POV-Ray Tutorial (III)

By Eric Albrecht

After reading the first two tutorials you should be to use LDView to convert an LDraw file to a POV-Ray file and do a simple render. You should also have a basic understanding of the contents of that LDraw file. This time, we will start trying to improve the quality and realism of the renders.
After the geometry itself, the most important part of making a realistic render is the lighting. After all, without lighting even the most magnificent geometry would only appear as a screen of black pixels! Everything that our eyes see appears the way it does because of light. Seeing and interpreting light is so innate that we can easily overlook just how complex the sources and results of light are. A single photon that hits your retina may have travelled millions of kilometers and been reflected, refracted, and filtered thousands of times. Our brains are extremely efficient at interpreting light, and are therefore also extremely adept at noticing when something looks wrong. Figuring out why it looks wrong is the hard part. The study of light is obviously far too complex a topic for this article, but there are some simple things we can do to make the default render look better.
For purposes of illustration I'm going to use a more complex model than the single red brick from previous lessons. I've chosen the 8110 Unimog because it offers a lot of shapes and colors for illustration. First let's render the model just as it is exported from LDView. This is what we get:

Notice how shadowed areas are totally black. For instance, the inside of the cabin has areas that are far too dark. Now look at the shadows on the ground. The shadows are sharp and distracting. The places where the shadows overlap look odd. So even though the model has reflection and shadow which makes it look better than the simple shading of LDView, it still doesn't look quite right. To understand why, let's look inside the POV-Ray file at the light definition.

```
// Lights
light_source {      // Latitude,Longitude: 45,0,LDXRadius*2
        <0*LDXRadius,-1.414214*LDXRadius,-1.414214*LDXRadius> + LDXCenter
        color rgb <1,1,1>}
light_source {      // Latitude,Longitude: 30,120,LDXRadius*2
        <1.5*LDXRadius,-1*LDXRadius,0.866026*LDXRadius> + LDXCenter
        color rgb <1,1,1>}
light_source {      // Latitude,Longitude: 60,-120,LDXRadius*2
        <-0.866025*LDXRadius,-1.732051*LDXRadius,0.5*LDXRadius> + LDXCenter
        color rgb <1,1,1>}
```

What we have here are 3 white point light sources distributed around the model. Point lights are the simplest types of light sources. They are the easiest to define and the quickest to render. The fact that the light comes from a tiny spot is what makes the shadows so hard. However, in real life there are no such things as point lights. Every light source, even a small one, is distributed over some finite area. Some lights, like the sun, are distributed over a very wide area. White (defined as RGB 1,1,1) is the simplest color. But in real life almost no light sources really emit every part of the visible spectrum, so white lights don't look right. Fortunately, both of these things can be easily changed. Let's replace the light definition with this:

```
// Lights
//front side light
light_source {
        <0*LDXRadius,-1.414214*LDXRadius,-1.414214*LDXRadius> + LDXCenter
        color rgb <1,1,0.9>*0.7
        area_light 100,100,3,3
        orient
        circular
        adaptive 1}

//back light
light_source {
        <1.5*LDXRadius,-1*LDXRadius,0.866026*LDXRadius> + LDXCenter
        color rgb <1,1,0.9>*0.7
        area_light 100,100,3,3
        orient
        circular
        adaptive 1}
//key light
light_source {
        <-0.866025*LDXRadius,-1.732051*LDXRadius,0.5*LDXRadius> + LDXCenter
        color rgb <1,1,0.9>*0.9
        area_light 100,100,3,3
        orient
        circular
        adaptive 1}
```

The most obvious thing I have done is to replace the point lights with area lights. The lights I have defined are circular with a diameter of 100 units (5 studs). The way POV-Ray creates area lights is by using a grid of points lights. In this case I have defined a 3x3 grid which means that each area light is really 9 point lights arranged in a circle. The next thing I've done is to reduce the blue in the light color by 10% (RGB is now 1,1,0.9). This makes the light a little more yellow like sunlight or incandescent light. Most professional photographers use 3 lights called a key light (the main front light), a back light, and another off axis light for highlights. I've replicated this here. I've also put a factor after each light that you can use to adjust the relative brightness of each. In this case I made the key light the brightest (0.9), but you might not always wish to do so depending on the effect you are looking for. You can also adjust the overall brightness by scaling all of them up or down the same amount. You can see how much difference this makes in the upper part of the next page.

Compared to the last render, this one has much softer and more realistic shadows. It also has a little better color. But the dark areas are still too dark. With only 3 lights, there is just no way to get any light into those crevices. One really easy trick is to add some reflectivity to the floor. This will allow some light to bounce up into the dark areas. The floor definition is at the very bottom of the POV-Ray file. Let's go to that section and add 40% reflectivity to the floor like this:

```
// Floor
object {plane { LDXFloorAxis, LDXFloorLoc hollow }
        texture {pigment { color rgb <LDXFloorR,LDXFloorG,LDXFloorB> }
                finish { ambient LDXFloorAmb diffuse LDXFloorDif reflection 0.4}}}
```

That last part in italics is what I added. Let’s see what difference it makes.

What a difference! Everything looks much brighter and the underside is less shadowed. More can be seen in the wheel wells, for example. But there is still room for improvement. Some areas are still very dark. In the real world, diffuse light is all around us reflecting from the objects near us and causing changes in the filtered color of light locally. POV-Ray simulates this with a feature called radiosity. To implement it, I'm first going to put a sky sphere around the whole scene. A sky sphere is a very large sphere (radius = 10000) which surrounds the image. Our sky sphere will be the same color as our lights for now, but in future lessons we'll make it much more complex to more realistically replicate real world surroundings. Once we have a sphere of background color, we'll turn on radiosity. Add this to the beginning of your POV-Ray file:

```
#include "rad_def.inc"
      global_settings {
      assumed_gamma 1.4
      max_trace_level 10
      adc_bailout 0.01/2
      radiosity {Rad_Settings(3, off, off)}}
sky_sphere{ pigment {color rgb <1,1,0.9>} }
```

Radiosity has a lot of different parameters to set. Luckily for us, the file we've included (rad_def) comes with POV-Ray and has lots of different predefined settings that do all the work for us. If you right click on "rad_def.inc" you can open it up and see what's inside. The main thing to note is that there 10 different sets of parameters (or quality levels) defined, numbered 1 through 10. We'll generally be using either 1, 3, or 5. Level 1 is really bad quality but renders fast. Use it just to get an idea of how your lighting levels look while you are setting brightness of your other lights. Level 3 is what you'll use most of the time and looks really good. Level 5 is crazy, but awesome. It is described as "for patient quality freaks with fast computers about to leave on vacation" and they aren't kidding. This setting will take forever to render, but it may be worth it for very special images. You can choose which setting to use by changing the first number after Rad_Settings in the text I gave you to insert in your file. I've italicized this setting, and you can see that I've chosen setting 3. Let's see how it looks.

Now that's more like it! The lighting is much better. The dark areas are gone. Everything has at least some light. The transparent parts on the roof look much better. The shadows on the floor are less distinct because the diffuse light has hidden them. One new problem now exists which is that the flat color sky sphere has made the lighting look a bit flat. We'll address that next time with High Dynamic Range lighting (HDR) after which it will look like this:

I have to confess that I cheated a little. When I first added radiosity, the result was too bright so I turned the quality down to 1 and experimented by lowering the brightness of the other lights until I got a result I liked. Then I turned the quality back up to 3 and did the render pictured. This is a good way to experiment without taking too much time. You probably won't get the effect you want the first time you try. But that's the point of this lesson: the lighting makes all the difference and you have to experiment until you get it right.

All this quality comes with a penalty. Interestingly, the size of your model does not have much effect on render time. Whether your model has 1 part or 1000 parts, either way the computer has to calculate the color of every pixel individually. A larger model has more parse time (the time to load it into memory) and requires more RAM, but not much more render time. On the other hand, lighting is a major CPU time factor. The first image with just the 3 point lights took 2.6 minutes to render on a very fast computer at 1280x960 pixels. The second image with the area lights took 14.5 minutes. This is not surprising considering it effectively has 9 times more light sources (each area light is 3x3). The final image with radiosity at quality level 3 took 45 minutes. If I had used Quality 5, it would have taken many hours. The largest model I ever rendered had almost half a million parts (the Klingon Bird of Prey below). Parsing took 15Gb of RAM, but it rendered in only a couple of hours. On the other hand, the longest render I've ever done (the tow truck below) rendered continuously for 48 days due to the many light sources and especially the ground fog. All this means you'll have to balance your patience with the quality you want to achieve.

Happy rendering!
#