Chapter 7. Haunted House – Adding Materials and Lights in Cycles

This chapter will be devoted to the Cycles render engine. You will learn how to achieve a convincing render of the haunted house by understanding the different types of light work and by creating complex materials using the previously made textures. You will learn some nice tricks such as how to produce normal maps of our hand-painted textures without leaving Blender or how to create realistic-looking grass. You will also discover how to use the Cycles baking tool. In order to conclude our project, we will show you how to integrate a mist effect in the final composition.

In this chapter, we will cover the following topics:

- Understanding the essential settings of Cycles
- Using lights
- Painting and using an Image Base Lighting
- Creating basic materials with nodes
- Using procedural textures
- Baking textures in Cycles for real-time rendering

Understanding the basic settings of Cycles

To switch to the Cycles render engine, you must select it in the list of proposed engines that Blender offers in the menu bar. We will see in the first part of this chapter some of the very useful settings that should be known while using Cycles.

The sampling

If you directly try to make a render with Cycles without changing the parameters of Blender, you will certainly see some noise in the image. To make this less visible, one of the first things to do is change the sampling settings. Unlike Blender Internal, Cycles is a Raytracer Engine. While rendering, Cycles will send rays from the camera in order to generate pixels. The noise is due to a small amount of the samples. Cycles, therefore, needs more samples; the more sampling, the more accurate the final render.

The following sampling settings are in Properties editor. Just select **Render** | **Sampling**:

- **Render samples**: This is the number of samples for your renders. The more samples you add, the longer the rendering time will be.
- **Preview samples**: This is the number of samples Blender will calculate to preview your scene in the 3D viewport in Rendered Viewport Shading. A value between 20 and 50 samples is correct. This value depends on the performance of your computer.

The Render sample must be higher than the Preview sample.

Note

The GPU device

If you have a fairly recent CUDA®-compatible graphic card, you can opt for GPU rendering. This allows you to make renders very quickly and visualize your scene in the 3D viewport nearly in real time.

For this, go to **User Preferences** | **System** | **Computer Device** and select **CUDA**. Then, in **Properties**, go to **Render** | **Device** and select **GPU**.

Note that the most recent AMD GPU has been supported since Blender 2.75.

Clamp direct and indirect

This allows us to clamp the intensity of the rays of light launched from the camera. This can also help to reduce the noise effect, but it blurs the pixels together.

Light path settings

You will find the following light path settings in the menu:

- **Max and Min Bounces**: This is the number of minimum and maximum bounces a ray of light can do in the scene to render a pixel. This mimics the way photons bounce from objects in real life. The higher the value of the maximum bounce, the greater the precision will be, and the quality of the rendering will increase. A high value of minimum bounce will also improve the quality but may considerably increase the rendering time. It is advisable to set the same minimum and maximum value.
- **Filter Glossy**: This will blur glossy reflections and reduce the noise effect. You can put a 1.0 value.
- **Reflective and refractive caustics**: Caustics are light effects related to transparent and reflexive materials. We can observe this light effect with diamonds, for instance. This uses a lot of resources and generates some noise. By unchecking these two options, you can completely turn off these effects during rendering. In the case of our haunted house scene, we are going to disable them.

Performances

You will find the following performance settings in the menu:

- **Viewport BVH Type**: There are two types of this: **Dynamic** and **Static**. This is a way to let Cycles remember some of its rendering calculations for the next render. The **Static** option is highly recommended to optimize the rendering time, if you have no more polygonal modifications in your scene. Otherwise, you can use the **Dynamic** mode.
- **The Tiles**: This helps to manage the pixel groups that are to be rendered. So you can control their size along the *x* and *y* axes and also control the method to render the image (if you prefer to start rendering through the center of the image, or from left to right, and so on). The size of the pixel group to be rendered should be chosen according to your machine settings. If you are rendering with your CPU, you can choose a smaller tile size than if you were rendering with your GPU.

Note

For more information, you can have a look at the official Blender manual at these addresses:

http://www.blender.org/manual/render/cycles/settings/integrator.html

http://wiki.blender.org/index.php/Dev:2.6/Source/Render

Lighting

We are now going to look at a very important aspect of the rendering process: the lighting. Without lights, you won't see any objects, as in the real world. Good lighting can be hard to achieve, but it can give a nice atmosphere to the scene. One of the things that is true with a scene with good light is that you won't even notice the lights as they look like natural lighting. In order to get our lighting job done correctly, we are going to add a basic shader to every object in our scene.

Creating a testing material

Let's add a very basic material with Cycles in order to see the effects of the lights. In order to better understand the lights in the next section, we are going to create a blank scene and test our lights on a cube that is laid on a plane:

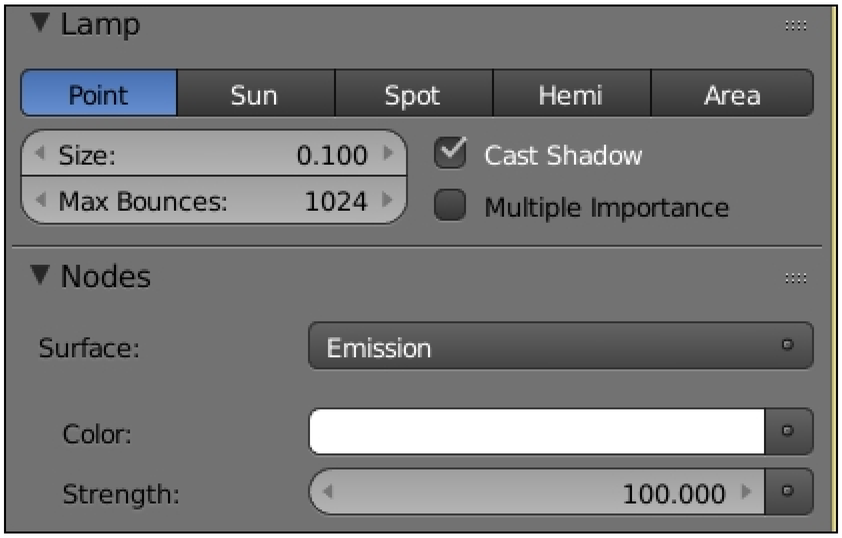
1. Let's start by creating a new scene and by adding a plane scaled ten times (**S > 10**) under the default cube.
2. We also want to delete the default light and turn the Cycles render engine on.
3. We aren't going to explore material creation in depth for now, so we advise you to follow these steps in order; later you will have more information about this process. We need to select the cube and open the **Material** tab of the **Properties** editor.
4. Now we will create a new material slot by clicking on the **New** button.
5. Under the **Surface** subpanel, we will click on the color and change its value to **1.0** in order to have a full white color. That's all for the material. It will be a handy material to test the different types of light.
6. The last thing we need to do is to add the same shader to the plane. To do this, we can copy the material of the cube. We will first select the object (or objects) to which we want to copy the material, in our case, the plane, and then we will select the object that owns the material that we want to copy. Then we will press *Ctrl* + *L* and select **Material**. The plane should now have the same white material.

Understanding the different types of light

The goal of this section is to understand how each type of light can affect our objects in a scene. We will give you a brief explanation and a short preview of what effects they can provide you with:

1. Before our tests, we will need to change the world shader that contributes to the lighting of the scene. If you press *Shift* + *Z* or change the **Viewport** Shading mode to **Rendered** in the 3D View header, you can see what the scene will look like, but you will see it in real-time in the viewport. Here you can clearly see the objects even if we don't have any lights. That's because the background color acts as if there was an ambient lighting.
2. If we go into the world settings of the Properties editor and change the Surface color to black, we will not see anything.
3. We can now add a light and have a better understanding of its effect without being disturbed by the world shader.

The settings of the lights can be found under the **Object Data** tab of the Properties editor (a yellow dot with a ray icon) while the light is selected. There are five types of lights (but four work in Cycles) that have many options in common: **Size** influences the hardness of the shadows they produce on objects and **Max Bounces** tells Blender the maximum number of bounces the light rays can travel. They also have the ability to cast shadows with the **Cast Shadow** checkbox turned on. Of course, they also have a strength that you can tweak in the **Nodes** subpanel. Note that, if you can't see the strength option, you need to click the **Use Node** button.

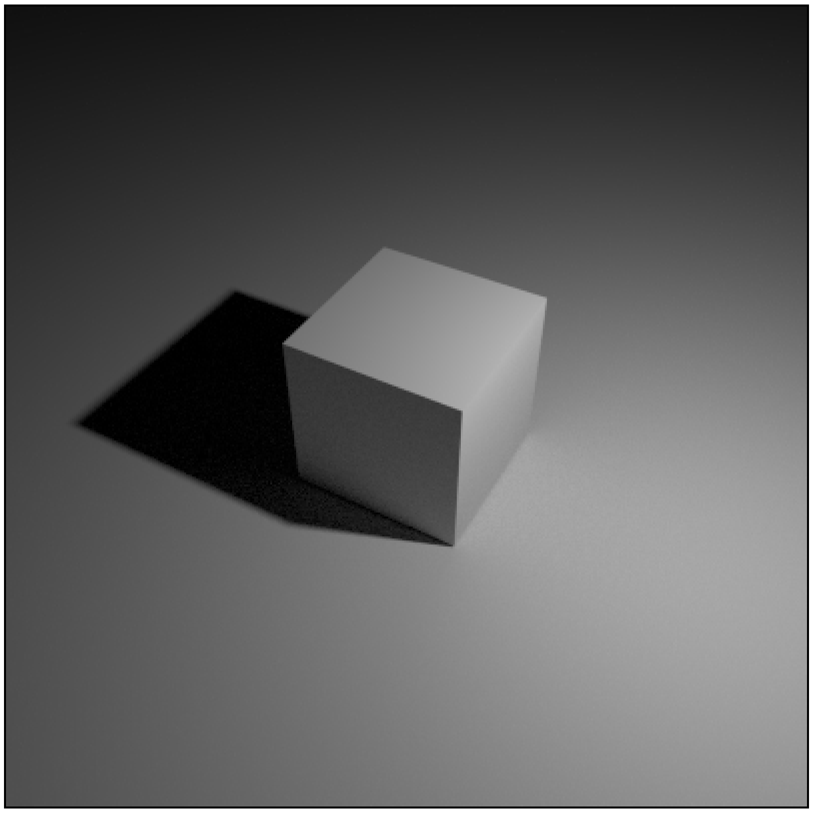


Size: 0.100
Max Bounces: 1024
Cast Shadow ✓
Multiple Importance
Surface: Emission
Color:
Strength: 100.000

The shared light options

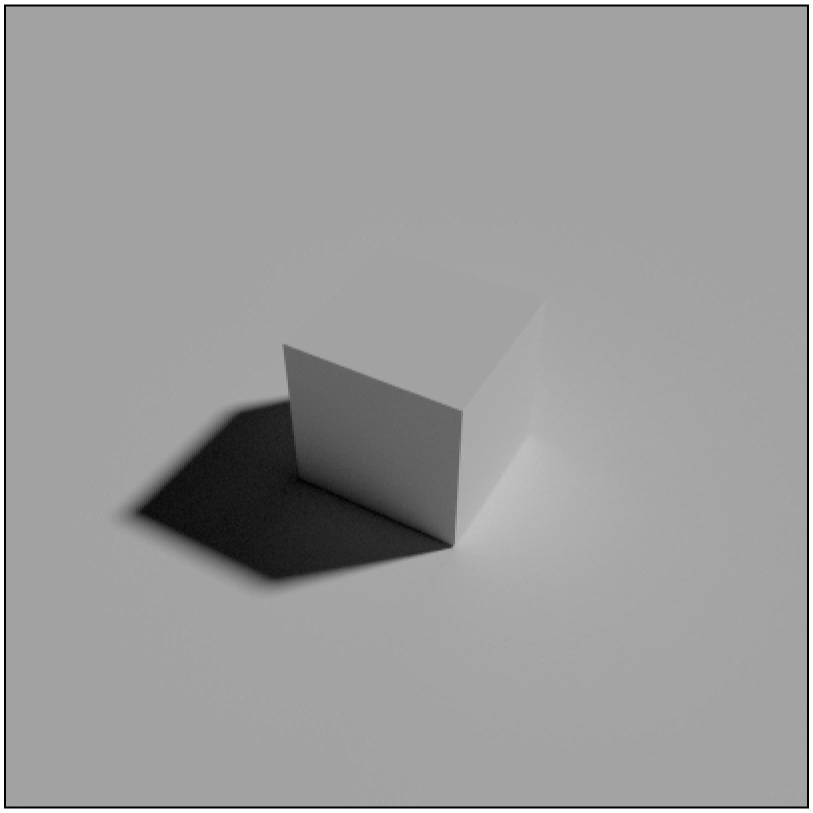
The different types of light are as follows:

- **Point**: As its name implies, it emits lights according to its position. It emits light rays in all directions, but these rays are limited to a certain distance from the center of the light. We often call it a spherical lamp.



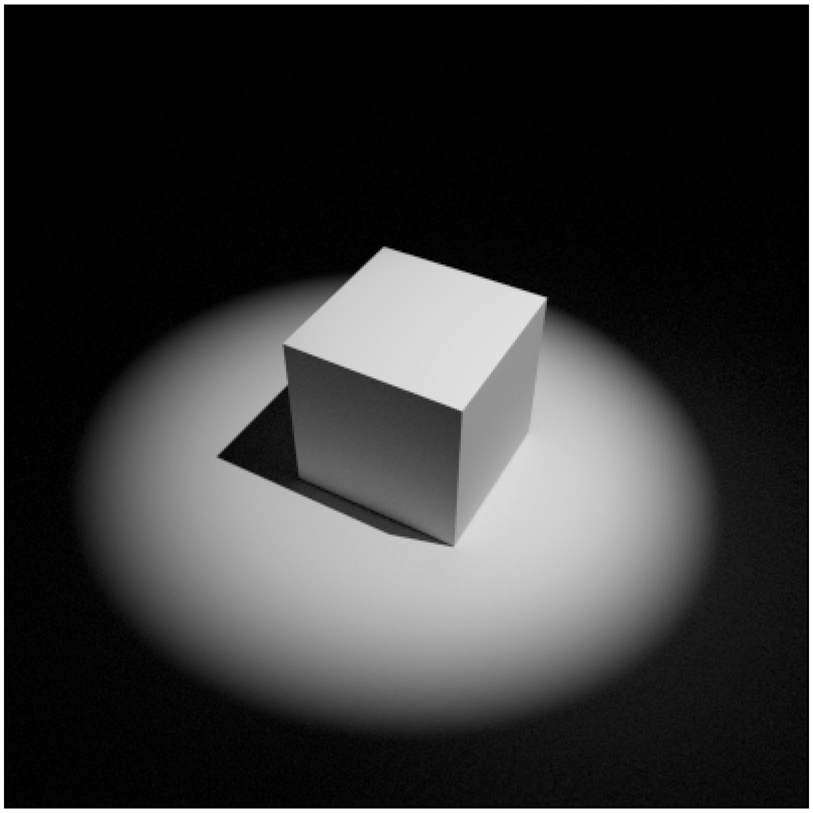
A point light with a strength of 500 and a size of 0.1

- **Sun**: As you can imagine, this type of light represents the way the sun works. As the sun is very far away from earth, we can admit the way we perceive its rays as parallel. So in Blender, the sun produces parallel rays, and we also don't care about its location, but only its orientation matters. With a small size, you can quickly represent the lighting of a bright day. You will mostly use it as a global light as it lights the entire scene. Also, notice that its strength should be less than the point light that we saw previously.



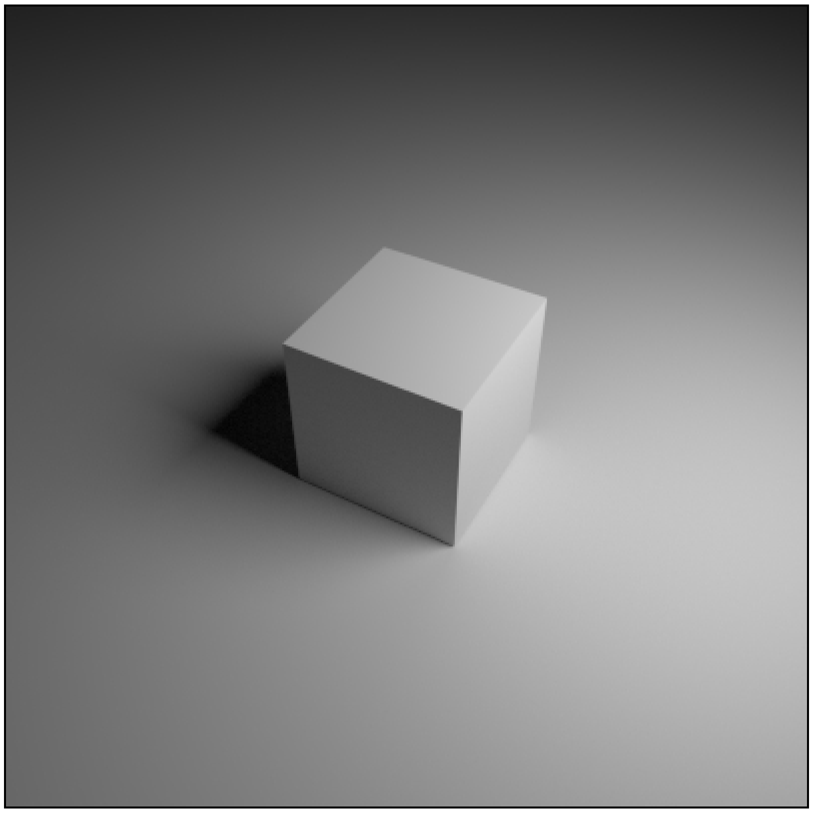
A 45-degree angle on Y and Z sun light with a strength of 2 and a size of 0.05

- **Spot**: The spot light is a conic light. It looks like the lamps used on stage in order to light the show presenter. The lighting that it will produce depends on its location, direction, and the spot shape. You can change its shape in the **Spot Shape** subpanel, and **size** will determine the size of the circle of light influence. The **blend** will define the hardness of the circle shadow. The circle size and the strength of the spot light will depend on its distance from the objects.



A 45-degree angle on a Y spot light with a strength of 5000, a size of 0.5, a shape size of 30 degree, and a blend of 0.8

- **Hemi**: For now, the Hemi type of light is not supported in Cycles. If you use it, it should react like a sun lamp.
- **Area**: This is one of the more common lights. It emits light rays from a plane according to a direction represented by a dotted line. It could be squared or rectangular. Its size, like for the other types, will affect the hardness of the shadows. The strength of the light will also depend on its distance from the objects. With this light, you can achieve a very precise lighting, so we strongly advise you to test this in order to be familiar with the way it reacts.

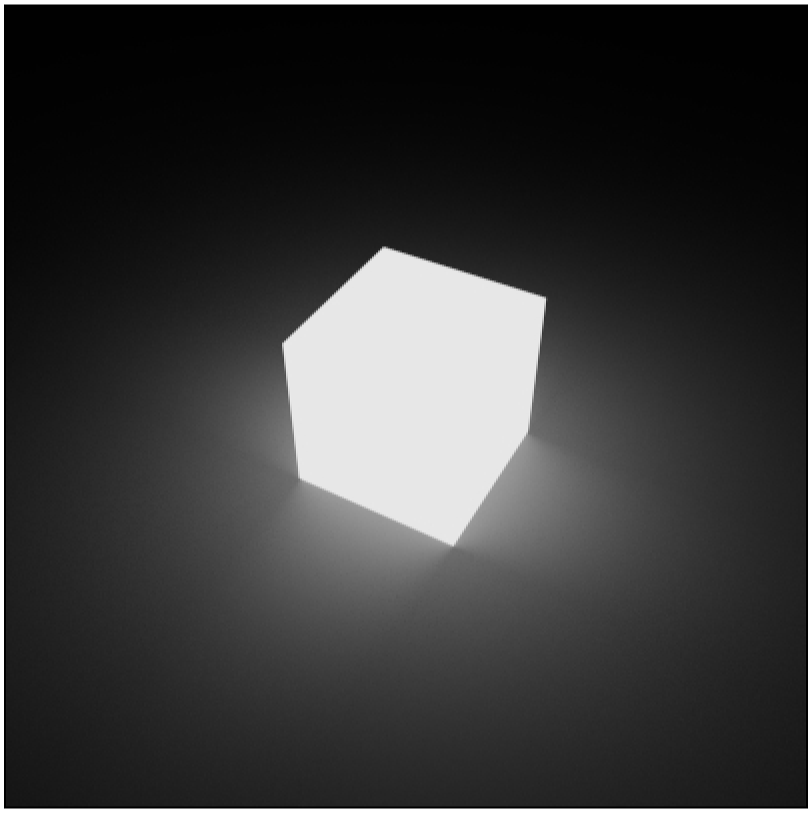


An area light with a strength of 500 and a square size of 5

Another option that many Blender users appreciate is using an emission shader to act as a light on an object (a plane, for instance). The emission shader is, in fact, the base shader of the other types of lights.

1. First let's add a plane.
2. Then, under the **Material** tab of the **Properties** editor, we will add a new material slot.
3. Change the diffuse surface shader from **Diffuse BSDF** to **Emission**.
4. As you may notice, if the plane is in the camera field, you can see it. We don't want this, so go into the **Object** tab of the **Properties** editor, and under the **Ray visibility** subpanel, uncheck **Camera**.

You may find this easier, but, in fact, with this method we lose a lot of control. The main problem we've found with this method is that we can't control the way the rays are emitted, for instance, with the area light. This can be useful when you want visible objects to emit light, but this is not very good for precise lighting.



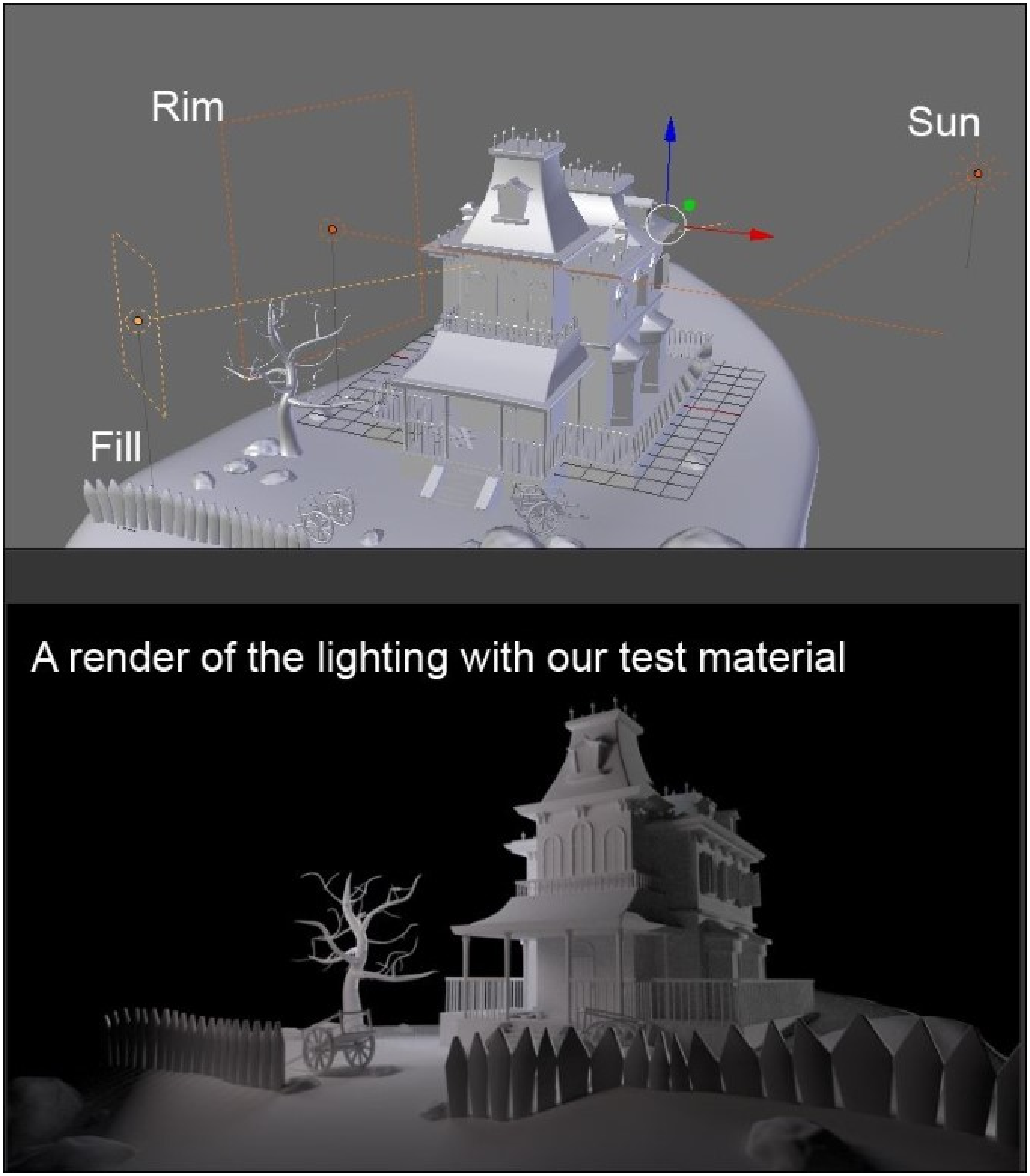
The cube with an emission shader

Lighting our scene

As you can see from the previous part, there are many types of light that we can manipulate in order to achieve a nice lighting effect. But in the case of our haunted house, we are only going to use area and sun lights because the other types of light are often used in specific situations.

1. We will start by opening our haunted house scene and saving it in another name (`HauntedHouseCyles.blend`, for instance).
2. Now we can delete all the lights that we used in the Blender Internal render.
3. While doing a lighting effect, it's a good to have an idea of the volume of your objects with a neutral material. So we will select one of the objects in the scene, remove its existing material, and create a new material in the **Material** tab of the Properties editor. As we did for our testing scene, we will change the color value to **1.0**.
4. Rename the material as `Clay`.
5. Now we will select all the objects in scene (*A*), and reselect the object that is the clay material while pressing *Shift* in order to make it the active object.
6. Press *Ctrl* + *L* and select **Material**. All objects will now share the same material.
7. We can now split our interface in two. One of the 3D views will display the camera point of the view (the *0* numpad key) and will be rendered in real time (*Shift* + *Z*) with a preview sampling of 50 (decrease it if you don't have a powerful computer).

8. The first light to be added is the sun. We will orient it, so it lights the right-hand side of the house. It will be nearly horizontal. Our goal here is to have a dawn lighting. The sun has a size of 5 mm in order to have harsh shadows and a strength of 1.0.
9. The next light that we will add will fill up the front of the house a little bit. It will be an area light that is a slightly tilted down and located on the front left side of the house. We want smooth shadows, so we will change its size to 5 m. Its intensity will be around 400. We can also change its color to be a little bit yellowish.
10. The last light will act as rim light. It will be an area light that comes from the back of the house on the left-hand side. Its size will be 10 m and its strength around 700. We have also tinted it a little bit towards blue.



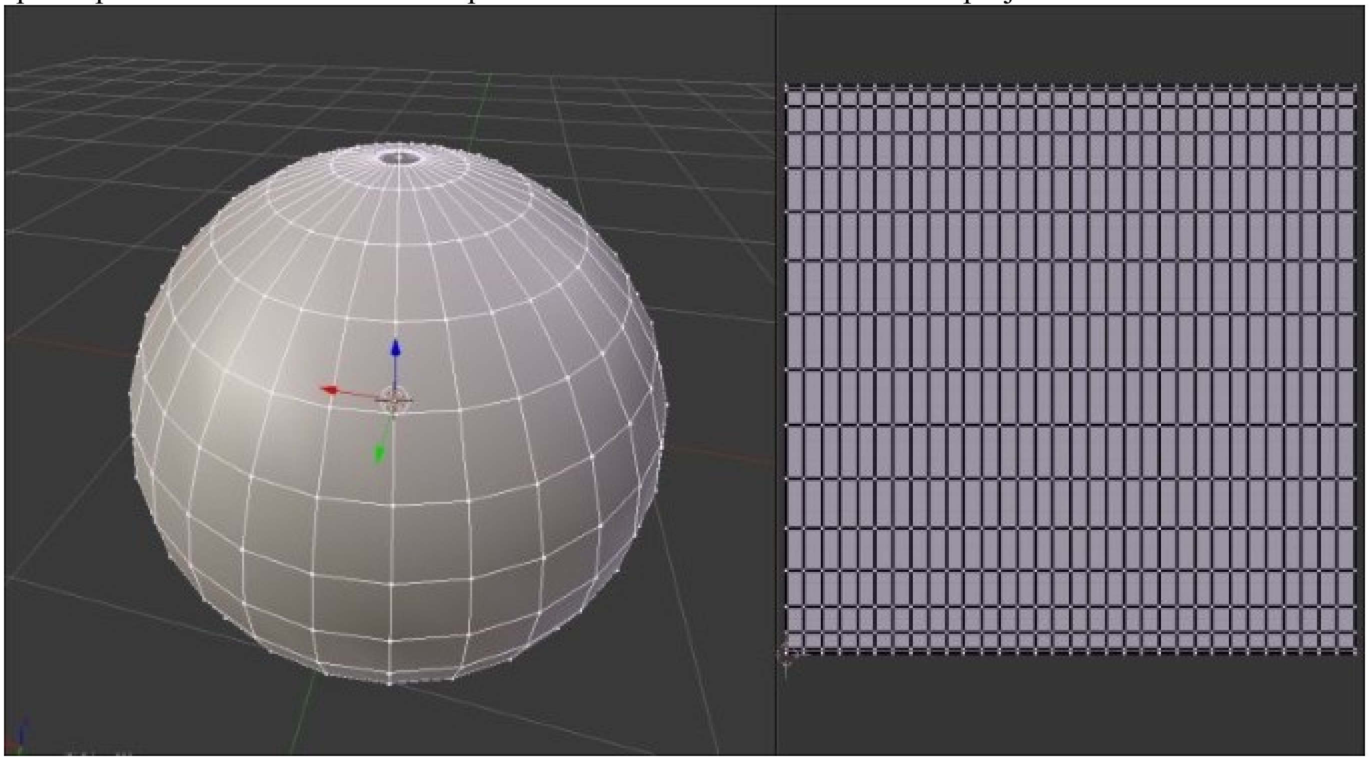
The lighting of the haunted house scene

11. That's all for the basic lights. The light settings are in constant evolution during the whole image creation pipeline, so don't be afraid to change them according to your needs later. Note that we are missing an environment lighting that we are going to set up in the next part of the chapter.

Painting and using an Image Base Lighting

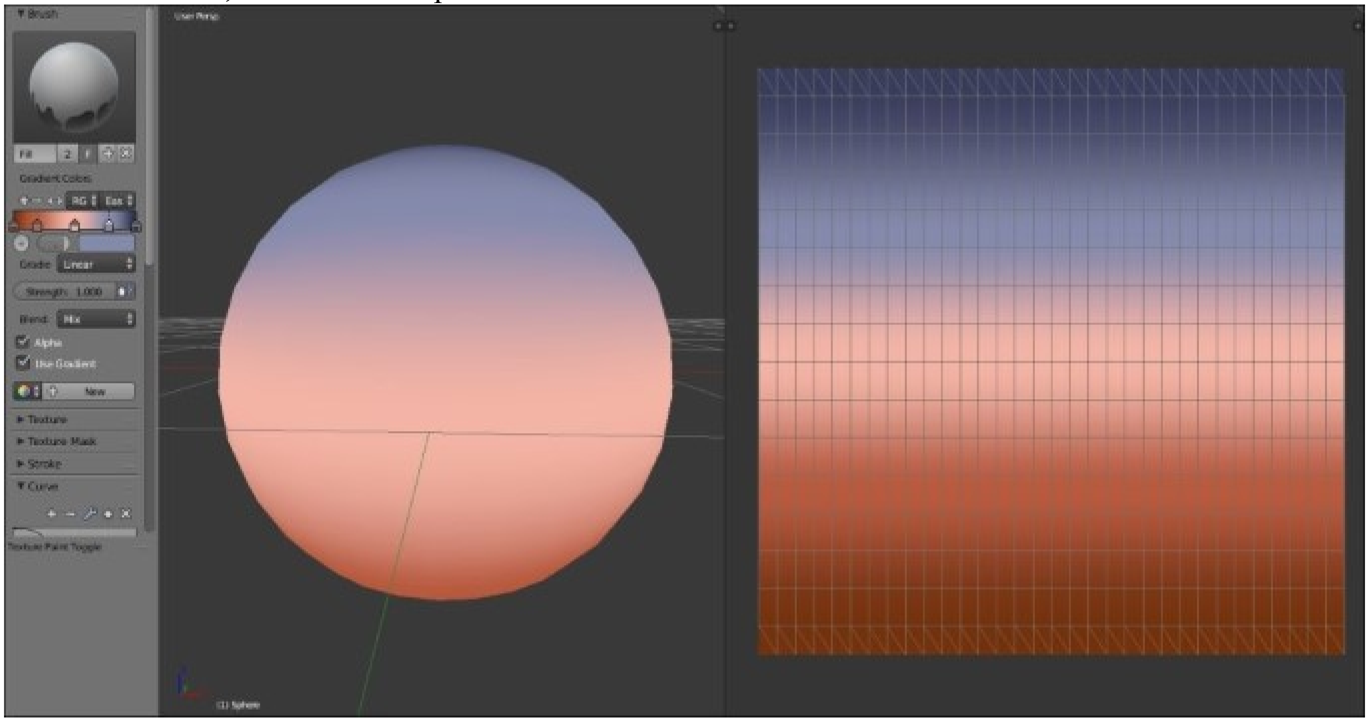
An **Image Base Lighting** (**IBL**) is a very convenient technique that allows us to use the hue and the light intensity of an image to lighten up a 3D scene. This can be a picture of a real place taken with a camera. HDR images provide very realistic results and may be enough to light a 3D scene, but for our haunted house scene, we will paint it directly in Blender with Texture Paint. This technique allows us to do complex lighting in less time and will enrich the lighting that we have prepared previously. We will start by seeing how to prepare the painting phase of a customized IBL:

1. We will open a new scene in Blender.
2. We will split the working environment in half with a **UV/Image Editor** on the right-hand side and a 3D View on the left-hand side.
3. We will add a UV Sphere at the center of the world (*Shift* + *A* and select **Mesh** | **UV Sphere**) on which we will paint the sky.
4. We will delete the vertices at the two poles of the sphere. We will make a scale extrusion (*E* and *S*) of the edges and slightly reposition them again for a well-rounded look. We will obtain a sphere pierced on both ends. It is important to have these holes for the UV projection.



5. We will select all the polygons of the sphere (*A*), then we will apply **Unwrap Cylinder Projection** (*U*). In the **Cylinder Projection** options on the left panel (*T*) of the 3D Viewport, we will change the **Direction** parameter by selecting the **Align to Object** option. This allows us to get straight UVs that occupy the most space on the entire UV Square.

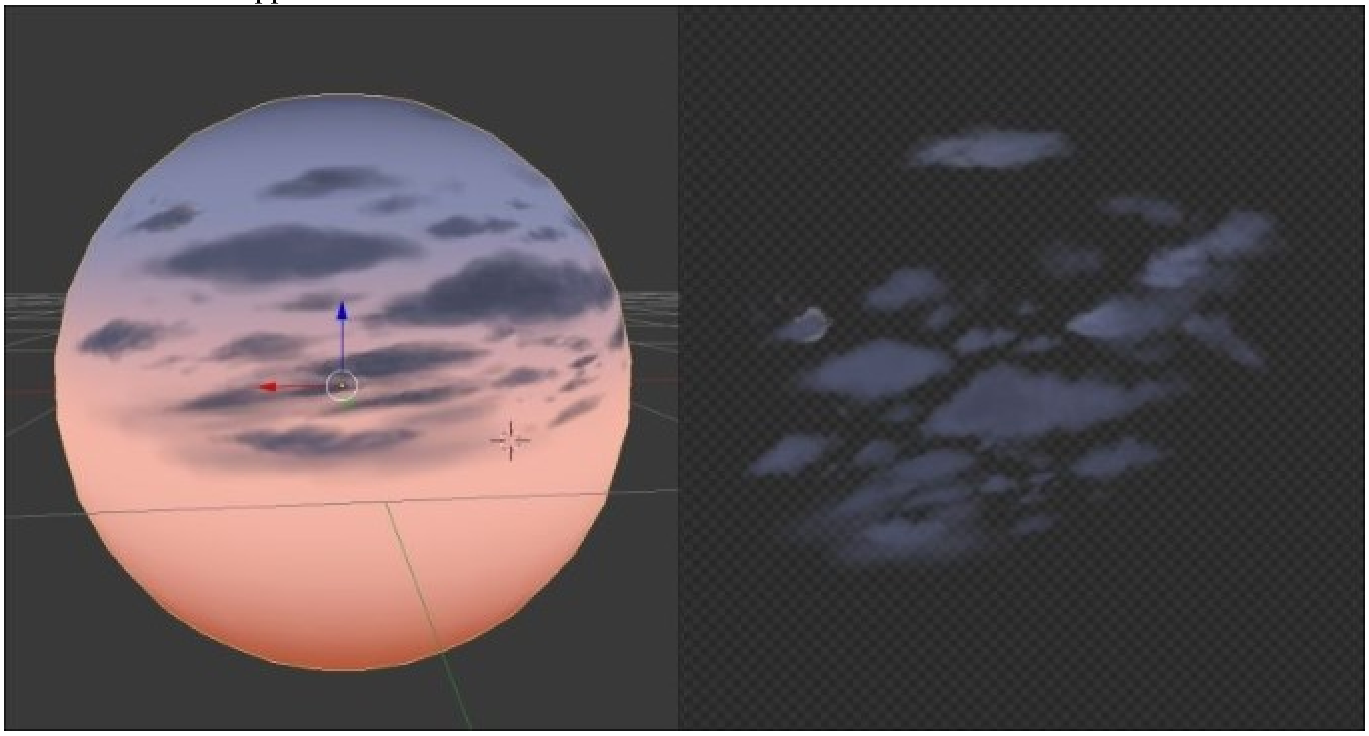
6. Once the UVs are created, we will select the edge loops that form the holes at the poles of the sphere, and we will merge them each in turn to form a complete sphere. This will form triangles in the UV, but it does not matter.
7. We will again select all the polygons of our sphere, and we will then add a new texture by clicking on the **+ New** button in the **UV Image** Editor.
8. In Blender Internal Renderer mode, we will create a new material on which we will place our IBL texture. To better visualize the texture, we will check the **Shadeless** option (**Material** | **Shading** | **Shadeless**).
9. In **Texture Paint,** we can start to paint.



10. We will use Fill Brush with the **Use Gradient** option in order to prepare the gradients of the sky. We will use the following Gradient Colors from left to right. The color marker number **1** on the far left is **R: 0.173**, **G: 0.030**, and **B: 0.003**. The color marker number **2** is located at **0.18**, and its color is **R to 0.481**, **G to 0.101**, and **B to 0.048**. The color marker number **3** is at position **0.5**, and its color is **R to 0.903**, **G to 0.456, and B to 0.375**. The color marker number **4** is located at **0.78**, and its color is **R to 0.232**, **G to 0.254,** and **B to 0.411,** and the last marker at the far right is the color **R: 0.027, G: 0.032,** and **B: 0.085**. We will then apply the gradient upwards on the sphere.
11. On the texture in the **UV/Image** Editor, we can see some black near the poles, which may interfere with the lighting calculation. Thus, in the **Paint Mode**, we will take the nearest color of the black triangles by pressing the *S* shortcut (without clicking), and we will fill the black triangles with **Fill Brush** (without Gradient).
12. When this is finished, we can save this image as **IBL_Sky**.
13. In the **Texture Paint** mode, in the **Slots** tab, we will add a **Diffuse Color** texture that will be transparent this time. For this, we will check the **Alpha** box, and we will change the alpha value

of the **default fill colo**r to 1 in the **Texture Creation** menu. This will allow us to create clouds on another texture while keeping our sky visible.

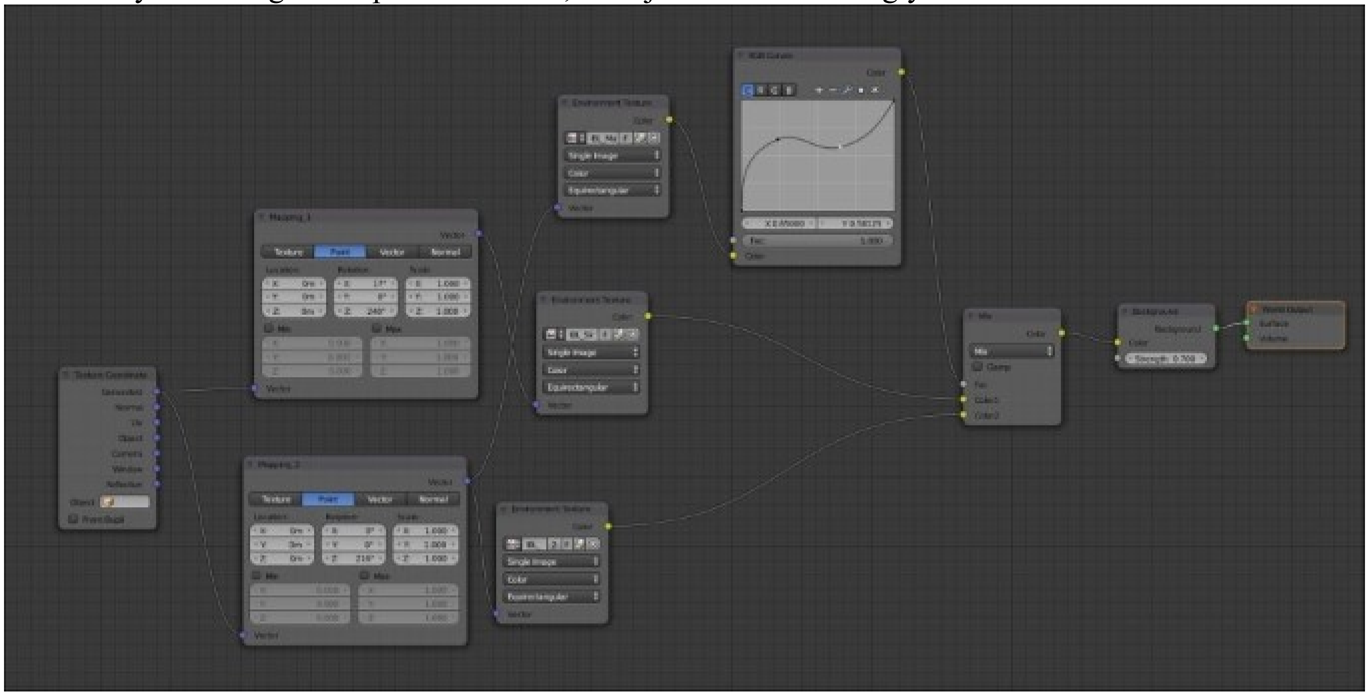
14. With the **TexDraw** brush and the **R: 0.644, G: 0.271, B: 0,420** color, we will draw a few clouds. We must think that there will be only the upper half that will be displayed on the framing of the haunted house. This part will have the greatest importance for the lighting. So we must focus on the upper half of the texture.



15. We will save the image as **IBL_Cloud**.
16. From this cloud texture, we will make a mask that allows us to properly mix the sky and the clouds. For this, we must save our image, with the **BW** (Black and White) option and not in RGBA, by naming it as **IBL_Mas**k.
17. We will then return to the haunted house scene, and in the **Node Editor**, we will click on the **World** icon that is represented by an earth, and we will check the **Use Node** option.
18. We have two nodes that appear: **Background** and **World Output**. We will add an Environment Texture node (press *Shift* + *A* and select **Texture** | **Environment Texture**).
19. We will duplicate the **Environment Texture** node twice (*Shift* + *D*). We will place them one above the other and to the left.
20. In each **Environment Texture** node, we will open the IBL textures created previously.
21. We will add a **Mix RGB** node (press *Shift* + *A* and select **Color** | **Mix RGB**) that will allow us to mix our textures. We will connect the **IBL_Sky** Color Texture Image Output socket to the **Color1** input socket of **Mix Shader**, the **IBL_Cloud** Texture Image Color Output socket to the **Color2** input socket of **Mix Shader,** and the **IBL_Mask** Color Output Socket to the **Fac** input socket. We will keep **Mix** as the Blending mode.
22. We will add a **Mapping** node (press *Shift* + *A* and select **Vector** | **Mapping**) that we will duplicate once, and we will position them one above the other on the left-hand side of the **Environment Textures** node. We will rename them as **Mapping_1** and **Mapping_2**. We will

connect **Mapping_1** to the **IBL_Sky** Texture Image node and **Mapping_2** to the two other **Environment Textures**.

23. We will add a **Texture Coordinate** node (press *Shift* + *A* and navigate to **Input** | **Texture Coordinate**) that we will position at the left. We will connect the **Generated** socket of the **Texture Coordinate** node to the **Vector Input** socket of the two **Mapping** nodes
24. To get a better contrast for the **IBL_Mask**, we will place a **RGB Curves** node (*Shift* + *A* and select **Color** | **RGB Curves**) between the **Environment Texture** node and **Mix RGB**. We will set the following two points: the first point at the **X= 0.24** and **Y= 0.65** position, and the second point at the **X= 0.65** and **Y= 0.58** position.
25. We have all the necessary nodes. To finish, we will need to modify the mapping of the **IBL_Sky** texture. Therefore, we will modify the **X= 6°**, **Y= 23.9°**, and **Z= 0°** rotation parameters. These values vary according to the painted texture, so adjust them accordingly.



To visualize your **Image Base Lighting** (**IBL**) better, you can display it in the viewport. In the **Solid** mode, in the right panel of the 3D Viewport, check the **World Background** option (**Display** | **World Background**).

Creating materials with nodes

It's now time to discover the material creation process with Cycles. In this section, we are going to create the basic shaders that are composed of our previously painted textures. The shaders won't be at their final stage here. Later, we are going to improve them with normal maps.

Creating the materials of the house, the rocks, and the tree

Let's start with the wall shader of the house:

1. We will first select the corresponding object.
2. We are going to duplicate the clay shader that we had added in order to test our lighting in the previous section. As you can see, it is used by 68 objects in the scene. If you click on the 68 button on the right-hand side of the material name in the material tab of the Properties editor, you will duplicate the shader and make it unique. At this time, we can now rename it as `HouseWall`.
3. We are now going to switch to the **Node Editor** in order to have more control on our shader. In fact, we can do everything in the Properties editor, but it will be quite hard to manage with a complex shader. So open a new editor and change it to a **Node Editor**.
4. As you can see, we already have **Diffuse BSDF** plugged into the **Surface** input of the **Material Output** node.
5. A diffuse shader has no shine on it. It looks flat. In real life, every surface is at least a little specular, so we are going to mix our diffuse shader with another shader that will bring us the shiny effect. To do this, we will first add a **Glossy BSDF** shader (press *Shift* + *A* and select **Shader** | **Glossy BSDF**) and place it under the diffuse shader. Don't connect it for the moment.
6. In order to mix the two shaders together, we will use a **Mix Shader** node (*Shift* + *A* and select **Shader** | **Mix Shader**). As you can see, this node has two shader inputs. Plug the BSDF output (green dot) of the diffuse shader to the first shader input of the **Mix** shader, and the BSDF output of the Glossy shader to the second shader input of the **Mix** shader. Now plug the **Mix** shader output to the surface input of the **Material Output** node. As you can see, both shaders have been mixed together. You can now use the **Factor** slider in order to choose which one is predominating. If you put a value of **0**, you will only use the shader connected to the first input (the diffuse), and if you put a value of 1, you will only use the shader connected to the second input (the glossy one).
7. The blend between these shaders is not going to look right with any value, so we are going to connect a **Fresnel** node to the **fac** input (press *Shift* + *A* and select **Input** | **Fresnel**).

   ### Note

   About the Fac input

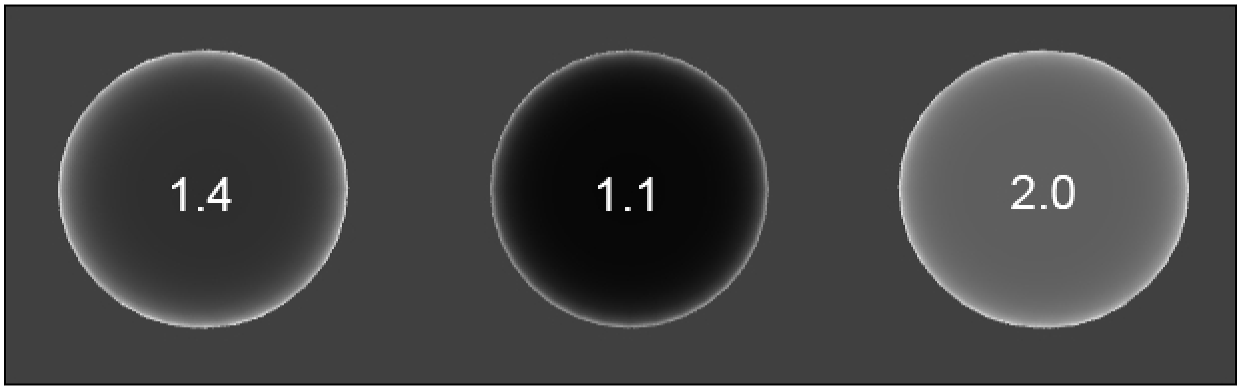
   The role of the fac input is to control how both shaders will be mixed. Usually, the **Fac** input needs to be fed with black and white information where the amount of black tells us how much the first shader will be used, and the amount of white tells us how much the second shader will be used for the final output.

8. We can now change the value of the Fresnel to **1.4**.

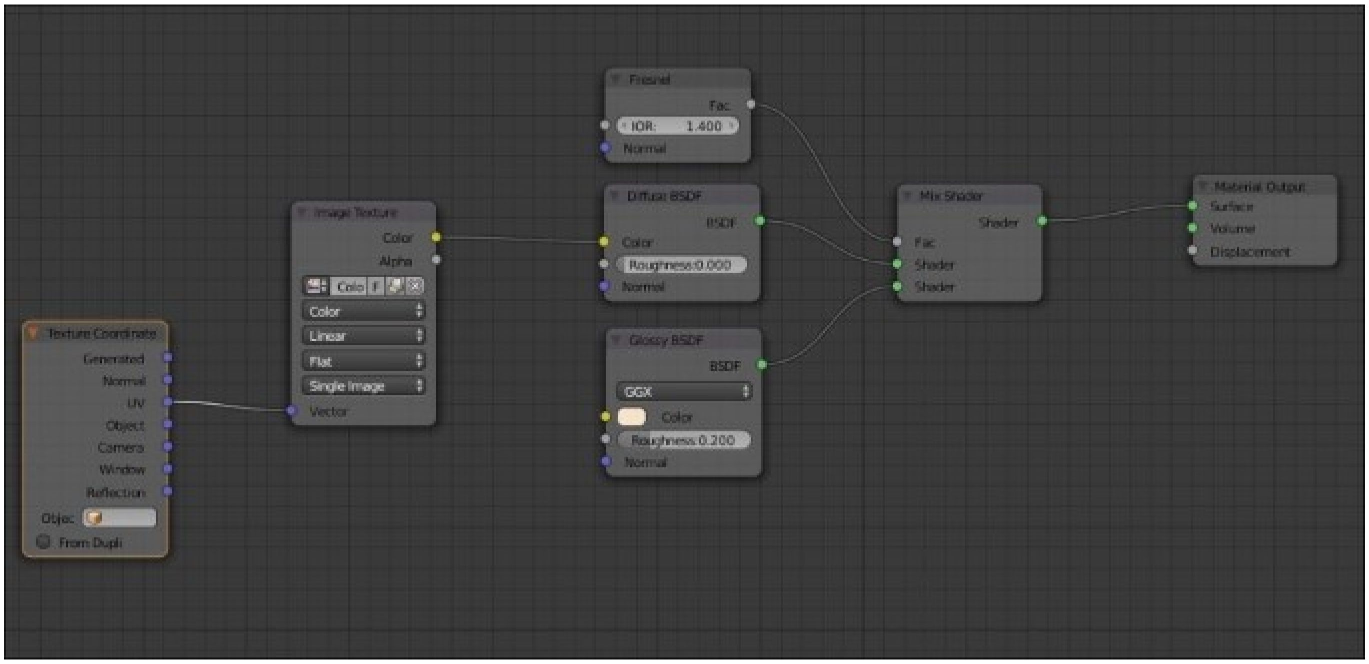
Note

About the Fresnel

The Fresnel node will produce a black and white texture according to the volume of the geometry. It will be calculated according to the light ray's incidence. We usually use a Fresnel node in order to catch the highlights better. You can use a pretty interesting add-on called node wrangler that allows you to quickly see the result of each node without shadows. In order to use it, right-click on the node you want to see while pressing *Ctrl* and *Shift*.



9. We will now change the glossy color to a yellowish tint. Don't forget to turn on real-time shading in order to have a preview of what this will look like in the render. You can also drag a rectangle to the zone you want to preview with the *Shift* + *B* shortcut while being in camera view. If you want to remove the rectangle zone, drag a new zone to the outer zone of the camera in the camera view.



IOR: 1.400

Roughness:0.000

GGX

Roughness 0.200

Color

Linear

Flat

Single Image

The base of our wall shader

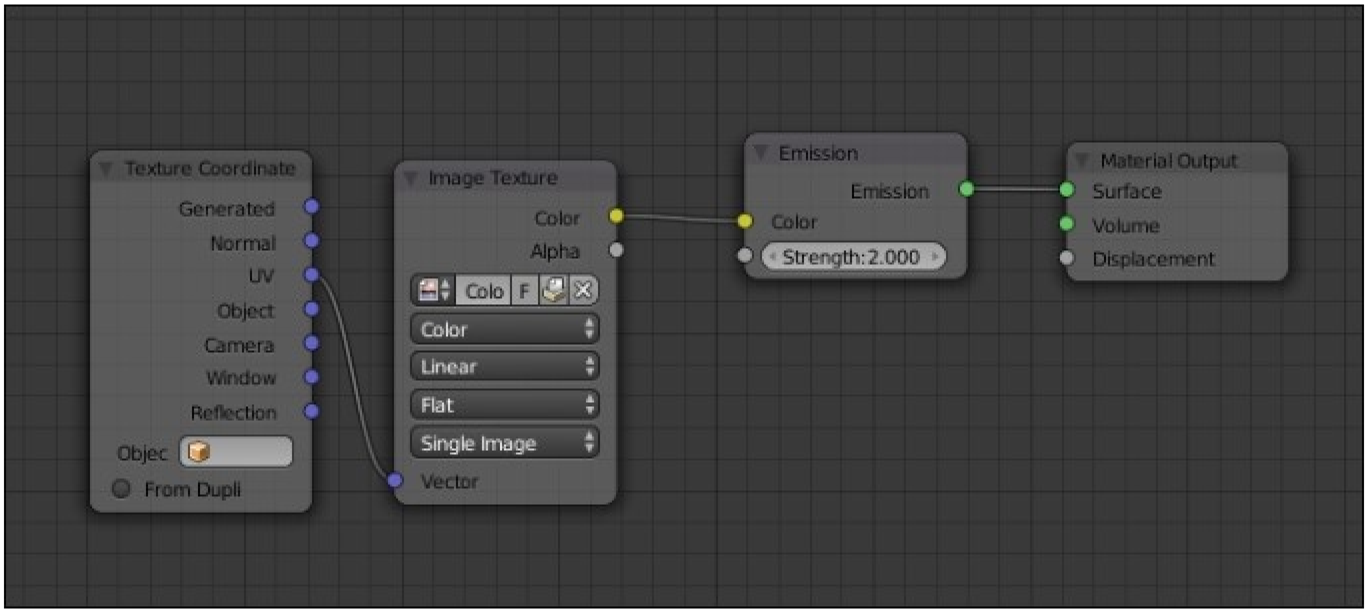
The last thing we will do with this material for now is plug it into our texture:

1. We will add a new Image Texture node (press *Shift* + *A* and select **Texture** | **Image Texture**).
2. We will need to connect the **Color** output of this node to the **Diffuse BSDF** color input.
3. It's also a good idea to add a **Texture Coordinate** (press *Shift* + *A* and select **Input** | **Texture Coordinate**) node and plug the UV slot to the **Vector** input of the **Image Texture** node.

By default, the Vector inputs are set to be UV, but with this node, we can clearly see the mapping method used for the textures.

1. We can now select one of the roofs and change its clay shader to the one we created because the roof shader will be nearly the same. Now, in order to break the link between the roof and the wall shader, we can press the button with the number of objects that share the same material in order to copy the material.
2. We will rename this shader to `HouseRoof1`.
3. The only thing we need to do for now is to change the texture of the Image Texture node to the corresponding roof texture.
4. We can now select all the objects that need to share the same material (the other blue roofs), and finally, we select the roof that has the shader that we want to share, press *Ctrl* + *L*, and select **Material**.
5. We will now repeat the process of creating a new material by copying it from the previous one, changing its name and texture information, and linking it to its corresponding objects.

We will now have a shader on the rocks, the tree, and all the different objects that make up the house. The only shader that will be different is the one on the top window. It needs to emit light. In order to do this, we will copy the previous material, delete the diffuse, glossy, and mix shader (*X* or *Delete*), replace them with an **Emission** shader (press *Shift* + *A* and select **Shader** | **Emission**), and plug the window texture to this. Now the top light is going to emit light! You can tweak the emission value if you want more light. As you can see, we need to do this for the other windows as well, but in the case of the other windows, we can't do this simply because they are not planes and their color information is located on the wall texture. That's why we need to paint a mask.

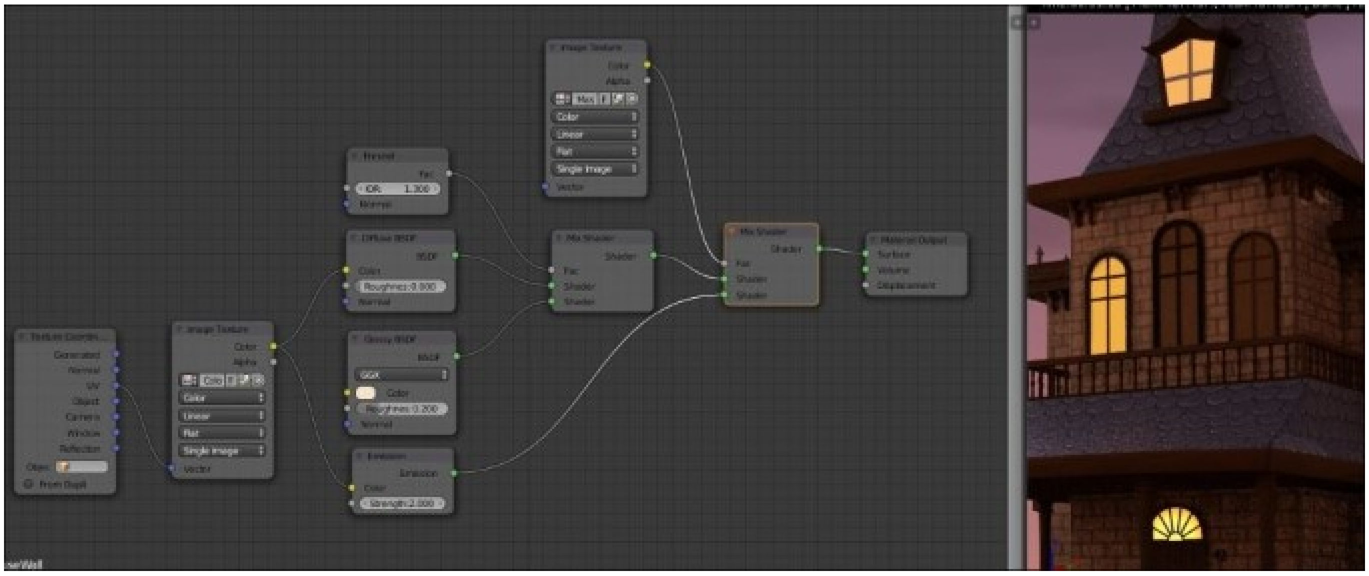


The top window shader

Adding a mask for the windows

We are now going to improve our wall material by creating a mask that will separate the windows that are shining from the rest. These windows are going to be painted white and the rest will be black. So when we plug the mask in the **Fac** input of a Mix material node, we will be able to choose an emission shader for the white parts.

1. We are going to paint our map in the **Blender Internal** context. Note that we can actually use the **Texture Paint** mode while being in Cycles, but this implies that we add and select a texture node that uses the texture that we want to paint.
2. So we will select the wall object, and in the **UV Image** editor, we will create a new 1024 x 1014 black texture. Usually, masks don't need large resolutions.
3. Now in pure white paint the windows that shine (the light yellow ones on the color map).
4. Let's go back to our wall material and add a mix shader just before the output node. If you want to save time, you can drag the node on the connection line of the previous **Mix** shader and the **Output** node. This will automatically do the connections for you.
5. The first shader input is already used by our old **Mix** shader. Now we are going to add a new **Texture** node with our mask and plug its output to the **Fac** input of our new **Mix** shader.
6. The second shader input will be fed with an **Emission** shader (press *Shift* + *A* and select **Shader** | **Emission**). In the **Color** input of this node, we will plug our color map (the same as in **Diffuse** shader).



The wall material with the mask on the left-hand side and the result in the real-time rendered 3D view.

7. Now we can increase the strength of the Emission shader to 2.0. As you can see, now our shining windows (and only them) emit light!

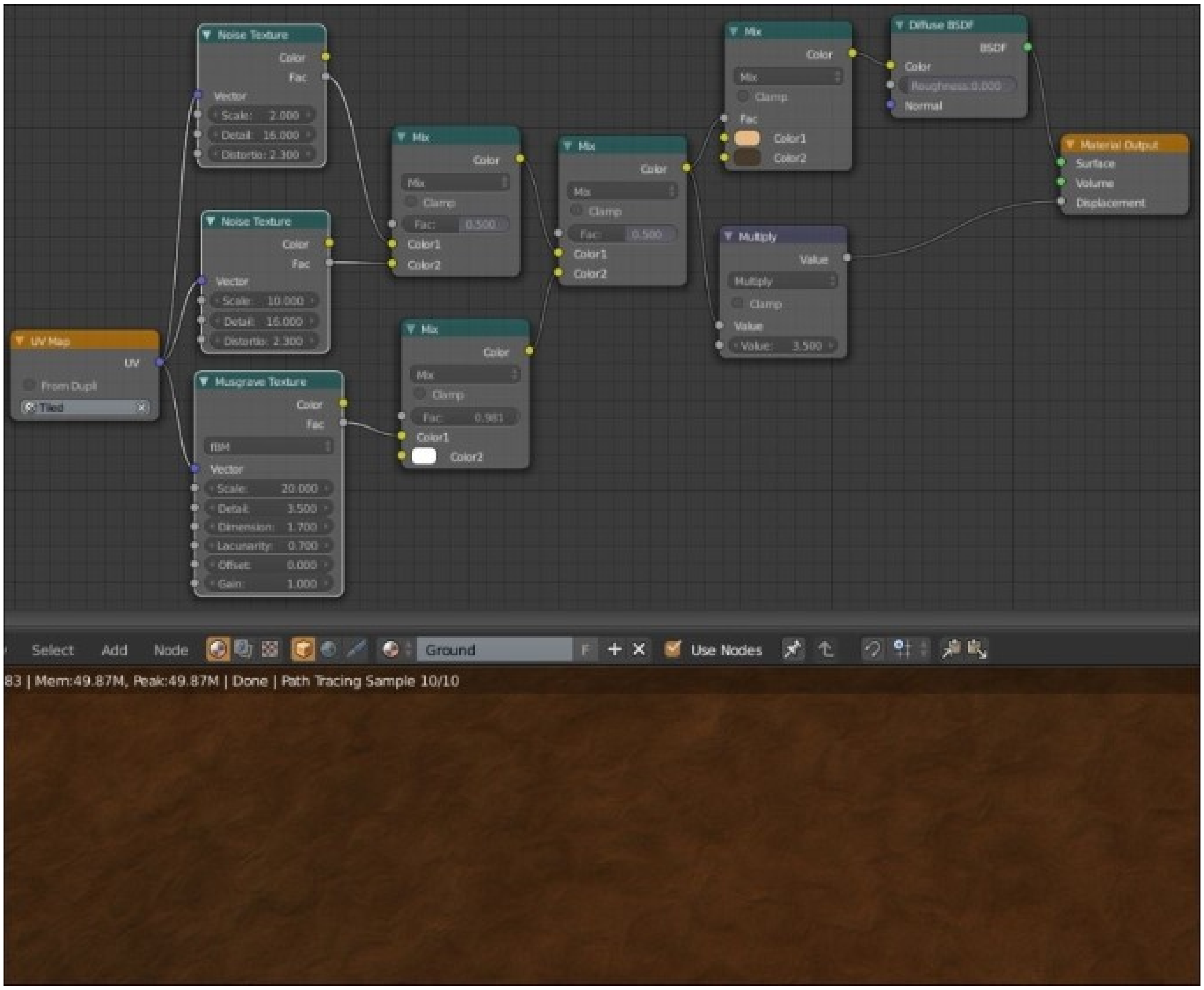
Using procedural textures

One thing that could be very interesting when creating materials is generating their textures procedurally. In this render, we are going to replace the hand-painted ground by a procedural material. This is done as follows:

1. We will select the cliff and create a new material for it.
2. The next thing is to add a **Diffuse BSDF** node. The color input of this diffuse material will be fed with a mix of procedural textures.
3. Let's add a Noise texture node (*Shift* + *A* and select **Textures** | **Noise**) and duplicate it (*Shift* + *D*). The first one will have a scale of 2.0, and the second one will have a scale of 10.0. Our goal is to have a mix of both levels of noise. Both of them will use the Tiled UV layer, so add a **UV Map** node (press *Shift* + *A* and select **Input** | **UV Map**), select the correct map, and feed the **Vector** input of the noise textures with the **UV Map** node output.
4. As these nodes are textures and not shaders, we aren't going to use the **Mix** Shader node but the **MixRGB** node instead. So we will add one of these nodes (press *Shift* + *A* and select **Color** | **MixRGB**), and feed the inputs with their noise **Fac** output. Don't use the color output as we want a black and white mix here. Remember that you can always test your results with the Node Wrangler add-on (press *Shift* + *Ctrl* and right-click on any node).
5. We are now going to mix this result with a **Musgrave** node (press *Shift* + *A* and select **Texture** | **Musgrave**). We also need the Tiled UV for the vector input. We will set the **Scale** to **20.0**, the **Detail** to **3.5**, and the **Dimension** to **1.7**. Its effect will be pronounced in the final result, so we are going to mix it with white. To do this, we will add the **MixRGB** node and plug the first **color** input with the **Fac** output of the **Musgrave** Texture. The second color slot can be changed

to white. We are going to change the **Fac** slider of the **MixRBG** node to 0.98; this will make Musgrave very subtle.

6. We can now mix our noises and the **Musgrave** results together with one more **MixRGB** node.
7. If we plug this directly to the color input of the diffuse, we will get a black and white result. In order to introduce color, we will need another **MixRGB** node, but instead of feeding the color inputs we are going to plug our texture in the **Fac** input and choose two brownish colors. Now we can plug the result to the color input of the Diffuse shader.
8. Lastly, we can plug the black and white texture to the displacement input of the Material output node. In order to raise the displacement effect, we can place a **Math** node in-between (press *Shift* + *A* and select **Converter** | **Math**). We can change its operation to **Multiply** and use a **3.5** value.



Noise Texture: Scale: 2.000, Detail: 16.000, Distortio: 2.300

Noise Texture: Scale: 10.000, Detail: 16.000, Distortio: 2.300

Musgrave Texture: fBM, Scale: 20.000, Detail: 3.500, Dimension: 1.700, Lacunarity: 0.700, Offset: 0.000, Gain: 1.000

Mix: Mix, Fac: 0.500

Mix: Mix, Fac: 0.500

Mix: Mix, Fac: 0.981

Multiply: Multiply, Value: 3.500

Diffuse BSDF: Roughness: 0.000

```
83 | Mem:49.87M, Peak:49.87M | Done | Path Tracing Sample 10/10
```

The ground material with the procedural texture made with a noise and Musgrave combination on the left-hand side and, the result in the real time rendered 3D view.

Making and applying normal maps in Cycles

As we saw it previously with the alien character, normal map allows us to simulate a relief on a 3D mesh very efficiently. It would be good to generate a few of them in order to add some relief to our scene. We will explore a method to easily generate normal maps from tiled, hand-painted textures:

1. We will open a new Blender scene.
2. We will delete the cube (*X*) and then we will add a plane in the middle of the scene (press *Shift* + *A* and select **Mesh** | **Plane**) that we name as **Plane-1**.
3. We will split the screen in two parts to open the **UV/Image Editor** at the right-hand side.
4. We will add a **Multires** modifier to our plane and a **Displace** modifier. We will place the Multires modifier above the Displace modifier.
5. In the Texture tab, we will create a new texture by pressing **New,** and then we will load the tiled texture of wood, that is, **WoodTilePlank.png**.
6. We will check that the texture is loaded in the **Displace** modifier.
7. In the **Multires** modifier, we will check the **Simple** mode, and we will click on **subdivide** until we get to **level 8**. The more the mesh is subdivided, the more the Displace effect is accurate, but we must pay attention to the RAM of the computer.
8. We will modify **Strength** to **0.25**. This can vary depending on the texture. We must avoid important deformations on the mesh.
9. We will duplicate the plane (*Shift* + *D*), and then we will delete the modifiers of this new plane that we name as **Plane-2**.
10. We will select all the faces (**A**) of **Plane-2** in the **Edit Mode**, and we will do an unwrap (**U** | **Unwrap**) with a square shape taking all the UV surfaces. Then, we will add a new image. In the **UVMap Editor**, we will click on **New Image** (**Image** | **New Image**). We will name it as **WoodTilePlank_NM.png**.
11. We will move **Plane-2** to the same height as **Plane-1**.
12. Enter the top view (the *7* numpad key) for a better view of the mesh.

If the Displace doesn't give exactly the effect we want, we can make a few modifications with the **Sculpt** mode after applying the displace modifier by pushing on **Apply**. This is what we will do with the rock and the tile roof textures.

We can now bake a normal map as follows:

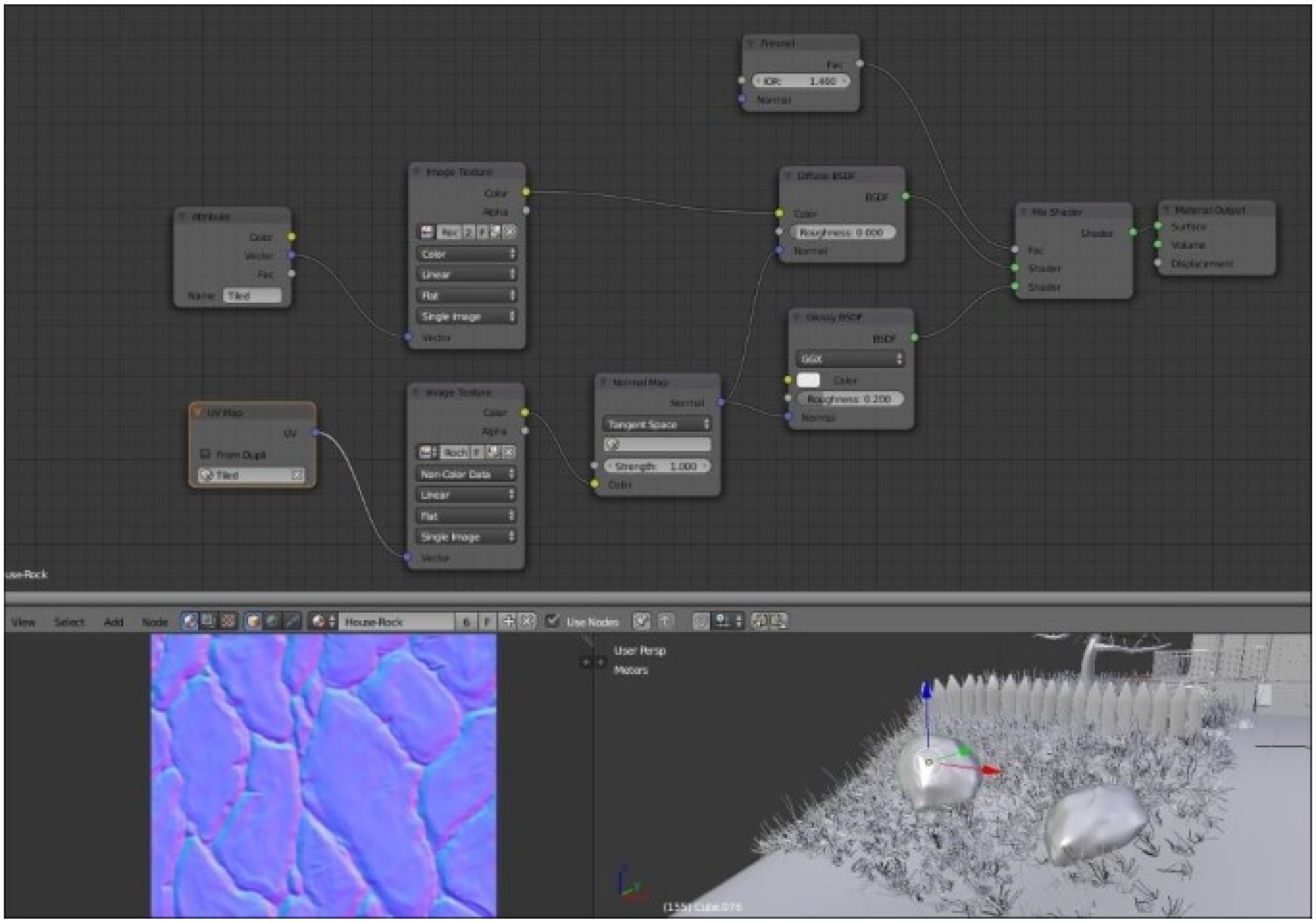
1. We will need to click on the **Smooth Shading** button, otherwise we will see the polygons on our bake.
2. Then, we will have to first select **Plane-1 (RMB)** and then **Plane-2** (press *Shift* and the RMB). This becomes the active object.
3. Now, in the **Properties** editor, under the **Render** section, we will expand the **Bake** subpanel.
4. The first option to choose is what type of map (or texture) we want to bake. So, in the **Bake Mode** drop-down menu, we will select **Normal Map.**
5. The next thing we'll have to check is the **Selected to Active** option that tells Blender to bake from the sculpture to the active object (our low poly plane).
6. Now you can click the **Bake** button. Don't forget to save your map (select **Image** | **Save As Image** or press *F3*), or it will be lost!

We will use the same process of Normal Map creation for every tiled texture. Once this is done, we can return to our shaders and apply our normal maps.

We will start with the House-Rock shader, which is very simple for the moment:

1. We will add a **Glossy** node (*Shift* + *A)* and navigate to **Shader** | **Glossy BSDF**, which we will mix with **diffuse** using a **Mix Shader** (*Shift* + *A* and select **Shader** | **Mix Shader**). To better visualize the normal maps, we will need glossiness.
2. We will add **Fresnel** (*Shift* + *A* and navigate to **Input** | **Fresnel**) that we will position in the **FAC** input socket of the **Mix Shader** node. We will put a **IOR** value of **1.4**.
3. We will add a **Normal Map** node (*Shift* + *A* and select **Vector** | **Normal Map**) with a **Strength** value of **1.0** and connect its output to the Normal input socket of the Diffuse and Glossy shaders.
4. We will duplicate an Image Texture node (*Shift* + *D*), and we will open the normal map texture named Roch-Tilling-NM.png file. We must switch the **Color Data** option of this second Image Texture to **Non-Color Data**. The data should not be interpreted as color data but as normal direction data.
5. We will add a UV Map node (*Shift* + *A)* and navigate to **Input** | **Glossy UV Map**. Then we will change the UVLayer to **Tiled**. You will recall that the rocks have two UV layers, so we must select one.

We can apply this process for almost every shader using hand-painted tiled textures, except the brick walls in our case. It is a special case that requires that we bake the normal map on a larger map with a little modification of painting in order to hide the bricks behind the windows.



The normal map of the rock (low left corner) and its material in the nodal editor (top)

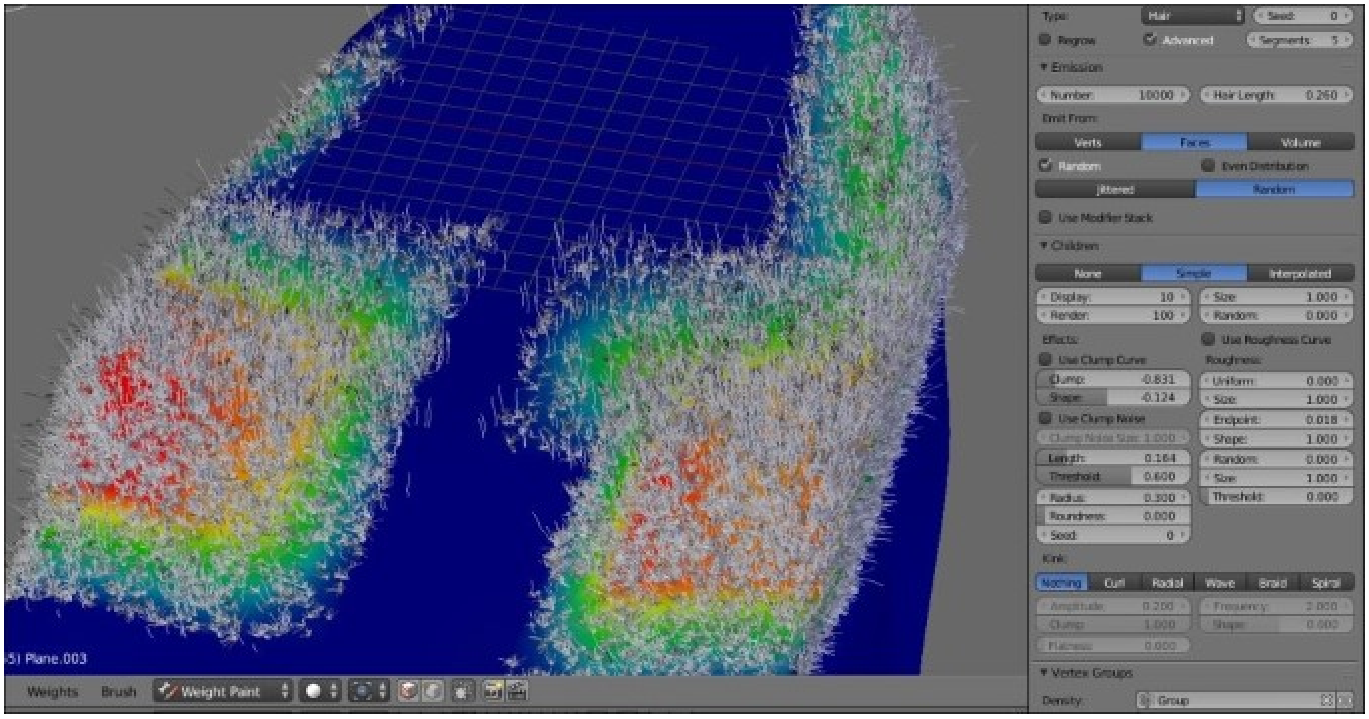
Creating realistic grass

In this section, you will see how we can create realistic grass in place of the actual grass planes. In order to create our realistic grass, we are going to use a hair particle system.

Generating the grass with particles

A particle system will be used here in order to generate the grass strand without modeling and placing them by hand:

1. We will first select the cliff and isolate it (/ **Numpad**). Then, we can go into the **Particle** tab of the **Properties** editor and add a new particle system. This will add a new modifier in the stack, but we can only control it here.
2. We will now have to change the type from **Emitter** to **Hair**. We can activate the **Advance** tab.
3. In the **Emission** subpanel, we can change the **Number** value to **10000**, which corresponds to the number of grass strands that we will have. We will also emit the particle from the faces in a **Random** manner. We can also change the **Hair Length** to **0.26**.
4. In the **Physics** subpanel, we can change the **Brownian** value to 0.120 in order to add more randomness to the grass.
5. In the **Children** subpanel, we can set the amount of strands we want to spawn around the main guides. In our case, we will activate the Simple mode and set **10** children for the **preview** (in the viewport) and **100** for the **render**. In the effect section, we can change the **Clump** value to **-0.831** so that the children start near the base of the guide strand and are sprayed out near the tip. We can also change the **Shape** value to **-0.124** to shrink the children in the middle a little. We will also change the **Endpoint** value to 0.018 to add more randomness.
6. We will now paint a vertex group with the **Weight paint** tool in order to choose where we want grass on the cliff. For instance, we don't want strands under the house. To do this, we will switch from **Object mode** to **Weight Paint** mode while the cliff is selected. As you can see, you have brushes as in the **Sculpt** mode. We can use the **Add** brush to add weight and the **Subtract** brush to remove weight. **Red** means that it will be full of grass, **Blue** means you won't have grass. Now, we can choose the vertex group that we have painted in the **Density** field of the **Vertex Groups** subpanel in the Particle System settings.
7. Now we can add a new particle system in order to add long grass. To do this, we will add a new slot in the particle settings. We will also copy the settings from the previous particle system but unlink them (the button with the number on the right-hand side of the name). We can lower the number of particles to **1000** and change the **Hair Length** to **1.120**. The number of children will be **5** in the **Simple** mode.



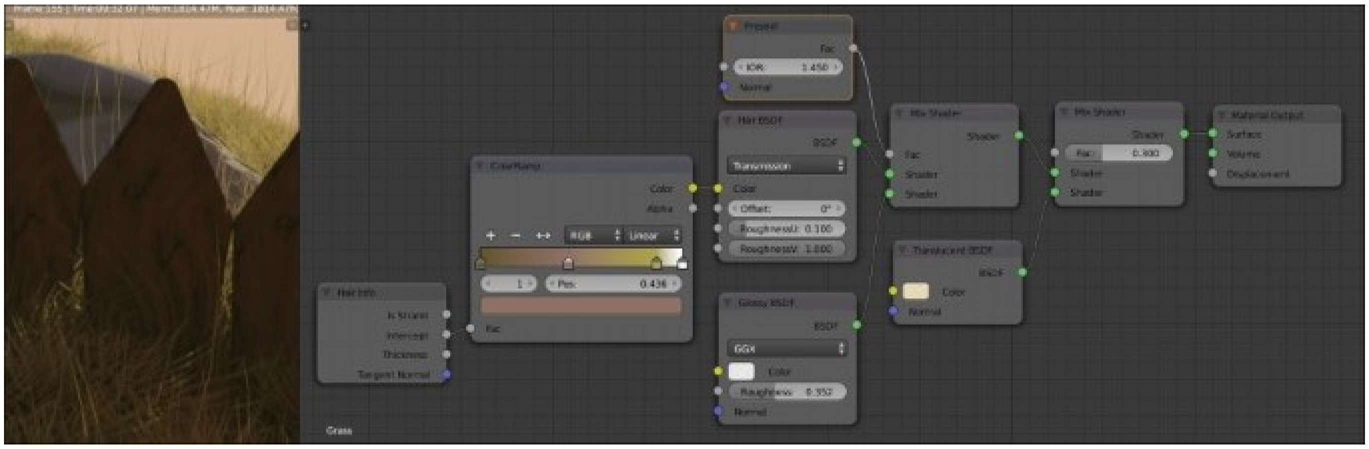
```
Type: Hair                    Seed: 0
Regrow    Advanced            Segments: 5
▼ Emission
Number: 10000                 Hair Length: 0.260
Emit From:
Verts | Faces | Volume
Random                        Even Distribution
Jittered | Random
Use Modifier Stack
▼ Children
None | Simple | Interpolated
Display: 10                   Size: 1.000
Render: 100                   Random: 0.000
Effects:                      Use Roughness Curve
Use Clump Curve               Roughness:
Clump: -0.831                 Uniform: 0.000
Shape: -0.124                 Size: 1.000
Use Clump Noise               Endpoint: 0.018
Clump Noise Size: 1.000       Shape: 1.000
Length: 0.164                 Random: 0.000
Threshold: 0.600              Size: 1.000
Radius: 0.300                 Threshold: 0.000
Roundness: 0.000
Seed: 0
Kink:
Nothing | Curl | Radial | Wave | Braid | Spiral
Amplitude: 0.200              Frequency: 2.000
Clump: 1.000                  Shape: 0.000
Flatness: 0.000
▼ Vertex Groups
Density: Group
```

The settings of the grass

Creating the grass shader

Now the last thing that we will create is the grass material:

1. The first thing we will need to do is change the **Cycles Hair Settings** in the **Particle Systems** tab. We will change **Root** of the strand to **0.20** and set **Tip** to **0.0**.
2. We can now add a new material slot for the cliff and rename it as Grass. In the particle settings, we will change the material to **Grass** under the **Render** subpanel.
3. We will now select the grass shader and open the **Node** editor. The first node that we will add is the **Hair BSDF** shader (press *Shift* + *A* and select **Shader** | **Hair BSDF**) and change it from **Reflection** to **Transmission**. We will mix it using **Mix Shader** with a **Glossy BSDF** shader. We will change **Glossy Roughness** to **0.352** so that the glossiness is more diffuse.
4. Next, we will have to plug a **Fresnel** node to the **Fac** input of **Mix Shader**.
5. For the color of **Hair BSDF**, we will add a **Color Ramp** node (press *Shift* + *A* and select **Converter** | **Color Ramp**). For its **Fac** input, we will add a **Hair Info** node (press *Shift* + *A* and select **Input** | **Hair Info**) and choose the **Intercept** output. This will enable us to set the different colors along the strand. We will do a gradient that starts from a brownish color (to the left) to a desaturated green (to the right). Usually, the tip of the grass strand is white, so we will add a small amount of white on the far right of the color ramp.



The grass shader (to the right) and the result (to the left)

6. We will also mix the result of our Mix shader with a **Translucent BSDF** shader (press *Shift* + *A* and select **Shader Translucent**). Indeed, the grass is very translucent. We will change its color to a desaturated yellow and change the **Fac** value of the shader to **0.3**. We can finally plug our latest **Mix** shader to the surface input of the **Material** output node.

Baking textures in Cycles

Cycles allow us to bake textures as does Blender Internal, but there are some differences between the two render engines.

Cycles versus Blender Internal

As we have seen previously, texture baking in Blender Internal can be very efficient to produce normal maps, ambient occlusion, color textures, and many other kinds of maps that we won't cover here. All of this in a very short time. So you might wonder why it is interesting to bake in Cycles.

Cycles is a ray tracer render engine based on physical parameters with global illumination. It is then possible to get some very realistic renders in a much more efficient manner. Baking in Cycles allows us, for example, to calculate a few heavy special effects only once, such as caustics. When a render is baked on a texture, you can visualize the effect in real time. This can be very useful if you want to change the frame and make several renders. In this way, it is possible to create a realistic environment in real time.

However, in the context of a video game, if you have many dynamic assets you must pay attention. You could be limited by fixed lighting. Even though baking in Cycles can be very interesting, it has some faults. Doing a good baking without noise requires the same settings as a normal render, so you do need a high sampling value, which greatly increases rendering time compared to Blender Internal.

Note

For more information, you can have a look at the official Blender Reference manual at this address:

http://www.blender.org/manual/render/cycles/baking.html

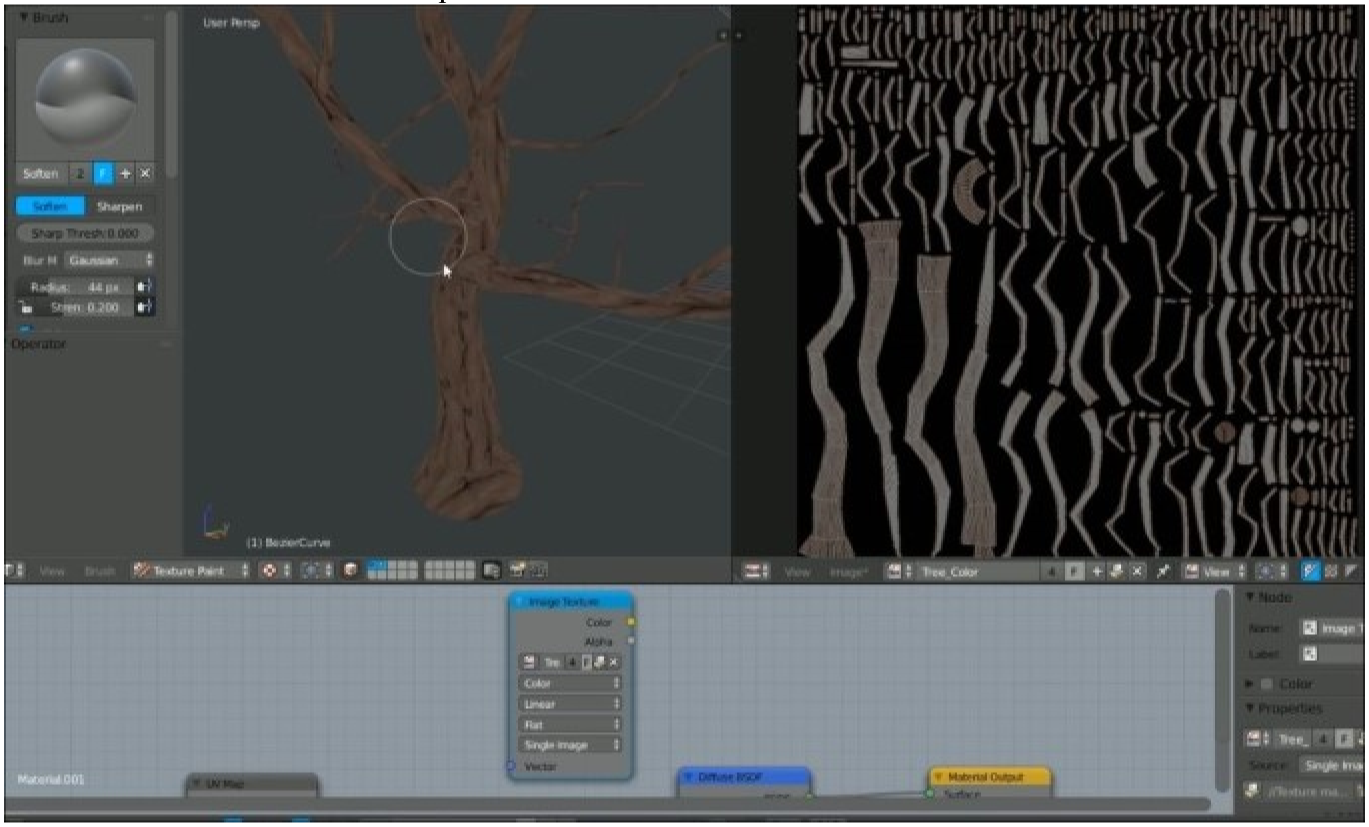
Baking the tree

We won't bake the maps of every objects in our scene with Cycles; however, we will see how to proceed with the 3D mesh of the tree as an example.

In order to optimize the render time, we will import our 3D mesh to another Blender window:

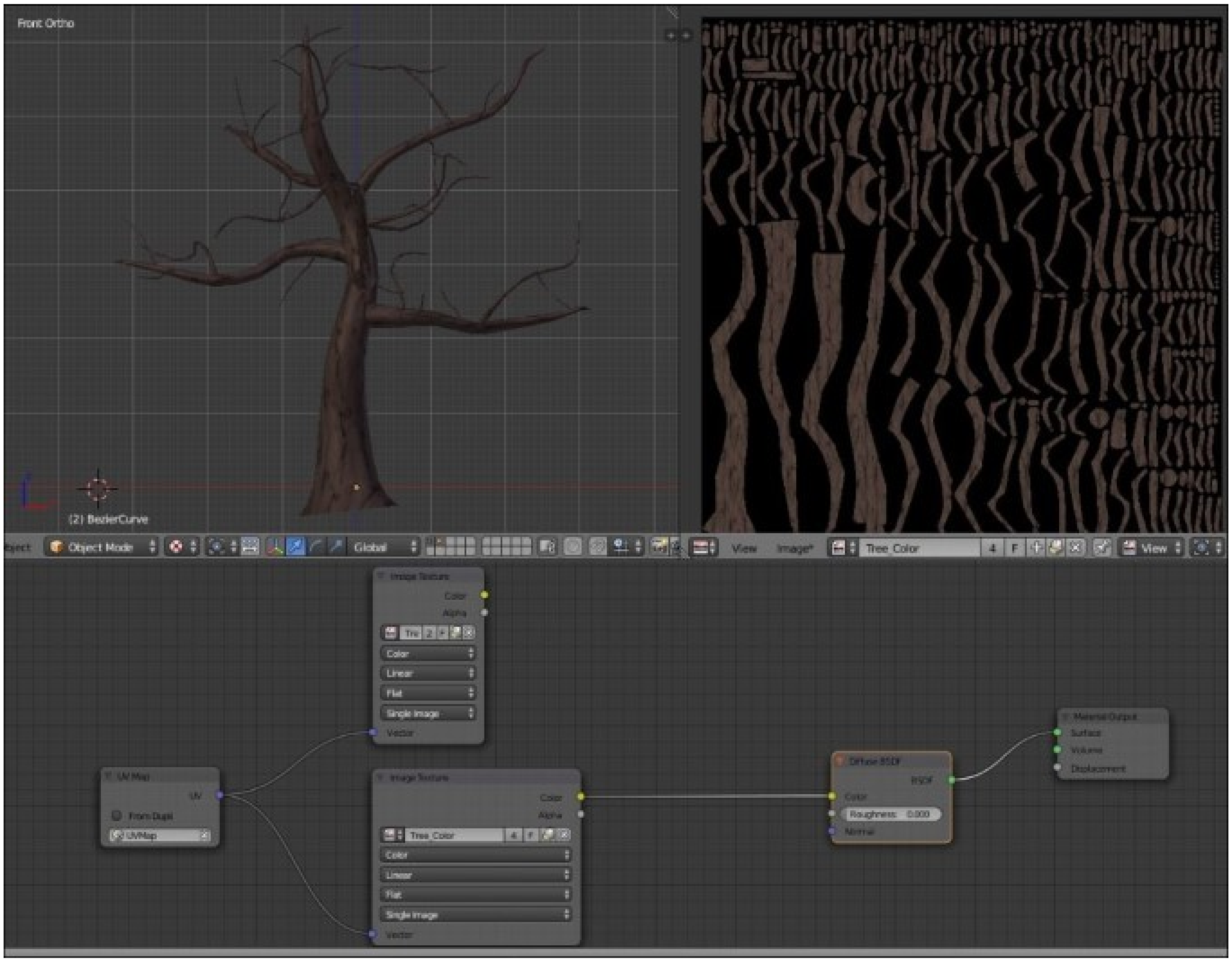
1. We will launch Blender a second time.
2. We will select the tree in the scene of the haunted house, and we will press *Ctrl* + *C* to copy it. You will see the **Copied selected object to buffer** message in the header of the work space.
3. In the other Blender window, we will press *Ctrl* +*V* to paste the tree. We will see the **Objects pasted from buffer** message.
4. In the same way, we will also import all the lights, and we must recreate the shader of the Image Base Lighting (we can append it from the main file). We don't modify the location of the tree and the lights in order to keep the same light configuration. But this wont be exactly the same lighting effect as we don't have the house here.
5. We will need a second UVs layer with UVs that are restrained in the UV square this time. We will use the *Ctrl* + *P* shortcut to automatically replace the UVs. Then we will adjust the margin in the options of the left panel of the 3D viewport.

6. We will start with color baking using the new UVs. For this, we will add an **Image Texture** node (*Shift* + *A* and select **Texture** | **Image Texture**) to the shader of the tree.
7. We will select all the polygons of the tree in the **Edit Mode** (*Tab* and *A*), and then we will create a new image named **Tree_Color**. A size of 2048 x 2048 is enough.
8. In the **Image texture** node that we have just created, we will select the **Tree_Color** texture. This node must stay u.nconnected.
9. Now we will need to go into the **Bake** tab in the **Render** options. Here, we will change the type to **Diffuse Color**. We will set the **Margin** value to **5**, and then we will press **Bake**.
10. When we have our color map, we must adjust the seams with Texture Paint. We will use the **Soften Brush** in order to blur the problems of the too visible seams.



Hiding the seams on the color bake

11. When the color texture is fine, we will again select the polygons of the tree in the **Edit Mode**, and we will create a new image (the same size) and rename it as **Tree_Combined**.
12. Now we can make another combined baking following the same process, which this time allows us to get all the lighting information on the texture. Make sure you open the good image in the **Image Texture** node with a high enough sample value. In our case, we have 500 samples to obviate the noise.
13. We can now go back to our haunted house scene and replace the tree with the one with new UVs (*Ctrl* + *C* and *Ctrl* + *V*); then we can replace the old texture with the **Tree Combined** texture in our shader.



The combined bake of the tree

Compositing a mist pass

As a bonus, we are going to learn how we can create and composite a mist pass with Cycles. But to do this, we will need to do a render. So let's do a render with 500 samples:

1. We will first have to activate the mist pass in the **Render Layer** tab of the Properties editor. Now we can access the mist settings in the World setting panel. We will set **Start** to **0 m** and Depth to **37 m**.
2. In order to see the mist, we will need to composite it over the render. We will learn more about compositing in further chapters, so don't worry if we don't go deep into the subject right now. In the Node Editor, we will have to switch to the Compositing Node mode (the second button after the material in the header). We will need to check **Use Nodes** and **Backdrop** in order to see our changes in real time. As you can see, we already have a RenderLayers node plug in the **Composite** node (the final output of the image).
3. In between, we can add a **Mix** node (press *Shift* + *A* and select **Color** | **Mix**) and feed the first color input with the Image output of the **RenderLayers** node. The **Fac** input of the **Mix** node will receive a **Map Value** node (press *Shift* + *A* and select **Vector** | **Map Value**) with **Offset** of **0.105** and **Size** of **0.06**. For the input of Map Value, we will simply plug our **Mist** pass (the fourth output of the **RenderLayers** node). The **Map Value** will control the amount of mist we see.
4. In order to view the result in the **Node Editor** in real time, you will have to add a **Viewer** Node. To do this, we will simply press *Ctrl* + *Shift* and right-click on any node.

The final Cycles render of the Haunted House project

5. We now have a nice mist! In order to change the aspect of the final render, we can tweak the **Color Management** options as we did in the previous chapter. Congratulations, you've completed the Haunted House project.

Summary

This chapter was really robust, but you now understand how to create nice materials with the Cycles Render engine and how to light a scene properly. You also learned how to produce normal maps and how to bake your objects. This last technique would be very interesting for video games. We also covered Blender's Compositing tool to a slight extent by mixing a mist pass. Now let's create a new project!

Chapter 8. Rat Cowboy – Learning To Rig a Character for Animation

This chapter will cover the rigging and the skinning of a character. This character will be a Rat Cowboy that has been already modeled for you. Here, you will understand what the rigging process involves. We will start by placing deforming bones. After this, we will learn how to rig these bones with controllers and constraints such as IK or Copy. Then, we will skin our character so that the mesh follows the deforming bones. As a bonus, you will learn how to use shape keys in order to add some basic facial controls that will be controlled by drivers. The rig, which is covered here, will be basic, but you will have all the necessary knowledge to go further. We are going to use this rig to animate our character in the next chapter. Enjoy!

In this chapter, we will cover the following topics:

- Making a symmetric skeleton
- Using the basic bones constraints
- Rigging the eyes
- Correcting the deformation of the meshes with weight painting
- Improving the accessibility of the rig with custom shapes
- Using shape keys

An introduction to the rigging process

We are now going to discover the process of character rigging. The point of this is to prepare objects or characters for animation in order to pose them in a simple way. For instance, when rigging a biped character, we will place virtual bones that mimic the character's real skeleton. Those bones are going to have relationships between them. In the case of a finger, for instance, we will usually add three bones that follow the phalanges. The tip bone will be the child of the mid bone, which in turn will be the child of the top bone. So when we rotate the top bone, it will automatically rotate its children. On the top of the network of the bones, we will need to add some constraints that define automation so that it is easier for the animator to pose the character. The next step is to specify to the geometry to follow the bones in some way. For instance, in the case of a character, we will tell Blender to deform the mesh according to the deformable bones. This stage is called **Weight Painting** in Blender and **Skinning** is a common term, too. However, we will not always face a case where skinning is necessary. For instance, if you have to rig a car, you will not want to deform the wheels, so you will create a bone hierarchy or constraints in order to follow the rig. The entire process could be tricky at some point, but mastering the rigging process allows you to better understand the animation process and is the reason why having a good topology is so important.

Note

Anatomy of a bone in Blender

A bone has a root and a tip. The root corresponds to the pivot point of the bone, and the tip defines the length of the bone. Bones can have a parent-child relationship in two ways. The first method is by connecting them, so the root of the child is merged with the tip of its parent. The other method is by