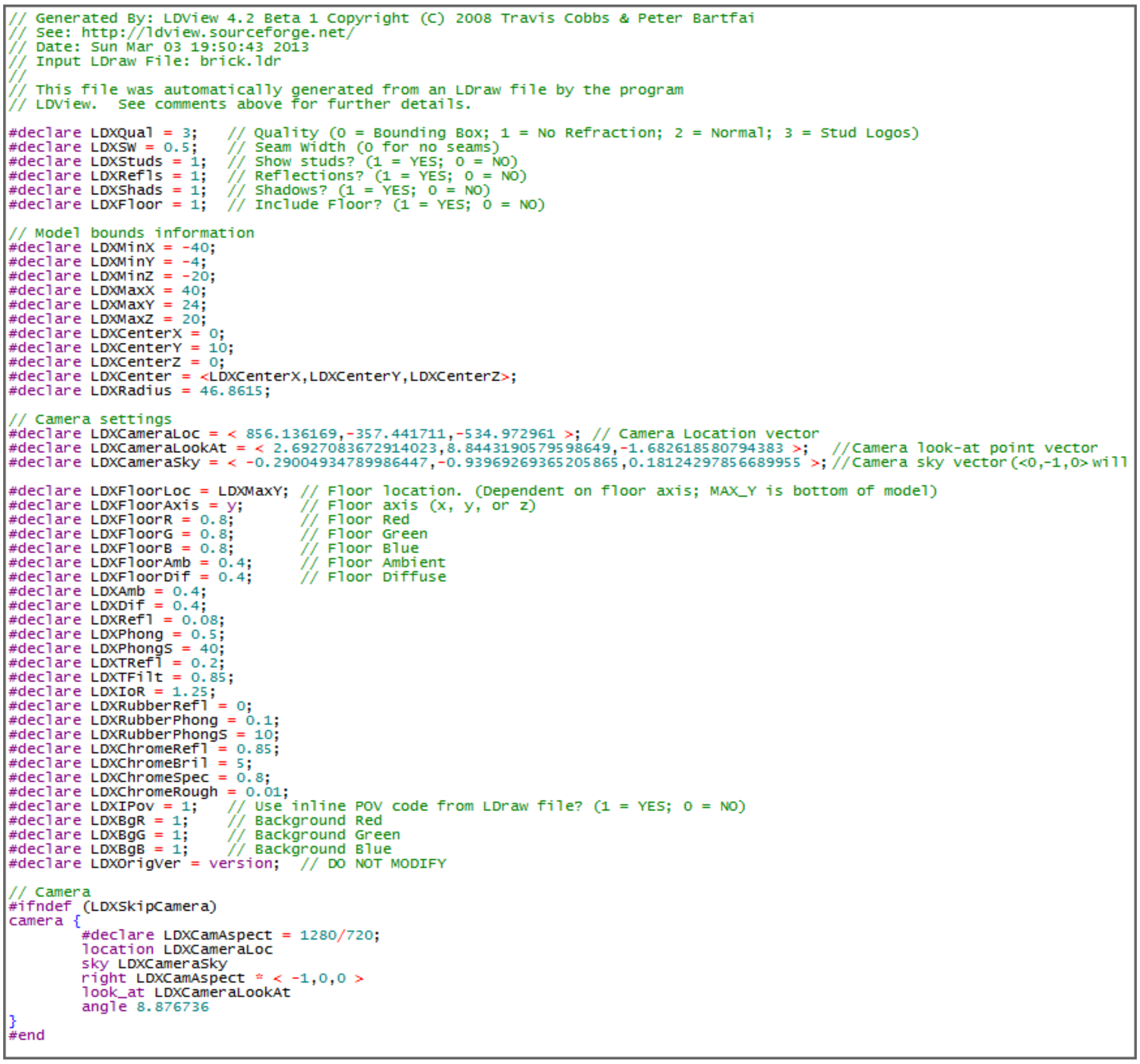
POV-Ray Tutorial (II)

By Eric Albrecht

Last time we learned how to create a POV-Ray render by starting with an LDraw file created in any editor, opening it in LDView, and exporting to POV-Ray. We also learned how to tell LDView to automatically use the LGEO library of POV-Ray parts to create better looking renders. Once this has all been set up per the last tutorial, all of it works without too much intervention on the part of the user. Open, export, open, render. If we want to advance further from here though, we're going to have to do some things manually and that means understanding what is actually inside the POV-Ray file. Let's start with that very simple file we created last time containing a single red brick. If you don't have that file, just export anything from LDView to get started.



```
// Generated By: LDView 4.2 Beta 1 Copyright (C) 2008 Travis Cobbs & Peter Bartfai
// See: http://ldview.sourceforge.net/
// Date: Sun Mar 03 19:50:43 2013
// Input LDraw File: brick.ldr
//
// This file was automatically generated from an LDraw file by the program
// LDView.  See comments above for further details.

#declare LDXQual = 3;   // Quality (0 = Bounding Box; 1 = No Refraction; 2 = Normal; 3 = Stud Logos)
#declare LDXSW = 0.5;   // Seam Width (0 for no seams)
#declare LDXStuds = 1;  // Show studs? (1 = YES; 0 = NO)
#declare LDXRefls = 1;  // Reflections? (1 = YES; 0 = NO)
#declare LDXShads = 1;  // Shadows? (1 = YES; 0 = NO)
#declare LDXFloor = 1;  // Include Floor? (1 = YES; 0 = NO)

// Model bounds information
#declare LDXMinX = -40;
#declare LDXMinY = -4;
#declare LDXMinZ = -20;
#declare LDXMaxX = 40;
#declare LDXMaxY = 24;
#declare LDXMaxZ = 20;
#declare LDXCenterX = 0;
#declare LDXCenterY = 10;
#declare LDXCenterZ = 0;
#declare LDXCenter = <LDXCenterX,LDXCenterY,LDXCenterZ>;
#declare LDXRadius = 46.8615;

// Camera settings
#declare LDXCameraLoc = < 856.136169,-357.441711,-534.972961 >; // Camera Location vector
#declare LDXCameraLookAt = < 2.6927083672914023,8.8443190579598649,-1.682618580794383 >;  //Camera look-at point vector
#declare LDXCameraSky = < -0.29004934789986447,-0.93969269365205865,0.18124297856689955 >; //Camera sky vector(<0,-1,0> will

#declare LDXFloorLoc = LDXMaxY; // Floor location. (Dependent on floor axis; MAX_Y is bottom of model)
#declare LDXFloorAxis = y;      // Floor axis (x, y, or z)
#declare LDXFloorR = 0.8;       // Floor Red
#declare LDXFloorG = 0.8;       // Floor Green
#declare LDXFloorB = 0.8;       // Floor Blue
#declare LDXFloorAmb = 0.4;     // Floor Ambient
#declare LDXFloorDif = 0.4;     // Floor Diffuse
#declare LDXAmb = 0.4;
#declare LDXDif = 0.4;
#declare LDXRefl = 0.08;
#declare LDXPhong = 0.5;
#declare LDXPhongS = 40;
#declare LDXTRefl = 0.2;
#declare LDXTFilt = 0.85;
#declare LDXIoR = 1.25;
#declare LDXRubberRefl = 0;
#declare LDXRubberPhong = 0.1;
#declare LDXRubberPhongS = 10;
#declare LDXChromeRefl = 0.85;
#declare LDXChromeBril = 5;
#declare LDXChromeSpec = 0.8;
#declare LDXChromeRough = 0.01;
#declare LDXIPov = 1;   // Use inline POV code from LDraw file? (1 = YES; 0 = NO)
#declare LDXBgR = 1;    // Background Red
#declare LDXBgG = 1;    // Background Green
#declare LDXBgB = 1;    // Background Blue
#declare LDXOrigVer = version;  // DO NOT MODIFY

// Camera
#ifndef (LDXSkipCamera)
camera {
        #declare LDXCamAspect = 1280/720;
        location LDXCameraLoc
        sky LDXCameraSky
        right LDXCamAspect * < -1,0,0 >
        look_at LDXCameraLookAt
        angle 8.876736
}
#end
```

The first thing you will notice is that there are multiple colors on the screen. Each color represent something specific in POV-Ray, so we'll talk about each separately.

Green is reserved for comments. POV-Ray totally ignores comments, so they are only there to help you, the user. LDView does a pretty good job of commenting its output to tell you what it is doing, and it is a good idea to create comments for yourself when you change the file so you can keep track of what is going on. This example file is quite simple and occupies only a couple of screens, but if you get thousands of parts your file will be many Mb of raw text and it can be pretty hard to find anything if you haven't commented it properly.

There are two ways to designate a comment line in POV-Ray. The most common way is simply to put two backslashes at the beginning of the line. Go ahead and try it. Type two slashes at the beginning of a line and you will see it turn green in the editor. Sometimes you may want to make a large block into a comment. For example, you might have two different lighting sets or two different cameras, and you can “comment out” one or the other to disable them. To turn multiple lines into a comment, you put /* at the beginning and */ at the end. You can span as many rows as you like, even the entire file. Try it out.

Purple text is used for commands. There are hundreds of different commands in POV-Ray, but you’ll only be using a small number of them. You can see a sample of those used by the LDView export in the file.

Blue text is used for values. For example, this could be coordinates or any other kind of number.

Red text is used for operators such as +, -, * and so forth. You can use it to do math inside your file. This becomes really important when you are programming animations, for example. Red is also used for filenames if you call external files, which you’ll do a lot.

Finally, black text is for everything else but this mostly amounts to variable names.

Now let’s start at the beginning of the file and see what’s in there. The first few rows are comments from LDView just telling you that this file is from LDView and crediting Travis Cobbs for all his excellent work writing the program.

First comes some variable definition.
The six variables here can be used to change the behavior of the render.

· The first variable, LDXQual, changes the overall quality of the render. If you put a 0 in here your parts will be replaced with simple rectangles which will allow you to do a render very quickly and check that the camera angles and lighting are correct.
If you put a 1 in here you get real parts, but light is not refracted through transparent parts which takes a long time. 2 is the standard setting. Finally, 3 turns on the stud logos which means a tiny LEGO® logo is represented on every stud. Stud logos add a lot of detail to your model but of course this comes at the expense of memory and processing time.

· Next is LDXSW which controls a seam width. This is a way of adding a little bit of space between parts so that they do not all look perfectly flush which is what you will get by default using LDraw geometry. A value of 0.5 (this means ½ of an LDraw unit) works really well here. One caveat is that if your model has stickers this will make them disappear because stickers are less than 0.5 LDU thick! It took me a long time to figure out why that was happening.

· LDXStuds turns all the studs on or off. I would never shut this off. The whole point of doing a render is to have it look real and LEGO without studs it just doesn’t. The only time I have ever shut this off is when I did a render so big (over a million parts) that you couldn’t see the studs anyway and shutting them off decreased the memory requirements massively.

· LDXRefls turns reflections on and off. See above. I always leave them on.

· LDXShads is for shadows. Same as above.

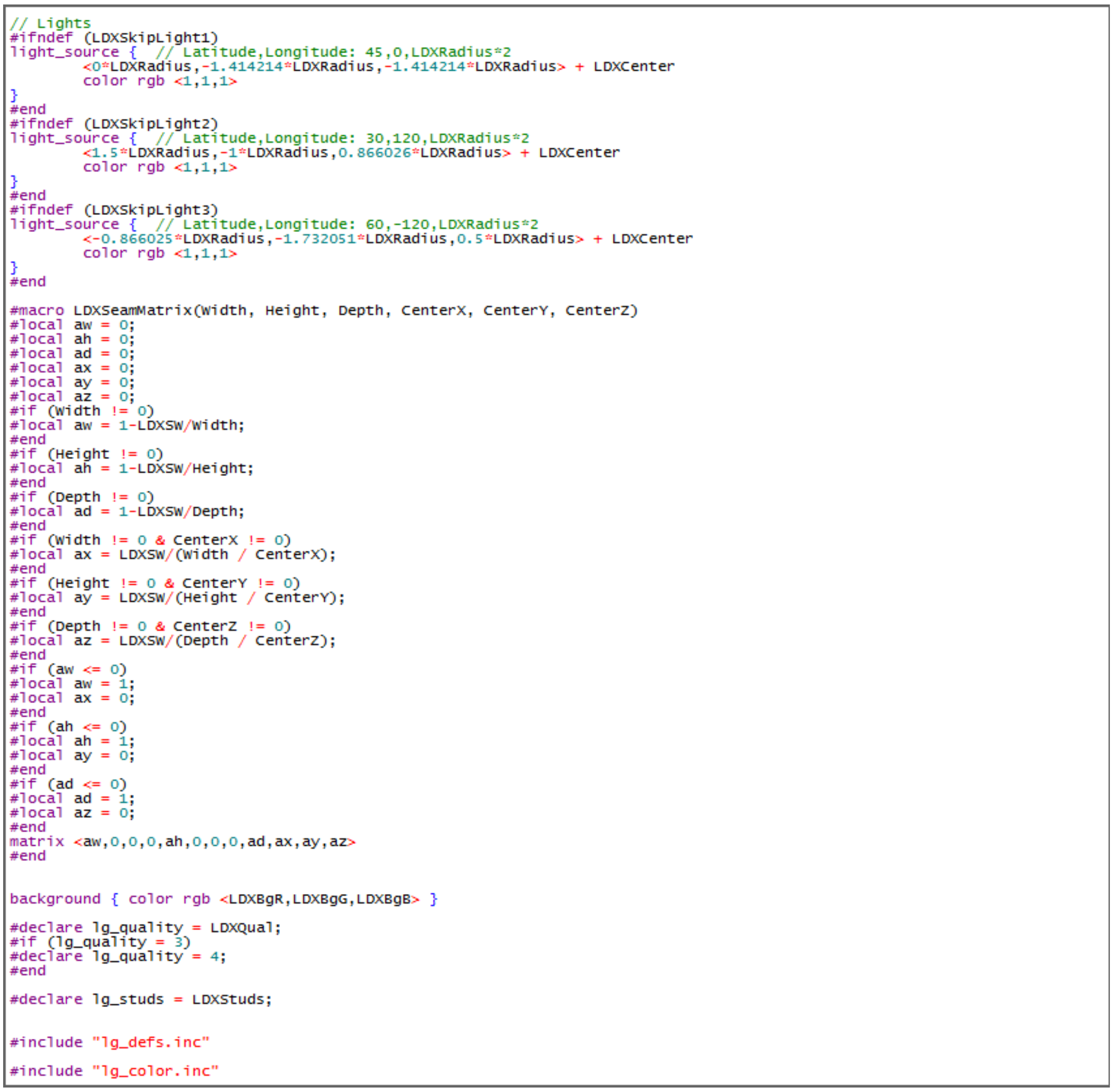
· Finally, LDXFloor turns a floor on and off. Note that LDView assumes that your model is built with -y as the up direction which is what MLCAD uses. To place the floor, it therefore finds the most positive y value in the model and creates an XZ plane at that location. You may or may not want to see this, and even if you want it, this may not be where you want it. Of course, it can be moved and the behavior of it can be changed.

After that is a section called Model Bounds. This defines the maximum extent of the model in x, y, and z. You may not see the point of this, but the availability of these variables is amazingly useful later. The maximum extents also allows the center to be calculated. There is also a radius which assumes the rectangular prism mapped by the other values is a sphere. This is used to place the camera and lights.

Next comes the section which defines some variables used by the camera. These include the position of the camera, the direction it is pointing, and a definition of which way is up. These are LDXCameraLoc, LDXCameraLookAt, and LDXCameraSky.

Next come a whole lot of other variable definitions. I won’t go through every one in detail, but they include the position of the floor, the color of the floor, some properties of the floor, some variables used to define colors, and the background color. In general, we aren’t going to mess with these, but of course anything can be changed. One thing to note here is that colors are defined as percentages of red, green, and blue. So each part of the RGB value can be between 0 and 1. If you are converting from another system like hexadecimal in which full color is FF or 255, you’ll have to do some math to make them percentages instead.

Now the camera itself is defined. There are lots of kinds of cameras in POV-Ray, but the default camera from LDView will always be a perspective camera. If you want to use any of the other specialty types (like fisheye), you will have define them yourself. The parameters for the camera come from the variables defined above. These, in turn, came from however you had the view set up in LDView. The camera simply matches what you saw in LDView. One thing to be careful of here is LDXCamAspect which defines the aspect ratio. This will be based on the shape of your window in LDView. When you render the image, you need to use this same aspect ratio or your render will be skewed. You do not need to match the pixel values, only the ratio. It is good idea to make sure your window in LDView is a standard ratio like 4:3 or 16:9 before exporting in the first place.



```
// Lights
#ifndef (LDXSkipLight1)
light_source {  // Latitude,Longitude: 45,0,LDXRadius*2
        <0*LDXRadius,-1.414214*LDXRadius,-1.414214*LDXRadius> + LDXCenter
        color rgb <1,1,1>
}
#end
#ifndef (LDXSkipLight2)
light_source {  // Latitude,Longitude: 30,120,LDXRadius*2
        <1.5*LDXRadius,-1*LDXRadius,0.866026*LDXRadius> + LDXCenter
        color rgb <1,1,1>
}
#end
#ifndef (LDXSkipLight3)
light_source {  // Latitude,Longitude: 60,-120,LDXRadius*2
        <-0.866025*LDXRadius,-1.732051*LDXRadius,0.5*LDXRadius> + LDXCenter
        color rgb <1,1,1>
}
#end

#macro LDXSeamMatrix(Width, Height, Depth, CenterX, CenterY, CenterZ)
#local aw = 0;
#local ah = 0;
#local ad = 0;
#local ax = 0;
#local ay = 0;
#local az = 0;
#if (Width != 0)
#local aw = 1-LDXSW/Width;
#end
#if (Height != 0)
#local ah = 1-LDXSW/Height;
#end
#if (Depth != 0)
#local ad = 1-LDXSW/Depth;
#end
#if (Width != 0 & CenterX != 0)
#local ax = LDXSW/(Width / CenterX);
#end
#if (Height != 0 & CenterY != 0)
#local ay = LDXSW/(Height / CenterY);
#end
#if (Depth != 0 & CenterZ != 0)
#local az = LDXSW/(Depth / CenterZ);
#end
#if (aw <= 0)
#local aw = 1;
#local ax = 0;
#end
#if (ah <= 0)
#local ah = 1;
#local ay = 0;
#end
#if (ad <= 0)
#local ad = 1;
#local az = 0;
#end
matrix <aw,0,0,0,ah,0,0,0,ad,ax,ay,az>
#end

background { color rgb <LDXBgR,LDXBgG,LDXBgB> }

#declare lg_quality = LDXQual;
#if (lg_quality = 3)
#declare lg_quality = 4;
#end

#declare lg_studs = LDXStuds;


#include "lg_defs.inc"

#include "lg_color.inc"
```

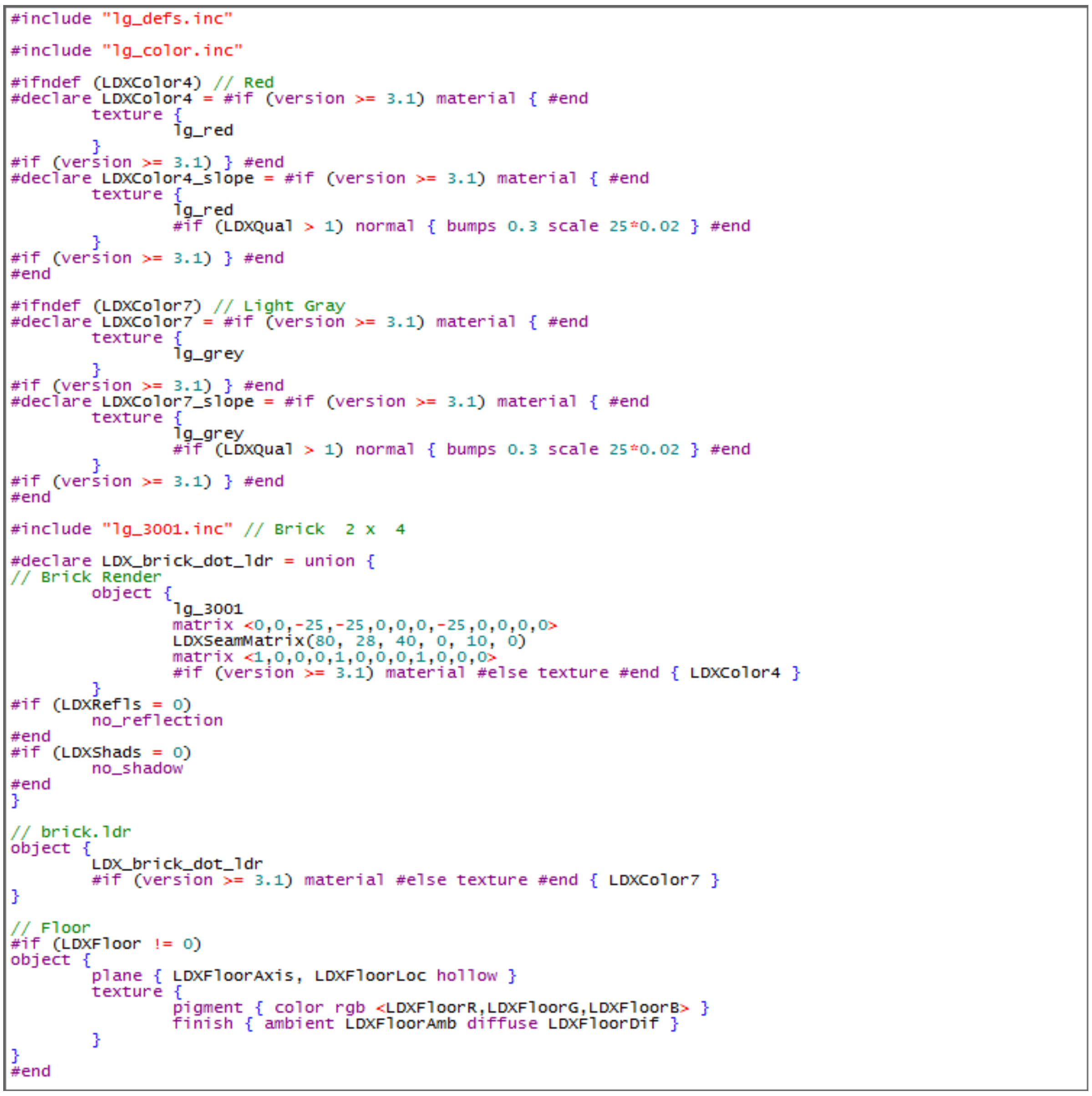
Now the lights are defined. The default lights are 3 white point lights. This section, more than any other, is what we will be changing later to make the scene more realistic. You can see that the lights are arranged using a latitude and longitude concept. Each light has a position and a color. The color is defined as a vector with RGB values. rgb <1,1,1> is white.

Next comes a bunch of math which defines a subroutine used to make the seams between the parts. You definitely don't want to mess with this.

Next the background color is defined. This is also white by default. You can shut off the background entirely by commenting out this line.

POV-Ray files don't usually contain all the required information right in the file. Instead they call other files called "include" files. A large POV-Ray file may calls hundreds or even thousands of other files. This keeps the size of your file relatively small, but you need to keep in mind when planning for memory requirements that everything needs to be in memory when you render. Sometimes it is useful to open some of these files just to understand how much work POV-Ray is really doing.

The first two include files that are loaded are related to the LGEO library. lg_defs contains the definition of many variables used by LGEO parts files. This includes things like the size of a stud, what the logo looks like, the width of a plate, and hundreds of other things. Go ahead and take a look at it. Just right click on the red title of the include file and you can open it. lg_color is no mystery. It defines the colors used by LGEO. There are a surprisingly large number of LEGO colors!



```
#include "lg_defs.inc"

#include "lg_color.inc"

#ifndef (LDXColor4) // Red
#declare LDXColor4 = #if (version >= 3.1) material { #end
        texture {
                lg_red
        }
#if (version >= 3.1) } #end
#declare LDXColor4_slope = #if (version >= 3.1) material { #end
        texture {
                lg_red
                #if (LDXQual > 1) normal { bumps 0.3 scale 25*0.02 } #end
        }
#if (version >= 3.1) } #end
#end

#ifndef (LDXColor7) // Light Gray
#declare LDXColor7 = #if (version >= 3.1) material { #end
        texture {
                lg_grey
        }
#if (version >= 3.1) } #end
#declare LDXColor7_slope = #if (version >= 3.1) material { #end
        texture {
                lg_grey
                #if (LDXQual > 1) normal { bumps 0.3 scale 25*0.02 } #end
        }
#if (version >= 3.1) } #end
#end

#include "lg_3001.inc" // Brick  2 x  4

#declare LDX_brick_dot_ldr = union {
// Brick Render
        object {
                lg_3001
                matrix <0,0,-25,-25,0,0,0,-25,0,0,0,0>
                LDXSeamMatrix(80, 28, 40, 0, 10, 0)
                matrix <1,0,0,0,1,0,0,0,1,0,0,0>
                #if (version >= 3.1) material #else texture #end { LDXColor4 }
        }
#if (LDXRefls = 0)
        no_reflection
#end
#if (LDXShads = 0)
        no_shadow
#end
}

// brick.ldr
object {
        LDX_brick_dot_ldr
        #if (version >= 3.1) material #else texture #end { LDXColor7 }
}

// Floor
#if (LDXFloor != 0)
object {
        plane { LDXFloorAxis, LDXFloorLoc hollow }
        texture {
                pigment { color rgb <LDXFloorR,LDXFloorG,LDXFloorB> }
                finish { ambient LDXFloorAmb diffuse LDXFloorDif }
        }
}
#end
```

Next are some color definitions. You'll only find definitions for the colors you used. In this case, that is just red. So why is light gray there? The top level assembly is always considered light gray, so this will always be there. If you open the lg_colors file, you will see that the colors are defined not only with a color, but with some other surface properties like reflectivity. We will also end up making a lot changes here in later tutorials.

Finally it is time for the actual parts. In this case, we have only a single part: a 2x4 brick. This part has been replaced with an LGEO part, lg_3001. So all POV-Ray needs to do is call the lg_3001.inc include file. Again, if you right click on that text you can open that file. In the next page you can see how amazingly complicated that single part is!

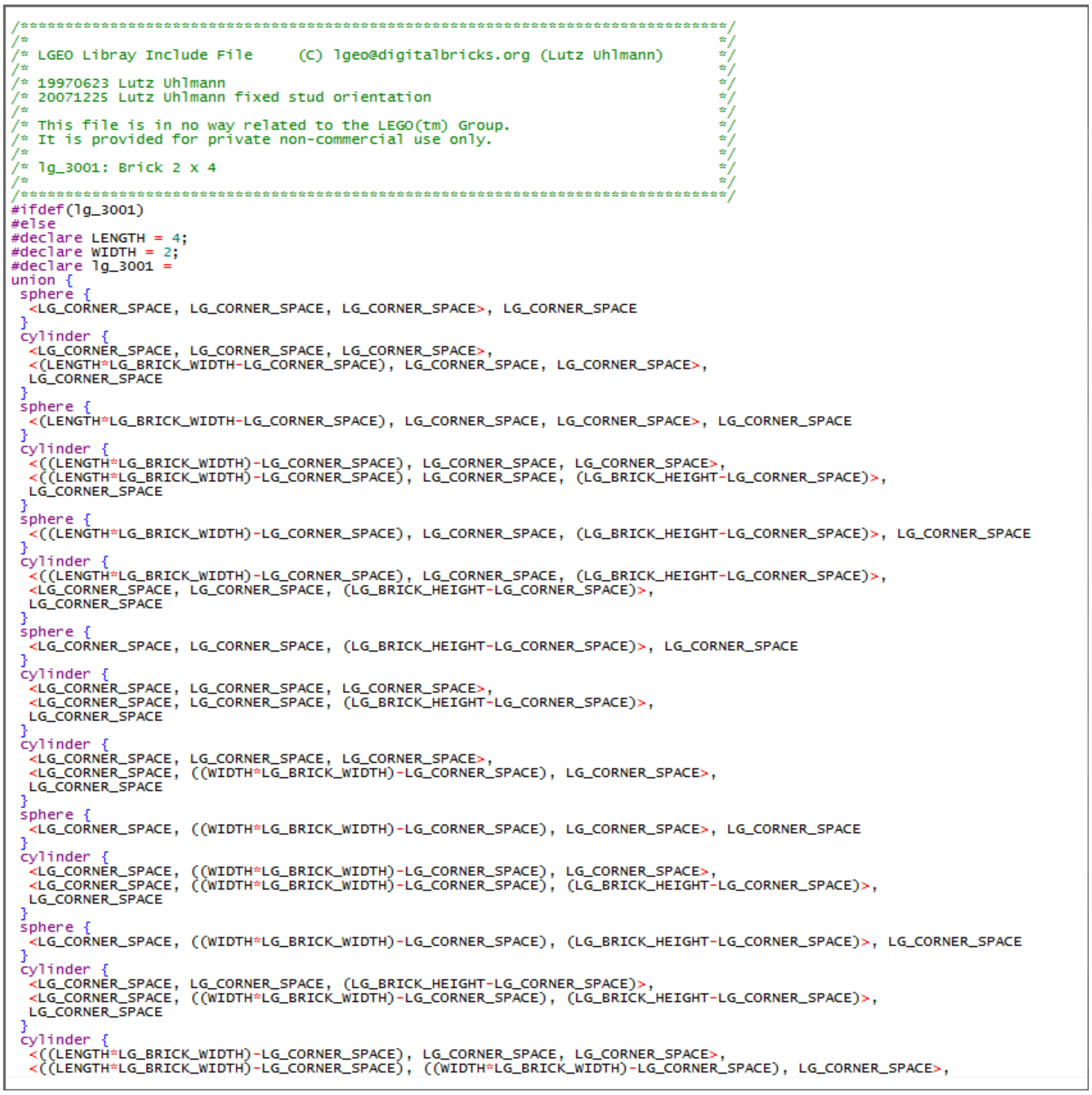
This will start to give you an appreciation for the work POV-Ray is doing for you. Remember that this is only the definition of one type of part, and you may have hundreds of types of parts in your model.

Some parts do not have an LGEO substitute. In those cases, you will see a large part definition right here within the main POV-Ray file generated by LDView. This definition may span hundreds of pages for a complex part like a curved Technic panel or a Fabuland head.

Our scene only included one part include file, but a larger model may include many. Each is only included once, but there may be many instances of that part called when your model is assembled. That's what we find next in our file. Now that POV-Ray knows what a lg_3001 part looks like, we can place some bricks in space. This is the section that starts with "object". Here, an object is created in space from lg_3001. The most important line is this one:

matrix <1,0,0,0,1,0,0,0,1,0,0,0>

Remember when we learned a little about transformation matrices in the last lesson? The first nine digits here are the rotation in the x, y, and z planes right out of our LDraw file. The next 3 are the position in space, in this case 0,0,0. This is a very simple



```
/**************************************************************************/
/*                                                                        */
/* LGEO Libray Include File     (C) lgeo@digitalbricks.org (Lutz Uhlmann) */
/*                                                                        */
/* 19970623 Lutz Uhlmann                                                  */
/* 20071225 Lutz Uhlmann fixed stud orientation                           */
/*                                                                        */
/* This file is in no way related to the LEGO(tm) Group.                  */
/* It is provided for private non-commercial use only.                    */
/*                                                                        */
/* lg_3001: Brick 2 x 4                                                   */
/*                                                                        */
/**************************************************************************/
#ifdef(lg_3001)
#else
#declare LENGTH = 4;
#declare WIDTH = 2;
#declare lg_3001 =
union {
 sphere {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, LG_CORNER_SPACE>, LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <(LENGTH*LG_BRICK_WIDTH-LG_CORNER_SPACE), LG_CORNER_SPACE, LG_CORNER_SPACE>,
  LG_CORNER_SPACE
 }
 sphere {
  <(LENGTH*LG_BRICK_WIDTH-LG_CORNER_SPACE), LG_CORNER_SPACE, LG_CORNER_SPACE>, LG_CORNER_SPACE
 }
 cylinder {
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 sphere {
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>, LG_CORNER_SPACE
 }
 cylinder {
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  <LG_CORNER_SPACE, LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 sphere {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>, LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <LG_CORNER_SPACE, LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE>,
  LG_CORNER_SPACE
 }
 sphere {
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE>, LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE>,
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 sphere {
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>, LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 cylinder {
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE>,
```

example, but this term can get really complicated. There may be cases in which you want to manually move or rotate a part by changing these values.

Next you can see that the color red is assigned to this part (LDXColor 4). Again, it is nice to be able to change these colors manually by knowing where to look.

If we had a larger model made from submodels, we'd see each submodel defined as a "union" or a combination of parts, and then those submodels would be joined with a larger union at the end. The last object in the file before the floor is always the top level assembly. In this case, that assembly is just our one brick. You can see what I told you earlier how this top assembly is defined as light gray (LDXColor 7). This doesn't mean anything. The parts already have a color and this will not overwrite them. Only part colors are important, not assembly colors.

The very last thing that is defined is the floor, and it uses the variables from the beginning of the file. Again, we'll be making changes to this later.

It may seem like most of the file was "overhead" defining variables and cameras and lights, but not much model. That is true for such a small model of a single part. When you get a big model, you'll find

Having now been through a file, you should have a basic understanding of what it all means. This is a small file that is relatively easy to follow. Now I encourage you to just try changing everything and see what happens. Change each variable one at a time and observe the effect. This is the best way to learn what they do. Next time I'll teach you how to change some of the most important variables and definitions to improve the quality of your renders.
#