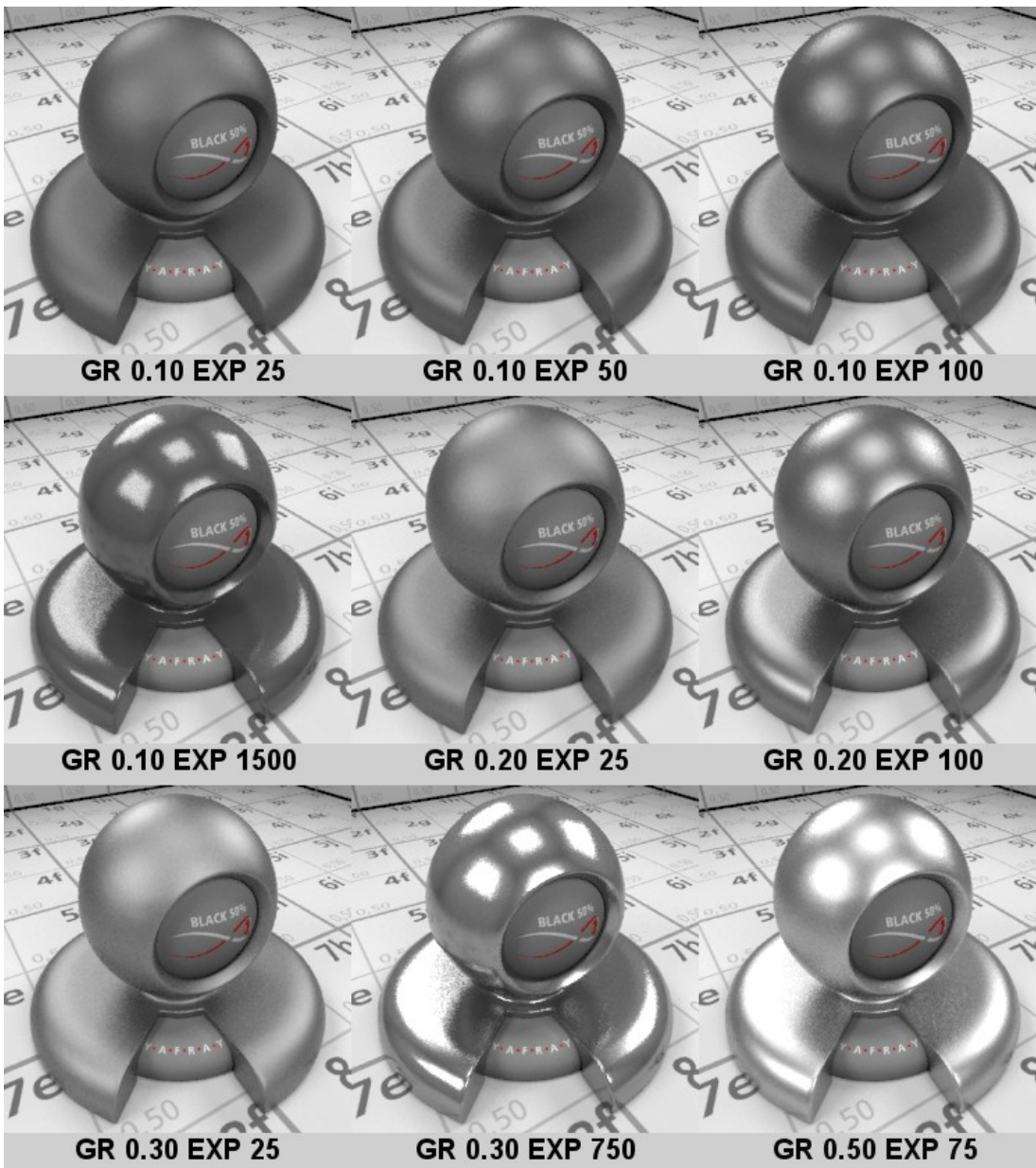
- [Diffuse color mapping.](#)
- [Specular color mapping.](#)

Glossy reflection and Exponent

Glossy reflection controls the strength of the reflection. The more reflective, the less diffuse.

Exponent controls blur of the glossy reflection; the higher the exponent, the sharper the reflection. Use values between 1 and 200 for plastics and higher values for metallic surfaces. Glossy reflection produces caustics.

Below there is an example of a material with different glossy reflection (GR) and exponent settings (EXP). Diffuse color is dark grey and glossy color is white:



Related article: [Specular intensity mapping.](#)

Glossy reflection sampling.

When these light sources are used in the scene, noise in the glossy reflection depends on:

- Area light (area light, sphere light, mesh light) samples.
- Sun light samples.
- HDR backgrounds (IBL, Darktide Sunsky, Sunsky) samples.
- AA samples, the higher the less noisy.

Below on the right, area lights are using 2 samples while on the left they are using 64 samples.



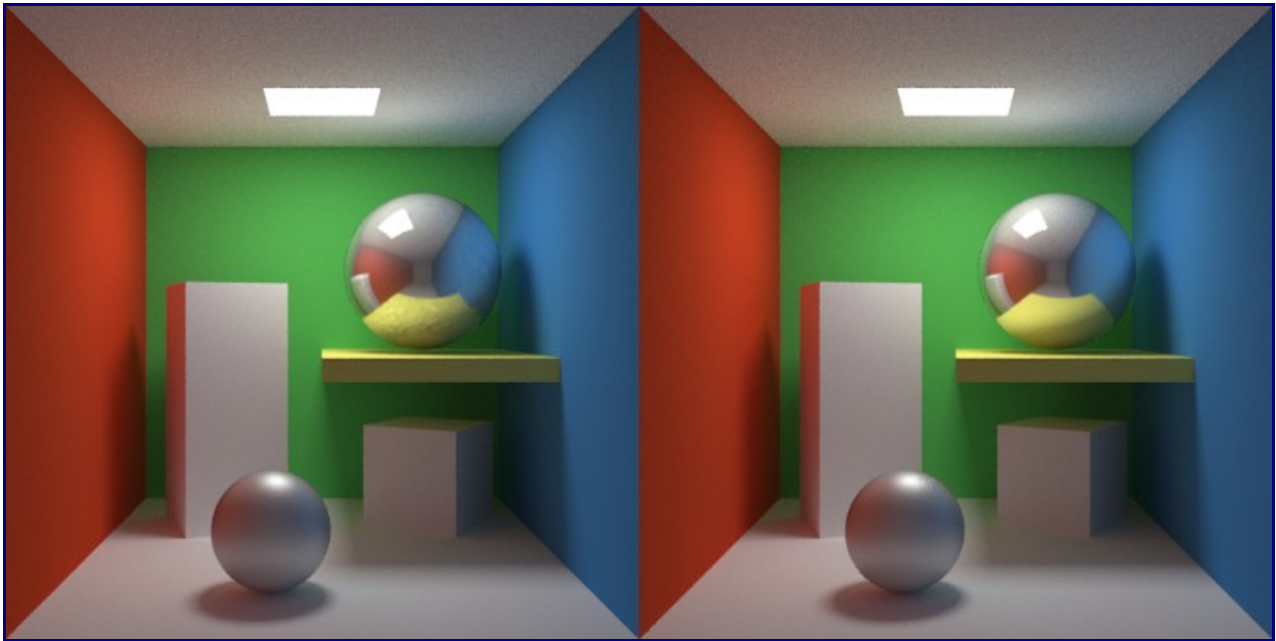
Related articles:

- [Arealight](#)
- [Sun light.](#)
- [First anti-aliasing pass.](#)
- [Background Settings.](#)

As Diffuse

When *As diffuse* is enabled in *Photon mapping*, *Glossy* surfaces will be treated as diffuse instead of specular. It means that the photon map will be used to calculate the surface reflections, which results in faster render times. In practice, this method is recommended only for glossy surfaces with a low *Exponent*, since the precision of glossy reflections calculated with photon mapping is always lower. Besides, *As diffuse* option enabled will be likely to produce flickering of the glossy reflection in animations.

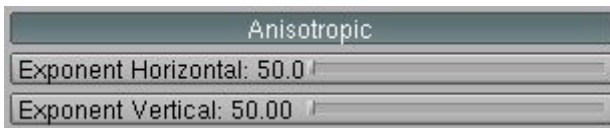
Both spheres in the left render have got *As diffuse* enabled and the render time is 166 seconds. Both spheres in the right render have got *As diffuse* disabled and the render time is 186 seconds. Notice the reflections!



Related articles:

- [Exponent.](#)
- [Photon Mapping.](#)

Anisotropic reflections



This material is useful to get [anisotropic reflections](#), which means that reflection is not equal in all directions. This kind of reflection happens when a defect in a reflective surface repeats with some regularity. When Anisotropic is enabled, the exponent value is divided into vertical and horizontal components. By using a different value for each component., the reflection will take an anisotropic oval shape. This effect can be reinforced by using a suitable bump map. When Anisotropic is enabled, the *Exponent* button is disabled. Horizontal and vertical direction of the anisotropic exponent depend on UV mapping coordinates. Below you have a comparison between anisotropic reflections (left) and default isotropic reflections (right).



Coated Glossy

Coated Glossy is basically a glossy material (see the previous section) with some kind of reflective coating layer on top. IOR is the setting that controls reflectivity of the coating top layer. This reflective layer can produce caustics. It is a good material for car paint. Below an example of different IOR values:



IOR 1.0

IOR 1.5



IOR 2.0

IOR 2.5

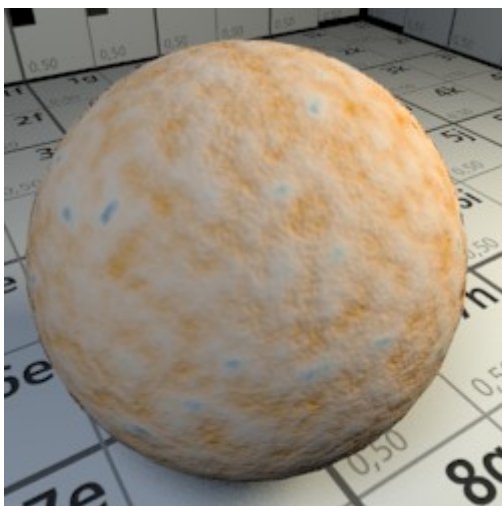
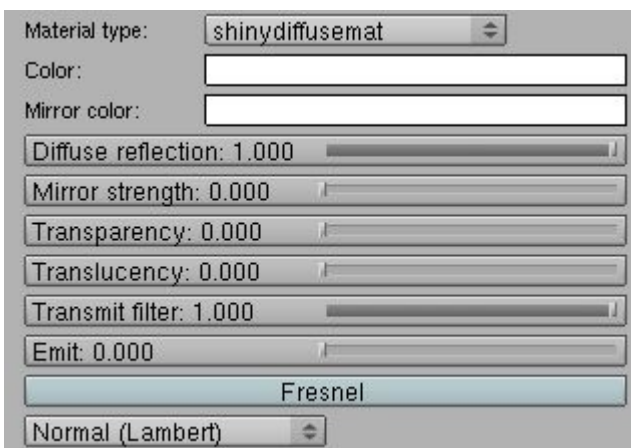
ShinyDiffuse

Shinydiffuse is a shader with many applications. It can be useful to get:

- Diffuse materials without any specular component.
- Perfect mirror reflection with or without Fresnel effect.
- Alpha mapping with shadows calculation derived from the map, for translucency and transparency effects.
- Translucency with color filtering.
- Transparency with color filtering.
- Emit surfaces.

For instance, this material can be used for rough stone, rusted metal, concrete, fabric, clay, asphalt, paper, rough wood, chrome balls, shiny plastics, basic car paint, curtains, leaves, billboards, etc.

The main concepts of the ShinyDiffuse shader are:



Diffuse reflection.

Diffuse reflection is the reflection of light from an uneven or granular surface such that the incident rays are randomly reflected and scattered in all directions. The amount of diffuse reflection is mainly controlled by the surface *Color*. YafaRay uses two models to render the diffuse component, which are Lambertian and Oren-Nayar.

Lambertian is a basic diffuse model with no view dependence. Brightness is constant from all viewing directions; only interaction between surface and light sources is modelled. It is a method valid only for very smooth matte surfaces, like paper, smooth plastic or polished wood.

Oren-Nayar is a view dependent, physically-based microfacet model for diffuse

reflections, which takes into account geometric optics and interaction at microfacet level produced by light sources. Oren-Nayar models the diffuse reflectance for rough surfaces more accurately than the Lambertian model. *Sigma* controls the roughness of the surface.

Diffuse Reflection value is just a diffuse color multiplying factor.



Real Image



Lambertian Model



Oren-Nayar Model

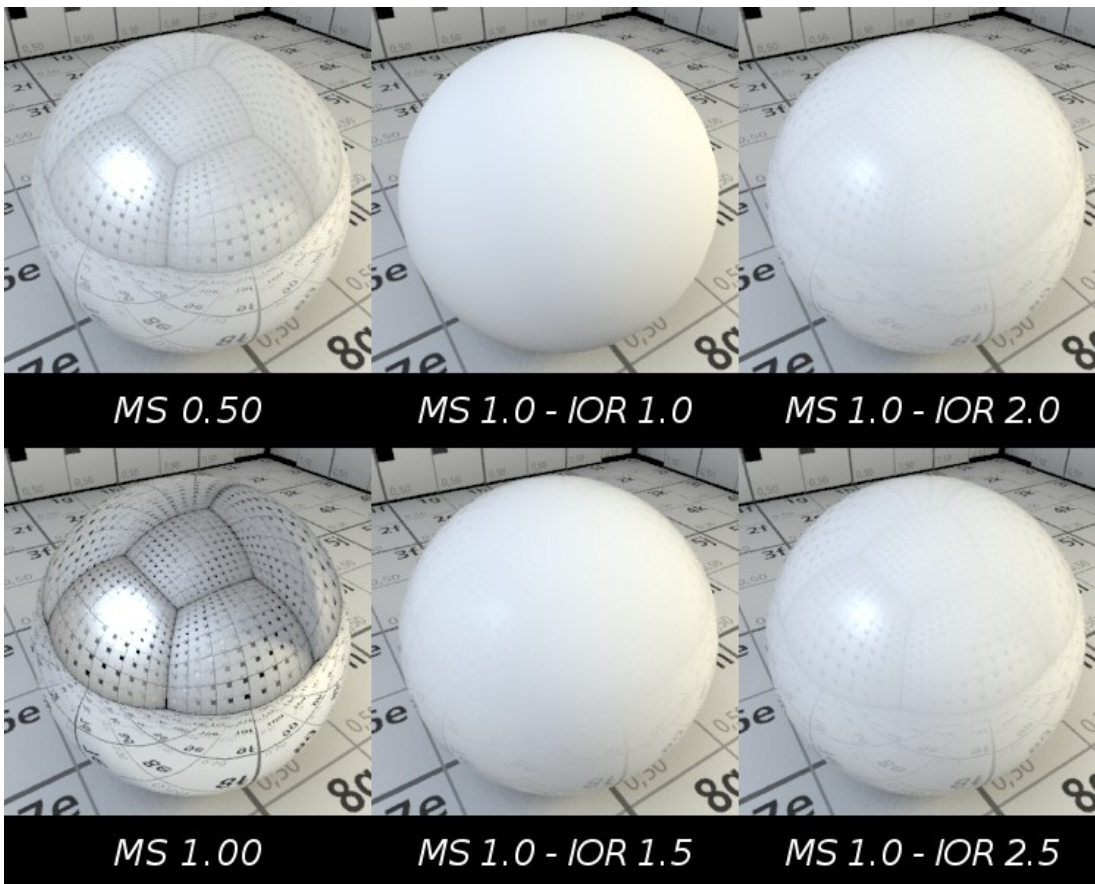
[GPL](#) Photograph of a comparison between a matte clay vase and its renderings with the **Lambertian** model and the **Oren-Nayar** model ([Source](#)). Some Sigma values, as per the [official project page for the Oren-Nayar model](#), are:

- Felt: 0.414686
- Rough plastic: 0.278057
- Leather: 0.179776
- Velvet: 0.751002
- Pebbles: 0.443289
- Plaster_b: 0.543788
- Rough paper: 0.311376
- Roof shingle: 0.819147
- Rug_b: 0.613889
- Sponge: 0.872413
- Wool: 0.978133
- Quarry tile: 0.360574
- Slate_b: 0.309590
- Human skin: 0.579386
- Brick_b: 0.275990
- Linen: 0.514593
- Cotton: 0.482679
- Stones: 1.107168
- Concrete_b: 0.308956
- Concrete_c: 0.461930
- Wood_b: 0.351271
- Tree bark: 0.293226

Related article: [Diffuse color mapping](#)

Mirror & Fresnel

Mirror produces pure specular reflection. **Mirror strength** is a blend factor between the specular and diffuse components, with their respective colors. The more specular, the less diffuse component will be shown. When **Fresnel** is enabled, the amount of specular reflection depends on how the viewer is oriented relative to the surface. Fresnel means that a surface is more reflective at grazing angles than at perpendicular ones. **IOR** controls the Fresnel reflection strength, the higher the more reflective at every angle. Mirror and Fresnel reflections can produce caustics. Below are several examples of Mirror and Fresnel, MS stands for *Mirror Strength*. White is being used for both diffuse and mirror color:



Related articles:

- [Specular color mapping.](#)
- [Specular intensity mapping.](#)
- [Ray Depth.](#)



Transparency

With this setting you can achieve a basic transparency effect, without refraction but with transparent shadows and color filtering.

Transparent shadows and color filtering means that light is filtered by the transparent surface, and gets coloured and diminished according to the surface properties. Transparency is basically a 'fake' feature intended for basic transparency and alpha mapping purposes, since it is a mappable feature. To get transparent shadows and color filtering, *Transparent Shadows* button must be enabled in the *Settings* main section.

Settings that affect this feature are:

- *Diffuse color Picker*: control diffuse color of the object, which results in color filtering.
- *Transparency*: Amount of transparency.
- *Transmit Filter*: color filtering strength. When it equals 0 there is no color filtering, therefore shadows are not tinted by the transparent surface.

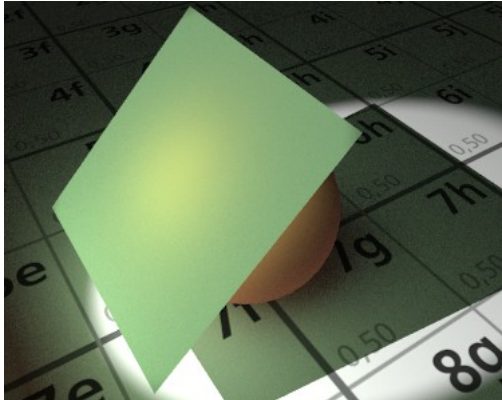
In *Settings* main section:

- *Raydepth*: for camera rays, to get through successive transparent events.
- *Transparent Shadows*: this option must be enabled to produce transparent shadows and color filtering. Technically speaking, it allows raytracing shadows rays to get through this mesh when looking for light sources.
- *Shadows Depth*: for raytracing shadow rays, to get through successive transparent surfaces.

With this feature you can emulate the Blender *OnlyCast* feature, which makes objects not to be rendered but to cast shadows only. Set the object transparency=1 while keeping *Transparent Shadows* button disabled.

Related articles:

- [Alpha intensity mapping.](#)
- [Ray Depth.](#)
- [Shadow Depth and Transparent shadows.](#)
- [Glass material.](#)



Translucency

With this setting you can achieve a basic 2D translucency effect with transparent shadows and color filtering. Translucent materials allow light to pass through them but only diffusely; you can not see through. Light is scattered after passing through the transparent surface.

Transparent shadows and color filtering means that light is filtered by the translucent surface, and it is coloured and diminished according to the surface properties. *Translucency* slider controls amount of translucency.

Scattered light with transparent shadows & color filtering only works with Global Illumination methods, which are Pathtracing, Bidirectional Pathtracing and Photon mapping.



Example of translucency, notice the curtains. 'My Room' by **ChojinDSL**.

Related article: [Alpha intensity mapping](#).



Emit

Amount of light a material emits, similar to the mesh light concept. Color of the emitted light is controlled by the *Diffuse color*. *Emit* slider controls strength of the emitted light.

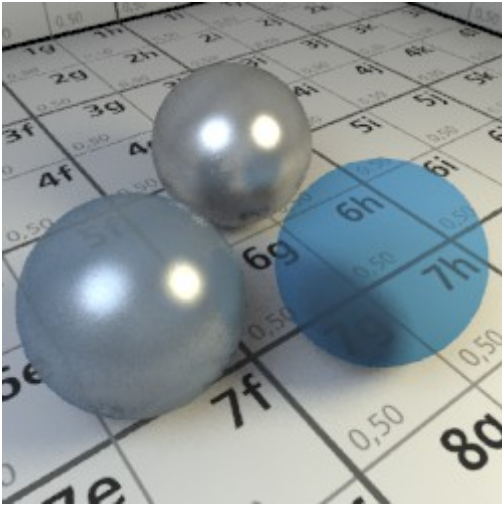
Emit objects can be used as diffuse light sources, but you will have to use it with path tracing or bidirectional path tracing lighting methods.

Related article: [Diffuse color mapping with emit value](#)

Blend Material

This feature takes two defined materials and mixes them into a third one. You need to define three materials then, two ones to mix and a third Blend material which is applied to the object to render. This feature can be useful to mix properties from two different materials, for instance glossy reflection blended with transparency. It can be useful as well to map two different parts of a mesh with different materials using a texture as a pattern, without resorting to multimaterial which depends on mesh polygons.

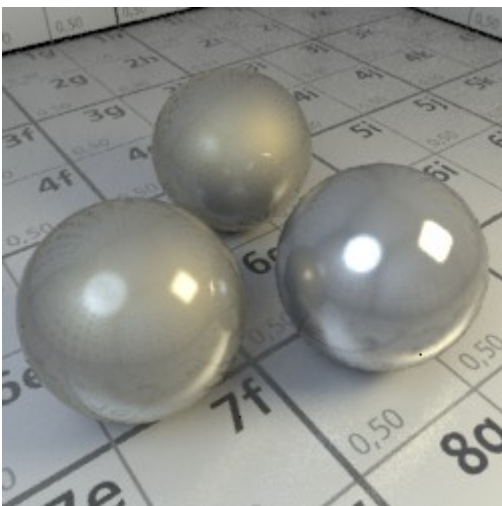
Blend value controls the percentage of each material in the final mix. A value of 0.50 means that each material contributes the same to the final look. These are examples of different blend materials; the sphere on the left has got the Blend material:



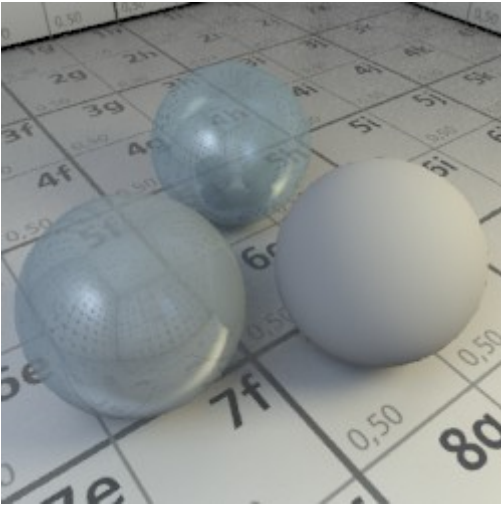
0.5 blend between a glossy material and a shinydiffuse with transparency=0.85.



0.50 blend between a glossy material and a shinydiffuse with mirror=0.80.



0.25 blend between two glossy materials. One material uses exponent=15, the other uses exponent = 1500.



0.25 blend between two shinydiffuse materials. One uses mirror=0.20 and transparency=0.85. The other is a completely diffuse material.

Related article: [Blend mapping](#)

Material modulation.

Table of Content:

- [Material Modulation introduction](#)
- [Diffuse color](#)
- [Specular color](#)
- [Specular intensity](#)
- [Alpha intensity](#)
- [Bump intensity](#)
- [Displacement mapping](#)
- [Blend mapping](#)

Material Modulation introduction.



APC from the movie Aliens, by **Gabich**. Example of texture mapping with YafaRay.

Some material properties can be modulated by a texture, instead of using a global value. This technique is called texture mapping. The texture used can be a RGB image or a procedural texture. In general, there are seven types of modulable settings in YafaRay materials, which are:

- **Diffuse color.** ShinyDiffuse, Glossy and Coated Glossy.
- **Specular color.** ShinyDiffuse, Glossy and Coated Glossy.
- **Specular intensity.** ShinyDiffuse, Glossy and Coated Glossy.
- **Alpha intensity.** ShinyDiffuse.
- **Bump intensity.** All materials.

- **Displacement mapping** through a mesh modifier. All materials.
- **Blend mapping**. Blend material.

For instance, you can use a texture to have different values of glossy reflection on a surface. Some of these modulation modes are available for all material types, like bump intensity. Others are available only for some of them, depending on the material features.

Negative toggle mode is supported in intensity-mapping types, so texture values are inverted.

Diffuse color.

There are several options when using a texture to map diffuse color. These options are:



- To map the diffuse color with a RGB texture, use the **COL** slot [4] in the 'Map to' panel. *Shinydiffuse*, *Glossy* and *Coated_glossy* materials support diffuse color mapping. The applied RGB texture will hide the material's own diffuse color [1] defined in the material settings. However, you can control texture opacity so it blends with the material diffuse color, by using *Col* slider [5].
- *Diffuse reflection* value [2] is just a diffuse color multiplying factor, affects textures as well.
- If a texture with alpha data is used (RGBA) and you want to render alpha as transparent (white otherwise), enable *UseAlpha* button [7] in the Blender Texture buttons (F6). The alpha channel will be rendered transparent and the underlying diffuse color [1] will be seen.
- You can use a procedural texture to map the diffuse color. Colors in a procedural texture are defined by two controls: *Color Swatch* [6] in the Blender 'Map to' panel (F5) and material diffuse color [1] in the YafaRay settings interface. More info about procedurals [here](#).
- When a diffuse color texture is used in a *ShinyDiffuse* material with *Emit* [3], the amount of light emission produced by that material will be defined by the texture brightness. At the moment, texture-based light emission only works with pathtracing.
- When a diffuse color texture is used in a *ShinyDiffuse* material with *Transparency*

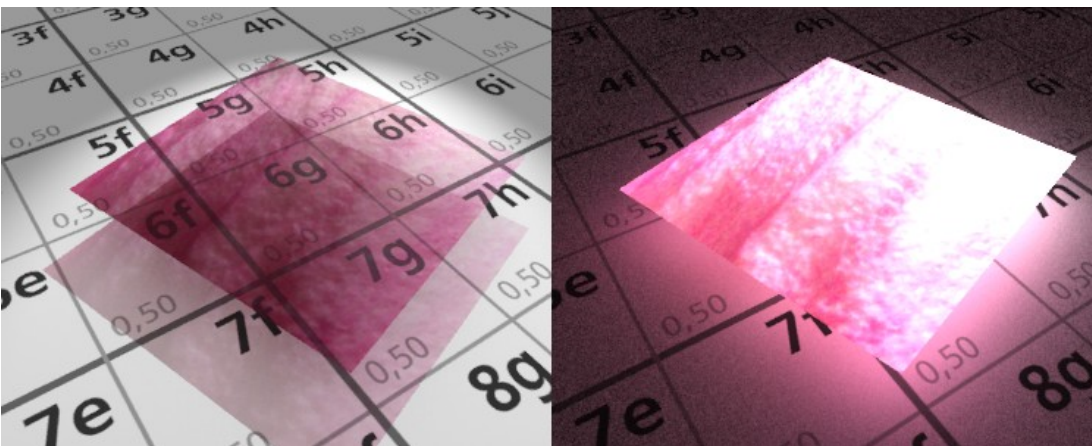
[8], the amount of transparency will be defined by the texture brightness. Enable *Transparent Shadows* in *Settings* section to get transparent shadows and color filtering from the texture.

- When a diffuse color texture is used in a ShinyDiffuse material with *Translucency* [8], the amount of translucency will be defined by the texture brightness. Enable *Transparent Shadows* in *Settings* section to get transparent shadows and color filtering from the texture. Translucent scattered light with transparent shadows & color filtering only works with Pathtracing, Bidirectional Pathtracing, and Photon mapping.

Below an example of diffuse color mapping in Shinydiffuse (left) and Glossy material (right). A bit of bump is applied in both cases.



Example of diffuse color mapping in Shinydiffuse material, with *Transparency* on the left and with *Emit* on the right.



Texture used:



Related articles:

- [Glossy material.](#)
- [Coated Glossy material.](#)
- [ShinyDiffuse material.](#)

Specular color.

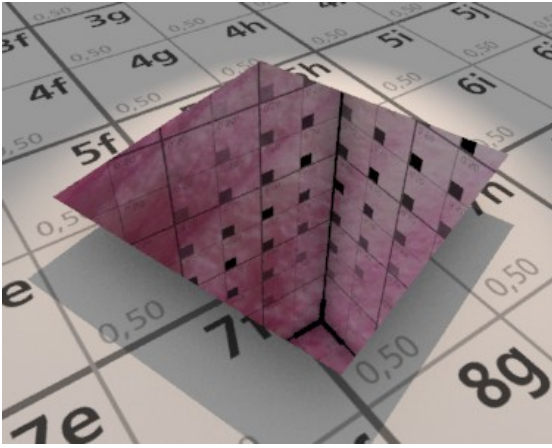
There are two kinds of mappable specular in YafaRay, *Mirror* in Shinydiffuse, and *Glossy* in Glossy material. Both are specular components that can use the uniform color defined in the settings UI, or they can be also mapped with a RGB texture. The applied texture will hide the material specular color (mirror or glossy) defined in the settings UI. However, you can control opacity of the texture so it blends with the material specular color, by using *Col* slider [3] in the Blender 'Map to' panel. You can use a procedural texture as well.

Remember that coloured reflections are a particular feature of conductive materials, while non-conductive have neutral-colored reflections (white).

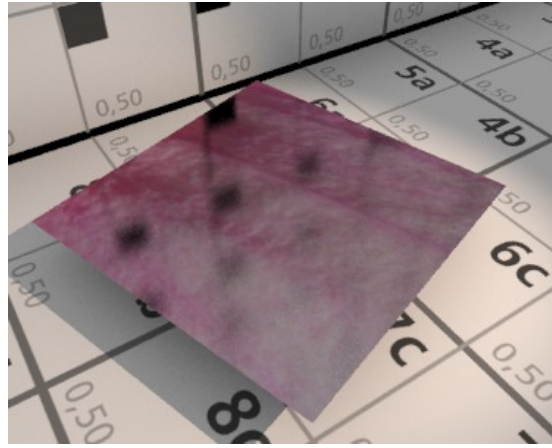


To map specular, ShinyDiffuse uses *Cmir* slot [2] while Glossy material uses *Csp* slot [1]. Blender *UseAlpha* button (F6) is supported when a RGBA texture is used, so the underlying specular color will be seen instead of white.

The images below are examples of specular color mapping. Both cases are using full specular strength:



Mirror color mapping.



Glossy color mapping.



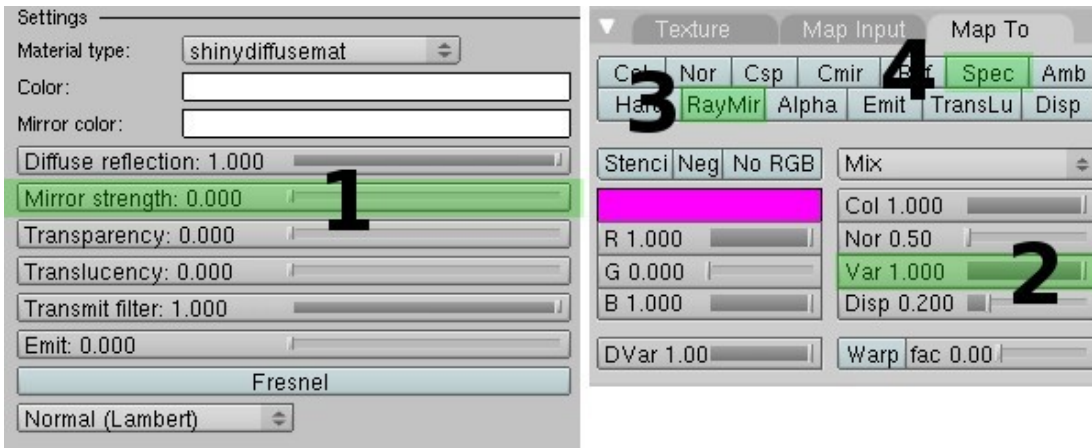
Texture used.

Related articles:

- [Glossy material.](#)
- [Shinydiffuse mirror material.](#)

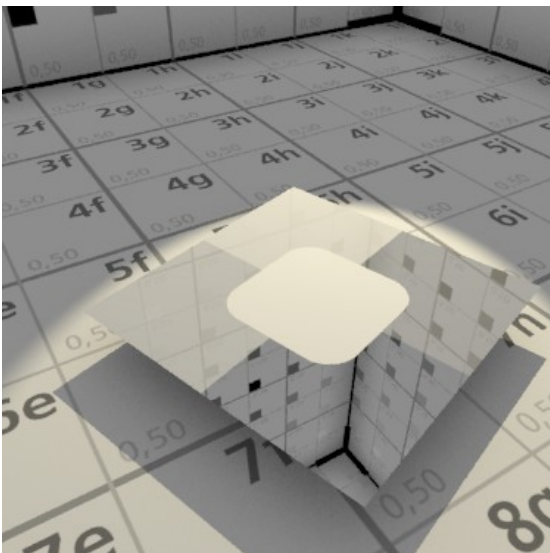
Specular Intensity.

You can use a texture to map the amount of specular reflection. There are two kinds of specular reflection in YafaRay: Mirror in Shinydiffuse and Glossy Reflection in Glossy material. Textures will be processed as a value scale. This means that if the RGB texture has got colors they will be transformed into tones of grey, so normally it is a wise idea to use desaturated textures to map intensity. This way users have more control over the final result. You can use a procedural texture as well.



The texture in fact will map in the range between the specular strength set in material setting [1] and *Var* [2]. For instance, if Mirror strength is 0.20, it means that black color in the texture will make the object to be at least 0.20 reflective. If *Var*=0.80, it means that white color in the texture will make the object to be as much as 0.80 reflective. The same happens with Glossy materials.

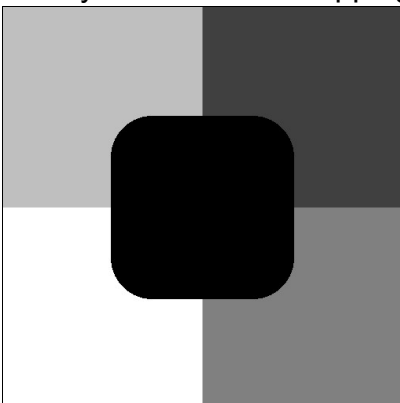
To map mirror strength in Shinydiffuse, use *RayMir* slot [3]. To map glossy reflection strength, use *Spec* slot [4]. Below are examples of specular mapping. Both cases are using strength=0 and *Var*=1:



Shinydiffuse mirror mapping.



Glossy reflection mapping.



Texture used.

Related articles:

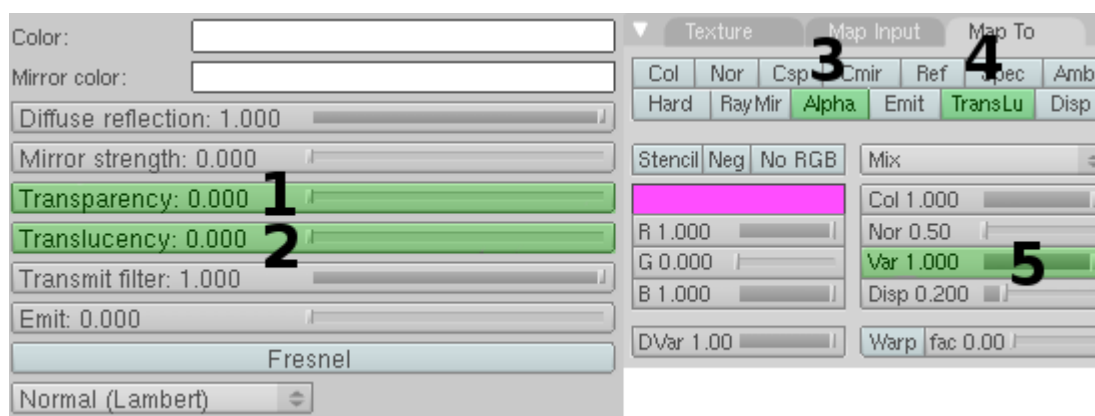
- [Glossy material.](#)
- [Shinydiffuse mirror material.](#)

Alpha Intensity.

This feature is used to map mesh transparency with a texture. Information from the RGB texture is translated into amount of mesh transparency. To make it possible, textures need to be transformed into a value-scale. This means that if the texture has got colors they will be transformed into tones of grey. It is a wise idea to use desaturated textures to map intensity, so there is more control over the final result. You can use a procedural texture as well.

Don't confuse 'alpha intensity' with 'texture alpha'. The texture alpha channel (RGBA) is used in [diffuse color mapping](#).

Alpha intensity mapping will just produce different values of transparency based on the texture values. Those different levels of transparency will produce colored transparent shadows, if transparency is applied onto a material with diffuse colors applied.



There are two kinds of transparency in YafaRay, specular *Transparency* [1] and diffuse transparency, also called *Translucency* [2]. To enable shadows calculation from alpha-intensity mapping, enable *Transparent Shadows* with enough *Shadows depth*, in the *Settings* section.

The texture will map in fact a range between the transparency value set in the material setting [1 or 2] and the *Var* value [5]. For instance, if *Transparency* is 0.20, it means that black color in the texture will make the object to be at least 0.20 transparent. If *Var*=0.80, it means that white color in the texture will make the object to be as much as 0.80 transparent. The same happens with *Translucency*.

To map *Transparency*, use *Alpha* slot [3]. To map *Translucency*, use *TransLu* slot [4].

At the moment, *Translucency* scattered light with transparent shadows only works with Global Illumination methods, which are Pathtracing, Bidirectional Pathtracing, and Photon mapping.

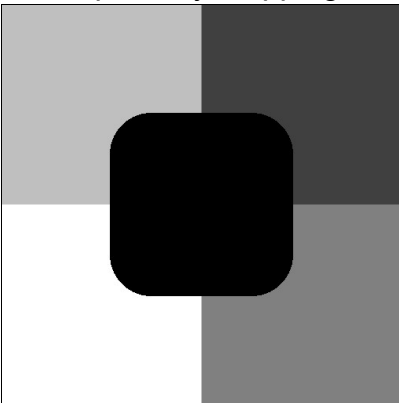
Below are examples of transparency and translucency mapping. Both cases are using *strength*=0, *Var*=1:



Transparency mapping.



Translucency mapping.



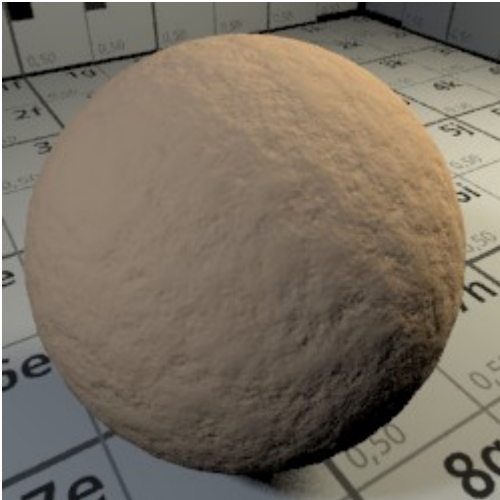
Texture used.

Related articles:

- [Shinydiffuse Transparency.](#)
- [Shinydiffuse Translucency.](#)

Bump Intensity.

All four materials (ShinyDiffuse, Glossy, Coated_Glossy and Glass) support bump mapping.



[Brian Lingard](#) gives this definition of bump mapping:

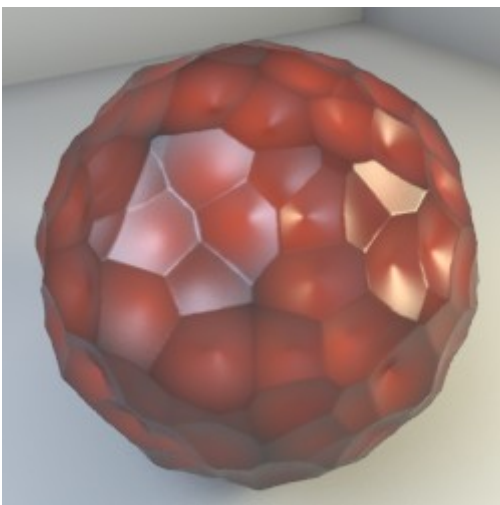
"Bump mapping simulates the bumps or wrinkles in a surface without the need for geometric modifications to the model. The surface normal of a given surface is perturbed according to a bump map. The perturbed normal is then used instead of the original normal when shading the surface using the Lambertian technique. This method gives the appearance of bumps and depressions in the surface."



Bump is an intensity-mapping type. Textures are transformed into a value-scale. It means that if the RGB texture has got colors, they will be transformed into tones of grey. It is a wise idea to use desaturated textures to map intensity, so there is more control over the final result. You can use a procedural texture as well.

To map bump, the *Nor* slot [1] must be enabled for the texture channel. *Nor* slider [2] controls amount of bump. YafaRay uses comparatively lower values than Blender for bump mapping.

Displacement mapping.



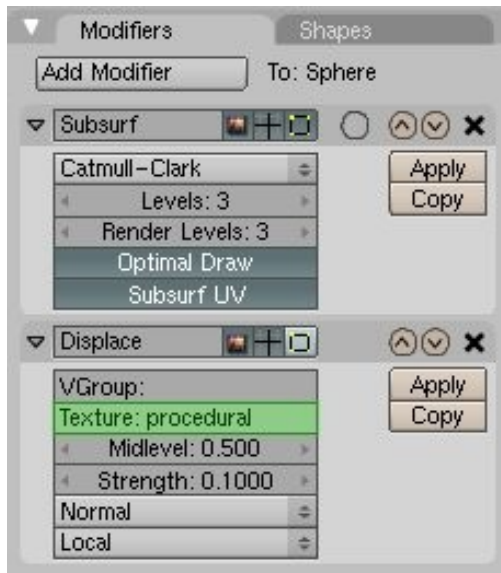
YafaRay supports displacement mapping applied not as a material modulator but as a Blender mesh modifier (F9, Modifiers panel). It can be applied to any mesh regardless of the material.

Displacement is a modern version of bump mapping, in which the object geometry is actually modified by a texture. Displacement needs a great amount of mesh tessellation to work; the more tessellation the better result. The tessellation needed is achieved by using subdivision surfaces modifiers (subsurf) on a mesh.

First, a texture must be created as usual, using Blender supported Texture buttons (F6). The texture will appear in the Texture channels stack (F5). Check your options to see

whether this displacement texture is needed for a modulation mode as well or not. For instance, you might want to enable this texture for diffuse color mapping as well while is still being used for displacement, like the example on the left.

This created texture is later used in the Displace mesh modifier (image below).



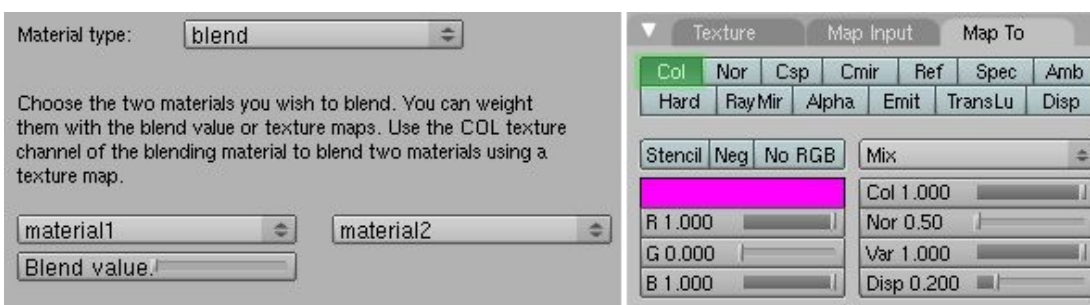
Displacement is an intensity-mapping type. RGB Textures are transformed into a value-scale. It means that if the texture has got colors, they will be transformed into tones of grey. It is a wise idea to use desaturated textures to map intensity so there is more control over the final result. You can use a procedural texture as well.

On the left there is an example of a Modifiers panel, using a subsurf modifier to add tessellation and a displacement modifier. The name of the texture used for displacement is input in the *Texture* field (green). This texture is previously defined in Blender texturing panels.

More information about subsurf and displace modifiers [here](#) and [here](#), respectively.

Blend mapping.

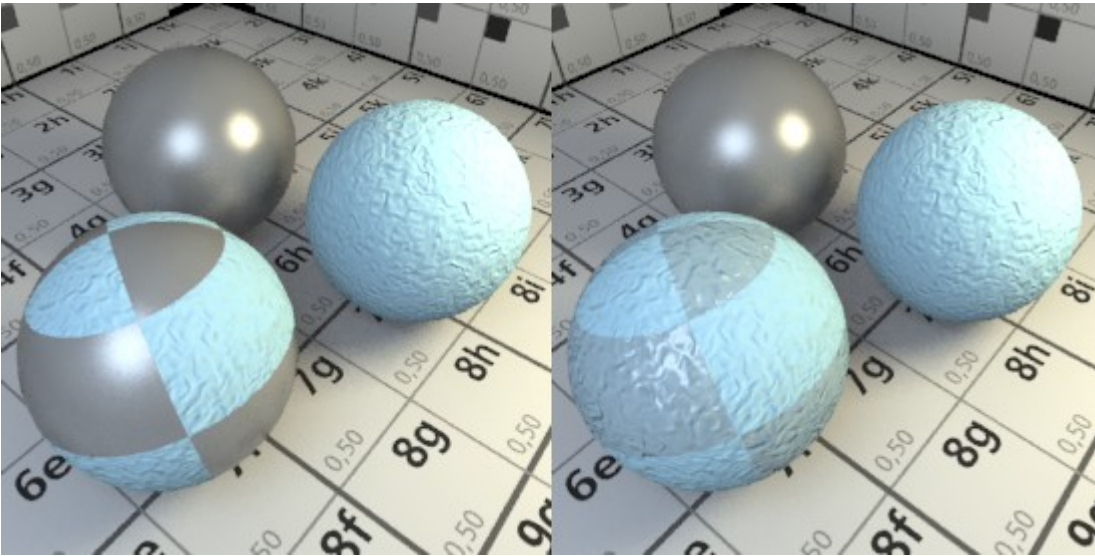
You can use a texture to control how two materials are blended. This texture is a mask in which values tell how big a percentage of each material is seen. If the RGB texture has got colors, they will be transformed into tones of grey. It is a wise idea to use desaturated textures to map blend, so there is more control over the final result. You can use a procedural texture as well. The texture should be mapped onto the object assigned with the blend material, using the *Col* slot.



If the texture is a black&white checker, it means black squares will show material 1 whereas white squares will show material 2 (example below). The texture can use tones of grey as well, which means that properties from both materials are blended in the same area.

Blend value works as a global opacity factor for the texture. If you want the texture to have a complete control of the blending, use a Blend value=0. If Blend value is 1, the texture will be completely transparent and only the material 2 can be seen.

Below an example of blend mapping, using a black&white checker. On the left a Blend value=0 is used whereas on the right Blend value is 0.50:



Related article: [Blend Material](#)

Lighting Methods

Table of Content:

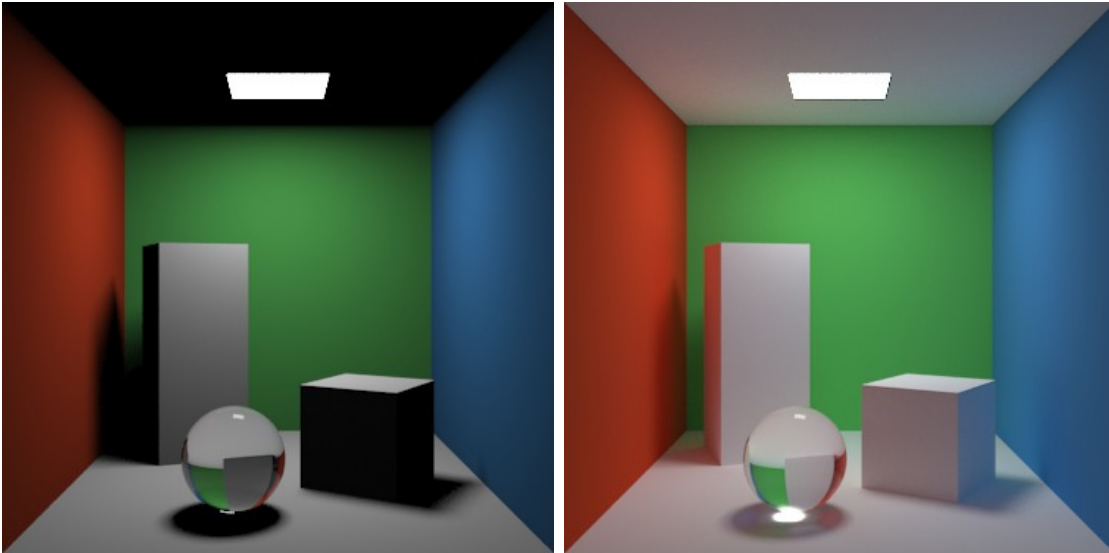
- [Illumination Types](#)
- [Lighting Methods Overview](#)
- [Use cases](#)
 - [Open scenes](#)
 - [Enclosed scenes](#)
 - [Animations](#)
- [Parameters:](#)
 - [Ambient Occlusion](#)
 - [Caustic Photon Map](#)
 - [Path tracing](#)
 - [Photon Mapping](#)
 - [Final Gather](#)
 - [Bidirectional](#)

Illumination Types.

In real life, objects are lit by several types of lighting. For instance, light casted from lamps and light bouncing from other nearby objects. Besides, some materials, such as glass, can modify the behaviour of light. Other lighting effects are color bleeding between adjacent surfaces and light scattering in participating media. The interaction and sum of all these effects produce a global result that is called global illumination.

Raytracers try to reproduce global illumination (GI) by means of techniques that follow light natural behaviour, such as casting samples of rays that bounce in the scene observing realistic rules. Those samples are computed so a general result can be extrapolated from a limited amount of rays. Theoretically, there are more than one way to achieve this, each way with its own share of advantages and shortcomings. There isn't such a thing as a perfect GI model, and new algorithms are created for solving the inconvenients of already existing techniques.

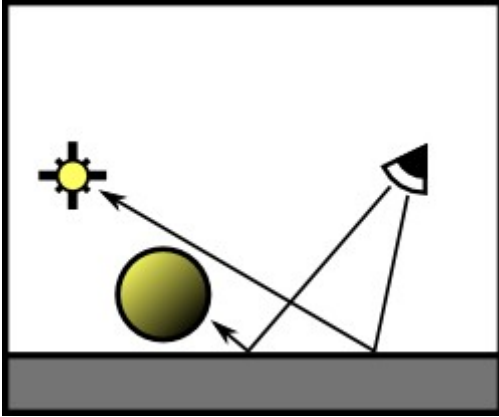
YafaRay has got a traditional raytracing method which only renders light cast from light sources, called **Direct Lighting**. Additionally, YafaRay has got three different Global Illumination models, which are **Path tracing**, **Photon Mapping** and **Bidirectional Path tracing**. Below you have a comparison between Direct Lighting (left) and a Global Illumination model, in this case Photon Mapping (right). Notice the differences between them:



A certain amount of borrowing is possible between these lighting methods, since they are more or less based on the same 'ray' concept. So it is possible to implement the same averaging technique in different GI methods, or mix features from different models. This is the case of caustic photons option in Direct Lighting, for instance.

Lighting Methods Overview.

Direct Lighting



How it works:

Direct Lighting only performs recursive raytracing. Primary rays are shot from the camera and intersect with the scene.

Then secondary rays are generated but just towards light sources, to calculate shadows. They're also called **shadow rays**.

Primary rays can be transmitted in reflective and refractive surfaces.

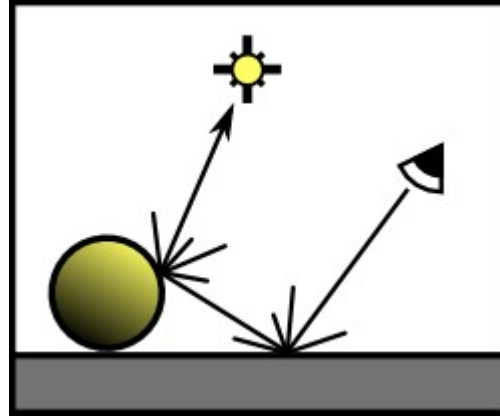
Advantages:

- Very fast in scenes where indirect lighting is not needed.
- Can render independent caustic photonmaps and ambient occlusion.
- HDR backgrounds (IBL, Sunsky) can work as light sources, simulating indirect lighting.

Disadvantages:

- No indirect lighting.
- Can't render caustics (you'll need to enable caustics photonmap).

Path Tracing



How it works:

Rays are shot from the camera. Each bounce casts a lot of secondary rays. Some of them eventually reach a light source. Then the light contribution along the path is calculated.

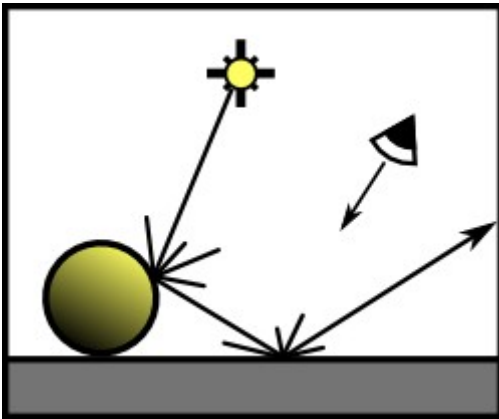
Advantages:

- It performs Global Illumination.
- Fast in outdoorsm, because paths find the background easily.
- Unbiased, delivers correct results.
- Soft indirect lighting if enough sampling.

Disadvantages:

- Variance shows up as noise.
- Lot of rays are needed for caustics.
- Inefficient in indoor scenes, when light sources are too hidden or too small.
- Doesn't like omni lights (spot, point) and mirror surfaces. Better use area light types and glossy.

Photon mapping



How it works:

Rays (photons) are cast from light sources and bounce around, regardless of the camera. A photon map is created, based on photon hits.

Then, a standard raytracing pass is performed to visualise the photon map.

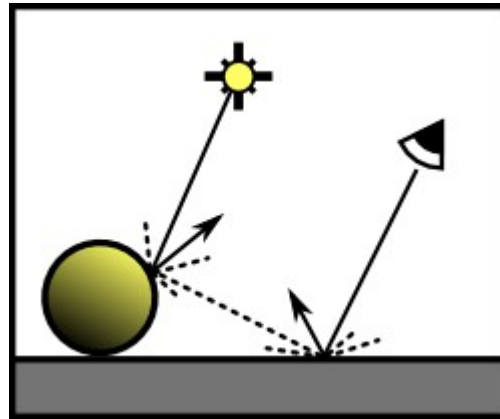
Advantages:

- It performs Global Illumination.
- Fast, efficient GI estimation in indoors.
- Best quality/speed ratio.
- Fast caustics.

Disadvantages:

- Not well suited for outdoors
- Sometimes requires photon map tweaking.
- Artifacts.

Bidirectional path tracing



How it works:

Ray paths are constructed from the camera and from light sources. Bounces are connected to each other with visibility rays.

Advantages:

- It performs Global Illumination.
- Combine advantages from path tracing and photon mapping.
- More efficient than path tracing for indoors and for caustic effects.
- Unbiased, delivers correct results.
- Good for scenes with lot of indirect lighting.

Disadvantages:

- Variance shows up as noise.
- Not well suited for outdoors.
- Inefficient in indoor scenes, when light sources are too hidden.

Related articles:

- [Ray Depth.](#)
- [Path tracing.](#)
- [Photon Mapping.](#)
- [Bidirectional.](#)
- [First anti-aliasing pass.](#)

Use Cases.

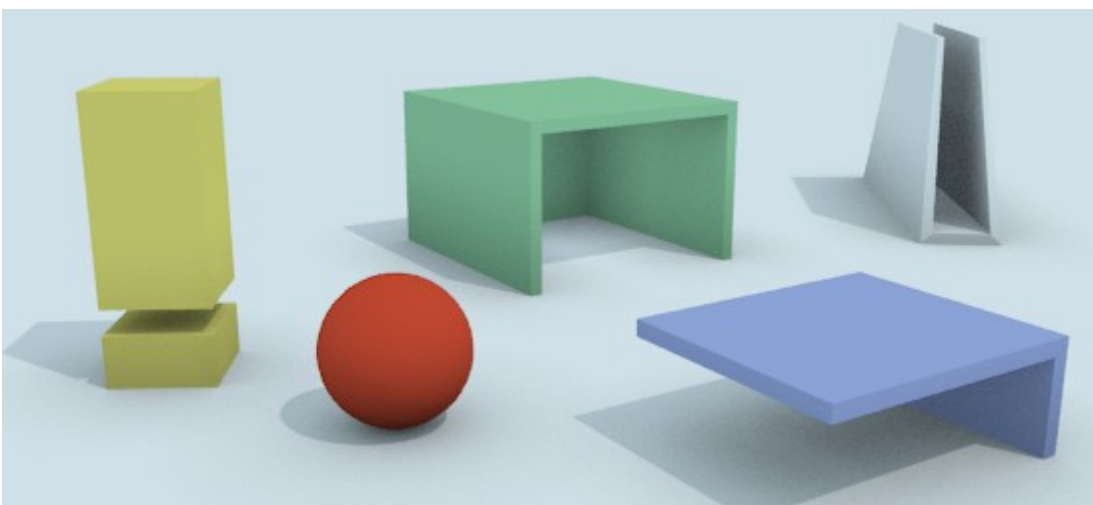
GI algorithms are born with shortcomings and compromises in order to solve very specific raytracing problems. There isn't such a thing as a universal lighting algorithm for all cases. In fact, the only method that can be used for all cases is Direct Lighting. In the next paragraphs we are going to explain use cases in YafaRay, and what are the best lighting methods for them.

Open scenes:

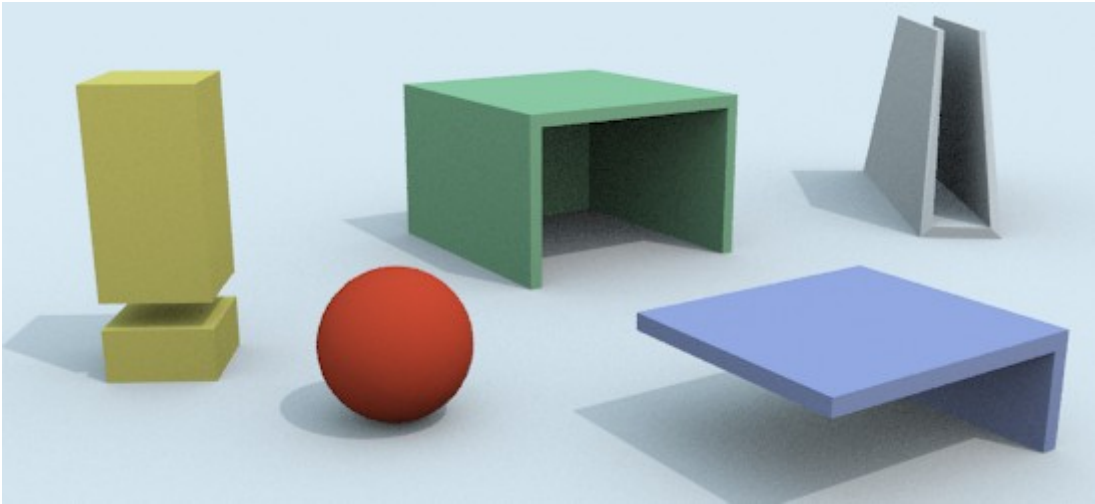
Open scenes means that the scene is not enclosed by a mesh, and the background can work as a main light source. Open scenes can be used to simulate studio lighting and indoor scenes as well, by using a suitable scene composition and/or a HDR indoor image as a background. The recommended methods for open scenes are:

- **Direct lighting.** It delivers fast results and you can use area light types (area, sphere, mesh) to achieve soft shadows.
- **Direct lighting + AO + caustics photons.** Fast. If you have refractive and/or reflective surfaces in the scene you can enable caustic photons, it adds realism. Use this 'combo' for fluid animations as well.
- **Direct lighting + HDRI texture.** HDR images can work as light sources in Direct Lighting. Fast for outdoors, when indirect lighting can be simulated with background lighting. HDRI backgrounds can shoot caustic photons as well.
- **Direct lighting + Sunsky.** Components of the Sunsky model (Sun and Skylight) work as light sources. Fast for outdoors, when indirect lighting can be simulated with background lighting. Sunsky can shoot caustic photons as well.
- **Path tracing + background** (HDRI, Sunsky, Gradient, Single Color). Fast because many pathtracing rays can find a light source (background) after the first bounce. Use it if you need color bleeding or if you want a precise simulation of the indirect lighting.

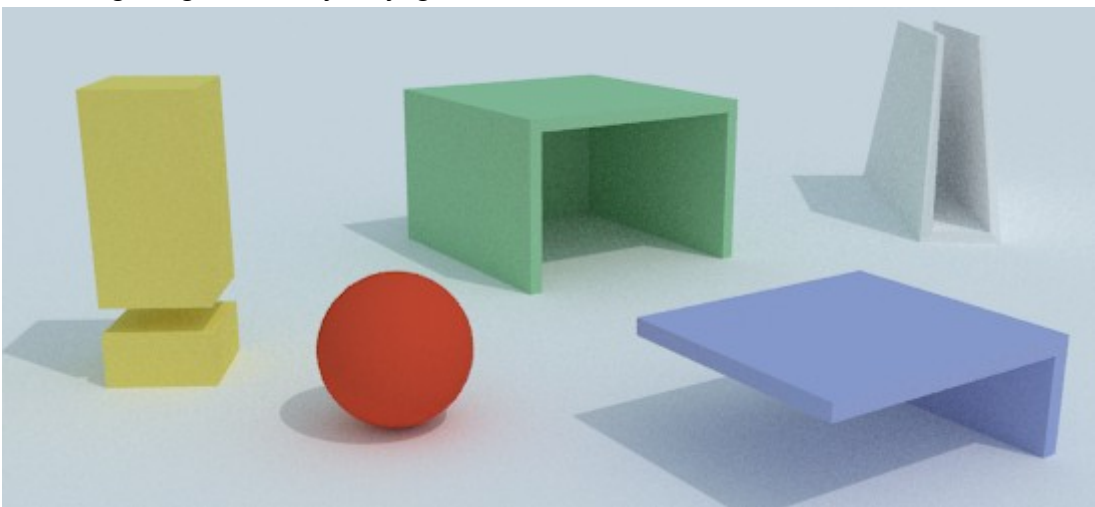
Comparison between Direct Lighting and Pathtracing used in an outdoors scene. Notice indirect light calculations and soft color bleeding effects in Path tracing.



Direct lighting + Ambient Occlusion, render time 42 s.



Direct lighting + Sunsky Skylight, render time 518 s.



Path tracing + Sunsky Skylight, render time 1205 s.

Related articles:

- [Ambient Occlusion.](#)
- [Background Settings.](#)
- [Path tracing.](#)

Enclosed scenes:

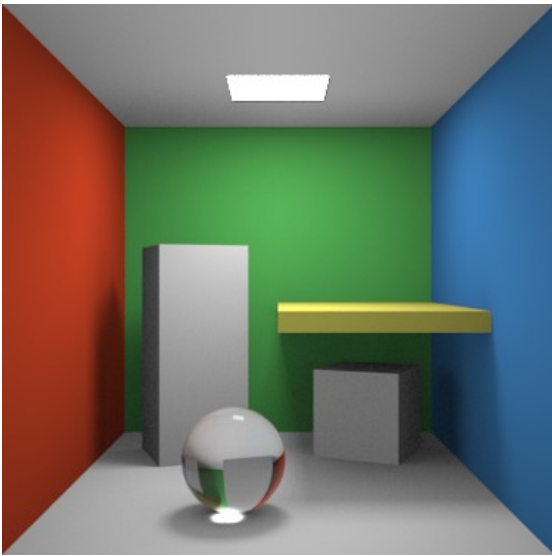
Enclosed scenes means that the scene is inside a mesh (thickness for walls recommended), which is good for some ray bouncing. This enclosing mesh can have windows to simulate a houseroom. The recommended methods for enclosed scenes are:

- **Direct lighting.** Use it for traditional lighting setups and studio lighting, if you don't need the indirect lighting component. You can use area light types (area, sphere, mesh) to achieve soft shadows. To simulate indirect lighting, many times photon mapping will be faster than a complex rig of fill lights.
- **Direct lighting + AO + caustics photons.** Fast. If you have refractive and/or reflective surfaces in the scene, enable caustic photons, it adds realism. Use this 'combo' for fluid animations as well.
- **Photon mapping + FG.** A good fast GI algorithm for all indoor cases, it isn't afraid

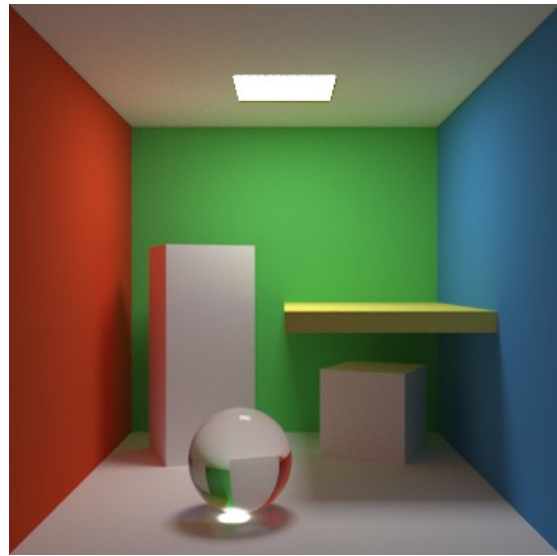
of complex lighting cases.

- **Bidirectional path tracing.** It works well if there is an optimal situation of light sources. For instance, if light sources can be seen from the camera or from the first bounce of camera rays. Use it in scenes with lot of indirect lighting and/or if you are looking for a correct simulation of the Global Illumination.
- **Path tracing.** The least efficient method for indoors. Use it if there is an optimal situation of light sources (see bidirectional) and an even distribution of lighting, for instance scenes with big windows or big light sources. You'd better use area lights (area, sphere, meshlight) rather than omni lights (spot, point) in path tracing.

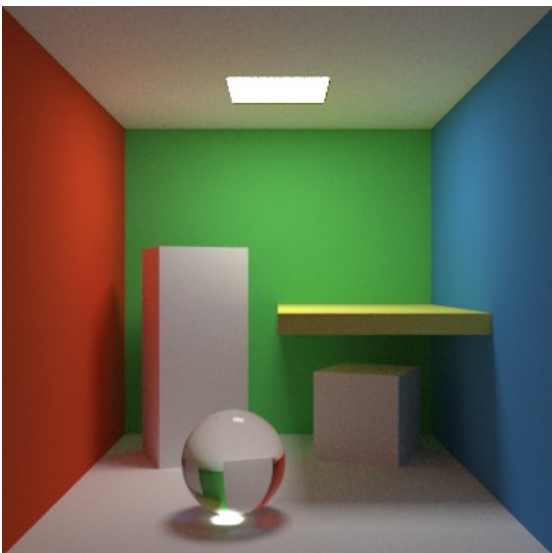
Comparison between different lighting methods, rendered on a Pentium IV. Notice lack of color bleeding in Direct Lighting. Photon mapping is the fastest GI method, and Bidirectional the slowest one. Notice how the two unbiased methods (Path tracing and Bidirectional) struggle in indirect lighting areas (noise). Bidirectional produces the brightest indirect lighting result and more color bleeding than the other GI methods.



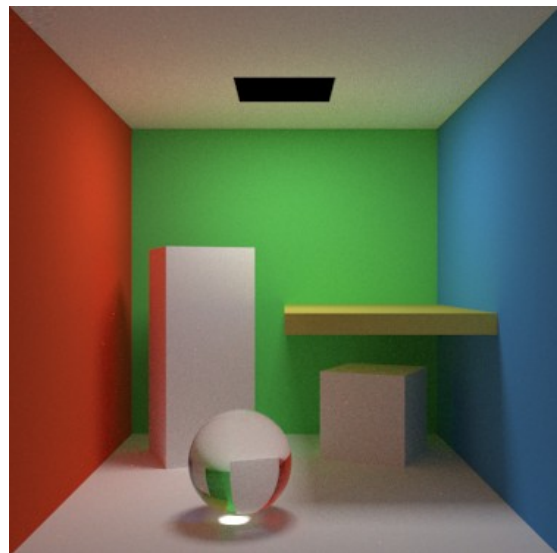
Direct lighting + AO + Cphotons. **0:1:35**



Photon mapping. **0:9:10**



Path tracing + Cphotons. **0:15:20**



Bidirectional. **1:32:18**

Related articles:

- [Caustic Photon Map.](#)
- [Ambient Occlusion.](#)
- [Path tracing.](#)
- [Photon Mapping.](#)
- [Bidirectional.](#)

Animations:

- **Direct lighting + caustics photons.** Fast for indoors, even faster with omni lights (point, spot). Caustic photon maps are compatible with frame rendering (no caustics flickering). HDR backgrounds optional (IBL, Sunsky), with caustic photons.
- **Photon mapping + FG.** If indirect lighting is needed, PM+FG is the fastest Global Illumination algorithm for animations. It's compatible with both frame-based animation and portion-based distributed rendering (no shadows flickering from frame to frame, no seams when render portions are joined together).

Related articles:

- [Glossy reflection sampling.](#)
- [Caustic Photon Map.](#)
- [Photon Mapping.](#)
- [Final Gather.](#)

Parameters

Ambient Occlusion

Ambient Occlusion is a shading method that takes into account attenuation of light due to object occlusion. Ambient occlusion is most often calculated by casting rays in every direction from a point on a surface. Rays which reach the background or “sky” increase the brightness of the surface, whereas a ray which hits any other object contributes no illumination. As a result, points surrounded by a large amount of geometry are rendered dark, whereas points with little geometry on the visible hemisphere appear light.



Ambient occlusion is often used as a fast approximation of the indirect lighting produced by Global Illumination models. It is also used as an independent pass for render post-processing, usually with *Clay render* enabled.

AO settings are:

- *AO Samples*: The number of rays used to detect if an object is occluded. Higher numbers of samples give smoother and more accurate results, at the expense of slower render times
- *AO Distance*: The length of the occlusion rays. The longer this distance, the greater impact that far away geometry will have on the occlusion effect. A high Dist value also means that the renderer has to search a greater area for geometry that occludes, so render time can be optimized by making this distance as short as possible, for the visual effect that you want.
- *AO Color*: Color for ambient occlusion rays. Use this setting to control AO power.

Related articles: [Use cases](#)

Caustic photon map

Caustics is a concentration of light, produced by refractive medium (glass, water) and by curved specular surfaces (mirror, glossy and fresnel). It is possible to produce independent caustic photon maps in YafaRay. This option is available in lighting methods that either can't render caustics (Direct Lighting) or aren't efficient at this kind of task (Path tracing). Caustics add realism and they are relatively cheap to compute. The caustics photon map is visualized directly and this is the reason why the number of photons in the caustics photon map must be high, since a high resolution map is needed.

Mix and *Radius* are two limit parameters to blur the caustic map. Photon hits will be averaged within a circular area. The center of each circular area is defined by camera rays. If there isn't enough photon density within the circular area, low frequency noise will appear. Those circular areas will use whatever limit is reached first, either *Mix* or *Radius*. Many times incremental changes in a limit won't have any blur effect since the other threshold has been already reached.

Below are two of examples of caustics:



Example of refractive caustics, using as photons source a spot light. Scene by **MarcoA**



Example of reflective caustics, using a caustic photon map in pathtracing. The HDRI background is the caustic photons source. Scene by **Sevontheweb**.

Caustic photon maps work better and more efficiently with concentrated light beams directed towards the 'caustic' surface, using a spot light. Caustic settings are:

- *Photons*: Number of photons to shoot. Increases photon map density and render times.
- *Caustic Depth*: amount of reflective or refractive events for caustic photons.
- *Caustic Mix*: amount of photons to mix (blur). The more photons to search for, the more render time.
- *Caustic Radius*: area of photons to mix (blur). The more photons to search for, the more render time.

Related articles:

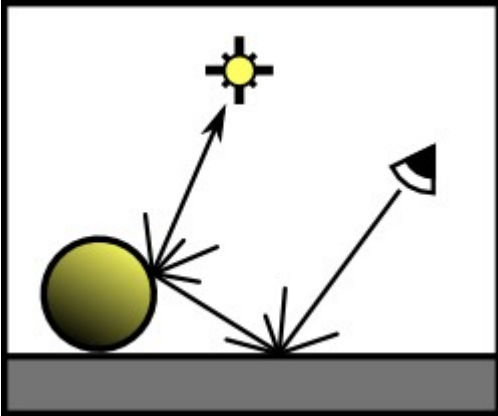
- [Use cases](#)
- [Shadow Depth and Transparent shadows.](#)
- [Photon Mapping.](#)

Materials that can produce caustics:

- [Glass](#)
- [Glossy](#)
- [Coated Glossy](#)

- [Shinydiffuse Mirror & Fresnel.](#)

Path tracing.



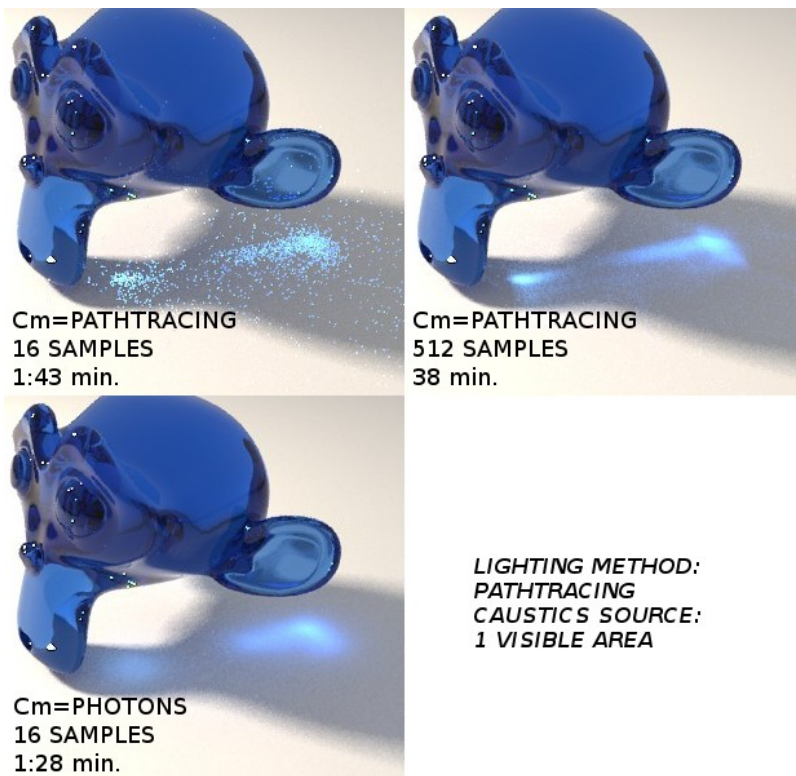
Path tracing is a GI unbiased method in which each ray is recursively traced from the camera along a path until it reaches a light source. When a light source is found, the light contribution along the path is calculated, taking into account surface properties. Many samples are needed to be taken and interpolated for each camera pixel to get a smooth result. A light source can be either a lamp, the scene background or both. Scenes with relatively small light sources and with a high contrast between light sources and their surrounding areas will need more samples to reduce noise. The smaller and less accessible the light sources are, the more noise. Pathtracing is a GI solution more suited for outdoor scenes and for indoor scenes with big light sources (area lights or big windows) and a regular distribution of light.

Path tracing caustic component

Pathtracing caustic paths tend to be very noisy and a very big amount of samples is needed to get a smooth result. In YafaRay we have four alternative methods to render the caustic component when pathtracing is used:

- *Path+Photon*: a mix of a caustic photon map and path tracing caustic rays are used.
- *Photon*: a fast photon map is used to render caustics. Path tracing caustics rays are not rendered.
- *Path*: Path tracing caustics rays are rendered.
- *None*: the caustic component is not rendered.

This is an example about how methods for the caustics component work in pathtracing. In the first image (upper left), *Path* is used to get caustics, which are very noisy when a low number of samples is used (16). In the second render, 512 pathtracing samples are used to improve path traced caustics, but it takes much more render time (38 minutes). In the third example, *Photons* are used to produce the caustics component and the render time is the lowest of them all (Cm stands for Caustic method):



Area light types (sphere and area) with the Make Light visible option enabled produce caustics in Path caustic mode. More information about the caustic photon map settings can be found in the previous section (they are the same)

Related articles:

- [Caustic Photon Map.](#)
- [Glass material.](#)

Path tracing settings

The other components of the global illumination model are rendered as usual. Other pathtracing settings are:

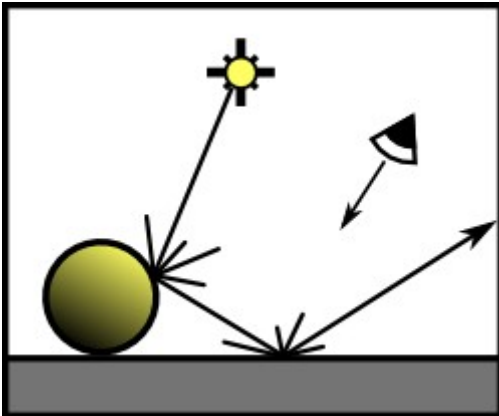
- *Depth*: defines the number of rays bounces in order to find the light sources. Higher raydepth produces brighter renders. In scenes with big light sources, like open scenes using background lighting, a high raydepth could be unnecessary. This setting controls depth of pathtracing caustic rays as well.
- *Samples*: This setting defines the number of samples to take per camera pixel, the more samples the better render quality and the less noise, but the longer render time as well. The relation between noise reduction and sampling isn't linear: to halve the noise, it is necessary to use four times as many samples. The total amount of Path tracing sampling depends as well on anti-aliasing settings, as explained in [this section](#).
- *Use background*: Makes use of the background as a light source. Sunsky models and IBL lighting work always as light sources, regardless of this setting.
- *No recursion*: only path tracing is performed without recursive raytracing, which means that shadow rays are not traced.

It is a good practice to increase & decrease your 'samples' settings in base 2 steps (2-4-8-16-32-64-128... etc)

Related articles:

- [Lighting Methods Overview.](#)
- [Use cases](#)
- [First anti-aliasing pass.](#)
- [How backgrounds work.](#)

Photon mapping



Photon mapping was developed as an efficient alternative to path tracing, since certain effects are more efficiently simulated with sampling from lights (caustics, indirect lighting) while some others effects are more efficiently sampled from the camera (mirror reflection, direct lighting).

Photon mapping is a two pass technique:

- The first pass is photon tracing, which consist in building the photon map by tracing photons from lights. It calculates indirect lighting and caustics effects.
- The second pass consist in raytracing the scene (camera rays) using the information in the photon map. This second pass calculates specular reflections (mirror & glossy) as well.

Photon mapping is biased, the average values might not be correct, but it is consistent: with more photons and less radius, it converges to a correct solution. Photon map produces low frequency noise (big patches) in contrast with path tracing, which produces high frequency noise (pixel-level).

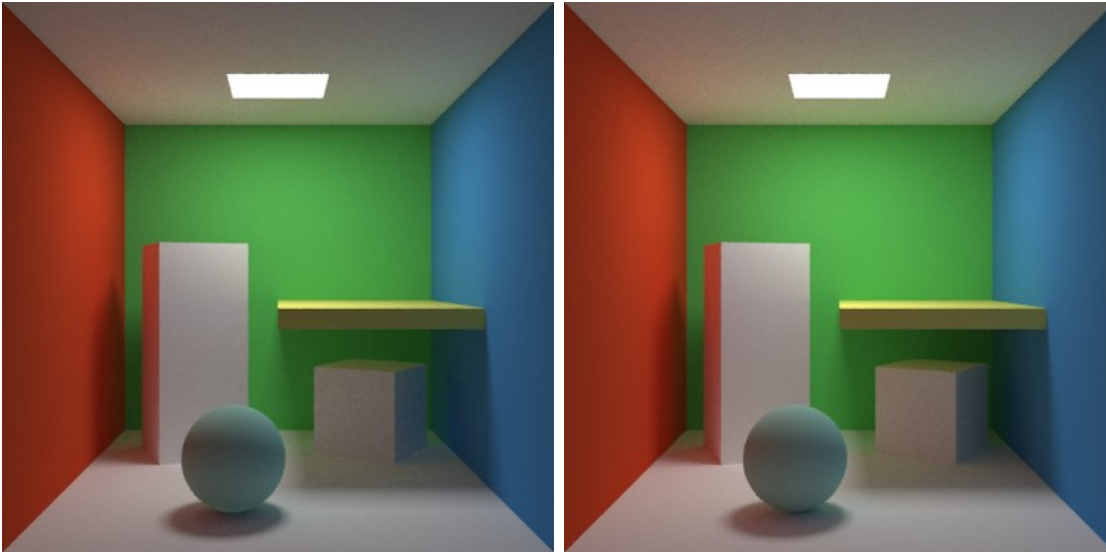
Photons

Photons propagate flux. They are emitted by light sources and stored in the photon map when they hit a surface. In fact, each emitted photon can be stored several times. The photon map represents the incoming illumination, also called Irradiance (incoming radiance). Two photon maps are produced in fact, which are:

- A low density photon map for diffuse surfaces called Global photon map.
- An independent high density map for caustic effects.

In general, the more photons are used, the more accurate is the lighting estimate, but increasing photons increases time to build the photon map. If the number of photons is too low, the irradiance estimation becomes blurry at the edges of sharp features in the illumination, for instance fast transition from light to shadows in corners.

However, in simple scenes with no caustic effects, it is possible to produce relatively good results with very low density maps of 1000-2000 photons. In this case the photon map is rendered very quickly. Below a comparison between two photon mapping cases with different photon counts, notice the bad estimation below the horizontal prism and below the sphere, in the render on the left:



2000 Photons, Depth=2, 213 seconds. 1.000.000 photons, Depth=10, 290 seconds,

Depth

Depth controls amount of consecutive bounces for both caustic and diffuse photons. However, it has got a different meaning depending on photon type:

- *Depth* for Caustic photons: Amount of consecutive refractive and/or reflective events for caustic photons, until they reach a diffuse surface.
- *Depth* for Diffuse photons: Amount of bounces in diffuse surfaces. More depth will produce more color bleeding and a denser photon map. Beyond certain point the amount of depth will have a limited effect in map density or color bleeding, since most photons will be already absorbed by diffuse surfaces.

Comparison between different values of *Depth*, with the same number of photons shot. Photon hits means the number of photons recorded in the photon map, which is a value shown in the back console. Notice the relatively small difference in hits between depth=10 and depth=50. Almost equal render times in all cases. Scene by **Kronos**:



Photon Depth=3, photon hits= 410.000



Photon depth=10, photon hits= 610.000



Photon depth=50, photon hits=650.000

Radius

The second pass is a traditional ray tracing pass performed by shooting rays from the

camera. Based on a single photon, we can not say how much light a region receives. This information is thus provided by the photon density. When a ray hits a point **P** on a surface, the illumination information of the neighboring photons is collected and interpolated at **P**. We don't need a photon for every polygon, but instead a few photons to estimate the incoming flux in the region around **P**. The photon density is higher in areas with strong incoming illumination.

A well chosen radius allows for a good pruning of the search. Two radii are used to perform the search of neighboring photons, whichever is reached first:

- *Diff. Radius*. The sphere of fixed radius improves the estimation slightly, but fails in scenes with high variation of density of photons. It gives bad estimates if there are too few photons and blurry estimates when there is a high density of photons.
- *Search*. User can specify a desired number of photons. It guarantees that there are *K* photons in the measurement. The more photons are used, the smoother the estimation will be. If too many are used, the estimation will tend to be blurry, while too few gives a splotchy appearance (low frequency noise). Interpolating 50 to 100 photons is often a good choice.
- *Caustic Mix*: Number of photons to mix, produces blur in the caustic photon map.
- *Use background*: The way it works depends on the background used:
 - With *Single Color*, *Gradient* and non-IBL *Texture*, the background works as a light source if this option is enabled. Use FG & AA sampling to remove noise.
 - *IBL Texture* and *Sunsky* work always as a light source, regardless of this option. However, when it is enabled, photons hitting the background will not be discarded but get energy from it. As a result, scenes will have more lighting power from the background than when this option is disabled. Use BG & FG & AA sampling to remove noise.

In general, Radius should be inversely proportional to the number of photons shot: the more photons the less radius. However, a too low radius without an adequate photon density, introduces noise (small patches). Radius settings are one of the main factors in render times. If the number of photons to look up and average increases, so does the render times.

Note: When using photon mapping in enclosed scenes such as house rooms, it is important that objects follow realistic techniques for modelling, such as closed meshes with real thickness. Follow this advice in walls, floor, ceiling and furniture. In this way we avoid estimation problems when searching for neighboring photons.

Related articles:

- [Lighting Methods Overview.](#)
- [Use cases](#)
- [Glossy As diffuse.](#)
- [How backgrounds work.](#)

Materials that can produce caustics:

- [Glass](#)
- [Glossy](#)

- [Coated Glossy](#)
- [Shinydiffuse Mirror & Fresnel.](#)

Final Gather

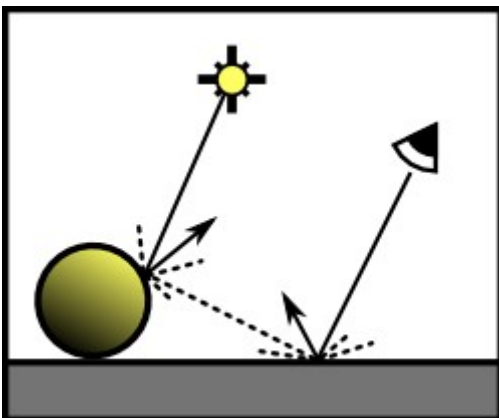
Final gather is a caching technique to improve and 'complete' photon mapping by gathering, after photon tracing, an approximation of the local irradiance by using several illumination bounces. This information is used at render time for further interpolation, with the obvious advantage of requiring a less accurate and therefore faster, but yet physically correct, photon map. Noise reduction in FG renders depends on FG samples and anti aliasing settings.

- *FG bounces*: in a precomputed phase, determines the number of bounces for Final Gather rays. The first bounce is the most important one; subsequent bounces have got a more modest impact on the result.
- *FG samples*: Final Gathering samples for interpolation, the more the better, but the longer it will take to render. The total amount of Final Gather sampling depends as well on anti-aliasing settings, as explained in [this section](#).
- *Show map*: Useful to tune the photon map. When using it, you must seek for results with a soft uniform appearance of patches, without noticeable groups of darker patches regularly distributed across surfaces, which is a hint of low-frequency noise. When you have problems of density there are two options, increasing number of photons shot and bounces, or increasing radius settings. A good well-balanced photon map is especially important for animations.

Related articles:

- [Photon Mapping.](#)
- [First anti-aliasing pass.](#)

Bidirectional Pathtracing



Bidirectional constructs rays from the camera and from light sources, and connect each others' bounces with visibility rays to ensure they are mutually visible. It is more efficient than pathtracing for caustics and indoor lighting, but variance show up as noise too.

Sampling of the bidirectional rays is performed by using antialiasing settings, which means that you'll need a big amount of antialiasing sampling. For instance, 16 AA passes x 16 AA samples would mean that 256 bidirectional samples are used. Another strategy is using an extreme amount of AA passes x AA samples, and stopping the render process when it is clean enough. The more AA sampling, the more convergence to the correct solution.

AA threshold must be 0 to sample all pixels in every pass. The proportion between AA passes and AA samples is irrelevant in this case for noise removal, since the whole image is resampled in every pass. Therefore, only the multiplication result matters, the higher the amount of samples, the less variation and the more noise reduction. However, having all samples in one pass will likely be faster than having many passes with one sample each. Rendering a small problematic portion of the image with border rendering (**Shift+B**) will give you an idea about how the result converges and the amount of sampling needed for the whole render.

Related articles:

- [Use cases](#)
- [Anti-aliasing](#)

Render Settings

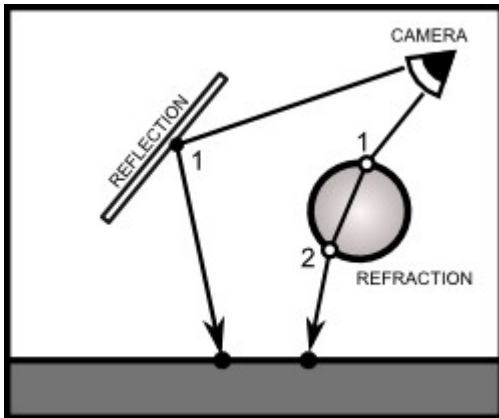
Table of Content:

- [General settings.](#)
 - [Ray Depth.](#)
 - [Shadow Depth and Transparent shadows.](#)
 - [Output Gamma.](#)
 - [Input Gamma.](#)
 - [Clamp RGB.](#)
 - [Clay Render.](#)
 - [Threads.](#)
 - [Result to Blender.](#)
 - [Autosave.](#)
 - [Alpha on autosave/animations.](#)
 - [Draw render parameters.](#)
 - [Output to XML.](#)
- [Anti-aliasing.](#)
 - [Introduction.](#)
 - [First anti-aliasing pass.](#)
 - [Adaptive sampling.](#)
 - [Filter type.](#)
 - [Filter Size.](#)
 - [Anti-aliasing strategy.](#)

General settings.



Ray Depth.



Raydepth means the amount of times that camera primary rays can be reflected in specular surfaces. Specular surfaces in YafaRay are:

- *Shinydiffuse Mirror*
- *Glossy material.*

It also defines the amount of times that camera rays can get through a transparent surface. Transparent surfaces are:

- *Shinydiffuse with Transparency.*
- *Glass material.*

This setting makes possible to render inter-reflections or consecutive transparent surfaces down to a certain depth. A typical effect of insufficient ray depth is a transparent surface or a reflection being rendered black.

Related articles:

- [Lighting Methods Overview.](#)

Shadow Depth and Transparent shadows.

Shadow Depth and *Transparent shadows* settings work together. *Transparent shadows* are shadows produced by 'fake' transparent surfaces without an index of refraction (IOR).

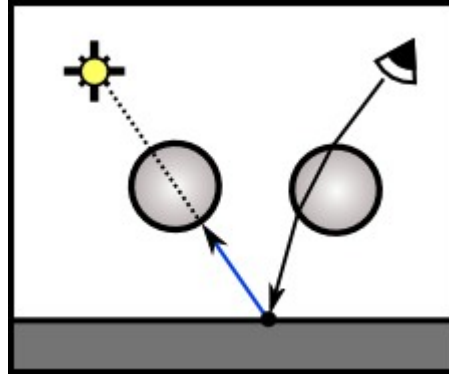
Transparent shadows enabled makes it possible to produce lighting filtered by a transparent surface, as an alternative to caustics effects produced by GI methods. In YafaRay, there are two kinds of 'fake' transparent materials without IOR, which are:

- *Shinydiffuse* with a *Transparency* value ≥ 0 .
- *Glass material* with *Fake glass* option enabled.

Shadow Depth controls the amount of 'fake' transparent surfaces that shadow rays can get through to find light sources.

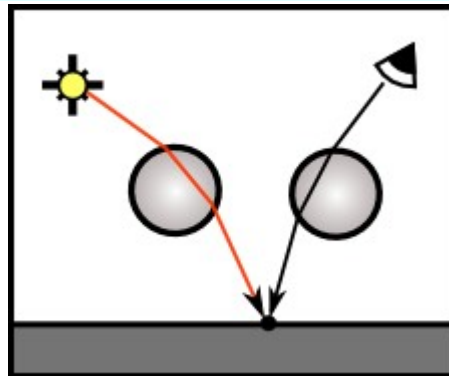
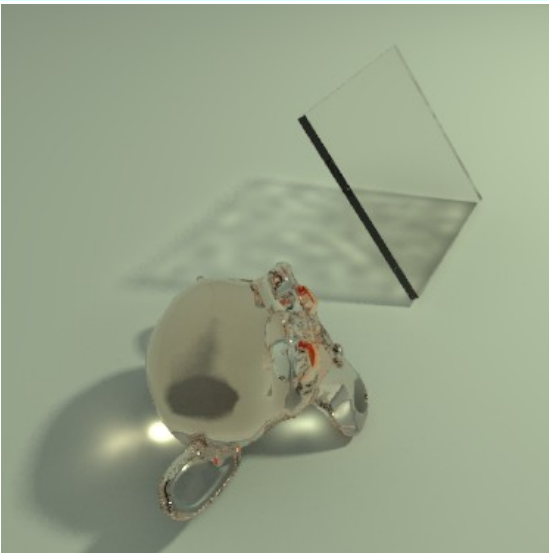
Below there is a comparison between different ways of handling the transparent shadows issue, and where *Transparent shadows* & *Shadows depth* options might be useful. *Direct lighting* is used in all cases, although the results can be extrapolated to the other lighting methods, which have got caustic and raytracing components:

Case 1



No method has been enabled for lighting rays to get through the transparent mesh, either for light sources rays (caustics) or for raytracing shadow rays. The mesh refraction is correctly rendered with raytracing primary rays but shadow rays (blue) are occluded by meshes as they look for light sources.

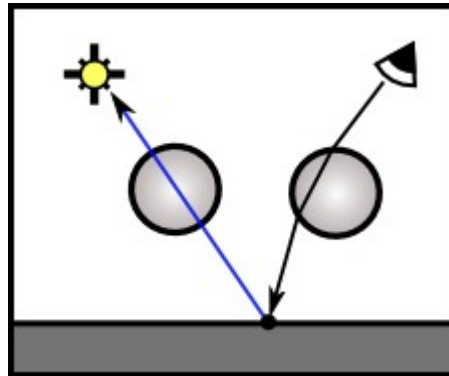
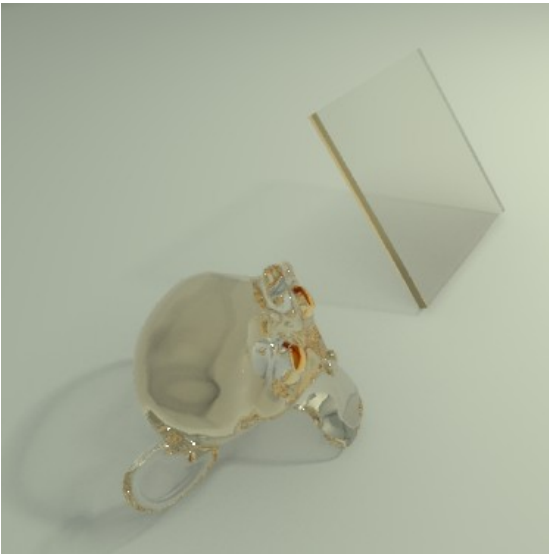
Case 2



Caustic photons, emitted from an arealight, produce lighting after getting through the glass objects with IOR=1.55. Suzanne's round shape can concentrate the flow of photons so a caustic map is produced with a bit of uniform density.

However, the prismatic glass produces a caustic map with density issues, since there isn't any concentration of the photons flow. The other area types (sphere, mesh) and HDR backgrounds (IBL, Sunsky) are prone to this kind of issue too.

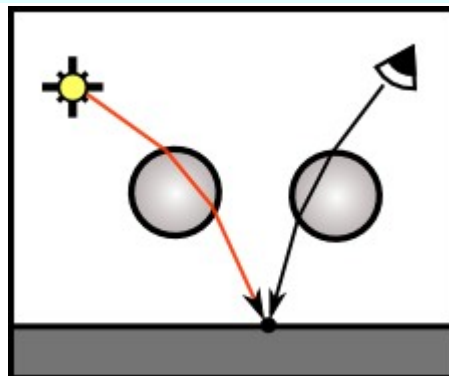
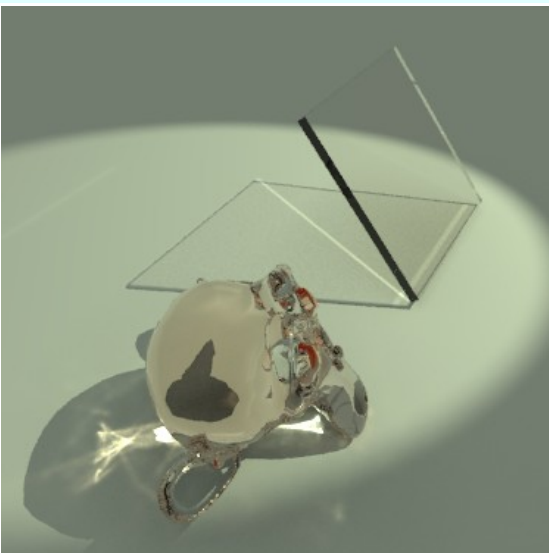
Case 3



An alternative could be enabling the *Fake glass* option, as well as *Transparent Shadows* with enough *Shadows depth*. Caustic photons aren't needed here since the transparent shadows are rendered with raytracing shadow rays.

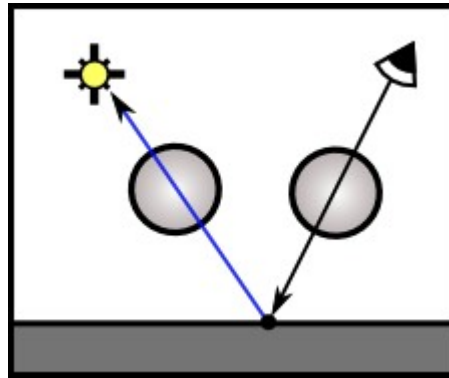
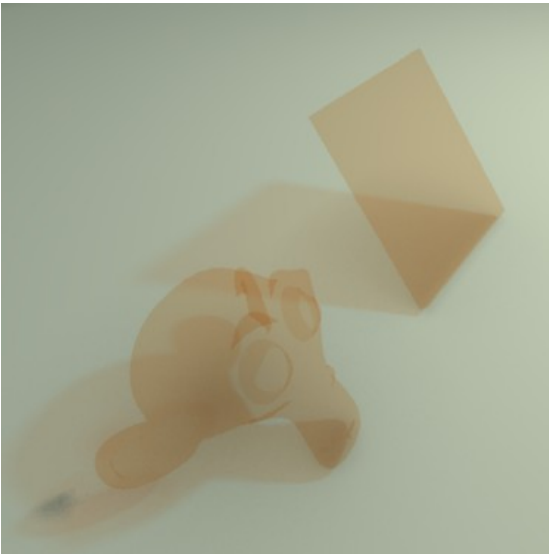
The transparent shadow produced by the prismatic glass is conveniently uniform, whereas the transparent shadow produced with Suzzane's head becomes completely unreal.

Case 4



Another alternative can be using a spot light instead of area types, shooting caustic photons. The narrower the spot beam the better. It is good for caustics because the spot light beam helps to focus and to concentrate the flow of caustic photons. Notice the superior quality of the caustic effects in both meshes (*Fake glass* and *Transparent shadows* are disabled here, so it's a 'real' glass).

Case 5



In this case, objects use a *Shinydiffuse* material with *Transparency*. Caustic photons aren't needed here since transparent shadows are rendered with raytracing shadow rays.

However, *Transparent shadows* must be enabled, and a sufficient *Shadow depth* should be set for shadow rays to get through all transparent surfaces when looking for light sources.

Related articles:

- [Lighting Methods Overview.](#)
- [Caustic Photon Map.](#)
- [Glass Material.](#)
- [ShinyDiffuse Transparency.](#)

Output Gamma.

Gamma correction performed on renders, to match gamma of the operating system. Windows systems are calibrated for gamma 2.2, while Mac and Linux are calibrated for 1.8.

Input Gamma.

Inverse gamma correction performed on render input, which are textures, shader colors and light colors, so the render engine can work internally in the linear space. Gamma input value should be equal to output gamma, unless you are looking for gamma correction effects.

Clamp RGB.

When Clamp RGB is enabled, the color depth is reduced to a low dynamic range, for better anti-aliasing on high contrast areas. For instance, the anti-aliasing of visible area light sources and their reflection on specular surfaces. The examples below were made by

sevontheweb. The upper image has got aliasing issues in areas with a strong contrast, but colors are crisp. In the lower image *Clamp RGB* has been enabled. The anti-aliasing is better but colors are duller.



Other ways of solving this issue are:

- Increasing the render resolution and then scaling back to the desired resolution using a good interpolation algorithm.
- Multipass rendering with render post processing.

Clay Render.

Produces a clay render overriding all materials.

Related article: [Clay Render tutorial, by Bupla.](#)

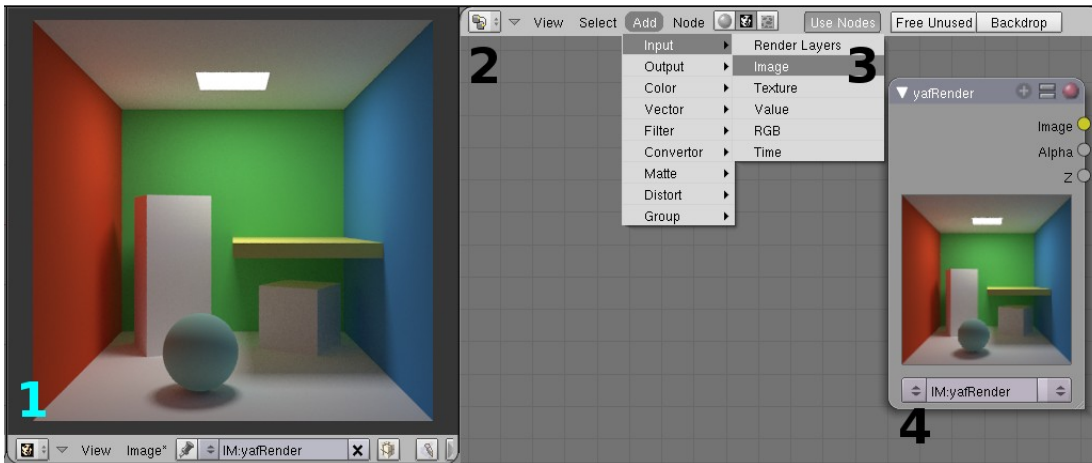
Threads.

Ray tracing is a highly parallelizable process. Most data structures in a typical ray tracer

can be shared by all available threads. In fact, Raytracing is among the few software fields that take full advantage of multi-core systems, the more cores the better. With this setting, users can fork the rendering calculation into several simultaneously running tasks, depending on CPU specs.

Result to Blender.

When this setting is enabled, apart from the render output, the image is saved as well in the *Blender UV/Image editor* [1]. This makes it possible to open the image in the *Blender Compositor nodes* as well [2]. Use *Compositor nodes > Add > Input > Image* [3] and *browse existing choices* button [4] to add the YafaRay render to the Blender compositor.



More information about Blender composite nodes in the link below:

http://wiki.blender.org/index.php/Doc:Manual/Composite_Nodes

Related Tutorial: [Compositing render passes.](#)

Autosave.

The render is automatically saved in the PNG format, with a time stamp in the file name so it isn't overwritten. The render file is located in the same folder as the Blender scene file.

Alpha on autosave/animations.

Images automatically saved with *Autosave* and with *Render anim* are saved in the PNG format, with RGB and alpha data instead of background.

Related article: [Render Windows Settings.](#)

Draw render parameters.

The most important render parameters are written in a badge in the rendered image. Use this feature to compare renders and to ask for advice in the YafaRay forums.

Output to XML.

The scene and render parameters are written in a YafaRay XML file. The XML file is located in the same folder than the Blender scene file.

Anti-aliasing.



Introduction.

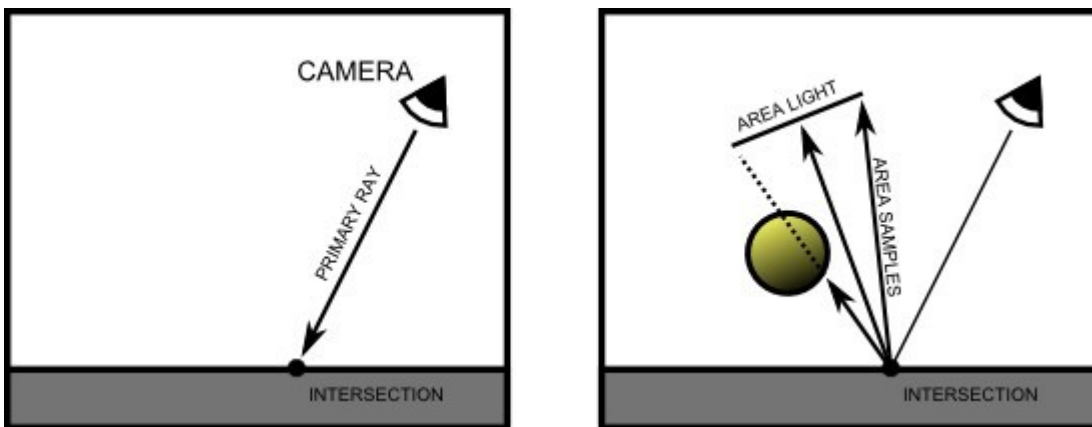
Aliasing is an approximation error in the discrete sampling of a 3D scene. The most usual manifestation of aliasing is jagged edges. Sources of aliasing are:

- Geometry.
- Very small details
- Textures
- Visible light sources
- Highlights in specular objects
- Contrast between bright and dark objects.
- Sharp shadows.

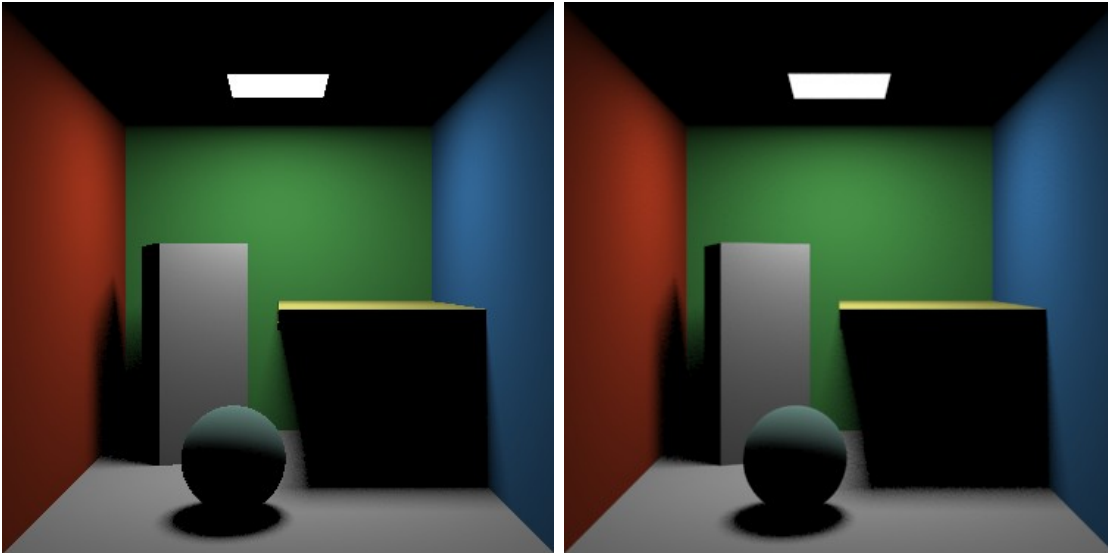
Anti-aliasing is a conjunt of sampling and reconstruction techniques used to mask or minimize the aliasing artifacts.

First anti-aliasing pass.

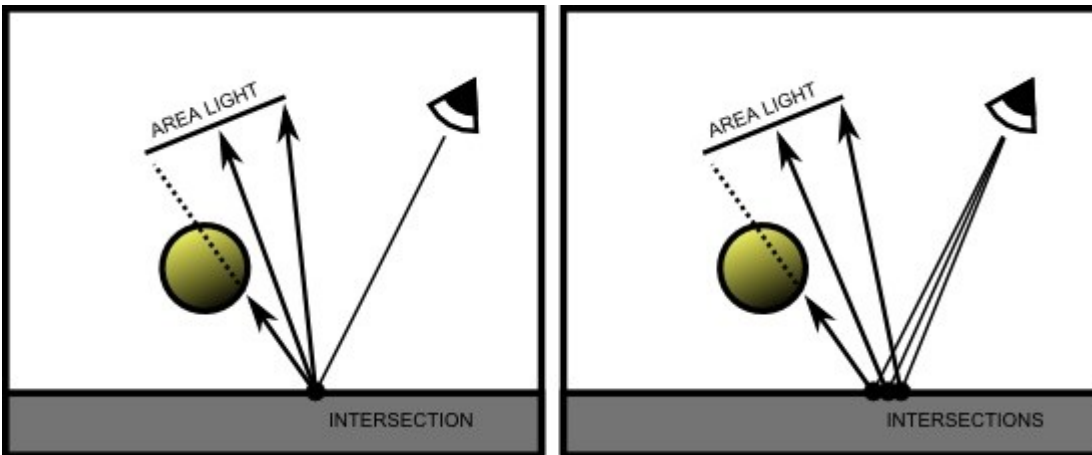
The scene is sampled with a first pass, shooting a number of rays per every camera pixel as specified in *AA samples*. These rays, also called primary rays, are shot from the camera and intersect with the scene. That intersection point is the origin of several types of secondary rays. For instance it is the origin of shadow rays used for sampling area lights and backgrounds. It is also the origin of pathtracing random paths and final gather directional samples.



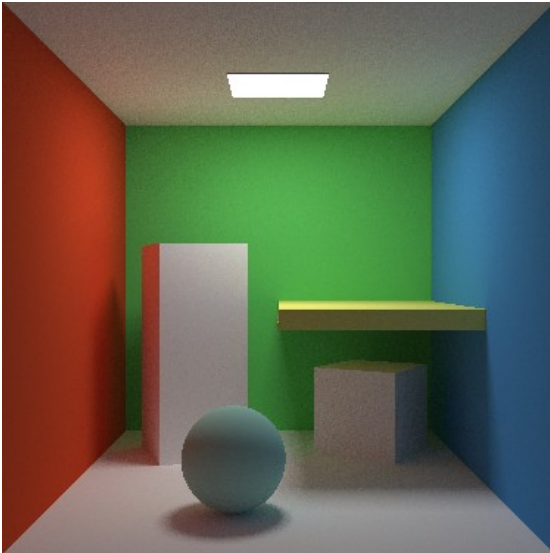
Anti-aliasing samples in the first pass are therefore connected to the sampling values used in area lights, backgrounds, path tracing and final gathering. The more *AA samples*, the more intersection points as origin of 'secondary' sampling methods. The first anti-aliasing pass is not only reducing aliasing artifacts, it is in fact a multiplication factor for area light, path tracing and final gathering samples. Look at the comparisons below:



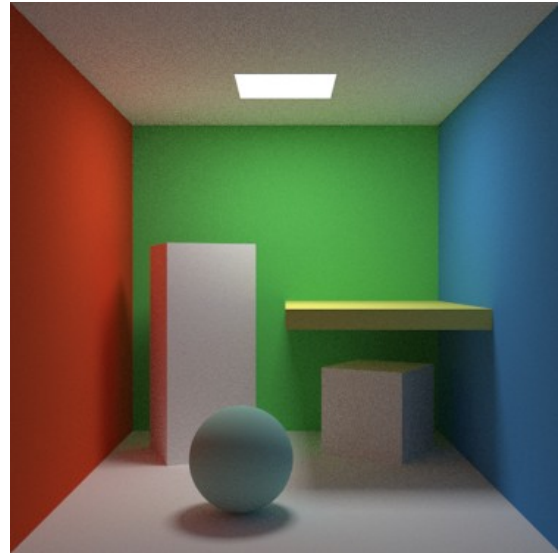
The image on the left is using 32 arealight samples, and 1 AA samples (1 AA passes, 1 AA samples, 1 AA inc. samples, time 9s.). The image on the right is using 1 arealight sample and 32 AA samples (1 AA passes, 32 AA samples, 1 AA inc. samples, time 29s). Notice how the arealight soft shadows are almost equal in both images. The number of shadow rays to sample the arealight is in fact the same in both images, but the method to achieve each of them is different, as explained below:



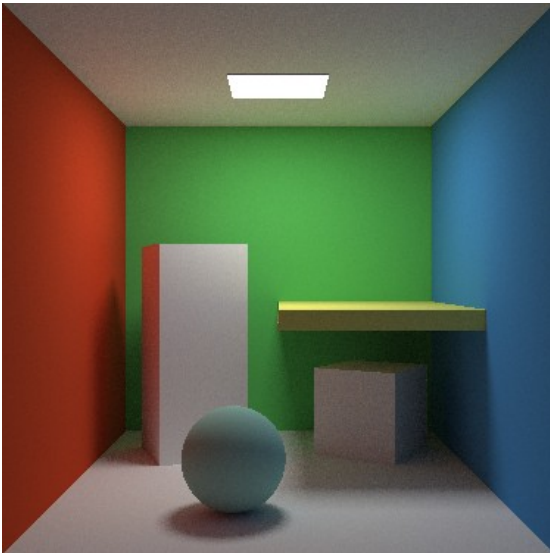
On the left, 3 arealight samples and 1 image sample are used. On the right, 1 arealight sample is used and 3 image samples. The number of shadow rays to sample the arealight is the same in both cases. However, the method on the left will be always more efficient, a bit less noisy and much faster in terms of render times. More examples below, with Photons & Final gather and Pathtracing:



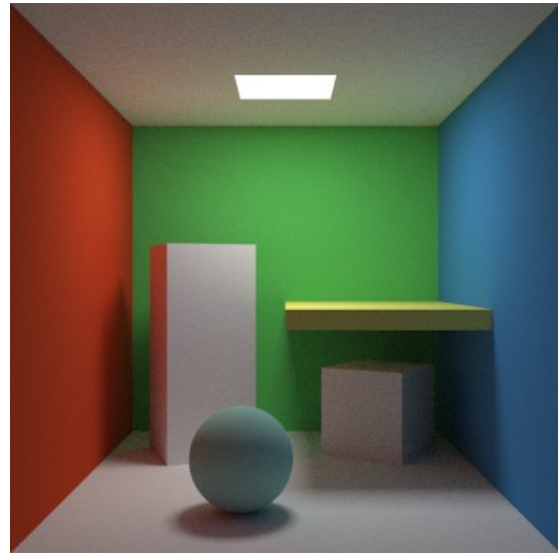
64 FG samples, 1 AA pass, 1 AA samples, 74s.



1 FG sample, 1 AA pass, 64 AA samples, 370s.



128 Pathtracing samples, 1 AA pass, 1 AA samples, 269 s.



1 Pathtracing sample, 1 AA pass, 128 AA samples, 996 s.

A conclusion is that in the first pass, it is more efficient to use lot of light samples and a few *AA samples*, rather than using a lot of *AA samples* and less lighting sampling. Therefore, the number of *AA samples* should be always kept low. A great deal of the anti-aliasing work as well as much of the noise removal could be performed in subsequent *AA passes* with adaptive sampling, as explained in the next section.

Related articles:

- [Lighting Methods Overview.](#)
- [Arealight.](#)
- [Path tracing.](#)
- [Final Gather.](#)
- [Texture background.](#)
- [Darktide Sunsky background.](#)
- [Sunsky background.](#)

- [Glossy reflection sampling.](#)

Adaptive sampling.

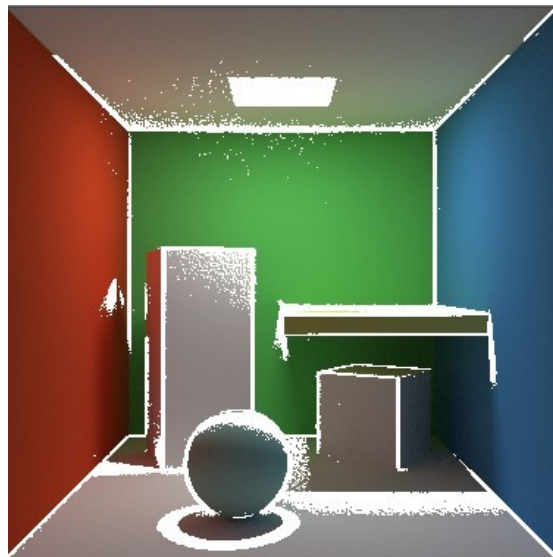
Anti-aliasing threshold is used to compare color of adjacent image samples. If their color discrepancy is higher than the limit defined by *AA Threshold*, then additional samples are taken in the second and subsequent *AA passes* until the discrepancy gets within limits. Sampling happens only in those problematic off-limit areas, without needing to sample everywhere. This technique is called adaptive sampling. The number of additional samples to be taken in problematic areas is defined by *AA inc. samples*.

If *AA Threshold* decreases, the number of areas needing additional sampling will increase, as well as the render time. Areas to be resampled decrease as new passes are performed and more areas get within the threshold. A good *AA Threshold* will focus anti-aliasing and noise reduction work only on problematic areas, without resampling the whole image in every pass.

If *AA Threshold* is 0, the whole image is resampled in every pass.



This render is using default 0.05 *AA Threshold*. The white points tell us what areas are being resampled in every pass.

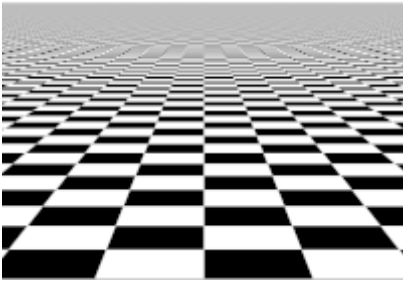


This render is using 0.01 *AA Threshold*. Not only more geometry is being resampled, but most of the arealight soft shadows as well, and some of the indirect lighting noise on the ceiling.

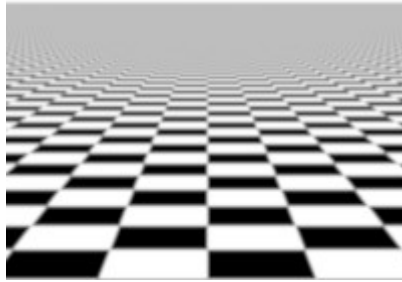
Filter type.

There are three reconstruction filters in YafaRay. These filters determine how multiple samples near a pixel are blended together. These filters are:

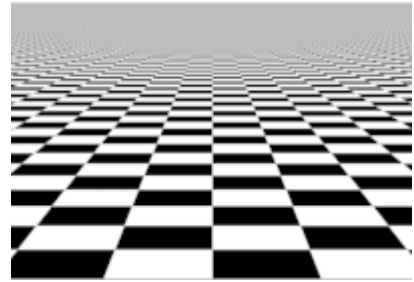
- *Box*. Equal weighting all samples. It is fast, but isn't efficient dealing with certain types of noise and produces post-aliasing.
- *Gaussian*. Gives good results. Tends to cause a slight blurring of the final image, but this blurring can help to mask remaining aliasing in the image.
- *Mitchell-Netravali*. Good results as well. Improves sharpness of the edges.



Box filter.



Gaussian filter.



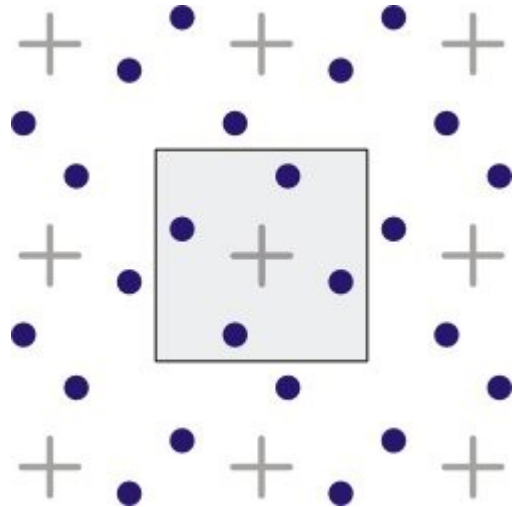
Mitchell-Netravali filter.

Filter Size.

Users can change the size of the reconstruction filter. Image samples are averaged inside a box around a pixel. How these pixels contribute to the final pixel value depends on the filter used (see previous article).

Lowering *AA Pixelwidth* means that less samples will be averaged, which makes renders sharper. If *AA Pixelwidth* increases, more image samples are averaged, and the result will be blurrier.

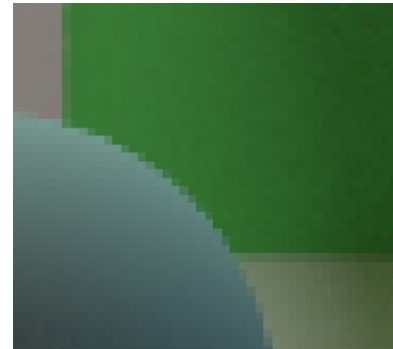
However, reducing pixel width means that the reconstruction filter is less effective against aliasing and high-frequency noise that could leak into the image. These issues can be solved by increasing the amount of anti-aliasing samples.



2.5 AA Pixelwidth, 49s.



Default 1.5 AA Pixelwidth, 54s.



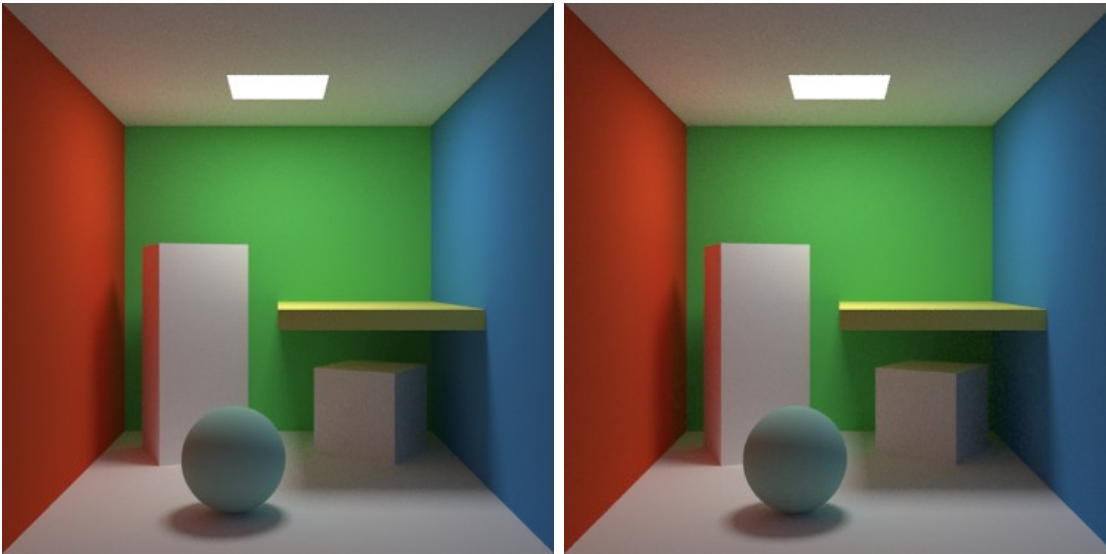
1.1 AA Pixelwidth, 82s.

Anti-aliasing strategy.

Based on concepts explained in the previous anti-aliasing sections, a good sampling strategy is divided in two steps, which are:

- The bulk of noise removal, particularly low-contrast noise in dark areas, is performed in the first *AA pass*, accumulating samples in light sources (arealights, background) and in indirect lighting algorithms (path tracing, final gathering), while using only a few *AA samples* (1 - 3) so a basic anti-aliasing is performed.

- The bulk of anti-aliasing is performed with adaptative sampling, using *AA Threshold* to determine the areas to sample. Adaptative sampling can also help us to remove high contrast noise on arealight shadows and areas with indirect lighting. Adaptative sampling is more efficient if we use several *AA passes* to do it. As some of the resampled areas get already within the *AA threshold*, a next *AA pass* will be usually faster.
- The idea is to exchange total amount of samples in the first pass (light sampling x *AA samples*) with adaptative sampling.
- If the *AA Threshold* is low (less than 0.005), then the best technique is using one *AA inc. sample*, and lots of *AA passes*. If a pixel needs 6 samples to look good, and we use 5 *AA inc. samples* in each pass, it means that we will use at least 10 samples in two passes to make that pixel look good. However, if we use only 1 *AA inc. samples*, that pixel will use just 6 samples to look good, with the 6th. *AA pass*.



- 32 Final Gather samples.
- 3 AA passes.
- 5 AA samples.
- 5 AA inc. samples.
- 0.05 AA Threshold.
- Time 178 s.

- 16 Final Gather samples.
- 5 AA passes.
- 1 AA samples.
- 2 AA inc. samples.
- 0.007 AA Threshold.
- Time 125 s.































Backgrounds

Table of content:

- [How backgrounds work.](#)
- [Single Color.](#)
- [Gradient.](#)
- [Texture.](#)
- [Darktide Sunsky.](#)
- [Sunsky.](#)

How backgrounds work

There are five types of backgrounds in YafaRay. The main purpose of the background is not only to map the scene background, but to work as a light source as well. The background lights the scene as a spherical area light casting light inside. However, background lighting is only possible with a limited set of Lighting Methods & Background types:

	Direct lighting	Pathtracing	Caustic photons Direct lighting Path tracing	Photon mapping diffuse and caustics	Bidirectional
Single color		 (1)		 (5)	
Gradient		 (1)		 (5)	
Texture (2)		 (1)		 (5)	
Texture - IBL (3)					
Darktide SunSky (4)					
SunSky (4)					

(1) Use *background* button enabled in path tracing settings.

(2) Use *IBL* button disabled or a RGB texture is used.

(3) Use *IBL* button enabled and a HDR texture is used.

(4) *Skylight* and *Sun* options enabled.

(5) *Use background* button enabled in photon mapping settings.

Related articles: [Lighting Methods Overview](#).

Single color

This background casts light in a single color. It can be useful for tests and simple setups. This background type works as a light source only in path tracing and photon mapping, by enabling the *Use background* option in the *Settings* section. Power of the background lighting can be controlled by the background color alone, or by a combination of background color and background power. *Power* setting is in fact a multiplier for background color.



Related articles:

- [Path tracing](#).
- [Photon Mapping](#).

Gradient

This option calculates a background composed of four colors: two sky colors above the horizon and two ground colors below it. The horizon position is calculated on camera specifications. Each pair of colors blend softly one each other to form a gradient. However, colors next to the horizon are not blended so a clear horizon line can be defined. This background can be very useful for:

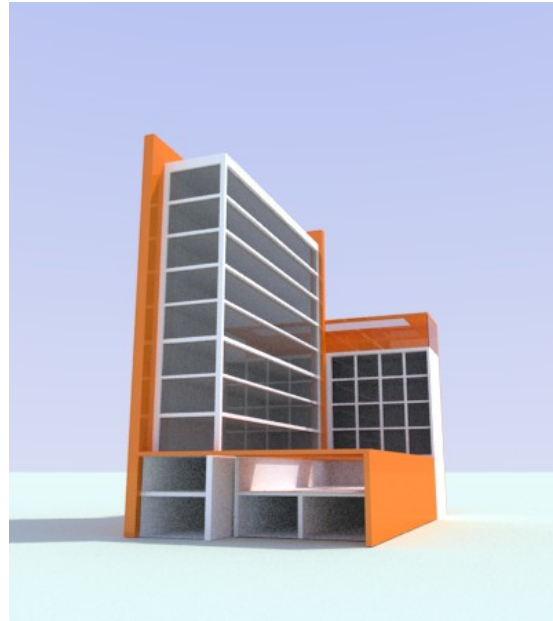
- Calculating the correct horizon line based on camera specifications, for compositing

works.

- Working out a 'fake' ground plane that extends the real ground mesh till the horizon line, as in the example below:



In this image, the background ground colors and the mesh used as a ground don't match.



In this image, the background ground colors have been corrected so it forms a continuous plane till the horizon line.

Background ground colors are given the same color as the ground mesh. The color used is not the mesh diffuse color, but the rendered one.

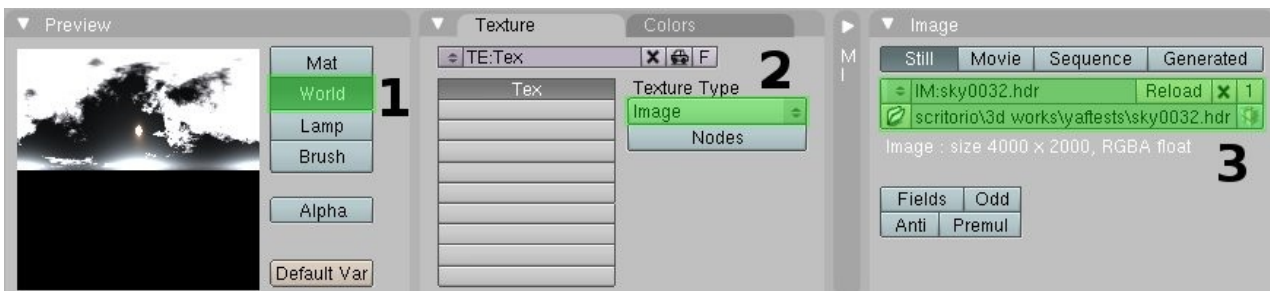
This background type works as a light source only in path tracing and photon mapping, by enabling the *Use background* option in the *Settings* section. Power of the background lighting can be controlled by the background colors alone, or by a combination of background colors and background power. *Power* setting is in fact a multiplier for background colors.

Related articles:

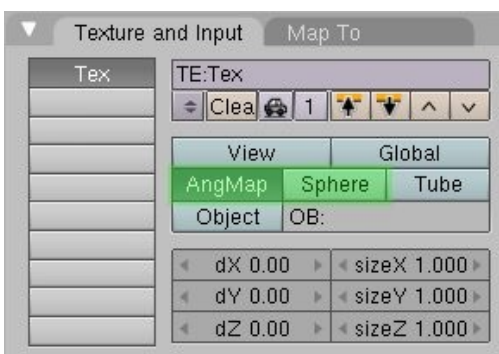
- [Path tracing.](#)
- [Photon Mapping.](#)

Texture

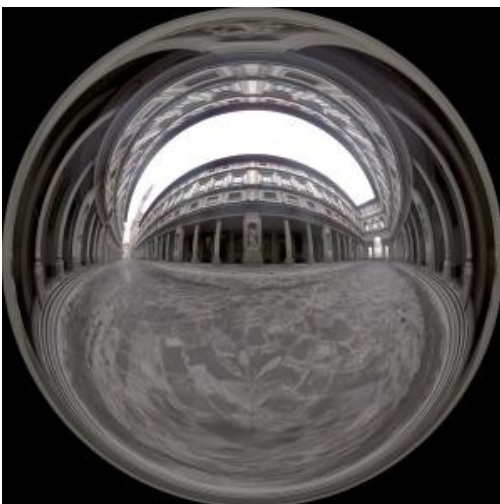
The main point of this background type is using HDR images to light the scene. Supported formats are **hdr** and **exr**.



The HDR images are loaded using Blender *Texture* buttons (F6) in *World* mode [1].



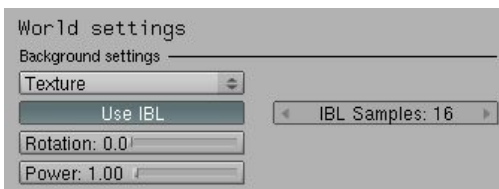
Once a texture is loaded, in the Blender *World* buttons (F8), we select the background coordinates in the *Texture and Input* panel, depending on the HDR image loaded (see images below). Supported coordinates are *AngMap* and *Sphere*.



Typical look of an **angular** HDR map, from <http://www.debevec.org/>



Typical look of a **spherical** HDR map, from <http://hdri.3dweave.com/library/park1.php>



YafaRay will use the texture datablock set in the Blender *World* buttons (F8). The other parameters needed to configure the HDR backgrounds are located in the YafaRay settings UI, under the *World* tab:

- *Use IBL*: Enable this option to use the background as a light source. A HDR image is needed.
- *Rotation*: Rotates the background.
- *Power*: A multiplier for background colors.
- *IBL Samples*: Defines the amount of samples taken to calculate the soft shadows. The more samples, the less noisy the shadows but the longer it will take to render. The total amount of light sampling depends as well on the anti-aliasing settings, as explained in [this section](#).

Related articles:

- [First anti-aliasing pass.](#)
- [Glossy reflection sampling.](#)

Works with:

- Direct lighting.
- [Path tracing.](#)
- [Photon Mapping.](#)

Darktide Sunsky

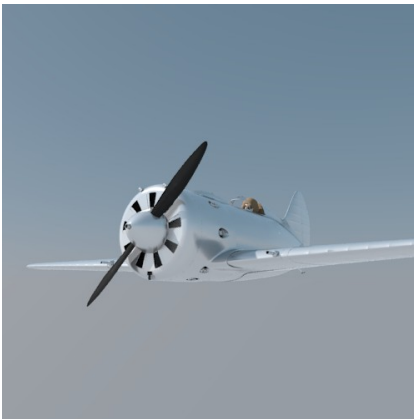
Sunsky is a background type that approximates the full spectrum of daylight for various atmospheric conditions. The main concepts of the Darktide sunsky model are:

Turbidity

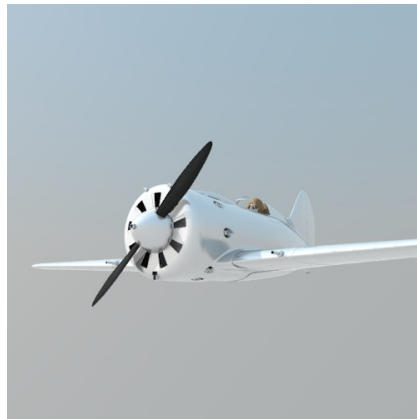
Turbidity is the haziness of a fluid caused by individual suspended particles that are generally invisible to the naked eye, similar to smoke in air. In another words, amount of dirtiness in the sky, the higher the more yellowish/reddish the sky becomes.

Recommended ranges are:

- (2, 3)= exceptionally clear sky
- (3, 5)= normal day
- (5, 12)= foggy day



Turbidity= 2



Turbidity= 5



Turbidity= 10

Horizon brightness

Brightness of horizon gradient controls brightness of the horizon relative to the sky.

Luminance of the horizon defines width of the gradient separating the horizon from the sky. Use similar values for both settings. Recommended ranges are:

- Use (0, 1.2) when turbidity is (2, 4).
- Use (0.9, 1.5) when turbidity is ≥ 5 .



- Turbidity= 2
- Horizon Gradient= 0.1
- Luminance of Horizon=0.1



- Turbidity= 2
- Horizon Gradient= 0.5
- Luminance of Horizon=0.5

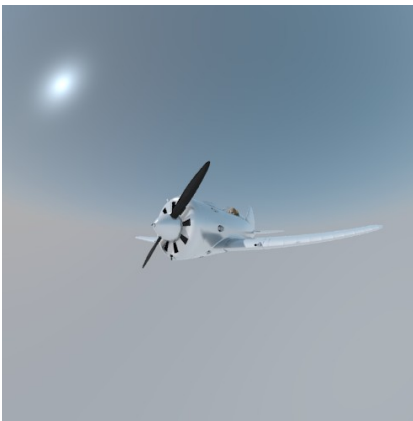


- Turbidity= 2
- Horizon Gradient= 1
- Luminance of Horizon= 1

Sun disk

Solar region intensity and *Width of circumsolar region* are settings to control brightness and size of the sun disk, respectively. The bigger these values are, the brighter and smaller the sun disk will be. To see the sun disk, use similar values for both settings in a range of (10, 40). You can achieve other results though, like a small sun disk but not so bright, by using different values in each setting.

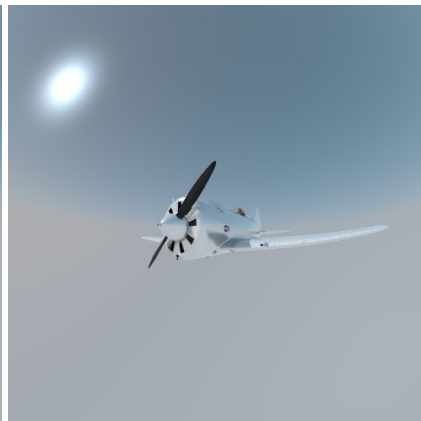
A brighter bigger sun means that highlights will be bigger and brighter on specular surfaces.



- Turbidity= 2
- Solar region intensity= 10
- Width of c. region= 10



- Turbidity= 2
- Solar region intensity= 20
- Width of c. region= 20

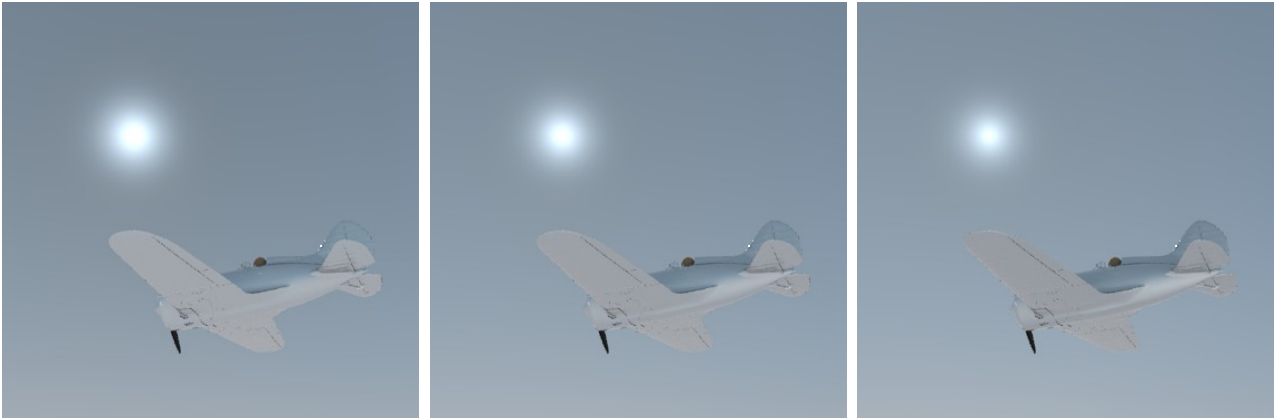


- Turbidity= 2
- Solar region intensity= 40
- Width of c. region= 10

Backscattered light

Backscattered light is the light scattered back by the atmosphere turbidity. In practice, higher values produce a brighter sky around a consequently smaller sun disk. The

backscattered light contributes less the lower the *Turbidity* values are, though.



- Turbidity= 2
- SCI=15
- WCR=20
- Backscattered light= 1

- Turbidity= 2
- SCI=15
- WCR=20
- Backscattered light= 5

- Turbidity= 2
- SCI=15
- WCR=20
- Backscattered light= 10

Sun position controls



- *From (get angle)*: Takes sun angle from a selected sun lamp in the Blender scene.
- *From (get position)*: Takes sun position from a selected sun lamp in the Blender scene.
- *From (update sun)*: Exports Sun position and angle to a selected sun lamp in the Blender scene.
- *Direction Sphere*: Drag your mouse cursor over this sphere to interactively select the direction of the sun rays.
- *Direction coordinates*: Coordinates for direction of the sun rays.

Altitude

With this setting, we can give the camera a different altitude relative to the background center. If *Altitude* increases, the background horizon lowers. This option can be useful to render objects from a camera which is above the ground level, while still getting a correct position of the background horizon.



Altitude= 0

Altitude= 0.25

Altitude= 0.50

Night

This feature renders the scene at night, with the Sun acting as the Moon.



Example of a Darktide Sunsky in **Night** mode.

Other Settings



- *Add real sun*: A realistic Sun model is added to the scene, taking into account Sun position and power settings.
- *Sun Power*: Power of the Sun added.
- *Samples*: Samples for both the Sun and the Skylight. The more samples the less noise and the better glossy reflections, but the more render time as well.
- *Add Skylight*: The sky casts light.
- *Sky Brightness*: Independent brightness control for background sky colors; it does not affect amount of cast light.
- *Power*: Multiplier for background sky colors.

Related articles:

- [First anti-aliasing pass.](#)
- [Glossy reflection sampling.](#)

Works with:

- Direct lighting.
- [Path tracing.](#)
- [Photon Mapping.](#)

Sunsky

Sunsky is a similar concept to Darktide Sunsky, albeit simpler and handling colors in a different way. It is also a background type that approximates the full spectrum of day light for various atmospheric conditions. In these tests we have used *Sun power*= 0.50 and *Skylight power*= 1.0:

Turbidity

Turbidity is the haziness of a fluid caused by individual suspended particles that are generally invisible to the naked eye, similar to smoke in air. In another words, amount of dirtiness in the sky, the higher the more yellowish/reddish becomes the sky.



Turbidity= 3



Turbidity= 6



Turbidity= 12

Horizon brightness

A (HorBrght) controls brightness of the horizon relative to the sky.



- Turbidity= 3
- *A (HorBrght)*= 0.10



- Turbidity= 3
- *A (HorBrght)*= 0.60



- Turbidity= 3
- *A (HorBrght)*= 1.30

Luminance of gradient near the horizon

B (HorSprd) defines width of the gradient separating the horizon from the sky.



- Turbidity= 3
- *A (HorBrght)*= 1.00



- Turbidity= 3
- *A (HorBrght)*= 1.00



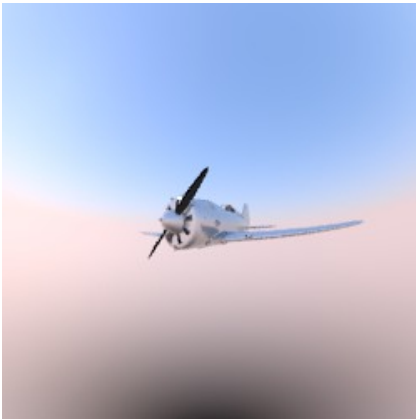
- Turbidity= 3
- *A (HorBrght)*= 1.00

- B (HorSprd)= 0.50
- B (HorSprd)= 1.00
- B (HorSprd)= 1.50

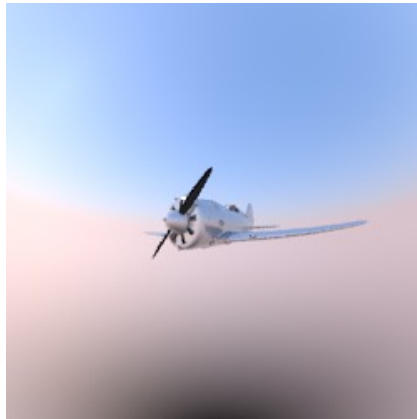
Sun Brightness and Size

C (SunBrght) and *D (Sunsze)* are settings to control brightness and size of the sun area, respectively. The bigger these values are, the brighter and bigger the sun area will be.

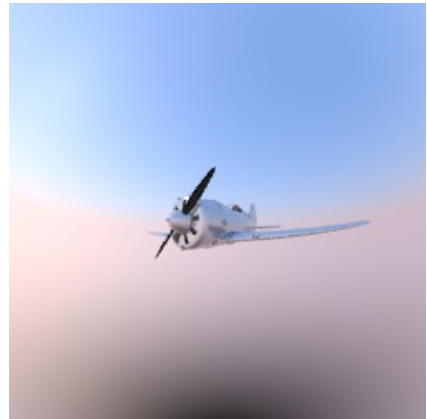
A brighter and bigger sun means that highlights will be bigger and brighter on specular surfaces.



- C (SunBrght)= 0.2
- D (Sunsze)= 0.2



- C (SunBrght)= 0.6
- D (Sunsze)= 0.2



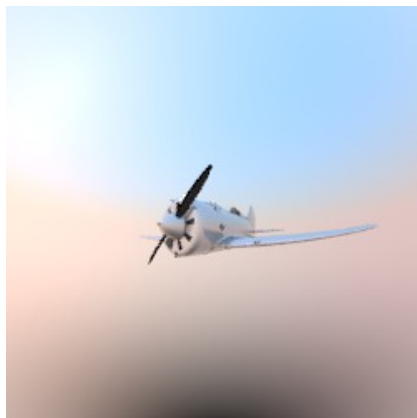
- C (SunBrght)= 1.0
- D (Sunsze)= 0.2

Back scattered light

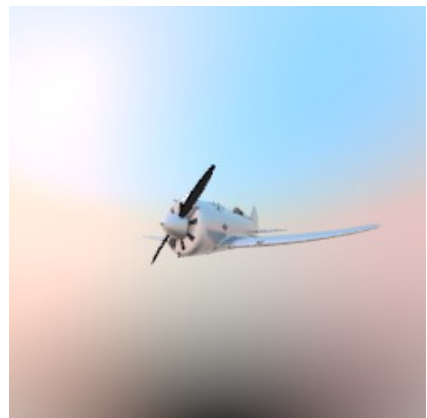
E (Backlight) is the light scattered back by the atmosphere turbidity. In practice, higher values produce a brighter sky around the sun disk. The backscattered light contributes less the lower the *Turbidity* values are, though.



- Turbidity= 5
- E (Backlight)= 0.5



- Turbidity= 5
- E (Backlight)= 2



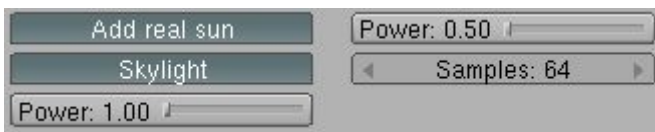
- Turbidity= 5
- E (Backlight)= 5

Sun position controls



- *From (get angle)*: Takes sun angle from a selected sun lamp in the Blender scene.
- *From (get position)*: Takes sun position from a selected sun lamp in the Blender scene.
- *From (update sun)*: Exports Sun position and angle to a selected sun lamp in the Blender scene.
- *Direction Sphere*: Drag your mouse cursor over this sphere to interactively select the direction of the sun rays.
- *Direction coordinates*: Coordinates for direction of the sun rays.

Other Settings



- *Add real sun*: A realistic Sun model is added to the scene, taking into account Sun position and power settings.
- *Sun Power*: Power of the Sun added.
- *Samples*: Samples for both the Sun and the Skylight. The more samples the less noise and the better glossy reflections, but the more render time as well.
- *Add Skylight*: The sky casts light.
- *Power*: Multiplier for background sky colors.

Related articles:

- [First anti-aliasing pass.](#)
- [Glossy reflection sampling.](#)

Works with:

- Direct lighting.
- [Path tracing.](#)
- [Photon Mapping.](#)

Volumetrics

Table of content:

- [Introduction.](#)
- [The Basics.](#)
 - [Workflow.](#)
 - [Volume types.](#)
- [Volume Integrator.](#)
 - [Step Size.](#)
 - [Other Volume Integrator Settings.](#)
 - [Examples.](#)
- [Volume Regions.](#)
 - [Volume Region tips.](#)
 - [Absorption and Scattering.](#)
 - [Examples.](#)
- [Volume Region Types.](#)
 - [UniformVolume](#)
 - [ExpdensityVolume](#)
 - [Example](#)
 - [NoiseVolume](#)
 - [NoiseVolume Settings](#)
 - [The NoiseVolume Texture](#)

Introduction.

YafaRay's volumetric features provide a simulation of light interacting with particles suspended in a region of space. A common volumetrics scenario is the "beam of light" effect, for example when light enters a dusty room through a window creating a visible shaft of light. YafaRay uses a realistic physics-based model to render volumetrics, and provides a basis for creating not just believable beams of light, but also smoke, clouds, fog, and other volumetric effects.

Important: Use PNG or any other lossless format to save renders with volumetrics. JPG compression kills volumetrics detail.

The Basics.

This section gives a brief overview of rendering volumetrics in YafaRay. Specific parameters and settings are covered in subsequent sections.

Workflow.

There are several basic steps necessary to render volumetrics in YafaRay. A brief outline of the workflow is:

1. In the *World* tab of the YafaRay UI select the single scatter integrator in the volume integrator settings.
2. Create a box-shaped object (the volume region) in the scene to act as a container for the volumetric effect.
3. With the volume region object selected go to the *Objects* tab of the YafaRay UI and activate the *Enable volume* button.
4. Ensure at least one light source in the scene illuminates the volume region.

Volume Types.

An important characteristic governing the final appearance of volumetric media is density. It is intuitively clear that a greater particle density in a volume region equates to a more pronounced volumetric effect. Also, a method to control density is needed to render effects such as clouds or smoke, which have a variable density. So in YafaRay there are three different volume types for a region based on how density is controlled:

UniformVolume: a uniform density volumetric effect, useful for dust, fog, mist, etc.



ExpDensityVolume: like a *UniformVolume*, but decreases the density of the volume as height (in positive z direction) increases. Useful for ground fog, etc.

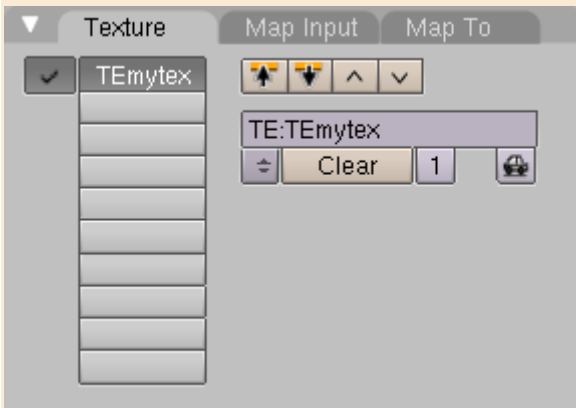


NoiseVolume: lets a blender procedural texture control the density of the volume, good for clouds, smoke, etc. Note that this method must be considered a "fake" compared to a realistic cloud or smoke density simulator that is unavailable in YafaRay.



One of these volume types can be selected for the volume region once *Enable volume* is selected.

Important: when using NoiseVolumes the blender texture used must be called TEmytex, so that the texture setup looks like this:

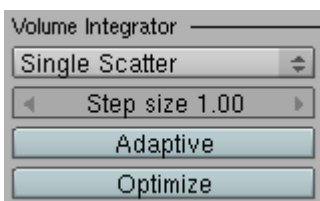


If this exact name is not used blender will crash during the render process and all work will be lost. This limitation will be removed in the future.

The following sections discuss volumetric settings and parameters in detail.

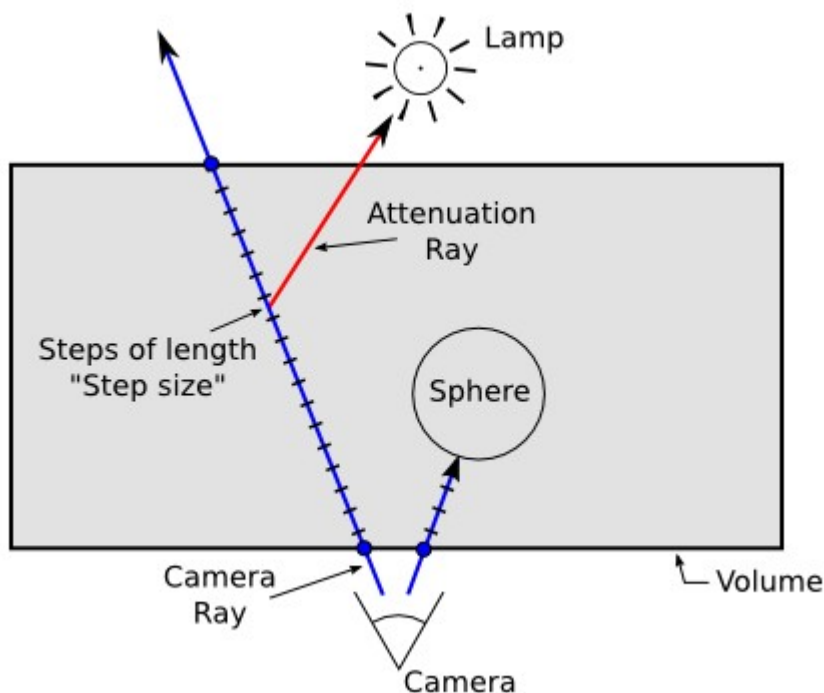
Volume Integrator.

The volume integrator settings are located in the *World* tab of the Yafaray UI, underneath the background settings. The settings here for the Single Scatter integrator pertain to the global precision of volumetric calculations. These settings have a considerable impact on render speed, and so they are important to control the balance of speed versus quality.



Step Size.

This parameter is the main control for the precision of volumetric rendering. Consider the following example setup with a volume, a camera, and a light source:



When a viewing ray shot from the camera (the blue lines) intersects a volume, the portion of the ray inside the volume is subdivided into "steps," with the length of each step being the value of the step size parameter (in blender units). At each step shadow rays are traced to all the light sources in the scene (the red line) to determine how much light reaches the step, and then from this information the total light contribution to the viewing ray is computed. This calculation at each step is called *attenuation*.

So the following conclusions about step size can be drawn:

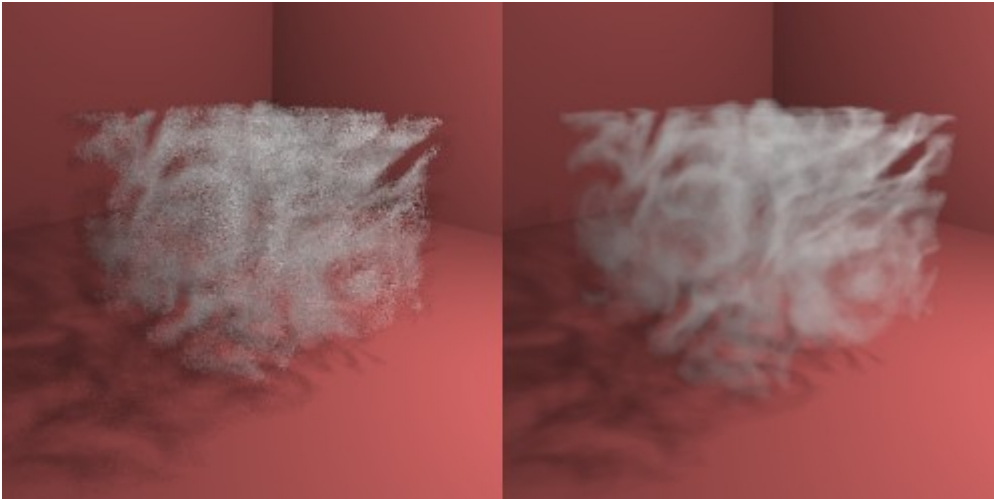
- Decreasing the step size value means more steps along the viewing ray and more attenuation calculations. This leads to a more accurate result, at the expense of increased render times.
- Longer viewing ray segments inside a volume have more steps and therefore require more attenuation calculations than shorter ray segments.

Other Volume Integrator Settings.

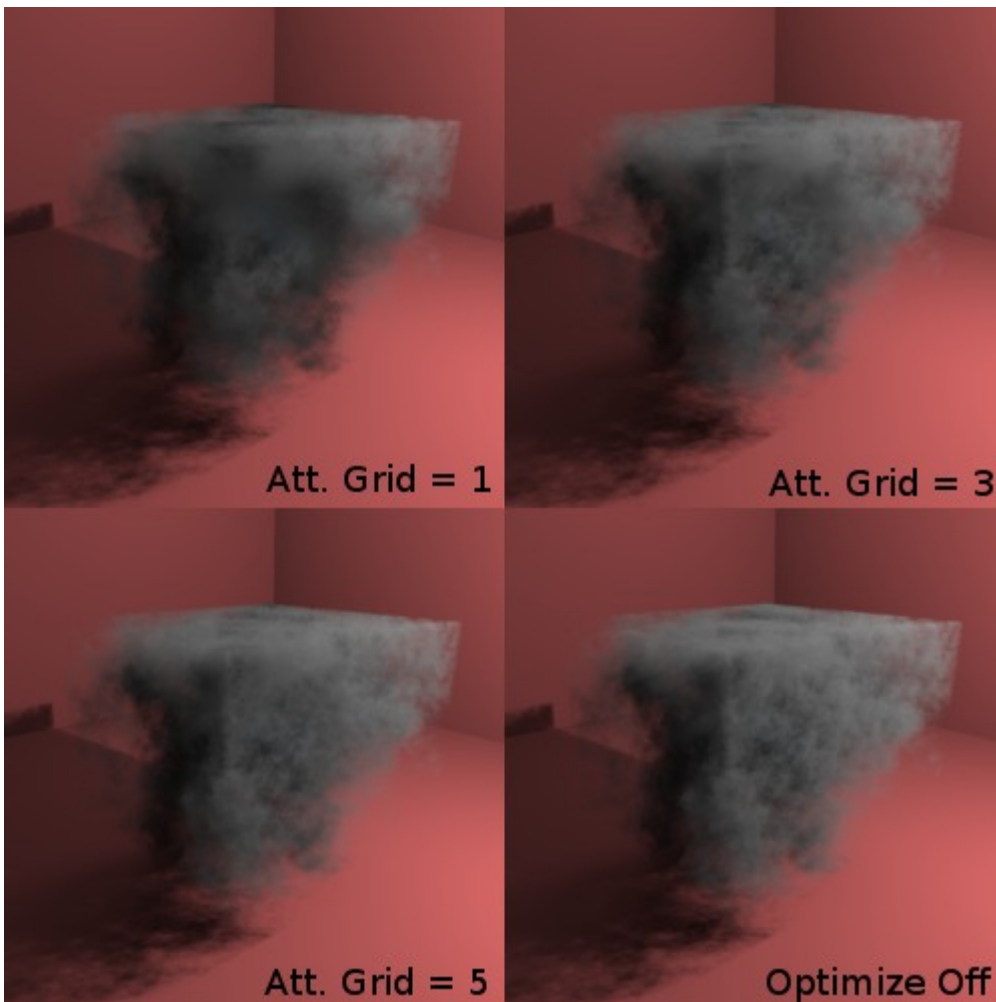
Here are the rest of the volume integrator settings:

- *Adaptive*: optimizes stepping calculations for NoiseVolumes, resulting in reduced render times for certain situations. There is no visual change to the render result by using this option.
- *Optimize*: attenuation calculations can be computationally expensive for certain volumes. This setting speeds this process by precomputing attenuation in the entire volume at a 3d grid of points. These results are then interpolated and used as attenuation values at each step of a viewing ray, rather than calculating attenuation directly at this point. This function is very helpful for reducing render times for NoiseVolumes and ExpDensityVolumes.
- *Att. Grid Resolution*: since the attenuation grid is an estimation, lower values can result in reduced detail in shaded areas. For many volumes a value of 2 or 3 is sufficient, but for darker ones a higher value may be necessary.

Examples.

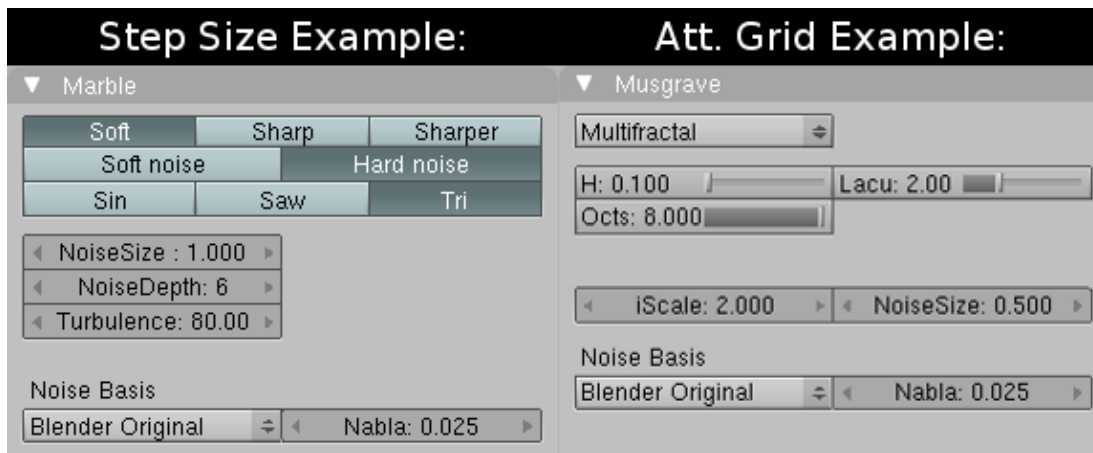


The volume on the left is grainy because the step size is too high (0.5). A lower step size on the right (0.1) corrects the issue, but is of course slower (four times slower in this case).



Volumes with *Att. Grid Resolution* of 1, 3, and 5, plus another with *Optimize* off for reference. Notice the improved shading detail with higher grid resolutions. Even at a grid resolution of 5 the render time was about one third compared to the render without *Optimize* on.

For reference, the texture settings used for the NoiseVolumes in the previous examples are shown below:



Volume Regions.

The volume region defines the general shape and characteristics of a volumetric effect. Volumetrics only occur inside of a volume region, allowing control over placement of a volumetric effect in a scene. When a volume region is the active object in a scene the volume settings in the *Objects* tab of the YafaRay UI control specific aspects of how a volume renders.

Volume Region Tips.

Some quick notes on volume regions:

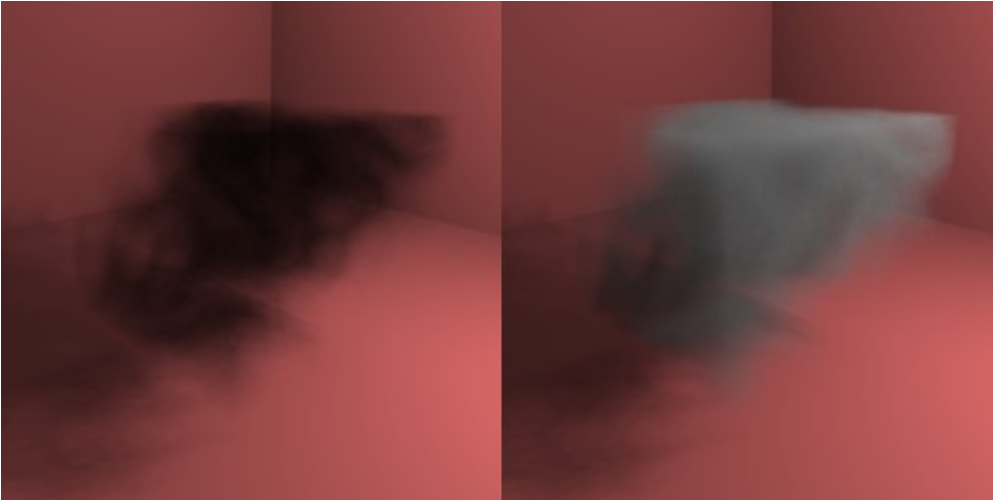
- The size of a volume region has a direct effect on render times. From the discussion on step size it is clear that a bigger volume requires more steps along a viewing ray and more attenuation calculations than a smaller volume. So it is important for efficient rendering of volumetrics that the volume region be as small as possible.
- As noted earlier, volume regions cannot be rotated in a scene. The edges of the volume region object must remain aligned with the global coordinate axes.

Absorption and Scattering.

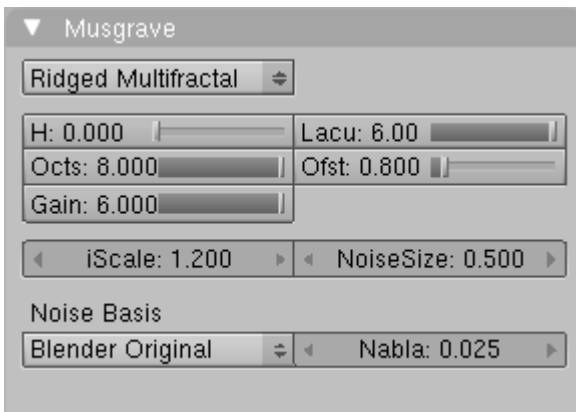
The volume region settings in the *Objects* tab control specific qualities related to the look of a volumetric effect. Each type of volume region (UniformVolume, ExpDensityVolume, NoiseVolume) may have its own unique set of parameters, but every volume region type has parameters controlling two fundamental properties -- absorption (σ_a) and scattering (σ_s).

- σ_a (absorption): some volumes absorb a certain amount of light that they receive, for example black smoke from a fire exhibits a lot of absorption.
- σ_s (scattering): scattering occurs when light traveling along a path encounters a particle in a volume and is redirected to a new path. Scattering affects how the camera detects light in two ways: light rays heading towards the camera can be redirected away from the camera, and light rays not originally aimed at the camera can be scattered in its direction. The "beam of light" effect is the result of light being scattered towards the camera.

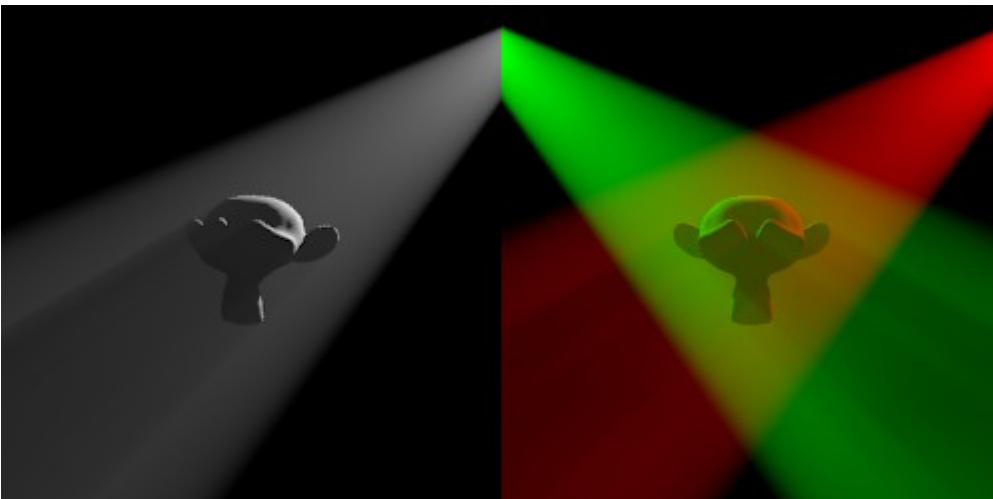
Examples.



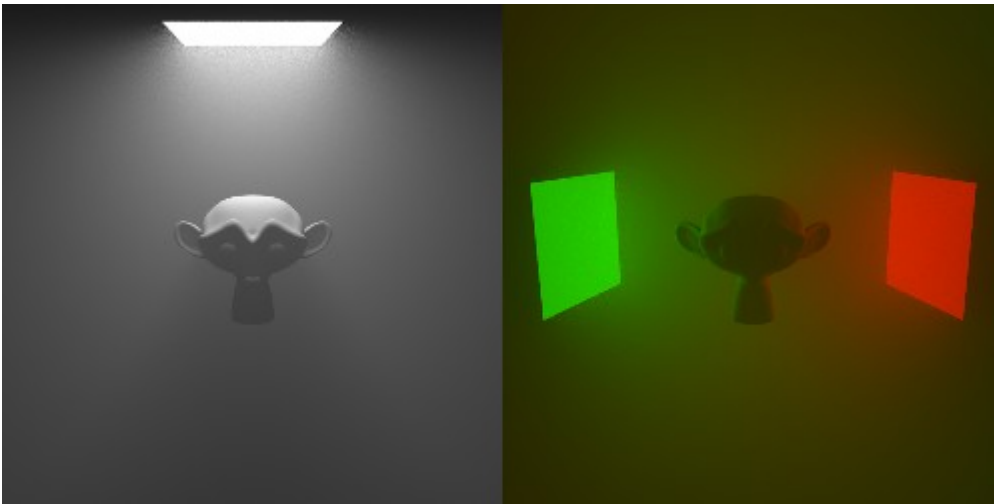
On the left is a volume with only absorption ($\sigma_a = 0.5$), and has a dark smoky appearance. On the right is the same volume but with scattering added ($\sigma_s = 0.05$), and has a denser, more complex look.



For reference, these are the texture settings used to create the previous example image.



These show the effect of scattering on light from spot lights. Notice how the the monkey object blocks light from the lamps, creating a shadowed area in the visible cone of light. Also note the interaction of the colored lamps in the volume.



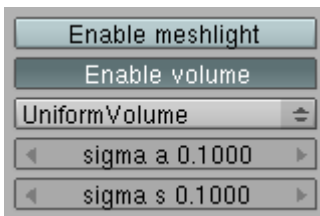
These show the result of area lights in a volume. Here light from the area lamps is scattered widely across the volume.

Volume Region Types.

This section discusses the different types of volume regions, including parameters and workflow specific to a particular volume type.

UniformVolume.

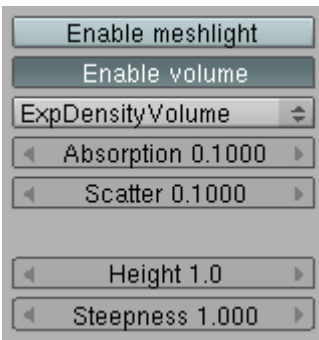
The UniformVolume type provides a basic uniform density volumetric effect within the volume region.



The parameters for the UniformVolume are limited to controls for absorption and scattering. These parameters were discussed in a previous section.

ExpDensityVolume.

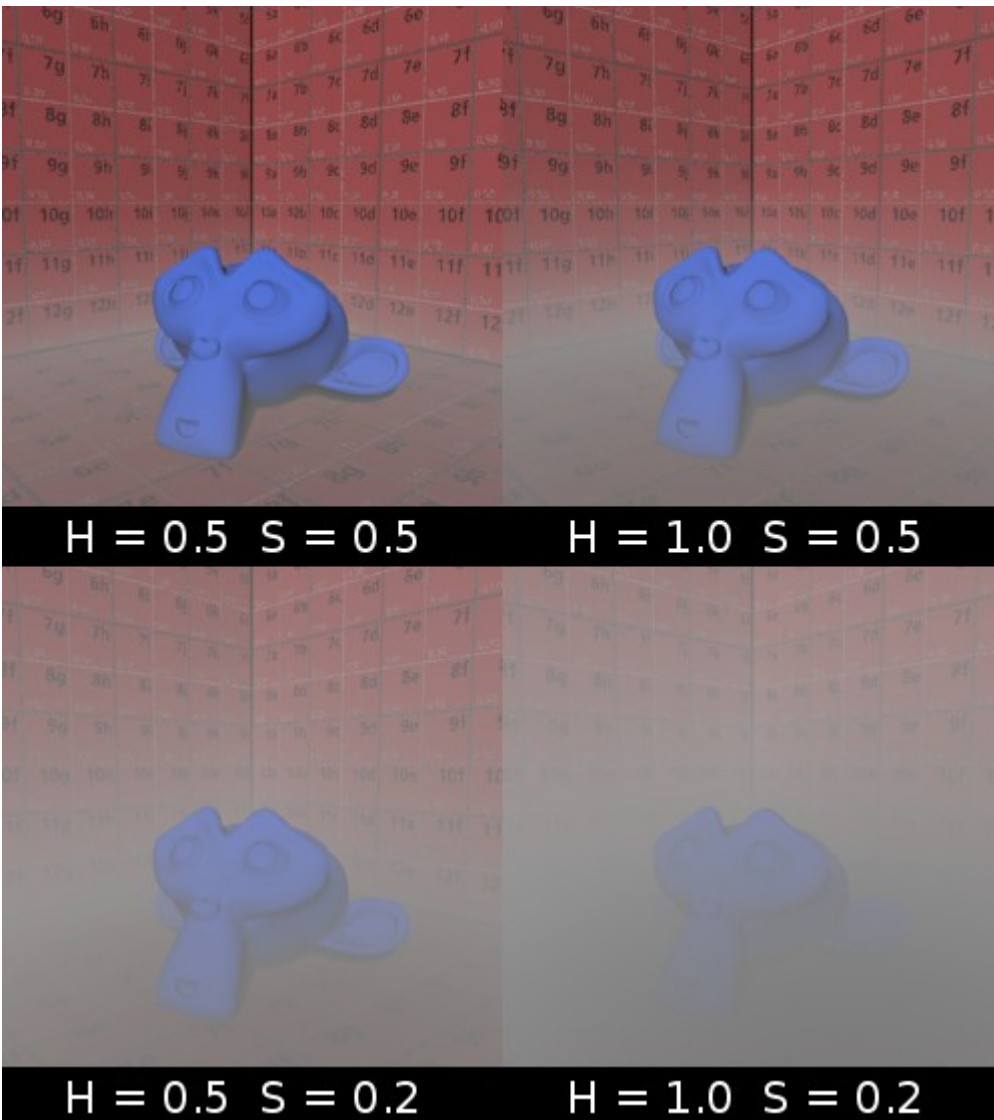
The ExpDensityVolume defines a density within the volume region that decreases with increased height (in the direction of the positive z-axis). This decrease is an exponential rate of decay, controllable using the two parameters *height* and *steepness*. This volume type is useful for effects such as ground fog where the effect is most pronounced close to the ground.



In addition to parameters for absorption and scattering, the ExpDensityVolume has the following settings:

- *height*: controls the density of the volume before it starts to fall off. Bigger height values will result in a denser volumetric effect close to the ground.
- *steepness*: controls how quickly the density falls off. Higher steepness values will result in a quicker transition between the areas of maximum density and no density.

Examples.



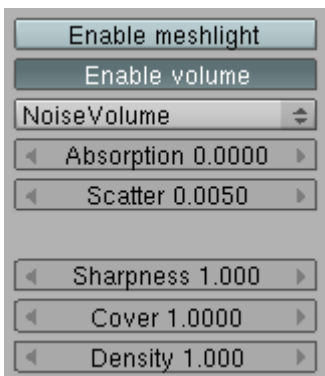
In the above example H is height and S is steepness. Notice that increasing the value for

height results in a denser volumetric effect near the floor, while decreasing the value for steepness results in a longer and slower transition from dense to not dense.

NoiseVolume.

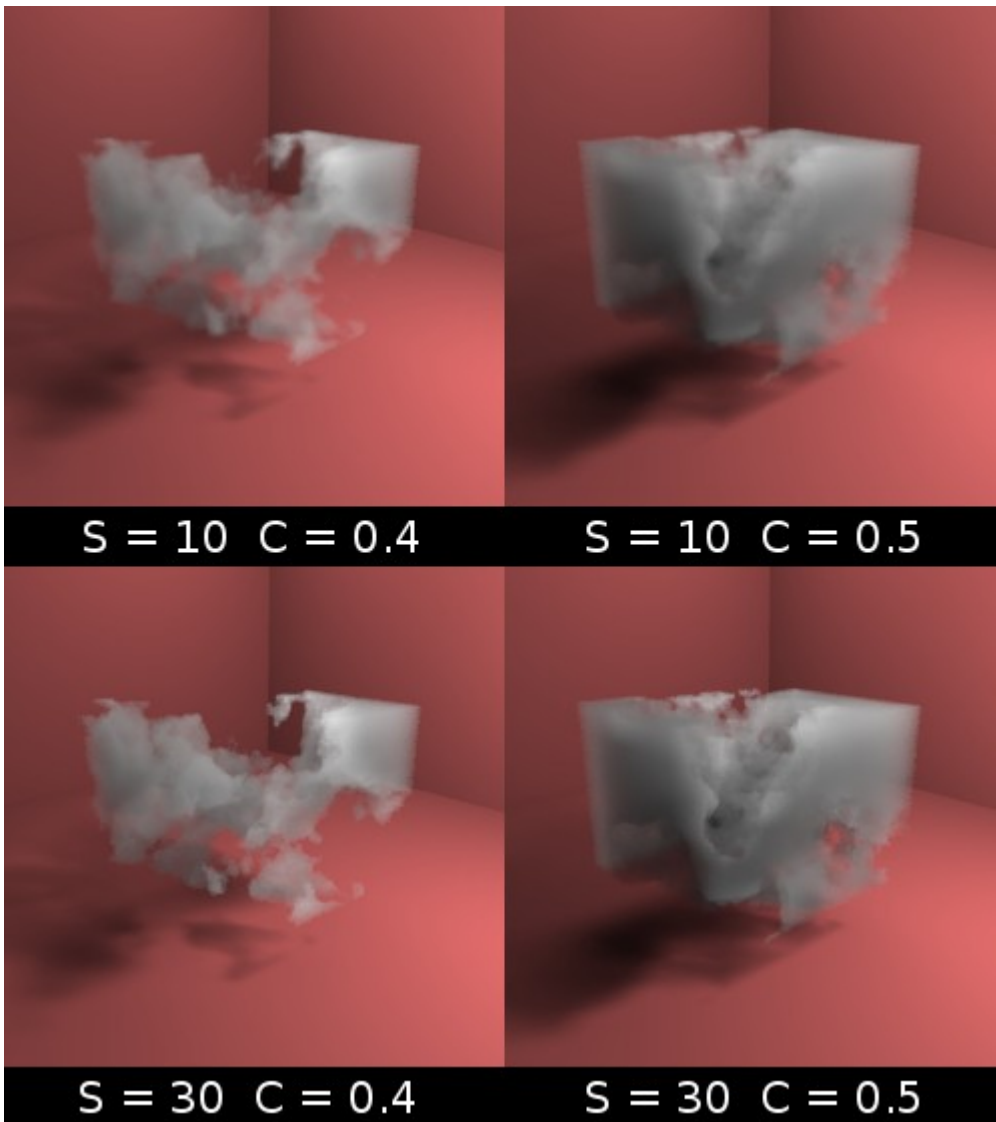
The NoiseVolume type allows the density of the volume to be controlled by a blender procedural texture. YafaRay translates the intensity values of a blender texture into density, with darker values on the texture corresponding to lower density and lighter values corresponding to higher density. This provides a method to approximate the look of different variable-density effects such as steam, smoke, clouds etc.

NoiseVolume Settings.

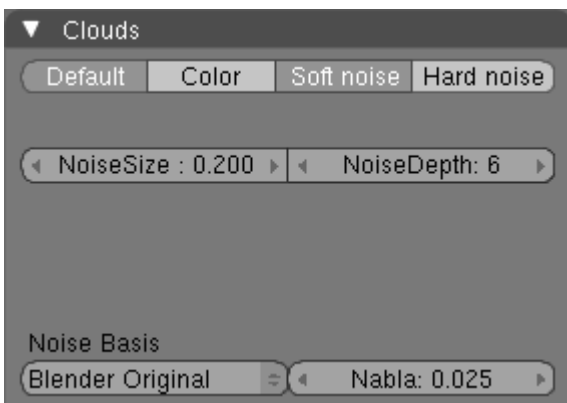


The NoiseVolume has the following unique parameters:

- *Sharpness*: controls how sharp a NoiseVolume looks at the border between areas of high and low density. With higher values of sharpness a NoiseVolume will look crisper and more distinct at the edges.
- *Cover*: has the effect of defining what percentage of a procedural texture maps to zero density. As values for cover decrease from 1, a greater percentage of the texture's lowest intensity values map to zero density, creating a sparser volumetric effect with more empty spaces.
- *Density*: a global density multiplier. At the default density value of 1 a procedural texture can sometimes correspond to densities too high or too low to give a useful output. Adjusting this density parameter can bring the result back to a workable range.



The above example illustrates the effect of both the sharpness and cover parameters (S=sharpness, C=cover). For those interested, the texture settings used to create this example volume are included below:

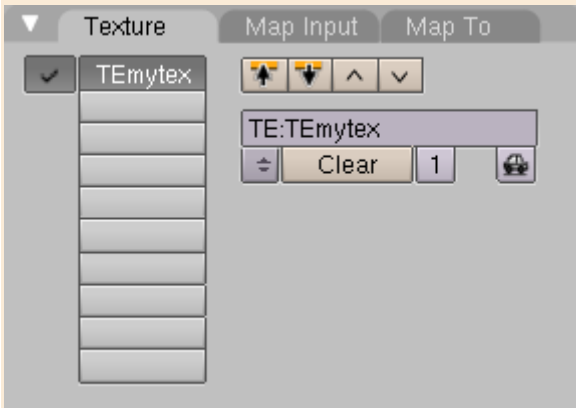


The NoiseVolume Texture.

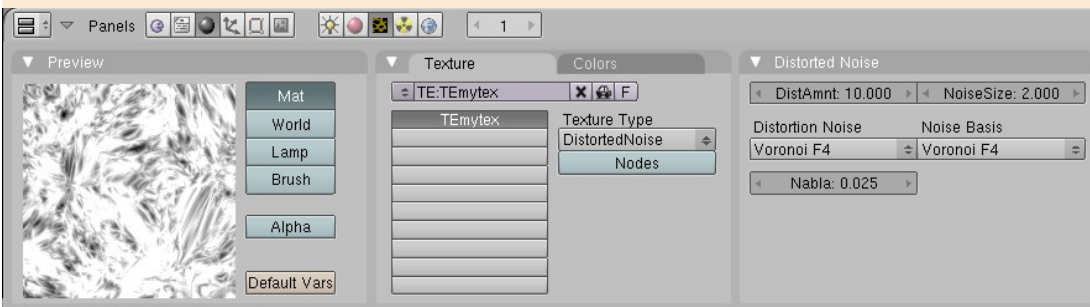
The blender procedural texture used to control the density of a NoiseVolume is the primary control for how this volume type looks. First, even though this information was mentioned

earlier, it is so important that it will be repeated:

Important: when using NoiseVolumes the blender texture used must be called TEmytex, so that the texture setup looks like this:



If this exact name is not used blender will crash during the render process and all work will be lost. This limitation will be removed in the future.



All settings relevant to the use of a procedural texture for a NoiseVolume can be found in blender's texture buttons panel (F6). Any procedural texture supported by YafaRay can be used as a texture for a NoiseVolume. More information on YafaRay and procedural textures is available in the [Procedural Textures](#) section of the manual.