COS 116
The Computational Universe
Laboratory 6: Computer Graphics

As mentioned in lecture, computer graphics has four major parts: imaging, rendering, modeling, and animation. In this lab you will learn about rendering and modeling. You will make use of two well known programs: Google's SketchUp, a free 3D modeling tool, and POV-Ray, an open source raytracing renderer. You will also learn about OpenGL, the rendering software used by SketchUp and games such as World of Warcraft.

If you get stuck at any point, feel free to discuss the problem with another student or a TA. However, you are not allowed to copy another student's answers.

Hand in your lab report at the beginning of lecture on Tuesday, March 27. Include responses to questions printed in bold (number them by Experiment and Step), as well as the Additional Questions at the end.

Before you begin:

1) You'll need to download `povview_extract.exe` from the course website. Once you have it on your desktop, double click it to run and then accept the default extraction folder, which is the `povview` folder on your Desktop.
2) Set your display color settings to Highest (32 bit). To do this, right click anywhere on the desktop and choose Properties. In the Display Properties dialog box, choose the Settings tab. In the Color Quality field, make sure "Highest (32 bit)" is selected (the Friend 005 laptops are usually set to "Medium (16 bit)").

Warm Up: The Povview viewer

Povview is a small graphical program written by the course staff which acts as a wrapper around POV-Ray. You will use it to set up simple scenes for raytracing.

1. **Open `povview.exe` in the `povview` folder on your Desktop.**
2. **Select File->Open and choose `bedroom.kmz`.**
3. After a few seconds, you should see a simple scene with a desk, chair, and set of bunkbeds.

The main window of Povview is a 3D camera view that can be manipulated with the mouse. Click and drag the left mouse button to rotate the camera. Click and drag the middle button to pan (move from side to side) the camera. Click and drag the right button to zoom the camera in and out of the screen.

4. The bedroom scene contains three incongruous (poorly) hidden objects. **Find all three by moving the camera using the mouse as described above.**
5. When you have found an object, try to frame it nicely in the view. **Then press Ctrl+S to take screenshot. Choose Save as type: Windows Bitmap. Save the file to something you will remember** (just accept the defaults in the second dialog box)**.**
6. Include a nicely framed shot of each object in your lab report.

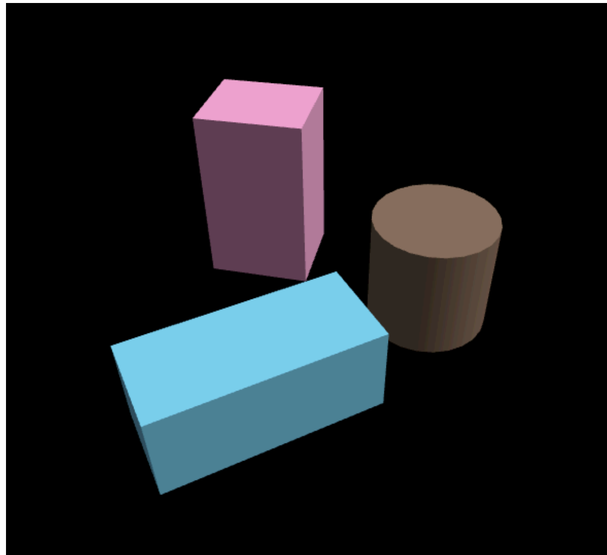
Experiment 2: Selecting Objects and Changing Pigments

Lets try to color our objects something other than gray.

1. **Open the file `three_obj.kmz`.**
2. **Double click on the cylinder.** A yellow outline appears around the cylinder and a sidebar appears with several new options.
3. **Click the button in the upper right of the sidebar, next to the word "Pigment."** A dialog box appears that allows you to choose a color for the cylinder. **Choose a red color and click OK.**
4. The cylinder should become red, and you should notice that the values in the Red/Green/Blue edit boxes have changed.

The simplest way to represent a pigment is as a triplet of red, green, and blue (RGB) color values. The RGB values describe how much red, green, or blue light is reflected by a pigment. Traditionally, these values lie in the range 0.0 (no light reflected) to 1.0 (all light reflected). In Povview, you can change the color of an object by either changing these edit boxes, or by picking a color from the color dialog.

5. **Using the same procedure, make one of the boxes green and the other one blue.**
6. **Now, try to match the colors in the following image. Write down the associated red, green, and blue values for each object in your notes.**



Experiment 3: Working with Lights

Povview has two lighting environments. One is meant to simulate outdoor light with a single, very distant light source (the Sun), while the other is meant to simulate an indoor situation with several nearby point light sources.

1. **Select View->Show Lights.** You should see a white line extending upwards from the center of your scene. This line represents the direction of the sun, and its color represents the color of the sun's light.
2. **Double click on the white line (you may have to be somewhat precise).** A new sidebar will appear giving you some options to change the light.
3. The position of the sun can be adjusted by moving the position slider. **Try sliding it back and forth from sunrise to sunset.**
4. Recall from lecture that a common lighting hack is an Ambient term that controls the general level of brightness in the scene. **Try adjusting the Ambient Power to see its effect.**
5. You can change the color of the sun much as you changed the color of the objects. **What color will each of the three objects appear if you change the sun's color to red? To blue? To green? Write down your predictions in your notes, and then try each color to see if you were right.**
6. **Try the indoor lighting environment: choose Lighting->Indoor.** You will see four point lights instead of the long sun line.
7. You can select the point lights in the same way as everything else. **Try changing the color of each light individually: make one or two green, blue, or red.**
8. **Were you surprised by any of these results? Do the results seem realistic? Why or why not?**

Experiment 4: Triangulation

As explained in class, 3D models are often constructed from a set of triangular pieces. This experiment explores this type of model, often called a *triangle mesh.*

1. Each rectangular box in your scene of three objects is a made up of eight *vertices* (the corners) and triangles connecting the vertices. Assume the triangles can only connect the eight corners of a box. **How many triangles make up a box?**
2. **Turn on wireframe rendering by selecting View->Wireframe. Check your calculation by counting the number of triangles you see in each box.** Also, look at how many triangles the "cylinder" contains!
3. This configuration is only one way of triangulating the objects. **How many unique triangulations of a box with eight vertices exist (ignore symmetry)?**
4. **Open `icos.ply`.** You should see an icosahedron, which is a shape with 20 triangular sides.

A triangle mesh can perfectly represent a cube with only a small number of triangles. Curved shapes, however, are more difficult to represent – remember how many triangles the cylinder used. A sphere is another shape that can only be approximated by a triangle mesh. One way to approximate a sphere is by *subdividing* a roughly spherical mesh such as an icosahedron.

5. **Select the icosahedron by double-clicking it, and then choose Model->Subdivide.** Look at the result in wireframe mode. **How many triangles does the new shape have?** Hint: don't count all the triangles – figure out what happened to each of the icosahedron's sides, then multiply that number by 20. It may help to reload `icos.ply` and subdivide over again.
6. **Subdivide the shape again. How many triangles does the doubly subdivided shape have? If you subdivided an icosahedron *n* times, how many triangles would the resulting shape have?**
7. **How many times do you have to subdivide the icosahedron before the mesh looks like a sphere? How many triangles does the almost spherical mesh have?** Warning: the program may start getting slow if you subdivide too many times.

Experiment 5: Finishes

You may have noticed that the sidebar has a bunch of other options besides pigment coloring. Lets see what the Finish box does.

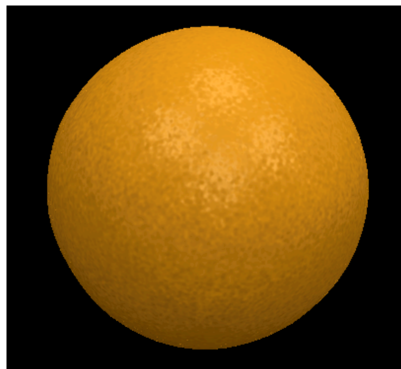
1. **Select your subdivided icosahedron (approximated sphere) and turn off wireframe mode.** You should see a nice, smooth, soft gray shape.
2. **Change the Specular edit box to 0.0.** The sphere should become slightly duller: this is pure *diffuse* lighting.
3. **Move the camera around the sphere. Does the color of a given point on the sphere change when you move the camera? Turn on Lighting->Outdoor and View->Show Lights. Move the sun around as in experiment 3. Does the color of a given point change when you move the light? Describe qualitatively the effects that you see.**
4. **Change the Diffuse edit box to 0.0.** The sphere should become a flat, gray circle. This is pure *ambient* lighting.
5. **Move the camera and the sun around again. Does the color of the sphere change as you move the camera? How about the light?**
6. **Select the sphere again and increase the Specular value to 1.0.** You should see a bright, white highlight on the sphere. This highlight is an example of a simulated *specular reflection*. It is meant to depict the bright, mirror like reflection of light off of the sphere.
7. **Move the camera around again. Does the highlight move when you move the camera? Move the sun again. Does the highlight move when you move the sun?**
8. The last finish parameter is Roughness. **Set the Diffuse and Specular boxes to reasonable values, say 0.5 and 0.5. Now try changing the Roughness value by clicking the up and down arrows next to its edit box.**
9. **How does the shape of the specular highlight change as you change the Roughness? Does higher or lower Roughness look “rougher” to you?**

Experiment 6: Surfaces

In this experiment we will start using POV-Ray. So far you have been viewing models using a renderer based on OpenGL. OpenGL is a *rasterizing* renderer, which basically means that it looks at each triangle, projects that 3D triangle into the 2D screen (much as your 3D body projects a 2D shadow on the ground), and then draws that whole 2D triangle in one shot. By comparison, POV-Ray is a *raytracing* renderer. A raytracer does not directly process each triangle; instead, it shoots virtual rays of light into the scene and computes where each ray hits a triangle. Raytracing is much more computationally intensive than rasterizing, as you will see. However, raytracing makes modeling some complex lighting effects easier. One of these effects is called *bump-mapping*.

Important: POV-Ray can be slow to render. The time taken by POV-Ray is directly proportional to the size of the Povview window: if you shrink the Povview window prior to rendering in POV-Ray, you will get a smaller output image but faster rendering times.

1. You should still have your smooth, subdivided and somewhat shiny sphere up on the screen. **Try creating an image of it using POV-Ray by hitting Ctrl+R or selecting File->Render.** POV-Ray will load and begin rendering the sphere (POV-Ray may give you a warning about not being installed correctly, just click OK).
2. Povview should now have an open Output window with an image of sphere in it. **Compare it to the OpenGL version: do they look similar? Different?**
3. **Close the Output window and select the sphere again. Click “Bumpy” in the Surface box.** You will see no effect on the shape in the OpenGL rendering.
4. **Now, render the sphere again (Ctrl+R) and compare the two images again. What changed?**
5. The Surface box has two parameters, Height and Scale. Height controls how tall the bumps appear to be, while Scale controls their width. **Try playing around with a few different values for each and rerendering the sphere in POV-Ray.**
6. Bump mapping a is cheat: it makes the surface look bumpy, but it doesn’t actually make the geometry of the shape bumpy. **Can you find part of the output image where the bump mapping illusion breaks down?**
7. **Using everything you have learned so far, try to make your sphere look like an orange.** You won’t be able to make a perfect orange, obviously, but you should be able to make something like the example image. **Use the “Save As…” button in the Output window to save a nice image of your orange for your notes. Record the Pigment, Finish, and Surface parameters as well.**



Experiment 7: Shadows and Reflections

Recall from lecture that there are in general two ways of simulating illumination: Direct Illumination and Global Illumination. OpenGL uses only direct illumination, while POV-Ray can simulate global illumination (at least, some parts of it). This experiment will make the difference obvious.

1. You should still have your orange up on the screen. **Choose View->Show Ground to turn on a simple ground plane.**
2. **Render the scene in POV-Ray.** You should see the orange casting a shadow onto the ground.
3. **Close the output window. Select Lighting->Outdoor and View->Show Lights (if not already selected). Select the sun and move it around a bit. Render the scene again. Did the shadow move? Is the shadow in about the place you expect it, given the location of the sun?**
4. **Turn on indoor lighting with Lighting->Indoor and render again. How many shadows do you see now? Is this what you expect?**
5. Even the POV-Ray shadow calculation is only an approximation of a real shadow. **Can you describe some ways in which the shadows do not look realistic?** Hint: try placing your hand right on your desk and look at the shadow it makes. Then, raise your hand about four inches off the desk and look again.
6. **Select the ground plane by doubling clicking on it. Set its Specular value to 0.5.** Again, you should see little or no change in the OpenGL viewer.
7. **Render the scene in POV-Ray again.** You should now see the orange casting a reflection in the ground. **Try setting different values of Specular and Diffuse for the ground plane. For what values does the reflection look strongest? Weakest?**
8. Part of the fun of using a raytracer is setting up interesting reflections. **Try this:** make sure you have all the images of your orange that you need. Then, open `sphere_stack.kmz`. You will see a small pyramid of five spheres on a plane (you will need to turn on View->Show Ground again). Change the color of each sphere to something distinctive, and change the Specular value of the ground plane to around 0.5. Render the scene in POV-Ray. **How many separate reflections can you see (approximately)?**

Experiment 8: Special Materials

POV-Ray also comes with the ability to simulate some kinds of interesting materials such as stone and wood. It does so using a *texture*, which can be thought of as a function that varies the pigment of a shape with position on the shape.

1. **Select one of the spheres in the stack and choose a special material from the Pigment list.** The sphere will go gray in the OpenGL viewer – OpenGL can render textures, but the feature is not implemented in Povview.
2. **Render the scene in POV-Ray.** You should see a sphere with a drastically different appearance than before.
3. **Try out all the different special materials.** You can make any object (including the ground plane) use a texture in this way. **Make a nice image of the sphere stack with a bunch of different textures.**
4. Each texture is meant to simulate some type of real world material. **In what ways do the textures succeed? In what ways do they fail?**

Experiment 9: Speed Issues

You may have noticed that as you started using more advanced features, POV-Ray became slower and slower. While the OpenGL viewer can create 20 or 30 new images per second (allowing you to smoothly move the camera around the scene), POV-Ray can take several seconds or even longer to create one image.

1. **Roughly time how long it takes POV-Ray to render your sphere stack scene. How much slower is this than the OpenGL viewer, assuming the viewer renders 30 images in a second (30 frames per second or 30 FPS)?**
2. There are a couple of factors that significantly impact POV-Ray's performance. **Try adjusting the Diffuse, Specular, and Materials of the scene to see if you notice any dramatic changes.** Hint: when a ray hits a dull surface, the raytracer can stop computation. When a ray hits a reflective surface, the raytracer must create another reflection ray and continue tracing.
3. Another important factor is the number of triangles in the scene. **Load up `icos.ply` again and try rendering it without subdividing at all. Now subdivide a few times and try again. Approximately what is the difference in time? Does this seem consistent with the number of triangles in the scene?**

Experiment 10: SketchUp Modeling

So far you have been using 3D models that have been provided for you. You can also create models yourself. SketchUp is a free 3D modeling tool that allows you to create 3D models and export them to Povview for rendering. SketchUp is a complicated program that takes some time to learn, so you should only attempt this part of the lab if you have plenty of time. You can also download SketchUp at home and play with it there.

1. **Load SketchUp from the Start menu. Try running through one of the tutorials in the help section to get your feet wet.**
2. **Once you have made a simple model, you can export it using File->Export->3D Model. Make sure you select "Google Earth 4" as the export format.** Put the file on the desktop or somewhere where you can find it easily.
3. **Load up Povview and open your file (it should be called `<something>.kmz`).** From here, you should be able to do everything as you have been so far.
4. The separate objects that you select in Povview correspond to Groups in SketchUp. **Try this: make a scene with two boxes. Select each box with the selection tool, then right click on each box and choose Make Group. Export the model and load it in Povview.** You should now be able to select each box separately and change its color.
5. SketchUp also has an extensive library of 3D models premade for you. **Choose File->Get Models and browse around. Try to make an interesting scene and render it in Povview.**

Extra Credit:

Now that you have some familiarity with SketchUp and POV-Ray, you can create some pretty cool images. If you have any time left over, try playing around with `bedroom.kmz` or another interesting scene that you have created. POV-Ray has many more advanced features than we have touched on here. You can use File->Preview POV-Ray Scene to see what the raw input to POV-Ray looks like. If you are feeling adventurous, read up on the POV-Ray documentation and try to apply some of those features using the Preview text editor. Some interesting things to try include the fish-eye projection, fog, and radiosity.

Some helpful links:

POV-Ray documentation: http://www.povray.org/documentation/
Examples of advanced use of POV-Ray: http://hof.povray.org/
Google Sketchup: http://sketchup.google.com/product_suf.html