Project2F17

From Immersive Visualization Lab Wiki

Contents

Project 2: 3D Models

From this point on we're no longer using the rasterizer code. In all future homework projects we're going to use OpenGL for all our rendering.

In this homework assignment you're going to learn how to:

- load polygonal OBJ files
- render polygons in modern OpenGL
- automatically center and scale 3D models
- control objects using the mouse
- set up light sources
- set material properties so that your objects look more interesting

1. 3D Model Loader (10 Points)

You already know how to load point clouds. It turns out that the 3D model files from project 1 (Bunny, Bear, Dragon) contain surface descriptions as well, by means of triangle connectivity.

Besides vertices and vertex normals you are going to have to parse the files for connectivity. It is defined with the letter 'f'

for face. Each line starting with the letter 'f' lists three sets of indices to vertices and normals, which define the three corners of a triangle. The numbers are indices into the vertex and vertex normal lists. Example:

```
f 31514//31514 31465//31465 31464//31464
```

Parsing the faces (2 Points)

Modify your OBJ loader so that it also parses the face lines, then modify your code to display triangles instead of vertices for the OBJ objects. Ensure stored faces values are 0-based.

Centering the model (4 Points)

In order to make mouse controls you are about to implement work well, you will need to center your OBJ models and standardize their sizes. This means you need to traverse the vertices of each of your models and create a translation and scale matrix that center and resizes it. Write function calls to do both of these things whenever a model is loaded from disk, and store the resulting matrix in memory. Center the model such that the geometric center of the model resides in the center of the rendering window. This can be done by looking at each dimension (x,y,z) independently and comparing the current object space origin with the geometric center of the model, calculated by finding the minimum and maximum coordinates along the respective dimension, and using the midpoint between them as the centering point for the model.

Scaling the model (4 Points)

Scaled all models to the same size, such that they fit within an imaginary 2x2x2 cube centered around the origin. Afterwards, the vertex position values should be in the range of [-1,1].

2. Rendering using modern OpenGL (20 Points)

The starter code has been modified so that the Cube now gets rendered with modern OpenGL, through the use of a VAO (Vertex Array Object), VBOs (Vertex Buffer Object), and EBOs (element buffer object). In order to make the use of the programmable pipeline possible, a shader class was added to the starter code, along with a sample vertex and fragment shader. At the moment, the sample fragment shader causes all objects to be colored pink. You will be fixing this later when you begin working with lights.

Using Cube's example of the modern OpenGL approach, add the ability to render your OpenGL using modern OpenGL. You no longer need to support rasterizer mode.

Notes:

- This (http://learnopengl.com/) is a good resource for learning modern OpenGL.

3. Mouse control (20 Points)

Since we'll be allowing the mouse to rotate the object, if you had your spin while idle code, now would be the time to disable it.

It is time to support the mouse to control your 3D models. Add functionality to allow the following operations on the displayed 3D model:

Rotating the model (6 Points)

While the left mouse button is pressed and the mouse is moved, rotate the model about the center of your graphics window. This is an extension of the orbit function ('o' key), but it should orbit the model around a sphere, not just a circle. We will refer to this operation as "trackball rotation". This video shows how this should work.

Translation in the x-y plane (6 Points)

When the right mouse button is pressed and the mouse is moved, move the model in the plane of the screen (similar to what the 'x' and 'y' keys did). Scale this operation so that when the model is in the screen space z=0 plane, the model follows your mouse pointer as closely as possible. If you don't have a right mouse button, use your mouse button along with a function key (Shift, Control, Alt, etc.) to enter this mode.

Translation along the z-axis (6 Points)

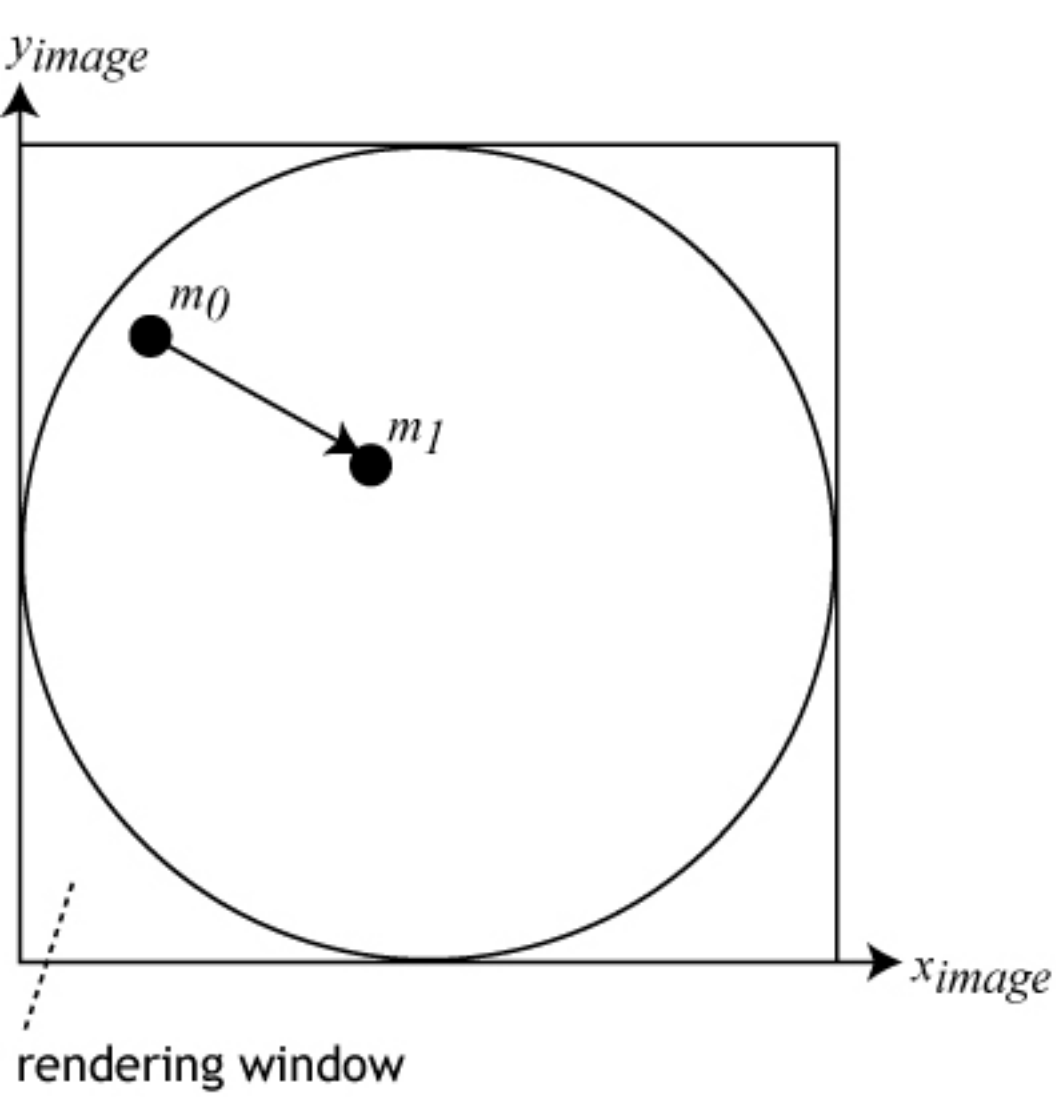
Use the mouse wheel to move the model along the screen space z axis (i.e., in and out of the screen = push back and pull closer).

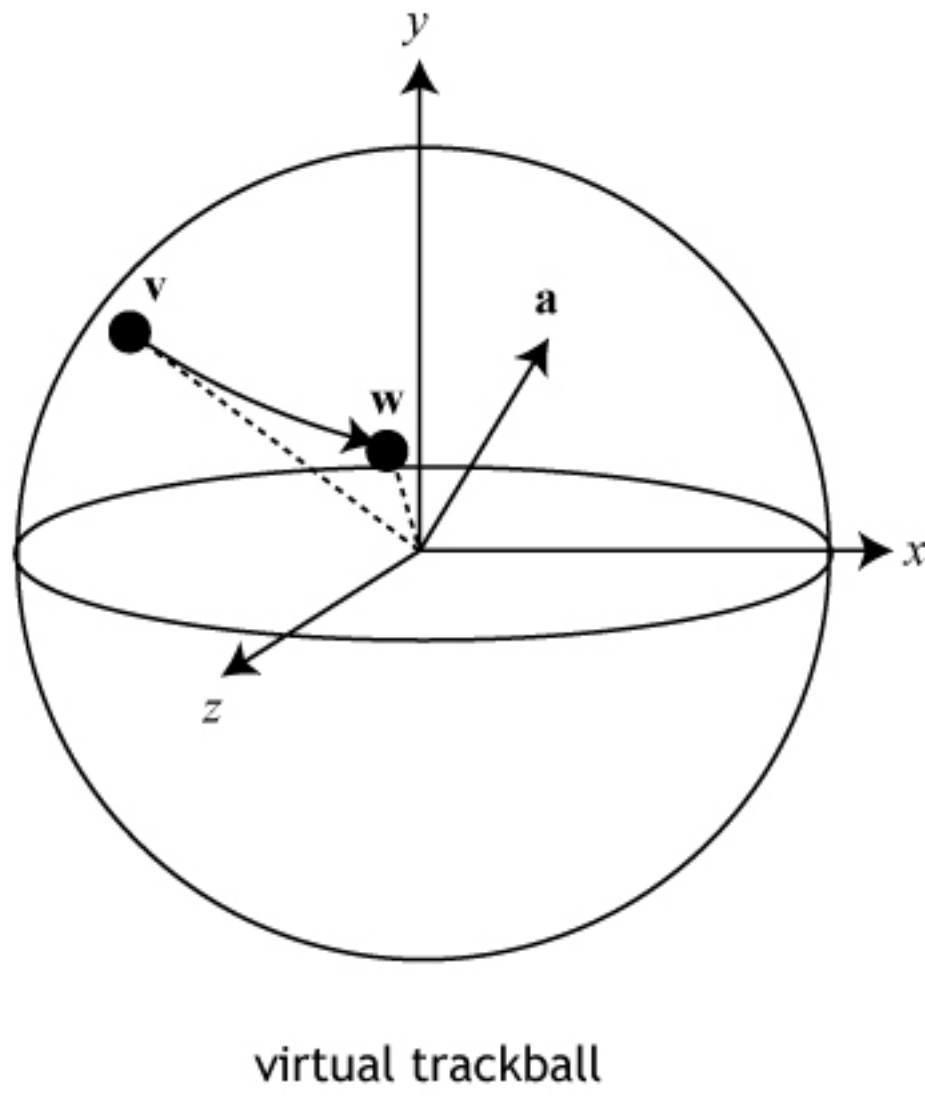
Retain reset and scale (2 Points)

Retain the functionality of the 's'/'S' keys to scale the model about its object space origin, as well as 'r' to reset the model to the origin. The other keyboard functions for the control of the 3D models are no longer needed, but it does not hurt to keep them supported.

Notes on the Trackball Rotation

The figure below illustrates how to translate mouse movement into a rotation axis and angle. m0 and m1 are consecutive 2D mouse positions. These positions define two locations v and w on an invisible 3D sphere that fills the rendering window. Use their cross product as the rotation axis a = v x w, and the angle between v and w as the rotation angle.





Horizontal mouse movement exactly in the middle of the window should result in a rotation just around the y-axis. Vertical mouse movement exactly in the middle of the window should result in a rotation just around the x-axis. Mouse movements in other areas and directions should result in rotations about an axis a which is not parallel to any single coordinate axis, and is determined by the direction the mouse is moved in.

Once you have calculated the trackball rotation matrix for a mouse drag, you will need to multiply it with the object-to-world transformation matrix of the object you are rotating.

For step by step instructions, take a look at this tutorial (http://web.cse.ohio-state.edu/~crawfis/Graphics/VirtualTrackball.html) . Note that the tutorial was written for Windows messages, instead of GLFW mouse events. This means that you'll need to replace the "CSierpinskiSolidsView::OnLButtonDown", "CSierpinskiSolidsView::OnMouseMove", etc. with an appropriate GLFW equivalent.

To help you understand the code better, here is a line-by-line commented version of the trackBallMapping function:

```
Vec3f CSierpinskiSolidsView::trackBallMapping(CPoint point)    // The CPoint class is a specific Windows class. Either us
{
    Vec3f v;    // Vector v is the synthesized 3D position of the mouse location on the trackball
    float d;     // this is the depth of the mouse location: the delta between the plane through the center of the trackb
    v.x = (2.0*point.x - windowSize.x) / windowSize.x;   // this calculates the mouse X position in trackball coordinates
    v.y = (windowSize.y - 2.0*point.y) / windowSize.y;   // this does the equivalent to the above for the mouse Y positio
    v.z = 0.0;   // initially the mouse z position is set to zero, but this will change below
    d = v.Length();    // this is the distance from the trackball's origin to the mouse location, without considering dep
    d = (d<1.0) ? d : 1.0;   // this limits d to values of 1.0 or less to avoid square roots of negative values in the fo
    v.z = sqrtf(1.001 - d*d);  // this calculates the Z coordinate of the mouse position on the trackball, based on Pytha
    v.Normalize(); // Still need to normalize, since we only capped d, not v.
    return v;  // return the mouse location on the surface of the trackball
}
```

4. Define Lights and Materials (30 Points)

In order to render the 3D models as realistically as possible, write shaders to render them with the **Phong illumination model** and **per pixel lighting**. Per pixel lighting means that you have to do all of your light reflection calculations in the fragment shader. For the shader to work, you will have to define light sources as well as different materials for each of your 3D models. Write vertex and fragment shaders to support the features listed below. In this part of the project, you should use fixed positions for the lights and all other lighting parameters. But know that in part 5 you're going to have to make some of these parameters user modifiable, so you may want to already introduce variables instead of using constant values.

Materials (5 Points)

Assign each of the 3D model files different material properties, following the instructions below. Each object should have a different color of your choice.

- One of the models should be very shiny, with no diffuse reflection.
- Another model should only use diffuse reflection, with zero shininess.
- The third object should have significant diffuse **and** specular reflection components.

Rendering Mode Switch (5 Points)

Support keyboard key 'n' to switch between the rendering modes of **normal coloring** and your new **Phong illumination model**.

- Normal coloring is useful to keep around so that you can check if your surface normals have been calculated correctly. Normal coloring should work just as you implemented it in project 1, except now you render the entire 3D model (all triangles, not just the vertices) with normal shading.
- In Phong render mode, you render your 3D models more realistically, determined by their materials and the type of light source shining on them.

Light sources

You will need to create three separate light sources: a directional light, a point light, and a spotlight. Use three C structs to define the parameters for the three types of light sources, with the parameters below.

Directional light properties (5 Points)

- Color (vec3)
- Direction (vec3)

Point light properties (5 Points)

- Color (vec3)
- Position (vec3)
- Use **linear** attenuation when calculating the light intensity.

Spotlight properties (10 Points)

- Color (vec3)
- Position (vec3)
- Direction of center of cone (vec3)
- Spot cutoff (float)
- Spot exponent (float)
- Use **quadratic** attenuation when calculating the light intensity.

5. Interactive Light Controls (20 Points)

To test out the effect of your light sources, make the light sources movable with the mouse, and add keyboard commands for certain illumination parameters as described below.

Light Selection (5 Points)

- Give the user control of the lights: make them selectable with the number keys '1', '2', and '3' (for the directional, point and spot lights, respectively). Use the '0' key to de-select and return to controlling the 3D model.
- Note that only the most recently selected light should be on at any given time, the other lights should be turned off.

Light Representation

- For directional lights no light representation is necessary.
- For the point light draw this sphere (http://web.eng.ucsd.edu/~jschulze/tmp/cone.obj) where the light source is, and in the color of the light that is selected. (2 points)
- For the spotlight draw this cone (http://web.eng.ucsd.edu/~jschulze/tmp/cone.obj) and rotate it so that its round end points in the direction of the light. The cone should also have the color of the spotlight. (3 points)

When a light source is selected, your mouse controls should only affect this particular light source. You need to support the following functions, depending on the type of light source:

Directional Light (2 Points)

- Rotations change the direction of the light

Point Light (3 Points)

- Rotations rotate the light around the origin of the world coordinate system (=center of window).
- Scaling with the mouse wheel changes the distance of the light from the origin of the world coordinate system.
- Light intensity should be visibly different on the model as the light is moved closer and farther from the model surface.

Spotlight (5 Points)

- Should do everything the point light does, plus:
  - 'w'/'W' should make the spot wider/narrower.
  - 'e'/'E' should make the spot edge sharper/blurrier.
  - Make sure the spot keeps pointing at the center of your window coordinate system even when its position is changed.
- The light intensity should be visibly different on the model as the light is moved closer and farther from the model surface, with an even more visible effect than with the point light (it uses linear vs. quadratic attenuation).

Notes

- This video (https://www.youtube.com/watch?v=rTKP50ePS54&feature=youtu.be) demonstrates what is expected (but it does not include normal coloring).
- Learn OpenGL (http://learnopengl.com/#!Lighting/Multiple-lights) provides vertex (http://learnopengl.com/code_viewer.php?code=lighting/lighting_maps&type=vertex) and fragment (http://learnopengl.com/code_viewer.php?code=lighting/multiple_lights&type=fragment) shader code, as well as the corresponding C++ code (http://learnopengl.com/code_viewer.php?code=lighting/multiple_lights-exercise2) for different lighting parameters. The shader does almost exactly what you need. Find out how it differs from the

equations given on the discussion slides for this homework project and make the few modifications.
- Lighthouse 3D also provides excellent tutorials for the necessary shaders: for directional lights (http://www.lighthouse3d.com/tutorials/glsl-tutorial/directional-lights-per-pixel/) , point lights (http://www.lighthouse3d.com/tutorials/glsl-tutorial/point-lights/) and spot lights (http://www.lighthouse3d.com/tutorials/glsl-tutorial/spotlights/) .
- This tutorial (http://www.tomdalling.com/blog/modern-opengl/06-diffuse-point-lighting/) provides very useful information on light parameters in chapters 6 and 7. An additional tutorial on different light types is provided in chapter 8.
- This table of material properties (http://devernay.free.fr/cours/opengl/materials.html) may inspire you to select interesting material parameters. But note that there are specific requirements for the materials you use, which make it necessary to eliminate one or more reflection components (ambient, diffuse, specular) when you define your own materials.

Grading:

- -5 if all that's done is passing the normals
- -10 if all code is there but lighting doesn't work

6. Optional: Mouse Controlled Parameter Editor (10 Points)

For extra credit, create a visual editor to allow the custom modification of the colors and shading parameters of your light source and surface materials. Use the 'c' key to display the color editor. The editor needs to be exclusively mouse controlled. You can either write your own OpenGL code to generate GUI widgets such as buttons or sliders, or you can use an existing open source library for it, such as NanoGUI (https://github.com/wjakob/nanogui) .

You'll get points for the modification of the following parameters:

- The object's colors and other parameters for ambient, diffuse and specular reflectivity. (5 points)
- The light's color and other properties: if using a spot light: cone angle, edge falloff; else: attenuation mode (constant, linear, quadratic). (5 points)