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.

Setting the library and the 3D scene layout

In this recipe, we are going to prepare a little both the file to be used as the library and the *hero* blend file, which is the file that will output the final rendered animation.

Getting ready

Start Blender and load the `Gidiosaurus_shaders_Blender_Internal.blend` file:

1. Go to the **Object Modifiers** window and check that the **Armature** modifiers are correctly enabled for *all the objects* (that is, the **Armature** modifiers must be enabled both for the rendering and for the 3D viewport visibility), then save the file.
2. Press *Ctrl* + *N* and click on the **Reload Start-Up File** pop-up panel to confirm a new brand file: immediately save it as `Gidiosaurus_3D_layout.blend`.

   ### Tip

   Saving the file at this point is necessary to automatically have a *relative path* for all the assets we are going to link.

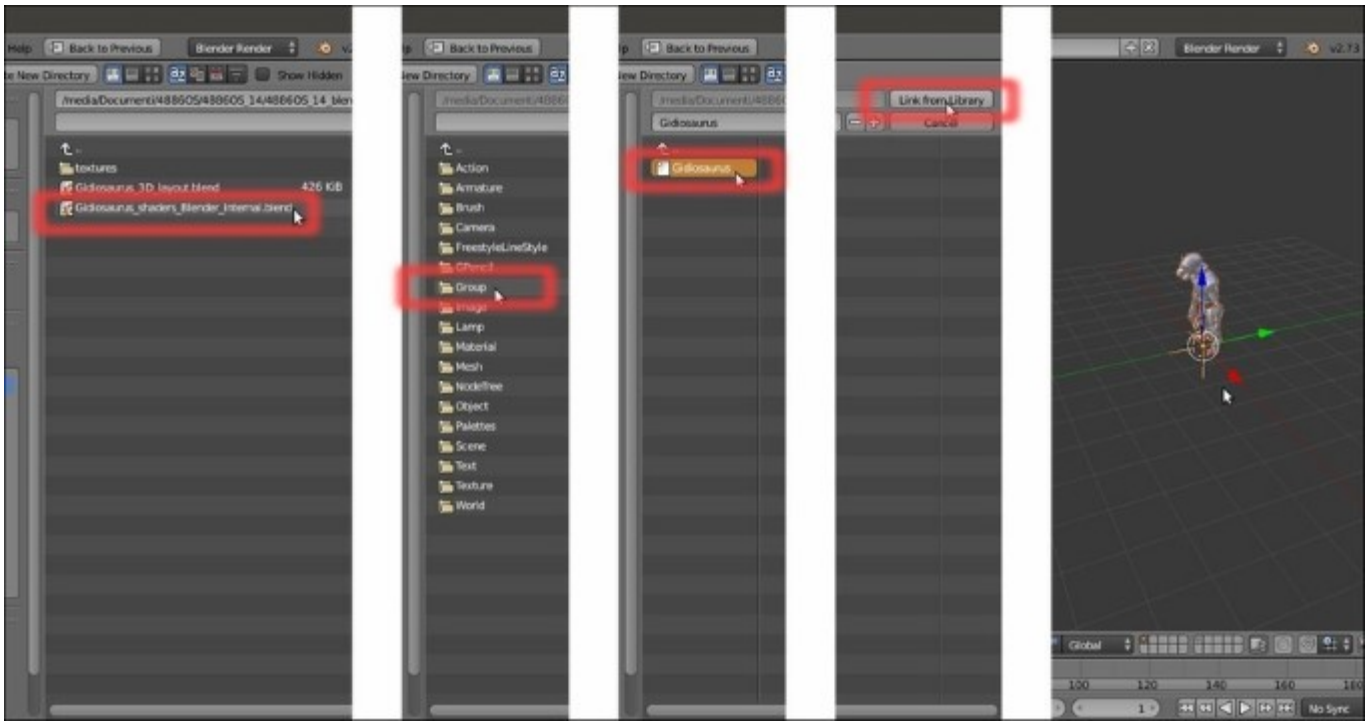
How to do it…

Let's load the assets as links in the file:

1. Select and delete (*X* key) the default **Cube** primitive in the middle of the 3D scene, then *Shift*-select both the **Camera** and the **Lamp** and move them (*M* key) to the **6th** scene layer.
2. Still in the **1st** scene layer, click on the **File** item in the top main header and then navigate to select the **Link** item; or else, just press the *Ctrl* + *Alt* + *O* keys shortcut.

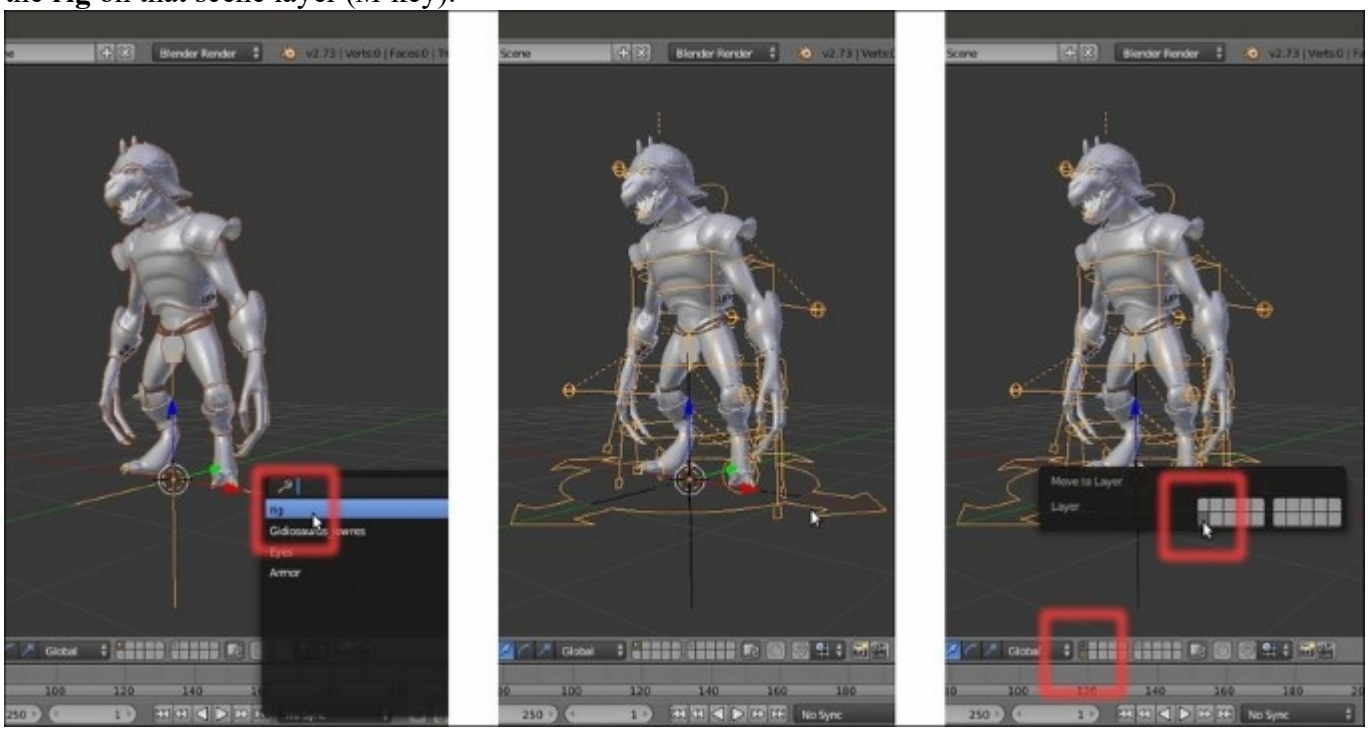
   In the blend files provided with this cookbook, I moved a copy of the `Gidiosaurus_shaders_Blender_Internal.blend` file to the `4886OS_14_blendfiles` folder, to simplify the process, but anyway:
3. Browse to the folder where the `Gidiosaurus_shaders_Blender_Internal.blend` file is saved; click on it and browse further to click on the **Group** item/folder, then select the **Gidiosaurus** item and click on the top right **Link from Library** button.

   The linked **Gidiosaurus** character appears at the **3D Cursor** position, in our case in the middle of the scene:



Linking the Gidiosaurus group

4. Zoom to the **Gidiosaurus** object and press *Ctrl* + *Alt* + *P* to make a proxy; in the pop-up menu panel that appears, select the proxified **rig** item, then *Shift*-enable the **11th** scene layer and move the **rig** on that scene layer (*M* key):

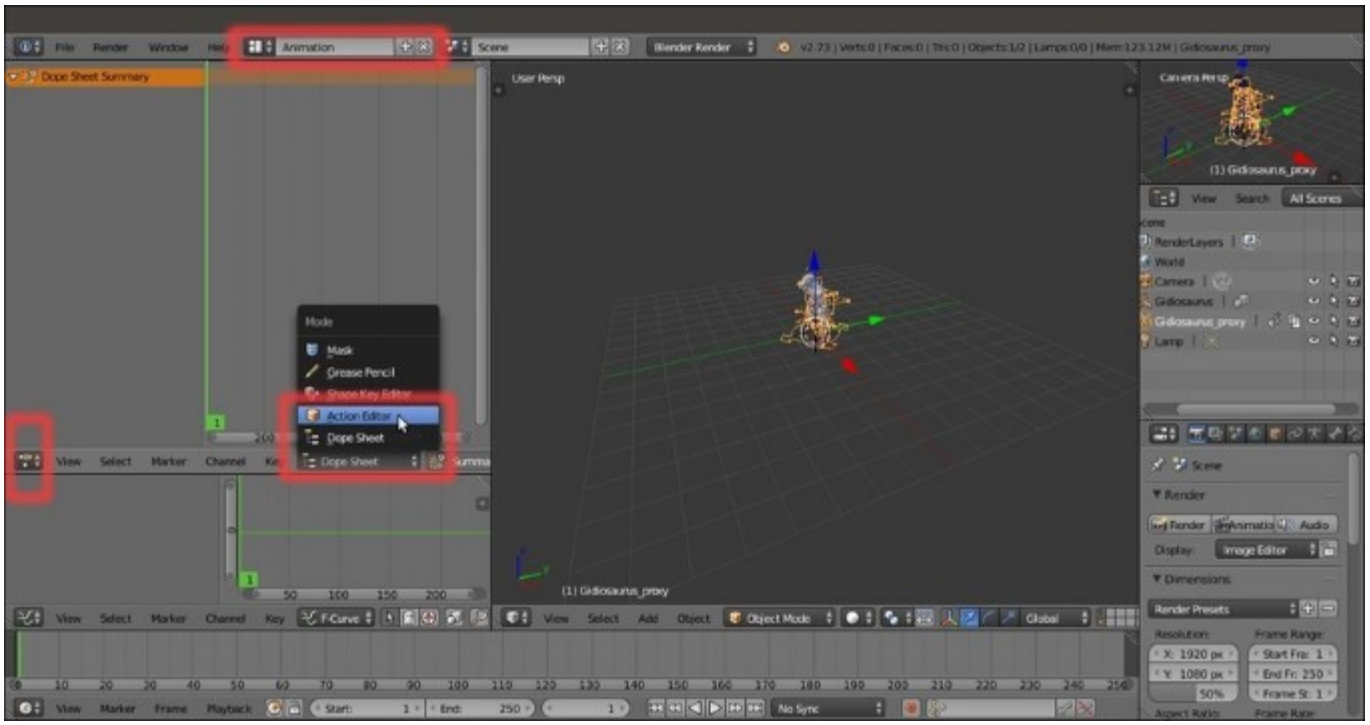


Making a proxy of the rig and moving it to the 11th scene layer

5. Click on the *Screen datablock* button on the top main header to switch from the **Default** screen layout to the **Animation** screen layout:

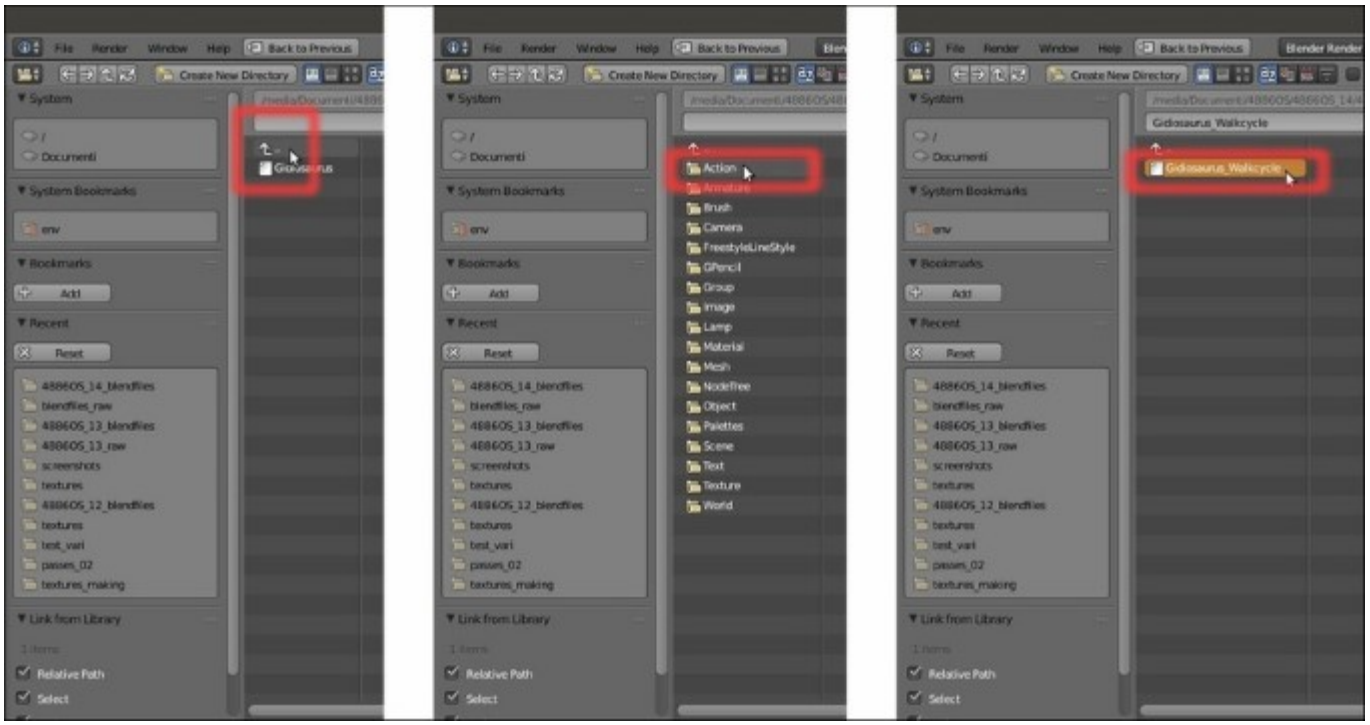
Switching to the Animation screen layout

6. Click on the **Mode** button in the **Dope Sheet** toolbar and switch from the **Dope Sheet** window to the **Action Editor** window:



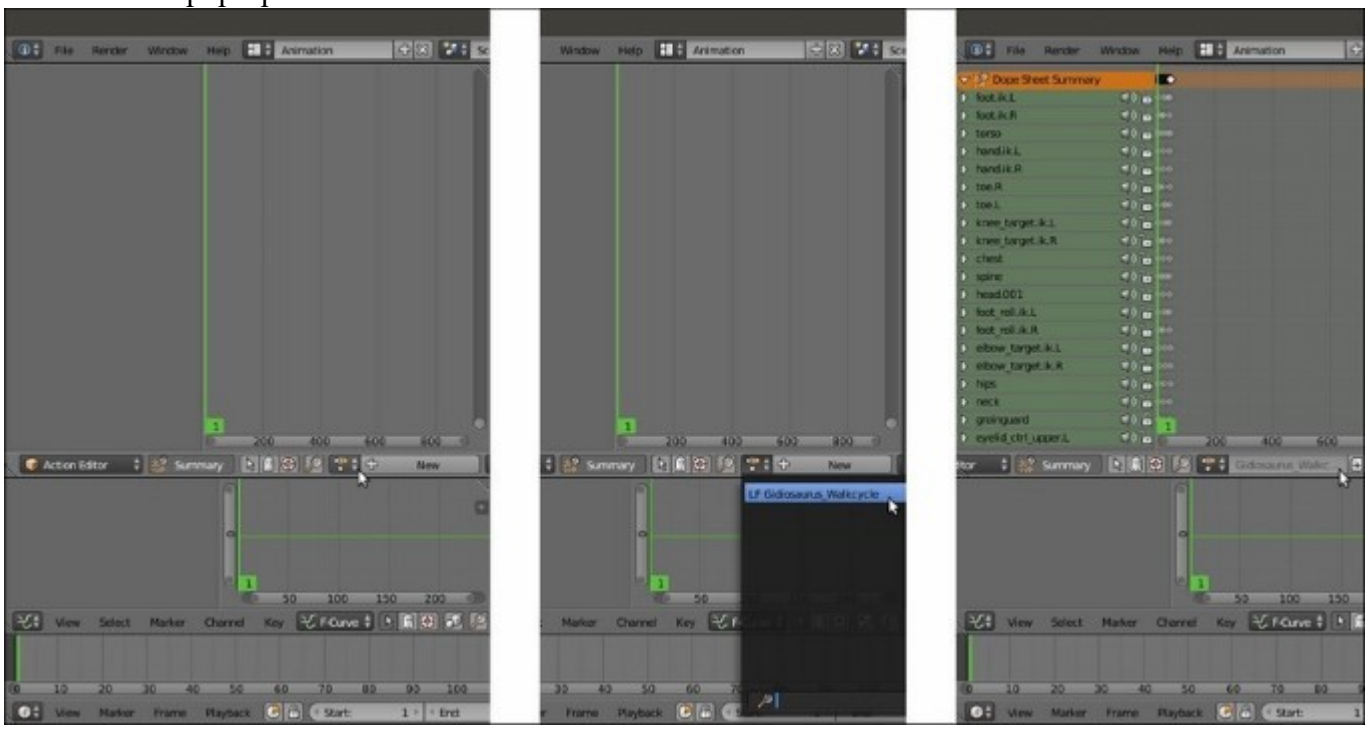
Switching to the Action Editor window

7. Click on the **File** item in the top main header and then navigate to select the **Link** item, or just press the *Ctrl* + *Alt* + *O* keys shortcut; the screen opens automatically at the last location we previously browsed to.
8. Click on the two dots above the **Gidiosaurus** item to navigate backward (to go up one level) and click on the **Action** item/folder to select the **Gidiosaurus_walkcycle** item; then click as before on the **Link from Library** button:



Browsing to the Action folder directory

9. Scroll a bit to the left of the **Action Editor** window's toolbar to reveal the **New** button; click on the double arrows to the left side of the **New** button to select the **LF Gidiosaurus_walkcycle** item from the pop-up menu.



Loading the linked action in the Action Editor window

10. Go back to the **Default** screen and click on the **Play Animation** button in the **Timeline** toolbar.

    Depending on the power of your system, you will see the animated character start to move, more or less fluidly, in the 3D viewport; the frame-rate (number of frames per second) played by Blender in real-time is shown in red at the top left corner of the 3D view-port:

The 3D view-port showing the animation and the frame-rate at the top left corner

    It should be around **24** frames per second; in my case, it barely arrives at **0.70** to **0.80**... so an arrangement must be found to show a faster and natural-looking movement.

11. Go to the **Scene** window and enable the **Simplify** subpanel: set the **Subdivision** level to **0**.

The Simplify subpanel under the Scene window in the main Properties panel

Without subdivision levels, even on my old laptop the real-time frame-rate is now **24** frames per second.

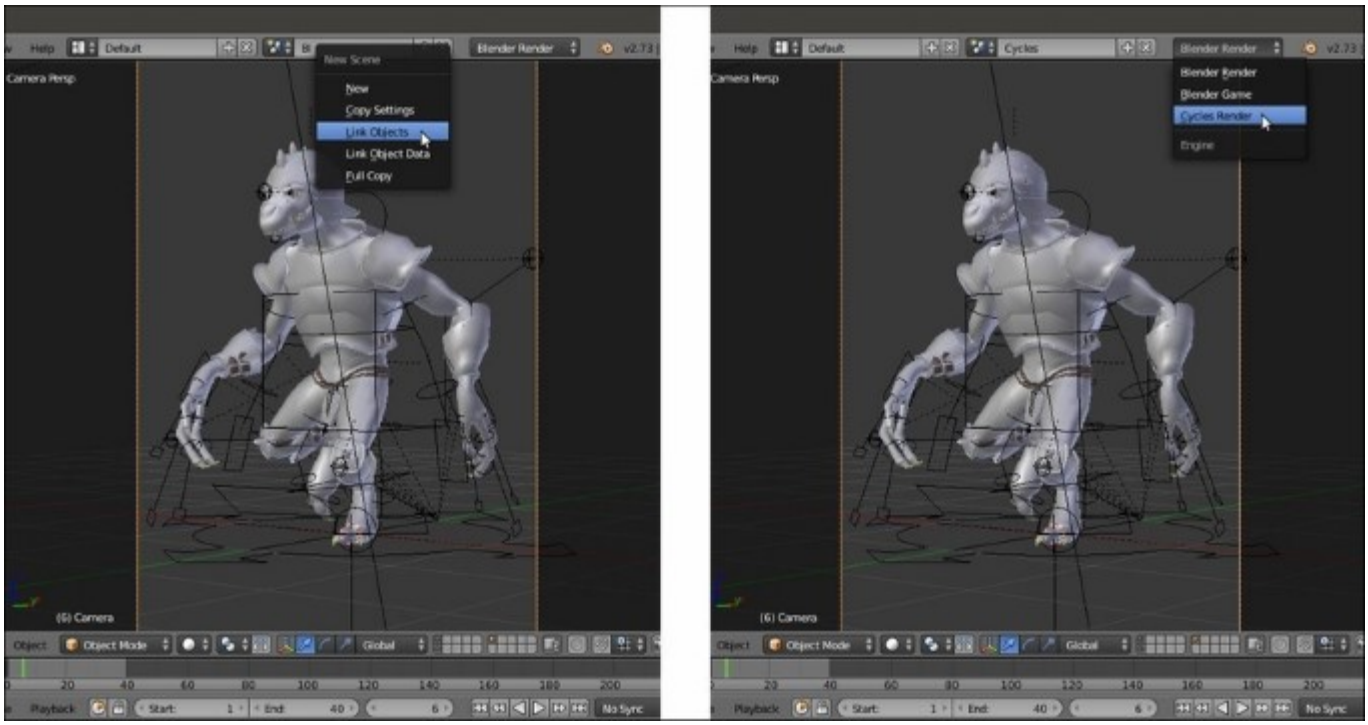
12. Go to the **Render** window and, in the **Dimensions** subpanel, under **Frame Range**, set the **End Frame** to **40**; under **Resolution** switch the **X** and **Y** values, that is **X** = **1080** px and **Y** = **1920** px.
13. Go to the **Outliner** and click on the **Display Mode** button to switch from **All Scenes** to **Visible Layers**. Then *Shift*-enable the **6th** scene layer and select the **Lamp**.
14. Go to the **Object Data** window and, in the **Lamp** subpanel, change the lamp type from **Point** to **Spot**; set the color to **R 1.000**, **G 1.000**, **B 0.650** and the **Energy** to **14.000**.
15. Put the mouse pointer in the 3D viewport and press *N* to call the side **Properties** panel; go to the top **Transform** subpanel and set **Location** as **X** = **6.059204**, **Y** = **-9.912249**, **Z** = **7.546275** and **Rotation** as **X** = **55.789°**, **Y** = **0°**, **Z** = **30.562°**.
16. Back in the **Object Data** window, go down to the **Shadow** subpanel and switch from **Ray Shadow** to **Buffer Shadow**; under **Filter Type** set the **Shadow Filter Type** to **Gauss**, the **Soft** to **12.000**, the **Size** to **4000** and the **Samples** to **16**. Set the **Clip Start** value to **9.000** and the **Clip End** value to **19.000**.

Setting the Lamp

17. Select the **Camera** and in the **Object Data** window set the **Focal Length** under **Lens** to **60.00**; in the **Transform** subpanel under the *N* side **Properties** panel input these values: **Location** as **X** = **5.095385**, **Y** = **-6.777483**, **Z** = **1.021429** and the **Rotation** as **X** = **91.168°**, **Y** = **0°**, **Z** = **37.526°**. Put the mouse pointer in the 3D viewport and press the *0* numpad key to go in **Camera** view:

Setting the Camera

18. Now click on the **Scene** datablock button on the top main header and rename it **BI** (**Blender Internal**). Click on the + icon button to the right and from the **New Scene** pop-up menu select the **Link Objects** item.
19. Change the **BI.001** name of the new scene in **Cycles** and click on the *Engine* button to the right to switch to the **Cycles Render** engine:



Adding a new scene with linked objects

20. Go to the **Outliner** and select the **Lamp**; go to the **Object Data** window and, in the **Nodes** subpanel, click on the **Use Nodes** button and then set the **Strength** to **10000.000**. Go to the **Lamp** subpanel and set the **Size** to **0.500** and enable the **Multiple Importance** item.
21. Save the file.

How it works…

A scheme of what we made in this recipe is: we prepared a blend file that *links* both the **character** and the **action**; this means that *neither* of them is *local* to the file and they cannot be directly edited in this file. Moreover, the character, in its library file, links the textures that are contained in the `textures` folder (which is at the same level as the blend files).

The **Simplify** subpanel in the **Scene** window allows us to *globally* modify some of the settings that can usually slow a workflow, such as the **subdivision** levels, the number of **particles**, the quality of **ambient occlusion** and **subsurface scattering**, and the **shadows** samples; through this panel they can be temporarily lowered or even disabled to have faster and more responsive previews of the rendering and the animation. Just remember that the **Simplify** subpanel also affects the rendering, so you have to disable it before starting the final rendering task.

See also

- http://www.blender.org/manual/data_system/introduction.html#copying-and-linking-objects-between-scenes
- http://www.blender.org/manual/data_system/scenes.html

- http://www.blender.org/manual/data_system/linked_libraries.html

Setting image based lighting (IBL)

The **image based lighting** technique is almost essential in computer graphics nowadays; as the name itself says, it's a technique to light a scene based on the pixel color information of an image, usually an **hdr** image (**High Dynamic Range** image); other image formats can also work, although not so well.

In Blender it's possible to obtain **IBL** both in **BI** and in **Cycles**, although with different modalities.

Getting ready

Start Blender and load the previously saved `Gidiosaurus_3D_layout.blend` file; save it as `Gidiosaurus_IBL.blend`.

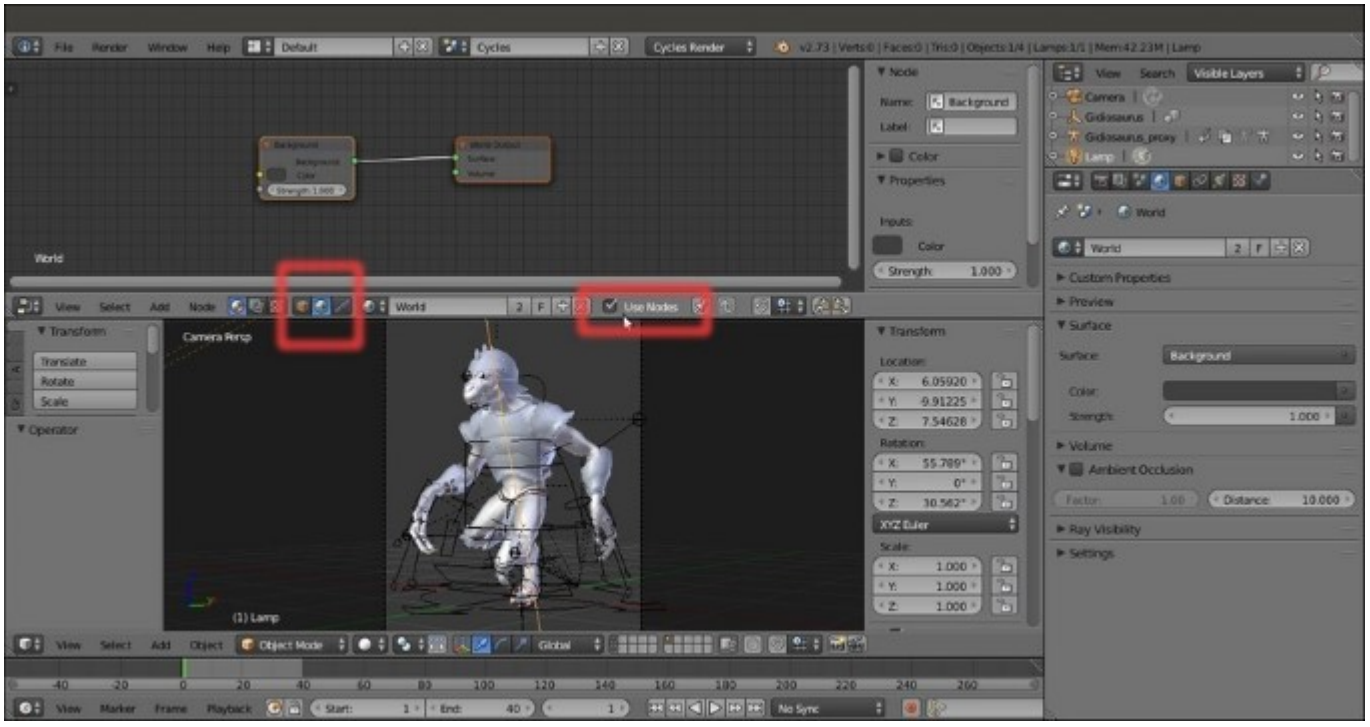
How to do it…

We can divide this recipe into two parts: **IBL** in **Cycles** and in **Blender Internal**.

Image based lighting in Cycles

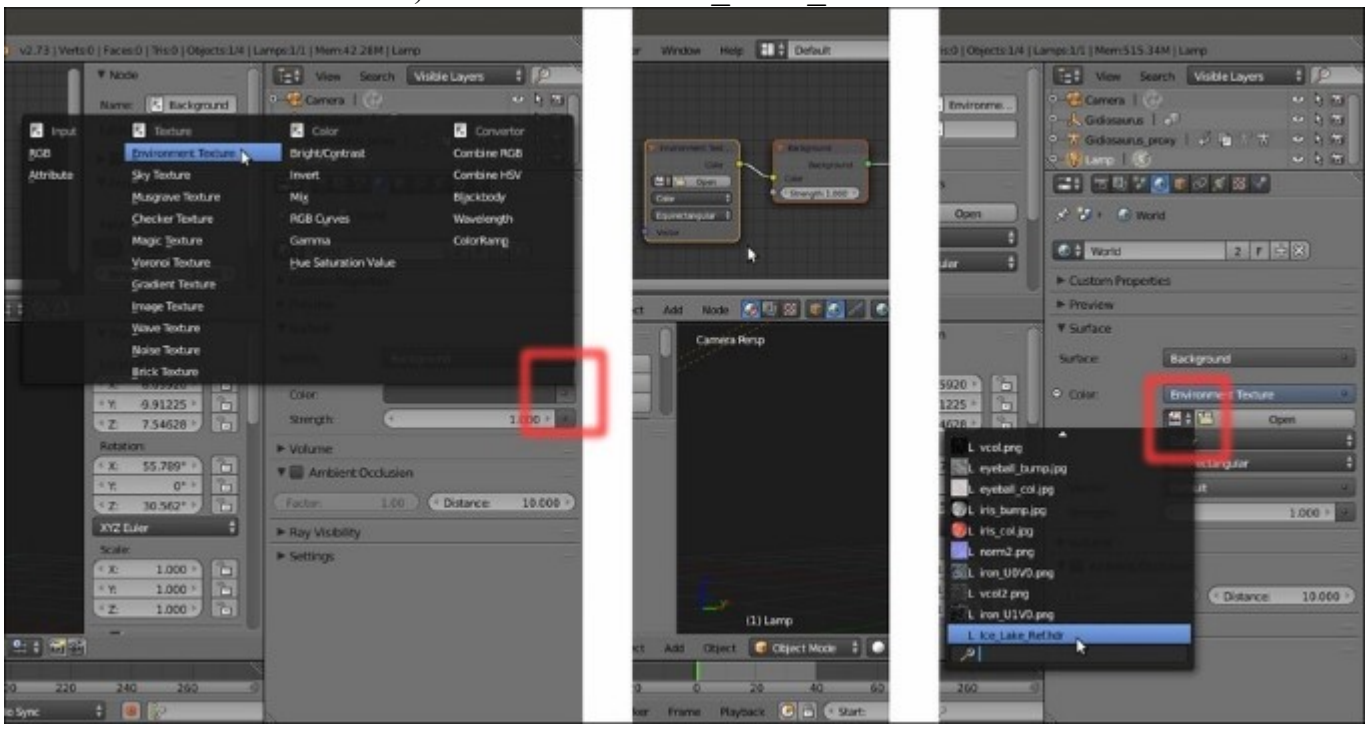
Let's start with the **Cycles Render** engine:

1. First, split the 3D window vertically into two windows, then change the upper one into a **Node Editor** window. In the toolbar, click on the **World** icon button to the right side of the **Object** icon button (selected by default; it's the one enabled for building the objects' shaders). Check the **Use Nodes** checkbox (or, click on the **Use Nodes** button inside the **Surface** subpanel in the **World** window); a **Background** node connected to a **World Output** node will appear in the **Node Editor** window.



Enabling the World nodes in the Node Editor window

2. Click on the dotted button to the right side of the **Color** slot in the **Surface** subpanel under the **World** window, to call the pop-up menu and select an **Environment Texture** node, which is automatically added and correctly connected to the **Color** input socket of the **Environment** node; then, click on the double arrows to the left side of the **Open** button (both in the **Node Editor** or in the **World** window) and select the `L_Ice_Lake_Ref.hdr` item.



Adding an Environment node to the World and loading the hdr image

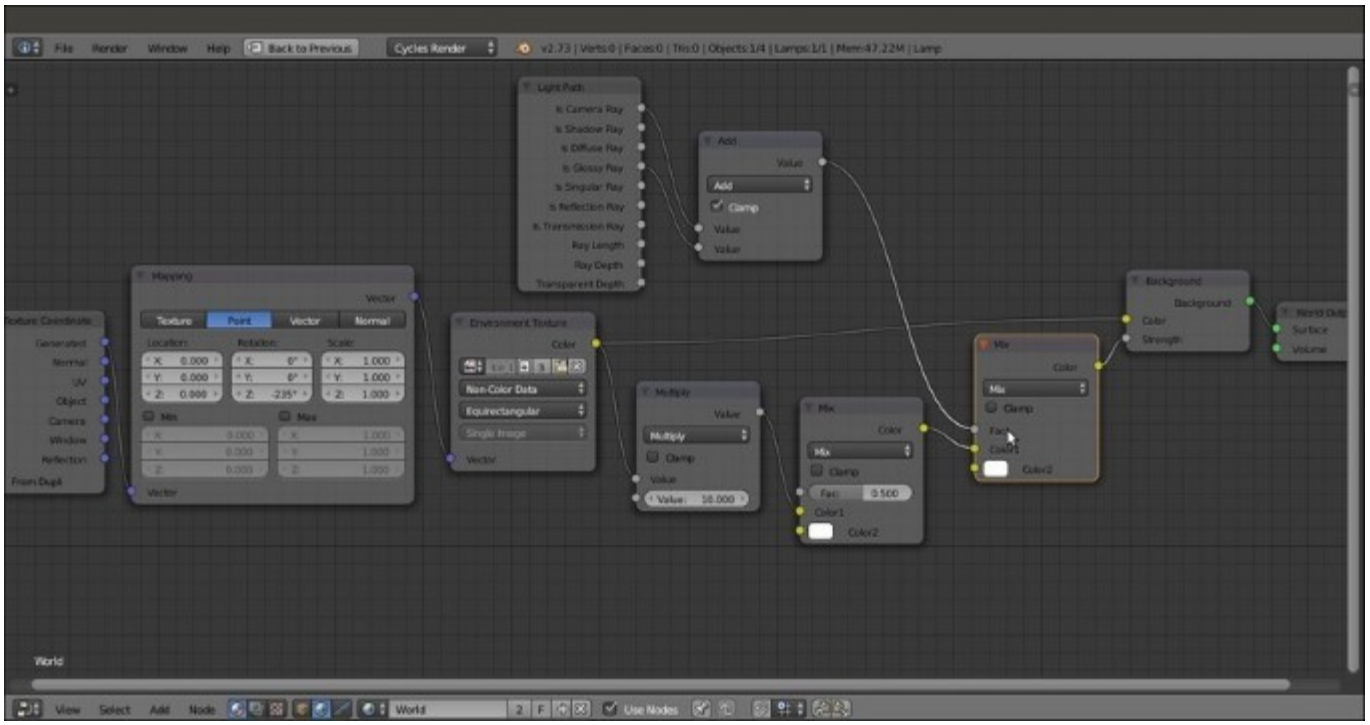
3. In the **World** window or in the **Node Editor** window, set the **Color Space** to **Non-Color Data**.
4. In order to gain some feedback, start the **Rendered** preview in the bottom **Camera** view, then go back to the **Node Editor** and add a **Texture Coordinate** node (*Shift* + *A* | **Input** | **Texture Coordinate**) and a **Mapping** node (*Shift* + *A* | **Vector** | **Mapping**).
5. Connect the **Generated** output of the **Texture Coordinate** node to the **Vector** input socket of the **Mapping** node and the output of this latter node to the **Vector** input socket of the **Environment Texture** node; set the **Rotation Z** value of the **Mapping** node to **-235**.

Rotating the hdr image to match the position of the Lamp

6. Now add a **Math** node (*Shift* + *A* | **Converter** | **Math**) and a **MixRGB** node (*Shift* + *A* | **Color** | **MixRGB**); connect the **Color** output of the **Environment Texture** 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 **10.000**.
7. Connect the output of the **Multiply-Math** node to the **Color1** input socket of the **MixRGB** node and set the **Color2** to pure white; connect the Color output of the **MixRGB** node to the **Strength** input socket of the **Background** node:

Adding nodes to the World

8. Press *Shift* + *D* to duplicate both the **Math** and the **MixRGB** nodes: paste the duplicated **MixRGB** node between the first **MixRGB** and the **Background** nodes; set the **Operation** of the duplicated **Math** node to **Add**.
9. Add a **Light Path** node (*Shift* + *A* | **Input** | **Light Path**); connect its **Is Camera Ray** output to the first **Value** input socket of the duplicated **Add-Math** node and the **Is Glossy Ray** output to the second **Value** input socket; connect the **Value** output of the **Add-Math** node to the **Fac** input socket of the second **MixRGB** node and enable the **Clamp** item:



The completed IBL World setup for the Cycles render engine

10. Go to the **Settings** subpanel and enable the **Multiple Importance** item, then click on the *World datablock* to change the name in **World_Cycles**.
11. Go to the **Render** window and in the **Film** subpanel enable the **Transparent** item.

Renaming the World and some more settings in the main window

12. Save the file.

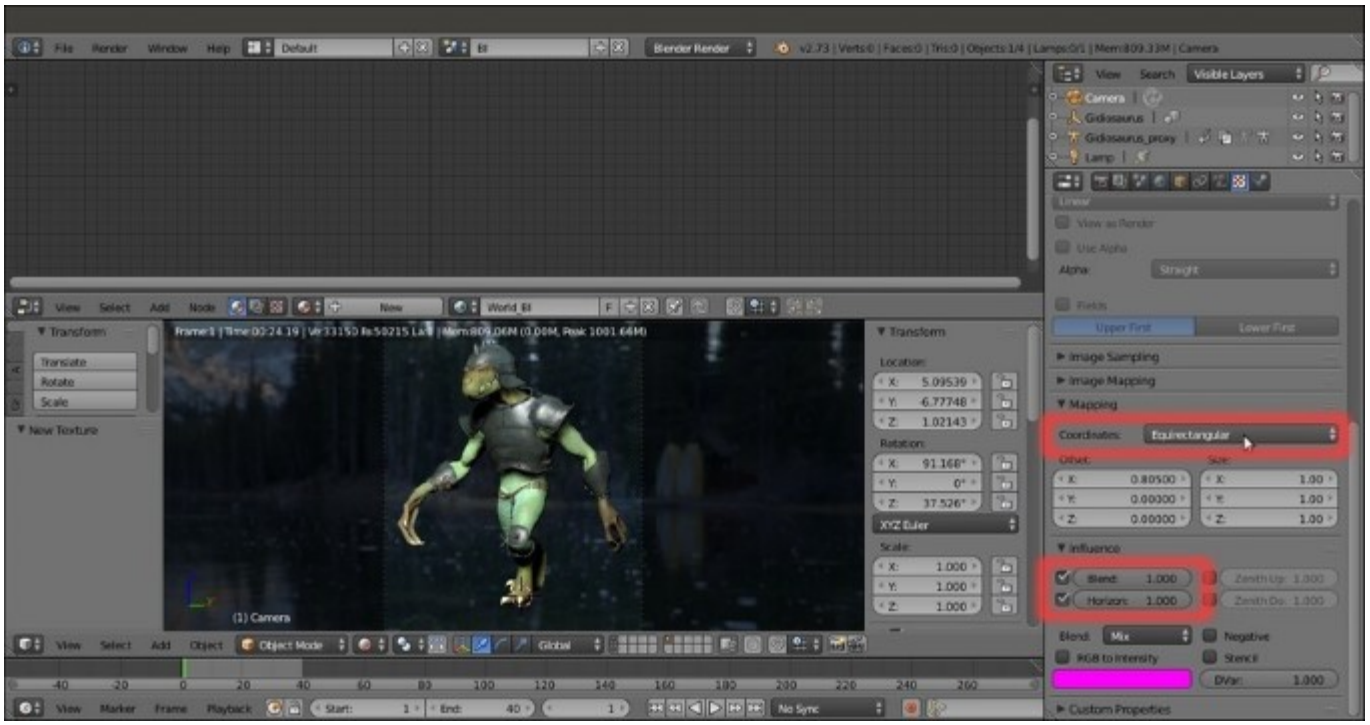
Image based lighting in Blender Internal

Now let's see the same thing in **Blender Internal**:

1. Click on the *Scene datablock* button in the top main header to switch from **Cycles** to **BI**.
2. In the **World** window to the right, click on the **2** icon button to the right side of the **World** name datablock to make it single user, then rename it **World_BI**.
3. Go directly to the **Texture** window: click on the **New** button, then click on the double arrows to the side of the image datablock to select the `L Ice_Lake_Refl.hdr` item from the pop-up menu:

Selecting the hdr image in the Blender Internal World

4. Rename the *ID name datablock* to **Ice_Lake_Refl**, then go down to the **Mapping** subpanel and click on the **Coordinates** slot to select the **Equirectangular** item; set the **Offset** to **X** = **0.80500** and then go further down to the **Influence** panel and enable the **Horizon** item.



First BI World settings

5. Back in the **World** window, in the **World** subpanel enable the **Real Sky** item.
6. Enable the **Environment Lighting** subpanel and click on the **Environment Color** button to select the **Sky Texture** item.
7. In the **Gather** subpanel, enable the **Approximate** method, the **Pixel Cache** item, the **Falloff** item and set the **Strength** value of this latter item to **0.900**.

More BI World settings

8. Go to the **Render** window and in the **Shading** subpanel click on the **Alpha Mode** slot to switch from the **Sky** to the **Transparent** item:

Enabling the transparent background for the rendering

9. Save the file.

How it works…

In **Cycles**: at steps 6 and 7 we added nodes to increase the source light intensity of the **hdr** image; because this also increased the contrast of the image, at steps 8 and 9 we made it less contrasted again but kept the same light intensity, thanks to the **Light Path** node. The light rays shoot from the **Camera** position and directly hit a surface (**Is Camera Ray**) or any glossy surface (**Is Glossy Ray**) and have value = **1.000**, hence corresponding to the **Color2** socket of the second **MixRGB** node, therefore giving a pure white (**1.000**) value to the **Background** node's **Strength**; any other ray (transmitted, shadows, reflected, transparent, and so on) has the high contrast **Strength** values we established at steps 6 and 7.

We used the **Mapping** node for the sole reason of matching (visually and thanks to the **World Background** item enabled in the **Display** subpanel under the *N* side **Properties** panel) the source light direction of the image with the position of the **Lamp** in the 3D scene: that's why we rotated the **hdr** image to negative **235** degrees on the *z* (vertical) axis.

In **Blender Internal**: we can't rotate the image, so instead we offset it on the *x* axis to (almost perfectly) match the position it has in **Cycles**.

The **Approximate** gathering method is the one developed during the production of the short open movie *Big Buck Bunny* (https://peach.blender.org/) to have faster rendering and absence of noise in **Ambient Occlusion**, inevitable with the default **Raytrace** method (that still remains the more accurate, by the way).

Note that, in both the render engines, we didn't load a brand new `Ice_Lake_Ref.hdr` image from the `textures` folder, but we instead used the linked one coming from the materials of the character, as indicated by the `L` in front of the name and by the name itself and all the settings grayed in the image datablock subpanel.

See also

The free **sIBL** addon currently, only works with **Cycles** materials but it can read the `.ibl` file provided with the free **hdr** images at the **sIBL Archive** (link provided further) and therefore, in one click, it can create the complete nodes setup to provide image based lighting in Blender.

- The official documentation about the addon (http://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/Import-Export/sIBL_GUI)
- An updated and bug-fixed version of the addon (https://raw.github.com/varkenvarken/blenderaddons/master/sibl.py)
- The sIBL archive (http://www.hdrlabs.com/sibl/archive.html)
- Official documentation about the **World** in Blender:
    - http://www.blender.org/manual/render/cycles/world.html
    - http://www.blender.org/manual/render/blender_render/world/index.html

Setting a three-point lighting rig in Blender Internal

Thanks to the global illumination, a path-tracer like **Cycles** doesn't necessarily need big lighting setups; in fact, in the recipes we made, we only used one single **Spot** lamp in addition to the **IBL** and the results have been quite good anyway.

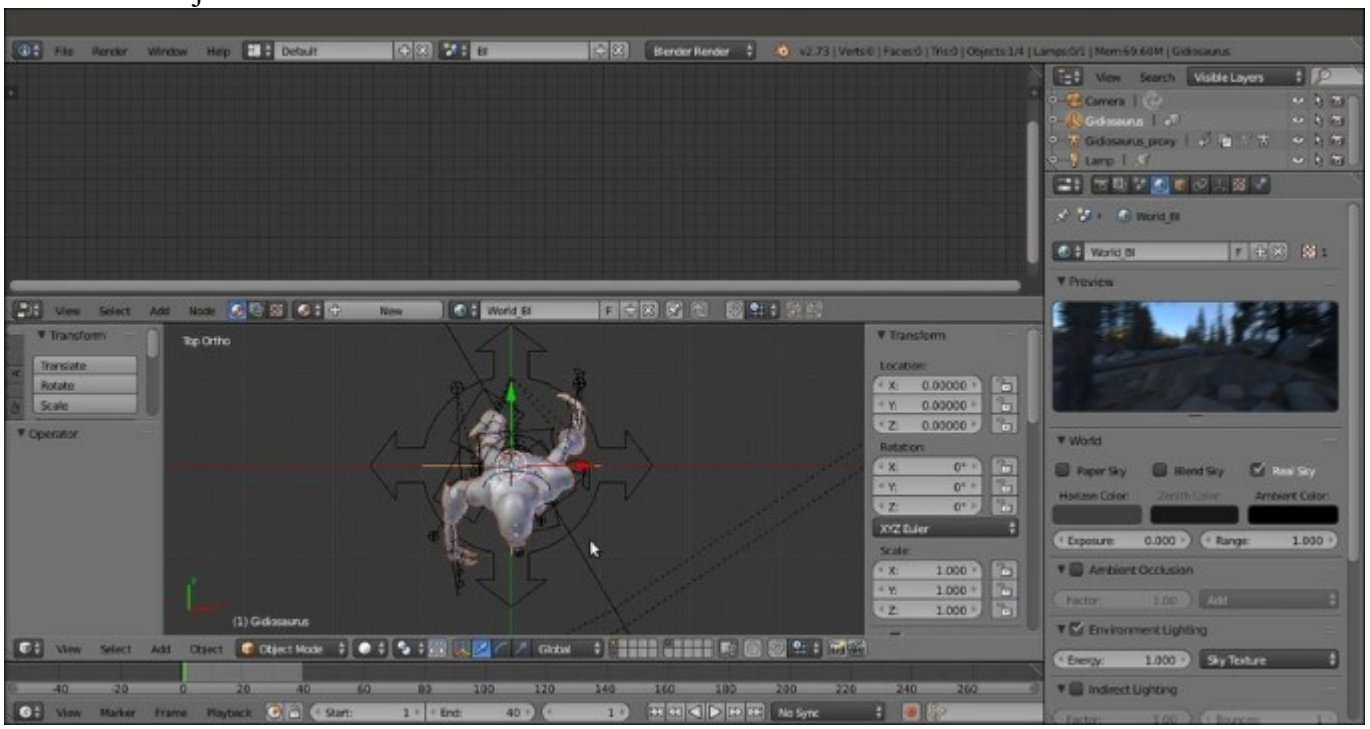
In **Blender Internal**, instead, a minimum arrangement of lamps must be done to obtain satisfying results, even with the aid of the **World** settings we have previously seen.

In this recipe, we are therefore going to see a classic *movie* three-point lighting rig, an industry standard. The effect of the main **key light** is enhanced by the other two lamps: the **fill light**, to brighten (and color) the shadow areas on the subject, and the **backlight**, to create a light rim on the subject edges thus making it stand out against the background.

Getting ready

Start Blender and load the previously saved `Gidiosaurus_IBL.blend` file; if necessary, switch to the **Blender Render** engine by the *Engine to use for rendering* button in the top main header.

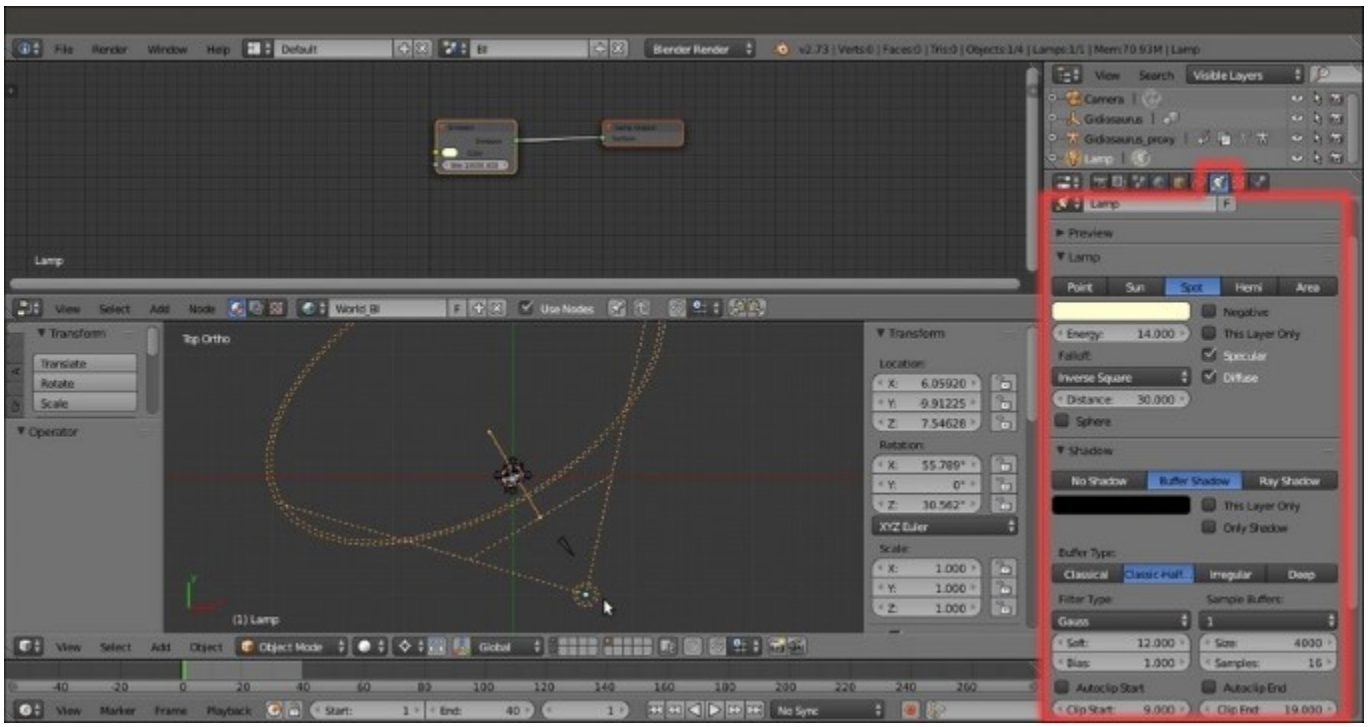
1. Put the mouse pointer inside the **Camera** view and press the numpad *7* key to go in **Top** view, then press numpad *5* to switch from **Perspective** to **Ortho** view.
2. Press *Shift* + *C* to put the **3D Cursor** at the center of the scene and then go to the **Outliner** to select the **Gidiosaurus** item: press the numpad period (.) key to center and zoom the view on the selected object:



Centering the top view on the character

3. Press *Ctrl* + Spacebar to disable the widget and scroll the mouse wheel to zoom backward and show the **Spot** lamp, then press the dot (.) key to switch the **Pivot Point** from **Median Point** (or whatever else) to **3D Cursor**.
4. Select the **Lamp**, then go to the **Object Data** window.
5. Remember to go to the **Scene** window and disable the **Simplify** subpanel!
6. Save the file as `Gidiosaurus_lighting.blend`.

Don't take into consideration the **Node Editor** window at the top showing the **Lamp** nodes under **Cycles**; the settings to look for are those inside the main **Properties** panel to the right:

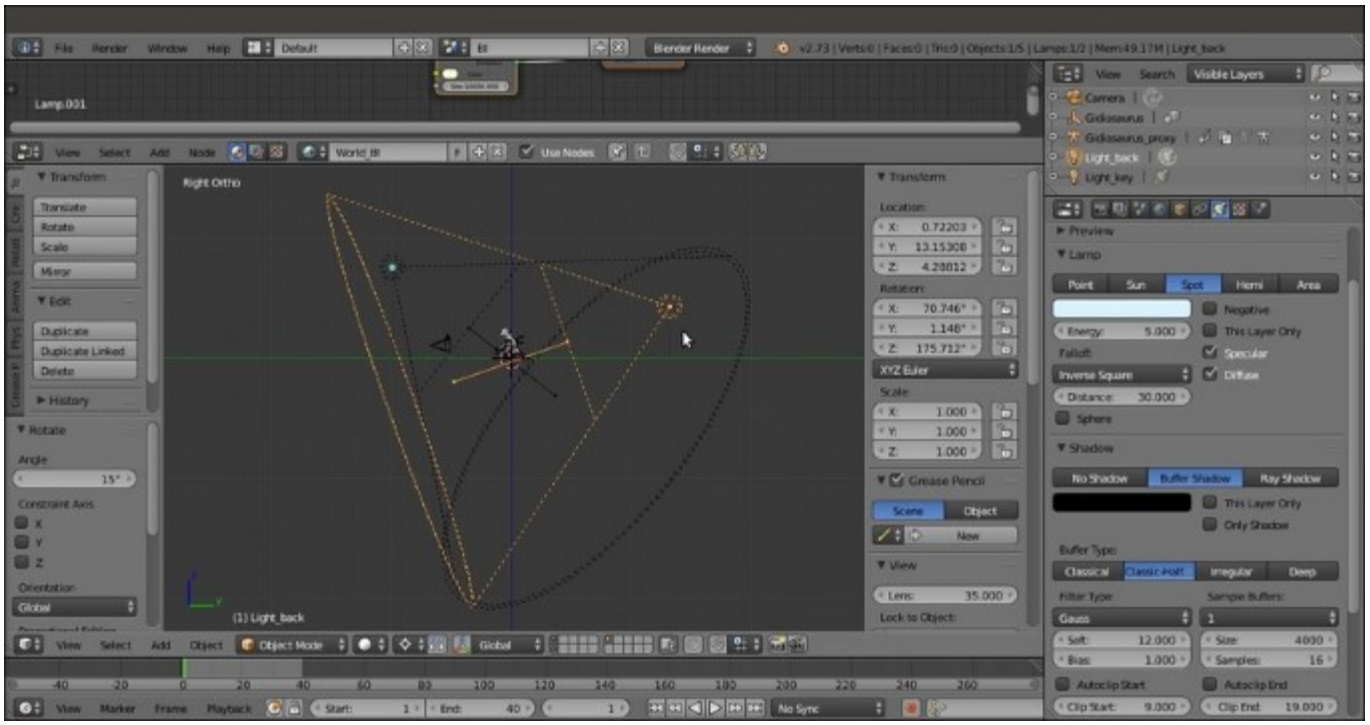


These are the settings you are looking for...

How to do it…

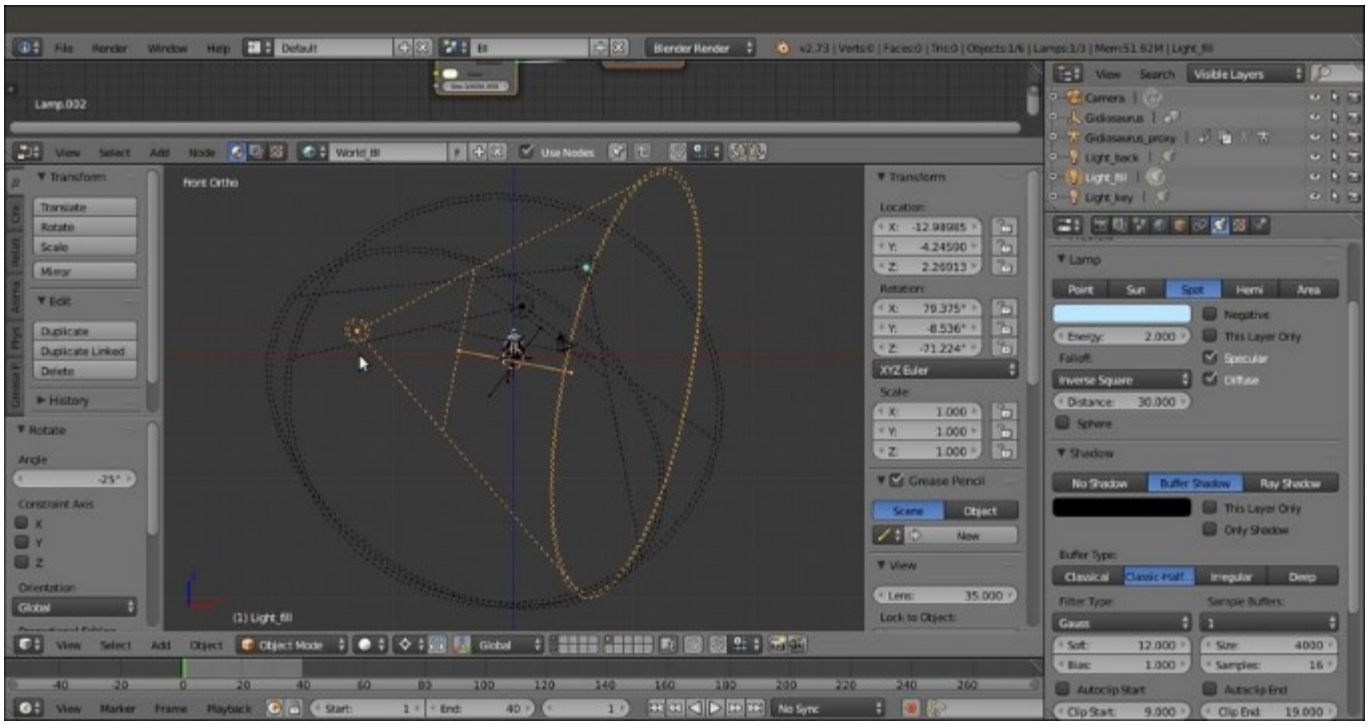
Let's go with the settings of the lights:

1. In the **Outliner**, rename the **Lamp** item as **Light_key**.
2. Press *Shift* + *D* to duplicate the **Key_light** lamp, press *R* and, while still in **Top** view, rotate the duplicated lamp approximately **-145** degrees, then go in **Side** view (numpad *3* key) and rotate it around **15** degrees: in the **Outliner**, rename it as **Light_back**.
3. In the **Object Data** window, set the color to a light blue = **R 0.700**, **G 0.900**, **B 1.000** and the **Energy** to **5.000**:



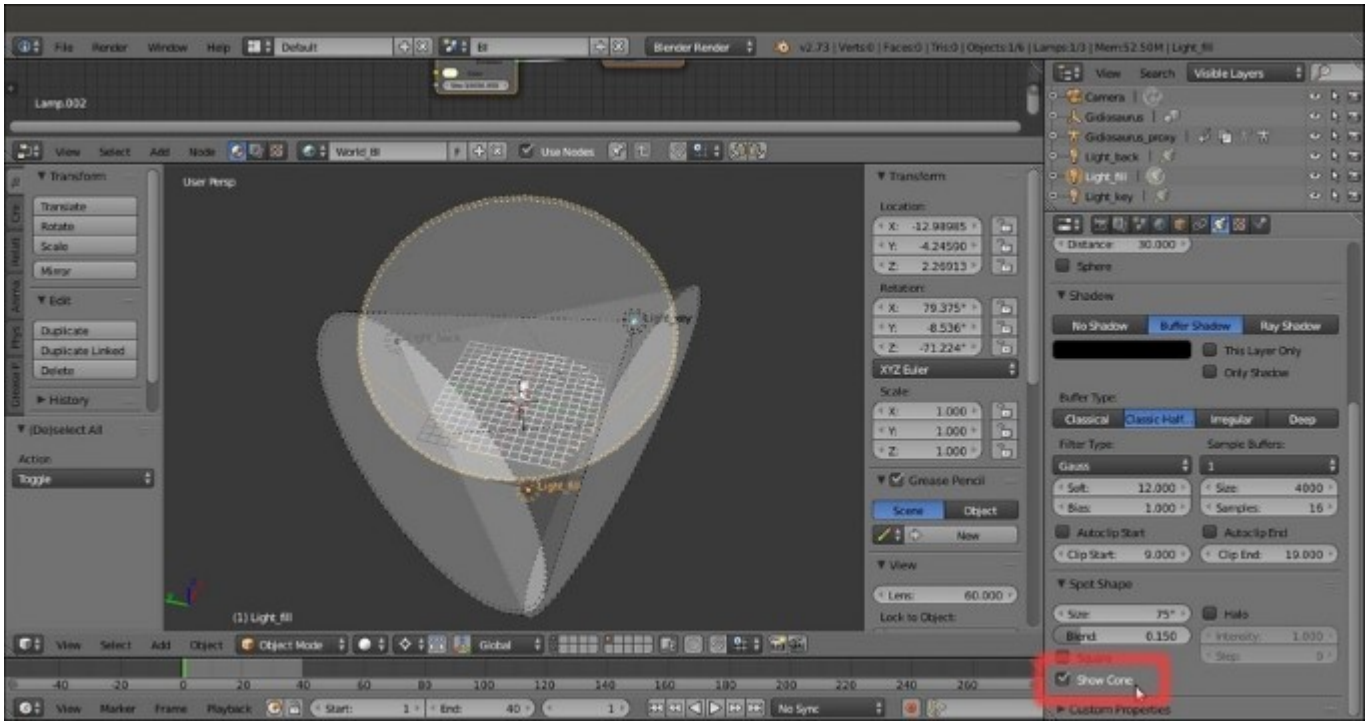
Positioning the Light_back lamp

4. Go back in **Top** view (numpad *7* key) and re-select the **Light_key** lamp, press *Shift* + *D* and rotate the duplicate by **100** degrees; in the **Outliner**, rename it as **Light_fill**. Go in **Front** view (numpad *1* key) and rotate it around **-25** degrees.
5. In the **Object Data** window, set the color to a lighter blue = **R 0.500**, **G 0.800**, **B 1.000** and the **Energy** to **2.000**:



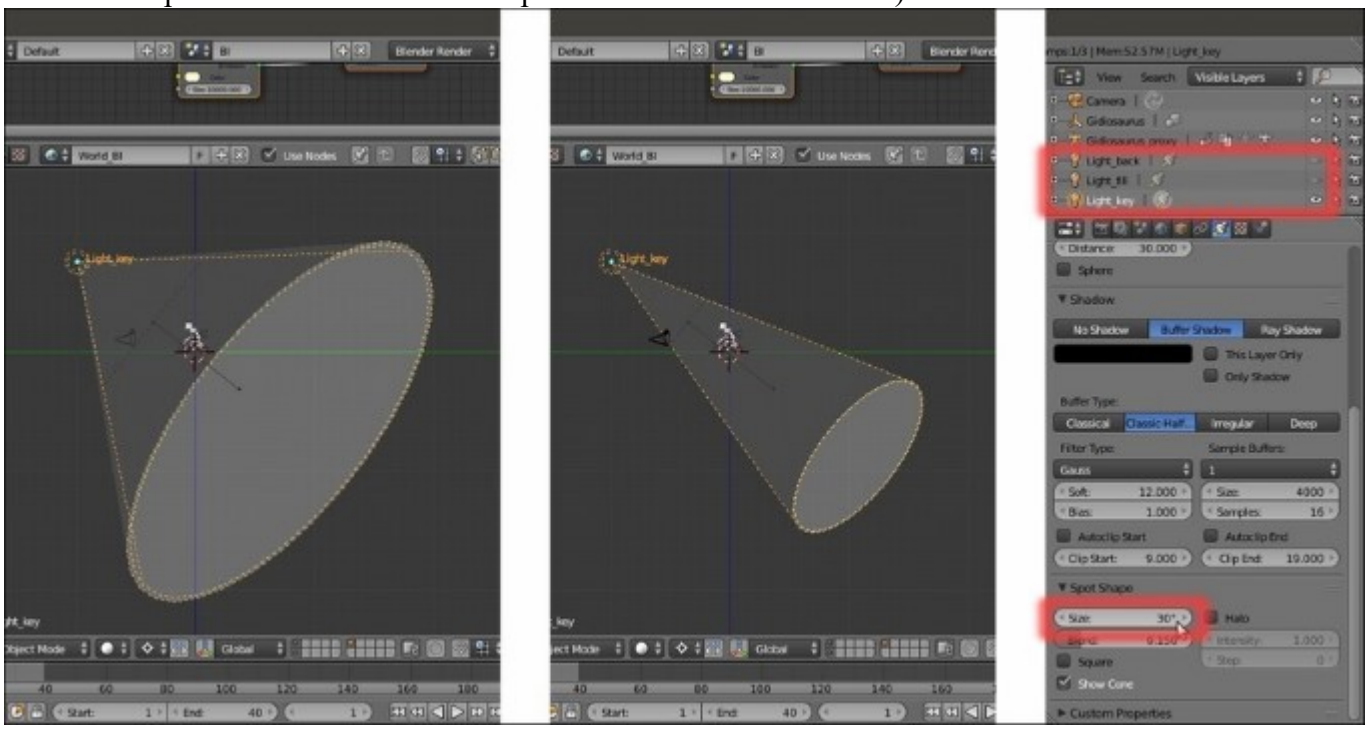
Positioning the Light_fill lamp

6. Go to the **Object** window and in the **Display** subpanel enable the **Name** item for the three lamps; then, back to the **Object Data** window and in the **Spot Shape** subpanel, enable the **Show Cone** item for each one:



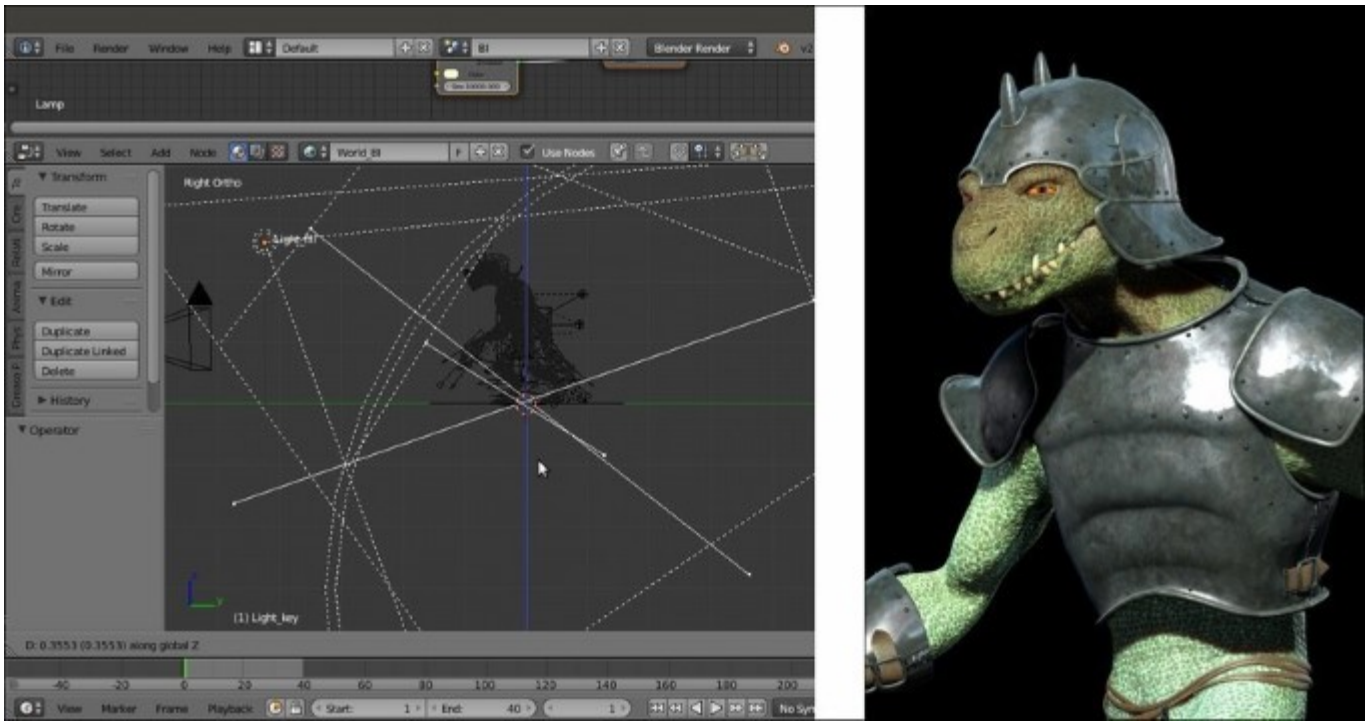
The three spot lamps showing their cones of influence

7. In the **Outliner**, disable the 3D viewport visibility of the **Light_back** and **Light_fill** lamps by clicking on the respective eye icon, then go in **Side Ortho** view and select the **Light_key** lamp.
8. Go to the **Spot Shape** subpanel again and lower the **Size** value from the default **75°** to **30°** (or the smallest possible value that still comprehends the whole character):



Lowering the spot lamp size value

9. Repeat steps 7 and 8 for the other two lamps as well, then *Shift*-select all the three lamps and move them upward (on the *z* axis) a bit, just to better center the light cones' centers on the position of the feet of the character.
10. Save the file.



The final result of the three-point lighting rig

How it works…

A classic three-point lighting rig can in some way compensate for the lack of real global illumination in **Blender Internal**, although to obtain really good results, three lamps are usually not enough; in any case, the lighting rig of this recipe can be used as a base for even more complex setups.

When using more than one lamp in **Blender Internal**, we should always be sure that the shadows are enabled for all of them, unless we want particular effects; in fact, a back lamp with disabled shadows can easily *shine* through the model and also illuminate parts that shouldn't be in light, giving unrealistic results.

To calculate the buffered shadows, **Spot** lamps take into consideration everything inside their cone from the **Clip Start** to the **Clip End** values; this is why we lowered the **Size** values of the cones as much as possible.

One other crucial factor that can slow the calculation and the rendering times is, obviously, the size of these buffers, which we set to **4000** for each one of the three lamps; quite big, but because we set the cones that large enough to just comprehend the shape of their target object. This means we could use big shadow buffers, to obtain more details in the shadows if needed.

We do all of this, even though the **Gidiosaurus** was the only object to be rendered in the scene.

See also

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

Rendering an OpenGL playblast of the animation

Playblast is a term used by a famous commercial package to indicate the preview of the animation in true speed; although I've heard only very few people using it in relation to Blender, I thought it might be a good way to indicate the fast OpenGL preview rendering obtained for checking the animated action.

Getting ready

Start Blender and load the `Gidiosaurus_lighting.blend` file.

1. In the **Outliner**, select the **Light_key** lamp item and go to the **Object Data** window, under the **Spot Shape** subpanel, to disable the **Show Cone** item.
2. Repeat the procedure for the **Light_back** and **Light_fill** lamps, then disable their visibility in the 3D viewport by clicking on the respective eye icon.
3. Disable the visibility in the viewport for the **Gidiosaurus_proxy** item (the linked and proxified rig) also and/or disable the **11th** scene layer.
4. Save the file as `Gidiosaurus_playblast.blend`.

How to do it…

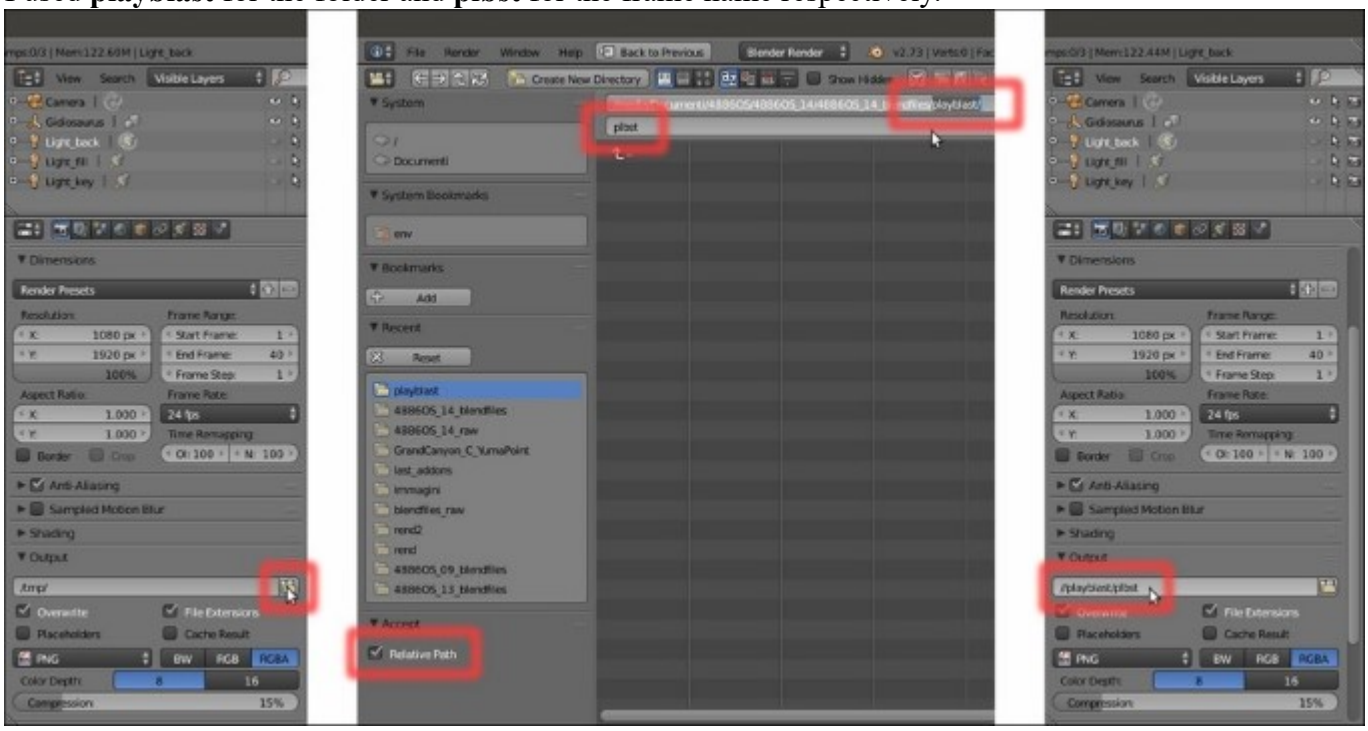
Here are the steps to begin with the OpenGL rendering:

1. Put the mouse pointer inside the 3D viewport and press the numpad *0* key to go in **Camera** view; press the *Z* key to go in **Solid** viewport shading mode, then scroll the mouse wheel to zoom the **Camera** view inside the window:

The Camera view in Solid viewport shading mode

2. Go to the **Render** window and to the **Dimensions** subpanel; check for the **X** and **Y** sizes of the rendering under **Resolution**, specified in pixels, and move the *Percentage scale for render resolution* slider, usually set to **50%**, to **100%**.
3. Go down to the **Output** subpanel and click on the folder icon button to the end of the path slot; browse to the location you want to save your rendering, then type in the first line of the path to the folder you want to create at that location, followed by the slash (/) and press *Enter*.
4. A pop-up will ask you to confirm the creation of the new directory; confirm and then type a generic frame name in the second line, go to the left side vertical bar to be sure that the bottom **Relative Path** item is enabled and finally click on the **Accept** button at the top left of the screen.

   I used **playblast** for the folder and **plbst** for the frame name respectively.



The new directory and the rendered frames name

5. Save the file, then go to the **Camera** view toolbar and click on the last *ciak* icon button to the left to start the OpenGL playblast:

The two buttons to start the OpenGL rendering (for a still to the left, for the animation to the right)

How it works…

In our example, the OpenGL playblast rendered single `.png` images with an alpha background because, as you can see in the **Render** window visible in the previous screenshot, these are the settings of the **Output** subpanel. Be aware that the resolution, the format and the path where the playblast frames are saved, always depend on the settings in the **Render** window, the same settings that will be used for the final real rendering (but of course the resolution of the playblast can be easily and temporarily be made smaller with the slider of the percentage scale).

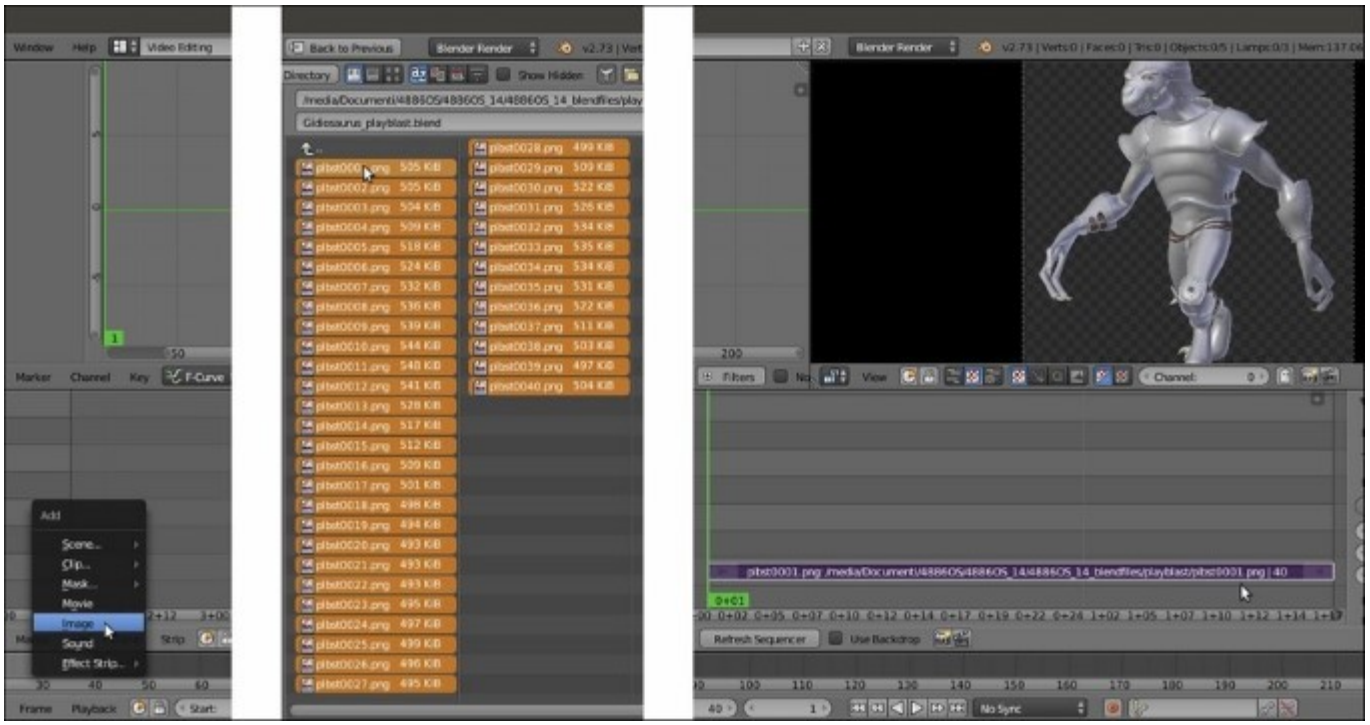
There's more…

Once we have rendered all the frames, we can use an external player to see them in sequence (in **Ubuntu**, I use the free player **DJV Imaging**, http://djv.sourceforge.net) or, just quickly build a movie through the **Blender Sequencer**:

1. Go to the *Screen datablock* button on the top main header and click it to switch to the **Video Editing** screen:

Switching to the Video Editing screen layout

2. Put the mouse pointer in the **Video Sequence Editor** window at the bottom and press *Shift* + *A*; from the pop-up menu select the **Image** item (*Add an image or image sequence to the sequencer*), then browse to the `playblast` folder location, click on it and once inside, press the *A* key to select all the contained frames, then press *Enter* to confirm. The frames are added to the **Video Sequence Editor** window as a single strip and the current frame appears in the preview window:



Loading the rendered frames in the Video Sequence Editor

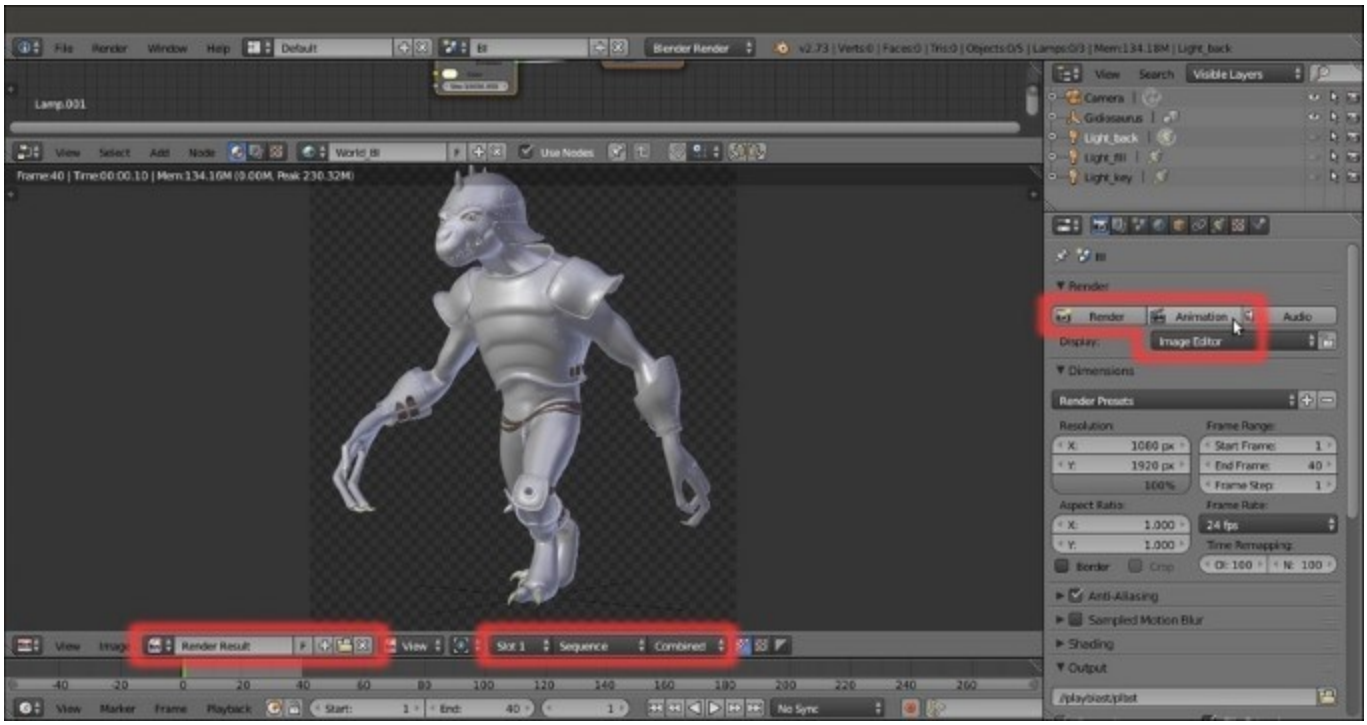
3. Go back to the **Default** screen and to the **Render** window under the main **Properties** panel. In the **Output** subpanel, where you can change the path to save the movie in a different location (or also leave it as it is), click on the **File Format** button to select a **Movie** format, for example, **AVI JPEG**. Choose **BW** or **RGB** and the **Quality** compression ratio (but the default **90%** is usually OK); then go to the **Post Processing** subpanel and ensure that the **Sequencer** item is enabled:

The Output and the Post Processing subpanels inside the Render window

4. Go to the top of the **Render** window and click on the **Animation** button; remember that Blender uses two different buttons to start the rendering of a still image or of an animation, both for the final rendering and for the 3D viewport toolbar OpenGL preview we have seen in the *How to do it…* section.

The rendering starts and the **Sequencer** processes all the `.png` images outputted by the playblast, transforming them into a single compressed `.avi` movie then saved in the same directory as the frames.

The process is visible in the **UV/Image Editor** window that replaced the **Camera** view, indicated in the toolbar by the **Render Result** label on the image datablock to the left (because the **Image Editor** item is the one selected in the **Display** slot under the **Render** subpanel) and by the **Sequence** label visible in the **Layer** slot to the right:



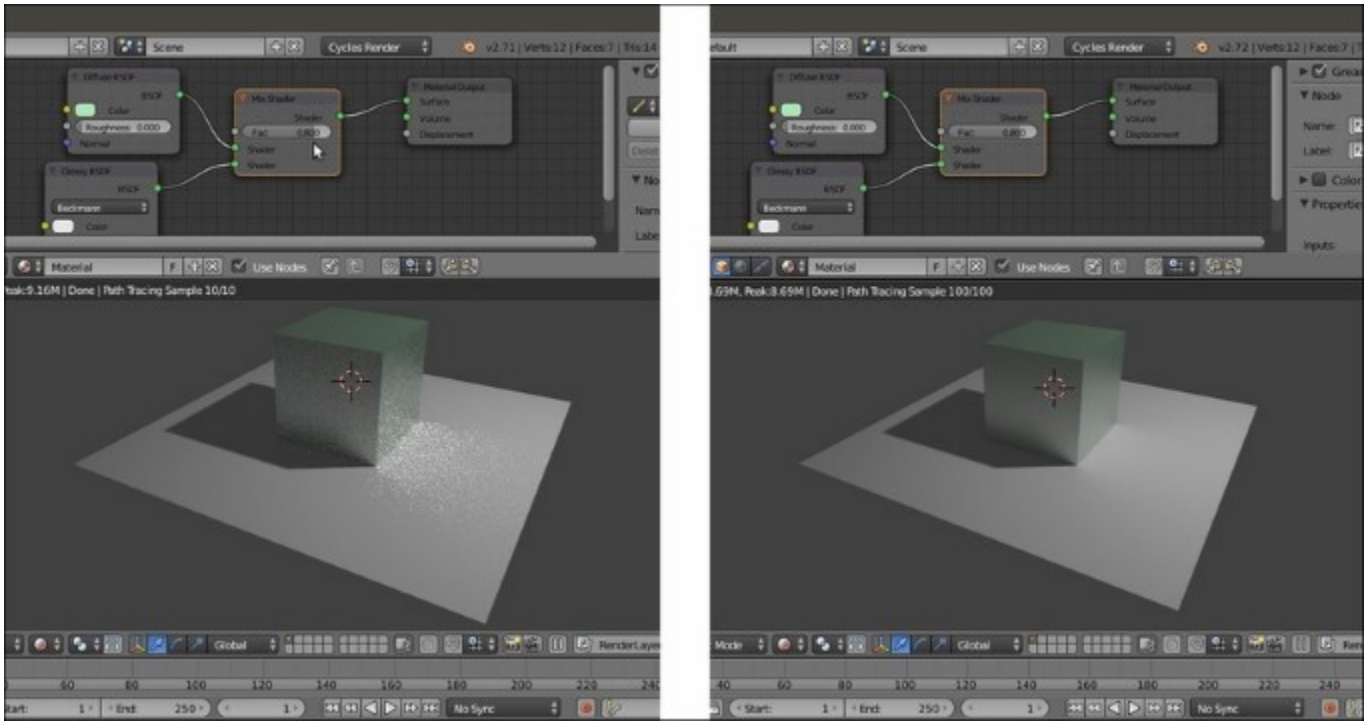
The Render Result window

See also

- http://www.blender.org/manual/render/display.html
- http://www.blender.org/manual/render/output/video.html
- http://www.blender.org/manual/render/output.html

Obtaining a noise-free and faster rendering in Cycles

The **Cycles Render** engine can be very slow compared to **Blender Internal**; by the way, some of the rendering settings can be tweaked to make it work faster; the goal here is to avoid fireflies and noise, usually due to low samples and a light source that is too bright.



Rendered previews of an example scene, showing a cube on a plane with and without noise and fireflies

Getting ready

Start Blender and load the `Gidiosaurus_playblast.blend` file.

1. Click on the *Scene datablock* button in the top main header to switch from **BI** to **Cycles**.
2. Go to the **Outliner** and enable the visibility of the **Light_key** lamp in the viewport by clicking on the grayed eye icon.
3. Put the mouse pointer inside the **Camera** view and press *Shift* + *B* to draw a box around the character's head, then zoom to it.
4. Save the file as `Gidiosaurus_render.blend` and press *Shift* + *Z* to start the **Rendered** preview.

How to do it…

If you have a capable graphic card supporting **GPU** (go to the last *See also* section for the link to a list of supported **GPU** graphic cards for **Cycles**), the next thing to do is:

1. Call the **User Preferences** panel (*Ctrl* + *Alt* +*U*) and go to the **System** tab; on the bottom left there is the **Compute Device** item and the slot you can click on to select the device for the rendering: if you have a graphic card that supports this feature, set the GPU instead of the default CPU, and to make this permanent, click on the **Save User Settings** button, or press *Ctrl* + *U*, and close the panel.
2. Go to the **Render** window under the main **Properties** panel and, in the top **Render** subpanel it is now possible to select the **GPU** item as a rendering device, but only if your graphic card supports **CUDA**.

   This will boost your rendering speed several times, making it possible to significantly increase the rendering samples in the **Sampling** subpanel to reduce or even avoid the noise and keep good rendering times. Using the **GPU**, it's also possible to increase the size of the **X** and **Y Tiles** in the **Performance** subpanel (two or three times the default size is **64**).

   But, not everyone has a **GPU** graphic card yet, and there are also cases where you have to mandatorily use the **CPU** instead (for example, for very big scenes with a lot of geometry that doesn't fit inside the somewhat limited **RAM** of a graphic card).

   In such cases, there are things you can do to try to obtain faster and better quality render results:
3. Select the **Light_key** lamp and in the **Node Editor** window add a **Light_Falloff** node (*Shift* + *A* | **Color** | **Light Falloff**); connect its **Linear** output to the **Strength** input socket of the **Emission** node, set the **Strength** to **1000.000** and the **Smooth** value to **1.000**.
4. Click on the color box of the **Emission** node and change the color to **R 0.800**, **G 0.800**, **B 0.650**.
5. Go to the **Render** window and in the **Sampling** subpanel set the **Samples** to **200** or a higher value both for **Render** and **Preview**, to reduce the noise as much as possible.
6. Set the **Clamp Direct** and the **Clamp Indirect** values to **3.00** or **4.00** or even higher (they are set to **0.00** by default); when possible, it is better to leave the **Clamp Direct** item at **0.00** or use values higher then **2.00**, otherwise you could get weird effects in the texturing.
7. In the **Light Path** subpanel, disable both the **Reflective Caustics** and the **Refractive Caustics** items (unless you really need to have caustics in your render) and set the **Filter Glossy** value to **4.00** – **6.00**.
8. In some cases, it won't be possible to totally eliminate the noise or the fireflies; but, because we are going to render an animation, that is several frames in sequence, at least we can make the noise less noticeable and more *natural* looking: go back to the **Sampling** subpanel and click on the **Seed** slot to type `#frame`. This creates an automated driver that takes the seed value from the current frame number, in order to have different noise at every frame.

The Light Falloff node for the Lamps, the Seed driver for the noise, the Caustics items and the Filter Glossy value

See also

- http://www.blender.org/manual/render/cycles/reducing_noise.html
- http://www.blender.org/manual/render/cycles/settings/index.html
- http://www.blender.org/manual/render/cycles/gpu_rendering.html
- http://www.blender.org/manual/render/workflows/animations.html
- http://www.blender.org/manual/render/blender_render/performance.html
- A list of supported GPU graphic cards for Cycles can be found at https://developer.nvidia.com/cuda-gpus

Compositing the render layers

We have seen that the rendering in the **Cycles Render** engine is quite slow but of very good quality, while the scanline **Blender Render** engine is faster but with a lower quality.

Thanks to the Blender integrated **Compositor** and to the **render layers**, it is possible to mix separated and different passes of both the renderers, obtaining a compromise between quality and speed, for example, by over-imposing the **glossy pass** obtained in **Cycles** on the **diffuse pass** obtained in **BI**, and so on.

Getting ready

To mix different passes obtained from the two render engines, we must first apply some modification to the materials of the library file:

1. Start Blender and load the `Gidiosaurus_shaders_Blender_Internal.blend` file, which is the file we used as library for the proxified character and the walkcycle action.
2. In the **Outliner** select the **Gidiosaurus_lowres** item, be sure to be in the **BI** scene and go to the **Material** window; select the first material slot, that is the `Material_skin_U0V0` slot, and in the **Node Editor** window select the **SPEC** material node:

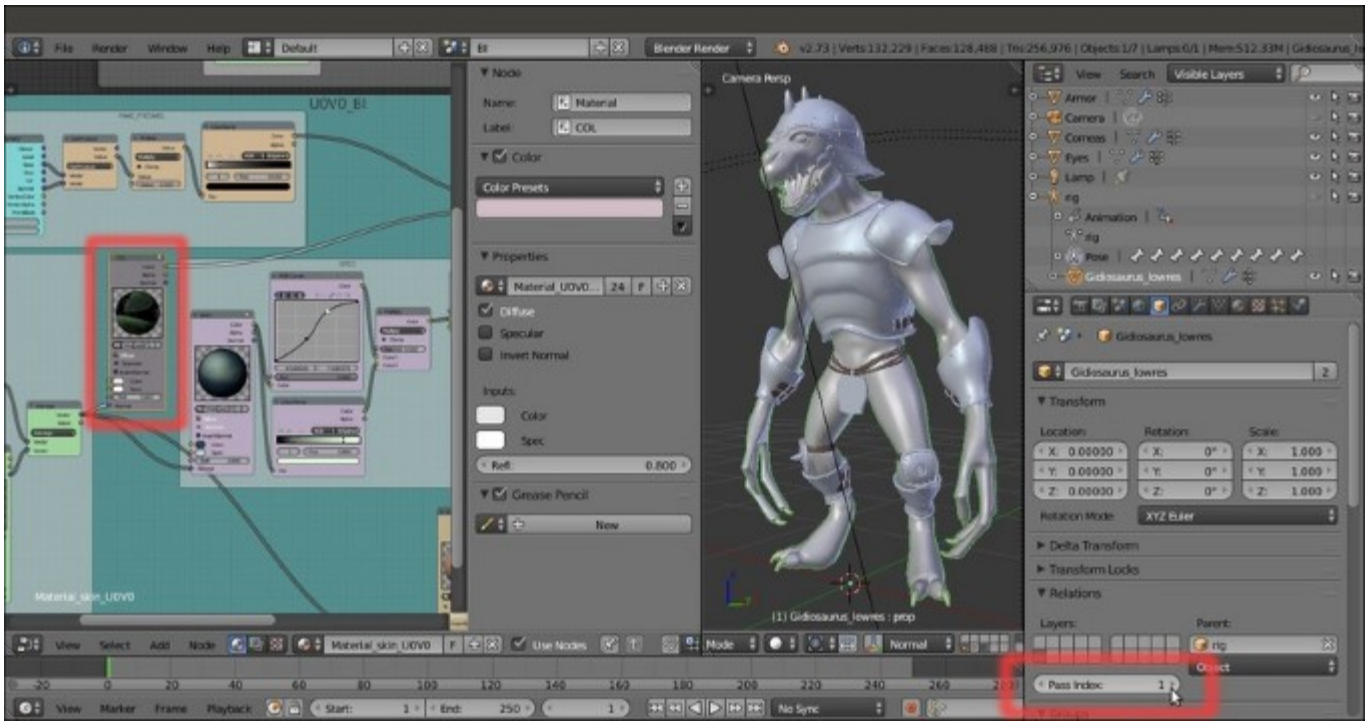
The SPEC node material for Blender Internal

3. Go to the **Texture** window and click on the *Enable/Disable each texture* checkbox at the right side of the **env_refl_skin** texture slot to disable it:

The disabled "env_refl_skin" texture slot

4. Go back to the **Node Editor** window and select the **COL** node (this seems to be important, because of a bug in the **Blender Internal** working method with linked material nodes and the **despgraph** that doesn't update the materials and, therefore, doesn't render the diffuse color correctly).
5. Select the second `Material_skin_U1V0` slot, go to the **Node Editor**, select the **SPEC** material node, disable the `env_refl_skin` texture slot, select the **COL** material node, then go to the third `Material_skin_U2V0` slot, and so on: repeat for all the materials of the **Gidiosaurus** and of the **Armor** objects (for the **Armor** disable the `env_refl_armor` slots on both the **SPEC1** and **SPEC2** material nodes; **Eyes** and **Corneas** have the real ray-tracing mirror enabled and don't need any textures disabled).

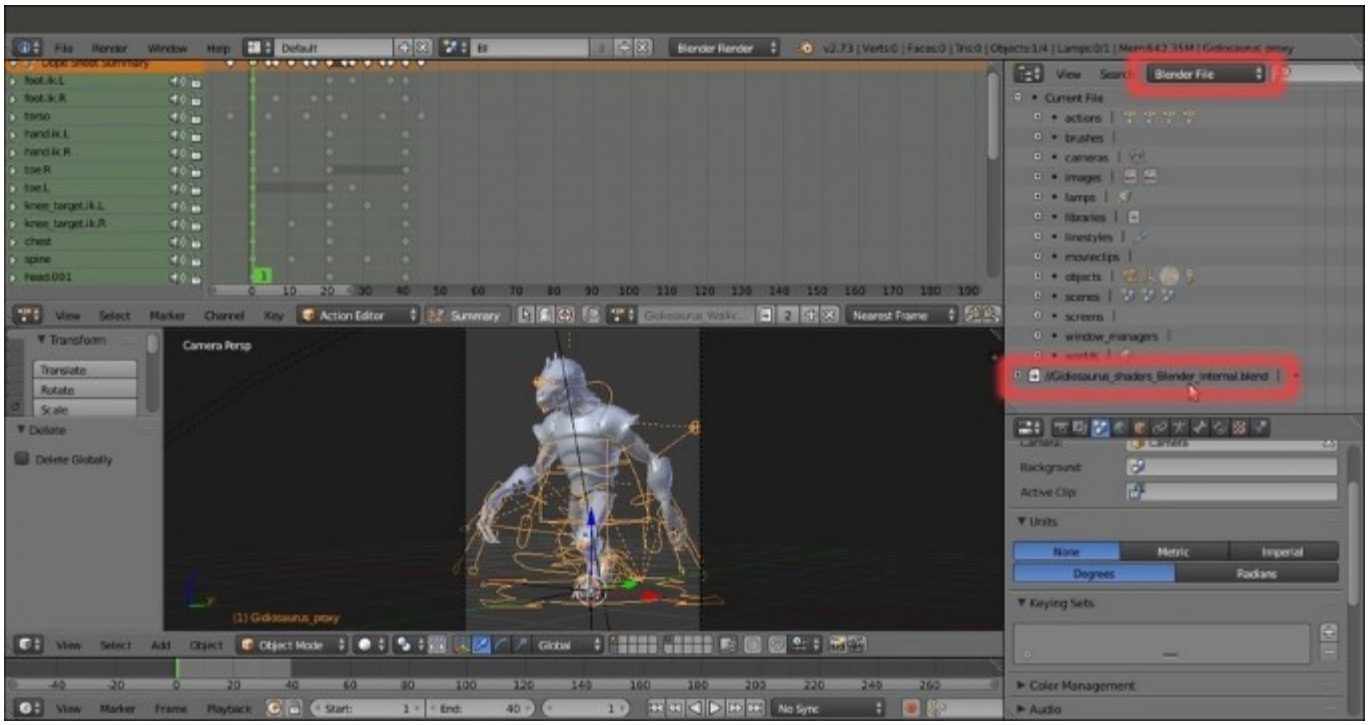
   Remember to leave the **COL** material nodes as the selected ones in the **Node Editor** window, or the diffuse color won't show in the rendering.
6. Go to the **Object** window and, in the **Relations** subpanel, assign a different **Pass Index** to the objects: assign **1** to the **Gidiosaurus_lowres** object, **2** to the **Armor** object, **3** to the **Eyes**, and **4** to the **Corneas**.



The Pass Index slot

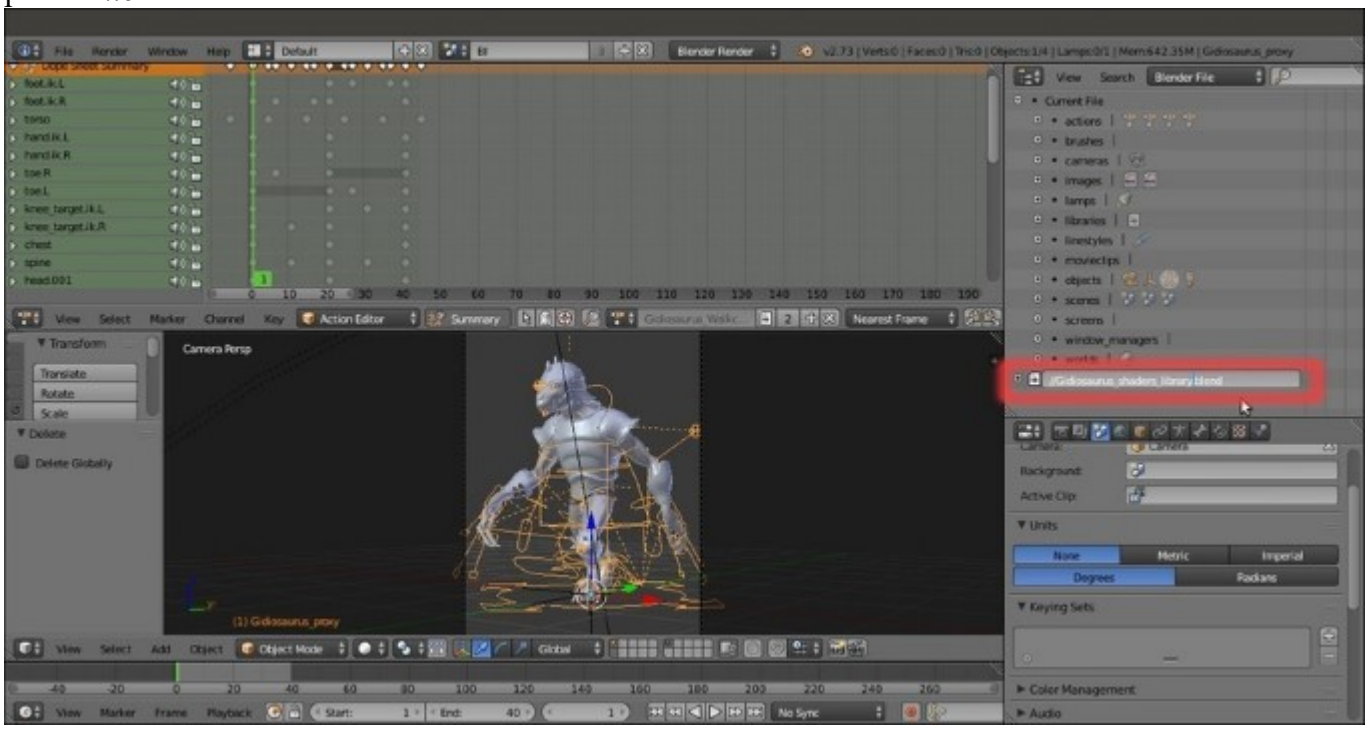
7. Save the file as `Gidiosaurus_shaders_library.blend`.

   Now, because up to this point we have used the `Gidiosaurus_shaders_Blender_Internal.blend` file as library source, we must now substitute the file to be rendered the path to the new library source:
8. Open the `Gidiosaurus_render.blend` file and go to the **Outliner** window: click on the *Type of information to display* button at the top to switch from **Visible Layers** to **Blender File**; expand the panel to find the **//Gidiosaurus_shaders_Blender_Internal.blend** item at the bottom:



The library path in the Outliner

9. Double left-click on the item and rename it as **//Gidiosaurus_shaders_library.blend**, then press *Enter* to confirm and save the file:



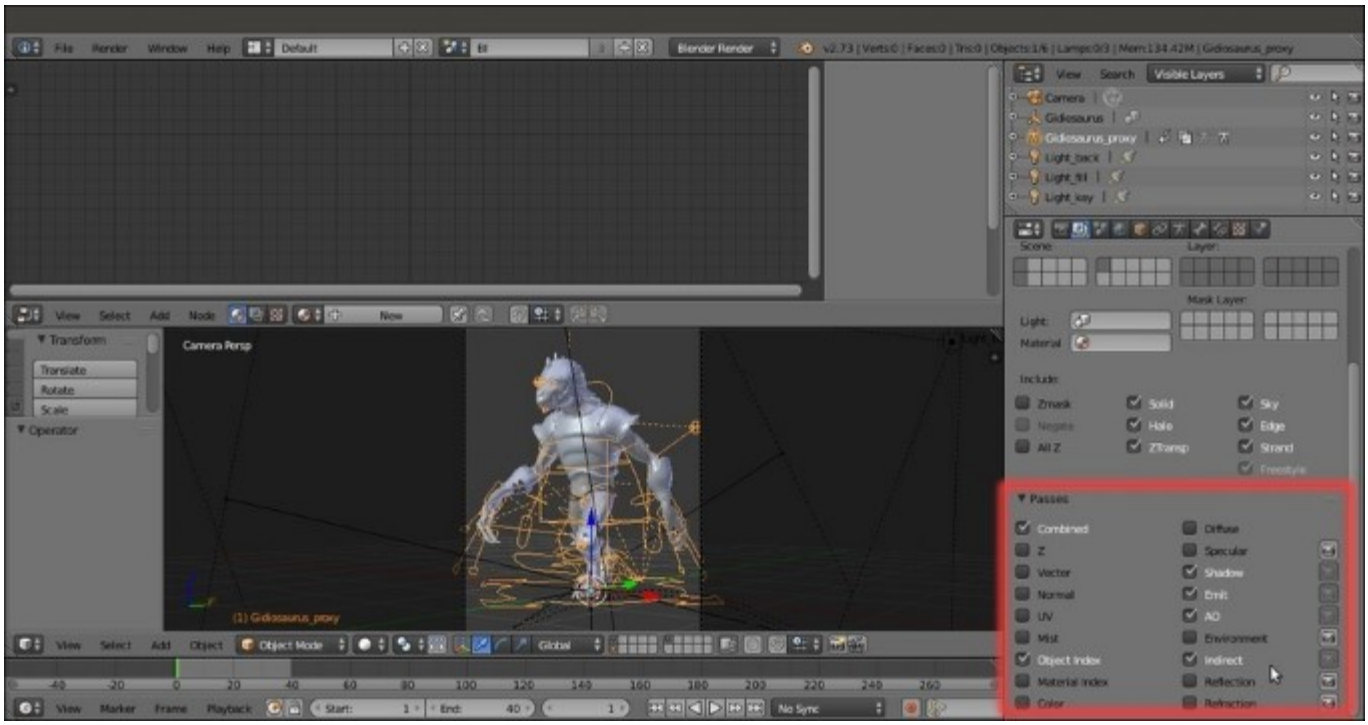
The modified library path

10. Press *Ctrl* + *O* | *Enter* to re-load the file: now all the assets should be linked from the new library file.
11. Save the file as `Gidiosaurus_compositing.blend`.

How to do it…

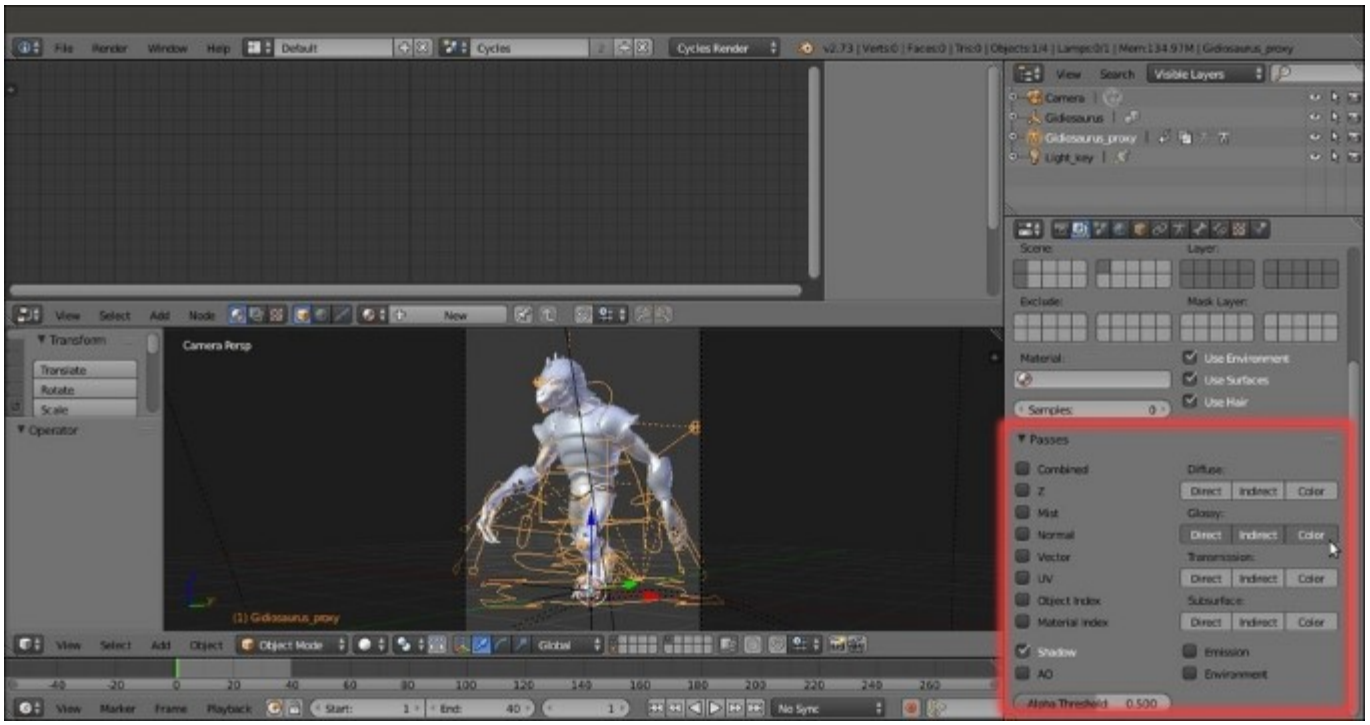
At this point we must prepare the passes for the two scenes that will be used later, as elements to be mixed through a third new *compositing* scene:

1. Click on the **+** icon button to the right side of the *Scene datablock* button in the main top header and, in the **New Scene** pop-up menu, select the **New** item: this creates a new **empty** scene; rename it **comp**.
2. Click on the *Screen datablock* button to the left and switch to the **Compositing** screen, then click again on the *Scene datablock* button and re-select the newly created **comp** scene.
3. Click again on the *Screen datablock* button and go back to the **Default** screen; go to the *Scene datablock* button and select either the **BI** or the **Cycles** scene.
4. From now on, it's enough to select the **Compositing** layout in the *Screen datablock* button to switch automatically to the **comp** scene, and the **Default** layout to go to the **BI** or the **Cycles** scene (depending on the last one selected).
5. Go to the **Default** screen and, if not already loaded, load the **BI** scene; in the main **Properties** panel, go to the **Render Layers** window (the second icon button from the left in the *Type of active data to display and edit* windows row).
6. Double click on the **RenderLayer** name in the first slot at the top of the subpanel to rename it as **BI**. Go down to the **Passes** subpanel, disable the **Z** pass item and enable **Object Index**; then go to the second column and enable the **Shadow** pass but then also click on the *Exclude shadow pass from combined* button to its extreme right side: do the same also for the **Emit**, the **AO** and the **Indirect** passes.



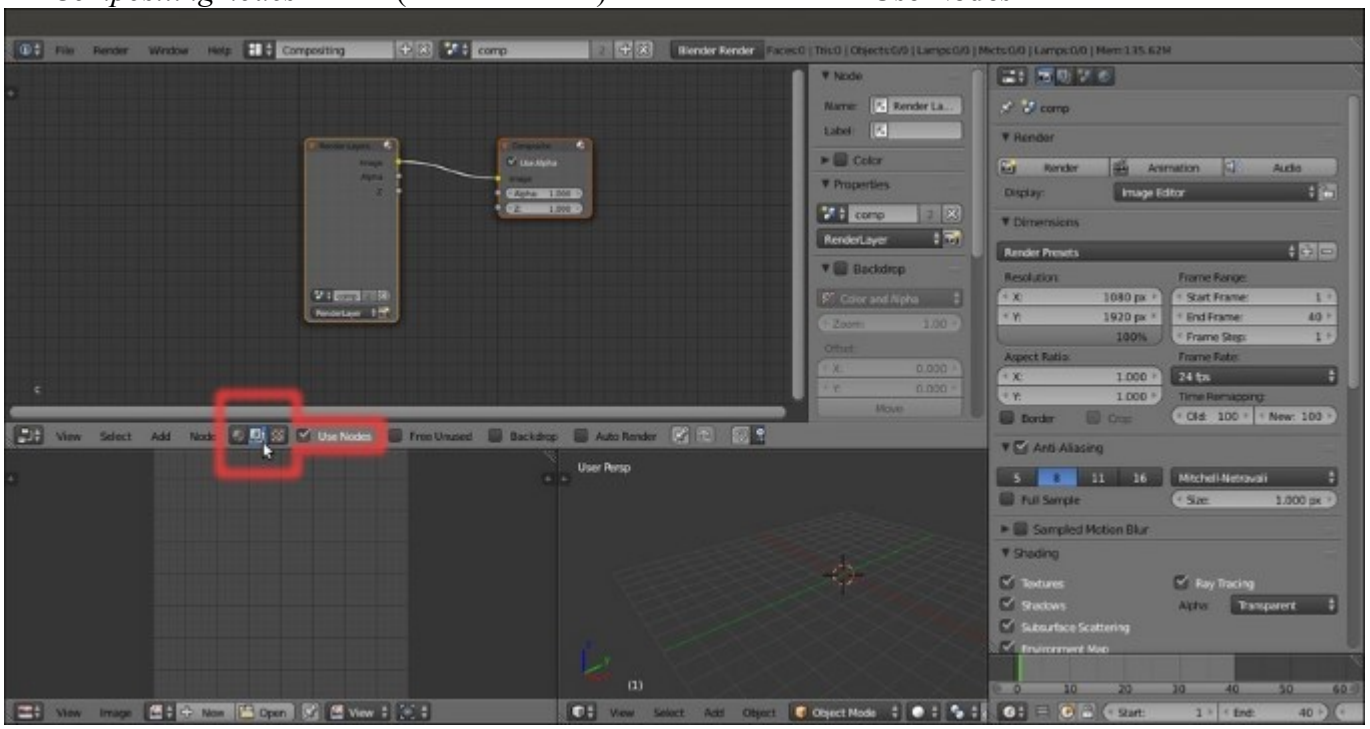
The Passes setting for the Blender Render scene

7. Click again on the *Scene datablock* button in the top main header and switch to the **Cycles** scene.
8. Double click on the **RenderLayer** name in the first slot at the top of the subpanel to rename it as **Cycles**, then disable the **Combined** and the **Z** passes, leave the already enabled **Shadow** pass as it is and enable also the **Glossy Direct**, **Indirect** and **Color** passes:



The Passes setting for the Cycles Render scene

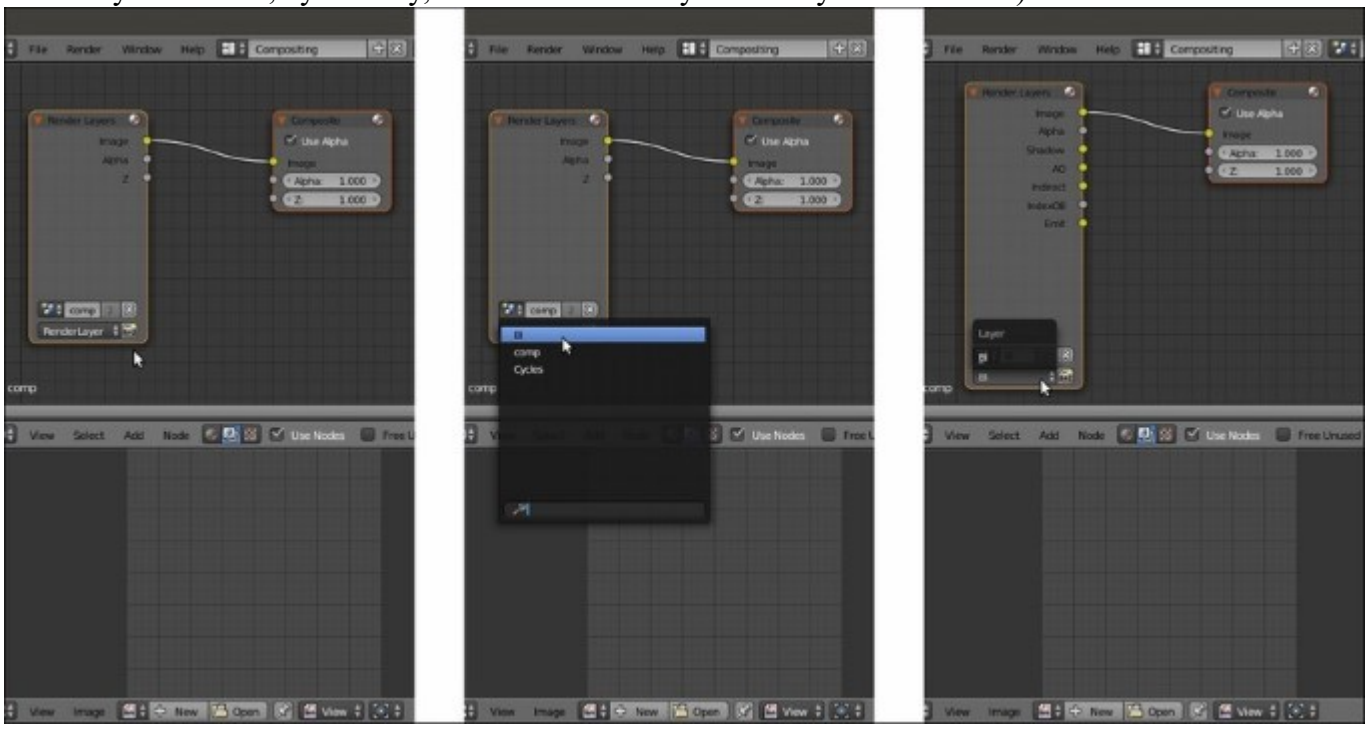
9. Now click on the *Screen datablock* button and switch to the **Compositing** screen.
10. In the **Node Editor** window toolbar, go to the *Node tree type to display and edit* row, click on the *Compositing nodes* button (the middle one) and then enable the **Use Nodes** checkbox:



The Compositing nodes button

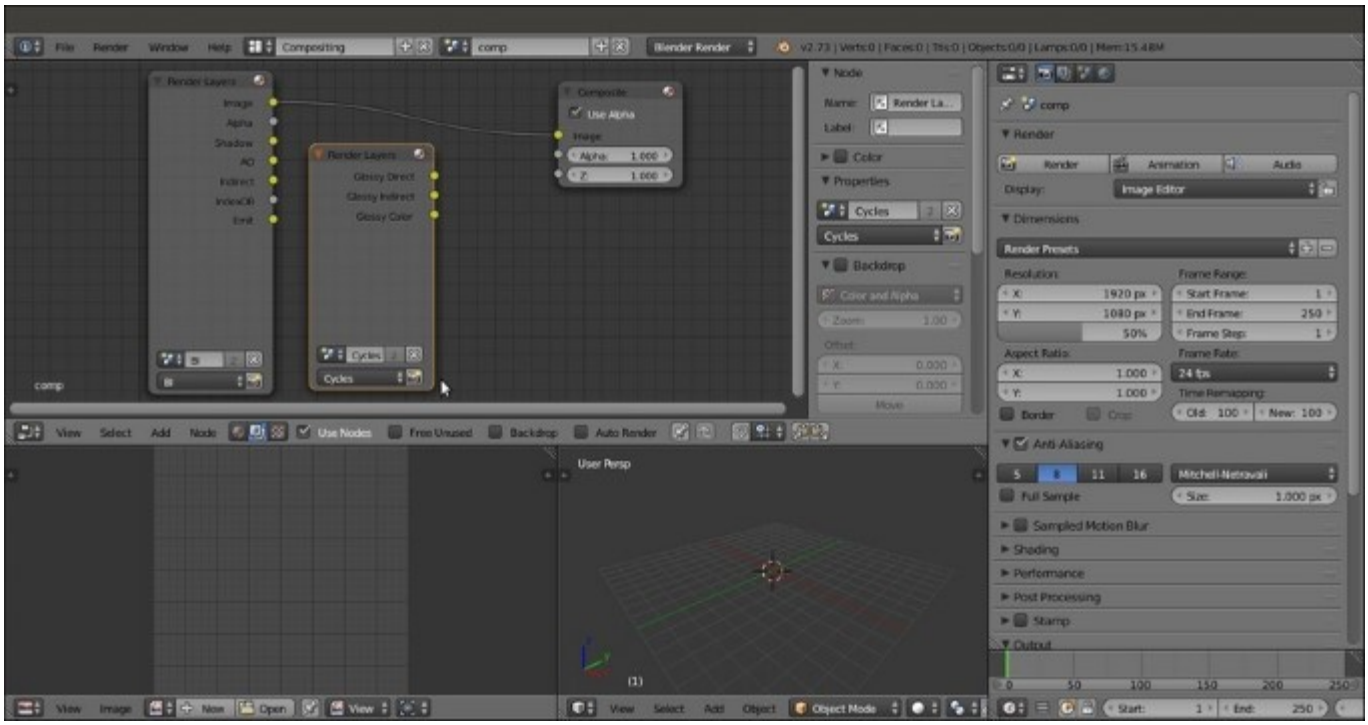
Two compositing nodes are automatically added in the **Node Editor** window: a **RenderLayers** node connected to a **Composite** node.

11. Click on the double arrows to the side of the *Scene datablock* on the **RenderLayers** node to switch from the **comp** scene to the **BI** scene, and, if necessary, in the bottom **Layer** button, select the name of the respective render layer (that we labeled as **BI** again; it shouldn't be necessary to select it, by the way, because it's the only render layer in the scene).



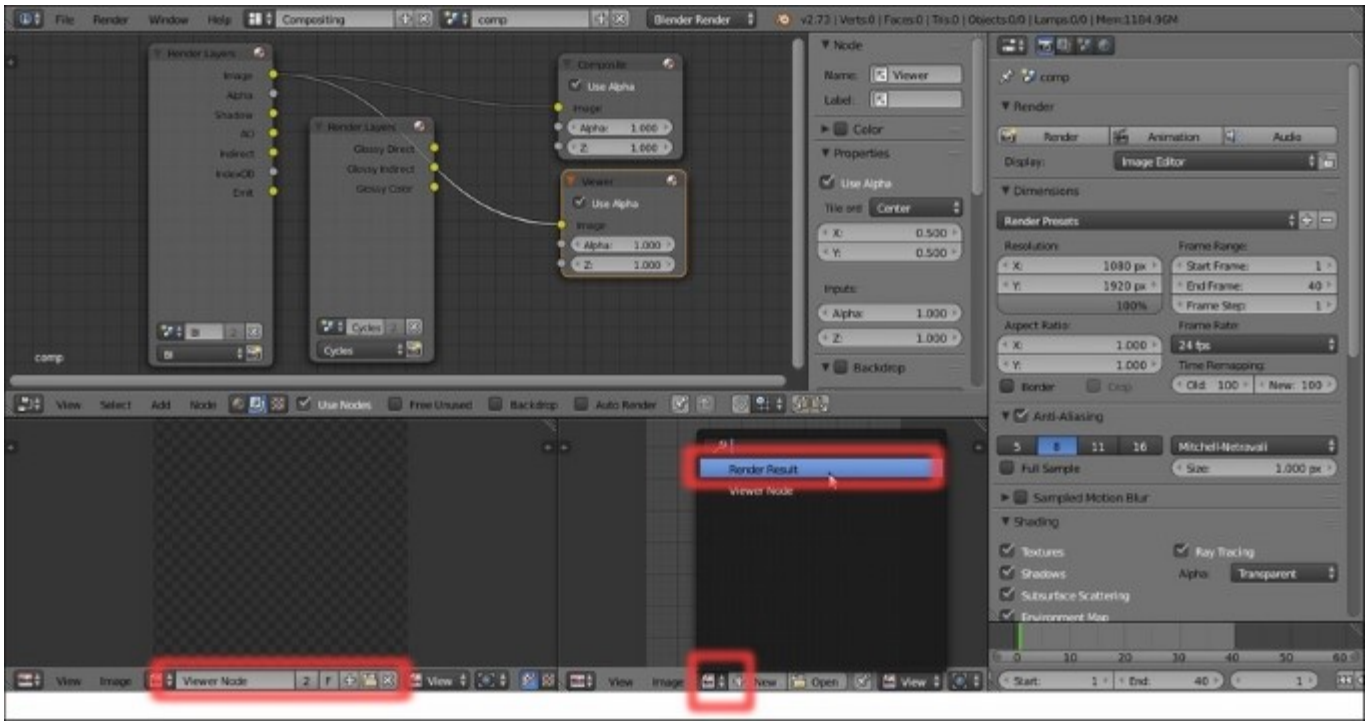
The render layer selector in the RenderLayers node and the output sockets to the Composite node

12. Press *Shift* + *D* to duplicate the **RenderLayers** node and repeat the procedure in the duplicated one, this time selecting the **Cycles** scene datablock and render layer:



The duplicated RenderLayers node

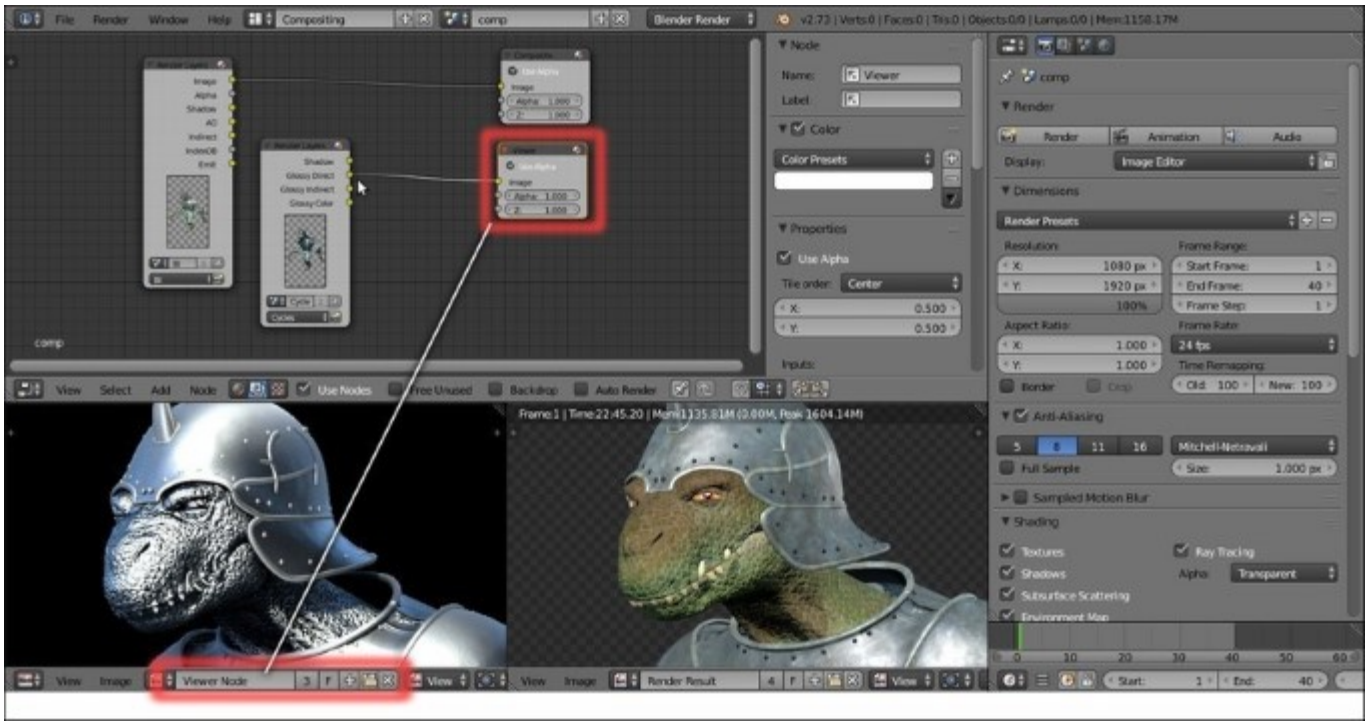
13. Put the mouse pointer inside the **Node Editor** window and press *Shift + A* to add a **Viewer** node (*Shift + A* | **Output** | **Viewer**); connect the **Image** output of the **BI RenderLayers** node also to the **Image** input socket of the **Viewer** node.
14. Move down and switch the **3D View** window with another **UV/Image Editor** window.
15. In the left image editor window, click on the double arrows to the left of the image datablock (*Browse Image to be linked*) and from the pop-up menu select the **Viewer Node** item.
16. In the right image editor window, instead, select the **Render Result** item.



The Viewer node and the Viewer Node and Render Result windows

17. At this point, press *F12* or click on the **Render** button inside the main **Properties** panel to start the rendering.

    When the rendering is done, only the **BI** render is visible in the two bottom editor windows, because at the moment it's the only **RenderLayers** node connected to the **Viewer** and to the **Composite** nodes.
18. Connect the **Glossy Direct** output of the **Cycles RenderLayers** node to the **Image** input socket of the **Viewer** node to see the single *specularity* pass in the left **UV/Image Editor**.



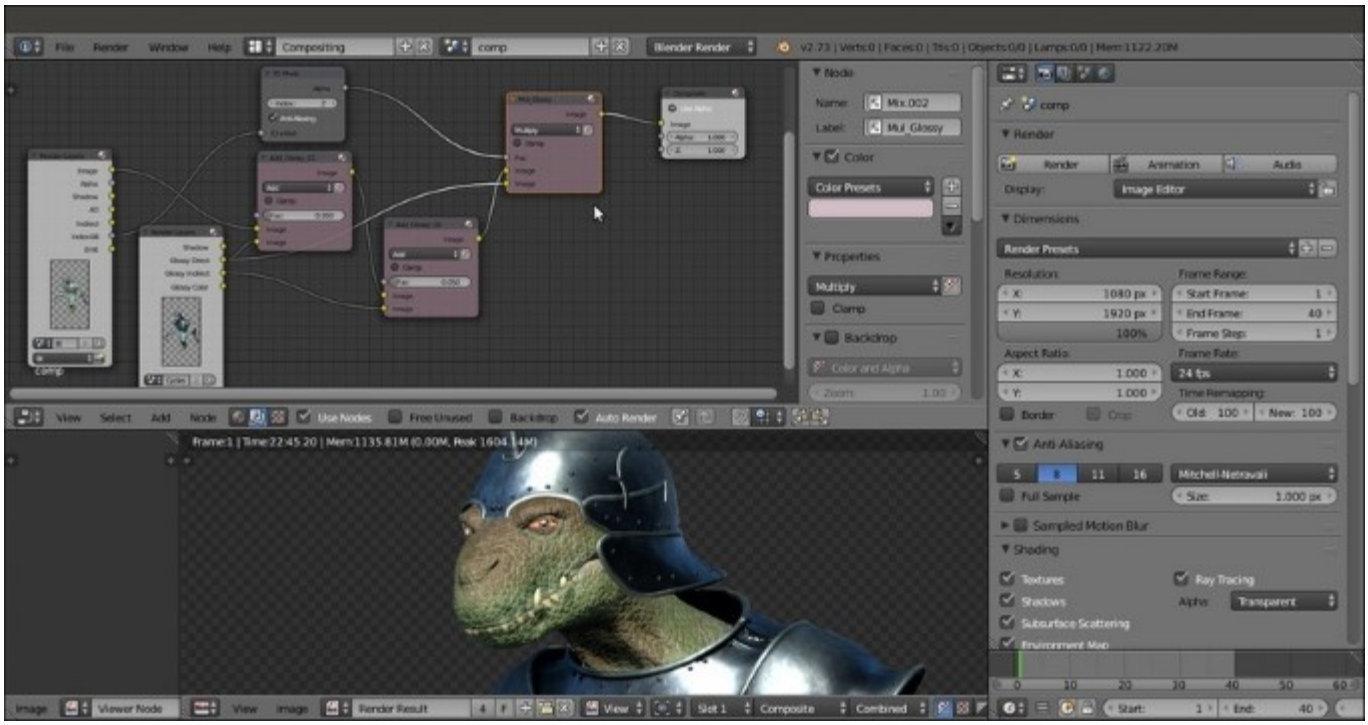
The Glossy Direct pass visualized in the Viewer Node window

In fact, we could also add more than one **Viewer** node to the **Node Editor** window and use them to visualize the different passes of **RenderLayers**, by connecting each pass output to each **Viewer** node; the last selected **Viewer** node will be the one visualized in the **Viewer Node** bottom window.

19. Enable the **Auto Render** item at the extreme right side on the **Node Editor** window toolbar, add a **Mix** node (*Shift* + *A* | **Color** | **Mix**) and paste it between the **Image** output of the **BI RenderLayers** node and the **Image** input socket of the **Composite** node; change the **Blend Type** to **Add** and then connect the **Glossy Direct** output of the **Cycles RenderLayers** node to its second **Image** input socket. Set the **Fac** value to **0.050** and label it as **Add_GLOSSY_01**.
20. If you want, also connect the output of the **Add_GLOSSY_01** node to the **Viewer** node.

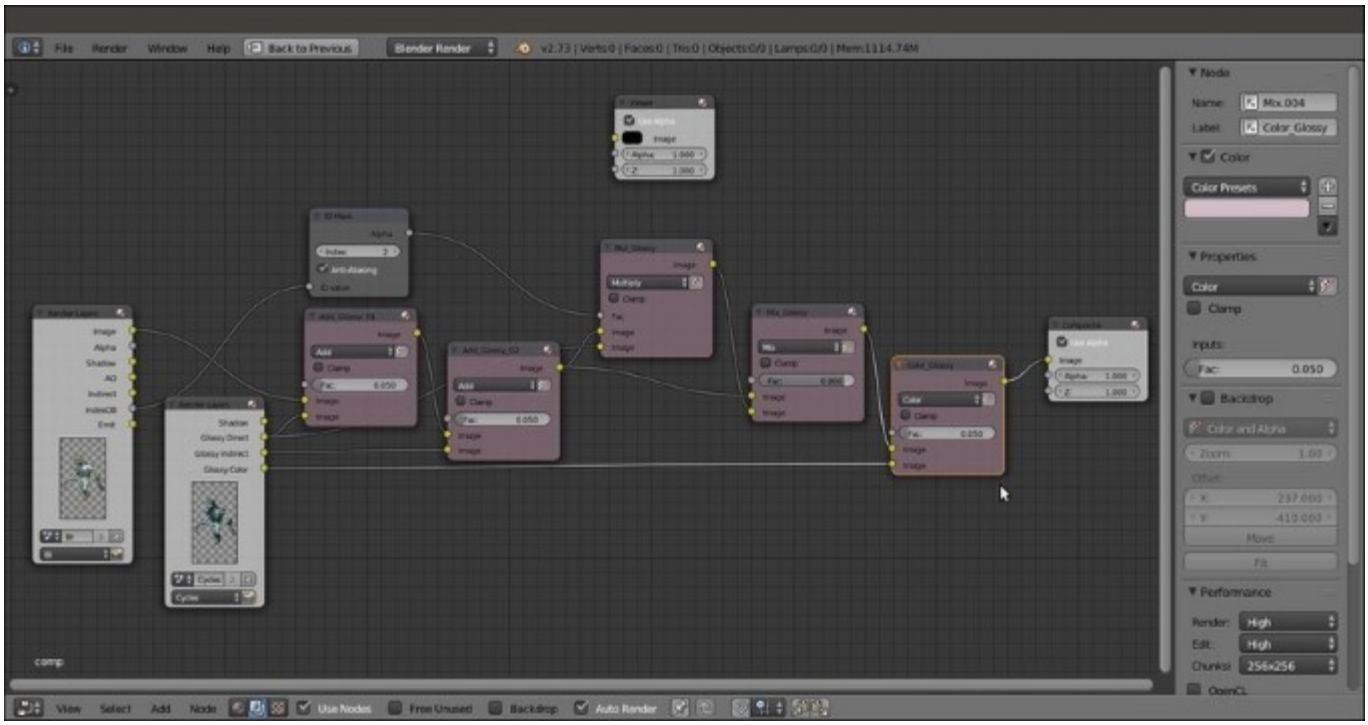
The Glossy pass, rendered in Cycles, added to the Image pass rendered in Blender Internal; note that now the armor looks too lightened

21. Press *Shift* + *D* to duplicate the **Add_GLOSSY_01** node, label it as **Add_GLOSSY_02** and paste it between the **Add_GLOSSY_01** and the **Composite** nodes; connect the **Glossy_Indirect** output of the **Cycles RenderLayers** node to the second **Image** input socket of the **Add_GLOSSY_02** node.
22. Press *Shift* + *D* to duplicate the **Add_GLOSSY_01** node again, label it as **Mul_GLOSSY**, change the **Blend Type** to **Multiply** and set the **Fac** value back to **1.000**; connect the output of the **Add_GLOSSY_02** node to its first **Image** input socket and the **Glossy_Direct** output of the **Cycles RenderLayers** node to its second **Image** input socket. Connect the output of the **Mul_GLOSSY** node to the **Composite** node.
23. Add an **ID Mask** node (*Shift* + *A* | **Converter** | **ID Mask**), set the **Index** value to **2** and connect the **IndexOB** output of the **BI RenderLayers** node to the **ID value** input socket, then connect the **Alpha** output to the **Fac** input socket of the **Mul_GLOSSY** node; enable the **Anti-Aliasing** item.
24. For the moment, disconnect the **Viewer** node.



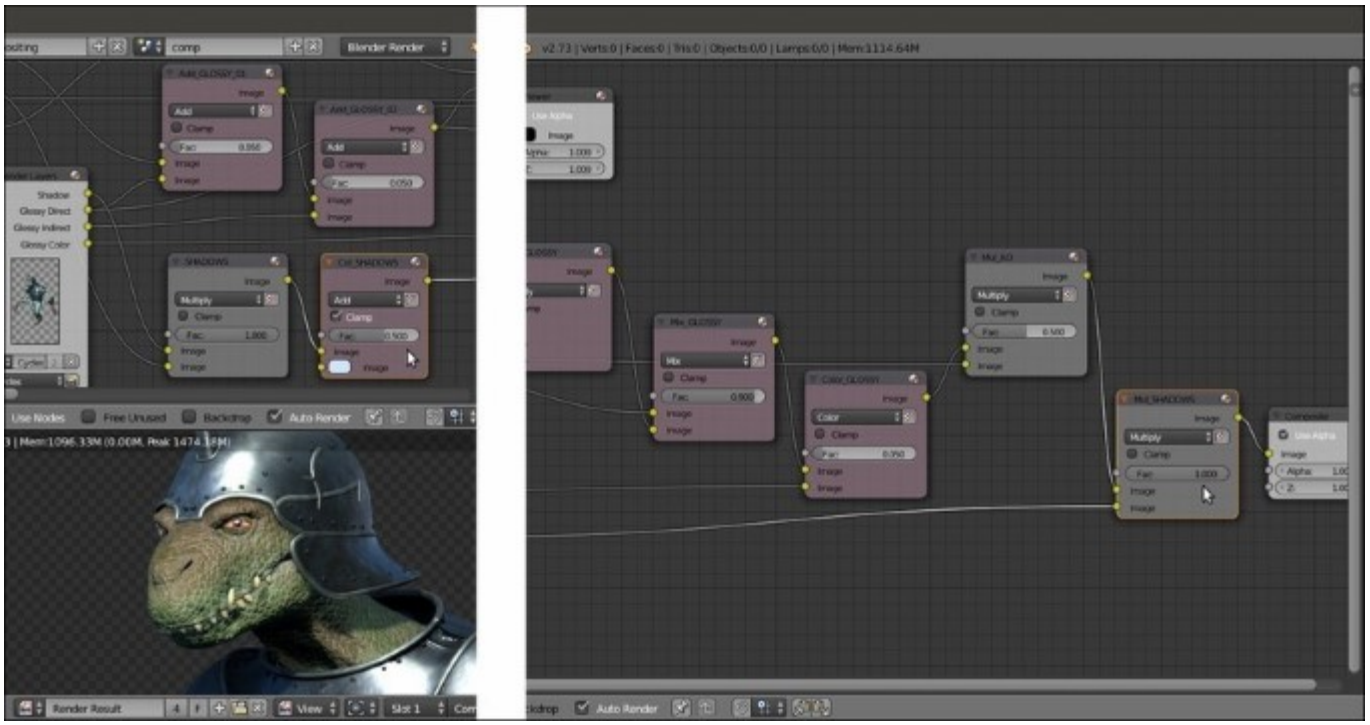
The IndexOB output used as a mask for the addition of the Glossy pass only on the character's skin

25. Press *Shift* + *D* to duplicate a new **Mix** node, label it as **Mix_GLOSSY** and set the **Blend Type** to **Mix** and the **Fac** value to **0.900**; connect the output of the **Add_GLOSSY_02** node to the first **Image** input socket and the output of the **Mul_GLOSSY** node to the second **Image** input socket.
26. Press *Shift* + *D* to duplicate another **Mix** node, label it as **Color_GLOSSY** and set the **Blend Type** to **Color** and the **Fac** to **0.050**; connect the output of the **Mix_GLOSSY** node to the first **Image** input socket and the **Glossy_color** output of the **Cycles RenderLayers** node to the second **Image** input socket:



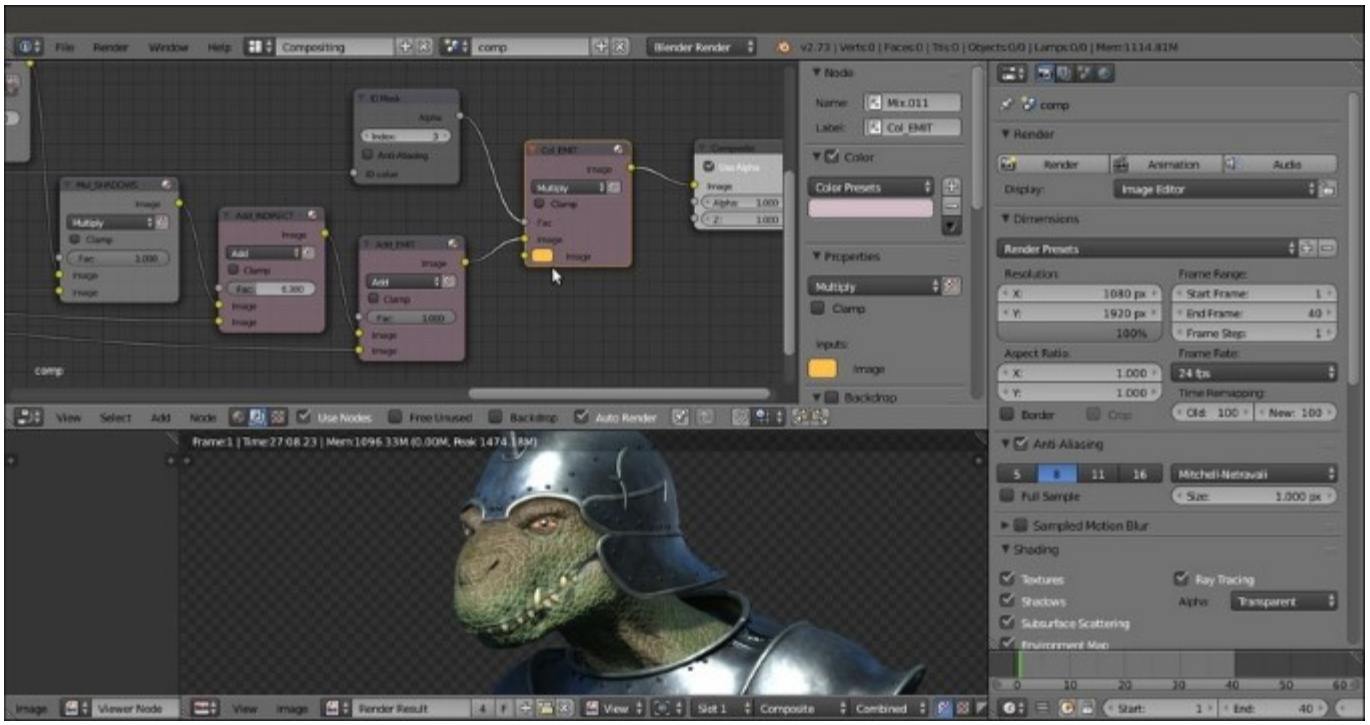
Adding more compositing nodes to re-build the separately rendered Glossy passes

27. Add a new **Mix** node (*Shift* + *A* | **Color** | **Mix**) right after the **Color_GLOSSY** one, label it as **Mul_AO**, set the **Blend Type** to **Multiply** and connect the **AO** output of the **BI RenderLayers** node to the second **Image** input socket; set the **Fac** to **0.500**.
28. Add a new **Mix** node (*Shift* + *A* | **Color** | **Mix**), label it as **SHADOWS**, change the **Blend Type** to **Multiply** and set the **Fac** value to **1.000**; connect the **Shadow** output of the **BI RenderLayers** node to its first **Image** input socket and the **Shadow** output of the **Cycles RenderLayers** node to the second **Image** input socket.
29. Press *Shift* + *D* to duplicate the **Mul_AO** node and label it as **Mul_SHADOWS**; paste it right behind the **Mul_AO** node and set the **Fac** value slider to **1.000**.
30. Connect the output of the **SHADOWS** node to the second **Image** input socket of the **Mul_SHADOWS** node.
31. Add a new **Mix** node (*Shift* + *A* | **Color** | **Mix**), label it as **Color_SHADOWS** and paste it right behind the **SHADOWS** node; set the **Blend Type** to **Add**, the **Fac** to **0.500** and enable the **Clamp** item: set the color of the second **Image** socket to **R 0.640**, **G 0.780**, **B 1.000**.



Multiplying and coloring the shadow pass

32. Add a new **Mix** node (*Shift* + *A* | **Color** | **Mix**) right behind the **Mul_SHADOWS** one, label it as **Add_INDIRECT**; set the **Blend Type** to **Add**, the **Fac** to **0.300** and connect the **Indirect** output of the **BI RenderLayers** to the second **Image** input socket.
33. Repeat the previous step but label the node as **Add EMIT**, set the **Fac** value to **1.000** and connect the **Emit** output of the **BI RenderLayers** node to the second **Image** input socket.
34. Add one more **Mix** node (*Shift* + *A* | **Color** | **Mix**), label it as **Col_EMIT**, set the **Blend Type** to **Multiply** and paste it behind the **Add_EMIT** node; set the color of the second **Image** socket to **R 1.000**, **G 0.542**, **B 0.073**.
35. Add a new **ID Mask** node (*Shift* + *A* | **Converter** | **ID Mask**), connect the **IndexOB** output of the **BI RenderLayers** node to the **ID value** socket, set the **Index** value to **3** and connect its **Alpha** output to the **Fac** input socket of the **Col_EMIT** node.



Coloring and adding the eyes

36. Connect the output of the **Col_EMIT** also to the **Image** input socket of the **Viewer** node:

The completed compositing network

37. Save the file.

How it works…

In the *Getting ready* section:

- From step 2 to step 5 we disabled the **env_refl_skin** and the **env_refl_armor** texture slots in all the material nodes of the **BI** shaders; in fact, because we are going to add the **Cycles** reflection on the **BI** diffuse, we don't need to fake the environment reflection on the character and on the armor surfaces anymore.
- In step 6 we assigned a different **Index Pass** number to each one of the objects; this is useful to later *separate* the objects in the **Compositor** for particular effects (in our case we only needed the armor **Index Pass** number, but it's a good habit to give index passes to all the objects for any eventuality).
- At step 7 we saved the file with a new name, and at steps 8 and 9 we changed, in the file to be rendered, the path to the new library file.

In the *How to do it…* section:

- From step 1 to step 3 we added a new empty scene, **comp**, to the blend file, linked to the **Compositing** screen layout and to be used for the compositing.
- In fact, in the **comp** scene, all the **compositing** nodes are connected together so as to recreate the best possible rendering look of the **Gidiosaurus**, using the different passes from the different **BI** and **Cycles** scenes through the **render layers**.
- From step 4 to step 7 we enabled the required passes in the **Render layers** windows of both the **BI** and **Cycles** scenes; note that basically we set an almost complete *only-diffuse* render in **Blender Internal** (besides the **Indirect**, **Ambient Occlusion** and **Emit** passes, subtracted from the total render and separately outputted), while we set the **Glossy** passes in the **Cycles** engine.
- From step 8 to step 15 we set up the **RenderLayers** nodes, the **Viewer** and the **Composite** nodes.

  The **RenderLayers** node outputs the rendering of a particular scene also delivering the enabled passes for that scene, through the **Render Layer** setup.

  The **Compositing** node is the mandatory final output node and must always be connected as the last step of the compositing chain.

  The **Viewer** node, instead, is optional but always used anyway to visualize the different steps of the compositing itself.
- At step 16 we started the rendering; Blender starts to render the **BI** and the **Cycles** scenes using the settings setup for each scene (and not the settings of the **comp** scene, which are true only for the compositing).
- From step 18 to step 34 we mixed the different passes together. Basically, we used the same approach that we have seen in the creation of the shaders, that is, by decomposing the final result into the different components; then, we obtained the diffuse from the **Blender Internal** engine because it's quite fast for that, and the glossy component from the **Cycles** render engine for the same reasons, and then we re-mixed them. The **Subsurface Scattering** pass is actually

rendered together at the diffuse component in **BI** and is delivered through the **Combined** (**Image**) pass.

- The **Mix_GLOSSY** node added at step 24 is to tweak the strength of the multiplied **Glossy Direct** pass; we couldn't use the **Fac** value, in this case, because it was already used by the **ID Mask** output to *isolate*, thanks to the **Object Index**, the **Armor** from the rest of the render.
- With the **Col_SHADOWS** node at step 30 we obtained two goals: first, we set the dark intensity of the shadows to **0.500** and, second, we gave them a bluish coloration.
- The emission pass for the eyes has been added to the rendering through the same technique used to multiply the glossy only on the **Armor**, that is, by an **ID Mask** node and the **Object Index**.

See also

- http://www.blender.org/manual/render/blender_render/layers.html
- http://www.blender.org/manual/render/blender_render/passes.html
- http://www.blender.org/manual/render/post_process/layers.html
- http://www.blender.org/manual/composite_nodes/index.html