Chapter 1. Overview of Materials in Cycles

In this chapter, we will cover the following recipes:

- An overview of material nodes in Cycles
- An overview of procedural textures in Cycles
- How to set the World material
- Creating a mesh-light material
- Using volume materials
- Using displacement

Introduction

Cycles' materials work in a totally different way than in Blender Internal.

In Blender Internal, you can build a material by choosing a diffuse and a specular shader from the **Material** window, by setting several surface options, and then by assigning textures (both procedurals and image maps as well) in the provided slots. All of these steps make one complete material. After this, it's possible to combine two or more of these materials by a network of nodes, thereby obtaining a lot more flexibility in a shader's creation. However, the single materials themselves are the same as those set through the **Material** window—shaders made for a scan-line-rendering engine—and their result is just an approximation of the simulated absorption-reflection behavior of light on a surface.

In Cycles, the approach is quite different. All the names of the closures describing surface properties have a **Bidirectional Scattering Distribution Function** (**BSDF**), which is a general mathematical function that describes the way in which light is scattered by a surface in the real world. It's also the formula that path tracers such as Cycles use to calculate the rendering of an object in a virtual environment. Basically, light rays are shot from the camera. They bounce on the objects in the scene and keep on bouncing until they reach a light source or an empty background (which, in Cycles, can emit light as well). For this reason, a pure path tracer such as Cycles can render in reasonable times an object set in an open environment. The rendering times increase a lot for closed spaces, for example, furniture set inside a room, because light rays can bounce on the floor, the ceiling, and the walls many times before reaching one or more light sources.

In short, the main difference between the two rendering engines is due to the fact that, while in Blender Internal, the materials use all the traditional shader tricks of a scan-line rendering engine such as the simulated specular component, the Cycles rendering engine is a path tracer that tries to mimic the real behavior of a surface as closely as possible as if the surface were real. This is the reason we don't have an arbitrary Specular factor simulating the reflection point of light on the surface in Cycles, but instead have a glossy shader that actually mirrors the light source and the surroundings to be mixed with other components in different ratios. Thus the glossy shader behaves in a more realistic way.

Just for explanatory purposes, in this module, I will refer to the more or less blurred point of light created by the reflection of the light source on a mirroring glossy surface as *specularity*.

Be aware that the rendering speed in Cycles depends on the device you use to render your scenes—CPU or GPU. This means that basically, you can decide to use the power of the CPU (default option) or the power of the graphic card processor, the GPU.

To set the GPU for the rendering, perform the following steps:

1. Call the **Blender User Preferences** panel (*Ctrl* + *Alt* + *U*) and go to the **System** tab, the last tab to the right of the panel.
2. Under the **Compute Device** tab to the bottom-left corner of the panel, select the option to be used for computation. To make this permanent, click on the **Save User Settings** button or press *Ctrl* + *U*. Now close the **Blender User Preferences** panel.
3. In the **Properties** panel to the right of the screen, go to the **Render** window and, under the **Render** tab, it's now possible to configure the GPU of the graphics card instead of the default CPU (this is possible only if your graphic card supports CUDA, that is, for NVIDIA graphic cards. OpenCL, which is intended to support rendering on AMD/ATI graphics cards, is still in a very incomplete and experimental stage, and therefore, not very usable yet).

A GPU-based rendering has the advantage of literally increasing the Cycles' rendering speed several times, albeit with the disadvantage of a small memory limit, so it's not always possible to render big complex scenes made up of a lot of geometry. In such cases, it's better to use the CPU instead.

There are other ways to reduce the rendering times and also to reduce or avoid the noise and the fireflies (white dots) produced in several cases by the glossy, transparent, and light-emitting materials. All of this doesn't strictly belong to shaders or materials. By the way, you can find more information related to these topics at the following addresses:

Information on *Cycles Render Engine* can be found at http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles.

More information on *Reducing Noise* is available on the Cycles wiki page, at http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Reducing_Noise.

A list of supported graphic cards for Cycles can be found at https://developer.nvidia.com/cuda-gpus.

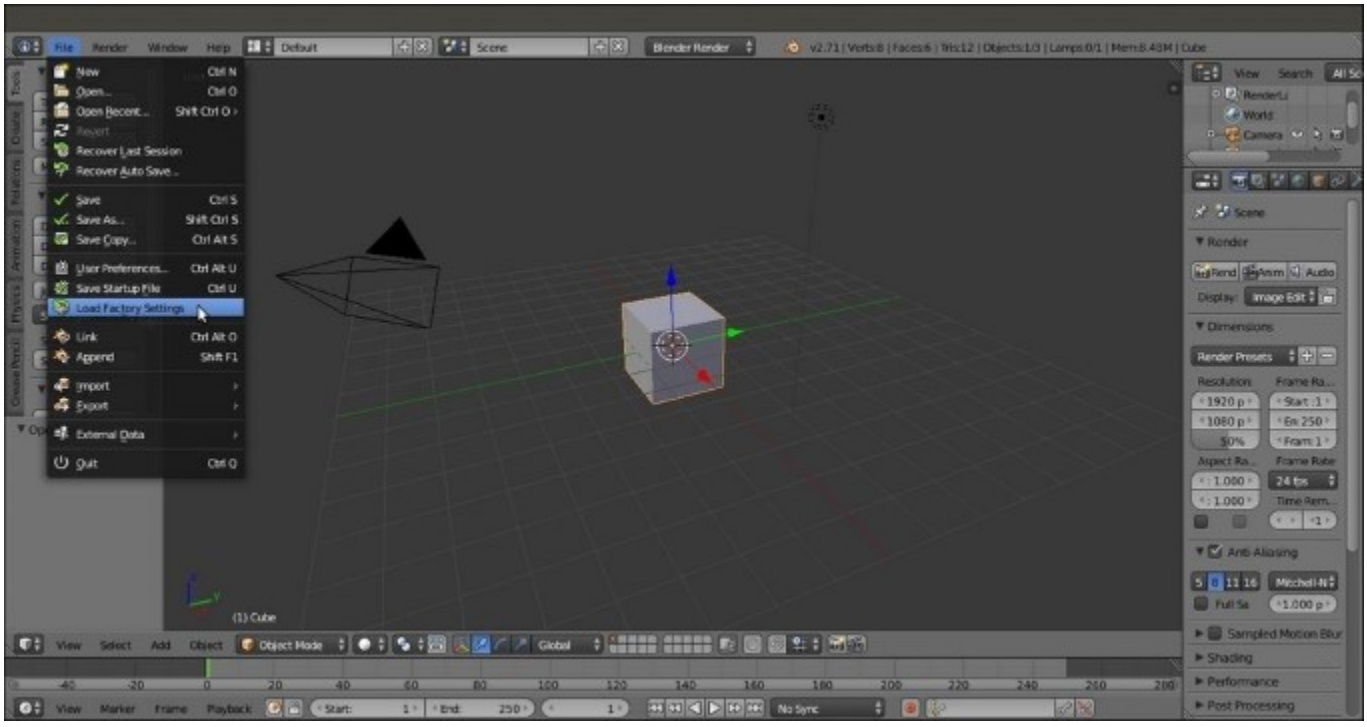
Material nodes in Cycles

A Cycles material is basically made up of distinct components named **shaders**. They can be combined to build even more complex surface or volume shaders.

In this recipe, we'll have a look at the basic, necessary steps required to build a basic surface Cycles material, to activate the rendered preview in the 3D window, and to finally render a simple scene.

Getting ready

In the description of the following steps, I'll assume that you are using Blender with the default factory settings. If you aren't, start Blender and just click on the **File** menu item in the top main header bar to select **Load Factory Settings** from the pop-up menu, as shown in the following screenshot:



The default Blender interface and the File pop-up menu with the Load Factory Settings item

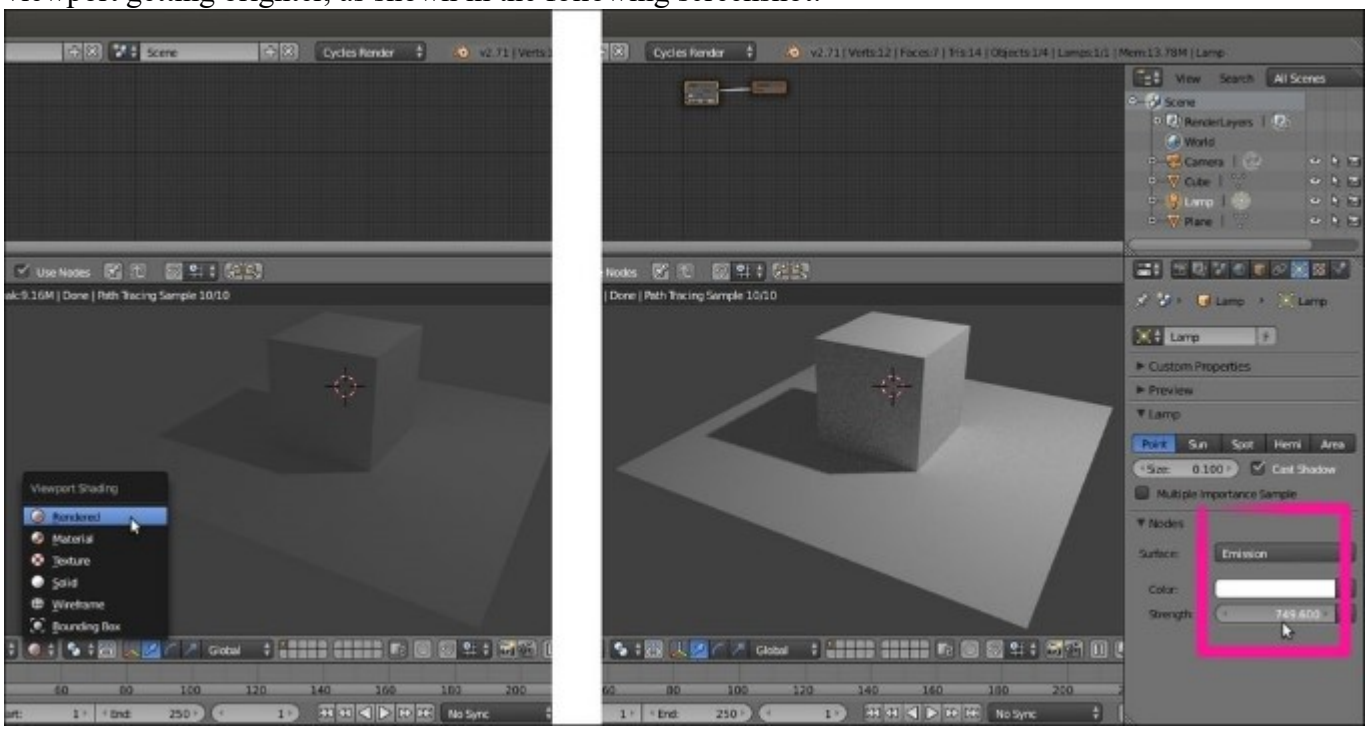
Now perform the following steps:

1. In the upper menu bar, switch from **Blender Render** to **Cycles Render** (hovering with the mouse on this button shows the engine to use to render a label).
2. Now split the 3D view into two horizontal rows, and change the upper row to the **Node Editor** window by selecting the menu item from the **Editor Type** button in the left corner of the bottom bar of the window. The **Node Editor** window is, in fact, the window we will use to build our shaders by mixing the nodes (actually, this is not the only way, but we'll see this later).

3. Put the mouse cursor in the 3D view and add a Plane under the Cube (press *Shift* + *A* and navigate to **Mesh** | **Plane**). Enter **Edit Mode** (press *Tab*), scale it 3.5 times bigger (press *S*, enter *3.5*, and then press *Enter*) and go out of **Edit Mode** (press *Tab* again). Now move the Plane one Blender unit down (press *G*, then *Z*, then enter *-1*, and finally, press *Enter*).
4. Go to the little icon (**Viewport Shading**) showing a sphere in the bottom bar of the 3D view and click on it. A menu showing different options appears (**Bounding Box**, **Wireframe**, **Solid**, **Texture**, **Material** and **Rendered**). Select **Rendered** from the top of the list (or press the *Shift* + *Z* shortcut) and watch your Cube being rendered in real time in the 3D viewport.
5. Now you can rotate and translate the view or the Cube itself, and the view gets updated in real time (the speed of the update is restricted only by the complexity of the scene and the computing power of your CPU or graphics card).

Let's learn more by performing the following steps:

1. Select the **Lamp** item in the **Outliner** window (by default, it's a Point lamp).
2. Go to the **Object data** window under the **Properties** panel on the right-hand side of the screen.
3. Under the **Nodes** tab, click on **Use Nodes** to activate a node system for the selected light in the scene. This node system is made by an **Emission** shader connected to a **Lamp Output** node.
4. Go to the **Strength** item, which is set to `100.000` by default, and start increasing the value. As the intensity of the Lamp increases, you will see the Cube and the Plane rendered in the viewport getting brighter, as shown in the following screenshot:



The Viewport Shading menu with the Rendered item and the Lamp Object data window with the Strength slider

Tip

Downloading the example code

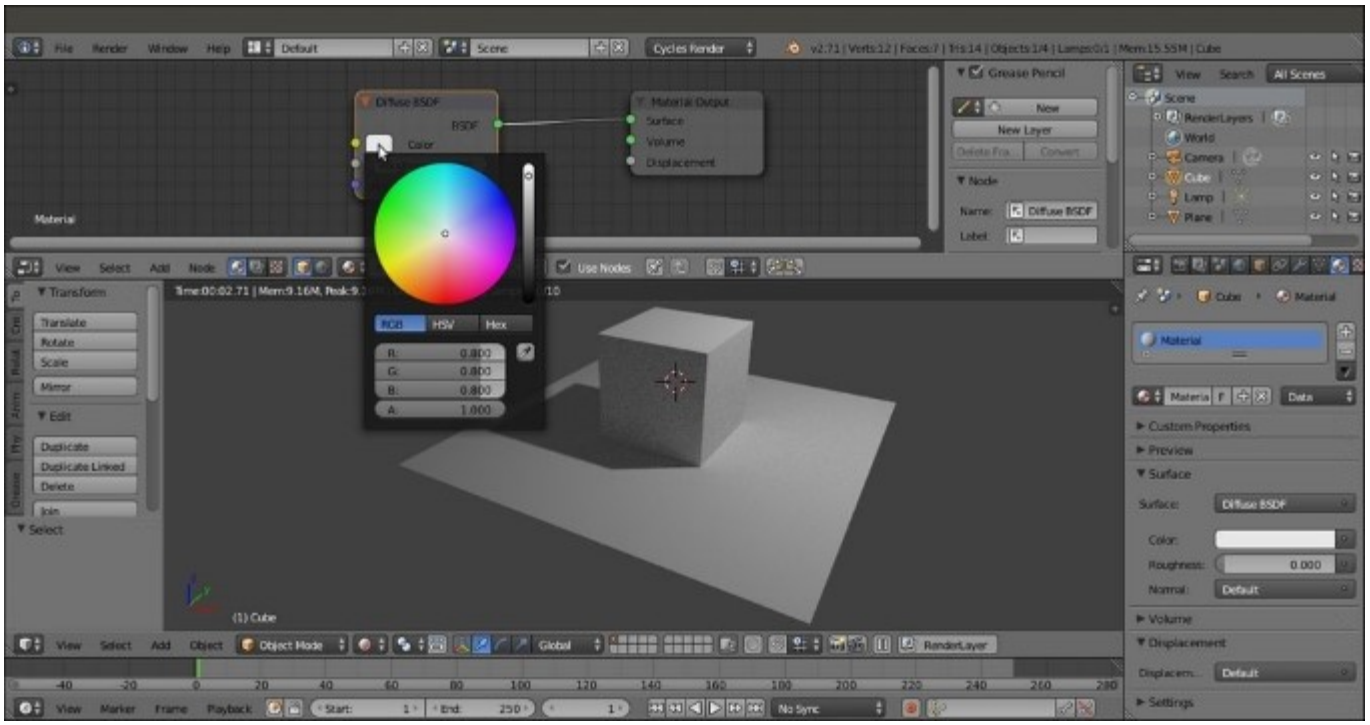
You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

How to do it...

We just prepared the scene and took the first look at one of the more appreciated features of Cycles since its first inclusion in Blender—the real-time-rendered preview (which, by the way, is now also available in Blender Internal but seems to work faster in Cycles).

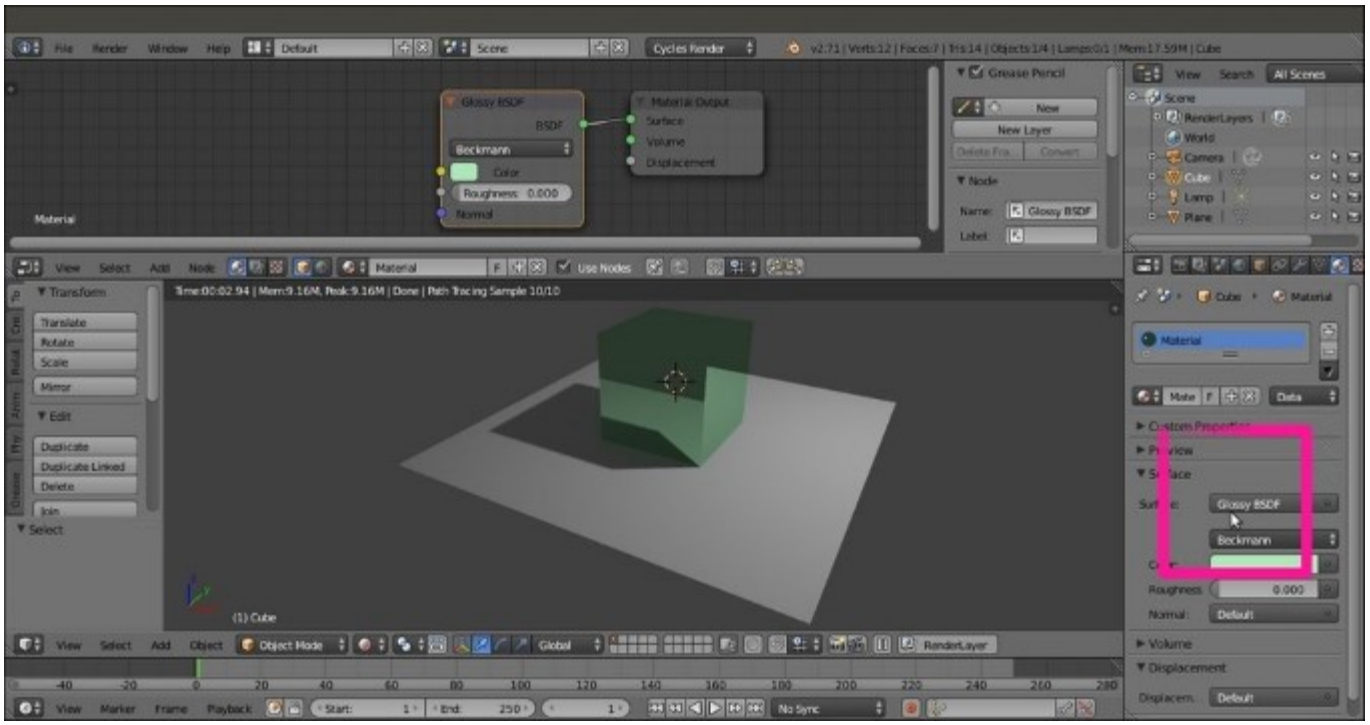
Now let's start with the object's materials:

1. Select the Cube to assign the shader to by clicking on the item in the **Outliner** window or right-clicking directly on the object in the **Rendered** viewport (but be aware that in the **Rendered** mode, the object selection outline usually around the mesh is not visible because it's obviously not renderable).
2. Go to the **Material** window under the **Properties** panel. The Cube already has a default material assigned (as you can precisely see under the **Surface** subpanel within the **Material** window). By the way, you need to click on the **Use Nodes** button under the **Surface** subpanel to activate the node system for the material. Instead of this, you can also check the **Use Nodes** box in the toolbar of the **Node Editor** window.
3. As you check the **Use Nodes** box, the content of the **Surface** tab changes, showing that a **Diffuse BSDF** shader has been assigned to the Cube and that, accordingly, two linked nodes have appeared inside the **Node Editor** window. The **Diffuse BSDF** shader is already connected to the **Surface** input socket of a **Material Output** node.
4. Put the mouse cursor in the **Node Editor** window, and by scrolling the mouse wheel, zoom in to the **Diffuse BSDF** node. Click on the **Color** rectangle. A color wheel appears, where you can select a new color to change the shader color by clicking on the wheel itself or by inserting the **RGB** values (note that there is also a color sampler and the alpha channel value, although the latter, in this case, doesn't have any visible effect on the object material's color).



The color wheel of a Diffuse shader node in the Node Editor window and the Rendered 3D viewport preview

5. The Cube rendered in the 3D preview changes its material's color in real time. You can even move the cursor in the color wheel and watch the rendered object switching the colors accordingly. Set the object's color to a greenish color by changing its **RGB** values to `0.430`, `0.800`, and `0.499`, respectively.
6. Go to the **Material** window, and under the **Surface** tab, click on the **Surface** button, which is showing the **Diffuse BSDF** item at the moment. From the pop-up menu that appears, select the **Glossy BSDF** shader item. Now the node changes in the **Node Editor** window, and so does the Cube's material in the **Rendered** preview, as shown in the following screenshot:



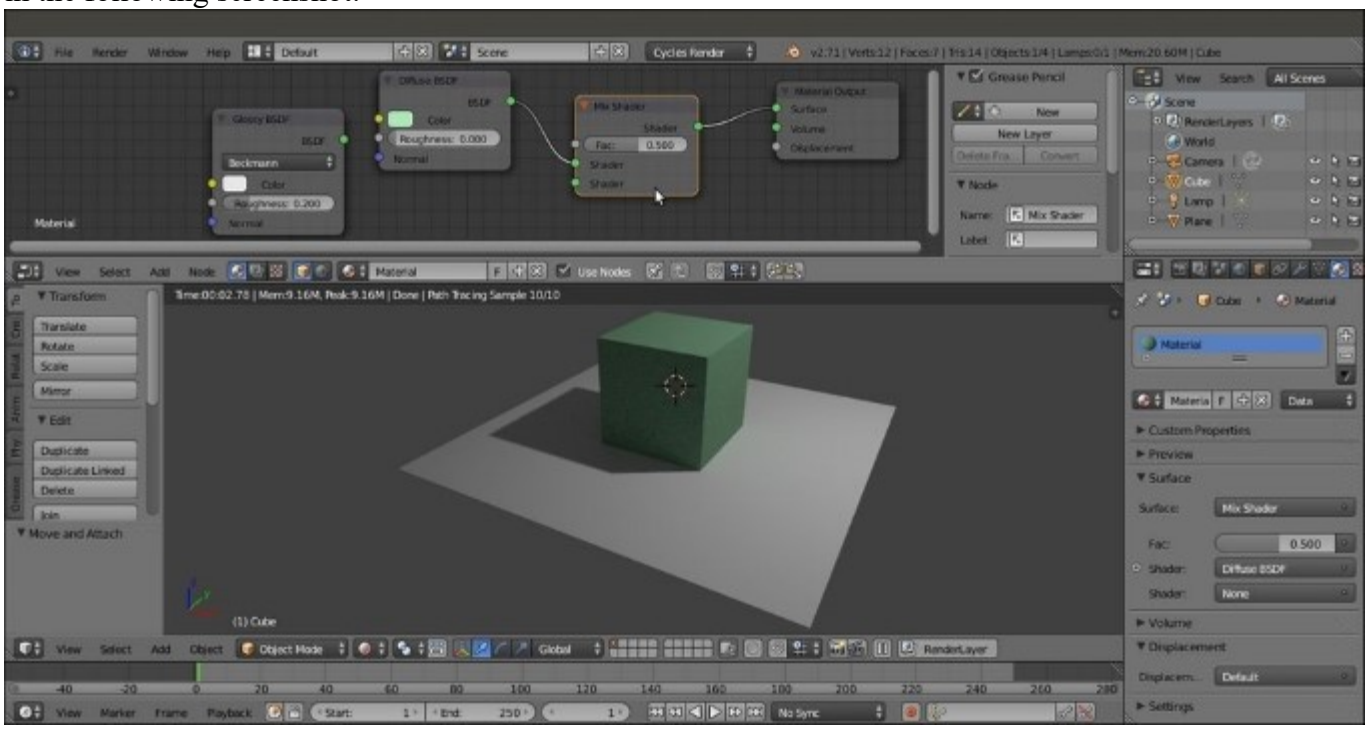
Real-time preview of the effect of the Glossy shader node and the Surface subpanel under the Material window

Note that although we just switched a shader node with a different node, the color we set in the former has been kept in the new one. Actually, this happens for all the values that can be kept from a node to a different one.

Now, because a material having a 100 percent matte or reflective surface could hardly exist in the real world, a more accurate basic Cycles material should be made by mixing the **Diffuse BSDF** and the **Glossy BSDF** shaders, blended together by a **Mix Shader** node, which in turn is connected to the **Material Output** node:

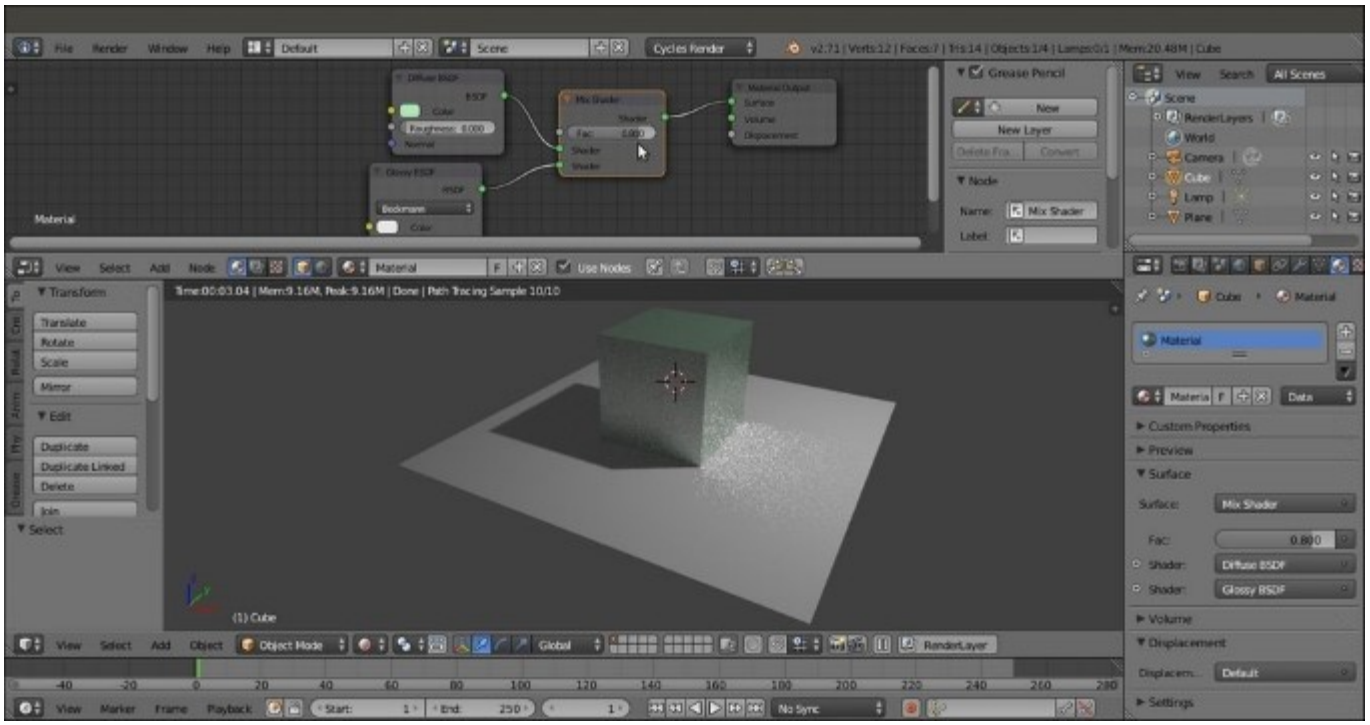
1. In the **Material** window, under the **Surface** tab, click again on the **Surface** button, which is now showing the **Glossy BSDF** item, and replace it with a **Diffuse BSDF** shader.
2. Put the mouse pointer on the **Node Editor** window, and by pressing *Shift* + *A*, make a pop-up menu appear with several items. Move the mouse pointer on the **Shader** item. It shows one more pop-up, where all the shader items are collected. Alternatively, press the *T* key to call the **Node Editor** tool shelf, where you can find the same shader items under the different tabs.
3. Select one of these items (in our case, the **Glossy BSDF** shader node again). The **Shader** node, which is already selected, is now added to the **Node Editor** window, although it is not connected to anything yet (in fact, it's not visible in the **Material** window but only in the **Node Editor** window).
4. Again press *Shift* + *A* in the **Node Editor** window, and this time, add a **Mix Shader** node.

5. Press *G* to move the node to the link connecting the **Diffuse BSDF** node to the **Surface** input socket of the **Material Output** node (you'll probably need to first adjust the position of the two nodes to make room between them). The **Mix Shader** node gets automatically pasted in between, and the **Diffuse** node output gets connected to the first **Shader** input socket, as shown in the following screenshot:



Mix Shader node pasted between a preexisting nodes connection inside the Node Editor window

6. Click on the green dot output of the **Glossy BSDF** shader node, and grab the link to the second input socket of the **Mix Shader** node. Release the mouse button now and see the nodes being connected.
7. Because the blending **Fac** (factor) value of the **Mix Shader** node is set by default to `0.500`, the two shader components, **Diffuse** and **Glossy**, are now showing on the Cube's surface in equal parts, that is, each component at 50 percent. Click on the **Fac** slider with the mouse and slide it to `0.000`. The Cube's surface now shows only the **Diffuse** component because the **Diffuse BSDF** shader is connected to the first **Shader** input socket, which is corresponding to a value of `0`.
8. Slide the **Fac** slider value to `1.000` and the surface now shows only the **Glossy BSDF** shader component, which is, in fact, connected to the second **Shader** input socket corresponding to a value of `1`.
9. Set the **Fac** value to `0.800` (keep *Ctrl* pressed while you are sliding the **Fac** value to constrain it to `0.100` intervals). The Cube is now reflecting the white Plane on its sides, even though it is blurred, because we have a material that is reflective at 80 percent and matte at 20 percent (the white noise you see in the rendered preview is due to the low sampling we are using at the moment. You will learn more about this later). This is shown in the following screenshot:



The Rendered preview of the effect of the mixed Diffuse and Glossy shader nodes

10. Lastly, select the Plane, go to the **Material** window, and click on the **New** button to assign a diffuse whitish material.

How it works...

In its minimal form, a Cycles material is made by any one of the node shaders connected to the **Surface** or the **Volume** input sockets of the **Material Output** node. For a new material, the node shader is **Diffuse BSDF** by default, with the **RGB** color set to `0.800` and connected to the **Surface** socket, and the result is a matte whitish material (with the **Roughness** value at `0.000`, actually corresponding to a **Lambert** shader).

Then the **Diffuse BSDF** node can be replaced by any other node of the available shader list, for example, by the **Glossy BSDF** shader as in the former Cube scene, which produced a totally mirrored surface material.

As we have seen, the **Node Editor** window is not the only way to build the materials. In the **Properties** panel on the right-hand side of the UI, we have access to the **Material** window, which is usually divided as follows:

- The material name, user, and the **datablock** subpanel.
- The **Preview** window.
- The **Surface** subpanel, including only the shader nodes added in a vertically ordered column in the **Node Editor** window, and already connected to each other.
- The **Volume** subpanel, with the similar feature as that of the **Surface** subpanel.

- The **Displacement** subpanel.
- The **Settings** subpanel, where we can set the object color, the alpha intensity, the specularity color, and the hardness as seen in the viewport in non-rendered mode (**Viewport Color, Alpha**, **Viewport Specular**, and **Hardness**). It also contains the **Pass Index** value of the material, a **Multiple Importance Sample** checkbox, the Volume sampling methods, the Interpolation, the Homogeneous item to be activated to accelerate the rendering of volumes, and an option to disable the rendering of the transparent shadows to accelerate the total rendering.

The **Material** window not only reflects what we do in the **Node Editor** window and changes accordingly (and vice versa), but can also be used to change the values to easily switch the shaders themselves, and to some extent, to connect them to the other nodes.

The **Material** and the **Node Editor** windows are so mutual that there is no prevalence in which window to use to build a material. Both can be used individually or combined, depending on preferences or practical utility. In some cases, it can be very handy to switch a shader from the **Surface** tab under **Material** on the right (or a texture from the **Texture** window as well, but we'll see textures later), leaving all the settings and the links in the node's network untouched.

There is no question, by the way, that the **Material** window can become pretty complex and confusing as a material network grows more and more in complexity, while the graphic appearance of the **Node Editor** window shows the same network in a clearer and much more readable way.

There's more...

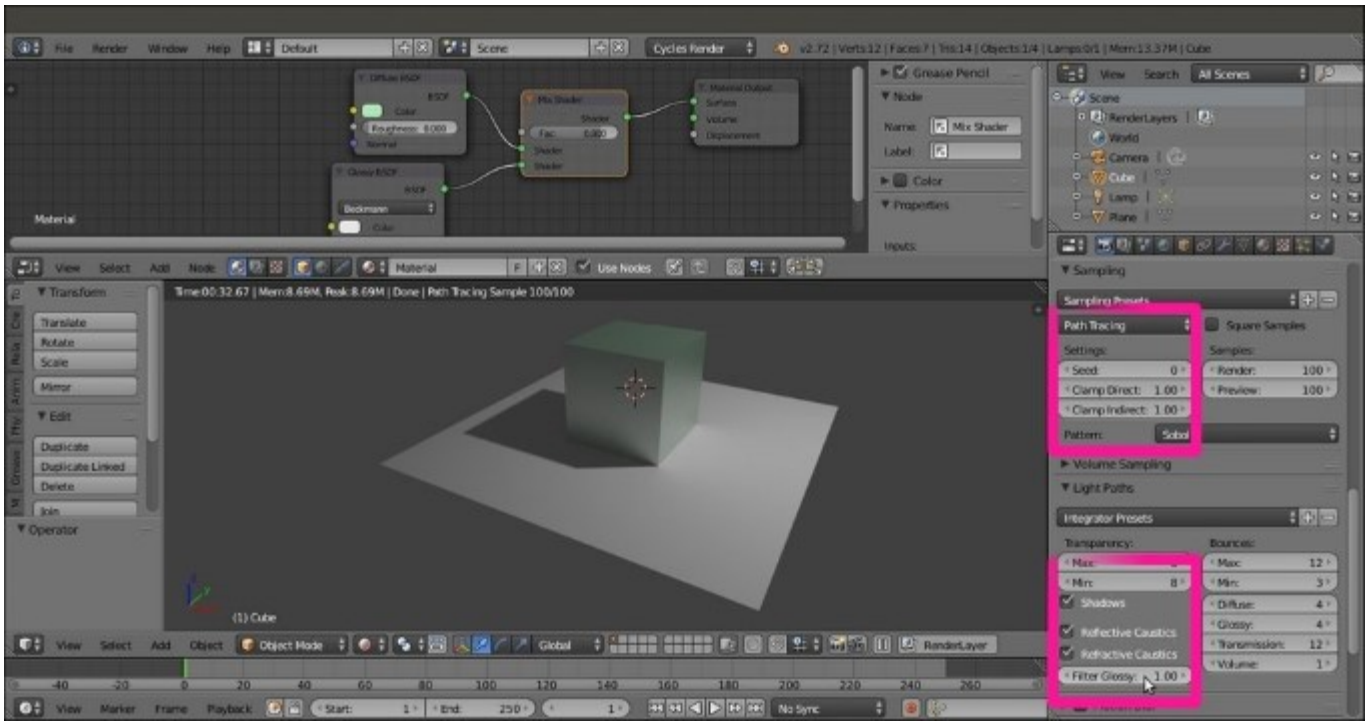
Looking at the **Rendered** viewport, you'll notice that the image is now quite noisy and that there are white dots in certain areas of the image. These are the infamous fireflies, caused mainly by transparent, luminescent, or glossy surfaces. Actually, they have been introduced in the rendering of our Cube by the glossy component.

Here is one way to eliminate the fireflies:

1. Go to the **Render** window under the **Properties** panel.
2. Uncheck both the **Reflective** and **Refractive Caustics** items under the **Light Path** subpanel.
3. This will immediately eliminate the white noise, but alas! It also eliminates all the caustics (which we would like to keep in the rendering in most cases).

Therefore, a different approach is as follows:

1. Go to the **Render** window under the **Properties** panel. In the **Sampling** tab, set **Samples** to `100` for both **Preview** and **Render** (they are set to `10` by default).
2. Set the **Clamp Direct** and **Clamp Indirect** values to `1.00` (they are set to `0.00` by default).
3. Go to the **Light Paths** tab, re-enable the **Reflective** and **Refractive Caustics** items, and then set the **Filter Glossy** value to `1.00`.
4. The resulting rendered image, as shown in the following screenshot, is now a lot smoother and noise-free, and also keeps the reflected caustics on the Plane:



Noise-free Rendered preview and settings under the Render window

5. Save the blend file in an appropriate location on your hard drive with a name such as `start_01.blend`.
6. The **Samples** set to `10` by default are obviously not enough to give a noiseless image, but are good for a fast preview. We could also let the **Preview** samples remain at the default value and increase only the **Render** value, to have longer rendering times but a clean image only for the final render (which can be started, as in Blender Internal, by pressing the *F12* key).

Using the **Clamp** value, we can reduce the energy of the light. Internally, Blender converts the image color space to linear, which is from `0` to `1`, and then reconverts it to **RGB**, which is from `0` to `255`, for the output. A value of `1.00` in linear space means that all the image values are now included inside a range starting from `0` and arriving to a maximum value of `1`, and that values greater than `1` are not possible, thus avoiding the fireflies problem in most cases. Be aware that **Clamp** values higher than `1.00` might also lower the general lighting intensity of the scene.

The **Filter Glossy** value is exactly what the name says, a filter that blurs the glossy reflections on the surface to reduce noise.

Remember that even with the same samples, the **Rendered** preview does not always have a total correspondence to the final render with regards to both noise and the fireflies. This is mainly due to the fact that the preview-rendered 3D window and the final rendered image usually have very different sizes, and artifacts visible in the final rendered image may not show in a smaller preview-rendered window.

See also

As you have seen, the several nodes that can be used to build Cycles shaders have both input and output sockets to the left and to the right of the node interface, respectively, and the color of these sockets is actually indicative of their purpose; green sockets are for shaders, yellow sockets are for colors, gray sockets for values, and blue sockets for vectors.

Each color output socket of one node should be connected with the same color input socket of another node. By the way, connecting differently colored sockets also works quite often; for example, a yellow color output can be connected to a gray value input socket and to a blue vector input.

A general overview of all the Cycles nodes can be found at http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Nodes.

Procedural textures in Cycles

In this recipe, we'll see several kinds of textures available in Cycles, and learn how to use them with the shaders.

Similar to Blender Internal, we can use both procedural textures and image textures in Cycles. However, the Cycles procedural textures are not exactly the same as in Blender Internal. Some textures are missing because they have been replaced by an improved version (for example, the **Clouds** procedural texture has been replaced by particular settings of the **Noise** procedural texture), and a few textures are new and exclusive to Cycles.

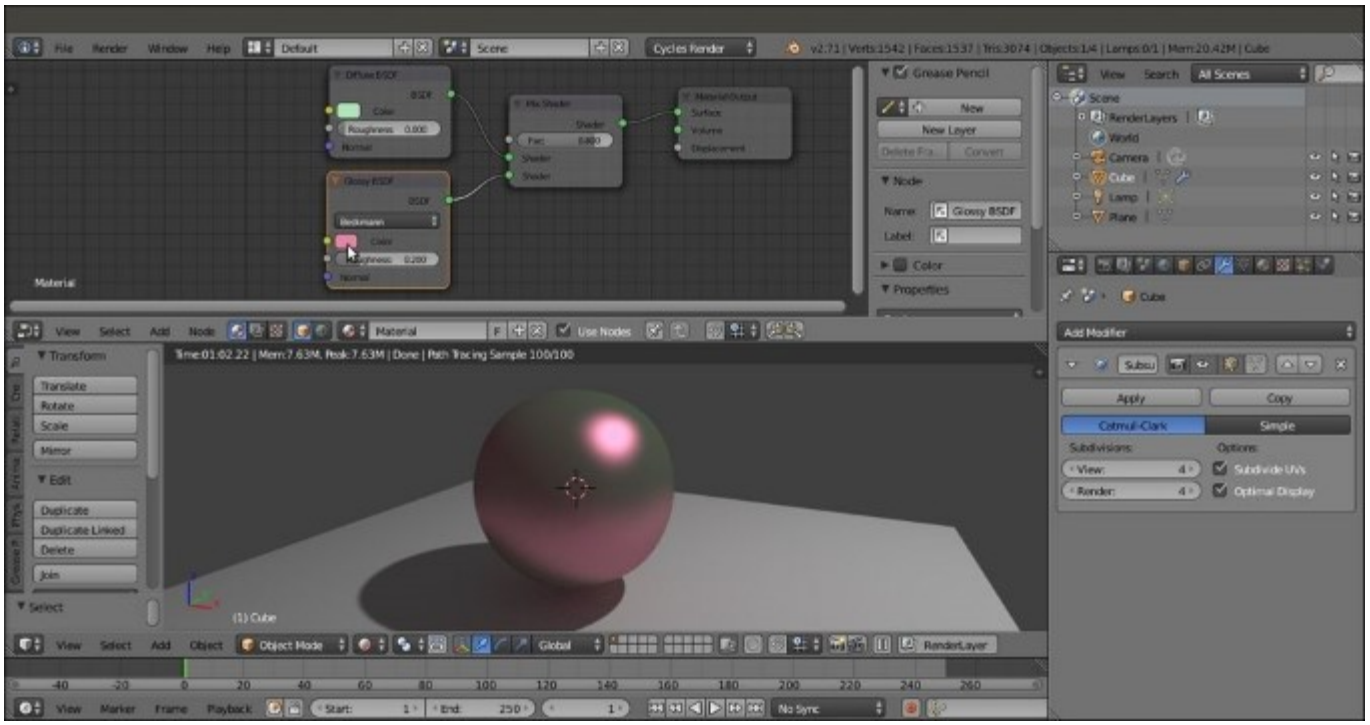
Getting ready

We have already seen a simple construction of a basic Cycles material by mixing the diffuse and the glossy (specular) components of a surface. Now let's take a look at the textures we can use in Cycles to further refine a material.

Because Cycles has a node-based system for materials, textures are not added in their slot under a tab as they are in Blender Internal. They get added in the **Node Editor** window, and are directly connected to the input socket of the shaders or other kinds of nodes. This gives a lot more flexibility to the material creation process because a texture can be used to drive several options inside the material network.

Let's see how they work:

1. Starting from the previously saved `start_01.blend` blend file, where we already set a simple scene with a Cube on a Plane and a basic material, select the Cube and go to the **Object modifiers** window inside the **Properties** panel to the right of the UI.
2. Assign to the Cube a **Subdivision Surface** modifier, set the **Subdivisions** level to `4` for both **View** and **Render**, and check the **Optimal Display** item.
3. Go to the **Tool** tab at the left of the 3D window, navigate to **Edit** | **Shading**, and set the subdivided Cube (let's call it Spheroid from now on) to **Smooth**.
4. Just to make things clearer, click on the color box of the **Glossy BSDF** shader to change it to a purple color (RGB set to `0.800`, `0.233`, and `0.388`, respectively). Note that only the glossy reflection part on the Spheroid is now purple, whereas the rest of the surface, which is the diffuse component, is still greenish.
5. Save the blend file and name it `start_02.blend`. The effect visible in the real-time **Rendered** preview is as follows:

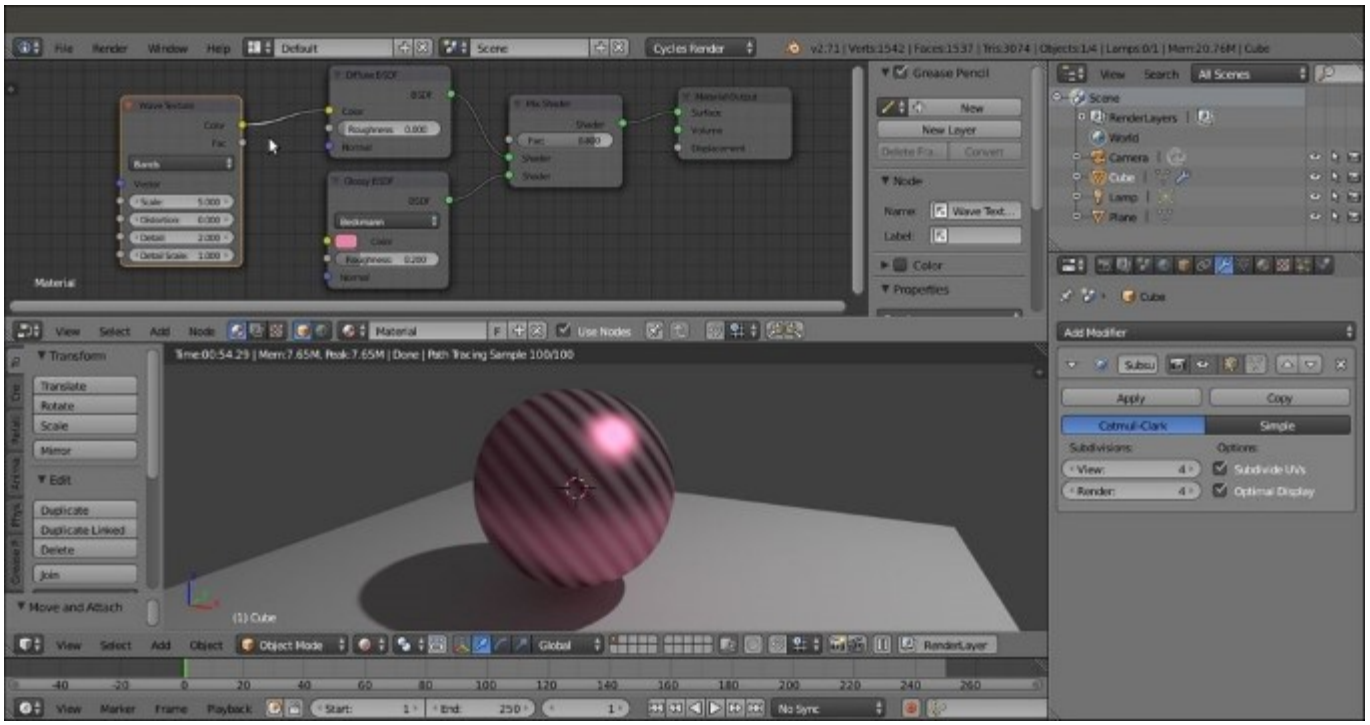


The Rendered preview of the effect of two differently colored Diffuse and Glossy components on the Spheroid

How to do it...

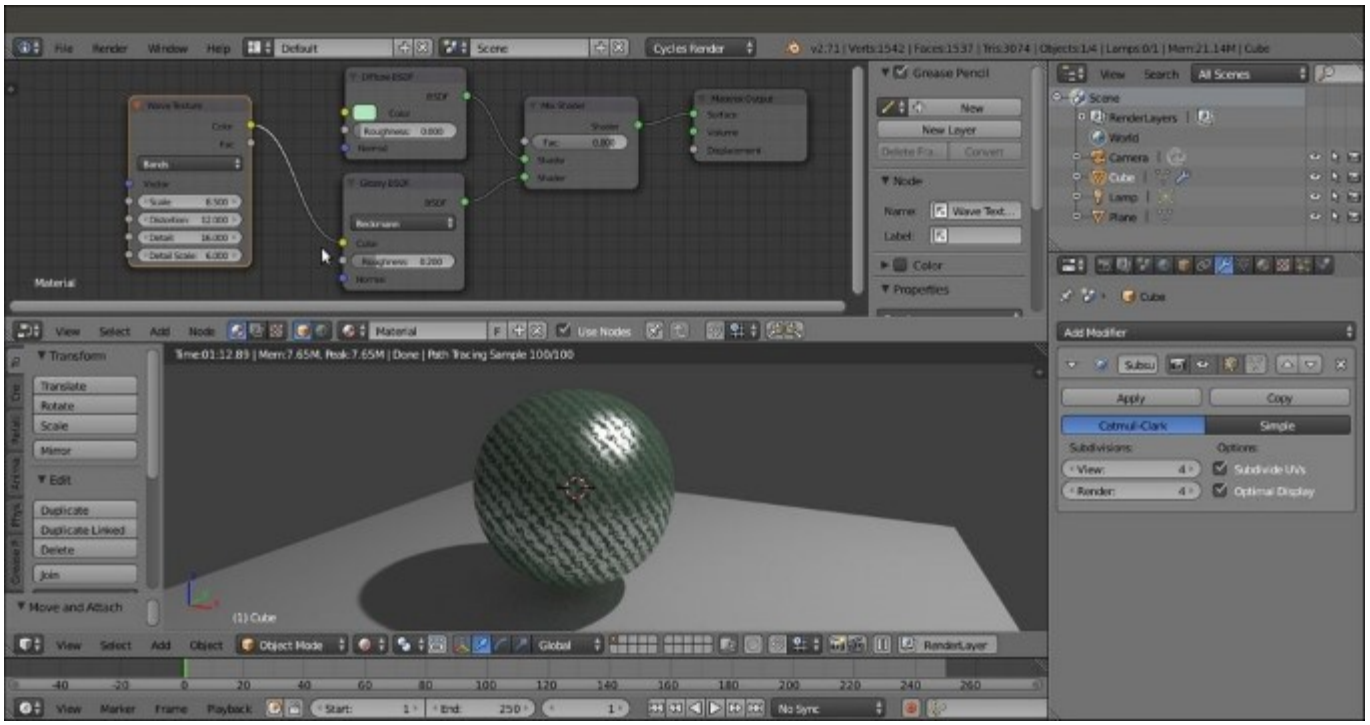
Perform the following steps to add a procedural texture to the object:

1. Put the mouse pointer in the **Node Editor** window and press *Shift* + *A*.
2. In the contextual pop-up menu, go to the **Texture** item, just under **Shader**, and click on **Wave Texture** to add the texture node to the **Node Editor** window.
3. Grab and connect the yellow **Color** output socket of the texture to the yellow input socket of the **Diffuse** shader, the socket close to the **Color** rectangle that we formerly set as a greenish color, as shown in this screenshot:



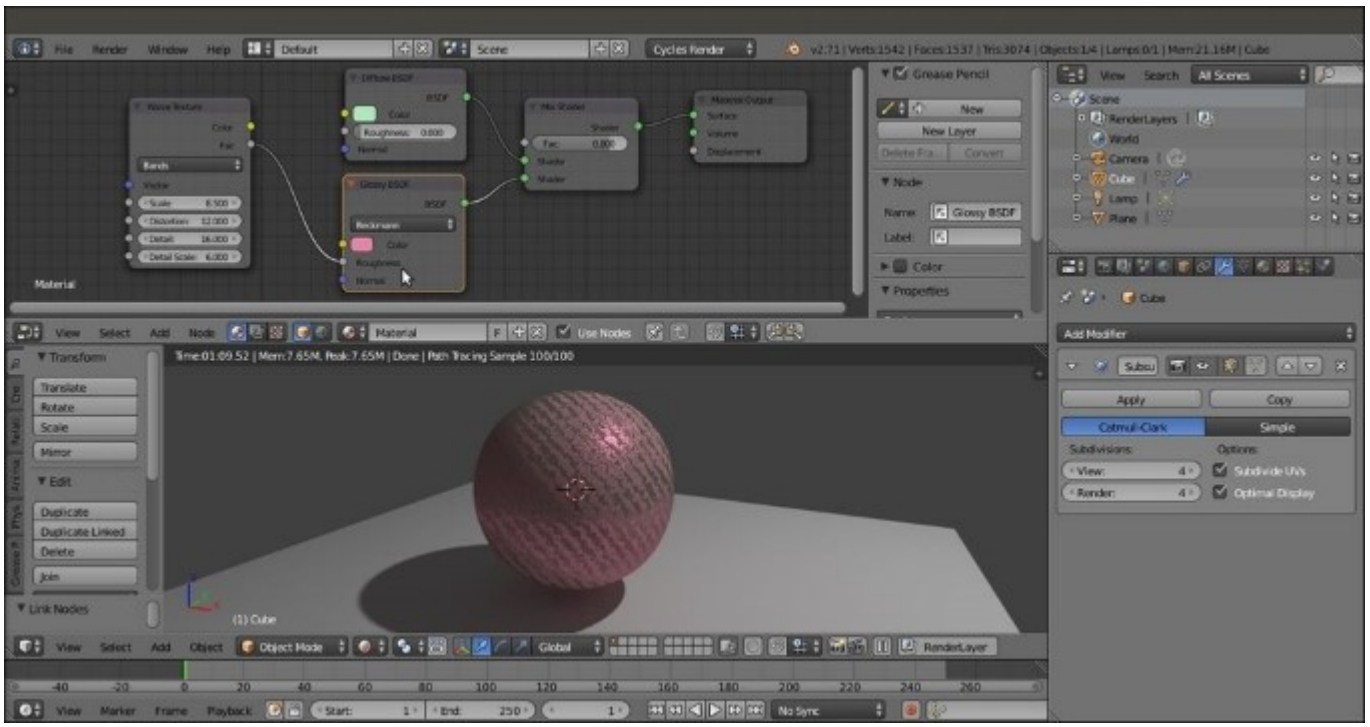
The Rendered preview of the effect of a Wave texture assigned as color to the diffuse component of the material

4. In the **Wave Texture** node, change the **Scale** value to `8.500`, **Distortion** to `12.000`, **Detail** to a maximum value of `16.000`, and the **Detail Scale** value to `6.000`.
5. Now disconnect the texture color output from the **Diffuse** node and connect it to the color input socket of the **Glossy** shader, as shown in the following screenshot:



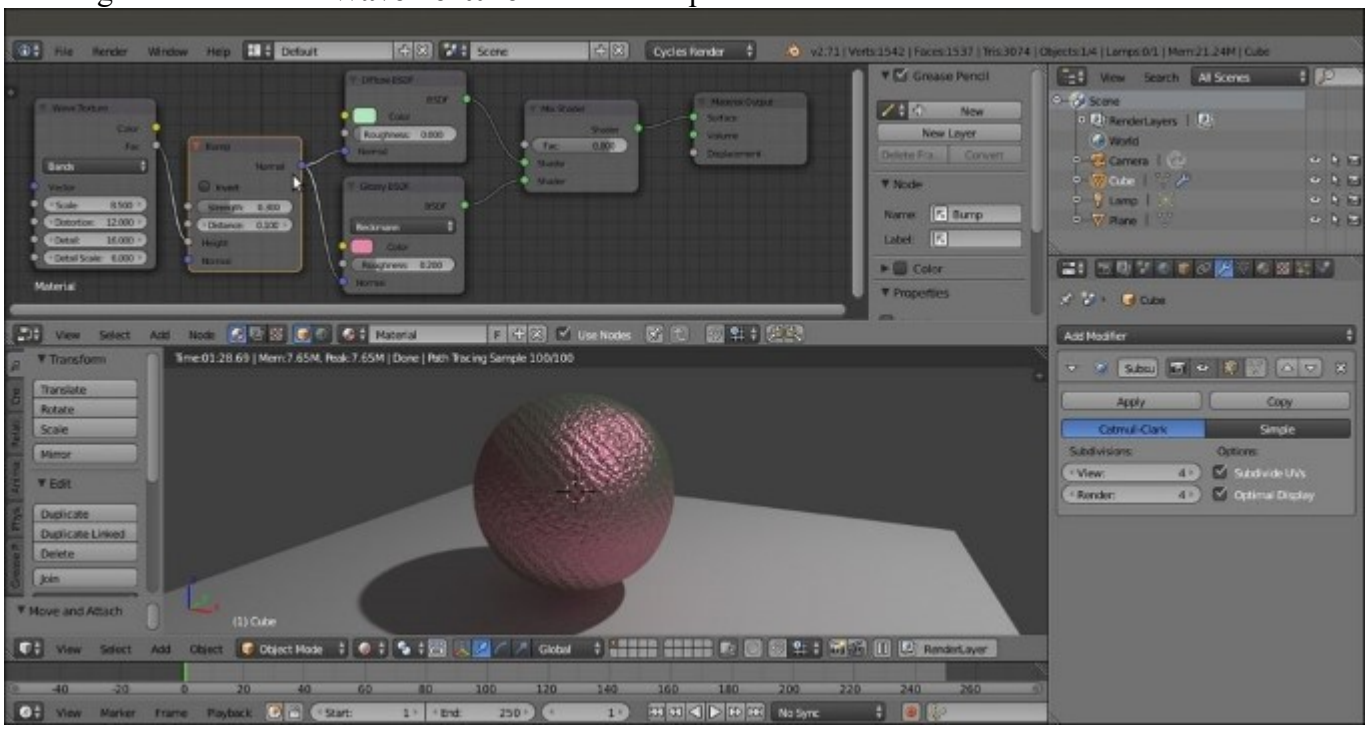
The effect of the Wave Texture assigned as color to the Glossy component of the material

6. Disconnect the texture color output from the **Glossy** shader. Grab and connect the texture node's **Fac** output to the **Roughness** input socket of the **Glossy BSDF** shader, as shown in this screenshot:



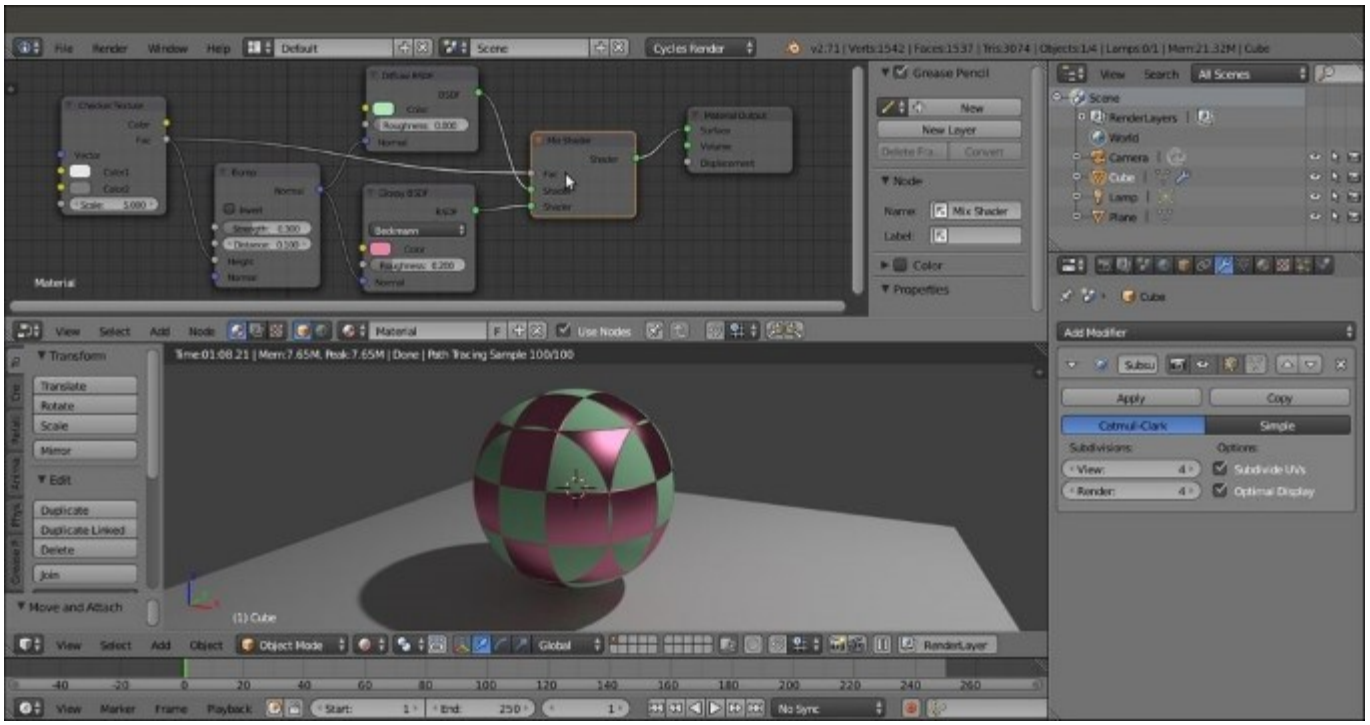
The effect of the Wave Texture assigned as Roughness factor to the Glossy component of the material

7. Disconnect the texture color output from the **Roughness** input socket of the **Glossy BSDF** shader. Move the **Wave Texture** node to the left and add a **Bump** node (*Shift* + *A* and navigate to **Vector** | **Bump**). Connect the **Fac** output of the **Wave Texture** node to the **Height** input node of the **Bump** node, and the **Normal** output of the **Bump** node to the **Normal** input socket of both the **Diffuse** and the **Glossy** nodes. Set the **Strength** to `0.300`. Here is a screenshot showing the effect of the **Wave Texture** node as bump:



The effect of the Wave Texture Fac output as Bump for both the components of the material

8. Save the file.
9. Delete the **Wave Texture** node (*X* key), press *Shift* + *A* with the mouse pointer in the **Node Editor** window, and add a **Checker Texture** node.
10. Connect the **Fac** output of the **Checker Texture** node to the **Fac** input socket of the **Mix Shader** node and to the **Height** input socket of the **Bump** node, as shown in the following screenshot:



The effect of a Checker Texture used as bump and especially as blending factor to mix the two components of the material

11. Save the file as `start_03.blend`.

How it works...

From step 1 to 3, the changes are immediately visible in the **Rendered** viewport. At the moment, the **Wave Texture** node color output is connected to the color input of the **Diffuse BSDF** shader node, and the Spheroid looks as if it's painted in a series of black and white bands. Actually, the black and white bands of the texture node override the green color of the diffuse component of the shader, while keeping the material's pink glossy component unaltered.

In step 5, we did exactly the opposite. We disconnected the texture output from the **Diffuse** shader to connect it to the **Glossy** shader color input. Now we have the diffuse greenish color back and the pink has been overridden, while the reflection component is visible only inside the white bands of the wave texture.

In step 6, in addition to the color output, every texture node also has a **Fac** (factor) output socket, outputting gray-scale linear values. When connected to the **Roughness** input socket of the **Glossy** shader, the texture output works as a factor for its reflectivity. The Spheroid keeps its colors and gets the specular component only in the white areas on the surface (that is, white bands represent total reflection and black bands represent no reflection).

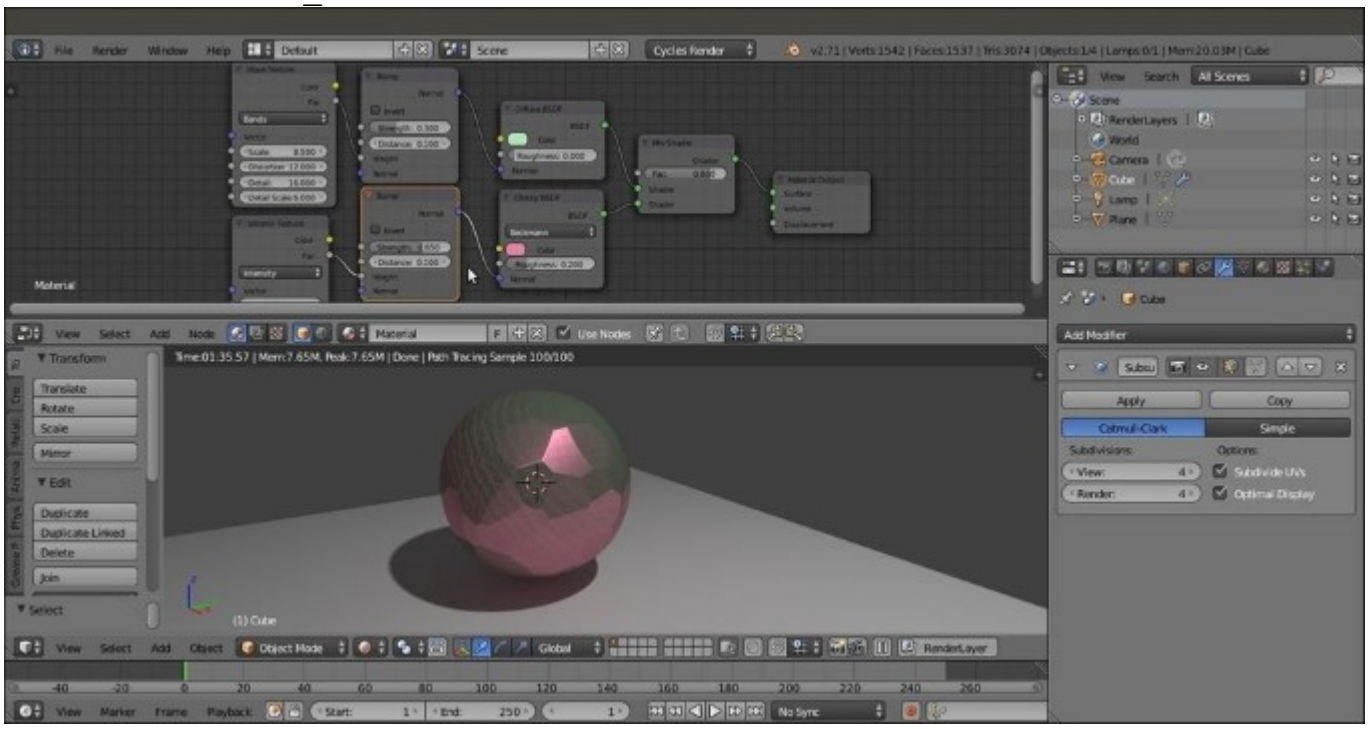
In step 10, the **Checker Texture** node's **Fac** output connected to the **Fac** input socket of the **Mix Shader** node works as a mask, or a stencil, based on the black and white values of the output. The numeric slider for the mixing factor on the **Mix Shader** node has disappeared because now we are using the black and white linear values of the **Checker Texture** output as a factor to mixing the **Diffuse** and **Glossy** components. Therefore, these components appear on the Spheroid surface according to the black and white quads of the checker.

Every texture node has several setting options. All of them have in common the **Scale** value to set the size of the procedural. The other settings change according to the type of texture.

The **Fac** output of the texture node can be used to feed the **Height** input socket of the **Bump** node (actually, the **Color** output also works quite well here). Hence, the **Normal** output of the **Bump** node can be connected to the **Normal** input sockets of each shader node, giving a per node bump effect. So, the bump can have an effect only on the diffuse component, or only on the glossy component, or on both, and so on.

Let's create an example of **Wave** and **Voronoi** textures:

1. Re-open the `start_02.blend` file.
2. Add a **Voronoi Texture** node (press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**) and a new **Bump** node (press *Shift* + *A* and navigate to **Vector** | **Bump**).
3. Connect the **Fac** output of the **Voronoi Texture** node to the **Height** socket of the new **Bump** node, and connect the latter to the **Normal** input socket of the **Glossy BSDF** shader node. Set its **Strength** value to `0.650` and the **Voronoi** scale to `6.000`.
4. Save the file as `start_02bis.blend`.



Two different procedural textures, a Wave and a Voronoi, used as bumps for the two components to have a per shader effect

In this case, we have two different bump types, affecting the diffuse and the glossy components independently, and building an effect of a layered bump.

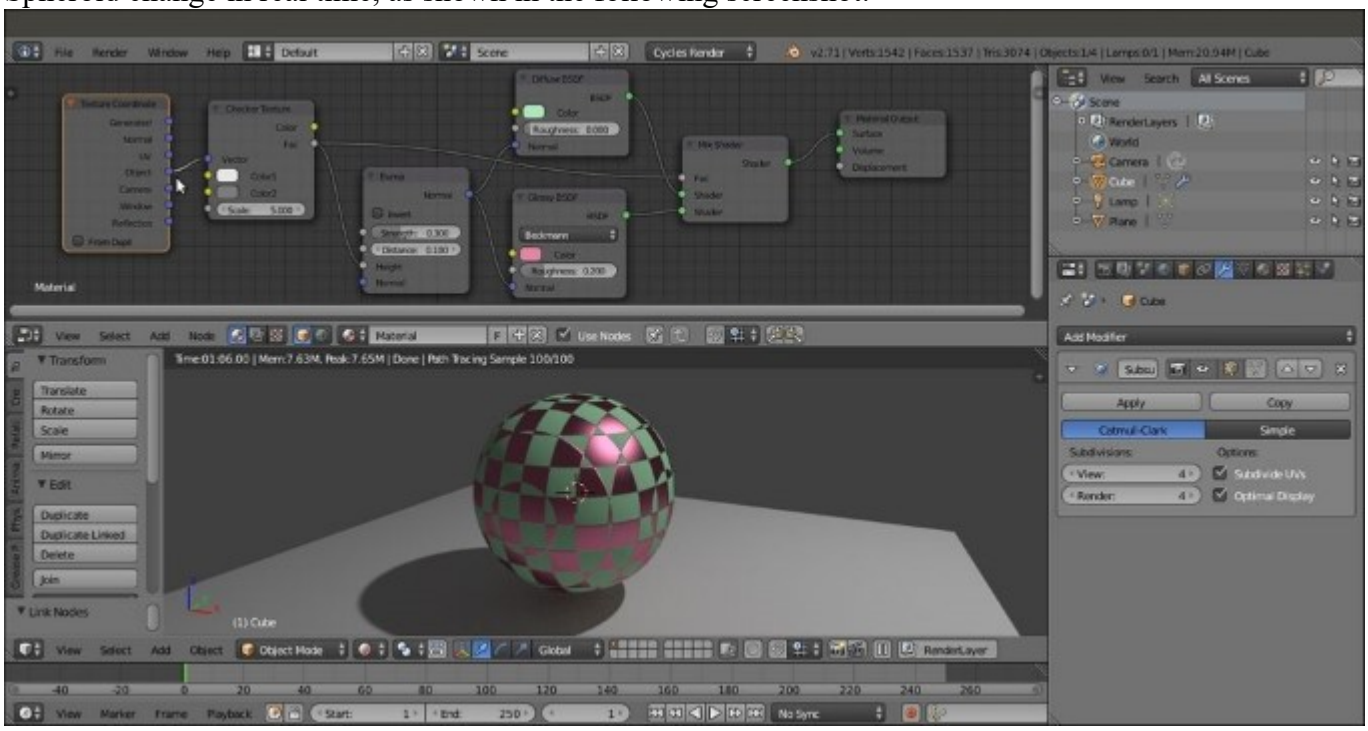
There's more...

At this point, you could wonder: "Okay, we just mapped textures on the Spheroid, but what's the projection mode of these mappings?"

Good question! By default, if the projection mode is not specified and if the object doesn't have any UV coordinates, the mapping is **Generated**, which is the equivalent of the **Original Coordinates** mode (now renamed **Generated** as well) in Blender Internal.

But what if you want to specify a mapping method? Then follow these steps:

1. Press *Shift* + *A* with the mouse pointer in the **Node Editor** window again, go to the **Input** item, and select the **Texture Coordinate** item, which is a node with several mapping modes and their respective output sockets.
2. Try to connect the several outputs to the **Vector** input (the blue socket on the left-hand side of the node), which can be found from **Checker Texture**, to see the texture mapping on the Spheroid change in real time, as shown in the following screenshot:



The Object output of the Texture Coordinate node connected to the Vector input of the Texture node

By the way, I'd like to point your attention to the UV coordinates output. Connect the link to the texture's vector socket, and you will see the mapping on the Spheroid disappear. Why is this so? Because we haven't assigned any UV coordinates to our Spheroid yet.

Go to the **UV Maps** tab in the **Object data** window, under the **Properties** panel on the right, and click on the **+** sign. This just adds a one-to-one **Reset UV projection UV** layer to the object, which means that every face of the mesh is covering the whole area of the **UV/Image Editor** window. Remember that although the Cube looks like a Spheroid now, this is only due to the effect of the assigned **Subdivision Surface** modifier. The UV coordinates work at the lowest level of subdivision, which is still a six-faced Cube.

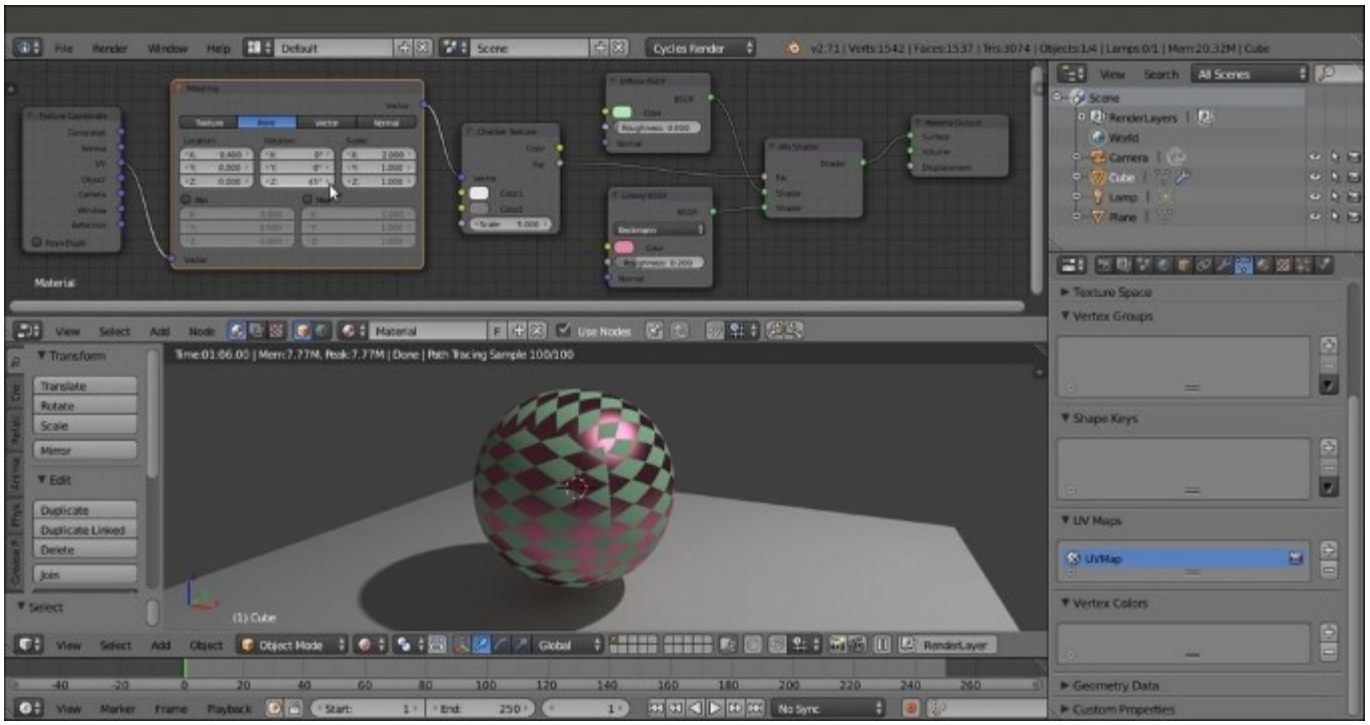
A second option is to place the proper seams on the Cube's edges and directly unwrap the object in the **UV/Image Editor** window, as demonstrated in the following steps:

1. Press *Tab* to go to **Edit Mode**, select the appropriate edges, press *Ctrl* + *E*, and in the **Edges** pop-up menu, select the **Mark Seam** item.
2. Now press *A* to select all the vertices (if deselected), press *U*, and choose an unwrapping method from the **UV Mapping** pop-up menu (**Smart UV Project** and **Cube Projection** don't even need the seams). Then go out of **Edit Mode** to update the **Rendered** preview.

The **Texture Coordinate** node is not mandatory to map an image texture on an unwrapped object; in such a case, Cycles will automatically use the (first) available UV coordinates to map the image map anyway.

Often, the only **Texture Coordinate** node is not enough. What we need now is a way to offset, rotate, and scale this texture on the surface:

1. First delete the **Bump** node, then select the **Texture Coordinate** node, and drag it to the left of the window as far as suffices to make room for a new node. In the **Add** menu, go to **Vector** and choose **Mapping**.
2. Grab the **Mapping** node in the middle of the link that connects the **Texture Coordinate** node to the **Checker Texture** node. It will be automatically pasted between them, as shown in the following screenshot:



The Mapping node pasted between Texture Coordinate and the Texture nodes

3. Now start playing with the values inside the **Mapping** node. For example, set the **Z Rotation** value to `45°`, set the **X Scale** value to `2.000`, and then slide the **X Location** value, while seeing, in the **Rendered** viewport, how the texture changes orientation and dimension and actually slide along the *x* axis.
4. Save the blend file as `start_04.blend`.

The **Min** and **Max** buttons on the bottom of the **Mapping** node are used to clip the extension of the texture mapping. Check both **Min** and **Max** to prevent the texture from being repeated *n* times on the surface, and it will be shown only once. A minimum value of `0.000` and a maximum value of `1.000` give a correspondence of one-to-one to the mapped image. You can tweak these values to limit or extend the clipping. This is useful to map decals, logos, or labels, for example, on an object and avoid repetition.

See also

In Cycles, it is possible to use normal maps by adding the **Normal Map** node (by navigating to **Add** | **Vector** | **Normal Map**) and connecting its output to the **Normal** input socket of the shader nodes.

To see an example of a **Normal Map** node used in a Cycles material, go to Chapter 8, *Creating Organic Materials*, of this cookbook and look at the `bark_seamless` material of the *Creating trees shaders – the bark* recipe.

Here is a link to the official documentation talking about the **Normal Map** node:

http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Nodes/More

Setting the World material

In this recipe, we'll see the properties and the settings of the World in Cycles.

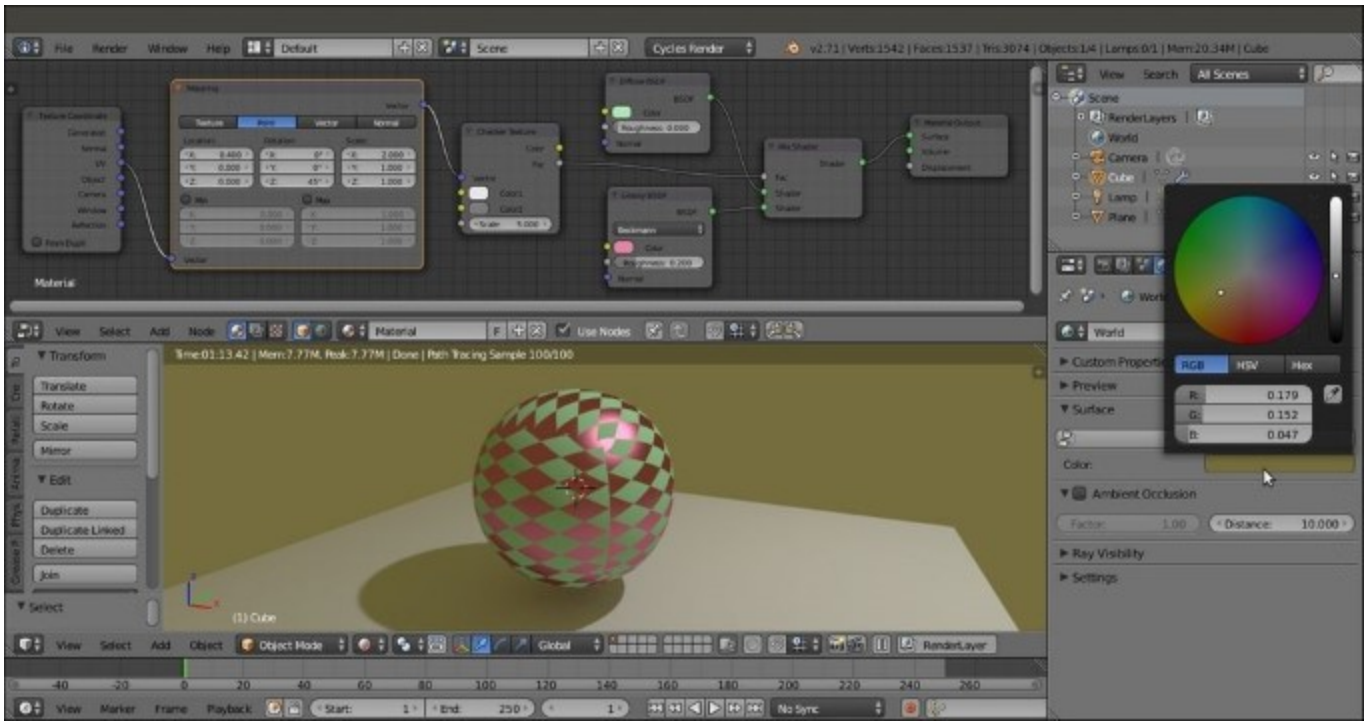
The main characteristic of the Cycles World is that it can emit light, so it practically behaves as a light source. Actually, its effect is the famous Global Illumination effect.

As in Blender Internal, the World is considered as a virtual dome at a large distance from the camera, never touching the scene's objects. Nothing in the 3D scene can affect the World. Actually, only the World can emit light on the scene and the objects.

Getting ready

1. Open the `start_04.blend` file and go to the **World** window under the **Properties** panel to the right of the screen. This is where we see the usual **Use Nodes** button under the **Surface** tab.
2. Although no node system for the **World** window is set by default, the **World** window already has a dark, medium gray color slightly lighting the scene. Delete the default Lamp or put it in a different and disabled layer to see that the Spheroid in the scene is dark but still visible in the rendered 3D viewport.
3. It's already possible to change this gray color to some other color by clicking on the **Color** button right under **Use Nodes** (the color at the horizon). This brings up the same color wheel that we saw for the shader colors. Set the color to `R 0.179`, `G 0.152`, and `B 0.047`, and save the file as `start_05.blend`.

Note that both the intensity and the general color graduation of the World are driven by this color. To have more light, just move the **Value** slider (the vertical slider) to a whiter hue. To give a general color mood to the scene, pick a color from inside the wheel. This will affect all of the scene's illumination but will show the effect mainly in the shadows, as shown in the following screenshot:

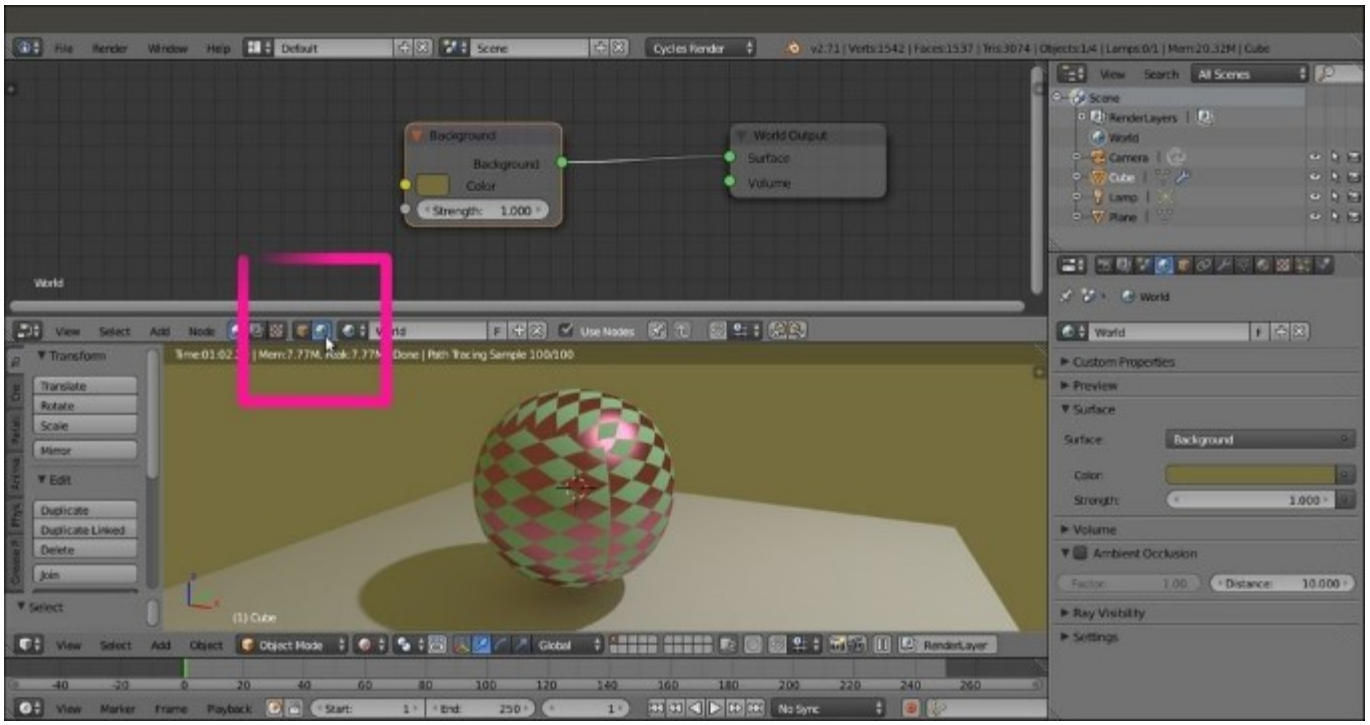


To the right is the color wheel to set the World's color, inside the World window, under the main Properties panel

How to do it...

However, to get access to all the options for the World, we have to initialize it as a node system, which is shown in the following steps:

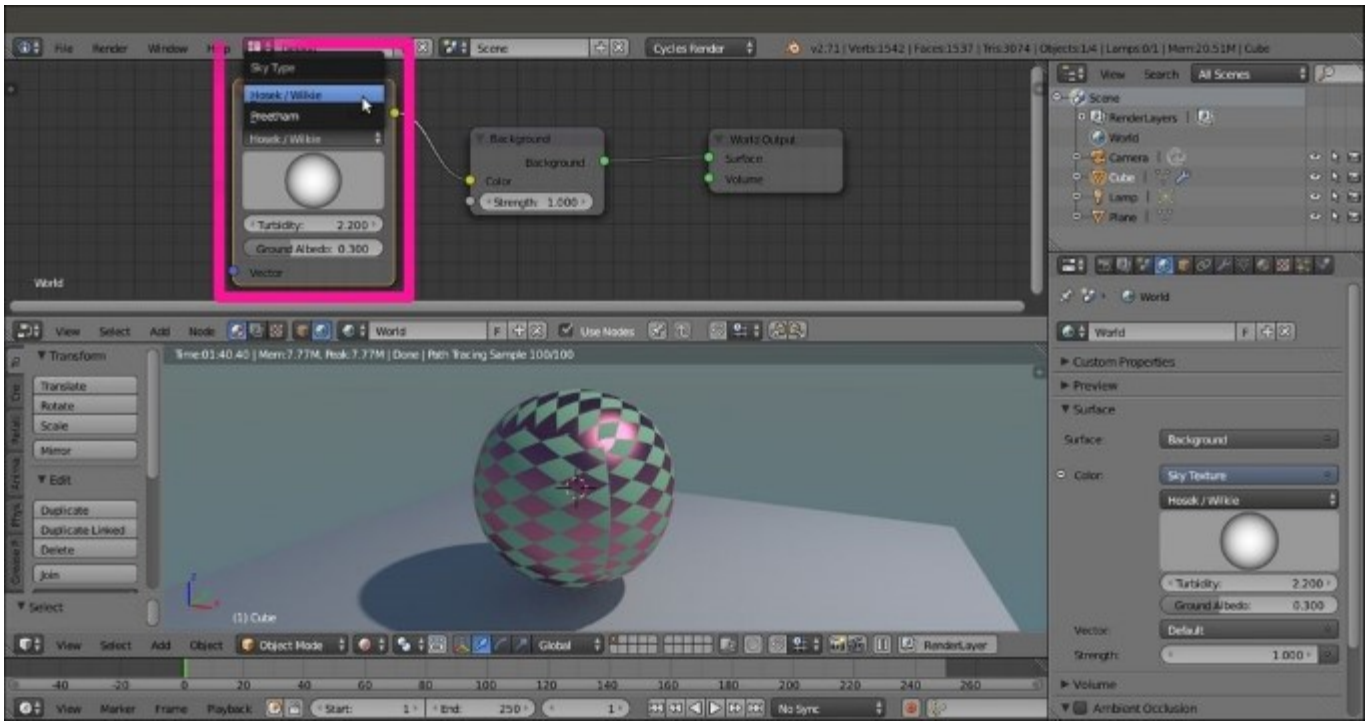
1. Look at the bottom header of the **Node Editor** window. On the left-hand side of the material data block, there are two little icons: a little cube and a little world. The cube icon is used to create materials, while the world icon is for the World. At the moment, because we were working on the Spheroid material, the cube icon is the one selected.
2. Click on the little world icon. The material's node disappears, and the **Node Editor** window is empty now because we entered the World mode. Check the little **Use Nodes** box on the right of the data block to make a default world material appear. Alternatively, go to the **World** window under the **Properties** panel and click on the **Use Nodes** button under the **Surface** tab. This is shown in the following screenshot:



The World button to be switched in the Node Editor toolbar

Just like the materials, the default material for the World is simply made up of two nodes. A **Background** node is connected to a **World Output** node. In the **Background** node, there are two setting options: the **Color** box and the **Strength** slider. Both of them are quite self-explanatory. Now, perform the following steps:

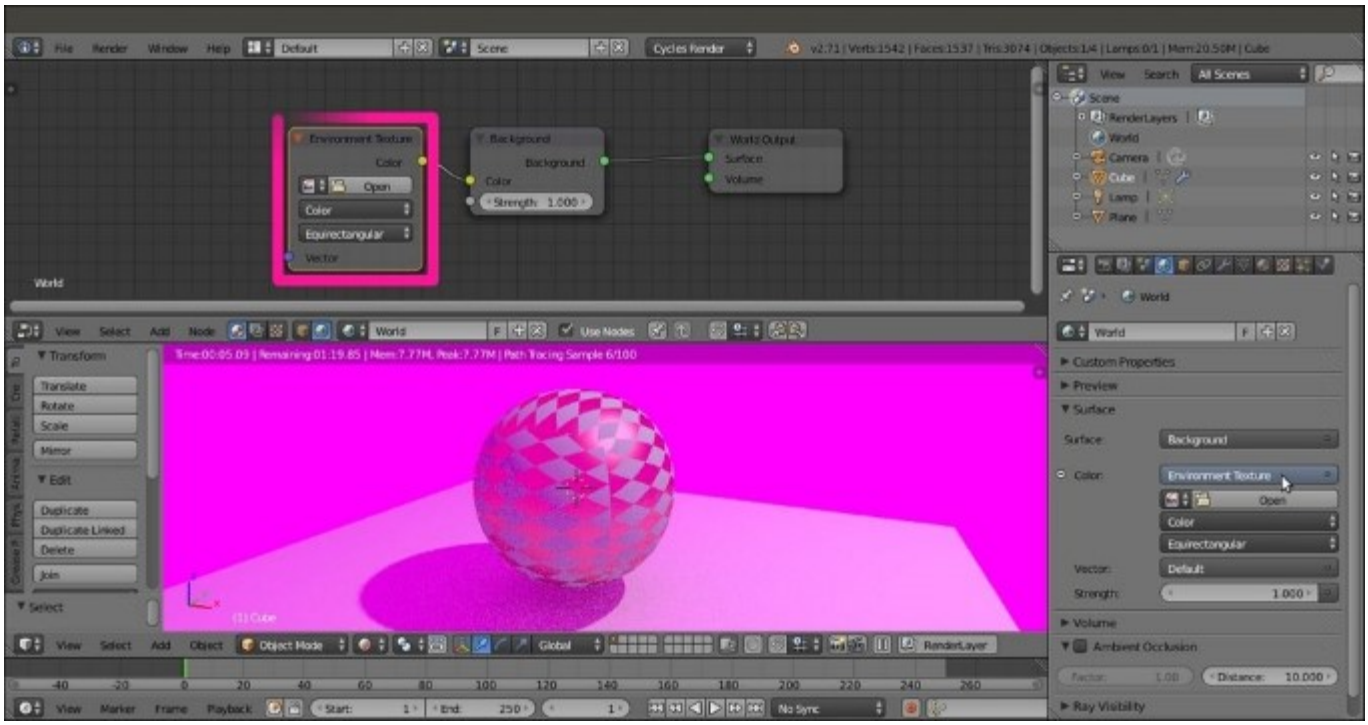
1. Go to the **World** window under the **Properties** panel, and click on the little square with a dot to the right side of the **Color** slot.
2. From the resulting menu, select the **Sky Texture** node item. This replicates a physical sky model with two **Sky** types, an atmospheric **Turbidity** value slider, a **Ground Albedo** value slider, and a **Strength** slider, as shown in this screenshot:



The Sky Texture node with options connected as Color to the Background node

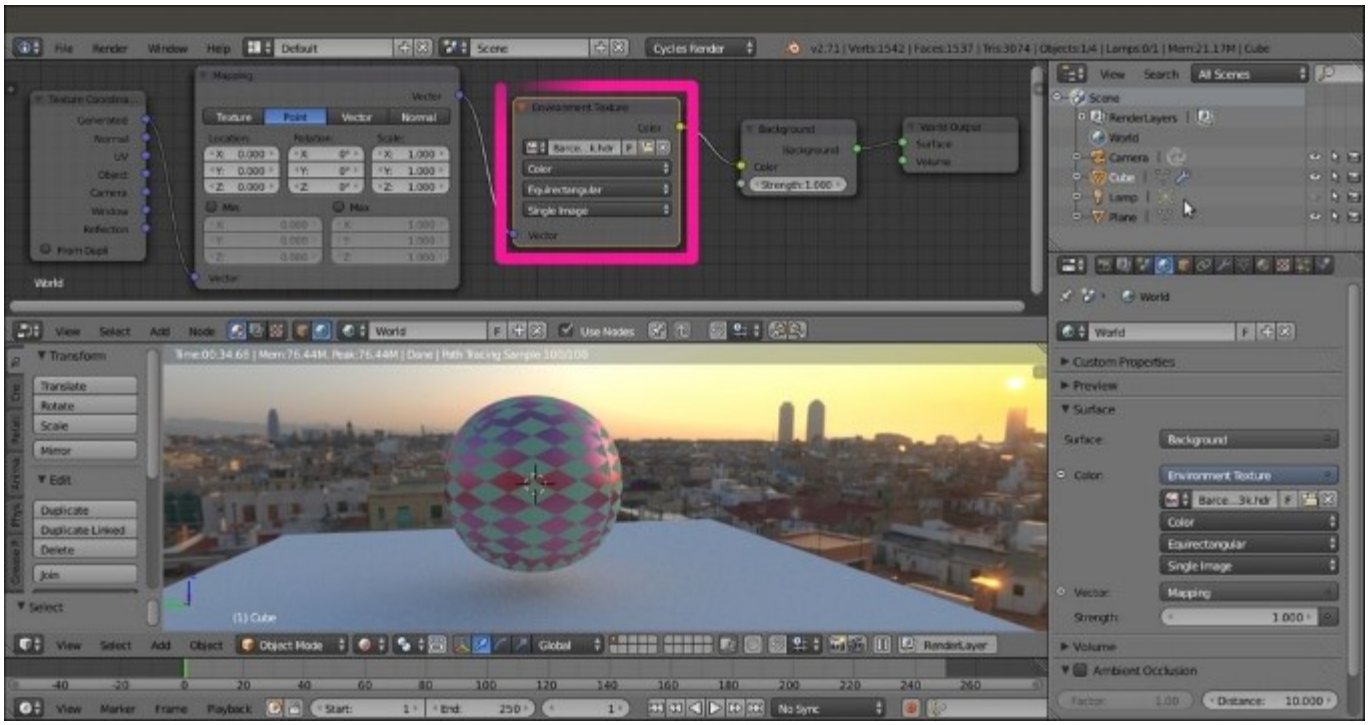
Note that you can also modify the incoming direction of the light, that is, the location of the sun, by rotating the sphere icon inside the node interface. This control isn't that much precise, by the way, and will hopefully improve in the future. The next steps are as follows:

1. Save the file as `start_06.blend`.
2. Click on the **Color** button, which is now labeled **Sky Texture**, under the **Surface** tab in the **Properties** panel, and select the **Environment Texture** node to replace it, as shown in the following screenshot:



The pink warning effect of a missing texture in the Environment Texture node of the World setting

3. Look in the **Rendered** view. You'll see that the general lighting has changed to a pink color. This is to show that the World material is now using an image texture to light the scene, but that there is no texture yet.
4. Click on the **Open** button in the **World** window, either under the **Properties** panel or in the recently added node inside the **Node Editor** window. Browse to the `textures` folder and load the `Barce_Rooftop_C_3k.hdr` image (a free, **High-dynamic-range** (**HDR**) image licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License from the sIBL Archive, at http://www.hdrlabs.com/sibl/archive.html).
5. To appreciate the effect, click on the little eye icon on the side of the **Lamp** item in the **Outliner** to disable its lighting. The Spheroid is now exclusively lit by the HDR image assigned to the World material. Actually, you can see the image as a background in the **Rendered** preview. You can also rotate the viewport and watch the background texture, pinned to the World coordinates, rotate accordingly in real time.
6. As for the object's materials, the mapping of any texture you are going to use for the World can be driven by the usual **Mapping** and **Texture Coordinates** nodes we have already seen. Generally, for the World materials, only the **Generated** coordinates output should be used, and actually, the **Generated** coordinates output is used by default if no mapping method is specified. Add the **Mapping** and **Texture Coordinates** nodes and connect them to the **Vector** input socket of the **Environment Texture** node, as shown in the following screenshot:

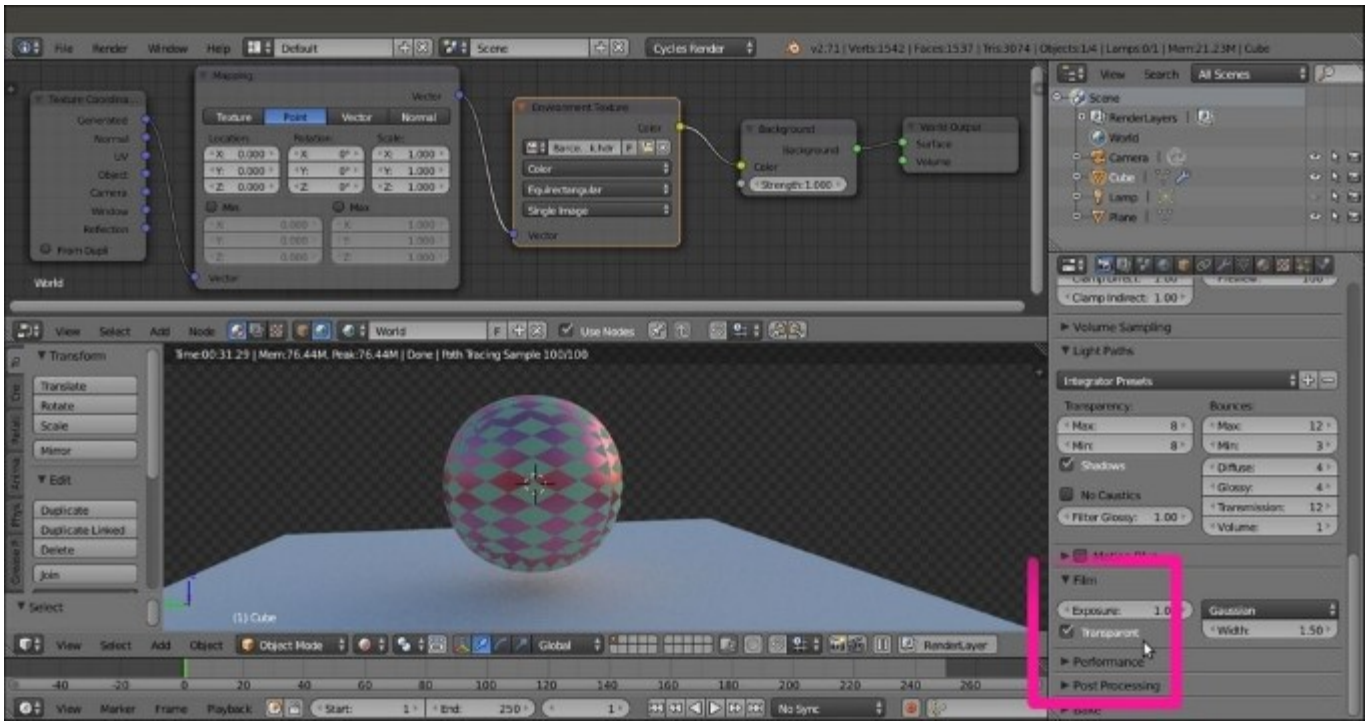


The Rendered preview of an HDR image assigned as a background to the World through the Environment Texture node

7. Save the file as `start_07.blend`.

Now let's imagine a case in which we want to assign a texture to the World material and use it for the general lighting of the scene, but we don't want it to show in the background of the render. In other words, we are using the HDR image to light the Spheroid and the Plane, but we want the two objects rendered on a uniform blue background; so how do we do it? This is how:

1. One way is to go to the **Render** window and check the **Transparent** option under the **Film** tab. This will show our Spheroid and Plane rendered in both the 3D viewport and the effective final rendered image on a transparent background, with a premultiplied alpha channel, as shown in the following screenshot:

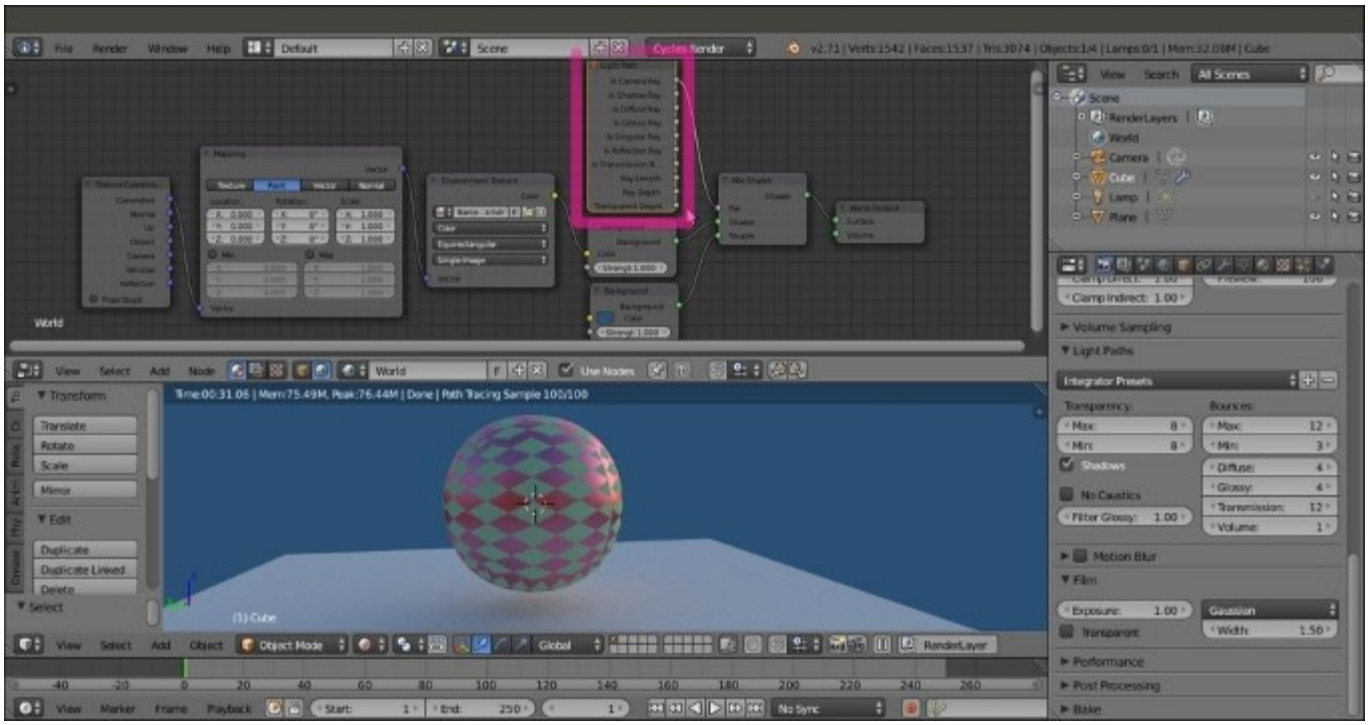


The previous World setting rendered with a transparent background

2. Now we can compose the rendered image with a blue background, both in external image editing software (such as GIMP, to stay inside FOSS) or directly in the Blender compositor.

A different way to render the two objects on a uniform blue background is to use a **Light Path** node:

1. If this is the case, deselect the **Transparent** item checkbox in the **Render** window to restore the sky background in the preview and in the rendering.
2. Click on the **World Output** node in the **Node Editor** window, press *G*, and move it to the right.
3. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** |**Mix Shader**) and move it to the link connecting the **Background** node to the **World Output** node, to paste it automatically between the two nodes.
4. Select the **Background** node in the **Node Editor** window. Press *Shift* + *D* to duplicate it and move it down.
5. Connect its output to the second input socket of the **Mix Shader** node. Click on its **Color** box to change the color to `R 0.023`, `G 0.083`, and `B 0.179`.
6. Now, add a **Light Path** node (press *Shift* + *A* and navigate to **Input** | **Light Path**).
7. Connect the **Is Camera Ray** output of the **Light Path** node to the **Fac** input socket of the **Mix Shader** node, and voilà! The objects in the scene are lit by the HDR image connected to the first **Background** node, but they appear in a sky that is colored as set in the **Color** box of the second **Background** node. This is shown in the following screenshot:



The use of the Path Light node as a factor to have a different background than the HDR image still illuminating the scene

8. Save the file as `start_08.blend`.

How it works...

To explain this trick better, let's say we just created two different world materials: the first material with the texture and the second material with a plain blue color (this is not literally true; actually, the material is just one, containing the nodes of two ideally different worlds).

We mixed these two materials using the **Mix Shader** node. The upper green socket of the **Mix Shader** node is considered equal to a value of `0.000`, while the bottom green socket is considered equal to a value of `1.000`. As the name suggests, the **Light Path** node can set the path for the rays of light that are shot from the camera, if you remember. **Is Camera Ray** means that only the rays directly shot from the camera have a value of `1.000`, that is, not the reflected ones, or the transmitted ones, or whatever, which have a value of `0.000`.

Thus, because the textured world is connected to a socket equal to the value of `0.000`, we don't see it directly as a background, but only see its effect on the objects lit from the reflected light or from the HDR image. The World of the blue sky, which is connected to the input socket of value `1.000` instead, is seen as a background because the light rays shot from the camera directly hit the sky.

There's more...

Just after the **Surface** subpanel, in the **World** window, there is the **Ambient Occlusion** subpanel. **Ambient occlusion** is a lighting method used to emphasize the shapes or the details of a surface, based on how much a point on that surface is occluded by the nearby surfaces. Ambient occlusion can replace the Global Illumination effect in some cases, though not the same. For example, to render interiors with fast and noise-free results, ambient occlusion is a cheap way to get an effect that looks a bit like indirect lighting.

There is a checkbox to enable **Ambient Occlusion**, along with the following sliders:

- **Factor**: This is used for the strength of the ambient occlusion. A value of `1.00` is equivalent to a white World.
- **Distance**: This is the distance from a shading point to the trace rays. A shorter distance emphasizes nearby features, while a longer distance takes into account objects that are further away.

The **Ambient Occlusion** feature is only applied to the **Diffuse BSDF** component of a material. The **Glossy** or **Transmission BSDF** components are not affected. Instead, the transparency of a surface is taken into account. For example, a half-transparent surface will only half-occlude other surfaces.

Creating a mesh-light material

In this recipe, we will see how to create a mesh-light material to be assigned to any mesh object and used as a source to light the scene.

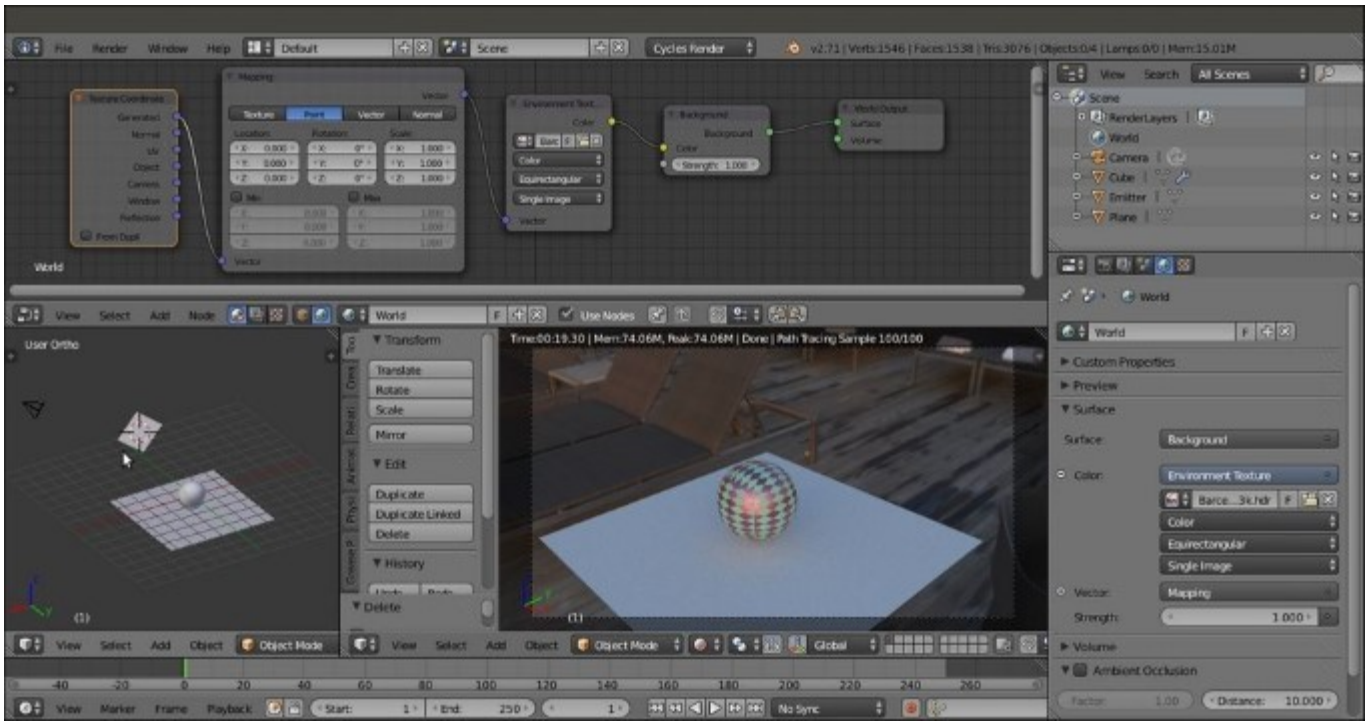
Getting ready

Until now, we have used the default Lamp (a Point light) already present in the scene to light the scene. By enabling the node system for the Lamp, we have seen that it uses a material created by connecting an **Emission** node to the **Lamp Output** node.

The good news is that just because it's a material node, we can assign an **Emission** shader to a mesh, for example, to a Plane conveniently located, scaled, and rotated to point to the scene that is the center of interest. Such a light-emitting mesh is called a mesh-light. Being a mesh, the **Emission** shader node output must be connected to the **Surface** (or the **Volume**) input socket of a **Material Output** node instead of the **Lamp Output** node.

Light emission coming from a surface and not from a point is a lot more diffused and softer than the light from a Lamp. A mesh-light can be any mesh of any shape, so it can be used as an object taking part in the scene and be the real light source of the rendering at the same time, for example, a table lamp, or a neon sign, or a television screen. As a pure light-emitting Plane, it's usually used as a sort of photographic diffuser. Two or three strategically placed mesh-lights can realistically simulate a photo studio situation. To replace the Lamp with a mesh-light, Plane perform the following steps:

1. Call the **Blender User Preferences** panel (*Ctrl* + *Alt* + *U*), navigate to the **Addons** tab, and click on **3D View** under **Categories** on the left. Check the **Copy Attributes Menu** box to the right-hand side of the **3D View** option, and click on the **Save User Settings** button in the bottom-left corner of the panel. Then close the panel.
2. Starting from the `start_07.blend` file, click on the eye icon of **Lamp** in the **Outliner** to enable its visibility again.
3. Right-click on the **Lamp** in the 3D view and press *Shift* + *S* to bring up the **Snap** menu. Click on the **Cursor to Selected** item.
4. Press *Shift* + *A* with the mouse pointer in the 3D view and add a Plane to the scene at the 3D Cursor's location.
5. Press *Shift* and select the Lamp. Now you have both the recently added Plane and the Lamp selected, and the latter is the active object.
6. Press *Ctrl* + *C* to open the **Copy Attributes** menu and select the **Copy Rotation** item.
7. Rename this Plane as `Emitter`.
8. Right-click on the Lamp in the 3D view and press *X* to delete it.
9. Put the mouse pointer on the 3D view and press *0* from the numeric keypad to go to Camera view.
10. From the **Viewport Shading** menu in the window's header, select the **Rendered** mode (or put the mouse cursor on the **Camera** view and press *Shift* + *Z*):



A Plane set as a mesh-light to replace the Lamp, and the previous HDR image as the background

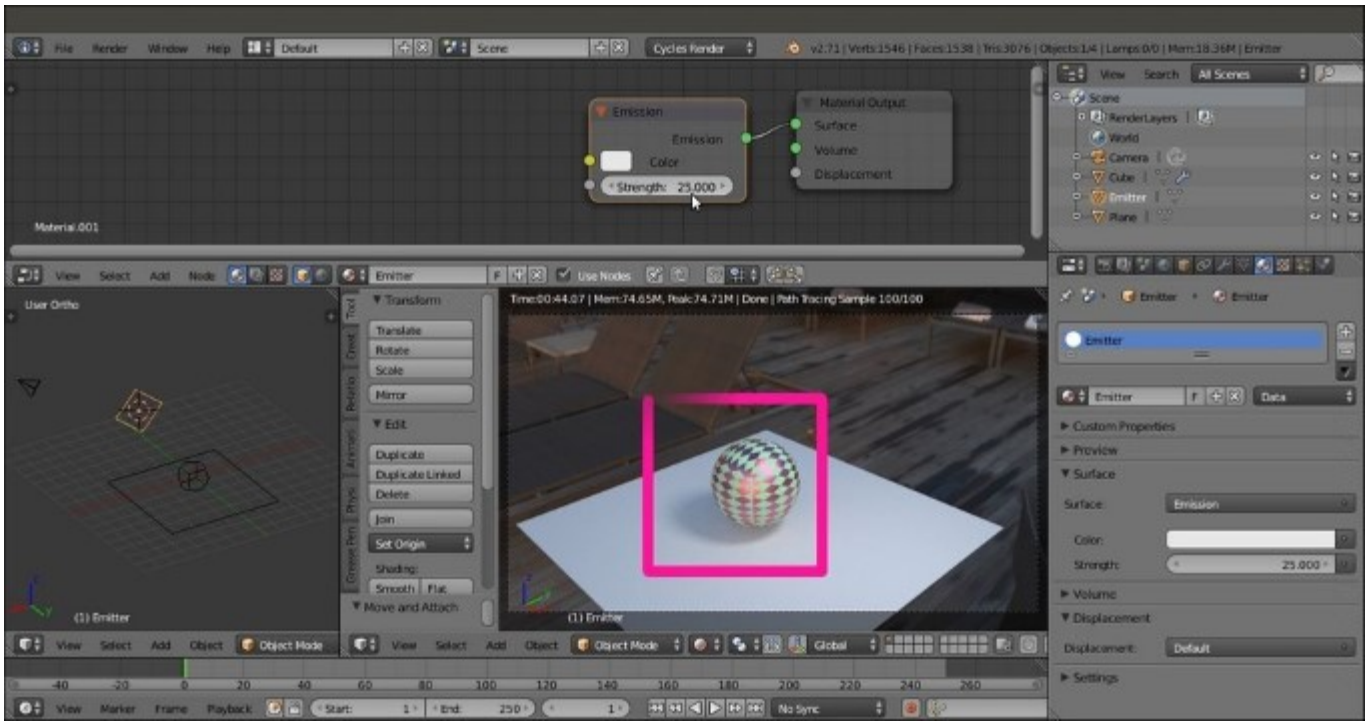
11. Save the file as `start_09.blend`.

How to do it...

Now let's create the emission material and also take a look at the setup for the softness of the projected shadows:

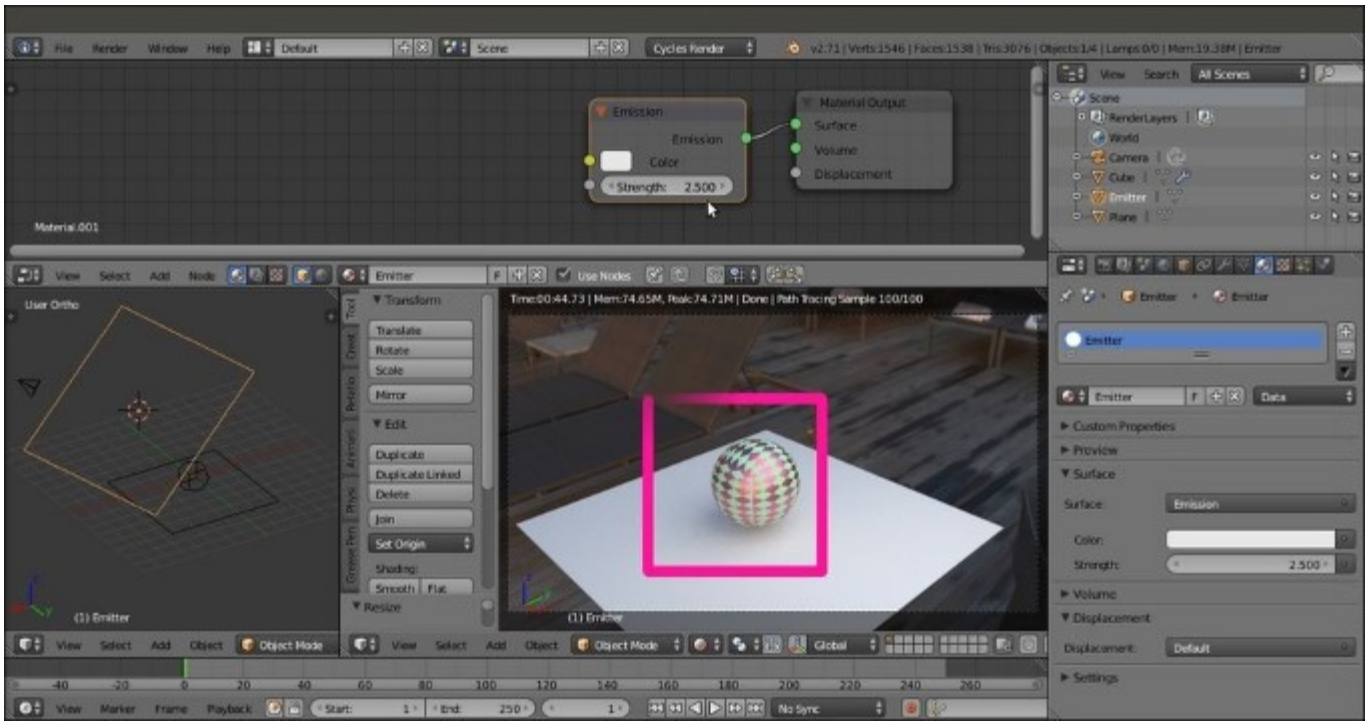
1. Select the **Emitter** plane and click on the little cube icon on the header of the **Node Editor** window.
2. Click on the **New** button in the header and rename the material as `Emitter`.
3. In the **Properties** panel, go to the **Material** window, and under the **Surface** tab, click on the **Surface** button to switch the **Diffuse BSDF** shader with an **Emission** shader. Leave the default color unchanged (**RGB** `0.800`) and set the **Strength** slider to `25.000`.
4. Save the file.

   The situation so far is as follows:



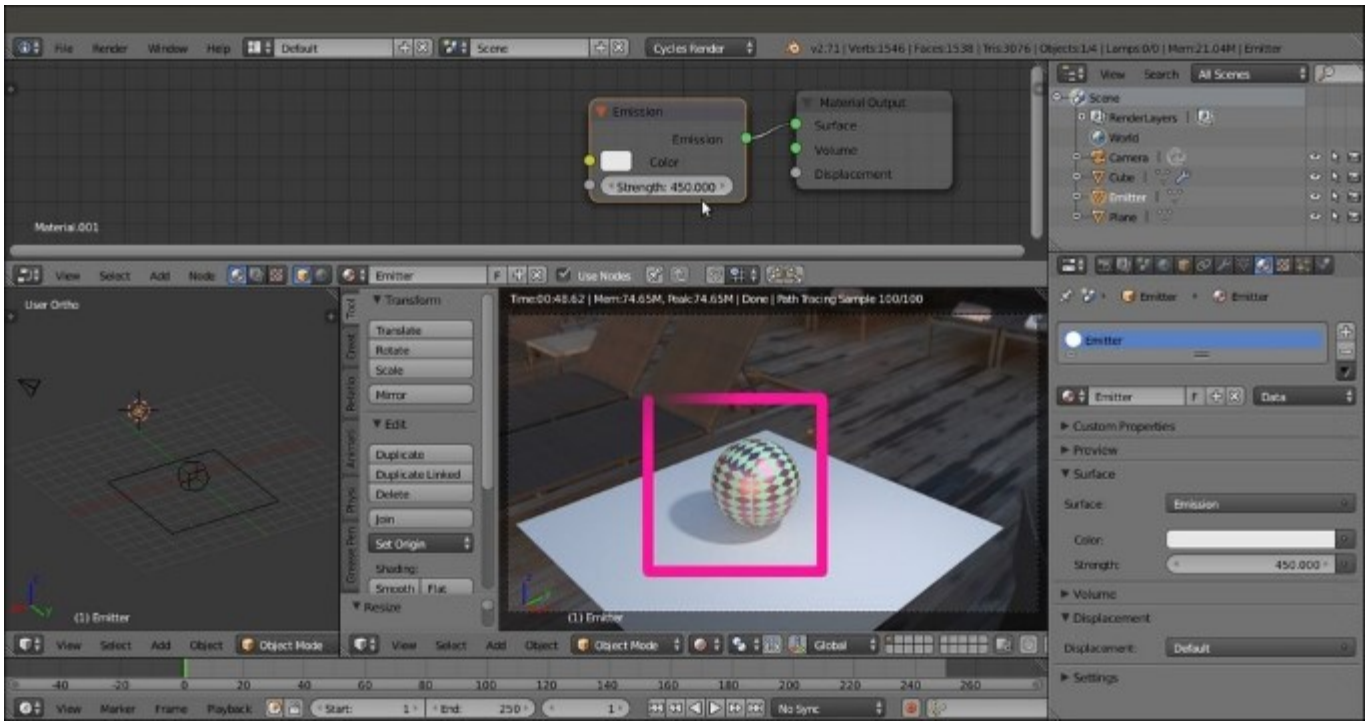
The mesh-light emission material with increased strength

5. In the 3D view, scale the **Emitter** plane five times bigger (press *S*, then enter *5*, and press *Enter*), and then set the **Strength** slider to `2.500`.
6. Save the file as `start_10.blend`. Now look at the softer shadow, as shown in the following screenshot:



Scaling the mesh-light bigger and decreasing the emission strength to have softer shadows

7. Now let's scale the **Emitter** plane a lot smaller (press *S*, then type *0.05*, and press *Enter*) and set the **Strength** slider to `450.000`.
8. Save the file as `start_11.blend`. Look at the crisper shadow in the **Rendered** preview, as shown in this screenshot:



Scaling the mesh-light smaller and increasing the emission strength to have crisper shadows

How it works...

From steps 5 to 7, we saw how a mesh-light can be scaled bigger or smaller to obtain a softer (in the first case) or a sharper (in the second case) shadow, respectively. The **Strength** value must be adjusted for the light intensity to remain consistent, or the mesh-light must be moved closer or more distant from the scene.

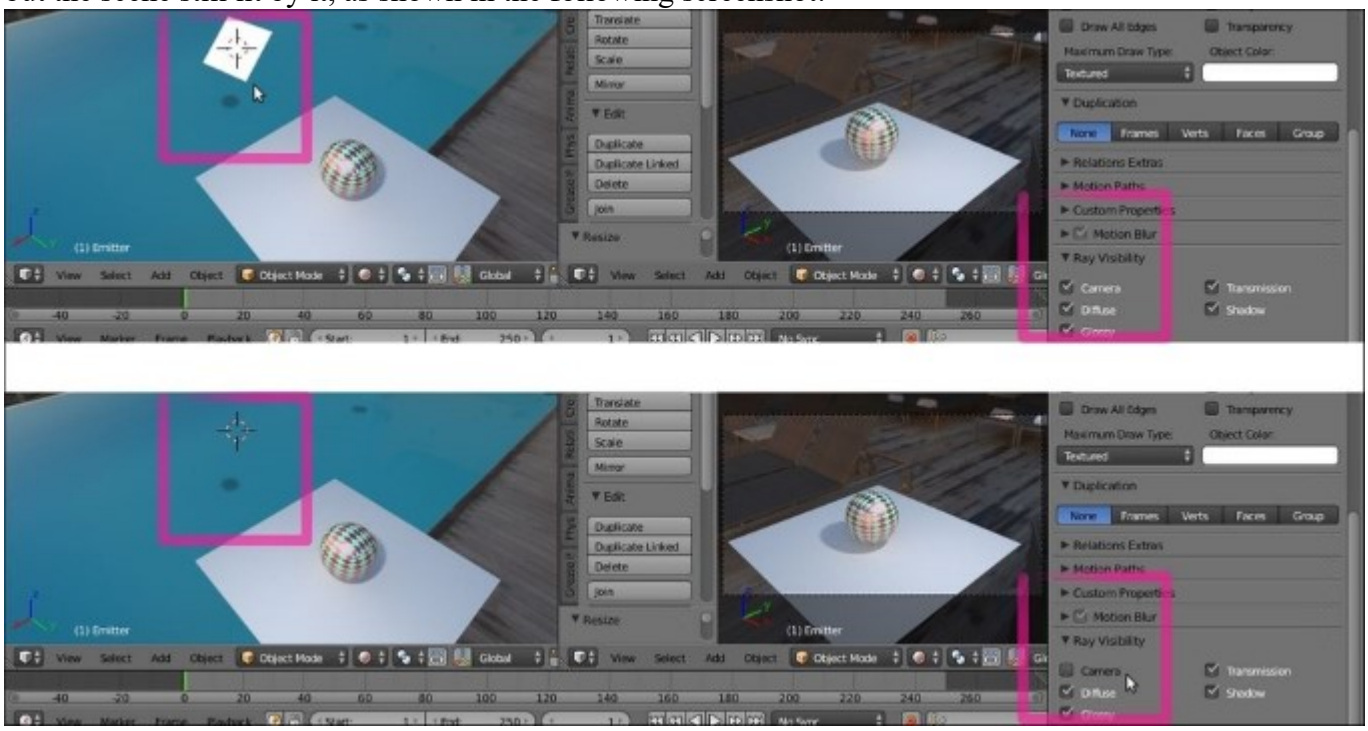
Scaling the mesh-light is basically the same as setting the size value for a Lamp. For Lamps, the softness of shadows can be set by the **Size** value to the left of the **Cast Shadow** option in the **Lamp** window, under the **Properties** panel (by default, the **Size** value is set to `1.000`). At a value of `0.000`, the shadow is at its maximum crispness, or sharpness. If the **Size** value is increased, the softness of the shadow increases too.

Unlike the mesh-light, varying the **Size** value of a Lamp doesn't require us to adjust the **Strength** value to keep the same light intensity.

There's more...

In several cases, you might not want the emitters to appear in your rendering. There are node arrangements to accomplish this (such as using the **Light Path** node in a way quite similar to the *Setting the World material* recipe we have seen before), but the easiest way to do this is as follows:

1. Start with the last saved blend (`start_11.blend`) and put the mouse cursor on the orthogonal 3D view to the left of the screen. Press the *3* key to navigate to the **Side** view. Then press *Shift* + *Z* to go in the **Rendered** mode to also see the **Emitter** plane rendered (be warned that if your computer can't easily render two windows at the same time, you must temporarily turn off the rendering for the **Camera** view).
2. With the Emitter plane still selected, navigate to the **Object** window under the **Properties** panel.
3. Look at the **Ray Visibility** tab (usually at the bottom of the **Properties** panel), where there are five items: **Camera**, **Diffuse**, **Glossy**, **Transmission** and **Shadows**, with the corresponding checked boxes.
4. Uncheck the **Camera** item and watch the **Emitter** plane disappear in the rendered 3D window, but the scene still lit by it, as shown in the following screenshot:

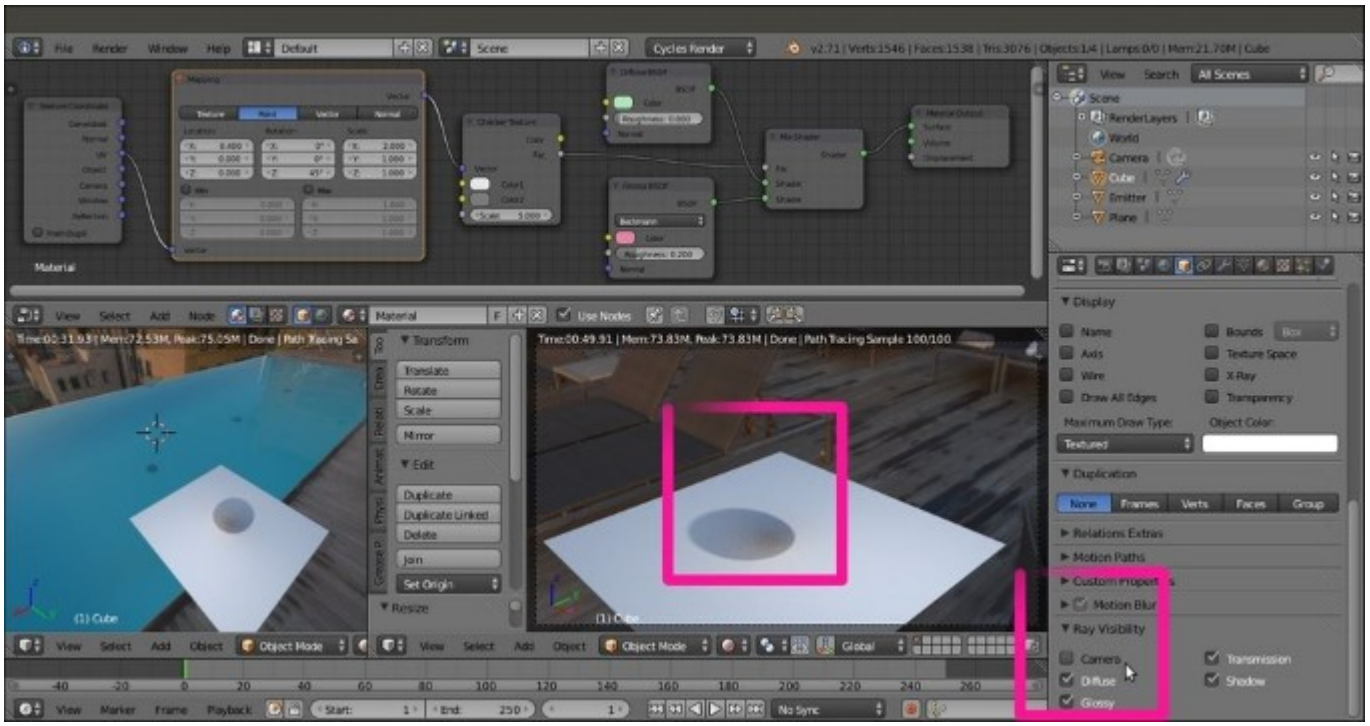


Disabling the Camera item in the Ray Visibility subpanel to hide the mesh-light Plane from the rendering

When you disable any one of the items, the corresponding property won't take part in the rendering. In our case, when the **Camera** box is unchecked, the mesh-light won't be rendered but it will still emit light. Be careful that the **Emitter** plane is not renderable at this moment, but because all the other items in the tab are still checked, it can be reflected and could cast its own shadow on other objects.

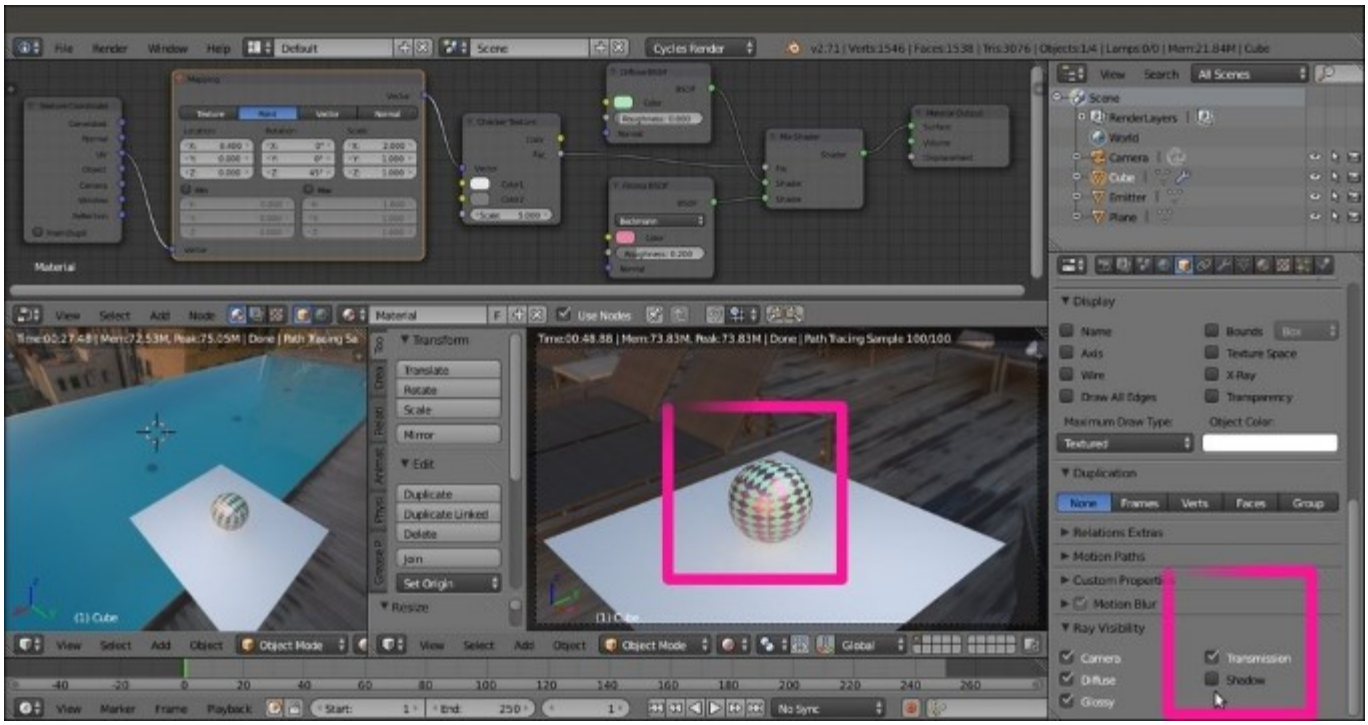
5. Now reselect the Spheroid (remember that unless you have renamed it, its name in the **Outliner** remains as Cube). Next, from the **Ray Visibility** tab in the **Object** window under the **Properties** panel, uncheck the **Camera** item.

Now the Spheroid has disappeared, but it's still casting its shadow on the floor Plane, as shown in this screenshot:



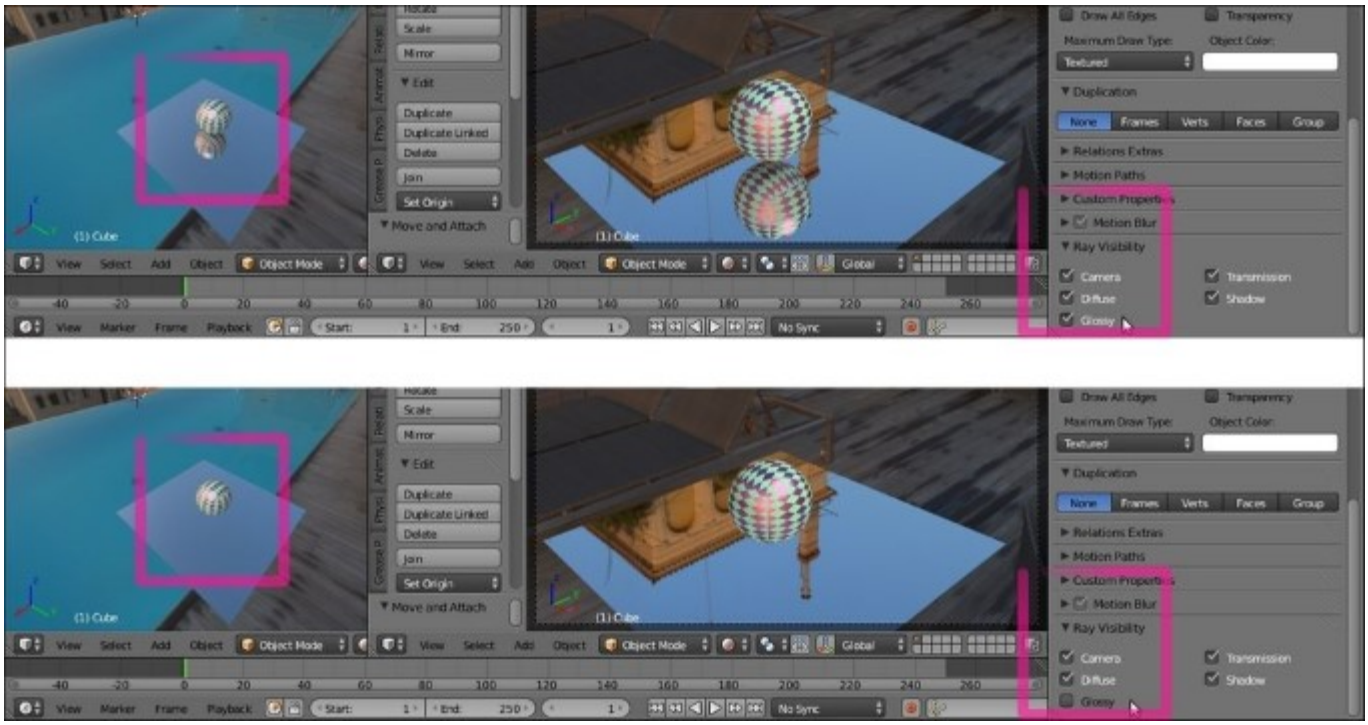
Disabling the Camera item to hide the Spheroid object from the rendering (but keeping the shadows on the floor)

6. Now check the **Camera** item again and uncheck the **Shadow** box. In this case, the Spheroid is visible again but doesn't cast a shadow, as shown in the following screenshot:



Disabling the Shadow item to have the Spheroid object rendered but without the shadows on the floor Plane

7. Save the file as `start_12.blend`. Let's try tweaking this a little.
8. Check the **Shadow** box for the Spheroid again, and select the floor Plane. Go to the **Material** window under the **Properties** panel, and click on the **New** button to assign a new material (**Material.001**).
9. Still in the **Material** window under the **Properties** panel, switch the **Diffuse BSDF** shader with a **Glossy BSDF** shader. The floor Plane is now acting as a perfect mirror, reflecting the Spheroid and the HDR image we formerly set in the World material.
10. Go back to the **Object** window and reselect the Spheroid. In the **Ray Visibility** tab, uncheck the **Glossy** item and watch the Spheroid, which is still rendered but not reflected by the mirror floor Plane, as shown in the following screenshot:



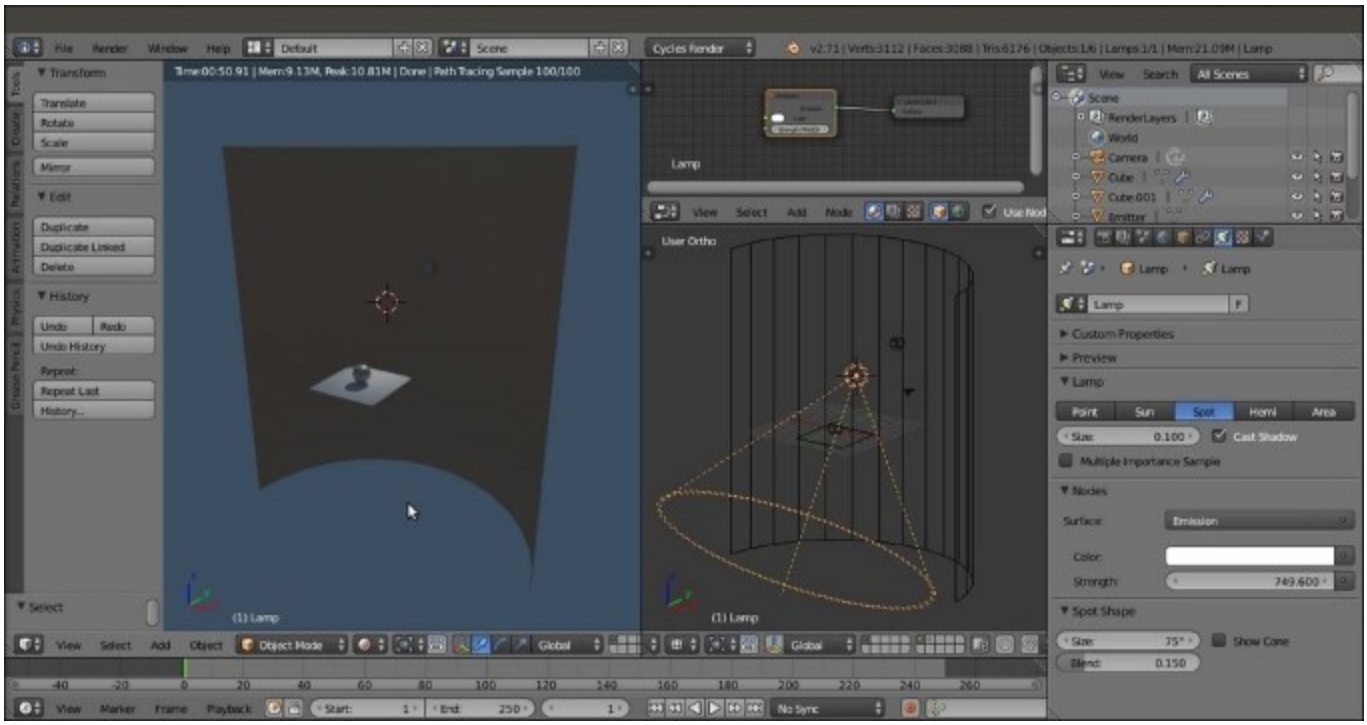
By disabling the Glossy item, we have the Spheroid object not mirrored by the glossy floor Plane

11. Save the file as `start_13.blend`.

Of course, the **Ray Visibility** trick we've just seen is not needed for Lamps because a Lamp cannot be rendered at all. At the moment, only **Point**, **Spot**, **Area**, and **Sun** lamps are supported inside Cycles. **Hemi** lamps are rendered as **Sun** lamps.

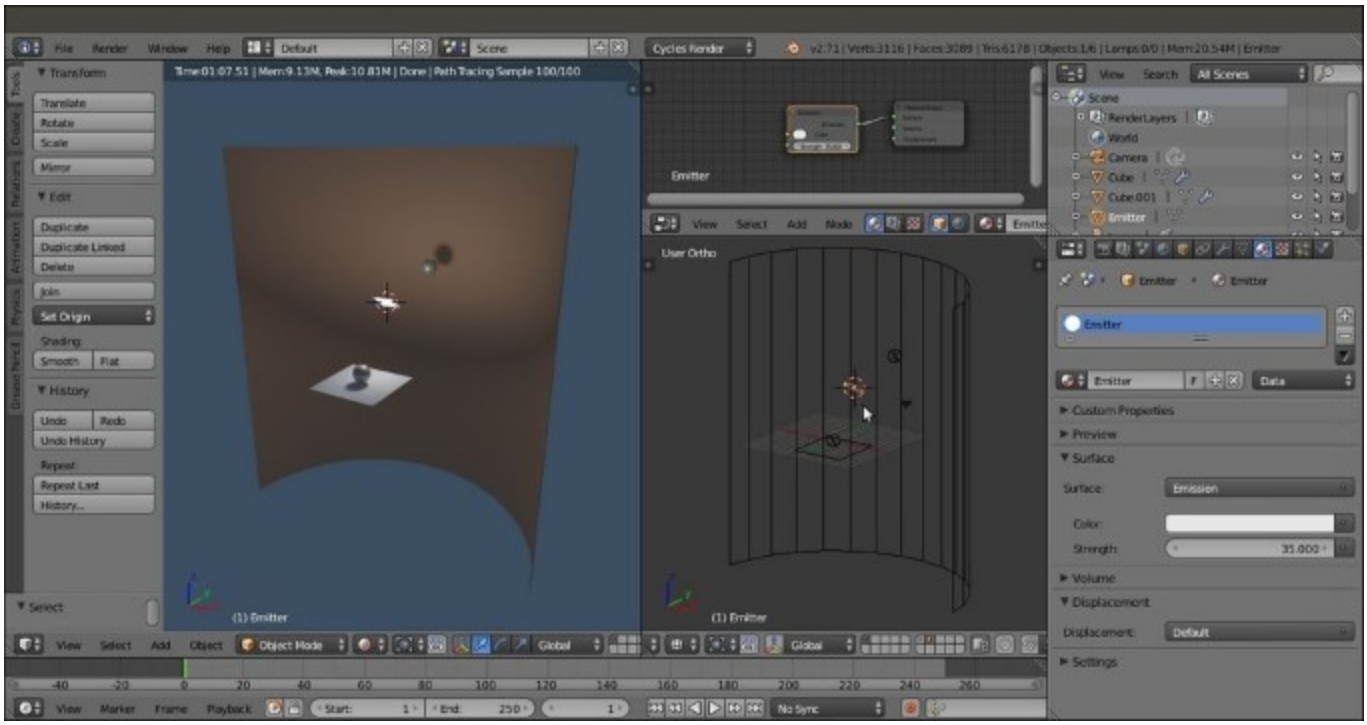
Both Lamps and mesh-lights can use textures too, for example, project colored lights on the scene, but only a mesh-light can be unwrapped and UV-mapped with an image map.

One advantage Lamps have over mesh-lights is that they can be made unidirectional easily, that is, apart from **Point** lamps, they cast light in only one direction. The following screenshot shows the casting of light with a Spot Lamp:



A Spot Lamp allows light to point in just one direction

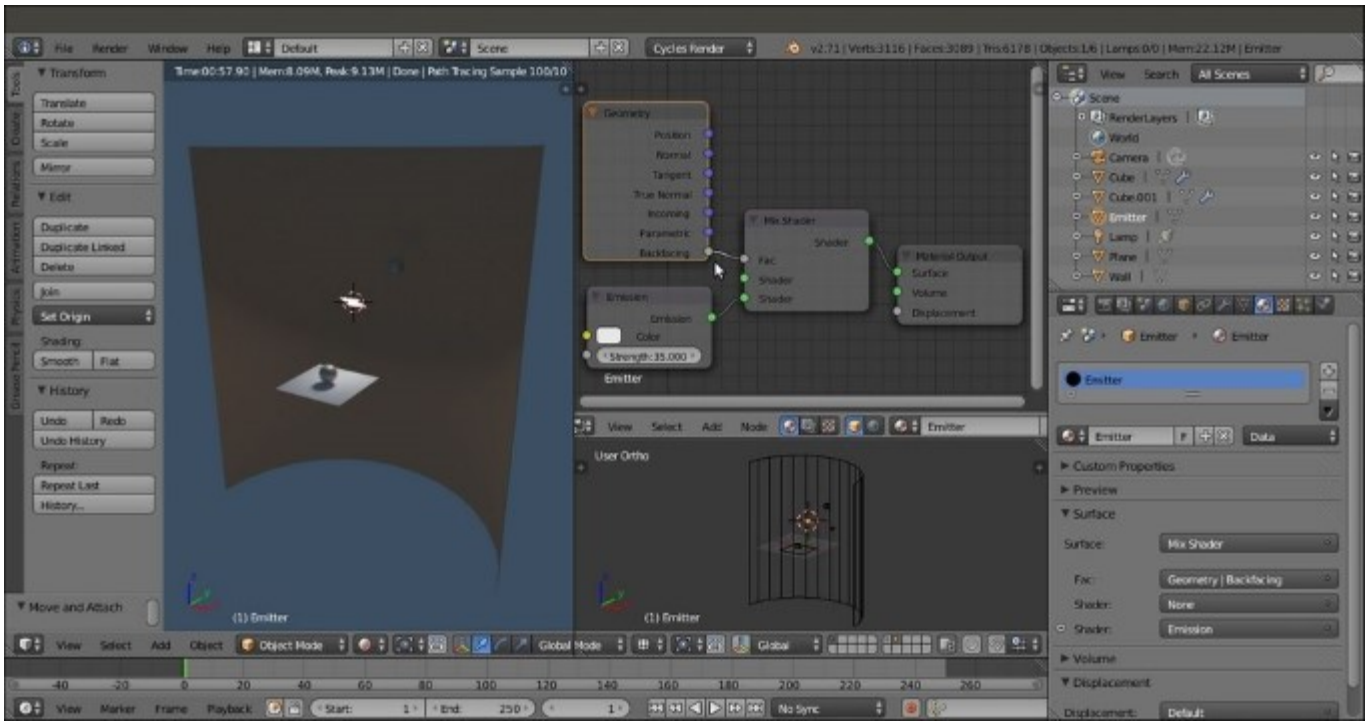
In the preceding screenshot, you can see that only the Plane and the Spheroid in front of the Spot lamp receive light. With a mesh-light plane replacing the Spot lamp, objects in both the front and the back (the half-cylindrical **Wall** and the second Spheroid) receive light.



A mesh-light emitter illuminates the region backward and forward by default

What if we want to light the object in only one direction (Plane and Spheroid in front) with a mesh-light? Is there a way to make a light-emitting plane emit light only from one side and not the opposite side? Yes, there is; follow these steps:

1. Open the `01_meshlight.blend` file, which has prepared the scene used for the preceding screenshots, and be sure to enable only the first and the seventh layer.
2. Put the mouse cursor on the left vertical 3D view, and press *Shift* + *Z* to navigate in **Rendered** view mode.
3. Click on the **Emitter** item in the **Outliner** to select it (if not already selected), and put the mouse pointer in the **Node Editor** window. Add a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**) and move it to the link connecting the **Emission** node to the **Material Output** node to paste it in between them.
4. Add a **Geometry** node (press *Shift* + *A* and navigate to **Input** | **Geometry**) and connect its **Backfacing** output to the **Fac** input socket of the **Mix Shader** node.
5. Switch the **Emission** node output from the first **Shader** input socket of the **Mix Shader** node to the second node, as shown in the following screenshot:



Thanks to the Backfacing output of a Geometry node as Factor, a mesh-light can illuminate in only one direction

6. Save the file as `01_meshlight_final.blend`.

We have already seen that in a **Mix Shader** node, the first (upper) green **Shader** input socket is considered equal to a `0` value, while the second socket is considered equal to a `1` value. So, the **Backfacing** output of the **Geometry** node is telling Cycles to make the mesh-light plane emit light only in the face-normal direction, and to keep the opposite back-facing side of the plane black and non-emitting (just like a blank shader).

By switching the **Emission** node connection to the first **Mix Shader** input socket, it's obviously possible to invert the direction of the light emission.

Using volume materials

Very briefly (because there are dedicated recipes in the last chapter of this Cookbook), let's take a look at how volumetric materials work in Cycles.

Volumetric materials are exactly what they sound like. Instead of the surface of an object, Cycles renders the inner volume of that object, and this gives space to a lot of interesting possibilities—not only can elusive materials such as smoke, fire, clouds, or light transmission effects through the medium be realized, but peculiar shapes can also be obtained from the volume itself by Boolean operations made through material nodes.

The drawback is that volume materials are slow—a lot slower compared to the surface materials, but hopefully, this is an issue that will be fixed in some way in the future (be aware that from Version 2.72, volume materials are available on GPUs too).

Getting ready

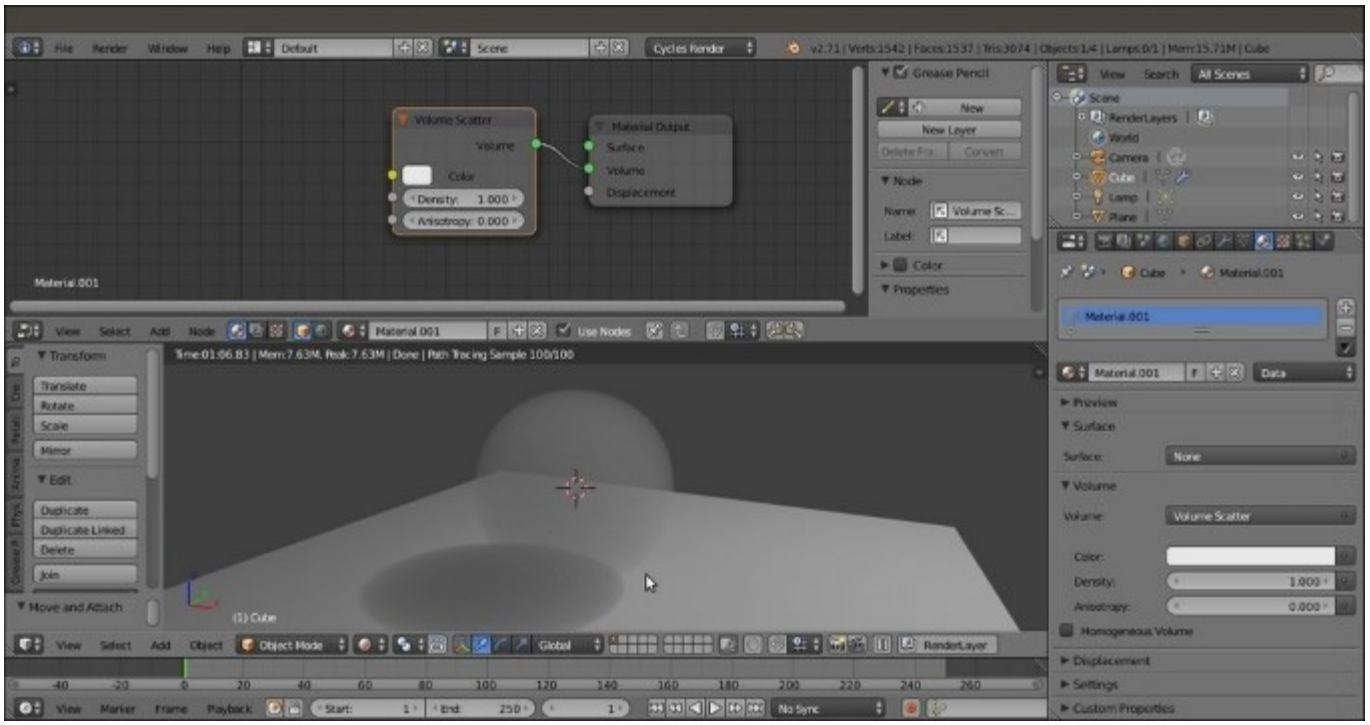
Let's start with our usual Spheroid blend file:

1. Open the `start_02.blend` file and delete the material assigned to the Spheroid.
2. Put the mouse cursor in the 3D view and press *Shift* + *Z* to navigate to the **Rendered** view.
3. Click on the **New** button to add a new material, and then switch the **Diffuse** node link from the **Surface** input socket to the **Volume** input socket of the **Material Output** node.

How to do it...

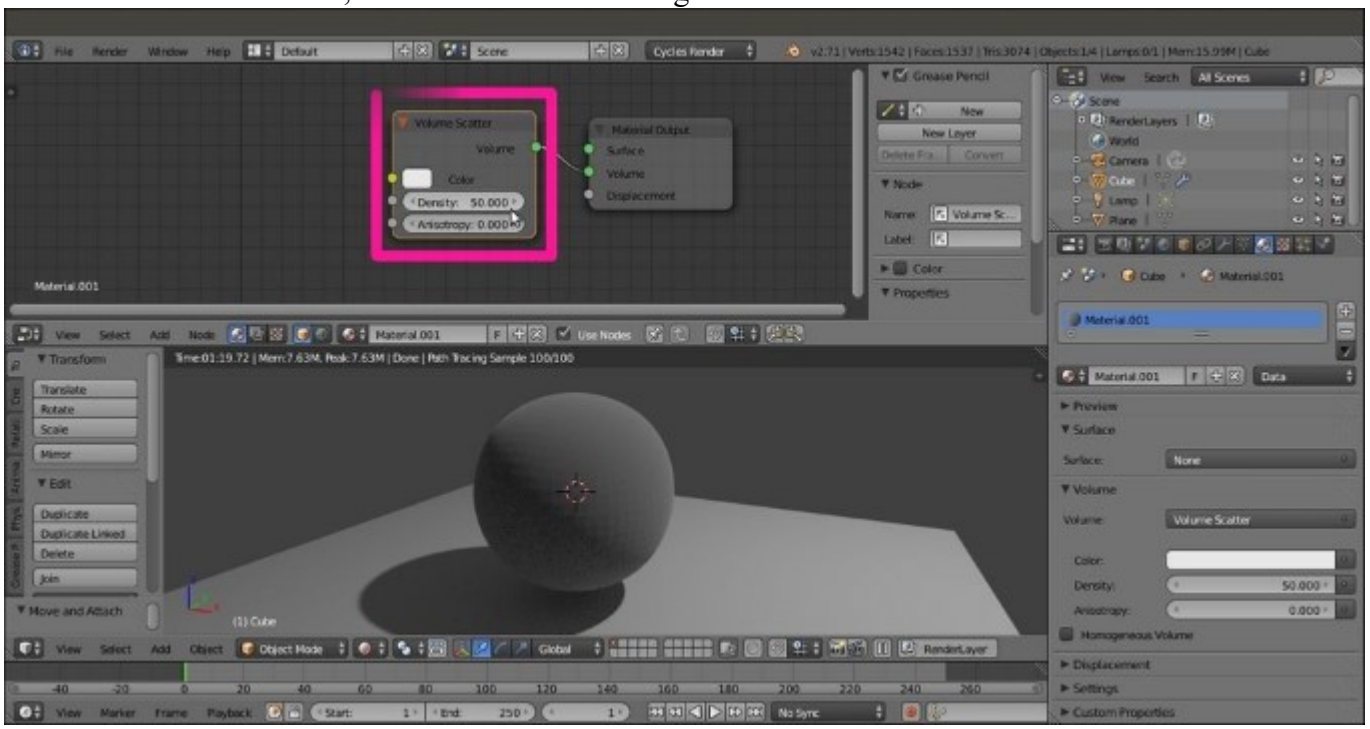
Now let's go to the volume section of the **Material** window with the following steps:

1. Go to the **Material** window under the **Properties** panel, and click on the **Diffuse BSDF** labeled button to the side of the **Volume** item. In the pop-up menu, select the **Volume Scatter** node as shown in this screenshot:



The Rendered preview of a Volume Scatter node assigned to the Spheroid

2. Change the **Density** value of the **Volume Scatter** node from `1.000` to `50.000`. The Spheroid looks a lot more solid now, as shown in the following screenshot:

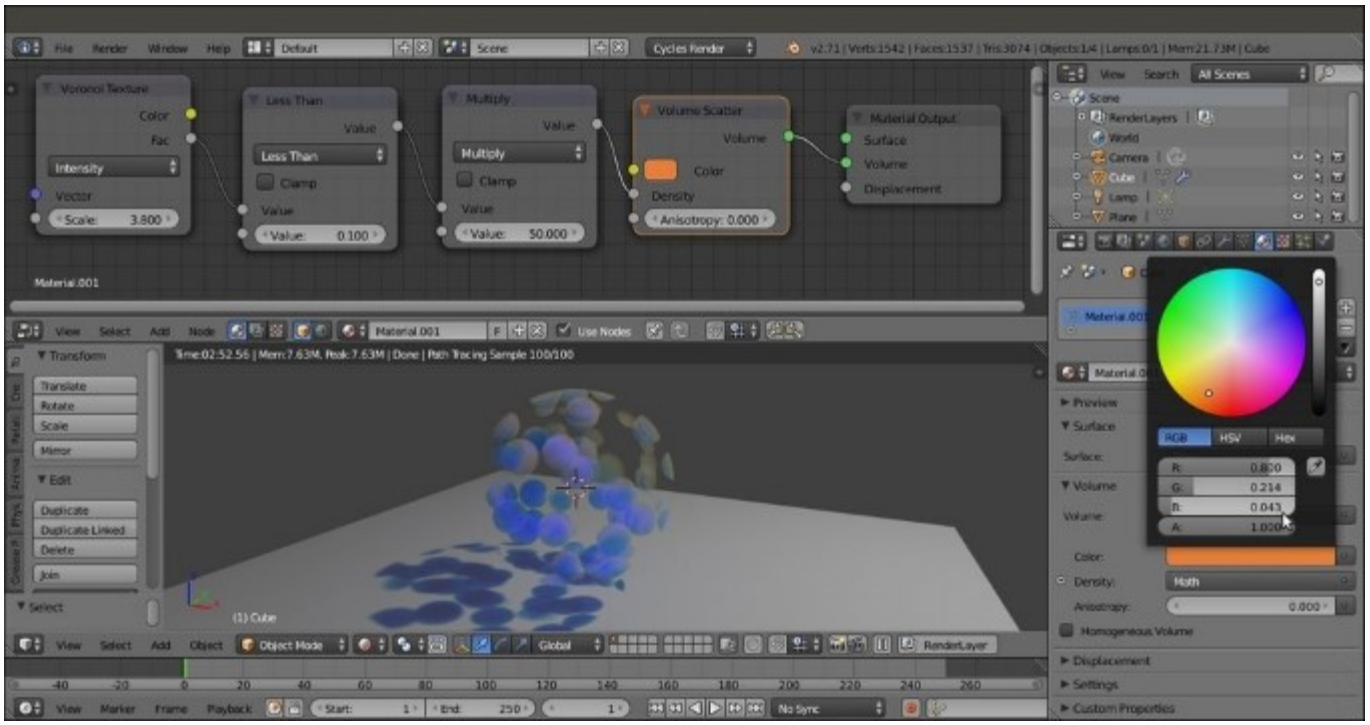


The effect of the Volume Scatter node with increased density

3. Add a **Voronoi Texture** node (Press *Shift* + *A* and navigate to **Texture** | **Voronoi Texture**). Connect the **Fac** output to the **Density** input socket of the **Volume Scatter** node. Set the **Voronoi** scale to `3.800`.
4. Add a **Math** node (Press *Shift* + *A* and navigate to **Converter** | **Math**) and paste it in the link between the **Voronoi Texture** and the **Volume Scatter** nodes. Set **Operation** to **Less Than** and second **Value** to `0.100`.
5. Add a second **Math** node and paste it right after the first node. Set the **Operation** to **Multiply** and second **Value** to `50.000`. Here is a screenshot of the output of a **Voronoi Texture** node for your reference:

The output of a Voronoi Texture node used as Factor for the density of the Volume Scatter node

6. Click on the **Color** button of the **Volume Scatter** node. Set the **RGB** values to `0.800`, `0.214`, and `0.043`, respectively.



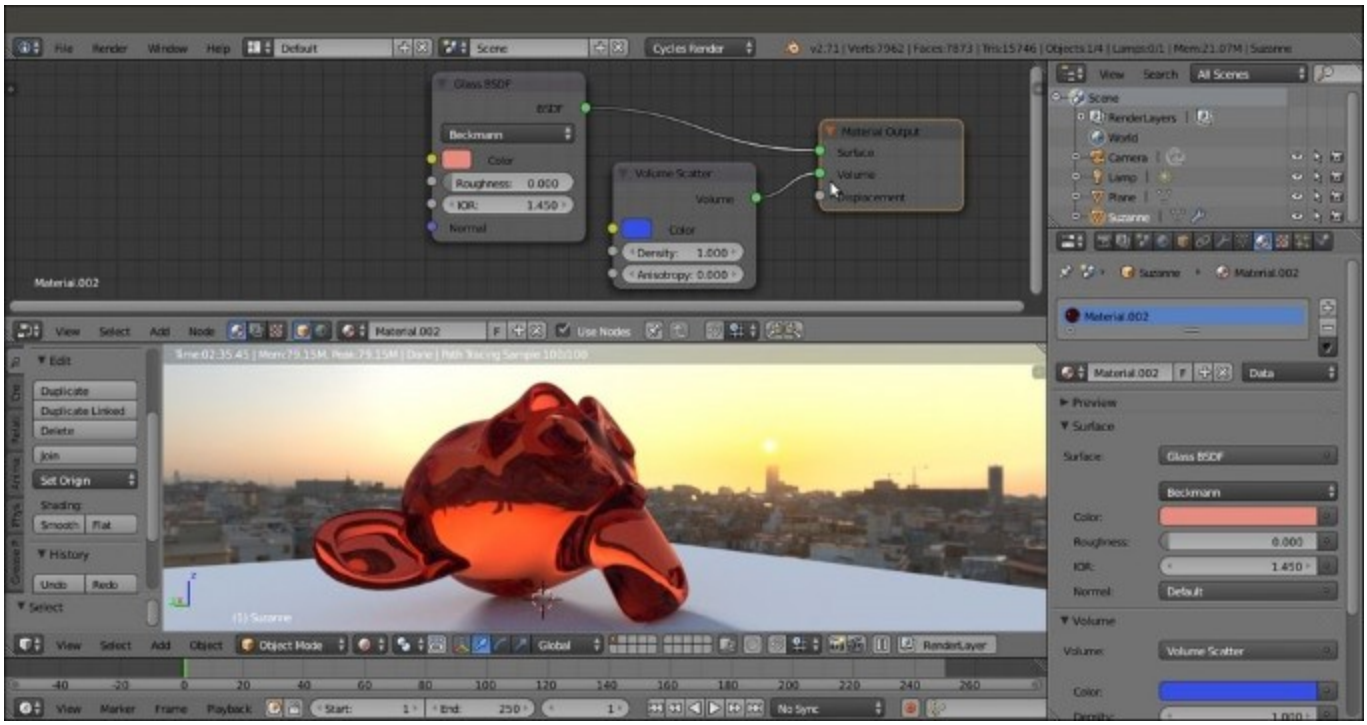
The scattered light is obviously of a hue complementary to the color assigned to the volume

7. Save the file as `01_volumetric.blend`.

How it works...

We have seen that when we increase the **Density** value of the **Volume Scatter** node, the Spheroid starts to look more and more solid. So, we used the output of a **Voronoi Texture** node and clamped it with a **Less Than** node to show only the values that are not beyond the `0.100` limit. Then we multiplied the value by `50.000`, thus increasing the density of the Voronoi spheres and making them appear as solid objects inside the Spheroid volume.

Remember that in this case, we rendered only the inside of the object and not the surface. Anyway, a combination of **Surface** and **Volume** is possible and can give interesting results, as shown in the following screenshot:



Combining a Glass shader for the surface with a Volume Scatter node for the inside of the mesh

There's more...

Volumes also work in the World. In fact, the **World Output** node now has a **Volume** input socket. By connecting a **Volume Scatter** or **Volume Absorption** node to the **World Output** node, it is possible to obtain several special effects, for example, fog, mist, atmospheric perspective, atmospheric scattering effects, and a body of water for an underwater scene. Clearly, it's also possible to fill this environment volume with textures.

In any case, you won't usually fill the entire World with a volumetric material because the World in Blender is considered as going to an infinite distance, and this would make the volume calculation too heavy. It's better to use a scaled Cube, properly placed and filled with the volume material.

To know more about volume materials, go to the last chapter of this Cookbook or to the documentation on the wiki at http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles/Materials/Volume.

Using displacement

The last input socket of the **Material Output** node is **Displacement**. Sadly, it seems that at the moment, its use is limited.

Getting ready

By enabling **Experimental** in the **Feature Set** option under the **Render** tab in the **Render** window, it's possible to have access to an incomplete displacement feature:

1. Open the `start_03.blend` file, select the Spheroid, and delete its material.
2. Go to the **Render** window under the **Properties** panel. In the **Render** tab, click on the **Feature Set** button, labeled with **Supported** by default, and select **Experimental**.
3. Go to the **Object data** window to find a new tab named **Displacement**, where we can choose between three options: **Bump**, **True**, and **Both** (the **Use Subdivision** and **Dicing Rate** buttons don't seem to work yet).

   ### Note

   Bump will give us the average bump effect, which is the same as connecting the texture output in the **Displacement** input of the **Material Output** node (this is a different way to have an overall bump effect, and it works without the need to set the **Feature Set** option to **Experimental**).

   By setting the method to **True**, we can have a displacement effect that is not different from the **Displace Modifier** output, and the mesh must be subdivided.

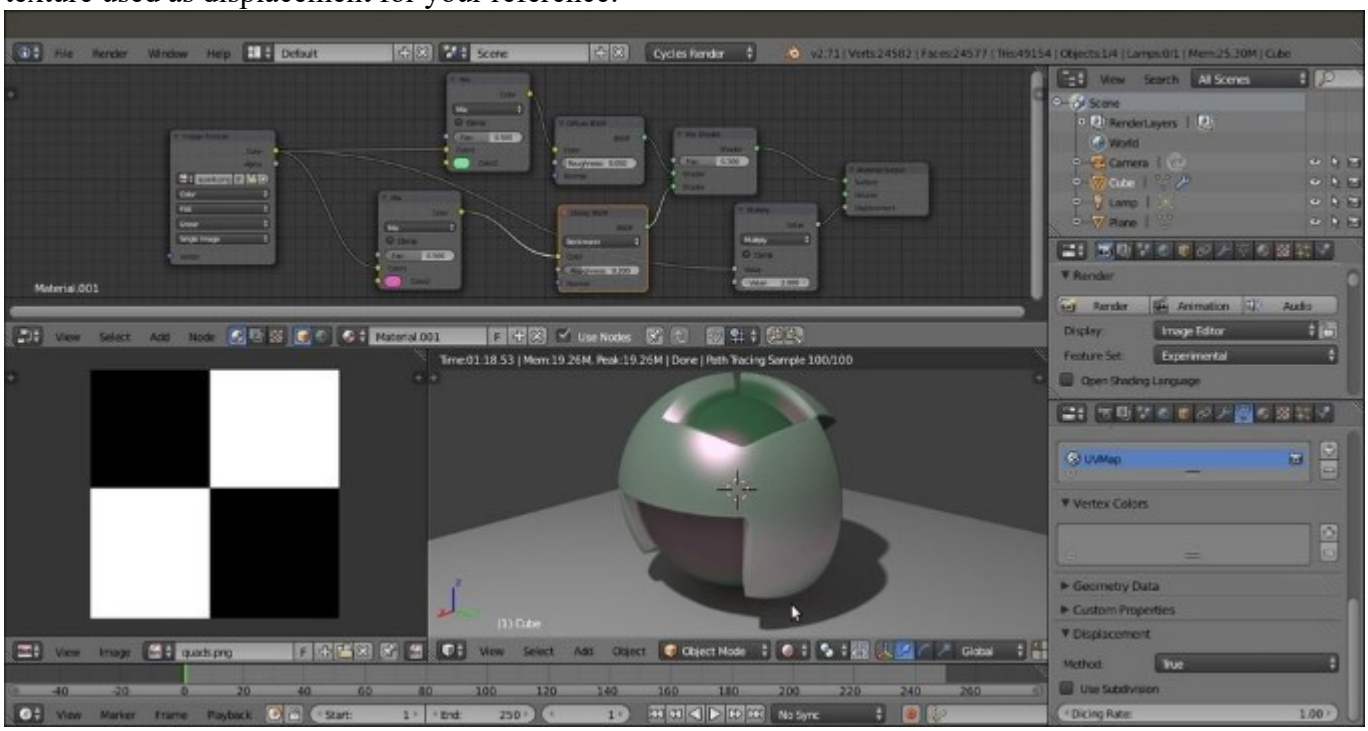
   Both will use the texture gray-scale values' information for a displacement and the bump effect together.

4. Select **True**.

How to do it...

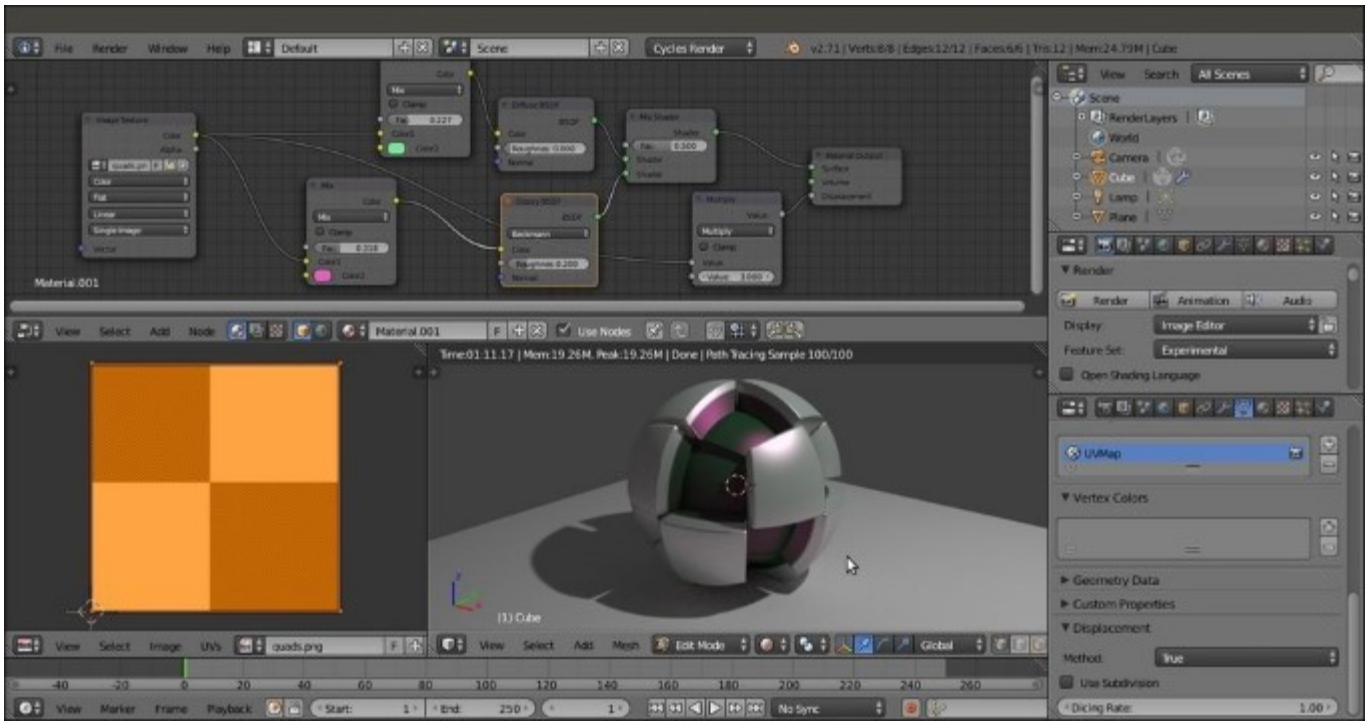
1. Go to the **Material** window under the **Properties** panel and click on the **New** button. In the **Displacement** tab, click on the **Default** button, and in the pop-up menu, select the **Image Texture** node.
2. Click on the **Open** button, browse to the `textures` folder, and load the `quads.png` image.
3. Split the bottom 3D window to open a **UV/Image Editor** window.
4. Press *Tab* to go to **Edit Mode**. Then press *U* with the mouse pointer in the 3D window. In the **UV Mapping** menu, select **Smart UV Project**, then load the `quads.png` image in the `UV/Image Editor`, and press *Tab* again to go out of **Edit Mode**. Note that this is the quicker way to unwrap the Spheroid, which is still a Cube at its lower level of subdivision. If you want, you can do a better unwrapping by placing seams to unfold it and by selecting a normal **Unwrap** option from the pop-up menu.
5. Go to the **Object modifiers** window and raise the **Subdivisions** levels for both **View** and **Render** to `6`.

6. Add a **Math** node (press *Shift* + *A* and navigate to **Converter** | **Math**) and paste it between the **Image Texture** node and the **Material Output**. Set **Operation** to **Multiply**, and the second option, **Value**, to `2.000` (if you don't see any modification in the rendered preview, it's an update issue, which can be solved by pressing *Tab* twice to go in and out of **Edit Mode**).
7. Add a **Glossy** node (press *Shift* + *A* and navigate to **Shader** | **Glossy BSDF**) and a **Mix Shader** node (press *Shift* + *A* and navigate to **Shader** | **Mix Shader**), and connect them to build the average basic material we already know.
8. Add two **MixRGB** nodes (press *Shift* + *A* and navigate to **Color** | **MixRGB**) and connect them to the color input sockets of the **Diffuse** and the **Glossy** nodes.
9. Finally, connect the color output of the **Image Texture** node to the **Color1** input sockets of the **MixRGB** nodes, and set colors for the **Color2** sockets. Here is a screenshot of a checker image texture used as displacement for your reference:



A checker image texture used as a color and output for the Rendered displacement of the Spheroid

Instead of the **Smart UV Project** option to unwrap the Spheroid, try the default `1:1` UV Mapping (the **Reset** item in the menu, which gives the whole image mapped on each face). The following screenshot shows the checker image texture used with the different unwrap:



The checker image texture used as a color and output for the rendered displacement of a Spheroid with a different unwrap

10. Save the file as `9931OS_01_displacement.blend`.

In any case, this is just for a temporary demonstration; the feature is still incomplete. At the moment, it seems to work quite well only if the texture is mapped with UV coordinates. This is definitely going to change in the future.

How it works...

Simply put, the gray-scale values of the texture are multiplied by the value we put in the second slider of the **Math** node. For example, if we set a value of `0.500`, the intensity of the effect will be the half of the default value (*1.000 x 0.500 = 0.500*). With a value of `3.000`, the effect would be three times the default value, and so on. Similar to Blender Internal, the value can also be set as negative, thereby inverting the direction of the displacement.

Chapter 2. Managing Cycles Materials

In this chapter, we will be covering the following recipes:

- Preparing an ideal Cycles interface for material creation
- Naming materials and textures
- Creating node groups
- Grouping nodes under frames for easier reading
- Linking materials and node groups

Introduction

As with Blender Internal materials, Cycles materials can (and should) be organized to optimize your workflow.

Material nodes in Cycles can easily grow quite complex, and it's sometimes a good idea to split and label the different parts of a shader's network, just to make the meaning of the different sections clearer (even to yourself because maybe at a certain point of your workflow, you will forget how exactly that 120-node material you made a couple of months ago works). Moreover, organized materials can be easily reused in other files and projects or as parts of bigger and different materials.

Organization of materials is basically done by grouping them or giving them proper names and defined locations so that they can be easily found in the hard disk.