8

Human Skin Materials and Faking Sub Surface Scattering in Cycles

In this chapter, we will cover:

- Simulating SSS in Cycles by using the **Translucent** shader
- Simulating SSS in Cycles by using the **Vertex Color** layer
- Simulating SSS in Cycles by using the **Ray Length** option in the **Light Path** node
- Creating a basic human skin material in Cycles
- Creating a layered human skin material in Cycles
- Creating a fake Sub Surface Scattering node group

Introduction

At present, that is, in the official Blender 2.66a release, Cycles does not have a **Sub Surface Scattering (SSS)** shader yet. However, there are a number of ways SSS can be simulated.

Sub Surface Scattering is the effect due to the light not directly reflected by a surface, but penetrates it and bounces around internally before getting absorbed or leaving the surface at a nearby point. In short, "scattered". The RGB channels of a surface color have different scattering values, depending on the material. For example, for the human skin, the red component is the more scattered (as a rough approximation, you could say that: blue = 1, green = 2, red = 4).

To be honest, all of the following fake SSS recipes use the **Translucent** shader node to achieve this effect, and the shifting of the colors is simulated by giving a main color to the translucent component. Keep in mind that these "tricks" do not even compare to a real Sub Surface Scattering effect (that, incidentally, will be available with the upcoming 2.67 release), but are just ways to "give the impression" that a scattering of the light through the material surface is happening. Also, depending on the recipe, you'll see that the effect can be quite varied; each has its pro and cons, and the more suitable one should be used according to the type of material you are going to create. The differences in the recipes are in the way the nodes are arranged, with the translucency driven by different types of inputs.

Simulating SSS in Cycles using the Translucent shader

In this recipe, we will create a fake Sub Surface Scattering material by using the **Translucent BSDF** shader node:

Getting ready

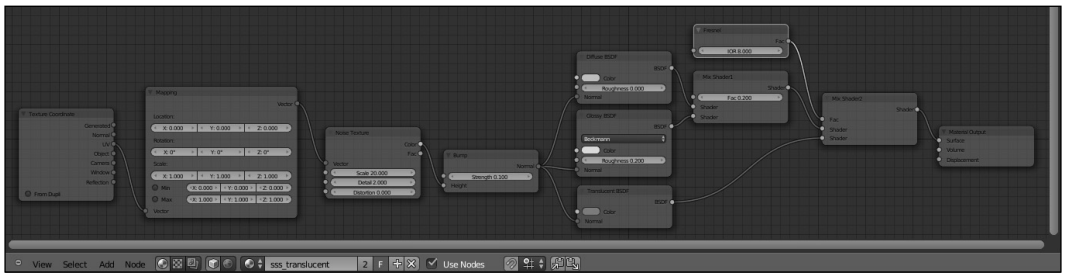
Start Blender and open the `1301OS_08_start.blend` file, where there is a Suzanne mesh leaning on a plane and two mesh-light planes.

How to do it...

Let's start with the material's creation:

1. Select the **Suzanne** object and click on the **New** button in the **Node Editor** window or in the **Material** window to the right. Rename the material to `sss_translucent`.

2. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node. In the first **Shader** slot, select a **Diffuse BSDF** shader and in the second one a **Glossy BSDF** shader node.
3. Set the **Diffuse** shader color to **R 0.031**, **G 0.800**, **B 0.000** (a bright green) and the **Glossy** color to **R 0.646**, **G 0.800**, **B 0.267** (a yellow). Also, set the **Glossy** shader's **Roughness** value to `0.200`.
4. Select the **Mix Shader** node and go to the **Active Node** panel to the right of the **Node Editor** window (if not present, place the mouse cursor in the **Node Editor** window and press *N* to make it appear). In the **Label** slot, rename the **Mix Shader** node as **Mix Shader1** and set its factor value to `0.200`.
5. Add a new **Mix Shader** node (press *Shift* + *A* and go to **Shader** | **Mix Shader**), rename it `Mix Shader2`, and paste it between the **Mix Shader1** node and the **Material Output** node.
6. Add a **Translucent BSDF** node (press *Shift* + *A* and go to **Shader** | **Translucent BSDF**) and connect it to the second **Shader** input socket of the **Mix Shader2** node. Set its color to **R 0.800**, **G 0.086**, **B 0.317** (a vivid pink).
7. Add a **Texture Coordinate** node (press *Shift* + *A* and go to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and go to **Vector** | **Mapping**) and a **Noise Texture** node (press *Shift* + *A* and go to **Texture** | **Noise Texture**).
8. Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node and the output of the latter to the **Vector** input socket of the **Noise** texture. Set the **Noise** texture's **Scale** value to `20.000`.
9. Add a **Bump** node (press *Shift* + *A* and go to **Vector** | **Bump**) and connect the **Color** output of the **Noise** texture to the **Height** input socket of the **Bump** node. Then connect the **Normal** output of the latter to the **Normal** input sockets of the **Diffuse**, **Glossy**, and **Translucent** nodes. Set the **Bump** node's strength to `0.100`.
10. Add a **Fresnel** node (press *Shift* + *A* and go to **Input** | **Fresnel**) and connect it to the **Fac** input socket of the **Mix Shader2** node. Set the **IOR** value to **8.000**, as shown in the following screenshot:



How it works...

This is probably the simplest form of the fake SSS effect you can get in Cycles, obtained by simply blending a translucent effect to a basic diffuse-glossy shader. By varying the amount of the **IOR** value in the **Fresnel** node, (set quite high as a starting point) it's possible to establish the amount of translucency on the mesh. We also added a **Noise** texture bump effect to the material, just to make it appear more jelly-like.

Note that we gave almost complementary colors to the **Diffuse** and to the **Translucent** shaders only to show the effect clearly, but closer colors can work better. Also note that the translucent effect actually follows the direction of the lighting: try to rotate the **Emitter** and the **Emitter_back** planes around the Suzanne mesh to verify this in real time through the **Rendered** view:

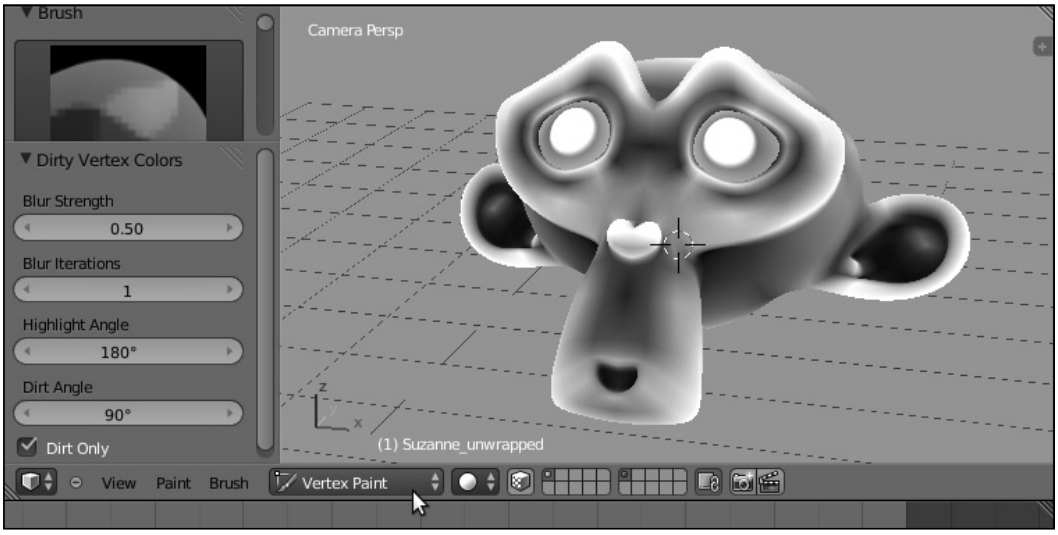
Simulating SSS in Cycles by the Vertex Color layer

In this recipe, we will create a fake Sub Surface Scattering material by using the Vertex Color feature, as shown here:

Getting ready

Start Blender and open the `1301OS_08_start.blend` file, where there is a Suzanne mesh leaning on a plane and two mesh-light planes.

1. Select the **Suzanne** mesh, click on the **Mode** button in the **Camera** view, header and choose **Vertex Paint**: **Suzanne** turns a shadeless white color.
2. Put the mouse cursor on the **Paint** item to the left of the **Mode** button, click and select **Dirty Vertex Colors**, and then press *T* and in the bottom window (**Dirty Vertex Color**) of the **Object Tools** panel, set **Blur Strength** to **0.50**, **Dirt Angle** to **90**, and check the **Dirt Only** option, as shown here:



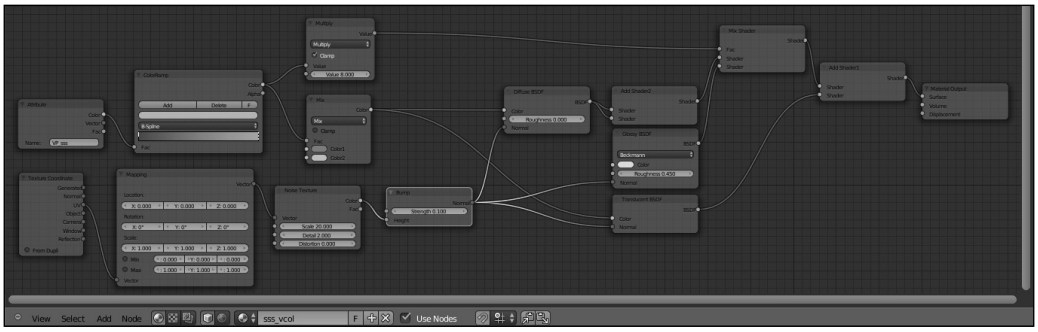
3. Go to the **Object Data** window under the **Properties** panel and, in the **Name** slot under the **Vertex Colors** sub-panel, rename the vertex color layer as `VP_sss`. Return in **Object Mode** and press *T* to close the **Object Tools** panel.

How to do it...

After the vertex color preparation, let's go with the material itself:

1. Click on the **New** button in the **Node Editor** window or in the **Material** window under the **Properties** panel. Rename the material as `sss_vcol`.
2. In the **Material** window, switch the **Diffuse BSDF** shader with an **Add Shader** node. In the first **Shader** slot, select a **Mix Shader** node and in the second one a **Translucent BSDF** shader node. In the **Active Node** panel to the right of the **Node Editor** window, rename the **Add Shader** node as `Add Shader1`.

3. Go to the **Mix Shader** node and in the first **Shader** slot select a new **Add Shader** node and in the second a **Glossy BSDF** shader node. Rename the new **Add Shader** node as `Add Shader2` and set the **Glossy** shader's **Roughness** value to `0.450`.
4. Add a **Diffuse BSDF** shader (press *Shift* + *A* and go to **Shader** | Diffuse **BSDF**) and connect its output to both the **Shader** input sockets of the **Add Shader2** node.
5. Add a **Mix** node (press *Shift* + *A* and go to **Color** | **Mix**) and connect its color output to the **Color** input sockets of the **Diffuse** and **Translucent** shader nodes. Set **Color1** to **R 0.800**, **G 0.086**, **B 0.317** (the same pink as the preceding recipe) and **Color2** to **R 0.031**, **G 0.800**, **B 0.000** (the same bright green as the preceding recipe). Set the **Glossy** color to **R 0.646**, **G 0.800**, **B 0.267** (yes, it's the previous recipe's yellowish color).
6. Add an **Attribute** node (press *Shift* + *A* and go to **Input** | **Attribute**) and a **ColorRamp** node (press *Shift* + *A* and go to **Convertor** | **ColorRamp**). In the **Name** slot of the **Attribute** node, write the vertex color layer name, that is, `VP_sss`, and then connect the color output to the **Fac** input socket of the **ColorRamp** node.
7. Connect the **ColorRamp** color output to the **Fac** input socket of the **Mix** node, and then set the interpolation to **B-Spline**. Set the white color marker to **RGB 0.500**.
8. Add a **Math** node (press *Shift* + *A* and go to **Convertor** | **Math**) and connect the color output of the **ColorRamp** node to the first **Value** input socket. Set the operation to **Multiply**, the second **Value** to `8.000` and check the **Clamp** option. Connect its **Value** output to the **Fac** input socket of the **Mix Shader** node.
9. Add a **Texture Coordinate** node (press *Shift* + *A* and go to **Input** | **Texture Coordinate**), a **Mapping** node (press *Shift* + *A* and go to **Vector** | **Mapping**), and a **Noise Texture** node (press *Shift* + *A* and go to **Texture** | **Noise Texture**).
10. Connect the **UV** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node, and the output of the latter to the **Vector** input socket of the **Noise** texture. Set the **Noise** texture's **Scale** value to `20.000`.
11. Add a **Bump** node (press *Shift* + *A* and go to **Vector** | **Bump**) and connect the **Color** output of the **Noise** texture to the **Height** input socket of the **Bump** node. After this, connect the **Normal** output of the latter to the **Normal** input sockets of the **Diffuse**, **Glossy**, and **Translucent** nodes. Set the **Bump** node's **Strength** value to **0.100**, as shown in the following screenshot:



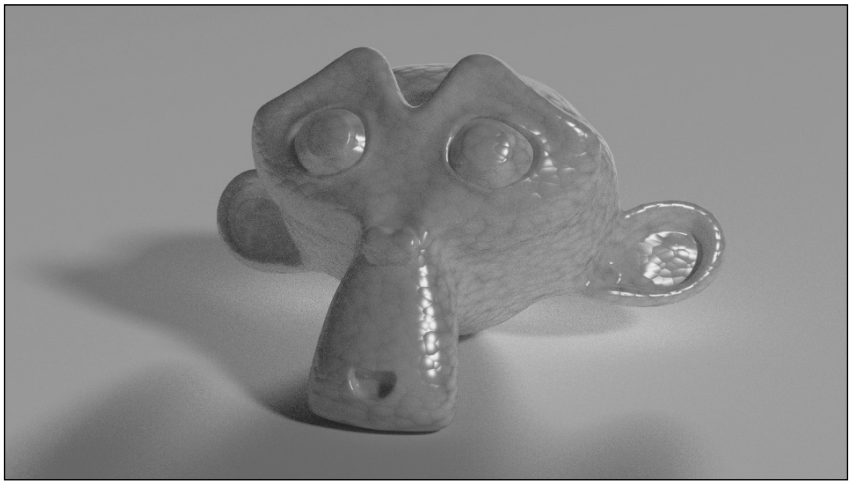
How it works...

Compared to the previous recipe, in this case we used the **Vertex Color** information to drive the mixing of the translucency with the other components of the shader. It's clear that the final result is largely due to the vertex painting, which, in our example here, we have quickly obtained using the **Dirty Vertex Color** feature.

Note that we used the same colors as used in the previous recipe, to make the comparison of the different effects of the two recipes a lot easier.

Simulating SSS in Cycles by the Ray Length option in the Light Path node

In this recipe, we will create a fake Sub Surface Scattering material by using the **Ray Length** output option of the **Light Path** node:



Getting ready

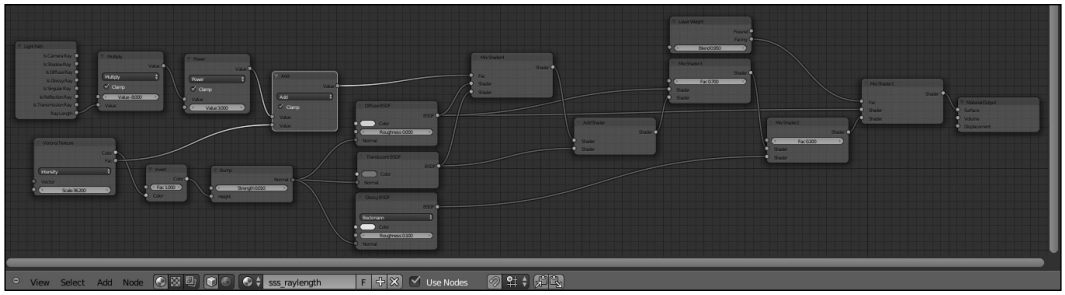
Start Blender and open the `1301OS_08_start.blend` file, where there is a Suzanne mesh leaning on a plane and two mesh-light planes.

How to do it...

Let's jump straight to the material's creation:

1. Select the **Suzanne** object and click on the **New** button in the **Node Editor** window or in the **Material** window to the right. Rename the material as `sss_raylength`.
2. In the **Material** window, switch the **Diffuse BSDF** shader with a **Mix Shader** node and rename it as `Mix Shader1`. In its first **Shader** slot, select a **Diffuse BSDF** shader and in its second one a new **Mix Shader** node that you'll rename `Mix Shader2`.
3. Go to the **Mix Shader2** node and in its first **Shader** slot, select a new **Mix Shader** node and rename this as `Mix Shader3`. In the second **Shader** slot, select a **Glossy BSDF** node.
4. Add a **Layer Weight** node (press *Shift* + *A* and go to **Input** | **Layer Weight**) and connect its **Facing** output to the **Fac** input socket of the **Mix Shader1** node. Set the **Blend** value to `0.950`.
5. Set the **Fac** value of the **Mix Shader2** node to `0.200` and the **Fac** value of the **Mix Shader3** node to `0.700`. Set the **Glossy** shader's **Roughness** value to `0.100`.
6. Connect the **Diffuse** node output to the first **Shader** input socket of the **Mix Shader3** node. Add an **Add Shader** node (press *Shift* + *A* and go to **Shader** | **Add Shader**) and connect it to the second **Shader** input socket of the **Mix Shader3** node.
7. Go to the **Add Shader** node and in the first **Shader** slot select a last **Mix Shader** node, and rename it as `Mix Shader4`. In the second **Shader** slot, select a **Translucent BSDF** shader node.
8. Connect the output of the **Diffuse** node to the first **Shader** input socket of the **Mix Shader4** node and the output of the **Translucent** node to the second one.
9. Set the **Diffuse** shader color to **R 0.031**, **G 0.800**, **B 0.000** and the **Translucent** color to **R 0.800**, **G 0.086**, **B 0.317**.
10. Add a **Voronoi Texture** node (press *Shift* + *A* and go to **Texture** | **Voronoi Texture**) and a **Bump** node (press *Shift* + *A* and go to **Vector** | **Bump**). Connect the **Color** output of the texture to the **Height** input socket of the **Bump** node and the **Normal** output of the latter to the **Normal** input sockets of the **Diffuse**, **Glossy**, and **Translucent** nodes. Set the **Voronoi** texture's **Scale** value to `32.600` and the **Bump** node's **Strength** value to `0.010`.
11. Add an **Invert** node (press *Shift* + *A* and go to **Color** | **Invert**) and paste it between the **Voronoi** texture and the **Bump** node.

12. Add a **Light Path** node (press *Shift* + *A* and go to **Input** | **Light Path**) and a **Math** node (press *Shift* + *A* and go to **Convertor** | **Math**). Set the **Math** node operation to **Multiply** and connect the **Ray Length** output of the **Light Path** node to the first **Value** input socket of the **Math** node, and then set the second **Value** to `-8.000`. Check the **Clamp** option.
13. Press *Shift* + *D* to duplicate the **Math** node, set the operation to **Power**, and connect the output of the **Multiply** math node to the first **Value** input socket of the latter. Set the second **Value** to `3.000` and check the **Clamp** option.
14. Press *Shift* + *D* to duplicate the **Power** math node, set the operation to **Add**, and connect the output of the **Power** math node to its first **Value** input socket. Connect its output to the **Fac** input socket of the **Mix Shader4** node and check the **Clamp** option.
15. Connect the **Fac** output of the **Voronoi** texture to the second **Value** input socket of the **Add** math node, as shown here:



How it works...

In this recipe, we used the **Ray Length** output of the **Light Path** node to drive the amount of translucency on the mesh. The **Ray Length** does exactly what its name suggests, it returns the length of a light ray passing through an object. So basically, it is possible for Cycles to know the thickness of a mesh: on the thicker parts the translucency will show less or even nothing at all, whereas it will be more visible on the thinner parts of the mesh.

Note that in the recipe's shader network, the **Ray Length** output has been enhanced by a set of **Math** nodes and added to the **Voronoi** texture output, and then connected to the factor input of the **Mix Shader** node to drive the blending of the diffuse and of the translucent components.

Creating a basic simple skin shader in Cycles

In this recipe, we will create a basic skin material using the Sintel mesh.

Sintel is the main character of the third Open Movie by the same name produced by the Blender Foundation. Designed by David Revoy and modeled by the Durian Team, the Sintel character (as well as all the other movie assets) is licensed under the Creative Commons Attribution 3.0 (`http://creativecommons.org/licenses/by/3.0/`):

Getting ready

Start Blender and open the `1301OS_08_skin_start.blend` file, where there is an already set scene with the Sintel character on a plane.

Because we are going to work on the skin shader, the eyeballs and the cornea already have their material assigned, and all the other meshes (cloth and hair) have a generic "gesso" material already assigned as well.

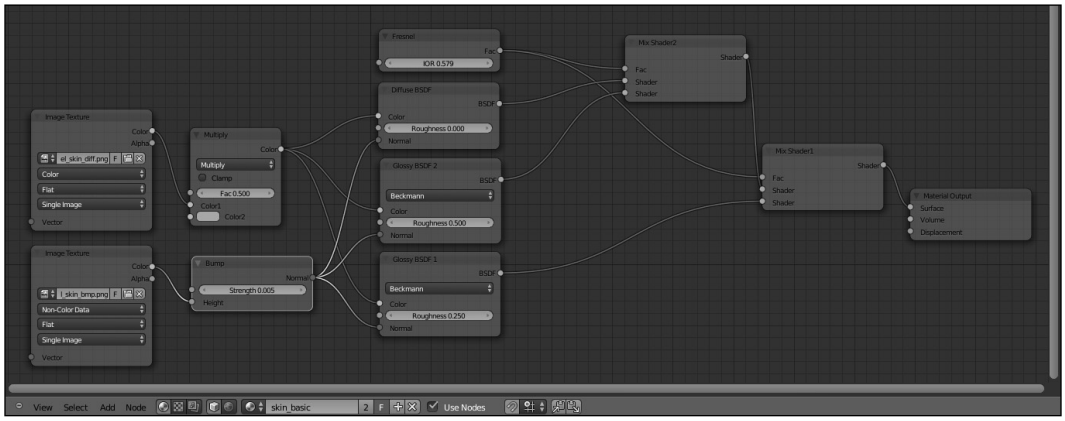
The Sintel mesh object is already selected and ready for the skin shader creation.

How to do it...

Let's jump right in with the skin material's creation:

1. Click on the **New** button in the **Node Editor** window or in the **Material** window under the **Properties** panel and rename the material as `skin_basic`.
2. In the **Material** window, switch the **Diffuse** shader with a **Mix Shader** node. Go to the **Active Node** panel to the right of the **Node Editor** window (if not present, place the mouse cursor in the **Node Editor** window and press *N* to make it appear) and, in the **Label** slot, rename the **Mix Shader** node as `Mix Shader1`.
3. In the first **Shader** slot of the **Mix Shader1** node, select a new **Mix Shader** node and rename it as `Mix Shader2`. In the second **Shader** slot of the **Mix Shader1** node, select a **Glossy BSDF** shader node. Rename the **Glossy** shader as `Glossy BSDF 1` and set its **Roughness** value to `0.250`.
4. Go to the **Mix Shader2** node and in the first **Shader** slot select a **Diffuse BSDF** node and in the second one a new **Glossy BSDF** node. Rename it as `Glossy BSDF 2` and set its **Roughness** value to `0.500`.
5. Add a **Fresnel** node (press *Shift* + *A* and go to **Input** | **Fresnel**) and connect it to the **Fac** input sockets of both the **Mix Shader1** and **Mix Shader2** nodes. Set the **IOR** value to `0.579`.
6. Add an **Image Texture** node (press *Shift* + *A* and go to **Texture** | **Image Texture**) and a **Mix** node (press *Shift* + *A* and go to **Color** | **Mix**). Connect the color output of the **Image Texture** node to the **Color1** input socket of the **Mix** node. Set **Blend Type** of the **Mix** node to **Multiply** and the **Color2** to **R 0.800**, **G 0.357**, **B 0.211**.
7. Connect the output of the **Multiply** mix node to the **Color** input sockets of the **Diffuse** shader and of the two **Glossy** shader nodes.
8. Click on the **Open** button of the **Image Texture** node, browse to the `textures` folder, and load the `sintel_skin_diff.png` image.
9. Add a new **Image Texture** node (press *Shift* + *A* and go to **Texture** | **Image Texture**) and a **Bump** node (press *Shift* +*A* and go to **Vector** | **Bump**). Connect the color output of this second **Image Texture** node to the **Height** input socket of the **Bump** node, and the **Normal** output of the latter to the **Normal** input sockets of the **Diffuse** shader and of the two **Glossy** shader nodes.

10. Click on the **Open** button of the second **Image Texture** node, browse to the `textures` folder, and load the `sintel_skin_bmp.png` image. Set the **Color Space** option to **Non-Color Data** and the **Bump** node's **Strength** value to `0.005`, as shown here:



How it works...

Although the final result is not that bad, this is a very basic human skin shader that is only good for secondary characters (it is actually quite fast in rendering). It uses just two image maps, one for the color and one for the per-shader bump effect, and it doesn't even have translucency, there is no scattering effect at all: it simply uses two **Glossy** shaders to mimic the smooth specularity of the human skin.

Note that in this case, we didn't use the **Texture Coordinate** and the **Mapping** nodes, because it would have been useless: for the **Image Texture** nodes, if a mesh has UV coordinates and no other mapping method is specified, Cycles uses the UV mapping by default.

Creating a layered human skin material in Cycles

In this recipe, again by using the open-content character Sintel, we will create a layered and more realistic skin material, as shown here:

Getting ready

Start Blender and open the `1301OS_08_skin_start.blend` file, where there is an already set scene with the Sintel character on a plane.

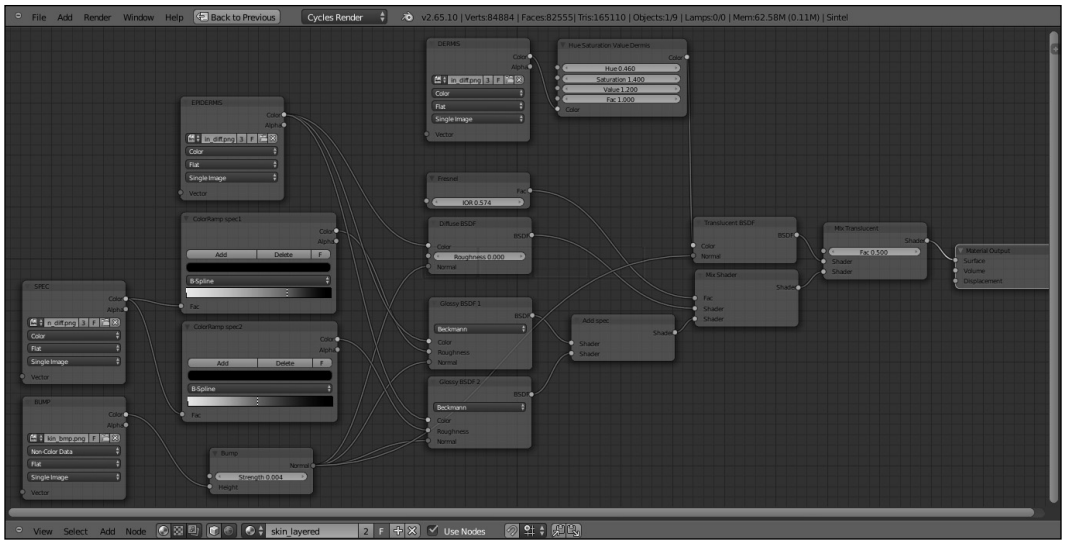
How to do it...

Let's jump right in and start with the layered skin shader's creation:

1. Click on the **New** button in the **Node Editor** window or in the **Material** window under the **Properties** panel and rename the material as `skin_layered`.
2. In the **Material** window, switch the **Diffuse** shader with a **Mix Shader** node. Now, go to the **Active Node** panel to the right of the **Node Editor** window (if not present, place the mouse cursor in the **Node Editor** window and press *N* to make it appear) and, in the **Label** slot, rename the **Mix Shader** node as `Mix Translucent`.

3. In the first **Shader** slot of the **Mix Translucent** node, select a **Translucent BSDF** shader node and in the second one a new **Mix Shader** node.
4. In the first **Shader** slot of this new **Mix Shader** node, select a **Diffuse BSDF** shader node and in the second one an **Add Shader** node. Rename this as `Add Spec`.
5. Add two **Glossy BSDF** shader nodes (press *Shift* + *A* and go to **Shader** | **Glossy BSDF**) and rename them as `Glossy BSDF 1` and `Glossy BSDF 2` respectively. Connect their output to the first and to the second **Shader** input sockets of the **Add Spec** node respectively.
6. Add a **Fresnel** node (press *Shift* + *A* and go to **Input** | **Fresnel**) and connect it to the **Fac** input socket of the **Mix Shader** node. Set the **IOR** value to `0.574`.
7. Add an **Image Texture** node (press *Shift* + *A* and go to **Texture** | **Image Texture**) and rename it as `EPIDERMIS`. Connect its color output to the **Color** input sockets of the **Diffuse** and of the two **Glossy** shader nodes.
8. Click on the **Open** button of the **EPIDERMIS** image texture node, browse to the `textures` folder, and load the `sintel_skin_diff.png` image.
9. Press *Shift* + *D* to duplicate the **EPIDERMIS** node and rename it as `SPEC`. Add two **ColorRamp** nodes (press *Shift* + *A* and go to **Convertor** | **ColorRamp**) and connect the color output of the **SPEC** node to the **Fac** input socket of both the two **ColorRamp** nodes. Rename the first **ColorRamp** as `ColorRamp spec1` and the second **ColorRamp** node as `ColorRamp spec2`.
10. Connect the color output of the **ColorRamp spec1** node to the **Roughness** input socket of the **Glossy BSDF 1** shader node. Connect the color output of the **ColorRamp spec2** node to the **Roughness** input socket of the **Glossy BSDF 2** shader node.
11. Set the **Interpolation** value for both the two **ColorRamp** nodes to **B-Spline**. Go to the **ColorRamp spec1** node and move the black color marker three-quarters to the right and the white color marker to the extreme left. Go to the **ColorRamp spec2** node and move the black color marker to the middle of the slider and the white color marker to the extreme left.
12. Add a new **Image Texture** node (press *Shift* + *A* and go to **Texture** | **Image Texture**) and a **Bump** node (press *Shift* + *A* and go to **Vector** | **Bump**). Rename this **Image Texture** node as `BUMP` and connect its color output to the **Height** input socket of the **Bump** node, and the **Normal** output of the latter to the **Normal** input sockets of the **Diffuse** shader and of the two **Glossy** shader nodes.
13. Click on the **Open** button of the **BUMP** image texture node, browse to the `textures` folder, and load the `sintel_skin_bmp.png` image. Set the **Color Space** option to **Non-Color Data** and the **Bump** node's **Strength** value to `0.004`.

14. Press *Shift* + *D* to again duplicate the **EPIDERMIS** node and rename it as `DERMIS`. Add a **Hue Saturation Value** node (press *Shift* + *A* and go to **Color** | **Hue Saturation Value**), rename it as `Hue Saturation Value Dermis`, and connect the color output of the **DERMIS** node to its color input socket. Connect the color output of the **Hue Saturation Value Dermis** node to the color input socket of the **Translucent BSDF** shader node.
15. Set the **Hue** value to **0.460**, **Saturation** to **1.400**, and **Value** to **1.200**, as shown here:



How it works...

In this recipe we used a layered approach to build the human skin shader, but what exactly does "layered" mean?

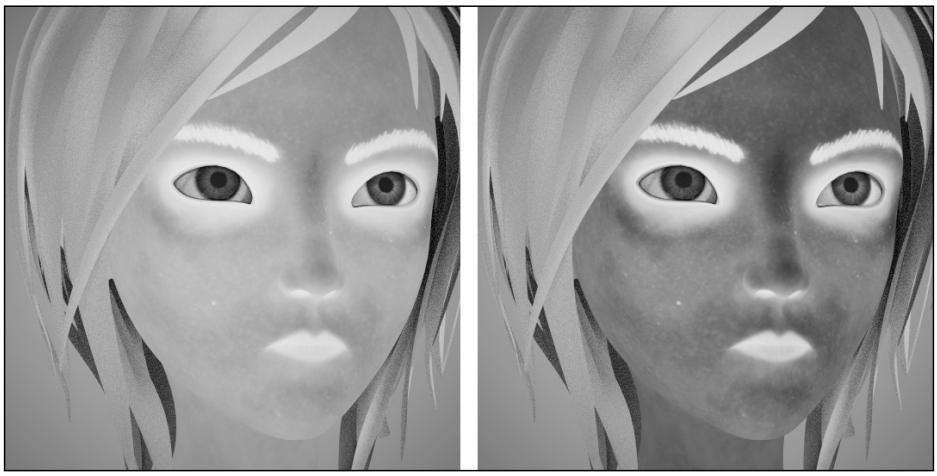
It means that the shader tries to simulate the behavior of real human skin in the most effective way possible: I'm referring to the fact that human skin is made up of several different overlapping and semi-transparent layers that reflect and absorb light rays in various ways, giving a reddish color to certain areas and also the famous SSS effect.

Now, a perfect reproduction of the real human skin model is not necessary; usually it's enough to use different image maps for the key components of the shader, each one added on the other: the base color, the dermis blood layer, the specularity map, and the bump map.

In our case, we had at our disposal only two image maps, the `sintel_skin_diff.png` color image and the `sintel_skin_bmp.png` gray-scale map, which we used straight for the bump. We could have obtained the "missing" maps with the aid of image editor software (as, for example, The Gimp), but for the sake of this exercise, to obtain the needed missing images, we altered by the nodes the color map: so, starting from the **EPIDERMIS** layer, that is, the color map:

We obtained, using the **Hue Saturation Value** node, the **DERMIS** version, that is, the blood-vessel layer lying beneath the epidermis:

And by the use of the **ColorRamp** nodes, two gray-scale versions for the specularity component, one sharp specularity map, and a softer one:

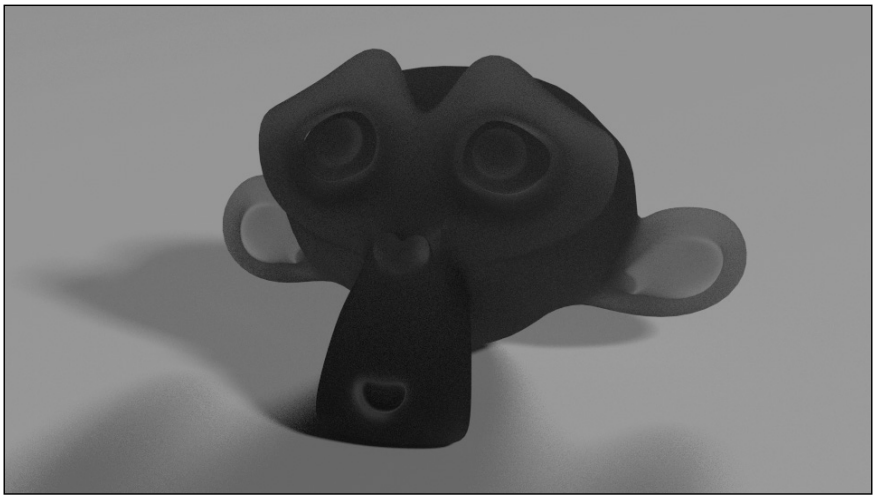


Then, the `sintel_skin_bmp.png` map was connected to the **Bump** node for the per-shader bump effect (in the following image, the details of the bump map are barely visible, but look at the real map in the `textures` folder):

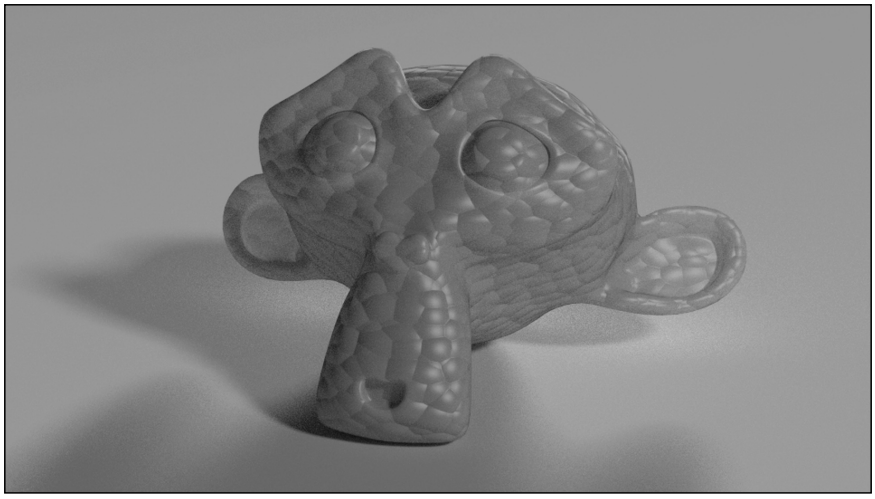
Note that, because we used the color map to obtain all the others, certain areas of the images are wrong: for example, the eyebrows, showing in pure white on the specularity maps, should have been removed. In any case, this doesn't show that much on the final render, and the result is more than acceptable.

Creating a fake Sub Surface Scattering node group

In this recipe, we will create a fake Sub Surface Scattering node group, which can be mixed with other nodes to add a scattering effect to a material. In the following image, you see the effect of just the SSS node alone on the Suzanne mesh:



And in this image, you see the effect of the node group added to an average material:



Getting ready

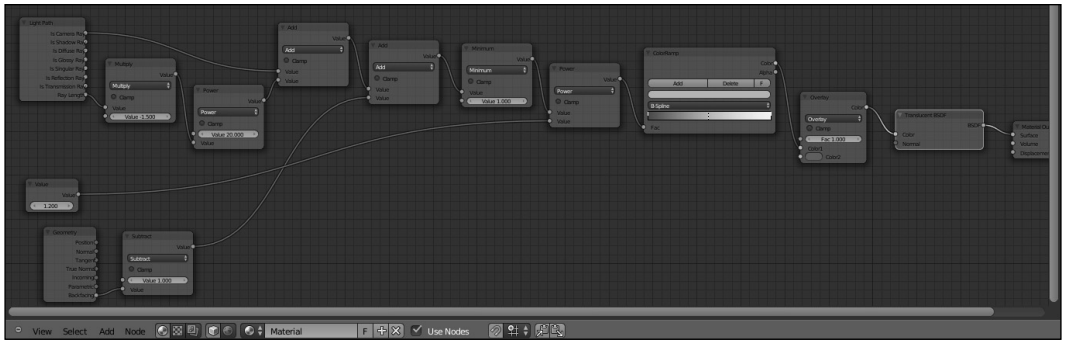
Start Blender and open the `1301OS_08_start.blend` file.

How to do it...

Let's immediately start with the node group creation:

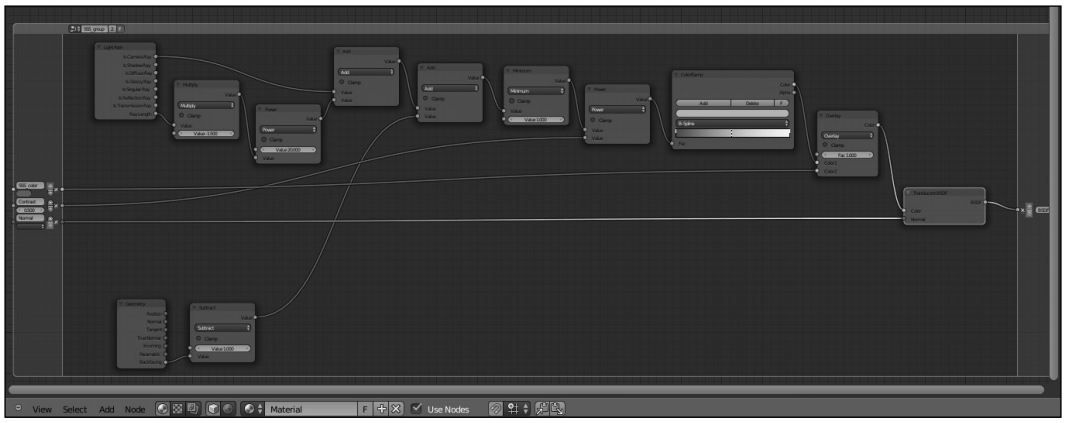
1. Click on the **New** button in the **Node Editor** window or in the **Material** window under the **Properties** panel. In the **Node Editor** window, delete the **Diffuse BSDF** shader node.
2. Add a **Light Path** node (press *Shift* + *A* and go to **Input** | **Light Path**) and a **Geometry** node (press *Shift* + *A* and go to **Input** | **Geometry**).
3. Add a **Math** node (press *Shift* + *A* and go to **Convertor** | **Math**). Set its operation to **Multiply** and connect the **Ray Length** output of the **Light Path** node to the first **Value** input socket. Set the second **Value** to `-1.500`.
4. Press *Shift* + *D* to duplicate the **Multiply** math node and set the operation to **Power**. Connect the **Multiply** math node output to the second **Value** input socket of the **Power** node. Set the first **Value** to `20.000`.
5. Press *Shift* + *D* to duplicate the **Power** node and set the operation to **Add**. Connect the **Power** node output to the second **Value** input socket of the **Add** math node and the **Is Camera Ray** output of the **Light Path** node to the first **Value** input socket of the **Add** math node.
6. Press *Shift* + *D* to duplicate the **Add** node and set the operation to **Minimum**. Connect the output of the **Add** node to the first **Value** input socket of the **Minimum** node and set the second **Value** to `1.000`.
7. Press *Shift* + *D* to duplicate the **Power** node and move it after the **Minimum** node. Connect the output of the latter to the first **Value** input socket of the duplicated **Power** node.
8. Add a **Value** node (press *Shift* + *A* and go to **Input** | **Value**), rename it as `Contrast`, and connect its output to the second **Value** input socket of the last **Power** math node. Set the **Value** to `1.200`.
9. Press *Shift* + *D* to duplicate a **Math** node (any one will do), set the operation to **Subtract**, and connect the **Backfacing** output of the **Geometry** node to its second **Value** input socket. Set the first **Value** to `1.000`.
10. Press *Shift* + *D* to duplicate the **Subtract** node, set the operation to **Add**, and paste it between the first **Add** and the **Minimum** nodes. Connect the output of the **Subtract** node to the second **Value** input socket of the last **Add** node.

11. Add a **ColorRamp** node (press *Shift* + *A* and go to **Convertor** | **ColorRamp**) and connect the output of the last **Power** math node to its **Fac** input socket. Add a **Translucent BSDF** node (press Shift + A and go to **Shader** | **Translucent BSDF**) and connect the color output of the **ColorRamp** node to its color input socket.
12. Set the **ColorRamp** node's interpolation to **B-Spline** and click on the **Add** button to add a new marker with color **RGB 0.500** at the middle of the slider.
13. Add a **Mix** node (press *Shift* + *A* and go to **Color** | **Mix**), set the **Blend Type** to **Overlay**, and the **Fac** value to `1.000`. Connect the color output of the **ColorRamp** node to the **Color1** input socket and set the **Color2** to **R 0.500**, **G 0.054**, **B 0.077**.
14. Connect the color output of the **Overlay** node to the color input socket of the **Translucent BSDF** node. Connect the output of the **Translucent** node to the **Surface** input socket of the **Material Output** node, as shown in the following screenshot:



15. Now, select all the nodes except for the **Value** and the **Material Output** nodes. Press *Crtl* + *G* and confirm to create a **Node Group**. Rename the node group as `SSS_group`.
16. Rename **Contrast**, the exposed input socket to the left of the node group envelope, and set the value on the group interface, then delete the original **Value** node. Click and drag the **Color2** socket of the **Overlay** node to expose it to the left of the group envelope. Rename the exposed socket as `SSS_color`. Also, click and drag the **Normal** socket of the **Translucent** shader node to the left.

17. Press *Tab* to close the node group.

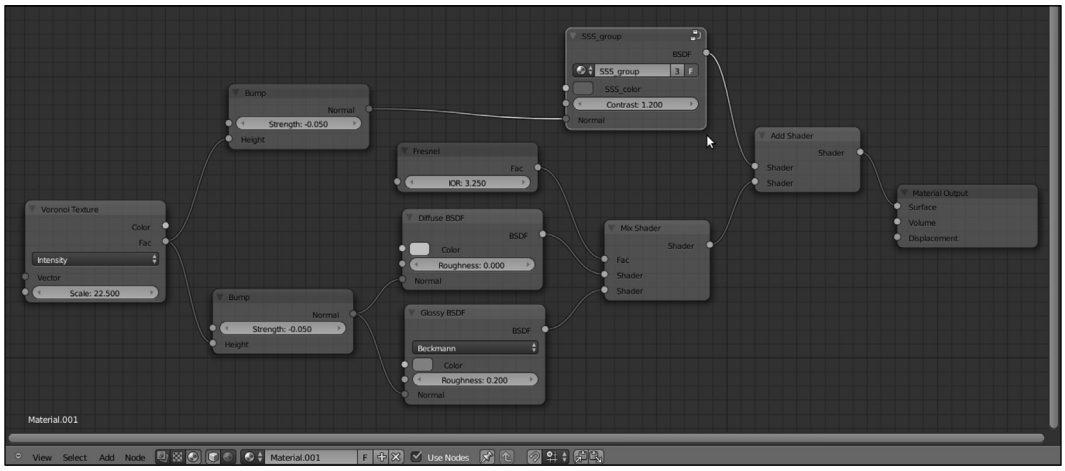


So, now we have made the SSS node group, ready to be combined with any surface material. Let's now create a simple material to mix the node group to:

1. Add a **Mix Shader** node, a **Diffuse BSDF**, and a **Glossy BSDF** shader node (press *Shift* + *A* and go to **Shader**). Connect the **Diffuse** node output to the first **Shader** input socket of the **Mix Shader** node and the **Glossy** shader output to the second one.
2. Add a **Material Output** node (press *Shift* + *A* and go to **Output** | **Material Output**) and connect the **Mix Shader** output to the **Surface** input socket of the **Material Output** node.
3. Set the color of the **Diffuse** shader node to **R 0.070**, **G 0.800**, **B 1.66** and the color of the **Glossy** shader to **R 0.800**, **G 0.089**, **B 0.514**.
4. Add a **Voronoi Texture** node (press *Shift* + *A* and go to **Texture** | **Voronoi Texture**) and a **Bump** node (press *Shift* + *A* and go to **Vector** | **Bump**). Connect the **Fac** output of the texture node to the **Height** input socket of the **Bump** node, and the output of the latter to the **Normal** input sockets of both the **Diffuse** and the **Glossy** shaders.
5. Set the **Bump** node's **Strength** value to `-0.050` and the **Voronoi** texture's **Scale** value to `22.500`.
6. Add a **Fresnel** node (press *Shift* + *A* and go to **Input** | **Fresnel**) and connect it to the **Fac** input socket of the **Mix Shader** node. Set the **IOR** value to `3.250`.

Now let's add the SSS node group:

1. Add an **Add Shader** node (press *Shift* + *A* and go to **Shader** | **Add Shader**) and paste it between the **Mix Shader** node and the **Material Output** node. Switch the connection from the first socket to the second one (this actually shouldn't be required, in this case, because the shaders get "added" anyway).
2. Connect the output of the **SSS_group** node to the first socket of the **Add Shader** node. Add a new **Bump** node, connect the **Fac** output of the **Voronoi** texture to its **Height** socket, and the **Normal** output of the **Bump** to the **Normal** input socket of the **SSS_group** node. Set the **Bump** node's **Strength** value to **-0.050** as well, as shown here:



How it works...

- The **Ray Length** and the **Is Camera Ray** outputs of the **Light Path** node are added together. The **Ray Length** defines the "thickness" of the mesh, and it's also "clamped" by the first **Multiply** node and by the **Power** node. The **Is Camera Ray** output tells Cycles to render only the surface points directly hit by the light rays directly shot from the camera. The two outputs, added to each other, give a sort of "stencil" effect, gray-scale values distributed on the ground of the thickness of the mesh.
- Then the **Backfacing** output of the **Geometry** node is added as well, so to take in consideration also the color of the "back" mesh faces into consideration. All this is multiplied by the second **Power** node for the **Contrast** value and further clamped by the **ColorRamp** node.
- At this point, the result is mixed with the **SSS_color** by the **Overlay** node and finally connected to the color input socket of the **Translucent** shader, resulting in the semi-transparent-looking shader of the first image at the beginning of this recipe.