Chapter 13. Creating the Materials in Blender Internal

In this chapter, we will cover the following recipes:

- Building the reptile skin shaders in Blender Internal
- Building the eyes' shaders in Blender Internal
- Building the armor shaders in Blender Internal

Introduction

In this chapter we'll see how to set up the materials for the **Gidiosaurus** and the **Armor** in the **Blender Render** engine; in fact, although not exactly of the same quality as in **Cycles**, it is also possible to obtain quite similar shader results in **Blender Internal**:

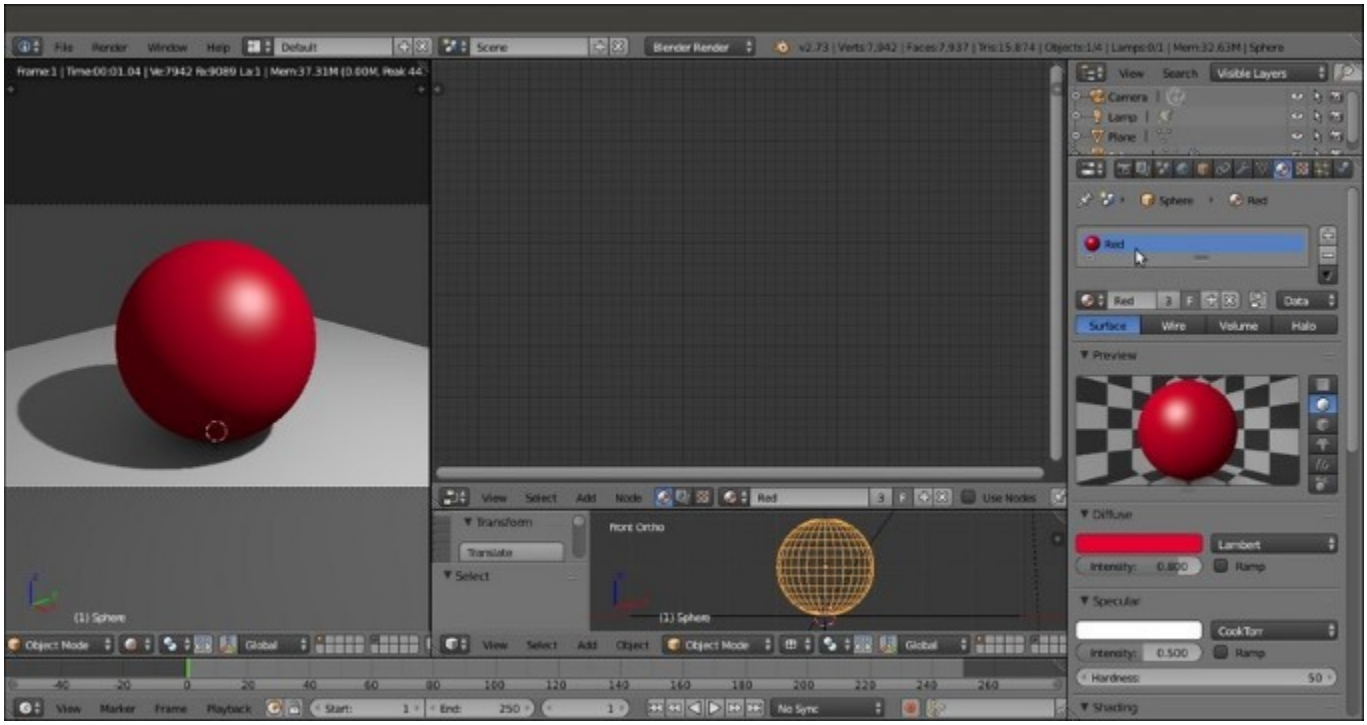
Comparison of the Gidiosaurus character rendered in Cycles (left) and Blender Internal (right)

If you are wondering why we should re-do in the **Blender Render** engine, which is quite old and no longer developed and/or supported, the same thing we have already done in **Cycles**, there are several possible reasons: for example, no doubt **Cycles** is superior in quality but, compared with the scanline **BI**, its rendering is (and, being a path-tracer, always will be) slower; even with the aid of a render-farm, rendering times are still a *money* issue in the production of animations.

The previous screenshot shows, for comparison, only the top parts of two full shot renderings of the **Gidiosaurus** character: the **Cycles** rendering to the left took around **1** hour and **20** minutes (**1920** × **1080** resolution CPU rendering with *Intel Core 2 Duo T6670 2.20 GHz* and **4** GB of RAM, in Ubuntu 12.04 64-bit); the **Blender Internal** rendering to the right took only **26** minutes.

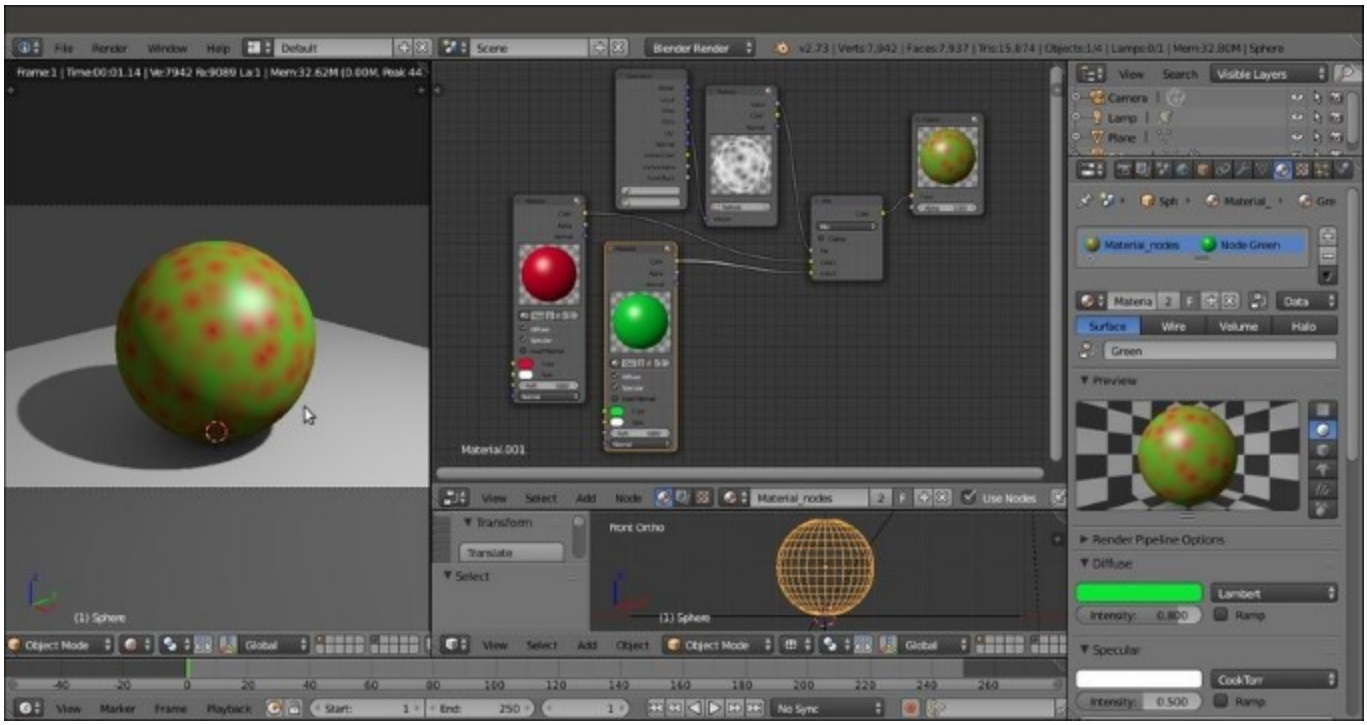
One other reason is that **Cycles'** normals baking capabilities are still not as good as in **Blender Internal** (at the moment, it bakes only the real geometry, contrary to **Blender Internal**, which can also bake the bump output of textures to normal maps), or that it's not as flexible for **Non-Photorealistic Rendering** (**NPR**) as the **Blender Render** engine.

Just a quick note: normally, materials under the **Blender Render** engine are created directly in the slots inside the **Material** window, often switching to the **Texture** window and back; in the following screenshot, you can see the **Rendered** preview of a generic `Red` *mono* material assigned to a **UV Sphere**:



A generic "mono" Blender Internal material

But, it's also possible to use node materials in **Blender Internal**, created and connected inside the **Node Editor** window; basically, let's say that *two or more materials can be mixed through nodes* to obtain more advanced results. In the following screenshot, for example, the mono `Red` material is mixed with a mono `Green` material through the output of a **Voronoi** texture connected to the **Fac** input socket of a **MixRGB** node:



Two mono materials mixed in the Node Editor window

This is the way we are going to create the **Blender Internal** shaders.

Building the reptile skin shaders in Blender Internal

Because we want to keep the materials we already created for **Cycles** in the same blend file (and the reason will be clear in the next chapter), before we start with the creation of the **Blender Internal** shaders, we must prepare the file a bit.

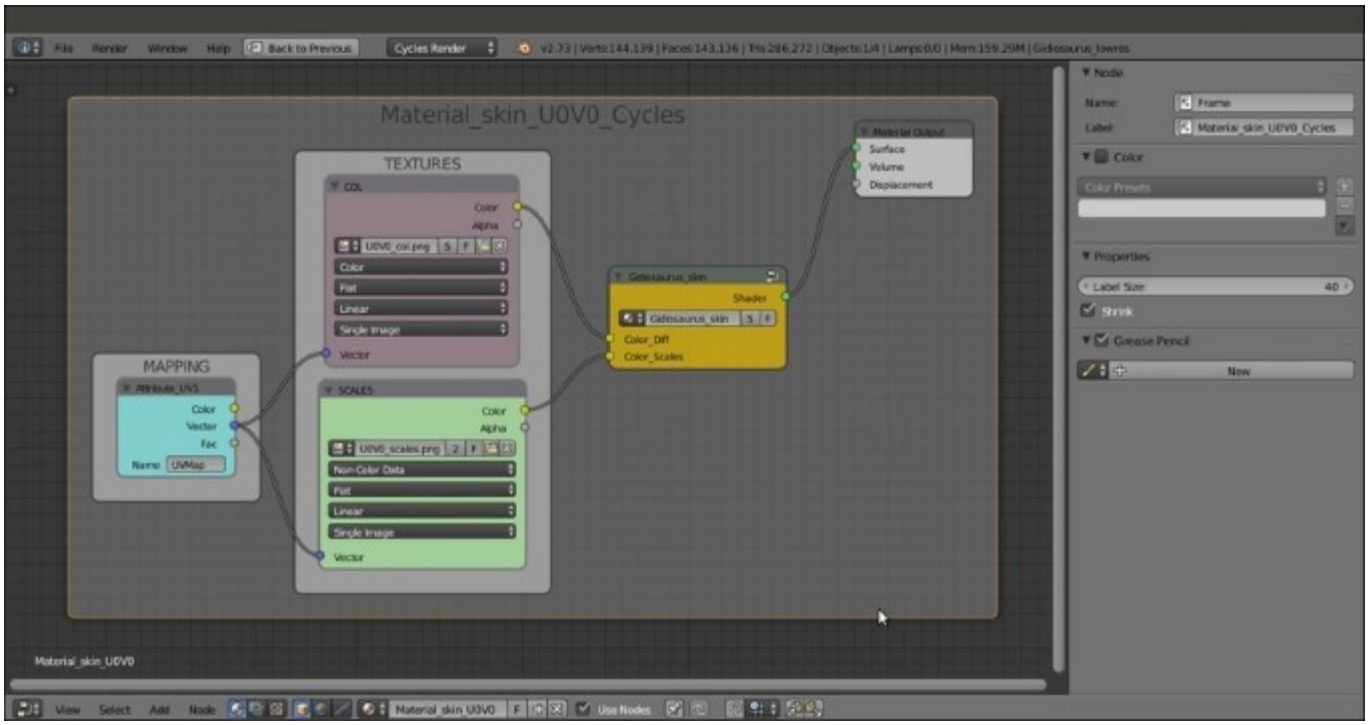
Getting ready

The first thing to do is to open the last saved blend file, add **Frames** to each material in the **Node Editor** window, and label them with the material name followed by the suffix **_Cycles**; this is to later distinguish them from the material we will build for **BI**.

Therefore:

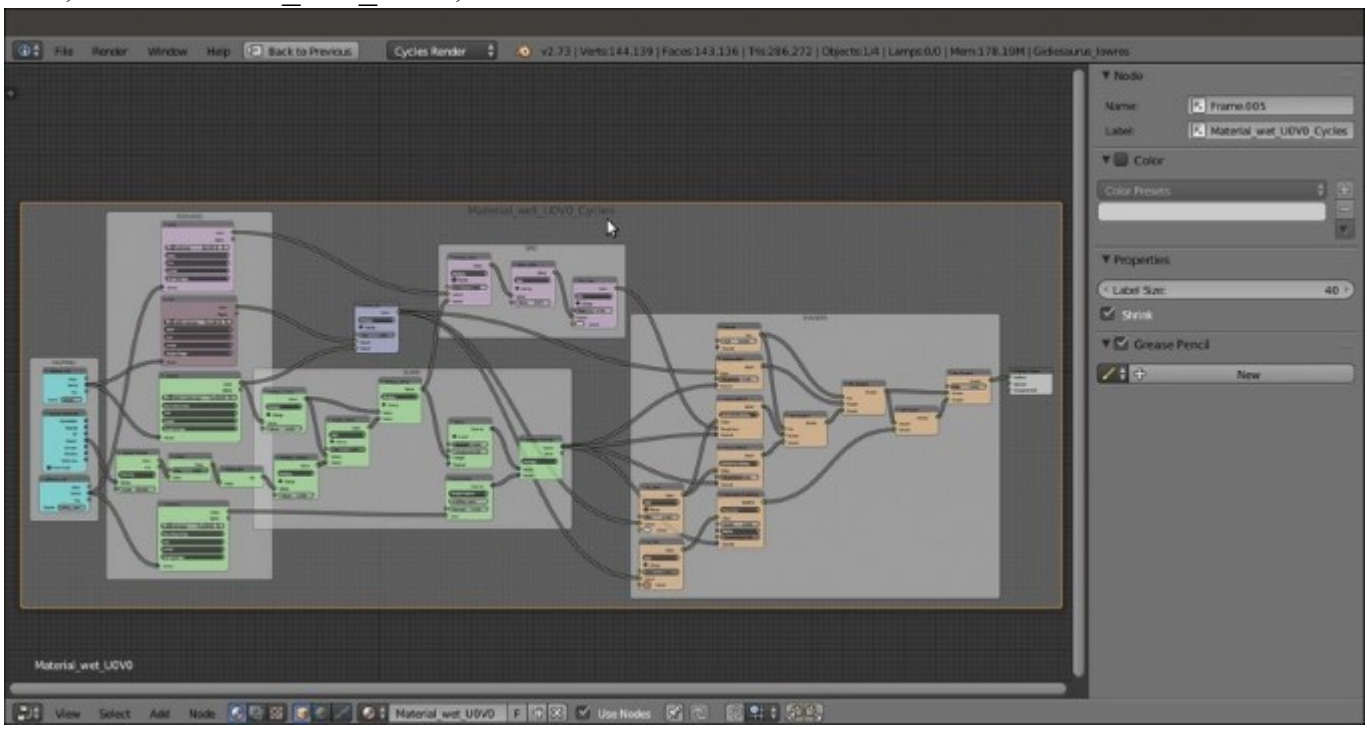
1. Start Blender and load the `Gidiosaurus_shaders_Cycles.blend` file.
2. In the **Outliner**, select the **Gidiosaurus_lowres** mesh, go to the **Material** window and click on the `Material_skin_U0V0` slot; put the mouse pointer inside the **Node Editor** window and press *Shift* + *A* to add a **Frame** (*Shift* + *A* | **Layout** | **Frame**).
3. Press *A* to select all the nodes (the added **Frame**, already selected, becomes the active one) and then press *Ctrl* + *P* to parent them to the active **Frame**.
4. Select only the **Frame** and press *N* to call the **Properties** sidepanel; in the **Label** slot under the **Name** subpanel, type **Material_skin_U0V0_Cycles**, then go down to the **Properties** subpanel and increase the **Label Size** to **40**.
5. Repeat the procedure for all the Cycles' **Gidiosaurus_lowres** materials, for the **Eyes** and **Corneas** and for the **Armor** materials.

   So, for example, the `Material_skin_U0V0`, in the **Node Editor** window, becomes this:



A "framed" Cycles material

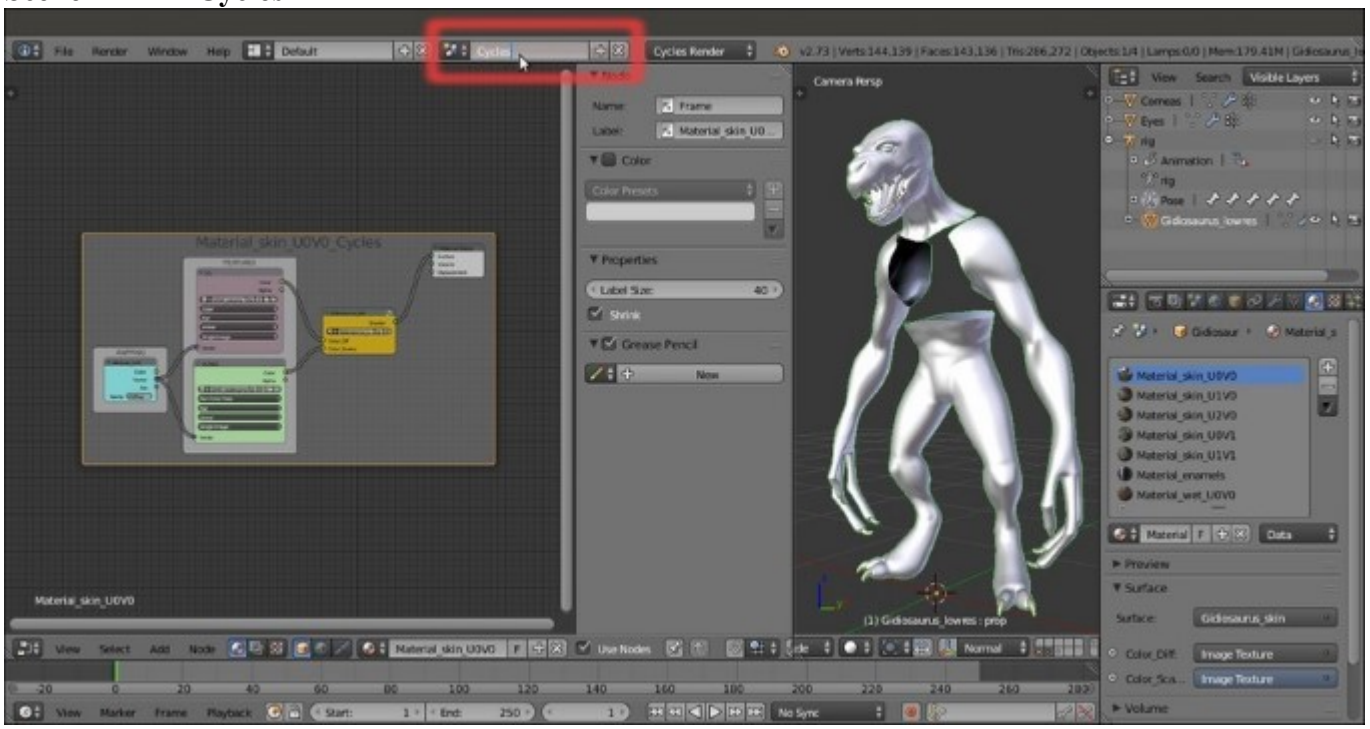
Also, the `Material_wet_U0V0`, becomes this:



Another "framed" Cycles material

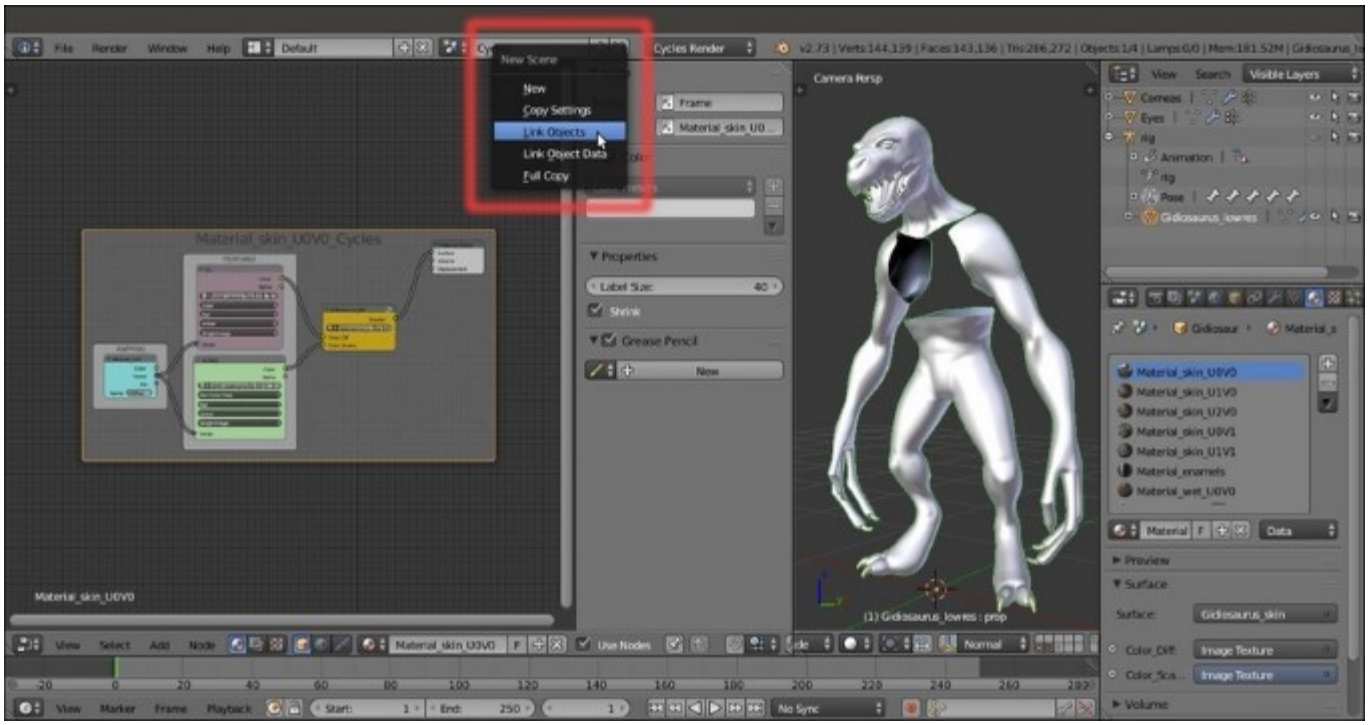
Note that the name of the material is the same as before, the only difference is that a **Frame** labeled with the **_Cycles** suffix has been added in the **Node Editor** window to visually group all the Cycles' nodes that are a constituent of the shader.

6. Now go to the *Scene data block* button on the main top header; left-click on it and rename the **Scene** label as **Cycles**:



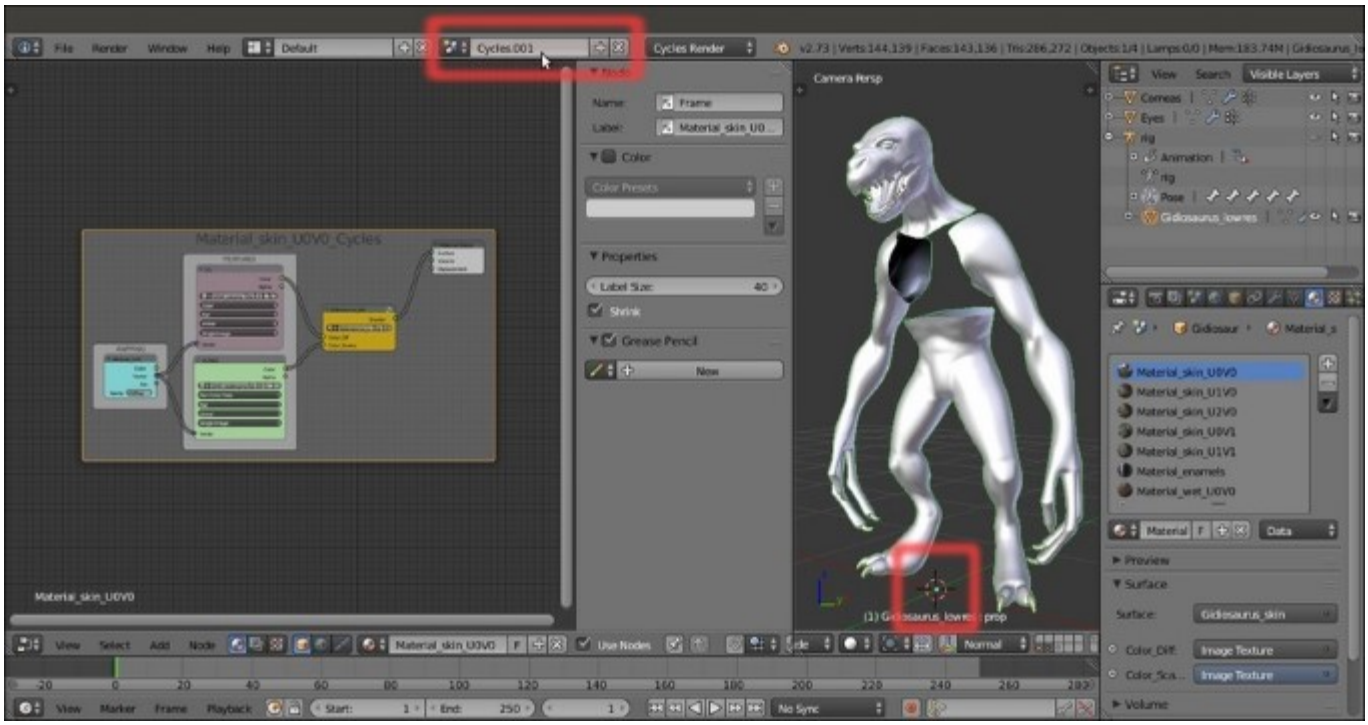
Renaming the Scene label

7. Click on the + icon button to the right of the datablock name; in the pop-up little **New Scene** panel, select the **Link Objects** item:



Adding a new scene with linked objects

At this point we have created a new scene (automatically labeled as **Cycles.001**) that is sharing the same objects of the other (**Cycles**) scene (be aware of this: the objects in one scene are not a copy of the others, *they are the same objects shared/linked between the two scenes*); you can say which objects are actually linked from one scene to another, by their blue pivot point (for example, look at the highlighted pivot point of the **Gidiosaurus_lowres** object in the following screenshot):

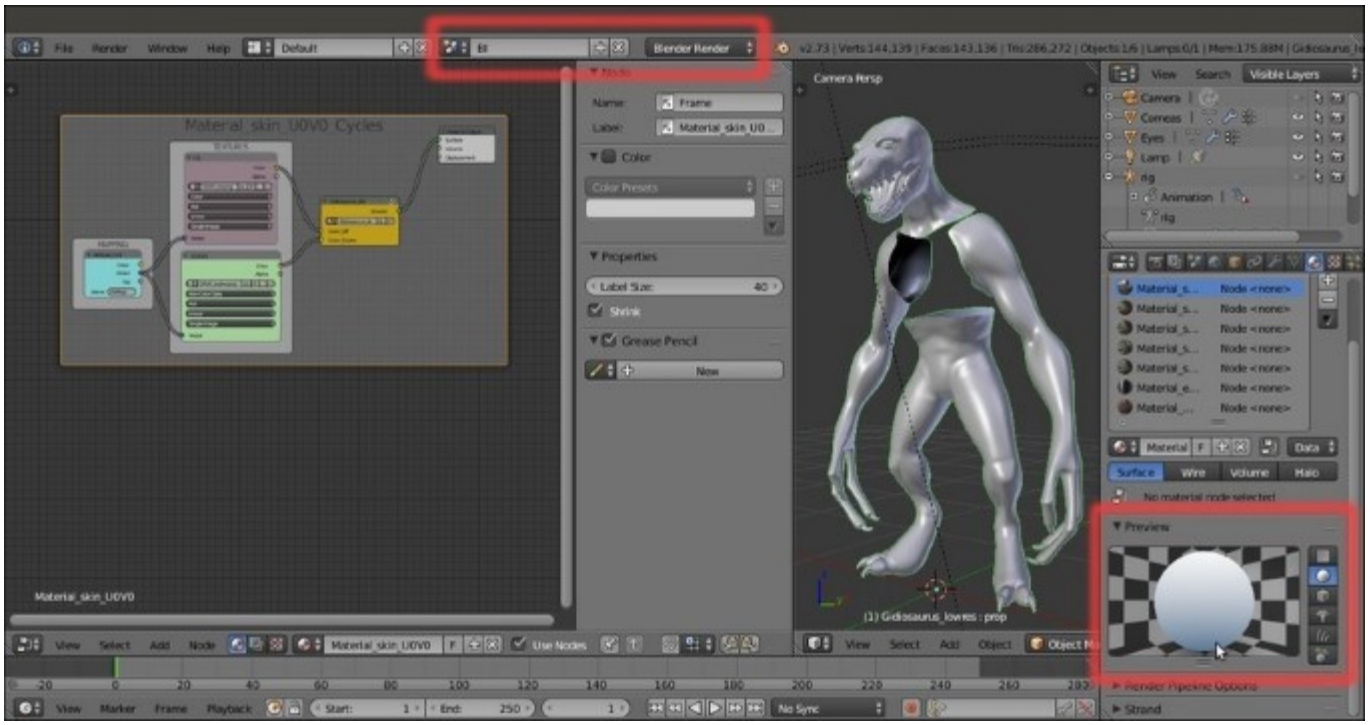


A new scene with linked objects

The advantages of creating new scenes with linked objects are obvious: we can have totally different rendering engines, or different worlds or lamps, in the different scenes and use the same objects and meshes data; so, for example, any modification to a linked object in one scene will automatically be transferred to the other scenes.

Furthermore, avoid duplicating the objects for each scene; this will help to keep a small file size.

8. Rename the scene from **Cycles.001** to **BI**, then move to the *Engine to use for rendering* button a bit to the right and switch from **Cycles Render** to **Blender Render**.



Switching to the Blender Render engine and the "empty material" preview

Note

Note that the **Preview** subpanel of the **Material** window shows an *empty material*, to point out that under the current **Blender Render** engine, the material slot, although filled with the **Cycles** material, doesn't have anything to render yet.

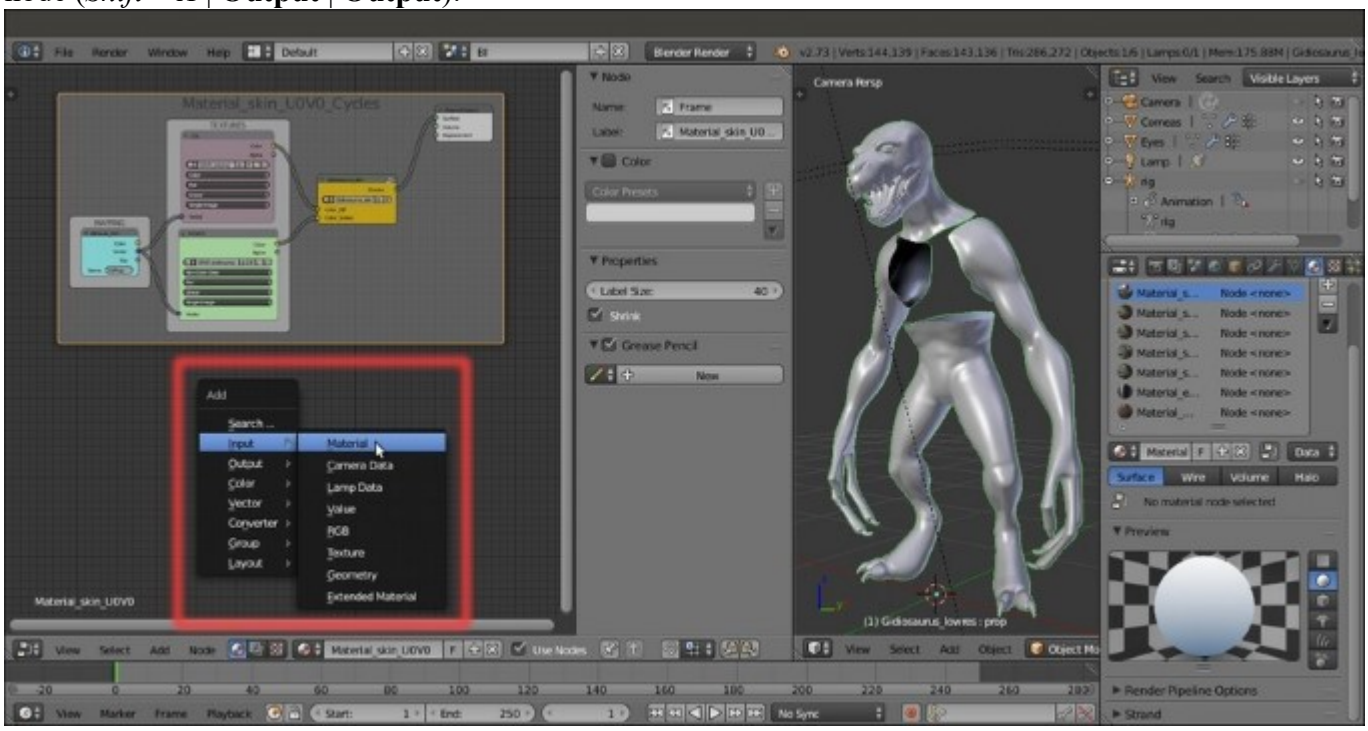
9. In the **Outliner**, select the **Lamp** (be sure to have enabled both the **11th** and the **6th** scene layers); go to the **Object Data** window, set the energy to **14.000** and the color to **R 1.000**, **G 1.000**, **B 0.650**; under the **Shadow** subpanel, enable the **Buffer Shadow** item, **Filter Type** to **Gauss**, **Soft** = **12.000**, **Size** = **4000**, and **Samples** = **16**. Set **Clip Start** = **9.000** and **Clip End** = **19.000**.
10. Go to the **World** window and enable the **Ambient Occlusion** by checking the item in the subpanel of the same name; leave the **Blend Mode** to **Add** and set the **Factor** to **0.35**.
11. Go further down to the **Gather** subpanel and click on the **Approximate** button: check the **Pixel Cache** item and then check also the **Falloff** checkbox under the **Attenuation** item; set the **Strength** to **0.900**.
12. Enable the **Indirect Lighting** item just above and set the **Factor** to **0.65**.

    These **World** settings are to obtain a sort of **Global Illumination** effect in the **Blender Render** engine; to learn more, have a look at http://www.blender.org/manual/render/blender_render/world/index.html.
13. Save the file as `Gidiosaurus_shaders_Blender_Internal.blend`.

How to do it…

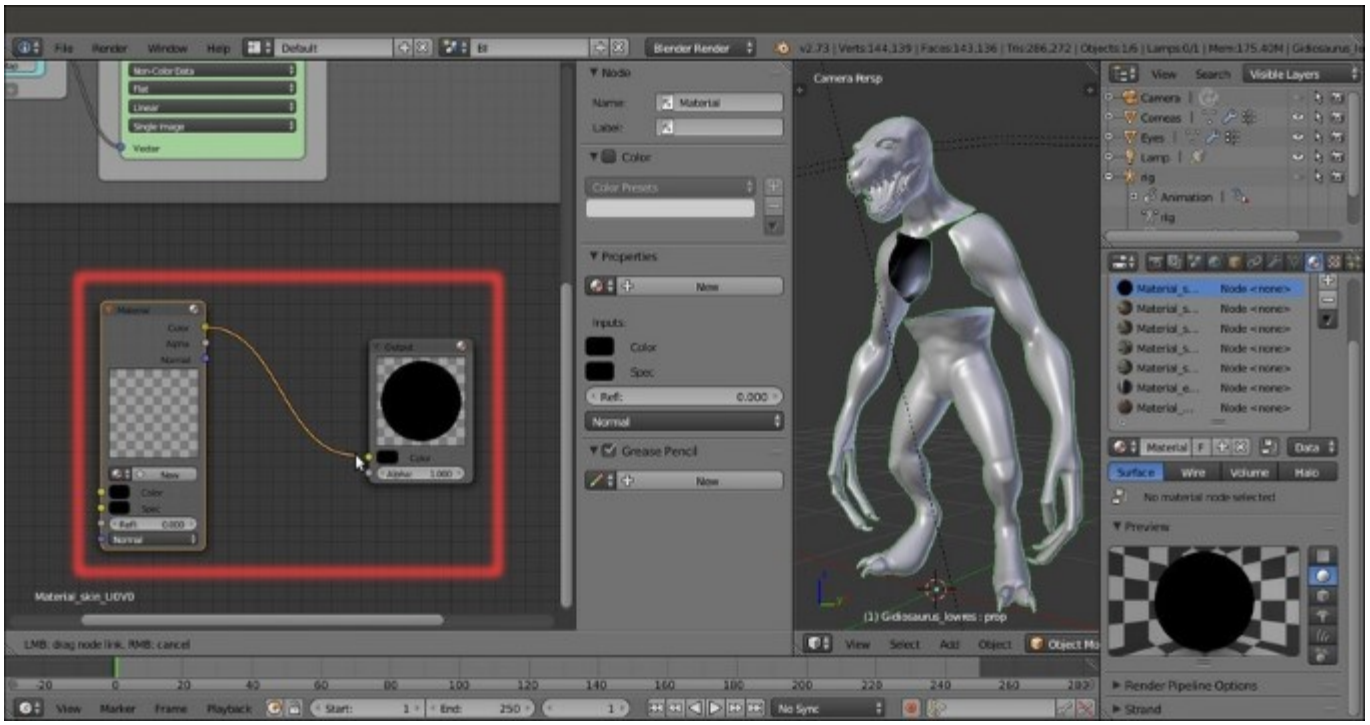
Let's start with the first top **Gidiosaurus** skin material, so:

1. Be sure to have the **Gidiosaurus_lowres** object selected and, back in the **Material** window, click on the `Material_skin_U0V0` slot.
2. Put the mouse pointer inside the **Node Editor** window and press *Shift* + *A* (*Shift* + *A* | **Input** | **Material**) to add a **Material** node to the window; then press again *Shift* + *A* and add an **Output** node (*Shift* + *A* | **Output** | **Output**):



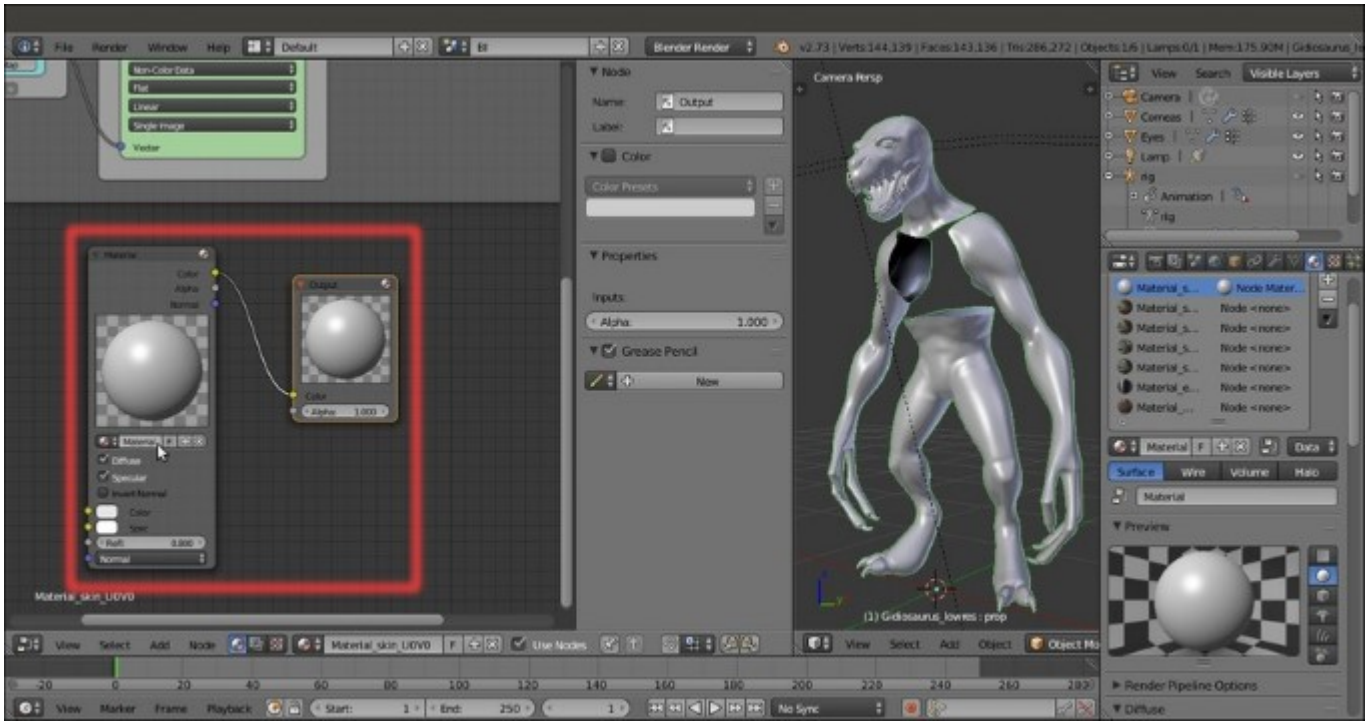
Adding a first material node in the Node Editor window

3. Connect the **Color** output of the **Material** node to the **Color** input socket of the **Output** node:



Connecting the material node to the output node

4. Now click on the **New** button on the **Material** node to create a new default **Blender Internal** material:



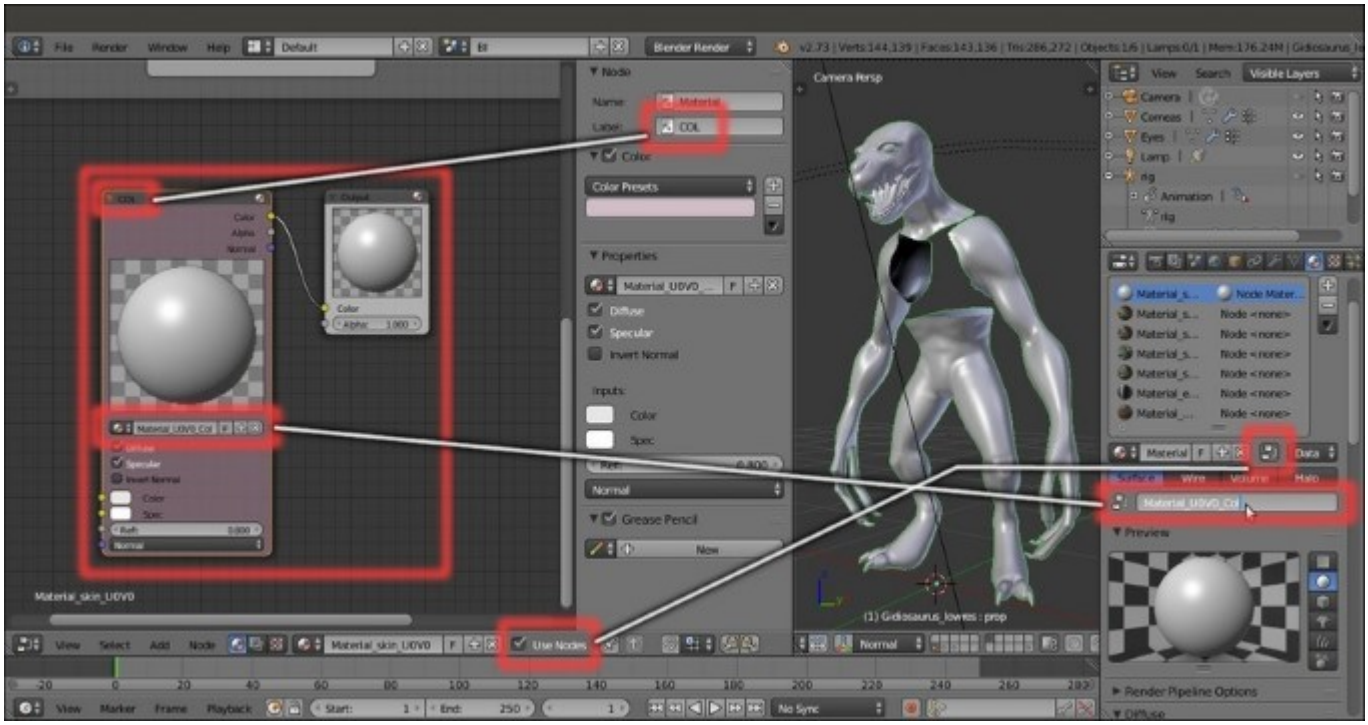
Creating a default "mono" material by clicking on the New button in the material node

5. In the **Properties** sidepanel of the **Node Editor** window (*N* key to call it) label the **Material** node as **COL** and assign a color.

   If you look now at the **Material** window, close to the right side of the material datablock (the name of the material), there is an already enabled and squared button with the symbol of the nodes.

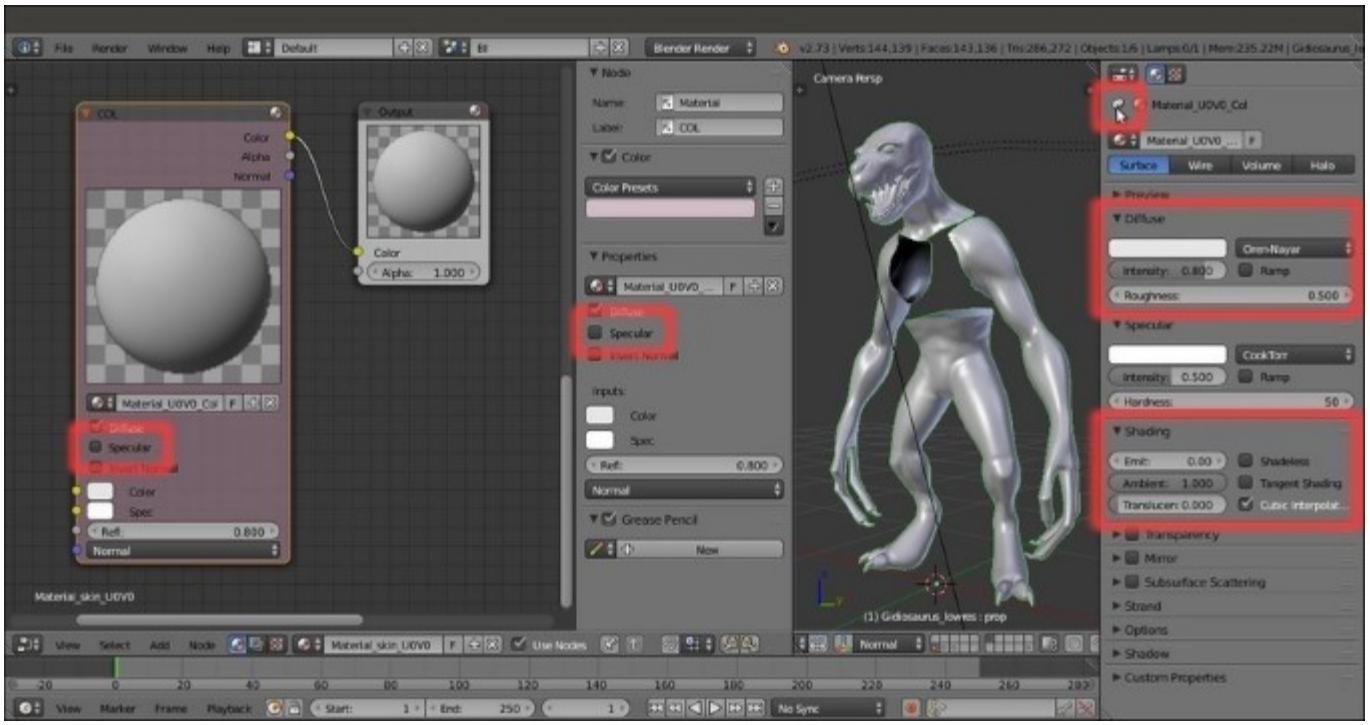
   In our case, that button is already enabled because we are already using material nodes; because it's enabled, a second material datablock slot has appeared just further down: that's the datablock slot for any node selected inside the **Node Editor** window and that is part of a material node.

   The purpose of this second datablock slot is to let us know which material is the selected one and we are therefore going to edit it by tweaking all the values in the subpanels below.
6. Go to the **Material** window to find the second material name slot: rename the material selected in the **Node Editor** window as `Material_U0V0_Col`; you can do the same thing by clicking on the name datablock on the **COL** node interface.



The corresponding datablock slots in the node interface and in the Material window

7. In the **Node Editor** window, or in the *N* **Properties** sidepanel, deselect the **Specular** item.
8. Go to the top of the **Material** window and click on the pin icon to the left of the contest; by doing this only the selected material is shown in the window.
9. Go to the **Diffuse** sidepanel and click on the **Diffuse Shader Model** button to select the **Oren-Nayar** item; then go down to the **Shading** subpanel and enable the **Cubic Interpolation** item:

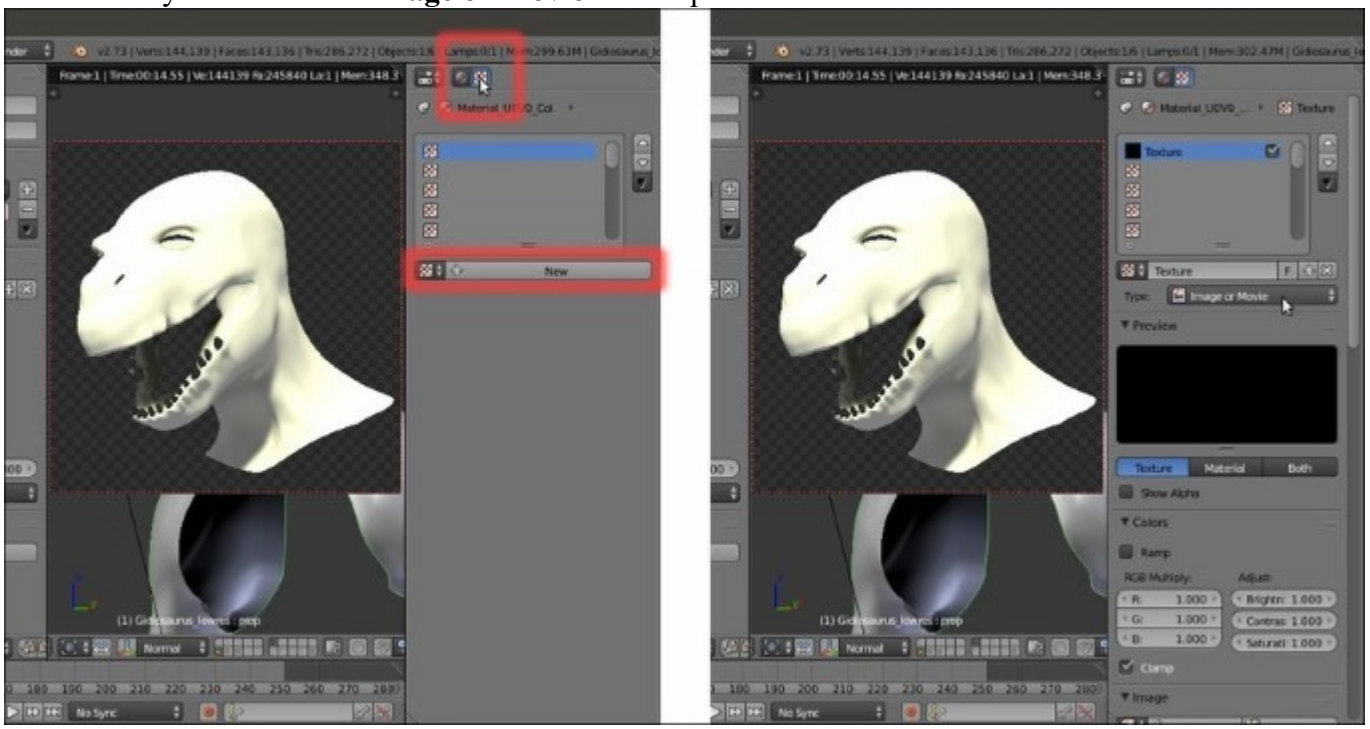


The Specular item to be disabled in the Node Editor window and the shader's parameters to be tweaked in the Material window

10. At this point, press *Shift* + *B* to draw a box around the character's head in the **Camera** view and then zoom to it. If your computer is powerful enough to allow you to work without slowing down, put the mouse pointer inside the 3D viewport and press *Shift* + *Z* to start the **Rendered** preview; in any case, you can easily enable or disable the preview every time you need it:

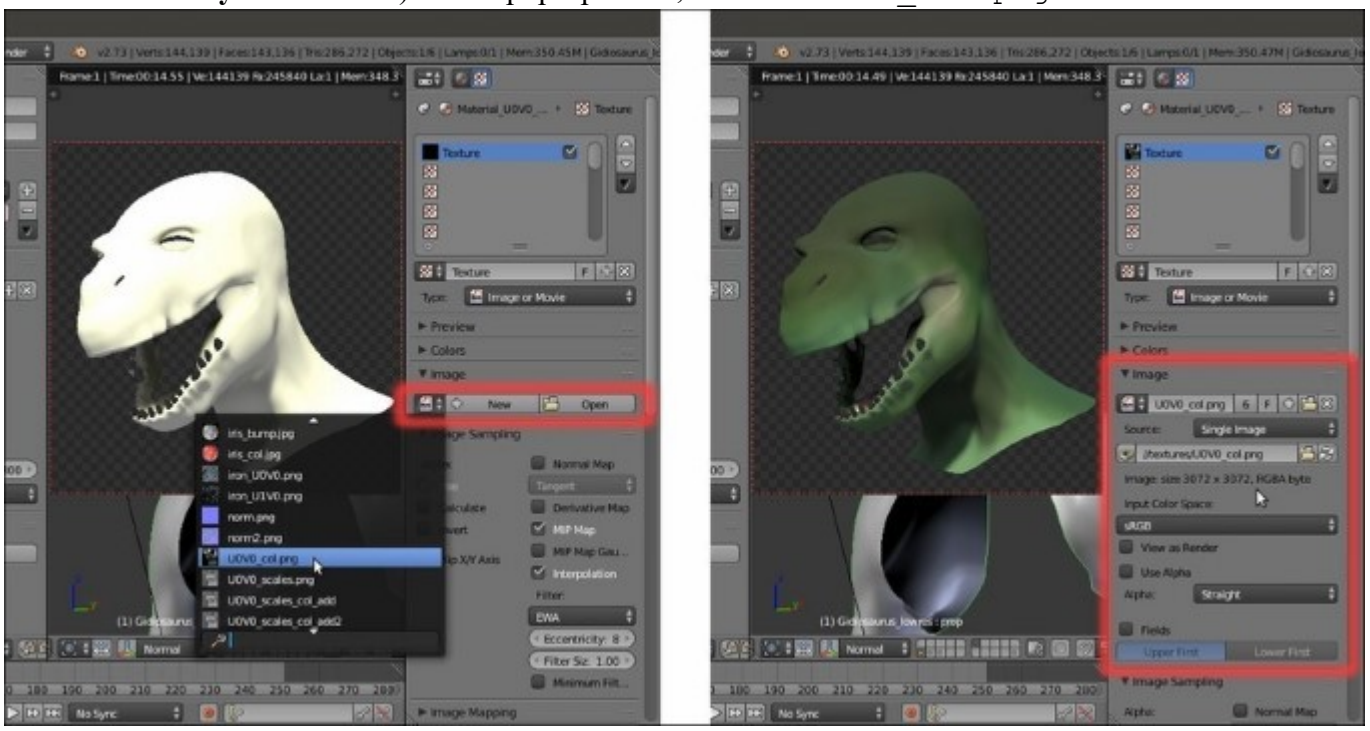
Cropping and starting the rendered preview

11. Click on the **Texture** window icon at the top right of the main **Properties** panel, just above the contest, be sure to have the **first** top texture slot selected and click on the **New** button to automatically load a default **Image or Movie** texture panel:



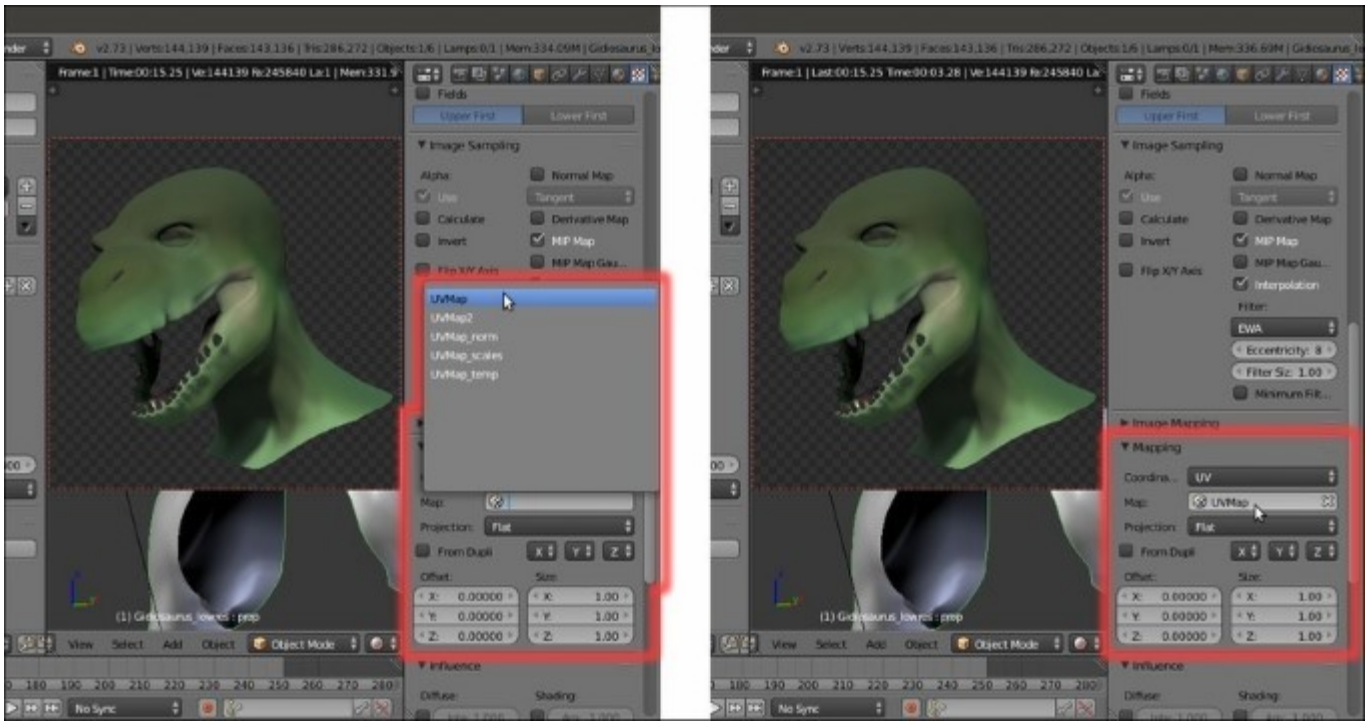
Adding a first texture slot to the material

12. Collapse the **Preview** and the **Colors** subpanels, which at this moment we don't need, and click on the double little arrows to the left side of the **New**/**Open** buttons in the **Image** subpanel (remember that we have already loaded inside the blend file all the image textures we need, because of the **Cycles** shaders!): in the pop-up menu, select the `U0V0_col.png` item:



Selecting the right image texture from the drop-down list

13. Go further down to find the **Mapping** subpanel: be sure to have the **Coordinates** set to **UV**, the **Projection** to **Flat** (default settings) and click on the **Map** empty slot to select the **UVMap** item.



Selecting the right UV coordinates mapping

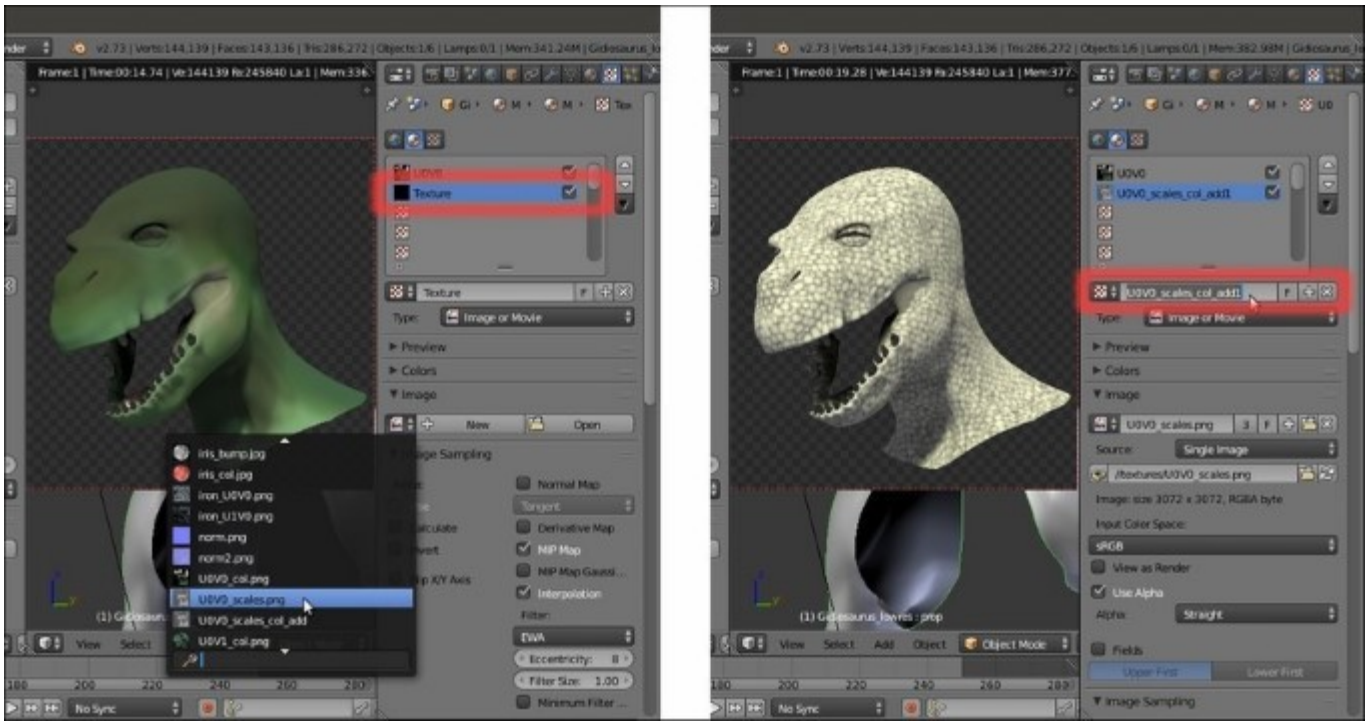
14. Go even further down to find the **Influence** subpanel: be sure that the diffuse **Color** channel is the one enabled and that the slider is set to **1.000** (again, default settings):

The Influence settings subpanel for the texture

15. Scroll back to the top of the **Texture** window and click on the *Unique datablock ID name* slot, where the generic **Texture** name is written; rename it as `U0V0` (as you can see in the following screenshot, this is the name that also appears in the textures list window):

Renaming the texture datablock

16. Now click on the empty **second** slot: again, click on the **New** button, click on the double arrows and this time load the image `U0V0_scales.png`.
17. In the *Unique datablock ID name* slot, rename it as `U0V0_scales_col_add1`.

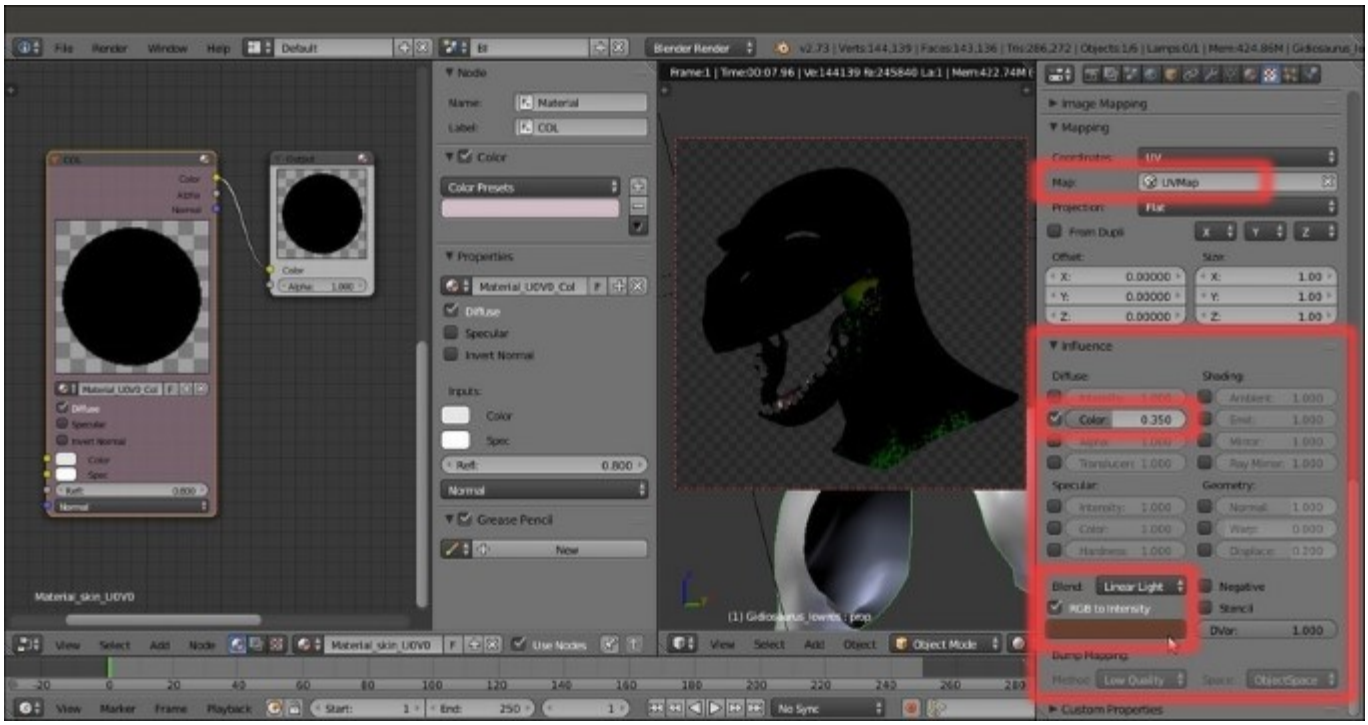


Adding a new texture slot, loading a new image texture and renaming it accordingly

Note

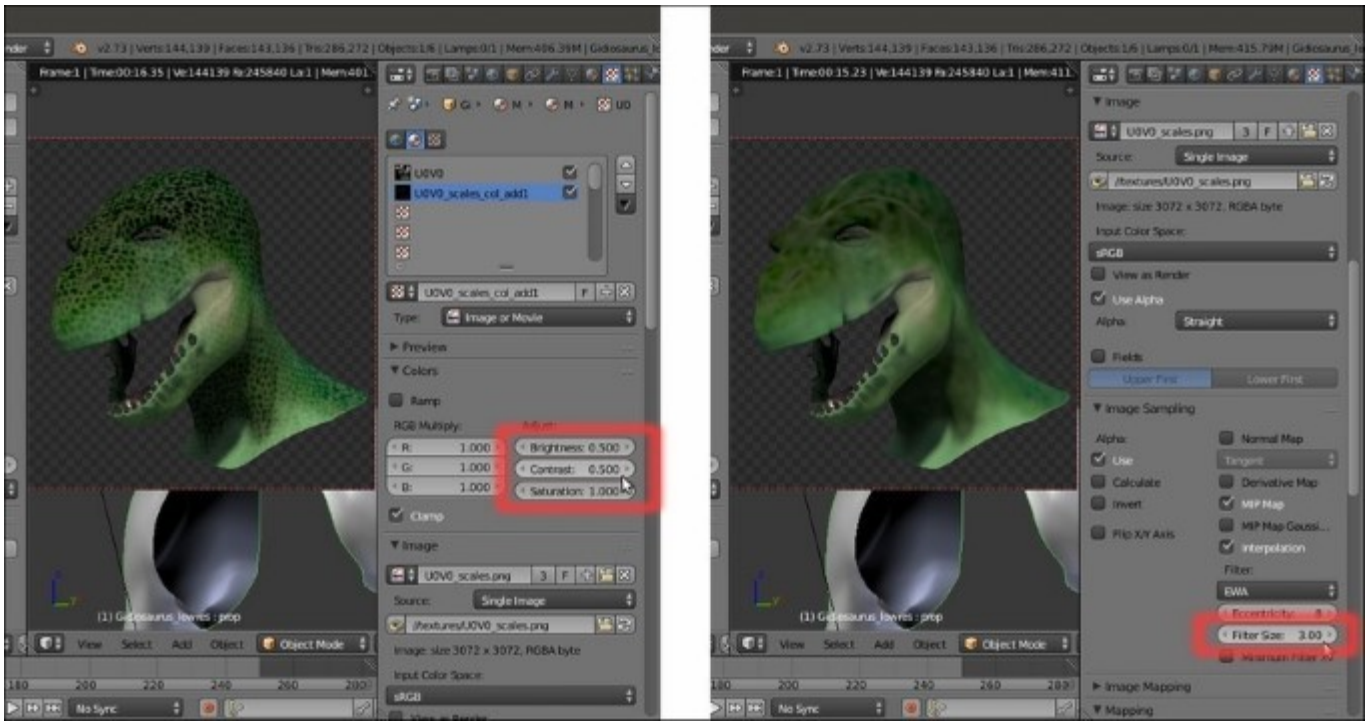
Note that as we load the `U0V0_scales.png` image in the **second** texture slot, the **Rendered** preview changes to show the grayscale image mapped on the model; this is because, by default, the **Influence** of any new added texture is set to the **Color** channel with a value **1.000** and **Blend Type** to **Mix**.

18. In the **Input Color Space** slot under the **Image** subpanel, change the default **sRGB** to **Non-Color**; then, scroll down to the **Mapping** panel to set the **UVMap** coordinates item and then go to the **Influence** subpanel: leave the **Color** channel enabled but move the slider to the lower value of **0.350**, then change the **Blend Type** to **Linear Light** (for the **Blender Internal** materials, the **Blend Type** works as the layer system of a 2D image editor such as **Photoshop** or **Gimp**); enable the **RGB to Intensity** item and change the pink color to **R 0.130**, **G 0.051**, **B 0.030**.



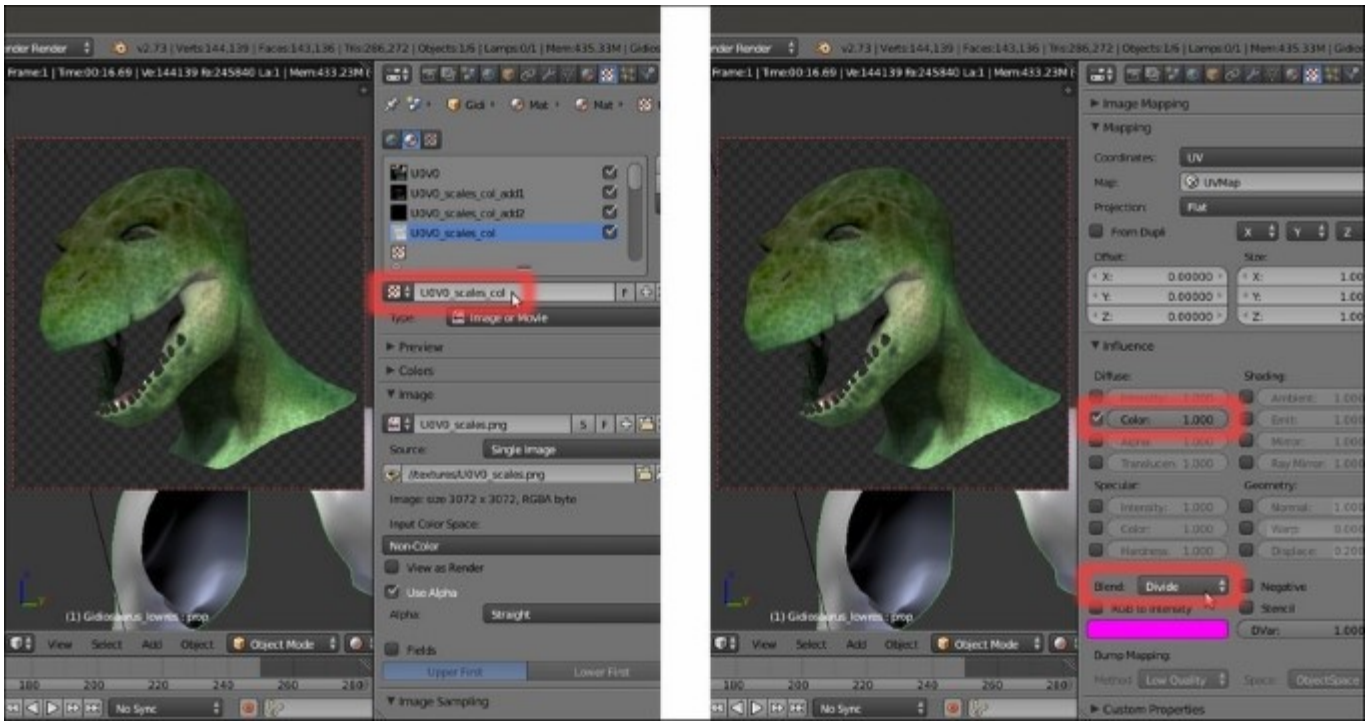
Tweaking the Influence settings for the second texture

19. Go up to expand the **Colors** subpanel: set the **Brightness** and the **Contrast** to **0.500**, to make the texture less bright and less contrasted. Go down to the **Image Sampling** subpanel and set the **Filter Size** to **3.00**, to blur the image (values beyond **1.00** start to blur the image more and more):



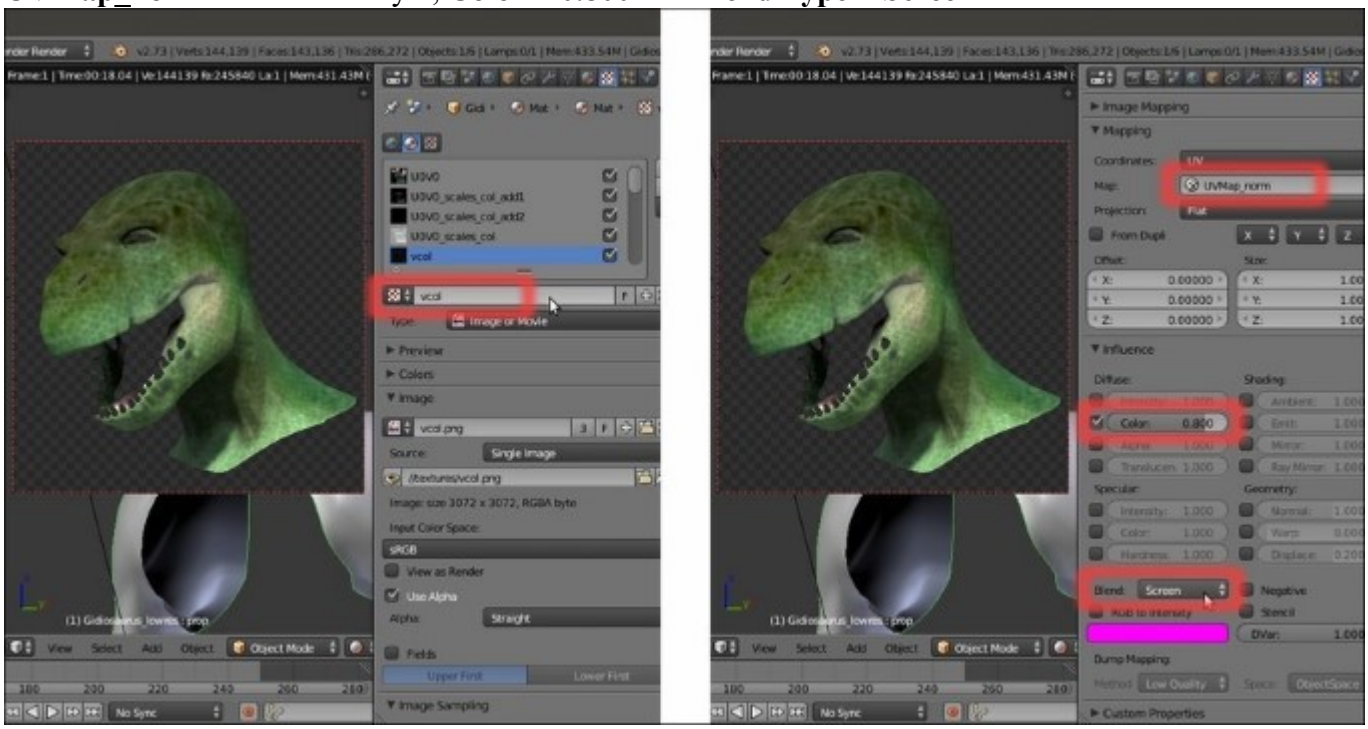
Modifying the appearance of the second image texture

20. Select the **third** empty texture slot and repeat the procedure, again loading the `U0V0_scales.png` image; in the *Unique datablock ID name* slot, rename it as `U0V0_scales_col_add2`.
21. Scroll down to the **Mapping** panel to set the **UVMap** coordinates item and then in the **Influence** subpanel, leave the **Color** channel enabled at value **1.000** but change the **Blend Type** to **Subtract**. Set the **Brightness** to **0.100** and the **Contrast** to **1.500**. Again, set the **Filter Size** to **3.00**.
22. Select the **fourth** texture slot, load again the `U0V0_scales.png` image, rename it `U0V0_scales_col`, set the **UVMap** coordinates layer, **Color** = **1.000** and **Blend Type** = **Divide**:



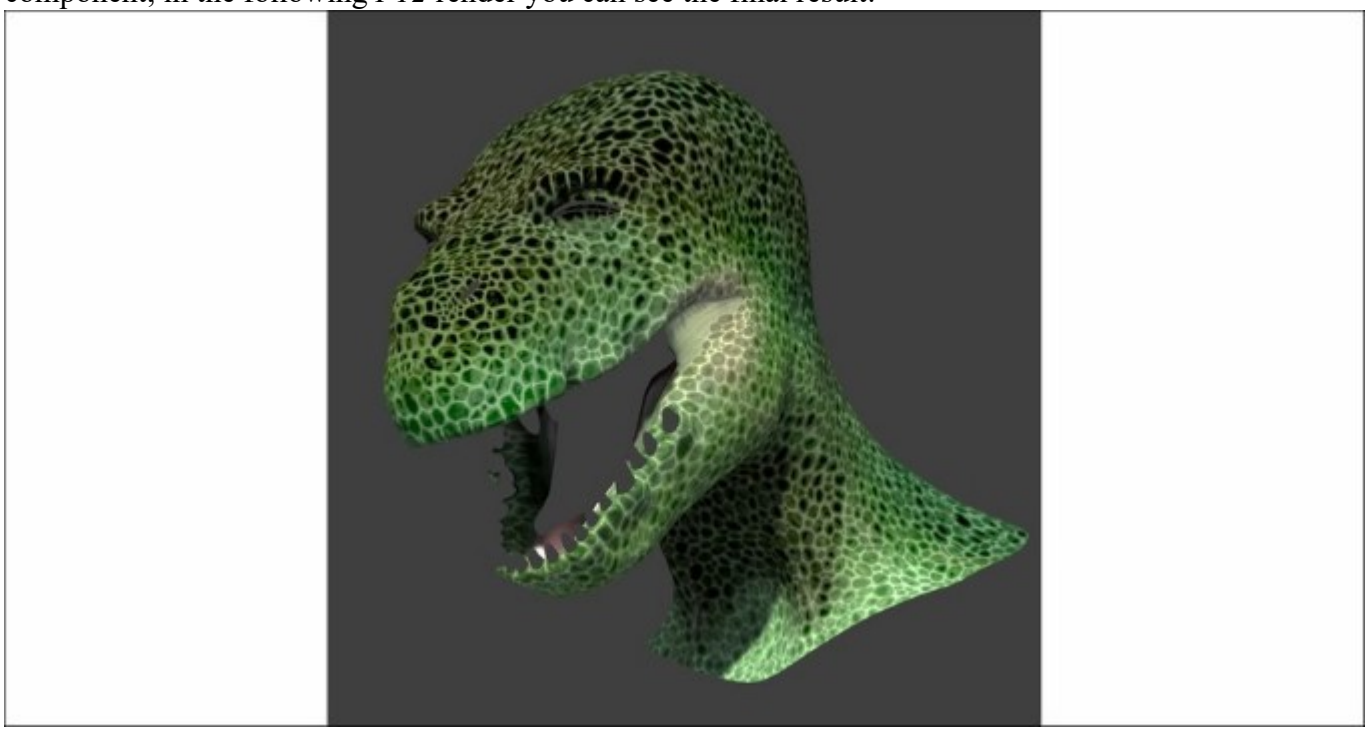
Adding more texture slots with different settings

23. Select the **fifth** texture slot, load the `vcol.png` image again, rename it `vcol`, set the **UVMap_norm** coordinates layer, **Color** = **0.800** and **Blend Type** = **Screen**:



Adding the baked Vertex Color image texture as well

At this point we have completed the **first** component of the skin shader, that is, the **diffuse color** component; in the following *F12* render you can see the final result:



The completed diffuse color component of the Blender Internal skin material

Note that this *F12* render result is quite different from the **Rendered** real-time preview; this is probably due to the complexity of using several textures inside a node material system with the (sadly) bad real-time viewport performances of Blender.

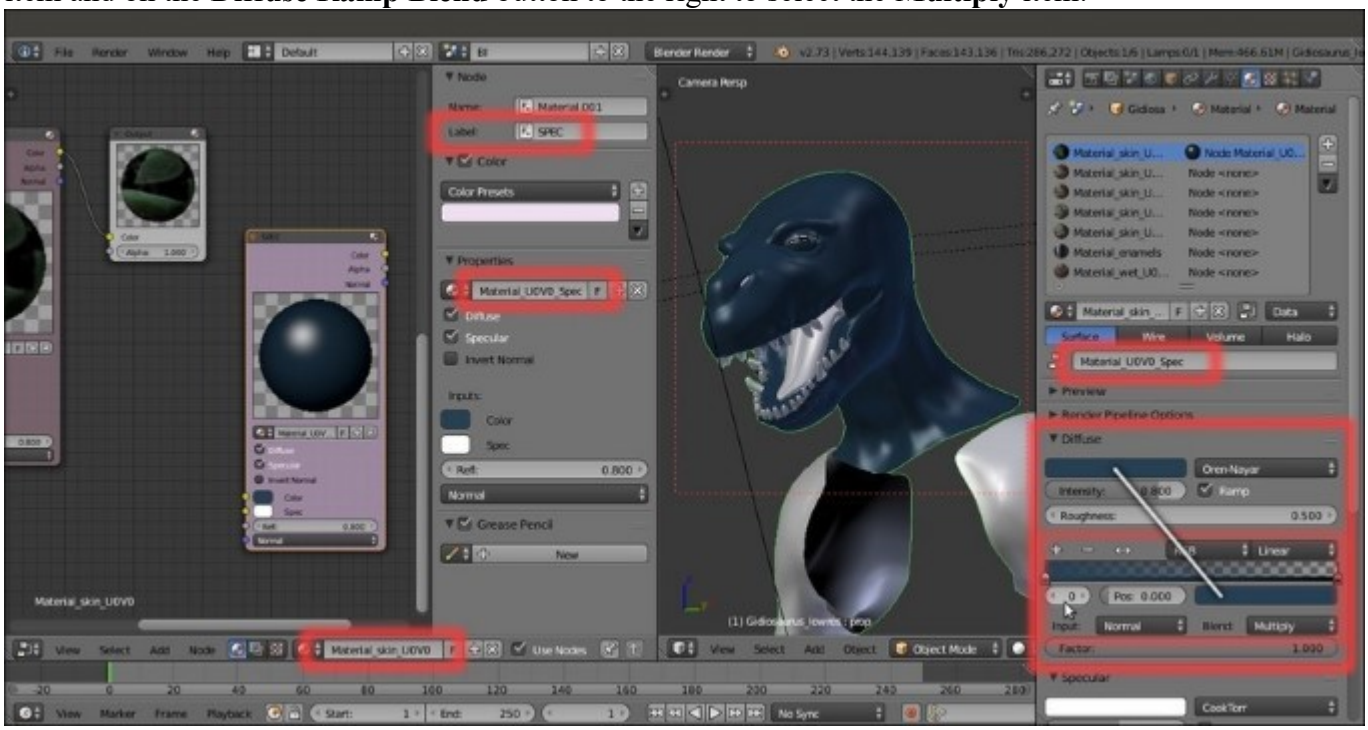
Note

Also note that the only parts of the **Gidiosaurus** mesh that appear in the rendered image are actually the parts we assigned a **Blender Internal** material to; in fact, the **teeth** and the **tongue** are rendered as blank shapes (even working as a mask).

Now, we can carry on with building the second component of the shader, the **glossy** component.

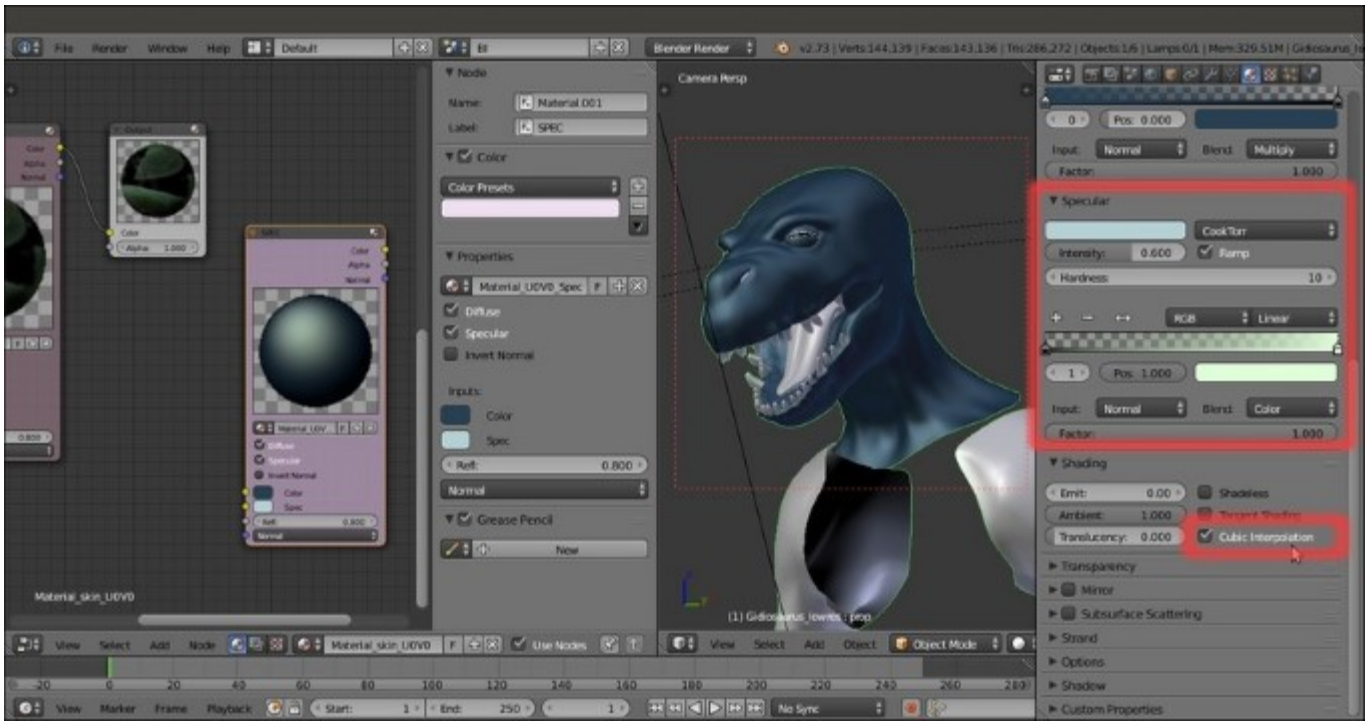
24. Put the mouse pointer inside the **Node Editor** window and add a new **Material** node (*Shift* + *A* | **Input** | **Material**); label it as **SPEC** and then click on the **New** button to create a new material: rename it `Material_U0V0_Spec`.
25. Go to the **Material** window; in the **Diffuse** sidepanel, change the shader model to **Oren-Nayar**, then change the color to a deep blue **R 0.020**, **G 0.051**, **B 0.089**.

26. Enable the **Ramp** item: in the slider, switch the positions of the two color stops (that is: white color stop to position **0.000** and black color stop to position **1.000**), then select the white color stop; put the mouse on the deep blue color slot of the **Diffuse** subpanel and press *Ctrl + C* to copy it; put the mouse pointer on the color slot of the selected color stop and press *Ctrl + V* to paste the deep blue color.
27. Click on the **Diffuse Ramp Input** button at the bottom of the subpanel to select the **Normal** item and on the **Diffuse Ramp Blend** button to the right to select the **Multiply** item:



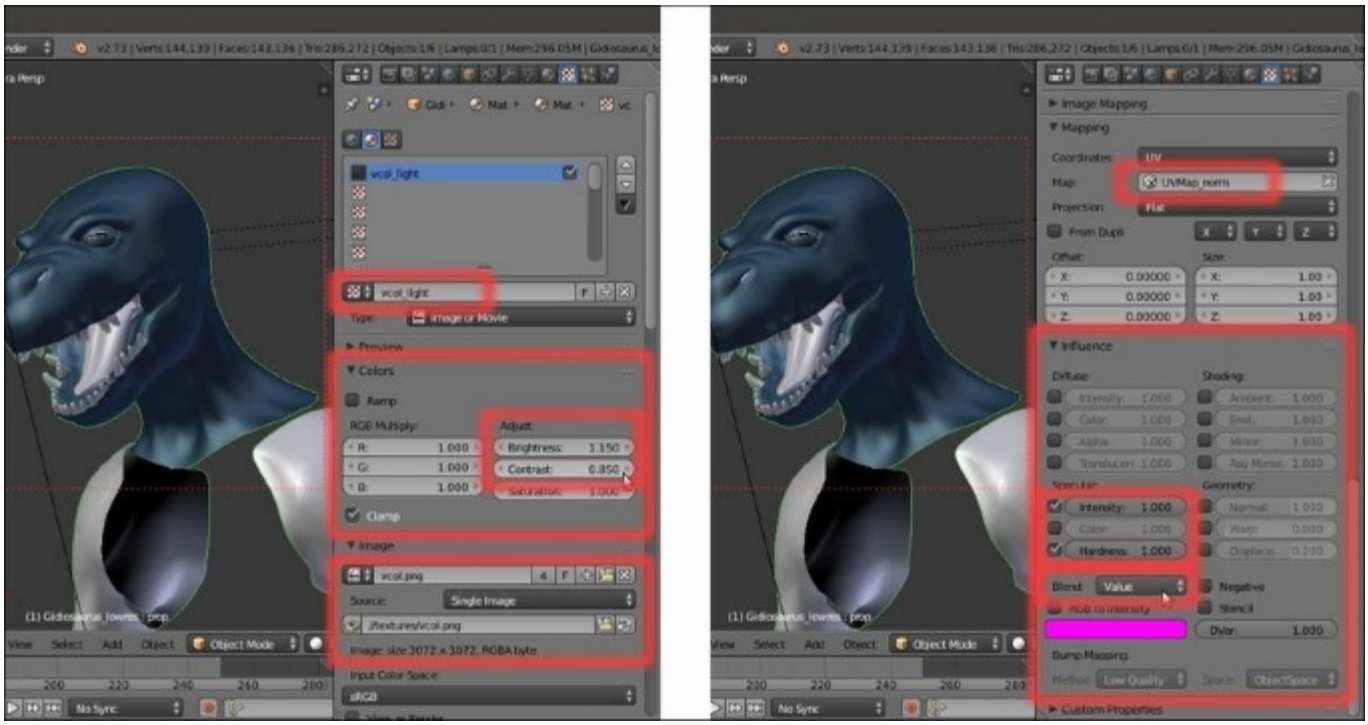
The "Material_U0V0_Spec", to be used inside the "Material_skin_U0V0" node material

28. Scroll down to the **Specular** subpanel: change the color to a light blue **R 0.474**, **G 0.642**, **B 0.683**; set the **Intensity** to **0.600** and the **Hardness** to **10**.
29. Enable the **Ramp** item: select the white color stop and change the color to **R 0.761**, **G 1.000**, **B 0.708**, then set the **Diffuse Ramp Input** button to **Normal** and the **Diffuse Ramp Blend** to **Color**.
30. Go to the **Shading** subpanel and enable the **Cubic Interpolation** item:



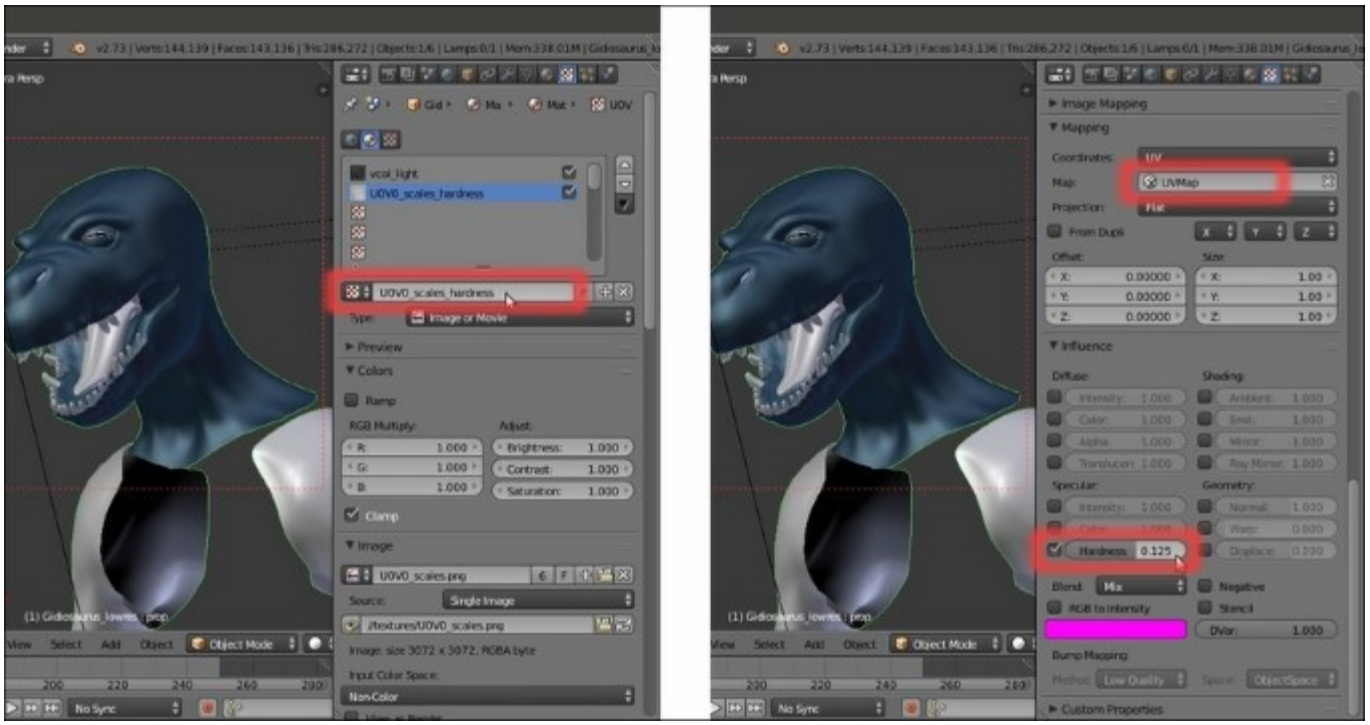
Setting the parameters of the specularity component

31. Go to the **Textures** window; select the top **first** empty texture slot and click on the **New** button. Load the image `vcol.png`, rename the ID datablock as `vcol_light` and go to the **Colors** subpanel: set the **Brightness** to **1.150** and the **Contrast** to **0.850**. Go down to the **Mapping** subpanel and set the **UVMap_norm** coordinates layer, then in the **Influence** subpanel disable the diffuse **Color** channel and enable both the **Intensity** and the **Hardness** channels under **Specular**; set the **Blend Type** to **Value**:



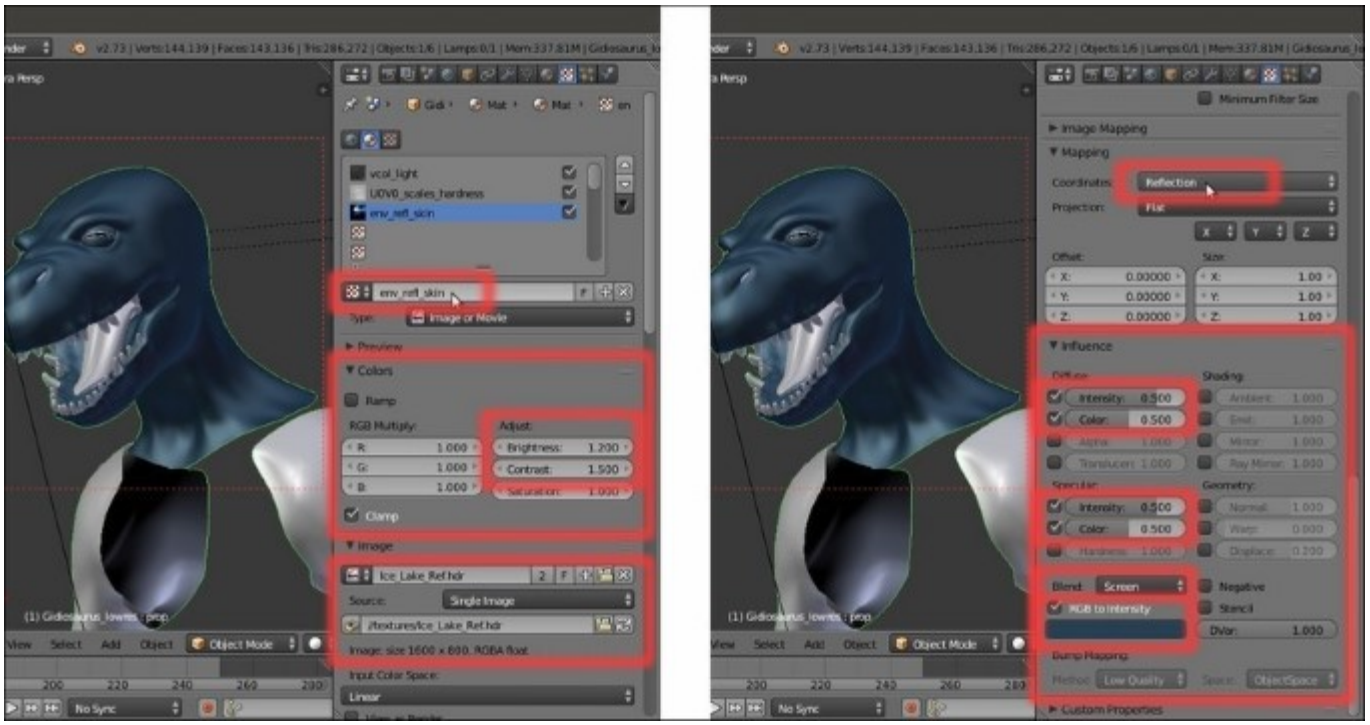
The settings for the specularity first texture

32. Go to the **second** slot and load the image `U0V0_scales.png`; rename it as `U0V0_scales_hardness`, in the **Mapping** subpanel, set the **UVMap** coordinates layer, in the **Influence** subpanel disable the diffuse **Color** and enable the **Hardness** channel under **Specular** to **0.125**. In the **Image Sampling** subpanel set the **Filter Size** to **5.00**.



Re-using the "U0V0_scales.png" image texture for the specularity hardness

33. In the **third** slot load the image `Ice_Lake_Ref.hdr`, a free high dynamic range image licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 License from the **sIBL Archive** (http://www.hdrlabs.com/sibl/archive.html); there is a reason we are now using the **hdr** image, and it's explained in the *How it works…* section.
34. Rename the image ID datablock as `env_refl_skin` and in the **Colors** subpanel, set the **Brightness** to **1.200** and the **Contrast** to **1.500**; go to the **Mapping** subpanel and set the **Texture Coordinates** to **Reflection**. Down in the **Influence** subpanel, enable both the **Intensity** channel under **Diffuse** and **Specular** and set their sliders to **0.500**; enable also, the **Color** channel under **Specular** and set the sliders of both the **Color** channels to **0.500** as well. Set the **Blend Type** to **Screen**, enable the **RGB to Intensity** item and set the color to the same deep blue of the diffuse color (**R 0.020**, **G 0.051**, **B 0.089**):



Using the environment hdr image as reflection map

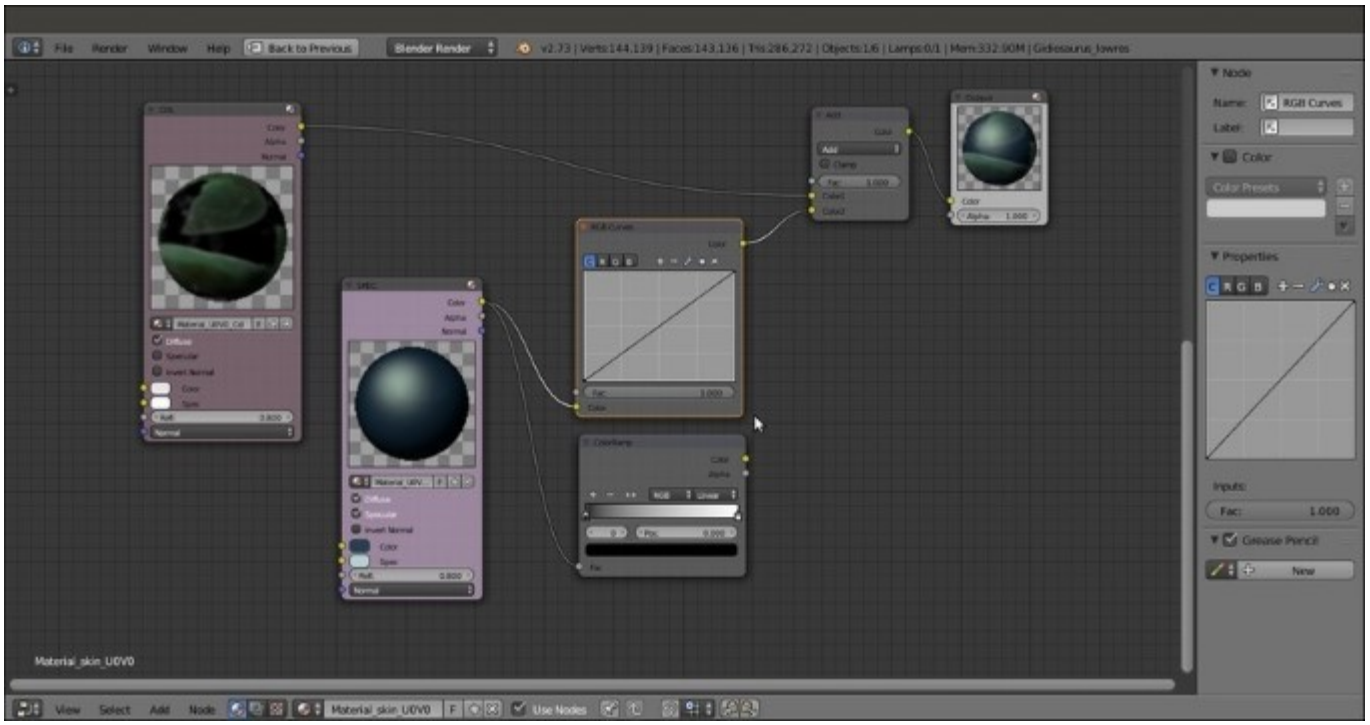
If you want to see the effect of the single components in the **Rendered** preview as we build the shader, just temporarily disconnect the **COL** node link to the **Output** node and replace it with the **Color** output of the **SPEC** node (in this case):

Testing the specularity component material in the rendered preview

35. At this point, add a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**) and move it on the link connecting the **COL** node to the **Output** node, to automatically paste it between them; then connect the **Color** output of the **SPEC** node to the **Color2** input socket of the **MixRGB** node, set the **Blend Type** of this latter node to **Add** and its **Fac** value to **1.000**:

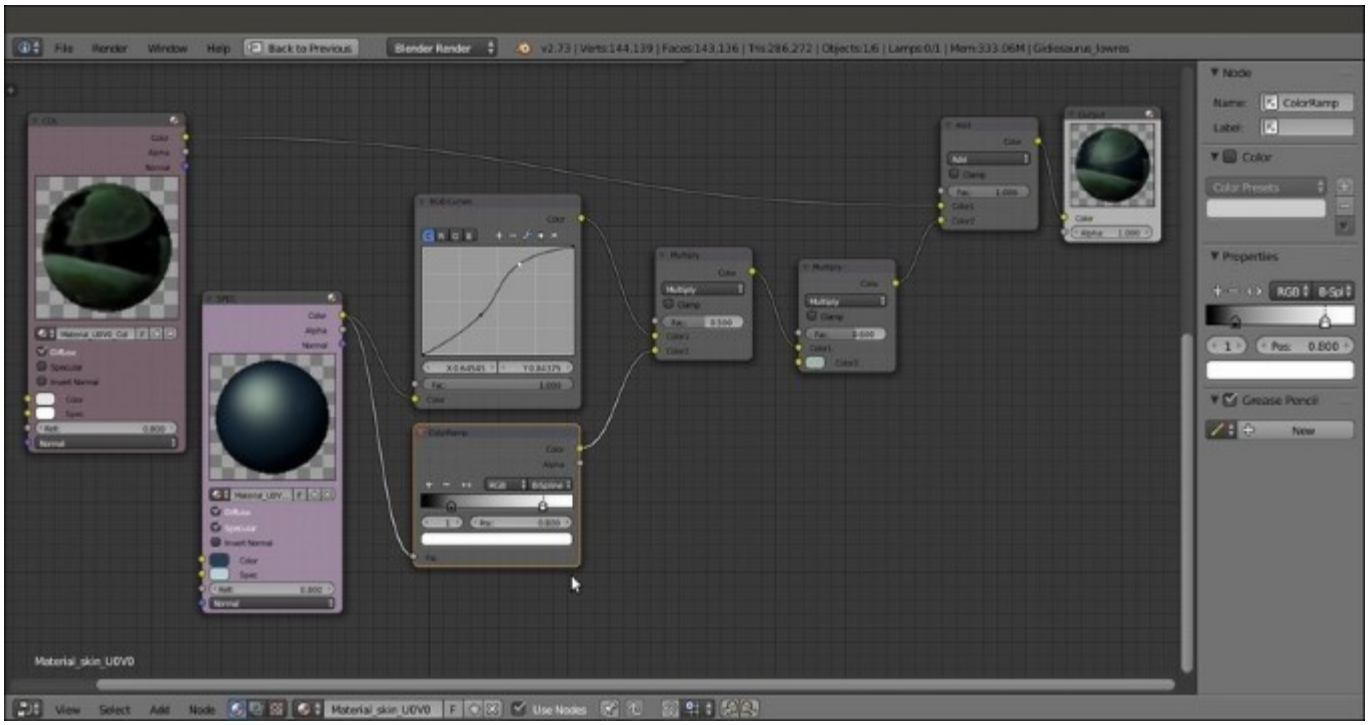
Finally adding the specularity component to the diffuse component

36. Add a **RGB Curves** node (*Shift* + *A* | **Color** | **RGB Curves**) and a **ColorRamp** node (*Shift* + *A* | **Converter** | **ColorRamp**); paste the **RGB Curves** node between the **SPEC** and the **MixRGB** node, then connect the **Color** output of the **SPEC** node also to the **ColorRamp** input socket:



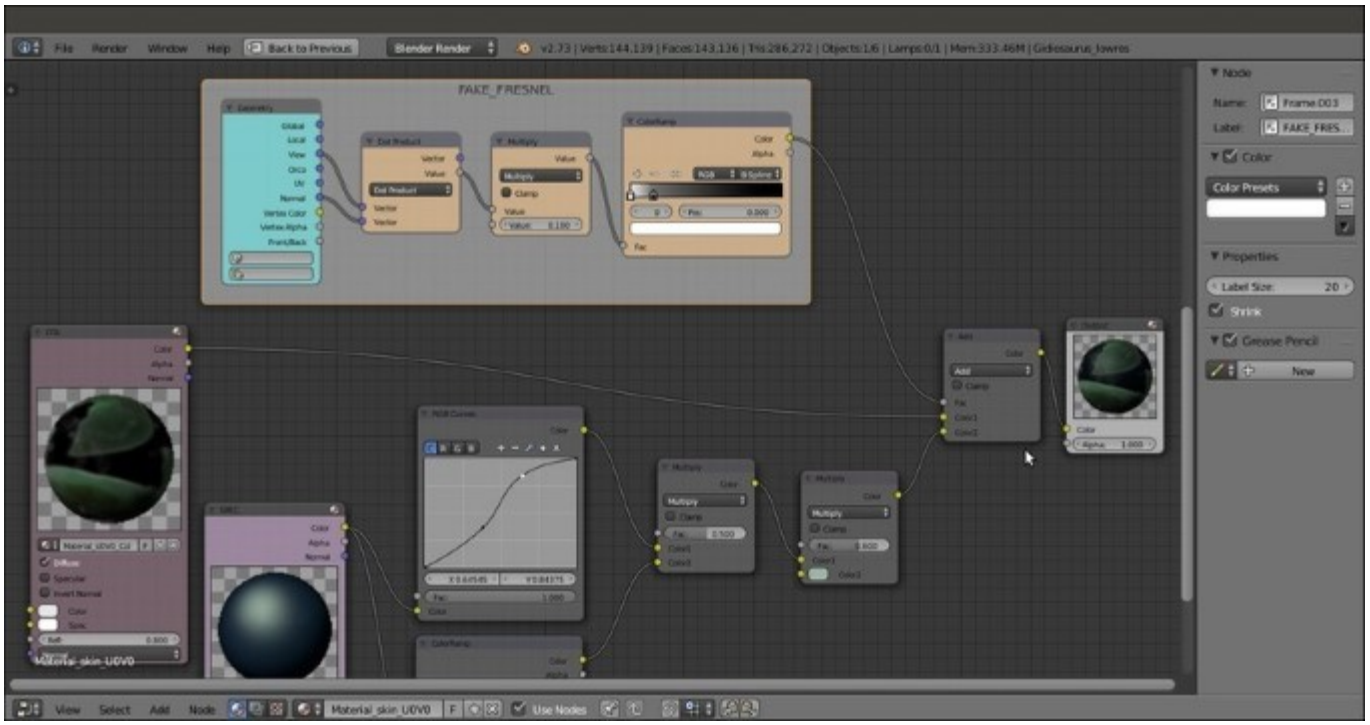
Adding new nodes

37. Press *Shift* + *D* to duplicate the **MixRGB** node, change the **Blend Type** of the duplicate to **Multiply** and paste it between the **RGB Curves** and the first **Add-MixRGB** nodes: set the **Fac** value to **0.500**. Connect the **Color** output of the **ColorRamp** node to the **Color2** input socket of the **Multiply-MixRGB** node.
38. Press *Shift* + *D* to duplicate the **Multiply-MixRGB** node and paste the duplicate between the first **Multiply-MixRGB** node and the **Add-MixRGB** node. Set the **Fac** value of the last **Multiply-MixRGB** node to **0.600** and the **Color2** to **R 0.347**, **G 0.462**, **B 0.386**.
39. Go to the **RGB Curves** node and left-click inside the interface window to add a point; set its coordinates to **X** = **0.38636** and **Y** = **0.36875**. Add a second point and set its coordinates to **X** = **0.64545** and **Y** = **0.84375**.
40. Go to the **ColorRamp** node and set the **Interpolation** to **B-Spline**, then move the black color stop to position **0.195** and the white color stop to position **0.800**:



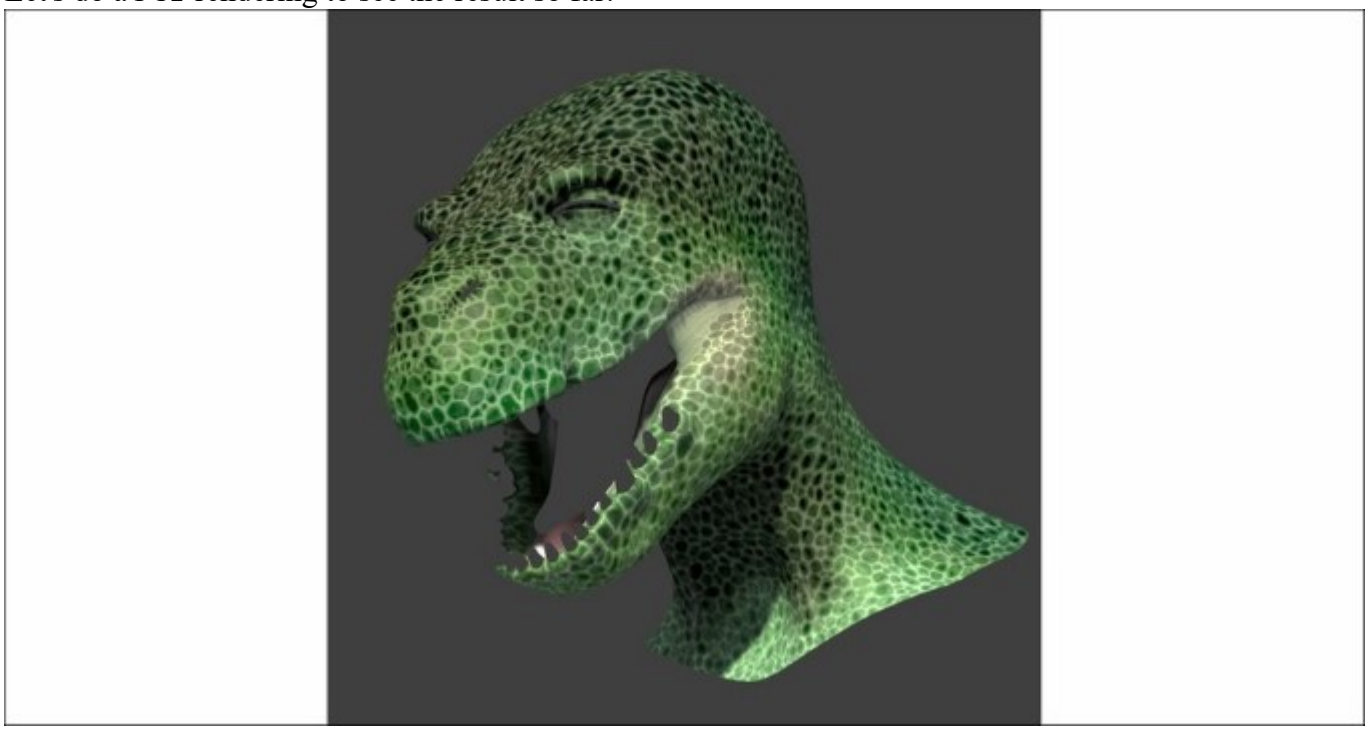
Tweaking the specularity component through the new nodes

41. Add a **Geometry** node (*Shift* + *A* | **Input** | **Geometry**), a **Vector Math** node (*Shift* + *A* | **Converter** | **Vector Math**), a **Math** node (*Shift* + *A* | **Converter** | **Math**), and a **ColorRamp** node (*Shift* + *A* | **Converter** | **ColorRamp**).
42. Add a **Frame** (*Shift* + *A* | **Layout** | **Frame**), label it as **FAKE_FRESNEL** and parent the last **four** added nodes to it.
43. Set the **Operation** of the **Vector Math** node to **Dot Product**, then connect the **View** output of the **Geometry** node to the first **Vector** input socket of the **Vector Math** node, and the **Normal** output of the **Geometry** node to the second **Vector** input socket of the **Vector Math** node.
44. Connect the **Value** output of the **Dot Product** node to the first **Value** input socket of the **Math** node; set the **Operation** of this latter node to **Multiply** and the second **Value** to **0.100**.
45. Connect the output of the **Math** node to the **Fac** input socket of the **ColorRamp** node; set the **Interpolation** of this latter node to **B-Spline**, then move the black color stop to position **0.150** and the white color stop to position **0.000**.
46. Connect the **Color** output of the **ColorRamp** node to the **Fac** input socket of the **Add-MixRGB** node:



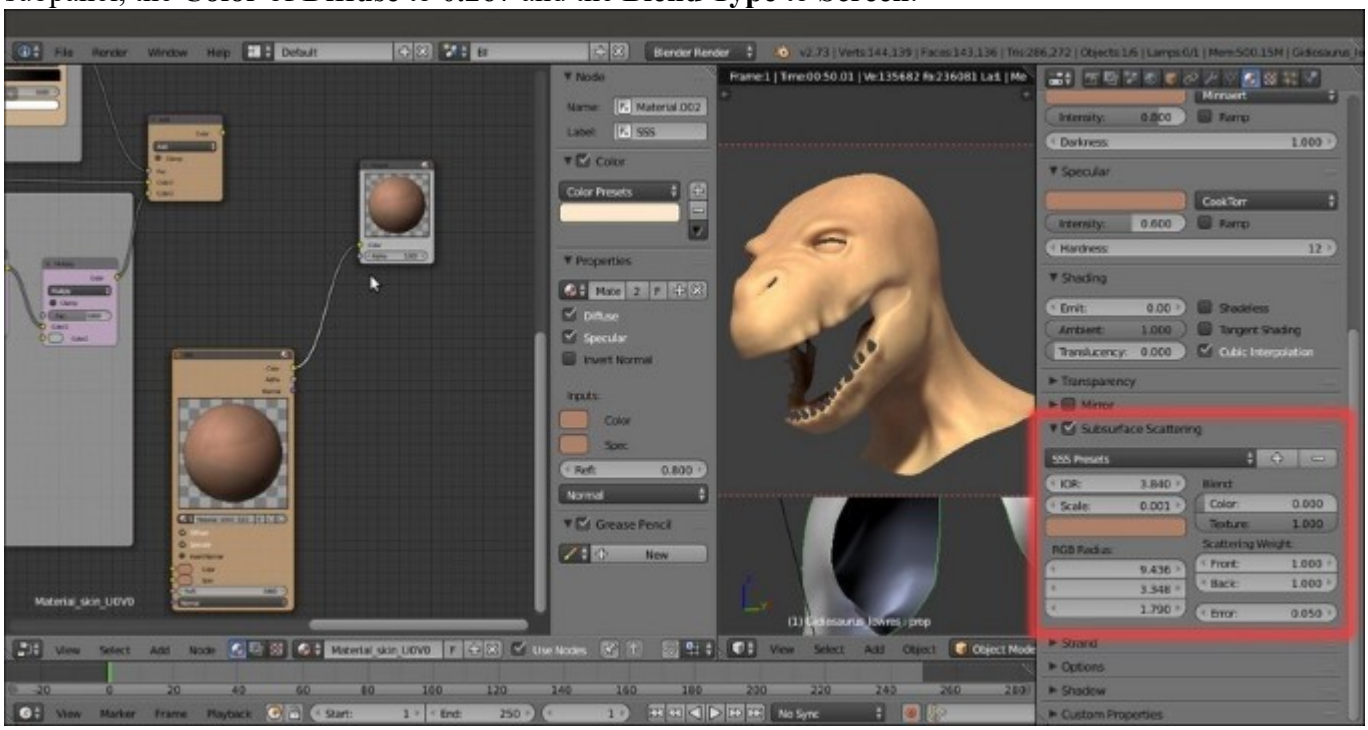
Adding the output of a fake Fresnel as factor for the blending of the two components

Let's do a *F12* rendering to see the result so far:



The F12 rendered result so far

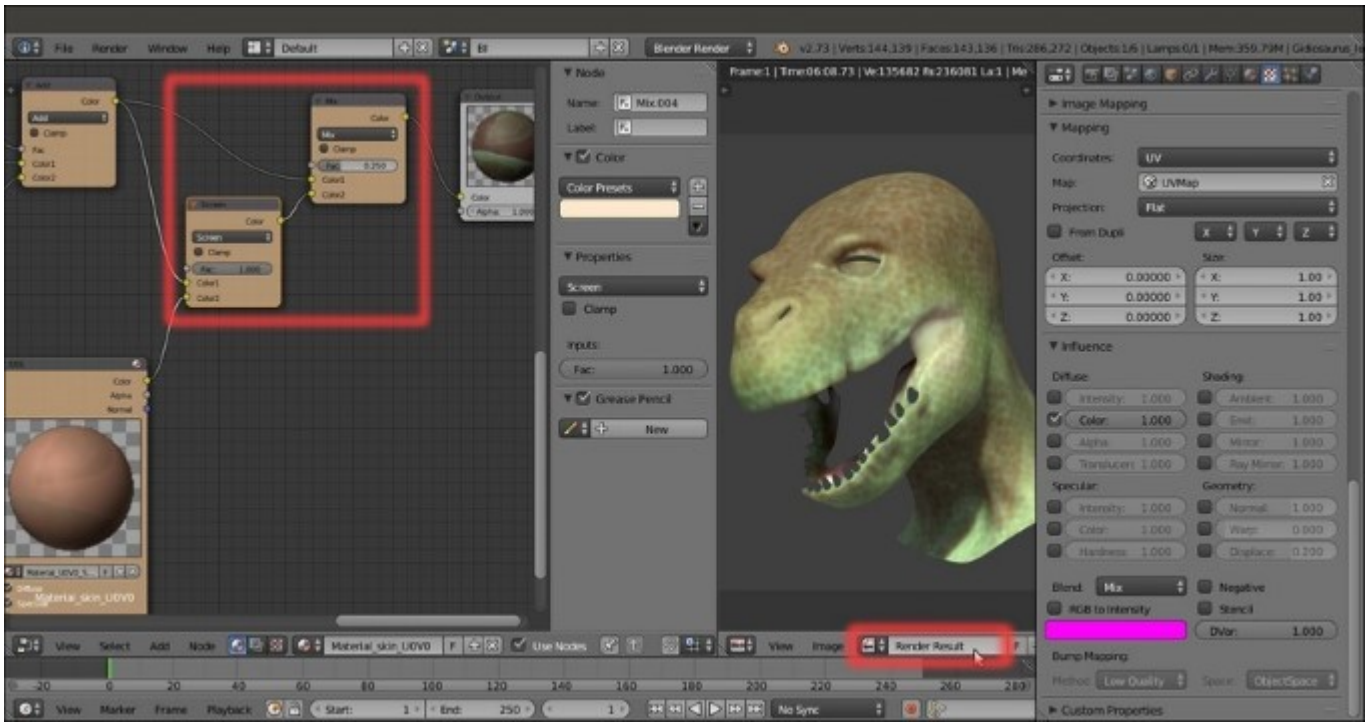
47. Now, select the **COL** material node and press *Shift* + *D* to duplicate it. Label the duplicated one as **SSS**, then through the **Node Editor** window, enable the **Specular** item. Click on the **2** icon button to the right side of the material name datablock to make it single user and rename the new copy of the material as `Material_U0V0_SSS`.
48. Go to the **Material** window and change the **Diffuse Shader Model** to **Minnaert** and the **Diffuse** color to **R 0.439**, **G 0.216**, **B 0.141**. Move down to the **Specular** subpanel and change the color to the same **R 0.439**, **G 0.216**, **B 0.141** brownish hue (copy and paste), then set the **Intensity** to **0.600** and the **Hardness** to **12**.
49. Go down to the **Subsurface Scattering** subpanel and enable it by checking the checkbox: set the **IOR** value to **3.840**, the **Scale** to **0.001**, copy and paste the brownish color also in the scattering color slot, set the **Color** slider to **0.000** and the **Texture** slider to **1.000**. Set the **RGB Radius**: **R 9.436**, **G 3.348**, **B 1.790**.
50. Go to the **Texture** window and set to **0.300** the **Color** channel sliders of the `U0V0`, `U0V0_scales_col_add2`, and `U0V0_scales_col` texture slots, then set to **0.117** the **Color** channel slider of the `U0V0_scales_col_add1` texture slot.
51. Select the last `vcol` texture slot and click on the **X** icon (*Unlink datablock*) button to clear it; click on the double little arrows to the left side of the **New** button and select the `vcol_light` item from the pop-up menu. Set the **Mapping** to **UVMap_norm**, and under the **Influence** subpanel, the **Color** of **Diffuse** to **0.267** and the **Blend Type** to **Screen**.



Testing the output of the SSS component material

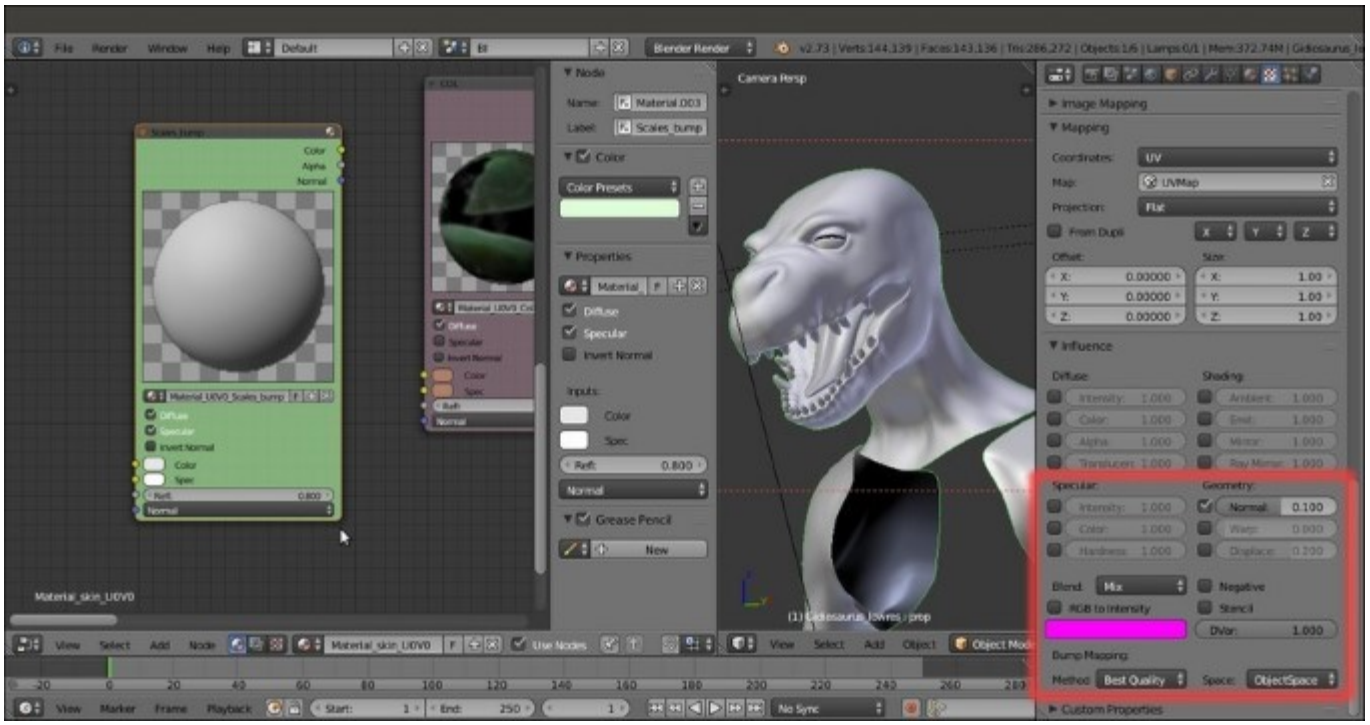
52. Add a new **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**), paste it between the **Add-MixRGB** and the **Output** node and set the **Fac** value to **0.250**.

53. Press *Shift* + *D* to duplicate this **Mix-MixRGB** node; change the **Blend Type** of the duplicated one to **Screen**, set the **Fac** value to **1.000** and connect the output of the **Add-MixRGB** also to the **Color1** input socket of the **Screen-MixRGB** node; connect the output of the **SSS** node to the **Color2** input socket of the **Screen-MixRGB** node.
54. Connect the output of the **Screen-MixRGB** node to the **Color2** input socket of the first **Mix-MixRGB** node.



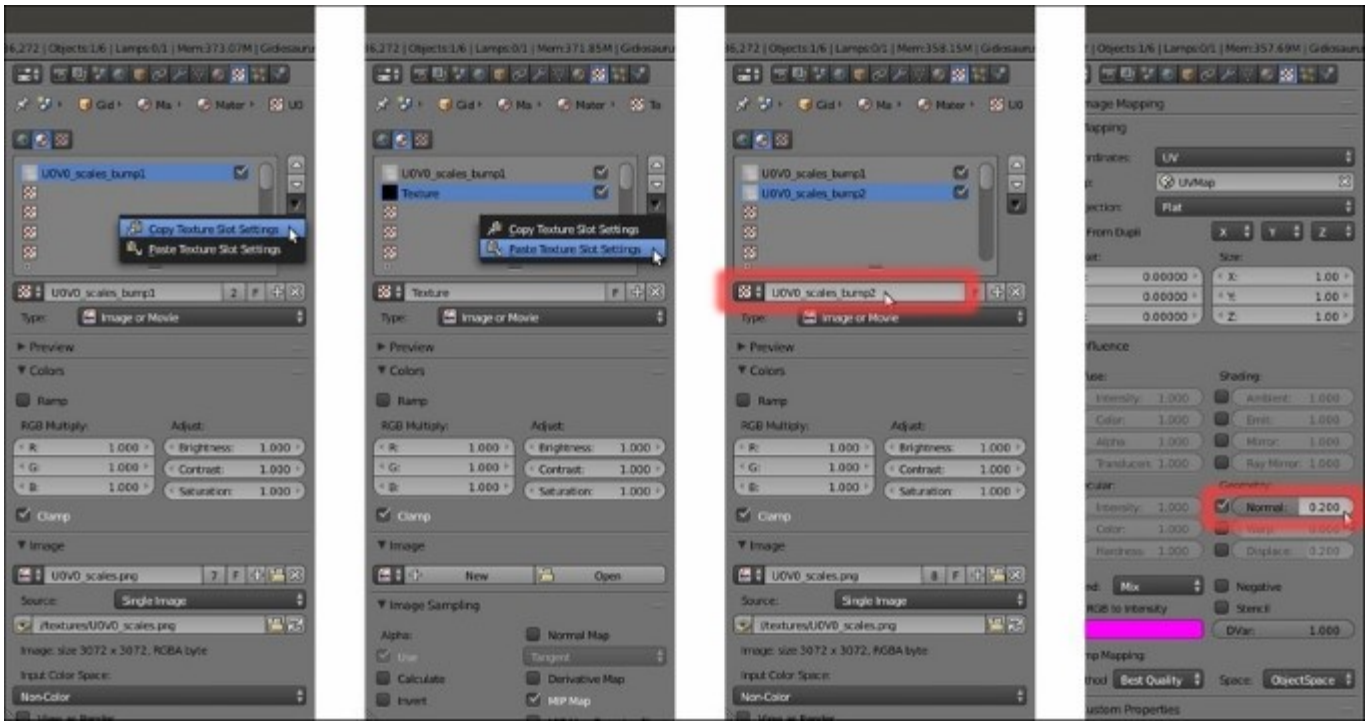
Adding the SSS component to the rest of the shader

55. Add a new **Material** node (*Shift* + *A* | **Input** | **Material**) and click on the **New** button to create a new material; label the node as **Scales_bump** and rename the material as `Material_U0V0_Scales_bump`.
56. In the **Material** window set the **Diffuse Shader Model** to **Oren-Nayar** and the **Specular Shader Model** to **Blinn**, **Intensity** = **0.100** and **Hardness** = 5. In the **Shading** subpanel enable the **Cubic Interpolation** item.
57. Go to the **Texture** window and in the **first** slot load the `U0V0_scales.png` image; rename the ID datablock as `U0V0_scales_bump1`. In the **Image Sampling** subpanel set the **Filter Size** to **5.00**, the **Mapping** to **UVMap** and in the **Influence** subpanel disable the **Color** channel and enable the **Normal** channel under **Geometry**: set the slider to **0.100** and go to the bottom to click on the **Bump Method** slot and select **Best Quality**:



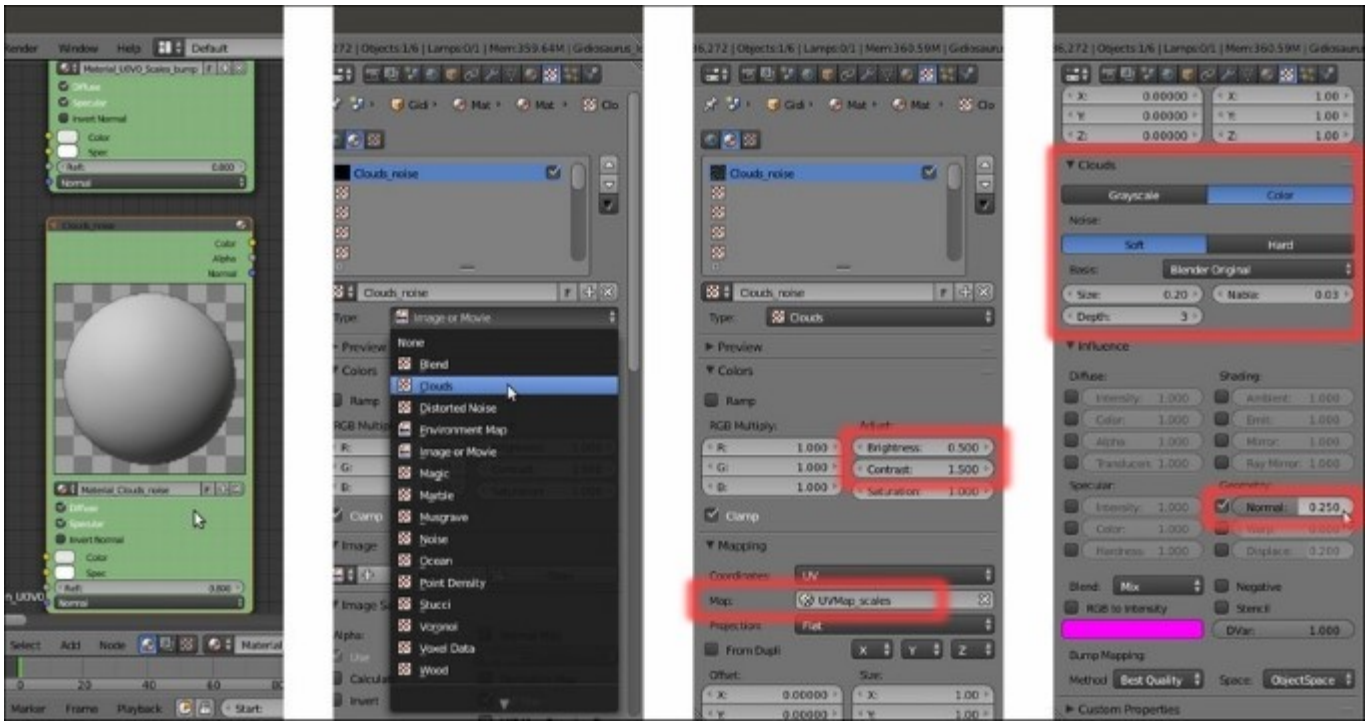
The bump material node

58. Go back to the top of the panel and click on the big black arrow to the right of the texture window; select the **Copy Texture Slot Settings** item.
59. Select the empty **second** texture slot and click on **New** to add a generic texture, then click again on the black arrow to select the **Paste Texture Slot Settings** item.
60. In the *ID datablock* slot, click on the **2** icon button to make it single user and rename it `U0V0_scales_bump2`.
61. Go down to the **Image Sampling** subpanel and set the **Filter Size** to **1.00**, then go down to the **Influence** subpanel and set the **Normal** slider to **0.200**.



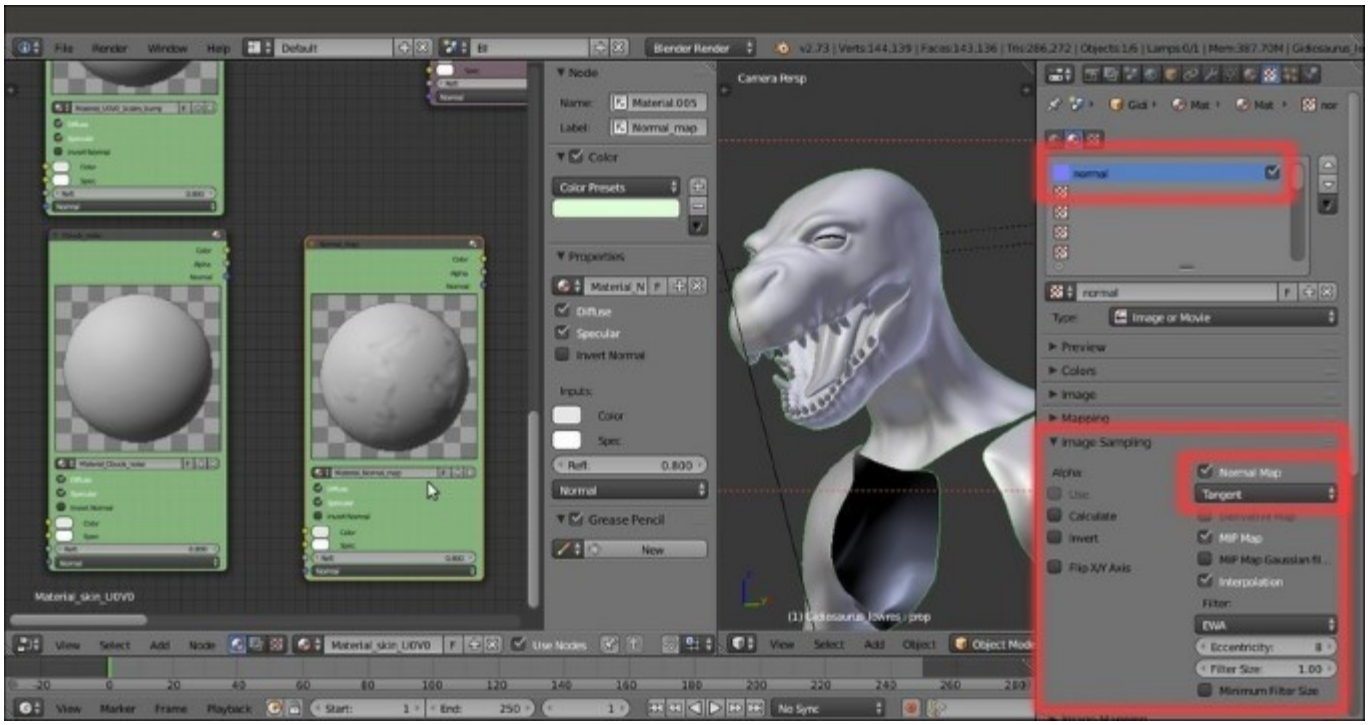
Copying and pasting a texture slot

62. Press *Shift* + *D* to duplicate the node; make the material of the duplicated one single user and rename it as `Material_Clouds_noise`, then label the node as **Clouds_noise**.
63. In the **Texture** window, delete (unlink) `U0V0_scales_bump1` and `U0V0_scales_bump2`, and then select the **first** slot and click on the **New** button: change the automatic `Texture.001` ID datablock name with `Clouds_noise`, click on the **Type** button and in the pop-up menu select the **Clouds** item.
64. In the **Colors** subpanel set the **Brightness** to **0.500** and the **Contrast** to **1.500**, in the **Mapping** subpanel set the **UVMap_scales** coordinates layer, in the **Clouds** subpanel switch from **Grayscale** to **Color**, set the **Size** to **0.20**, the **Depth** to **0.3** and the **Nabla** to **0.05**.
65. In the **Influence** subpanel disable the **Color** channel and enable the **Normal** channel under **Geometry**: set the slider to **0.250** and go to the bottom to click on the **Bump Method** slot and select **Best Quality**:



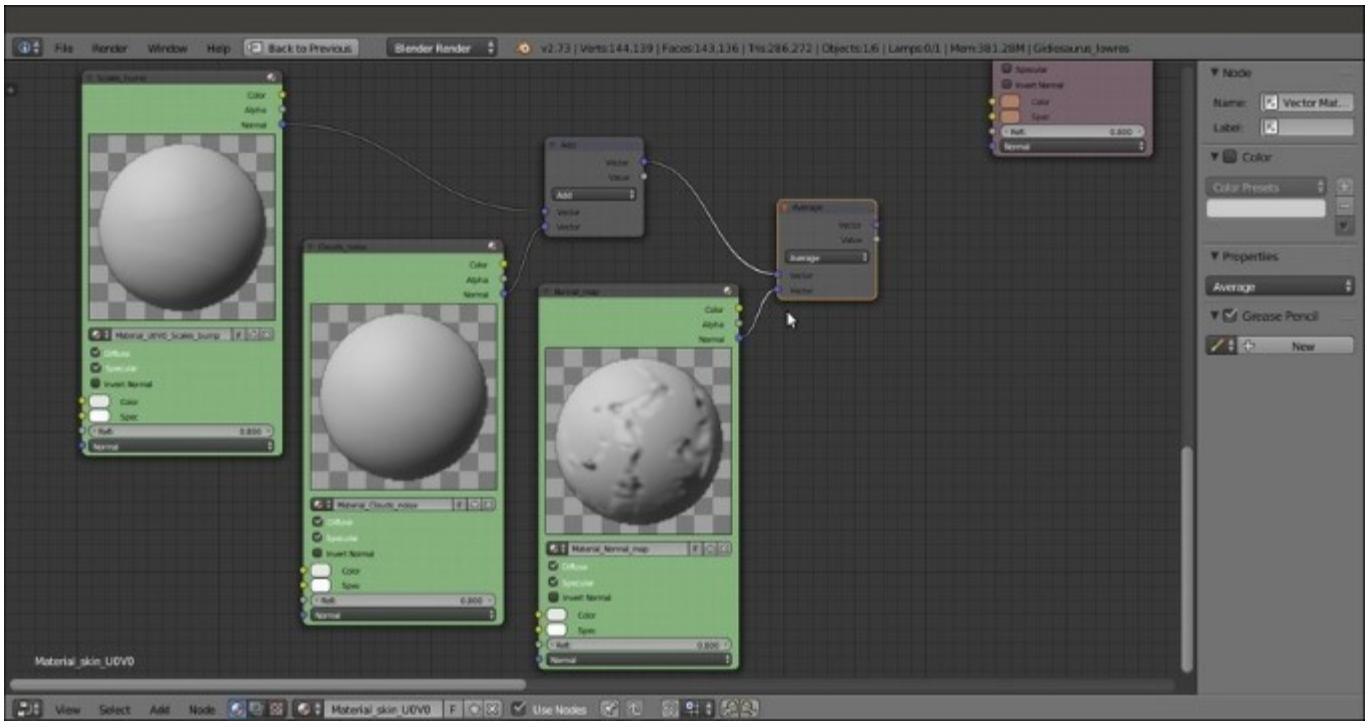
The second bump material node

66. Press *Shift* + *D* to duplicate the **Scales_bump** node; make the material of the duplicated one single user and rename it as `Material_Normal_map`, then label the node as **Normal_map** as well.
67. In the **Texture** window, delete (unlink) the `U0V0_scales_bump1` and `U0V0_scales_bump2`; select the **first** slot and click on the **New** button: change the automatic `Texture.001` ID datablock name to `normal` and then load the `norm.png` image.
68. In the **Mapping** subpanel set the **UVMap_norm** coordinates layer, then go to the **Image Sampling** subpanel and enable the **Normal Map** item; go to the **Influence** subpanel, disable the **Color** channel and enable the **Normal** channel: set the slider to **1.000** (higher values don't have an effect with normal maps in **Blender Internal**).



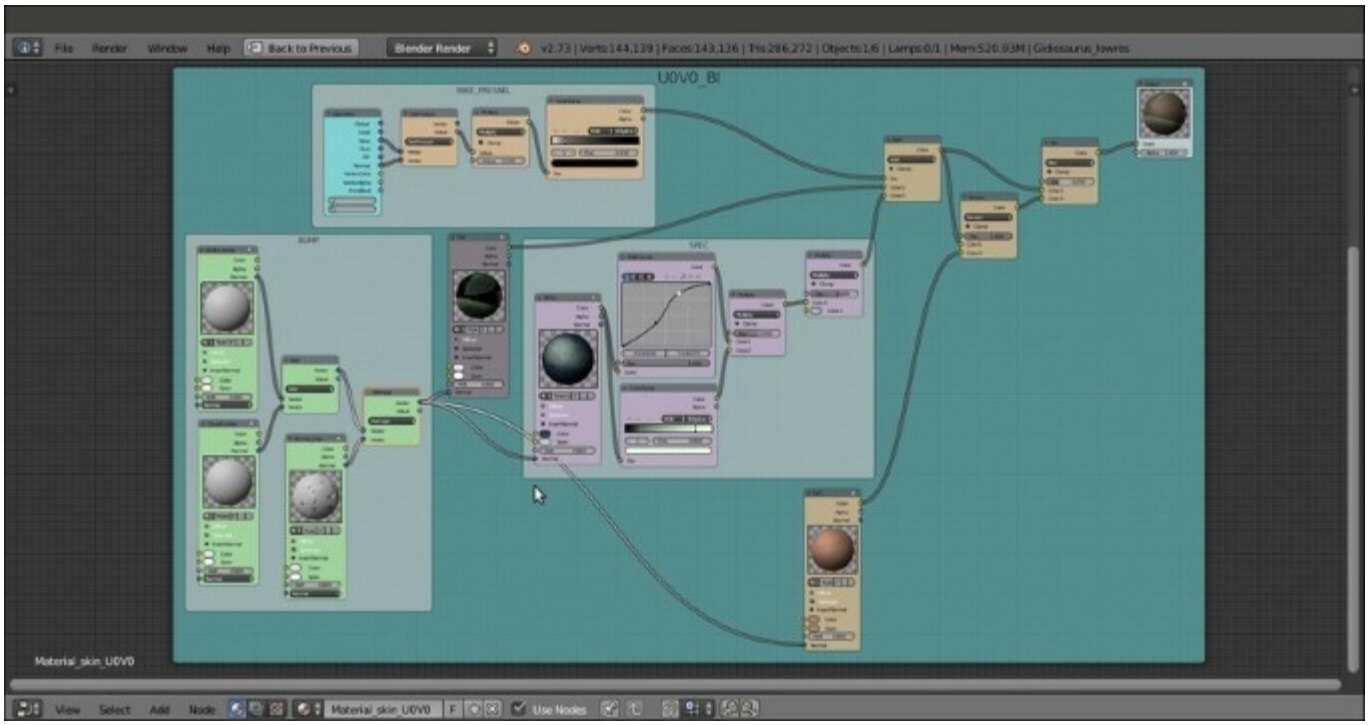
The normal map material node

69. Add a **Vector Math** node (*Shift* + *A* | **Converter** | **Vector Math**) and connect the **Normal** (blue) output of the **Scales_bump** node to the first **Vector** input socket of the **Vector Math** node, and the **Normal** output of the **Clouds_noise** node to the second **Vector** input socket.
70. Press *Shift* + *D* to duplicate the **Vector Math** node, set the **Operation** of the duplicate to **Average** and connect the **Vector** output of the **Add-Vector Math** node to the first **Vector** input socket of the **Average-Vector Math** node, and the **Normal** output of the **Normal_map** node to the second **Vector** input socket.



Connecting the outputs of the three bump nodes

71. Connect the **Vector** output of the **Average-Vector Math** node to the **Normal** input sockets of the **COL**, **SPEC**, and **SSS** material nodes.
72. Add frames everywhere to make things clear but especially to visually group and separate the nodes of the **Blender Internal** material from the **Cycles** ones.



The completed "U0V0_BI" node material

73. Save the file.

How it works…

You have probably noticed that a few of the nodes we can find in the **Cycles** material system are also available for the material nodes in **Blender Internal**; sadly, some are still missing (and probably forever will be), as, for example, a **Fresnel** node that, in fact, we had to approximate with a combination of other different nodes.

Anyway, although not all the same nodes are at our disposal, we had enough of them to try to obtain a result as close as possible as the result we obtained in the **Cycles** material (in the previous Chapter 12, *Creating the Materials in Cycles*).

Note

One thing you should absolutely keep in mind when loading the textures into the material nodes in **Blender Internal** is their order in the texture stack. This is important and must be taken into consideration according to the result we need, because a texture can totally overwrite the texture in the above slot (with the default **Mix** blend type) but can also be added, subtracted, multiplied, divided, and so on; the textures stack works the same as the layer stack system of a 2D graphic editor (**Gimp**, for instance), with the order from the top to the bottom and the different blending options (the **Blend Type** items).

Having said that, let's see the steps:

- From step 1 to step 9 we created a basic **Blender Internal** material node by using both the **Node Editor** and the **Material** window.
- From step 10 to step 23 we assigned the proper textures to the basic material node that becomes, in this case, the `Material_U0V0_Col`, the basic **diffuse color** component of the shader.

  These steps have been described in the most detailed way possible because they are the same steps for all the textures added to the materials; of course, the values and the settings can be different, but basically:
    - We add a texture (image or procedural)
    - We set a mapping orientation
    - We set the influence value on the selected channel (also more than one at a time)
    - Because the same texture can be used (with different settings) more than once, we always rename the *Unique datablock ID* name to make them easily recognizable in the pop-up menu list.
- The **Filter Size** value in the **Image Sampling** subpanel is really useful for blurring an image texture: a value of **1.00** is the default sharpness, while a higher value makes the image more and more blurred.
- From step 24 to step 30 we created the **glossy**/**specular** `Material_U0V0_Spec` node.
- From step 31 to step 34 we added the textures to the `Material_U0V0_Spec` material. This material should represent the **glossy**/**specular**/**mirror** component of the shader, that is probably the most important thing for obtaining a correct visual result; in **Cycles** the **Glossy BSDF** shader node provides the result perfectly, while in **Blender Internal**, we have two options: one, by enabling the (slow and imperfect) internal ray-tracing **Mirror** item, or by faking it. We faked it by setting an image (the same **hdr** we'll use in the next chapter in the **World** both for **Cycles** and **BI**) on the **Reflection** channel of the shader, hence giving the impression of an environment (slightly) mirrored by the character's skin.
- At step 35 we added together the outputs of the **COL** and of the **SPEC** nodes.
- From step 36 to step 40 we tweaked the output of the **SPEC** nodes to obtain a more realistic output/distribution of the glossiness on the mesh's surface, trying to mimic, as much as possible, the glossy output of the **Cycles** shader version.
- From step 41 to step 46 we built a fake **Fresnel** to work as a factor for the blending of the **glossy** and the **diffuse** components; this works by calculating the dot product of the vectors of the point of view and the mesh's normals. Be aware that it isn't actually working as the real **Fresnel** node that you find in **Cycles**, and that it has several limitations. By varying the second **Value** of the **Math** node and/or the black color stop position of the **ColorRamp** node, we can obtain several nice effects in some way visually similar to the real output. If you are wondering why we don't use the output of a **BI** material with a **Fresnel** diffuse shader model, sadly it doesn't seem to work correctly (actually, it doesn't seem to work at all).
- From step 47 to step 51 we built the **SSS** material node by duplicating the **COL** node, making a new copy of the material and renaming it as `Material_U0V0_SSS`, then enabling **Subsurface Scattering** and modifying the influence values of the textures; the values of the subsurface scattering (**IOR**, **Scale**, **RGB Radius**), instead, were borrowed from the **Cycles** version of the shader.
- From step 52 to step 54 we added the output of the **SSS** node to the rest of the shader by using a **Blend Type** set to **Screen** plus a **Mix** one set to a low **Fac** value; basically, the exact copy of what we did in **Cycles**.

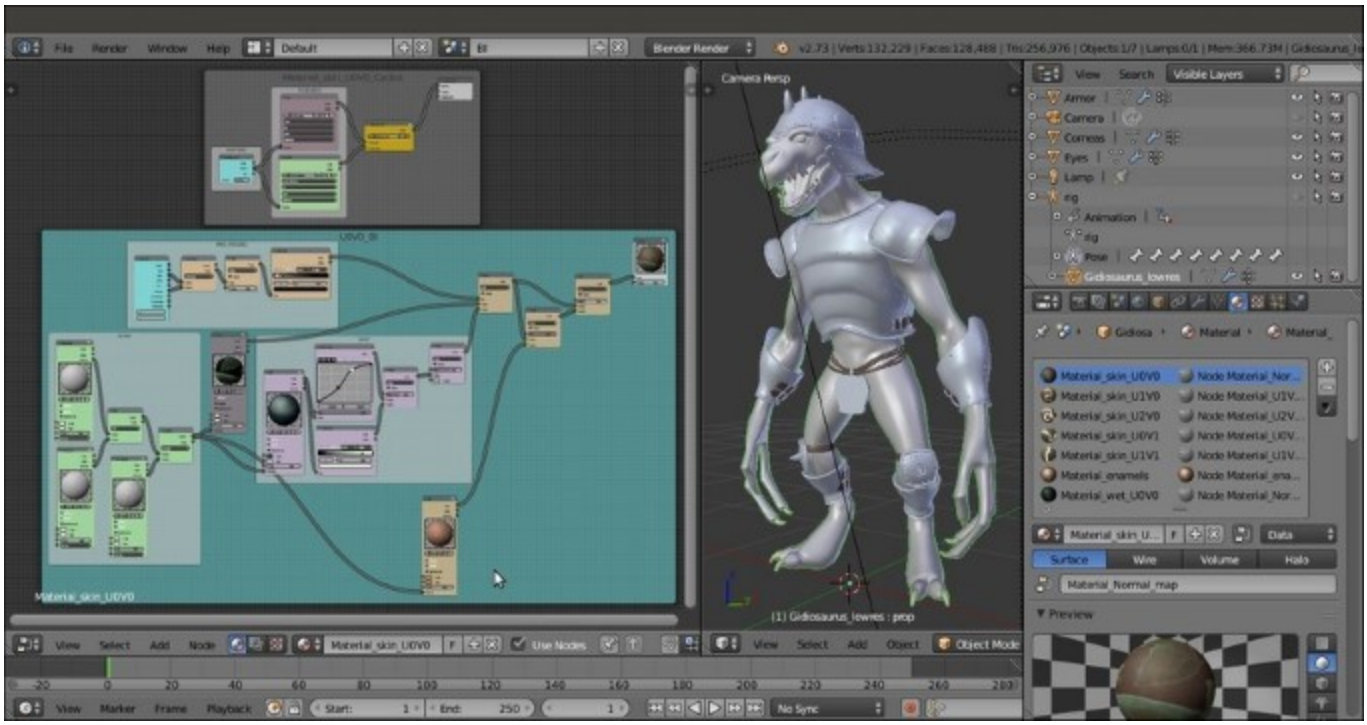
- From step 55 to step 71 we created the **bump pattern**, divided into **three** different material nodes to give us more flexibility in adding and averaging them together in a way that is as similar as possible to **Cycles**:

The F12 rendered final result in Blender Internal

There's more…

The other missing shaders for the **Gidiosaurus** skin are solved in exactly the same way we used in the previous chapter for the other **Cycles** skin shaders: by selecting the entire **BI** frame with all the parented nodes and pressing *Ctrl* + *C* to copy them, then selecting the different material slots and pressing *Ctrl* + *V* to paste everything; in **Blender Internal**, we have to make the single materials inside the nodes single user one by one and then substitute the textures according to the **UDIM** tile the material corresponds to (`U1V0_col.png`, `U1V0_scales.png`, `U2V0_col.png`, and so on).

So, in the end, each material will have *two sets of nodes*, one for the **Cycles** shader and one for the **Blender Internal** shader, and each one works under the respective render engine; this will be useful, as we'll see in the next chapter, for the rendering stage.



Two different sets of nodes for the same material

See also

- http://www.blender.org/manual/render/blender_render/materials/index.html
- http://www.blender.org/manual/render/blender_render/textures/index.html

Building the eyes' shaders in Blender Internal

We'll now see how to make the shaders for the **Gidiosaurus's eyes**; they are composed of two objects, the **Corneas** and the **Eyes** objects, so let's start with the first one.

Getting ready

Enable the **6th** and the **12th** scene layers and select the **Corneas** object; in the **Outliner**, disable the **Eyes** object's visibility in the viewport to hide it, put the mouse pointer inside the **Camera** view, zoom to one of the **eyeballs** and then start the **Rendered** preview.

How to do it…

Let's start to create the `Corneas` material:

1. Put the mouse pointer in the **Node Editor** window and add: a **Material** node (*Shift* + *A* | **Input** | **Material**), a **Geometry** node (*Shift* + *A* | **Input** | **Geometry**), a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**) and an **Output** node (*Shift* + *A* | **Output** | **Output**).
2. Connect the **Color** output of the **Material** node to the **Color** input socket of the **Output** node, then click on the **New** button on the **Material** node to create a new material and rename it `Cornea_bump`.
3. In the **Material** window, expand the **Render Pipeline Options** subpanel and enable the **Transparency** item; in the **Diffuse** subpanel set the shader model to **Oren-Nayar** and the color to a bright orange = **R 0.930**, **G 0.386**, **B 0.082**. In the **Specular** subpanel set the shader model to **WardIso** and the **Slope** to **0.070**. In the **Shading** subpanel enable the **Cubic Interpolation** item.
4. Go down to expand the **Transparency** subpanel; set the **Fresnel** value to **1.380** and the **Blend** to **1.700**.
5. Go further down to enable the **Mirror** item in the subpanel with the same name, set the **Reflectivity** to **0.200**, the **Fresnel** to **1.380** and the **Blend** to **1.500**.
6. Go to the **Texture** window and in the **first** slot load the image `eyeball_col.jpg`, rename the ID datablock as `eyeball_col` and set the **UVMap.001** as the coordinates layer; in the **Influence** subpanel set the diffuse **Color** channel to **0.200** and the specular **Color** channel to **0.200** as well, set the **Blend Type** to **Color**.
7. In the **second** texture slot load the image `eyeball_bump.jpg`, rename the ID datablock as `Eyeball_bump`, set **UVMap.001** as the coordinates layer and disable the **Color** channel to enable the **Normal** one at **0.007**; set the **Bump Method** to **Best Quality**:

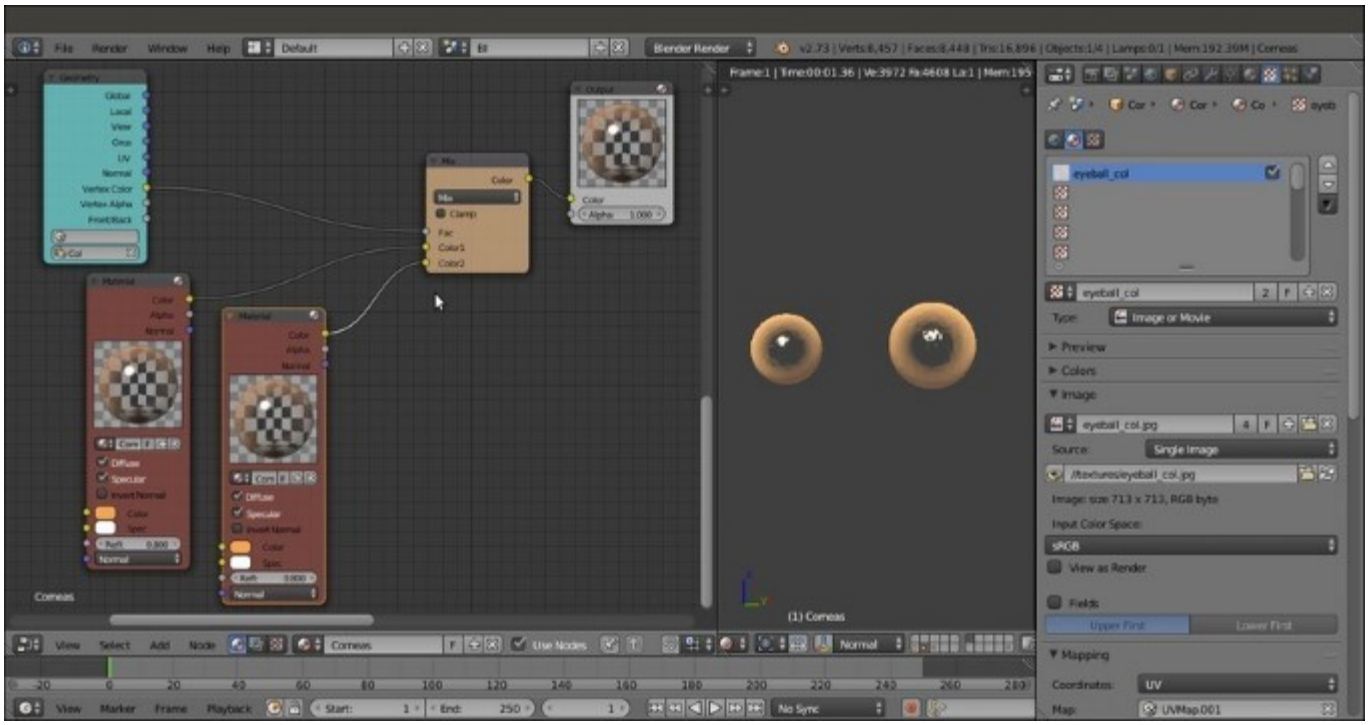
The "Corneas" material nodes

8. Paste the **MixRGB** node between the **Material** and the **Output** nodes; then, connect the **Vertex Color** output of the **Geometry** node to the **Fac** input socket of the **MixRGB** node. Click on the last empty field at the bottom of the **Geometry** node to select the **Col** item from the pop-up list (it's the name of the **Vertex Color** layer that has been created, and mentioned, at the beginning of the *Building the eyes' shaders in Cycles* recipe in Chapter 12, *Creating the Materials in Cycles*).

The gray color of the Color2 socket of the MixRGB node showing in the rendered preview at the location established by the Vertex Color layer output used as factor

In the preceding screenshot, the effect of the **Vertex Color** layer is visible: the two gray dots on the eyeballs are actually the crystalline lens areas filled, only at the moment, with the gray color of the empty **Color2** input socket of the **MixRGB** node.

9. Press *Shift* + *D* to duplicate the **Material** node, click on the **2** icon button to the right side of the name datablock of the duplicated node to make the material single user, rename the new material, simply, `Cornea` and, in the **Texture** window, select the **second** slot texture, `eyeball_bump`, to click on the **X** icon button and delete it.
10. Connect the **Color** output of the second **Material** node to the **Color2** input socket of the **MixRGB** node.



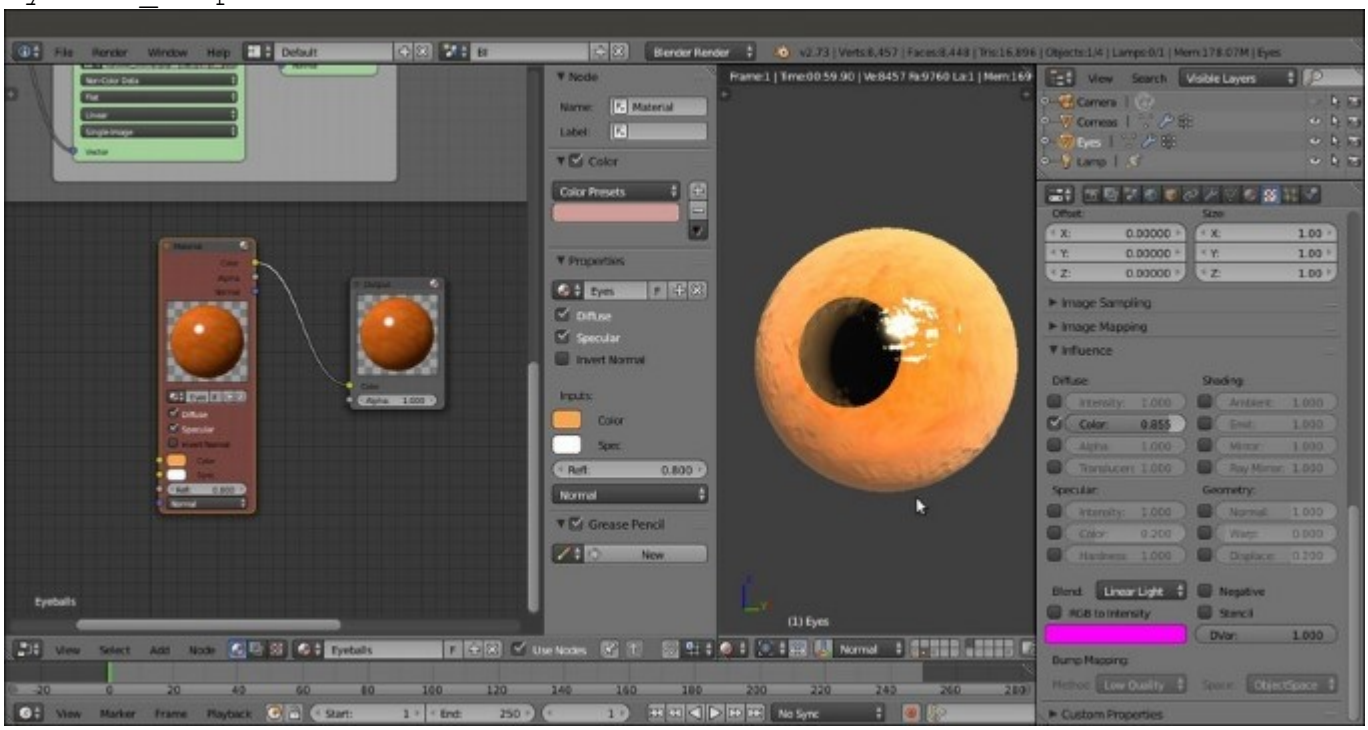
The completed "Corneas" material

Now, go to the **Outliner** and enable the **Eyes** object visibility in the viewport to show it:

11. With the **Corneas** object still selected, put the mouse pointer on the first **Material** node in the **Node Editor** window and press *Ctrl* + *C* to copy it.
12. Select the **Eyes** object and, in the **Material** window, select the `Eyeballs` material slot; put the mouse pointer in the **Node Editor** window and press *Ctrl* + *V* to paste the material node we copied before.
13. Click on the **2** icon button to make the material single user and rename it `Eyes`. Add an **Output** node (*Shift* + *A* | **Output** | **Output**) and connect the **Color** output of the **Eyes** material node to the **Color** input socket of the **Output** node.
14. Go to the **Material** window and disable the **Transparency** item in the **Render Pipeline Options** subpanel; go to the **Mirror** subpanel and disable it.
15. Enable the **Subsurface Scattering** subpanel: set the **IOR** to **1.340**, the **Scale** to **0.001**, the scattering color to the orange **R 0.930**, **G 0.386**, **B 0.082** and the **RGB Radius** to the **R 9.436**, **G 3.348**, **B 1.790** values.

The "Eyeballs" material SSS settings

16. Go to the **Texture** window; select the `eyeball_col` texture and set the diffuse **Color** to **0.855**, disable the specular **Color** channel and set the **Blend Type** to **Linear Light**; select the `eyeball_bump` texture and set the **Normal** channel slider to **0.005**.



The "Eyeballs" material texture settings

Now let's see the **iris**:

17. Box-select both the `Eyes` material node and the connected **Output** node and press *Ctrl* + *C* to copy them; go to the **Material** window and select the `Irises` material slot, then put the mouse pointer in the **Node Editor** window and press *Ctrl* + *V* to paste them.
18. Make the duplicated node's material single user and rename it `Iris`; change the **Diffuse** subpanel color to **R 0.429**, **G 0.153**, **B 0.000**, then go to the **Shading** subpanel and set the **Emit** value to **0.07**. Go to the **Subsurface Scattering** subpanel and change the scattering color to **R 0.220**, **G 0.033**, **B 0.032**.
19. Go to the **Texture** window and delete (unlink) the two texture slots. In the **first** slot, load the image `iris_col.jpg`, rename the ID datablock `iris_col`, and set **UVMap.001** as the UV coordinates layer. In the **second** slot, load the image `iris_bump.jpg`, rename as `iris_bump`, set **UVMap.001** as the UV coordinates layer and in the **Influence** subpanel disable the **Color** channel and enable the specular **Intensity** and **Hardness** channels with value **1.000**, then enable also the **Normal** channel with value **1.000**. Set the **Bump Method** to **Best Quality**.
20. In the **third** texture slot, load again the `iris_bump.jpg` image; rename the ID datablock as `iris_ST`. In the **Image Sampling** subpanel, under **Alpha**, disable the **Use** item and enable the **Calculate** item. Set the **UVMap.001** coordinates layer and in the **Influence** subpanel disable the **Color** channel and enable only the **Stencil** item at the bottom:

The "Irises" material and the Stencil item

21. In the **fourth** texture slot, load the `iris_col.jpg` image, rename the ID datablock as `iris_emit`. Set the **UVMap.001** coordinates layer and in the **Influence** subpanel disable the **Color** channel and enable the **Emit** channel at value **1.000**.

The "Irises" material emitting (fake) light

Regarding the `Pupils` material, it's a simple basic material with pure black as **Diffuse** color and the **Intensity** slider under the **Specular** subpanel set to **0.000** to be totally matte.

22. Save the file.

How it works…

For the **Corneas** object: we created two copies of the same transparent material, but then we removed the bump from one of them, because usually a **cornea** has bumps due to the veins on the **eyeball** but not on the **crystalline lens**, that is smooth; the two materials are mixed, exactly as in the **Cycles** version, through the output of the **Col Vertex Color** layer.

Regarding the `Irises` material: the `iris_ST` texture, set as a **stencil map**, works as a mask for the following texture to appear through its black areas.

Although it could have been solved by simply leaving a blank material slot, I assigned a black matte material to the **pupils**; I preferred to assign a material anyway, to avoid possible issues in the following stages such as, for example, in the rendering of the character against an alpha backdrop, of the separated passes and in the compositing.

Note that the `Corneas` is the only material where I enabled the ray-tracing mirror, which in the character's **skin** and in the **armor** are instead faked, to obtain faster rendering times (the **eyes** are really a small surface to be rendered).

Building the armor shaders in Blender Internal

We arrive finally at making the **Armor** shaders under the **Blender Internal** engine; we have **four** materials, here: the two **UDIM** plate shaders, the rivets shaders and the leather material for the tiers.

Getting ready

Enable the **6th** and the **13th** scene layers and select the **Armor** object; if your computer is powerful enough, use the **Rendered** preview while you are working.

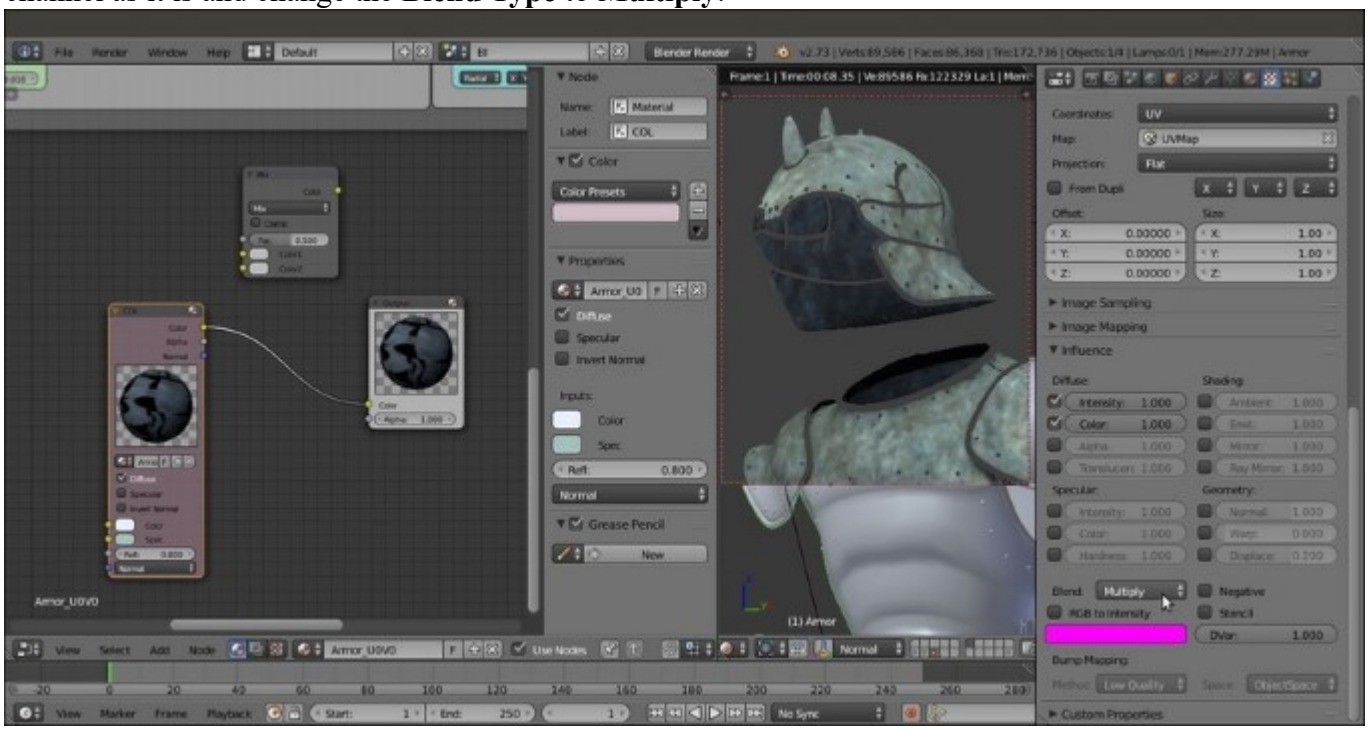
How to do it…

Let's start with the first **UDIM** tile material creation, the main **armor** plates:

1. Put the mouse pointer in the **Node Editor** window and add a **Material** node (*Shift* + *A* | **Input** | **Material**), a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**) and an **Output** node (*Shift* + *A* | **Output** | **Output**). In the *N* **Properties** sidepanel, label the **Material** node as **COL**.
2. Connect the **Color** output of the **COL** node to the **Color** input socket of the **Output** node, then click on the **New** button on the **Material** node to create a new material and rename it `Armor_U0V0_col`; in the **Node Editor** window, disable the **Specular** item.
3. Go to the **Material** window and find the **Diffuse** subpanel; change the shader model to **Oren-Nayar**, set the color to **R 0.817**, **G 0.879**, **B 1.000** and the **Roughness** value to **0.313**.
4. Go down to the **Specular** subpanel: set the shader model to **WardIso**, the **Intensity** to **1.000**, the **Slope** to **0.270**, and the color to **R 0.381**, **G 0.527**, **B 0.497**. In the **Shading** subpanel enable the **Cubic Interpolation** item.

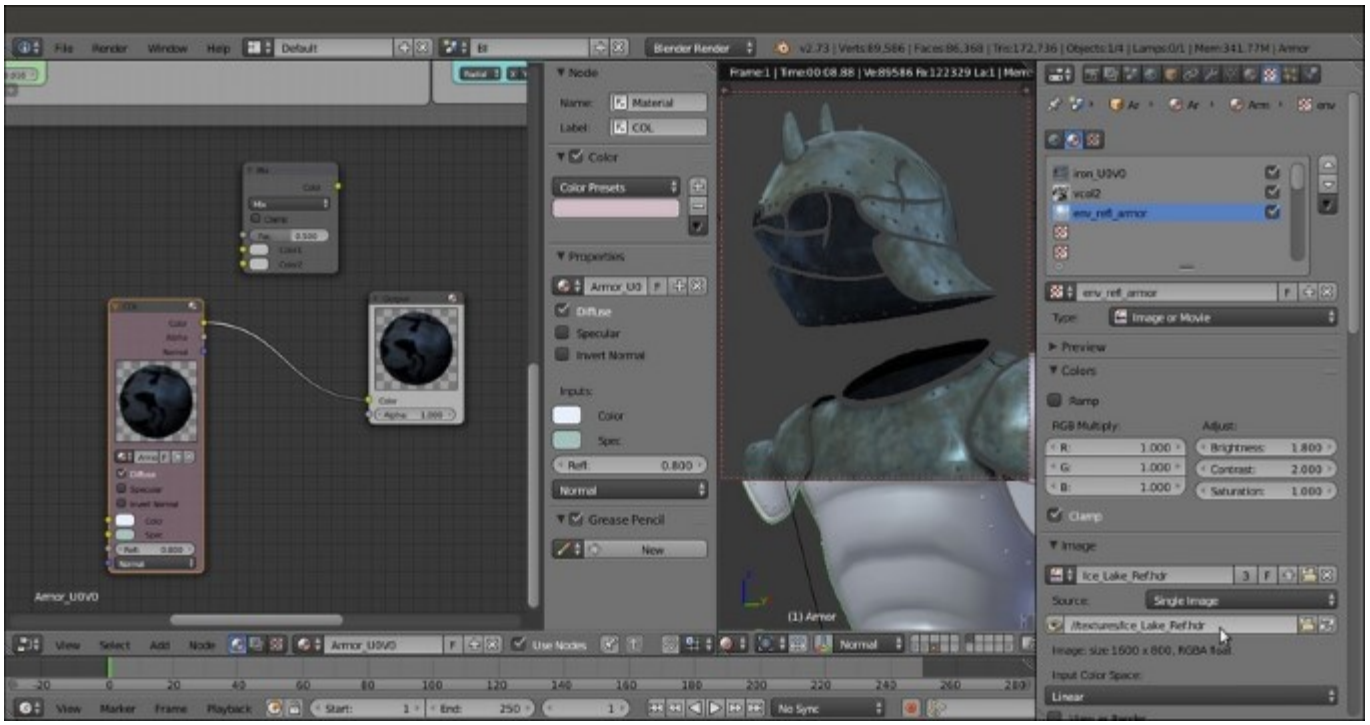
Starting the "Armor_U0V0" material in Blender Internal

5. Go to the **Texture** window and in the **first** texture slot, load the image `iron_U0V0.png`; rename the ID datablock as `iron_U0V0` and set the **UVMap** coordinates layer, then go to the **Influence** subpanel and enable the diffuse **Intensity** channel at value **1.000**, leave the **Color** channel as it is and change the **Blend Type** to **Multiply**:



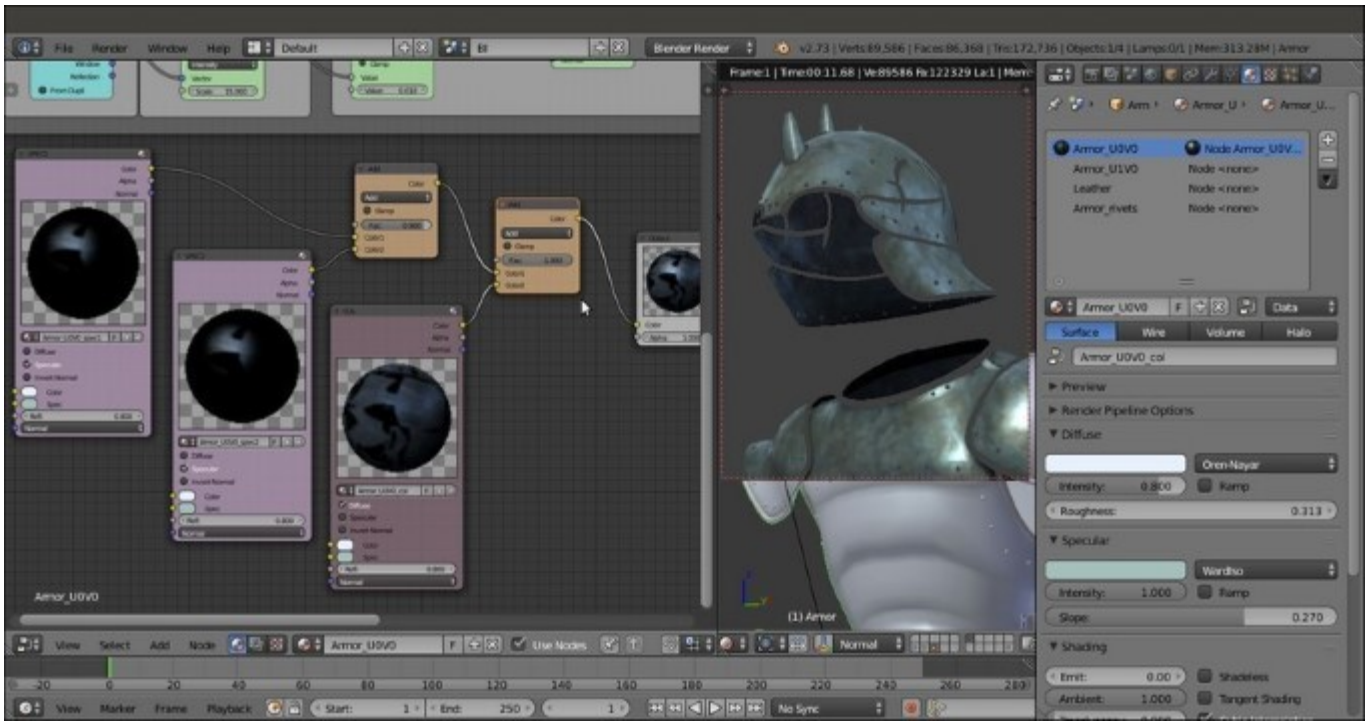
Adding the first texture image

6. In the **second** texture slot, load the image `vcol2.png`, rename the ID datablock as `vcol2` and set the **UVMap_norm** UV coordinates layer; go to the **Colors** subpanel, enable the **Ramp** item and set the **Interpolation** to **B-Spline**, then move the black color stop to position **0.245** and the white color stop to position **0.755**. In the **Image Sampling** subpanel set the **Filter Size** to **1.10** and in the **Influence** subpanel enable the diffuse **Intensity** channel at value **0.500**, the diffuse **Color** channel at **0.300**, set the **Blend Type** to **Difference** and enable the **Negative** item.
7. In the **third** texture slot, load the image `Ice_Lake_Ref.hdr` and rename the ID datablock as `env_refl_armor`. Set the **Mapping** coordinates to **Reflection** and, in the **Image Sampling** subpanel, the **Filter Size** to **6.00**; in the **Colors** subpanel set the **Brightness** to **1.800** and the **Contrast** to **2.000**, then move to the **Influence** subpanel and set both the diffuse **Intensity** and **Color** channels to **0.600** and the **Blend Type** to **Multiply**:



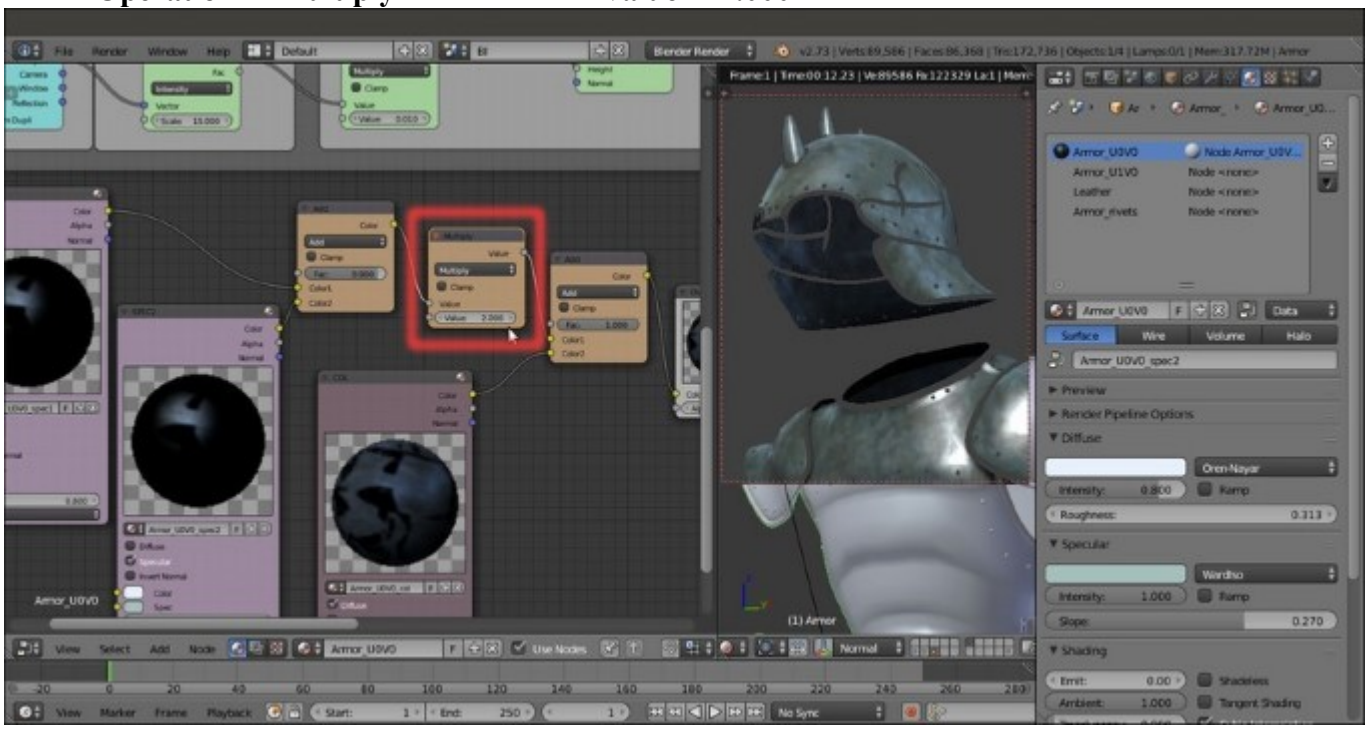
Adding the hdr image as reflection map

8. Go to the **Node Editor** window and press *Shift* + *D* to duplicate the **COL** node; label the duplicate as **SPEC1**, then make the material single user and rename it `Armor_U0V0_spec1`; disable the **Diffuse** item on the node interface and enable back, the **Specular** one.
9. Go to the **Texture** window; select the first `iron_U0V0` texture slot and go straight to the **Influence** subpanel: disable the diffuse **Intensity** and **Color** channels and enable the specular **Intensity**, **Color** and **Hardness** channels at **1.000**. Set the **Blend Type** to **Mix**.
10. Select the **second** `vcol2` texture slot, disable the diffuse **Intensity** and **Color** channels and enable the specular **Intensity** channel at **0.300**.
11. Select the **third** `env_refl_armor` texture slot, disable the diffuse **Intensity** and **Color** channels and enable the specular **Intensity** and **Color** channels at **0.500**.
12. Press *Shift* + *D* to duplicate the **SPEC1** node and label the duplicated one as **SPEC2**: make the material single user and rename it as `Armor_U0V0_spec2`. In the **Material** window go to the **Specular** subpanel and set the **Slope** to the maximum = **0.400**.
13. Now, connect the **Color** output of the **SPEC1** material node to the **Color1** input socket of the **MixRGB** node and the **Color** output of the **SPEC2** node to the **Color2** input socket; set the **Blend Type** of the **MixRGB** node to **Add** and the **Fac** value to **0.900**.
14. Press *Shift* + *D* to duplicate the **MixRGB** node and connect the output of the first **MixRGB** node to the **Color1** input socket of the duplicated **MixRGB** node, and the **Color** output of the **COL** node to the **Color2** input socket; set the **Fac** value of the second **MixRGB** node to **1.000** and connect its output to the **Color** input socket of the **Output** node.



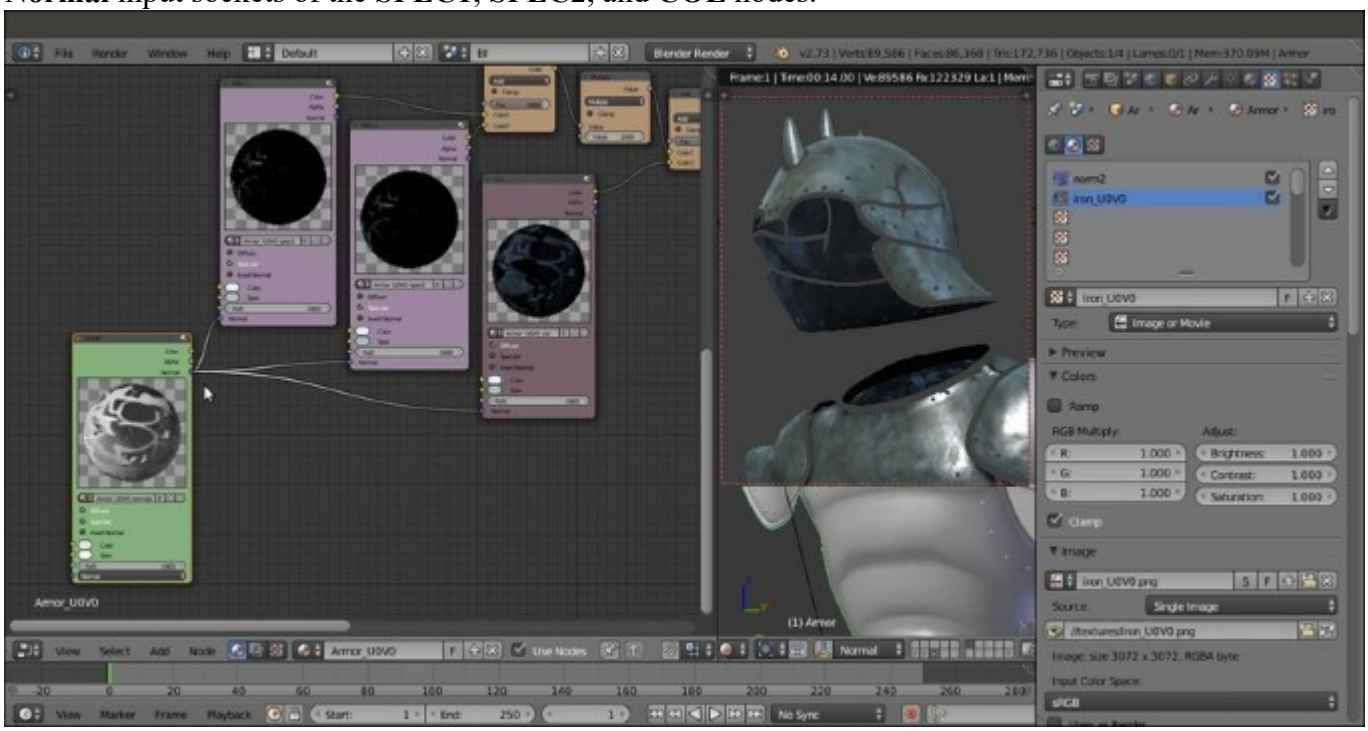
Adding the specular component

15. Add a **Math** node (*Shift + A* | **Converter** | **Math**) and paste it between the two **MixRGB** nodes; set the **Operation** to **Multiply** and the second **Value** to **2.000**.



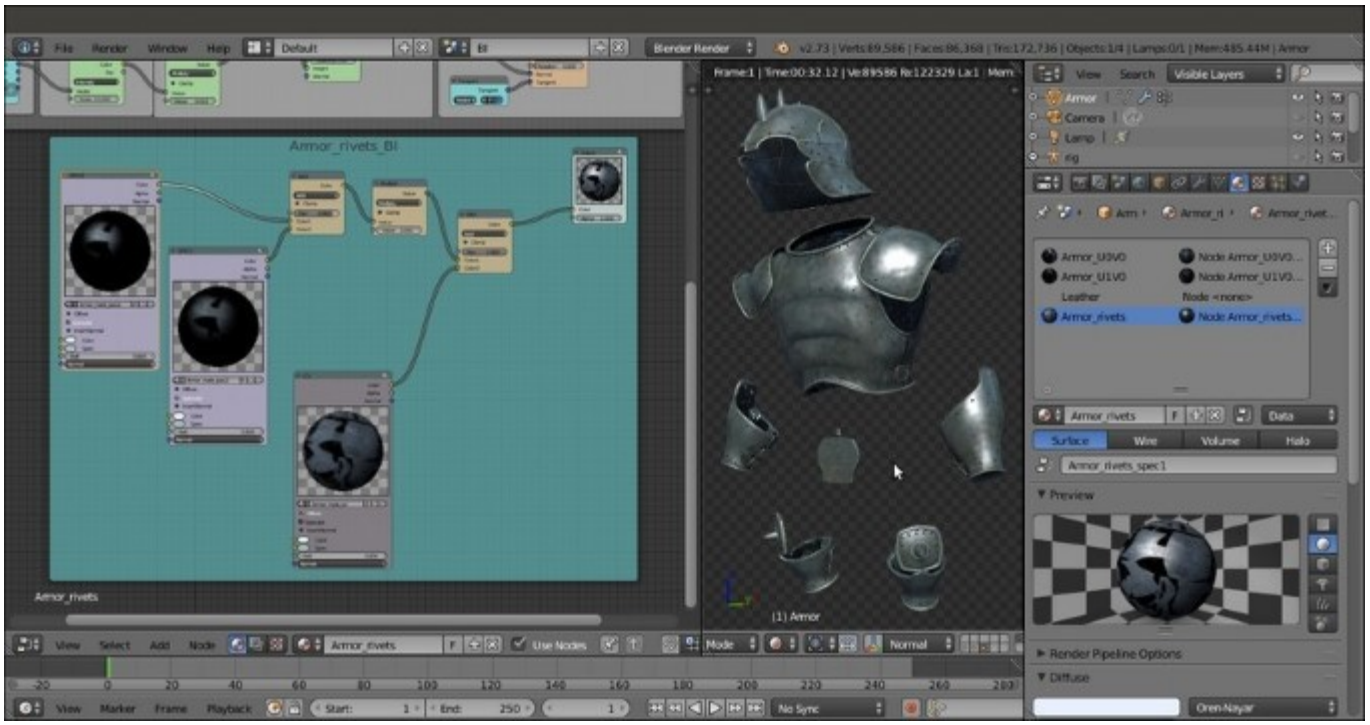
Enhancing the specularity

16. Add a **Material** node (*Shift* + *A* | **Input** | **Material**) and label it as **BUMP**; create a new material and rename it as `Armor_U0V0_normals`; in the **Shading** subpanel enable the **Cubic Interpolation** item.
17. Go to the **Texture** window and in the **first** texture slot, load the image `norm2.png`, rename the ID datablock as `norm2` and in the **Image Sampling** subpanel enable the **Normal Map** item; in the **Mapping** subpanel set the **UVMap_norm** coordinates layer and in the **Influence** subpanel disable the **Color** channel and enable the **Normal** one at **0.500**.
18. In the **second** texture slot, load the image `iron_U0V0.png`, mapping to **UVMap** layer and **Influence** to **Normal** at **0.010**; set the **Bump Method** to **Best Quality**.
19. Go to the **Node Editor** window and connect the **Normal** output of the **BUMP** node to the **Normal** input sockets of the **SPEC1**, **SPEC2**, and **COL** nodes.



Adding the bump pattern

20. Box-select and press *Ctrl* + *C* to copy all these nodes, go to the **Material** window to select the `Armor_U1V0` material slot and, back in the **Node Editor** window, paste the copied nodes: then make the materials inside the nodes as single users, rename them accordingly and go to the **Texture** window to substitute the `iron_U0V0.png` image with the `iron_U1V0.png` image.
21. Copy and paste again, the nodes for the `Armor_rivets` material slot, but don't substitute the texture image: instead, simply delete the **BUMP** node, which wouldn't be of any use in such small parts.



The completed armor shaders in Blender Internal

22. Save the file.

How it works…

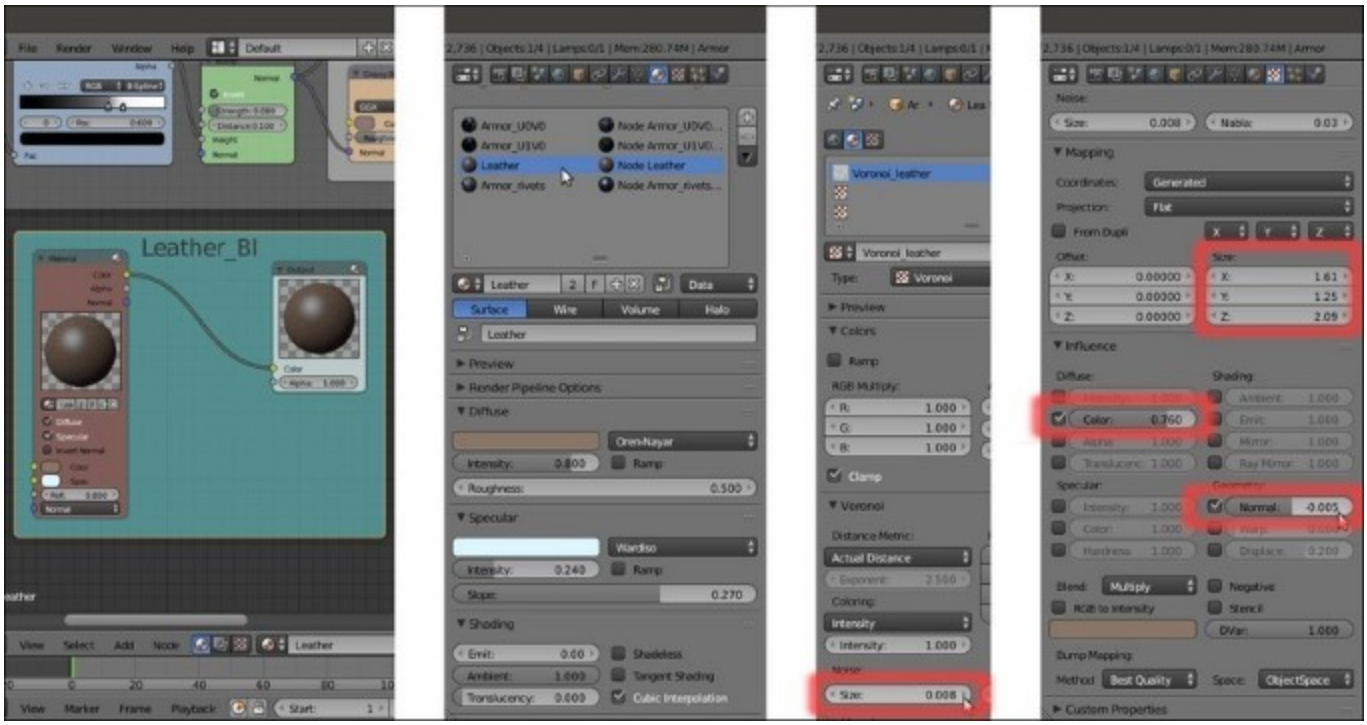
These shaders work, and have been built, exactly the same way as for the **Gidiosaurus' skin** and **eyes**; the only thing worth noting here is the order of the normal map and of the texture used for the bump pattern: in fact, to work together, in **Blender Internal**, the normal map must be placed higher in the texture stack, otherwise it will overwrite the effect of the bump map.

There's more…

The `Leather` material is a simple basic **Oren-Nayar** diffuse shader with the usual **WardIso** specular shader model, provided with a bump effect obtained through a **Voronoi** procedural texture with default values, except for the **Size**.

Note that the **Voronoi** texture influences the **Color** channel with the **Multiply** blend type and the **Normal** channel with a **negative** low value to obtain an actual bulging out pattern, instead of a concave one; negative values, in fact, reverse the *direction* of the bump.

The size of the procedural texture along the three axes is also further tweaked in the **Mapping** subpanel, scaling the three axes differently to resemble the dimensions of the **Texture Space** (basically the mesh bounding box, than can be made visible through the subpanel of the same name in the **Object Data** window), in order to avoid stretching along the mesh.



The "Leather" material in Blender Internal

Note also that in all these **Blender Internal** materials, simple mono materials (as for example the `Leather BI` material shown here earlier) are loaded inside a **Material** node and then connected to an **Output** node in the **Node Editor** window even if this wouldn't be necessary for the material itself to work: but, to let the **Blender Internal** and the **Cycles** render engines work together (through the compositor, as we'll see in the next chapter), it is mandatory to have all the shaders as nodes.

Chapter 14. Lighting, Rendering, and a Little Bit of Compositing

In this chapter, we will cover the following recipes:

- Setting the library and the 3D scene layout
- Setting image based lighting (IBL)
- Setting a three-point lighting rig in Blender Internal
- Rendering an OpenGL playblast of the animation
- Obtaining a noise-free and faster rendering in Cycles
- Compositing the render layers

Introduction

In this last chapter, we are going to see recipes about the more common stages needed to render the complete final animation: lighting techniques in both the render engines, fast rendering previews, rendering settings, and the integrated compositing.

But first, let's see the necessary preparation of the 3D scene layout.