Venkata Chakradhar Reddy Challa

3D Scene Editor

Compiling Instructions:

In order to compile the downloaded source code on your machine, you need to do the following:

1. Install CMAKE
2. Download the source code.
3. Each task has a main file typically name as "main_task<task number>". Rename the main file (of whatever task you want to execute) to "main".

   Note: main_task5 and main pretty much carries all the functionalities developed on the preceding tasks. So, you can just use main/main_task5 to check all the tasks functionalities. If some functionality is not available or doesn't work as expected, please do check the respective task file.

4. Create a directory called build in the downloaded folder directory e.g. by typing in a terminal window: cd TOPDIR; mkdir build
5. Create the necessary makefiles for compilation and place them inside the build/ directory, using the CMAKE GUI (windows), or typing: cd build; cmake ../
6. Compile and run the compiled executable by typing: make; ./ SceneEditor3D_bin

Note: If your machine runs on the latest macOS Mojave, you need to resize the window once to see OpenGL output.

More : https://developer.apple.com/macos/whats-new/,
https://stackoverflow.com/questions/52509427/mac-mojave-opengl

Task1: Scene Editor

Implementation:

1. Created a matrix 'V' to hold vertices of the loaded mesh. V is of dynamic size and is resized without losing previous data whenever there is a request from the user to add an object to the scene through keywords. Created a vector I to hold indices.
2. Addition of objects will be done in Insertion mode which will be enabled by pressing Key 'I'. On launching the application you will enter insertion mode directly. Only after switching back from a different mode you need to press the Key 'I' to enable insertion. Key 1 will insert a cube, Key 2 will insert a bumpy cube and Key 3 will insert a bunny.
3. Respective mesh file from the /data directive will be called and vertices and Indices are loaded into corresponding matrices and vectors.
4. Created VAO and VBO to hold vertex value for the Vertex shaders. VAO and VBO are binded to the program and VBO is updated whenever there is a change in vertices matrix 'V'. Similarly IBO (Element array buffer) is created to hold

indices values. Normals are computed and saved into a dynamic size matrix called 'N' and a VBO named NBO is used to update N values to the shaders.

5. Created "vertex shader" and "fragment shader" to process and draw respective objects in the GPU using the passed on information from CPU through VAO, VBO and uniform variables.
6. Program, VAO and VBO, IBO, NBO are freed at the exit.
7. Camera is positioned at (-1.0, 1.0, 2.0)and targeted at the origin (0.0, 0.0, 0.0). With a worldup value (0.0, 1.0, 0.0)a look_at matrix is computed which is referred as view.
8. A light source is positioned at (0.0, 1.0, 2.0) with white (1.0, 1.0, 1.0) as light color
9. Normals are used in the shaders to computer fragment color with respect to the light source. Ambient, Diffuse, and Specular light shading are applied with the request form the user. By default, Ambient and Diffuse are used. Specular is also applied but it will be discussed in the task2.
10. For saving lot of bandwidth and to make process faster the following are used:
    a. glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    b. glEnable(GL_DEPTH_TEST);
    c. glEnable(GL_CULL_FACE);
11. **Vertices are loaded only the first time the user requests a different object. And later, transformation matrices are computed/used to construct multiple instances of these objects. So, in the current case 3 objects (Cube, bumpy cube, and bunny) are loaded into the vertices buffer for the first time upon corresponding object request and will be reused for later instances. This saves a lot of bandwidth on the shader and makes editor faster for the user.**

Issues:

Index buffer is acting weird for some unknown reasons in a few scenarios. Currently it can create multiple similar objects but finding it difficult to draw a different object. Cube has no issues in drawing multiple similar cubes and then multiple bunny or multiple bumpy cube. But after drawing bumpy cubes or bunnies it is finding difficult to perfectly render a different one.

Results:

1. Addition of objects will be done in Insertion mode which will be enabled by pressing Key 'I'. On launching the application you will enter insertion mode directly. Only after switching back from a different mode you need to press the Key 'I' to enable insertion. Key 1 will insert a cube, Key 2 will insert a bumpy cube and Key 3 will insert a bunny.
2. All objects are sized to fit the unit cube and centered at the origin. A random translation is provided to better visualize different objects/instances of the same object.

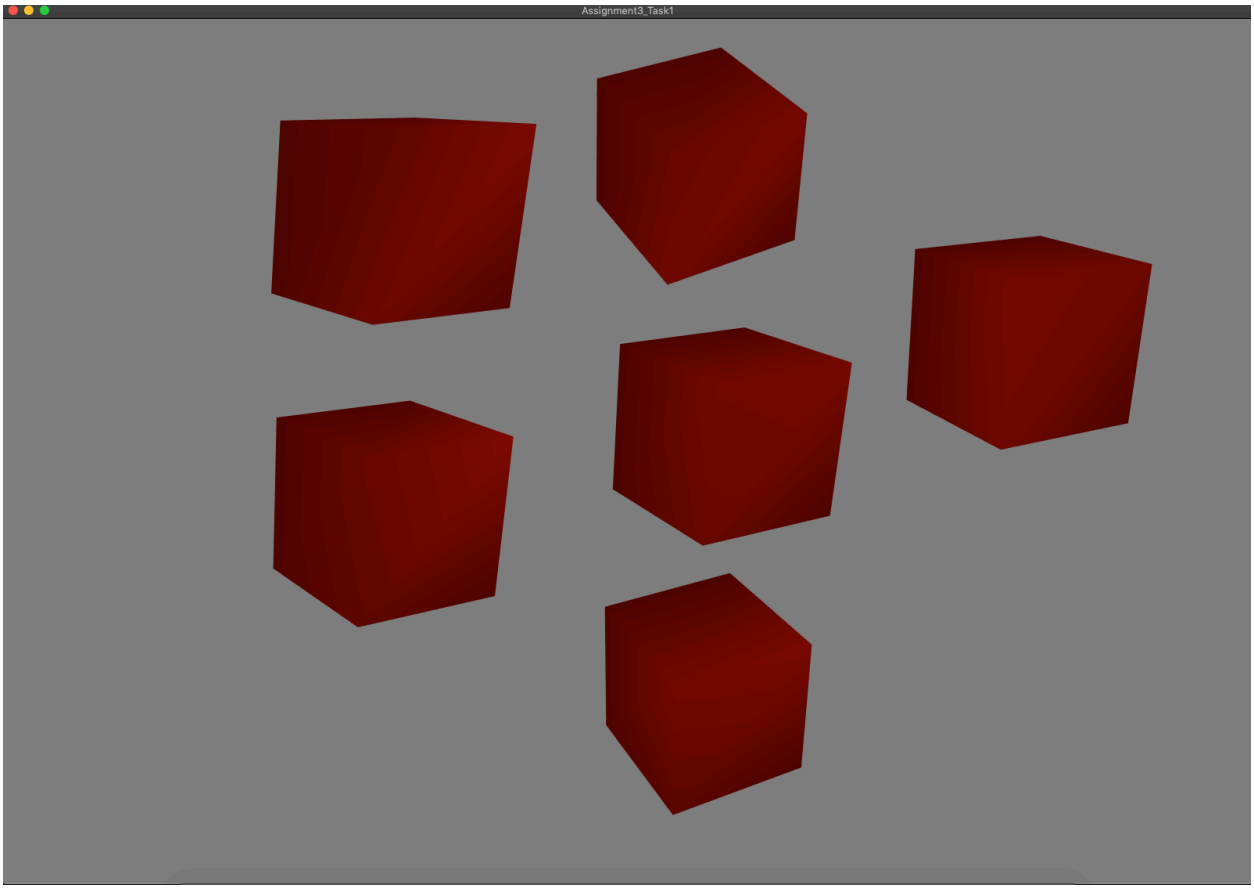

Figure 1: Multiple cubes are inserted at random locations around the origin

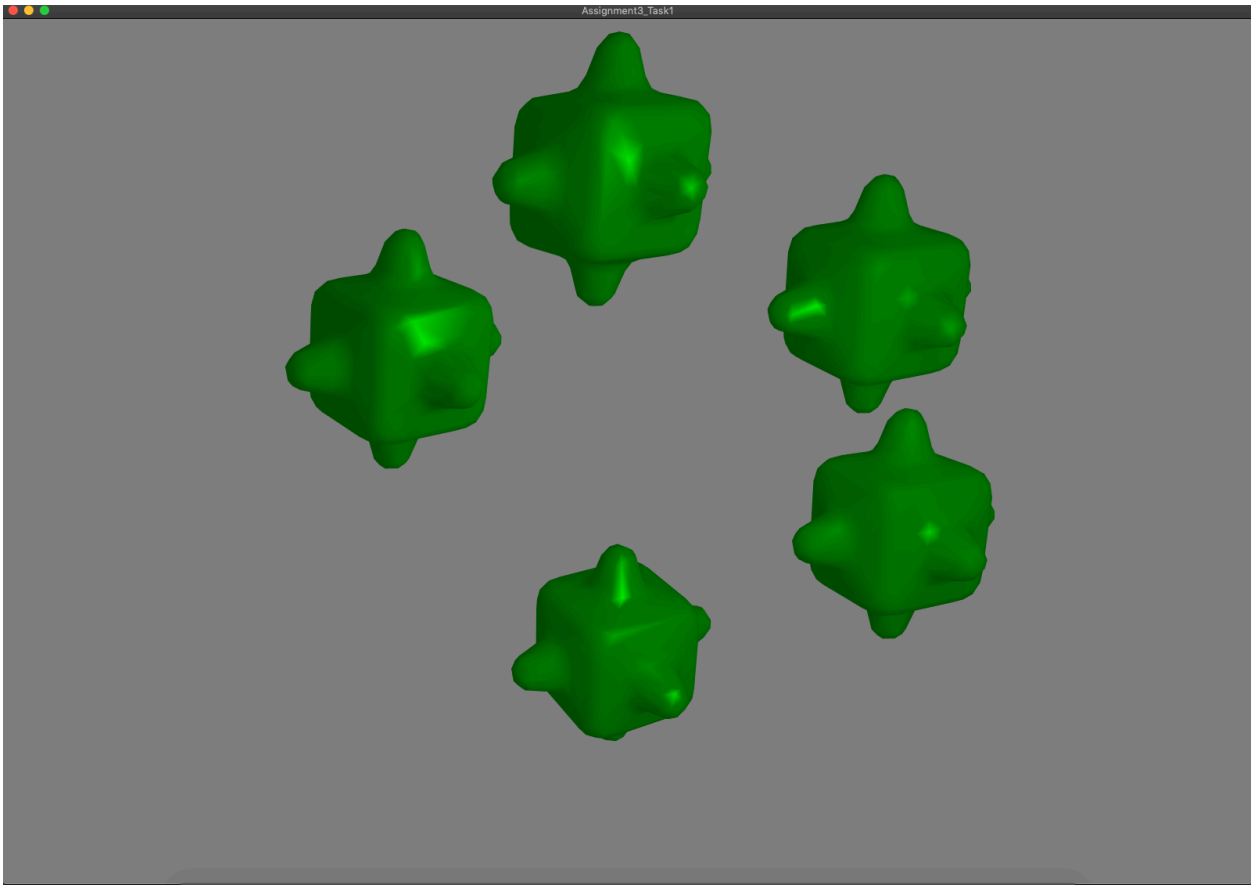

Figure 2: Multiple bumpy cubes are inserted at random locations around the origin

Figure 3: Multiple bunnies are inserted at random locations around the origin

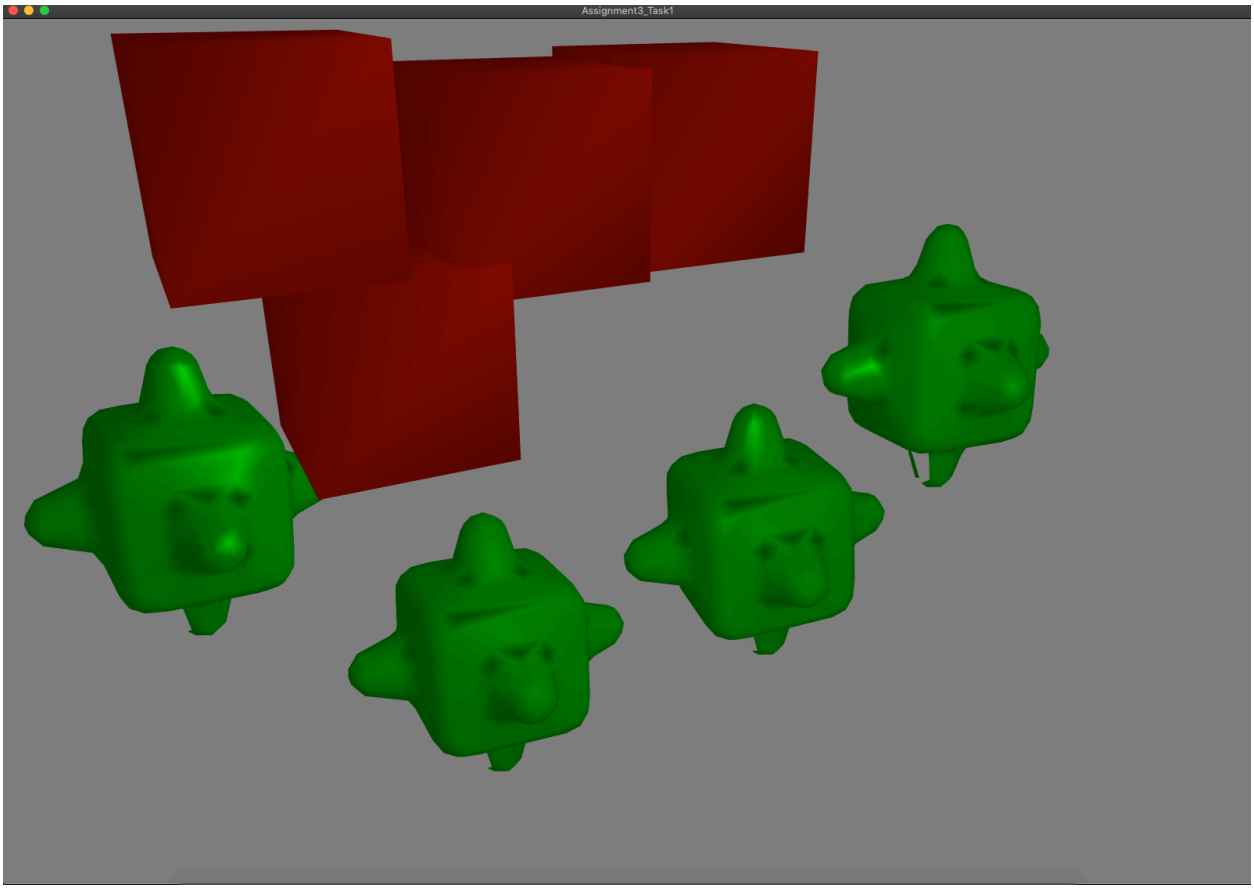

Figure 4: Cubes and bumpy cubes are inserted at random locations around the origin

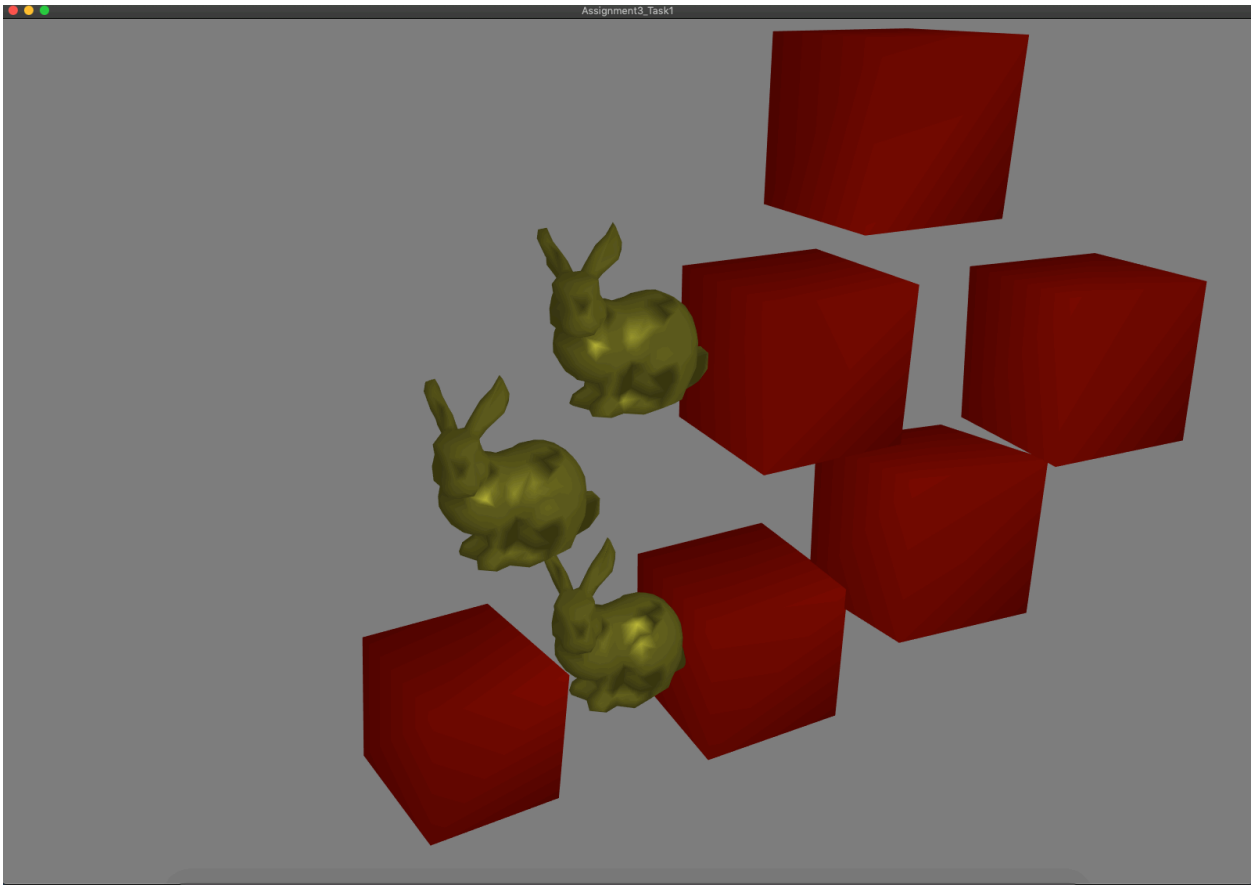

Figure 5: Cubes and bunnies are inserted at random locations around the origin

Task2: Object Control

Implementation:

1. You need to press Key ‘O’ to enable editing mode. Most of the steps are more similar to the Rasterization project.
2. One mouse click on the object will select the object in the scene. Click one time then hold and then drag to move object around in the scene. One click will select the object press any Key between 1-9 to apply a different color to the object. Once the object is selected Key Z and X can be used to SCALE up and down the selected object. Key R and T can be used to Rotate the object Clockwise and Counter clock wise respectively.
3. Use Key W to enable wireframe, Key F to apply flat shading and Key P to use Phong Shading.
4. Normals are computed as said in the instuctions.
5. glReadPixels is used to find the objected selected by the user.
6. Stencil buffer is used for the above purpose.
    a. glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);
    b. glEnable(GL_STENCIL_TEST);
    c. glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);

Results:

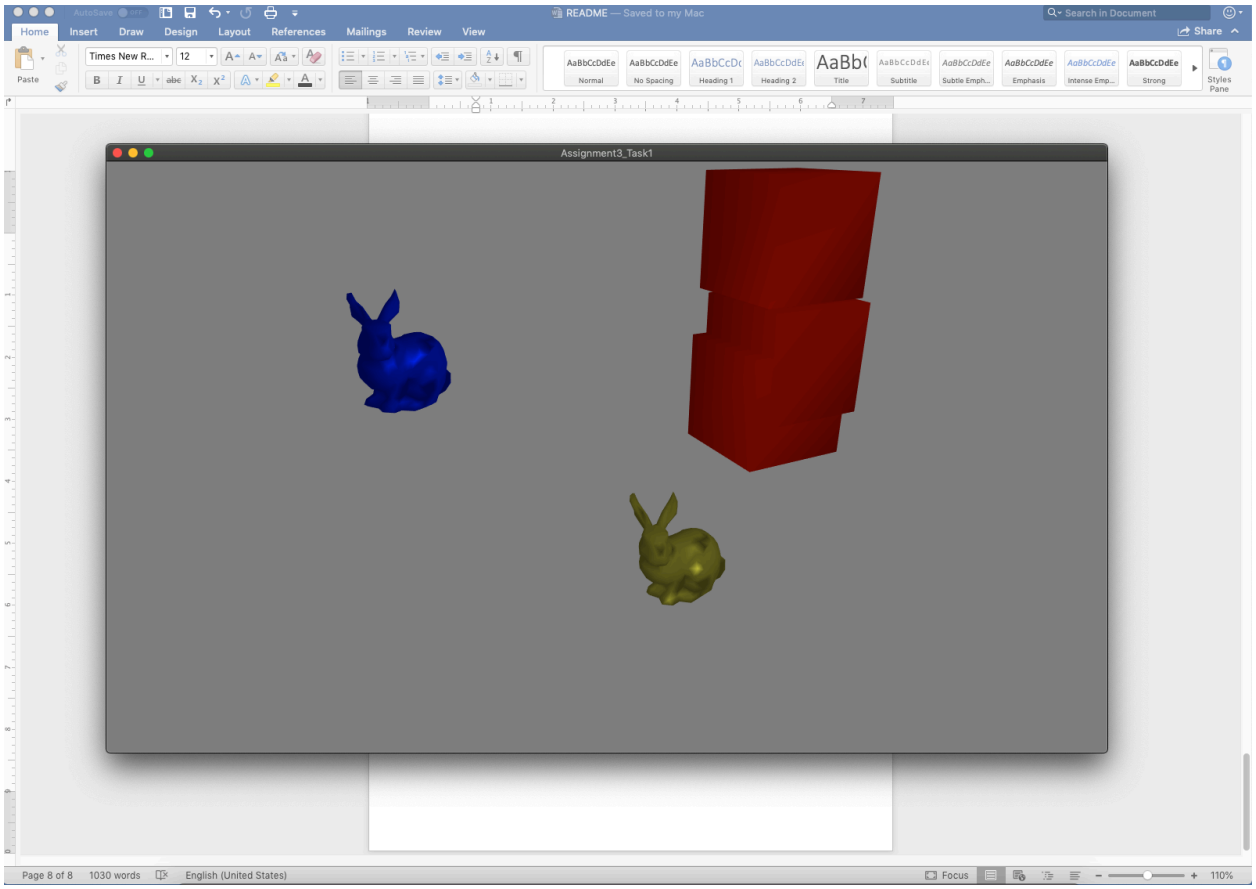

Figure 6: A Bunny is selected for transformations by clicking on top of it

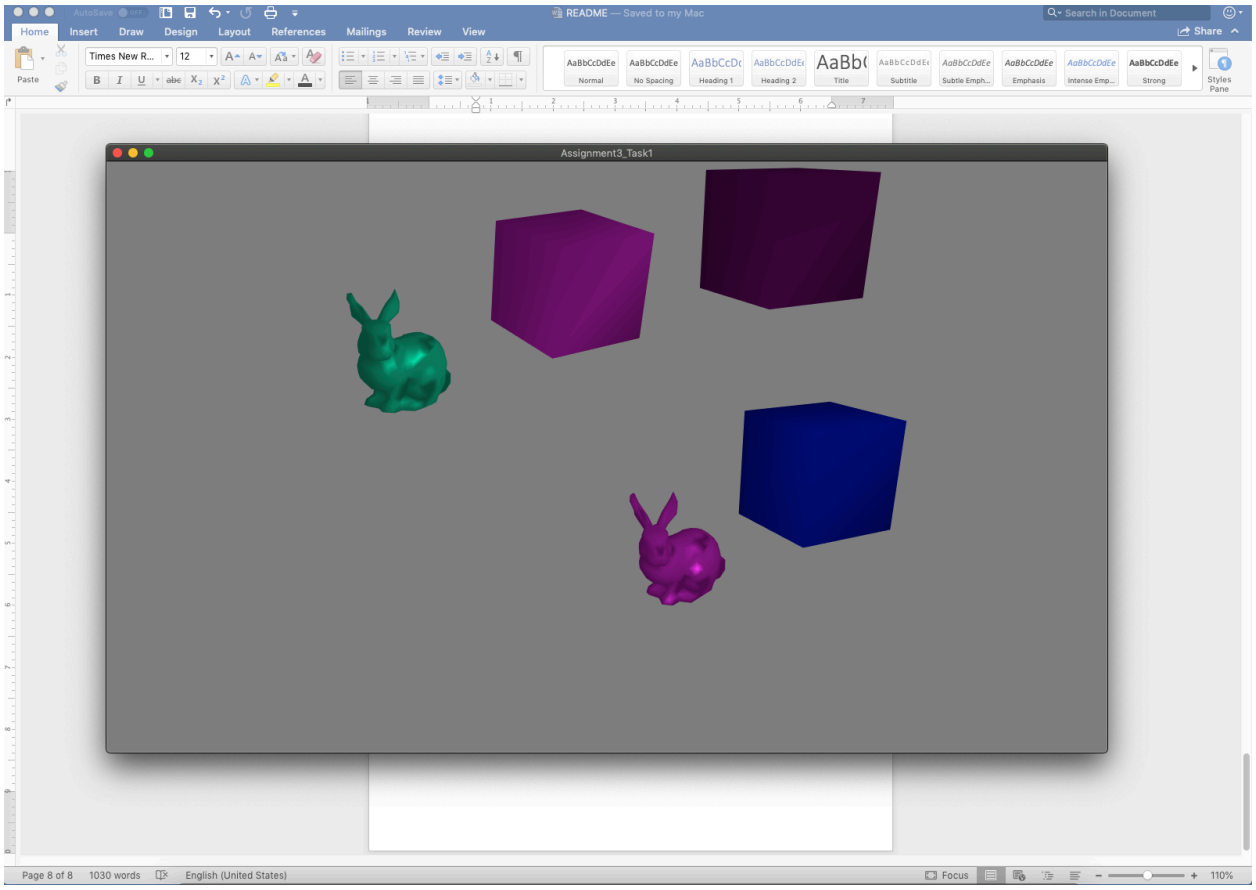

Figure 7: Color modifications are done to the objects by selecting them with a mouse click

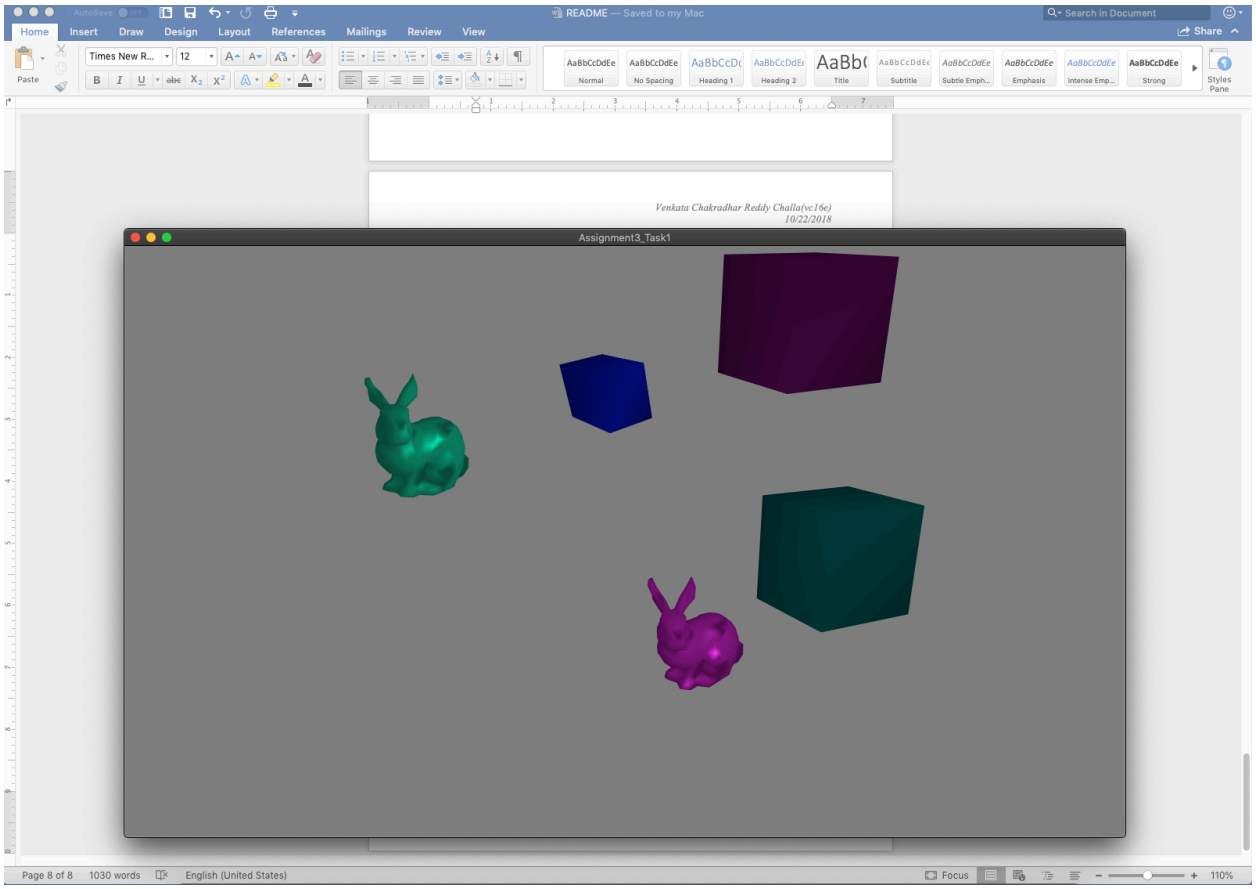

Figure 8: Scale and rotations can be applied by selecting the objects

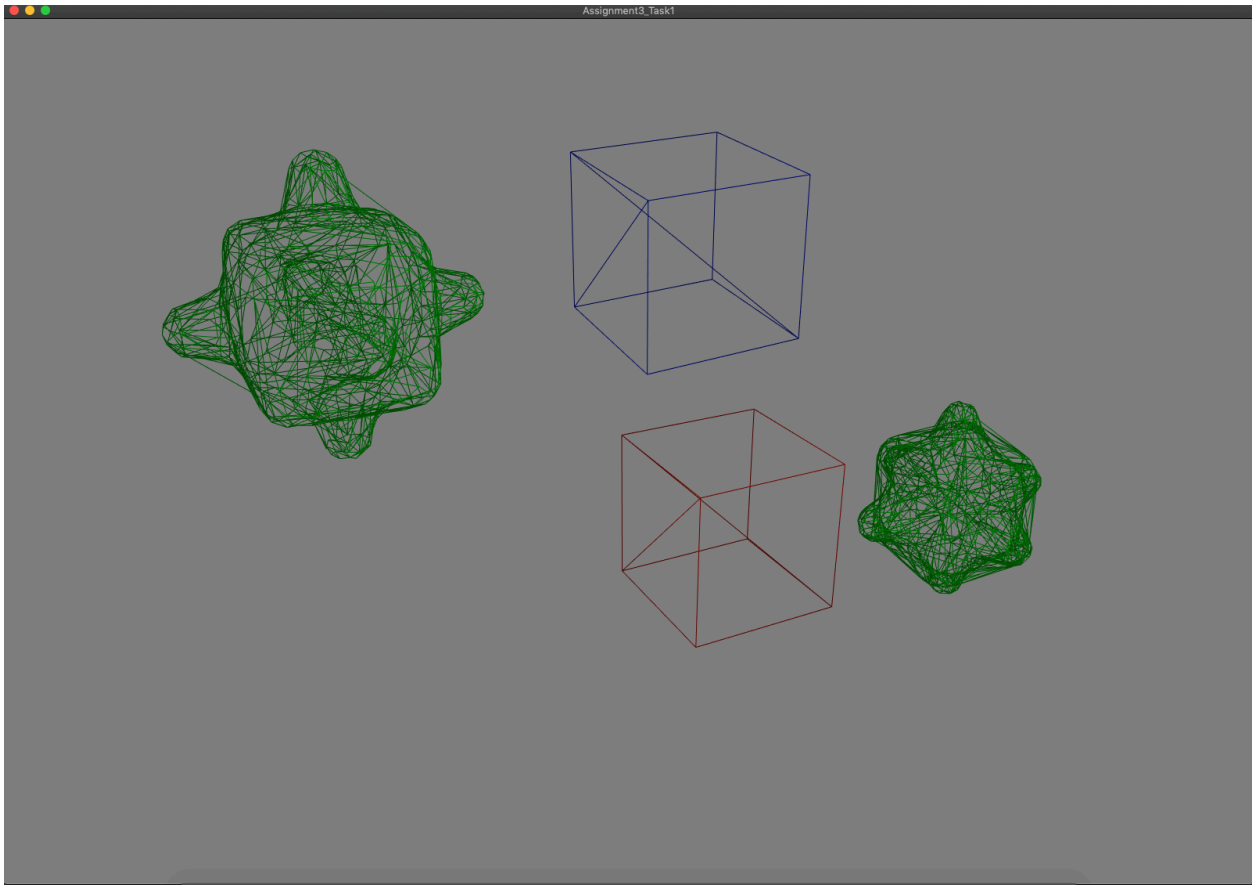

Figure 9: Wire frame mode is enabled, Since GL_LINE_LOOP is used with Index buffer the outline is not perfect

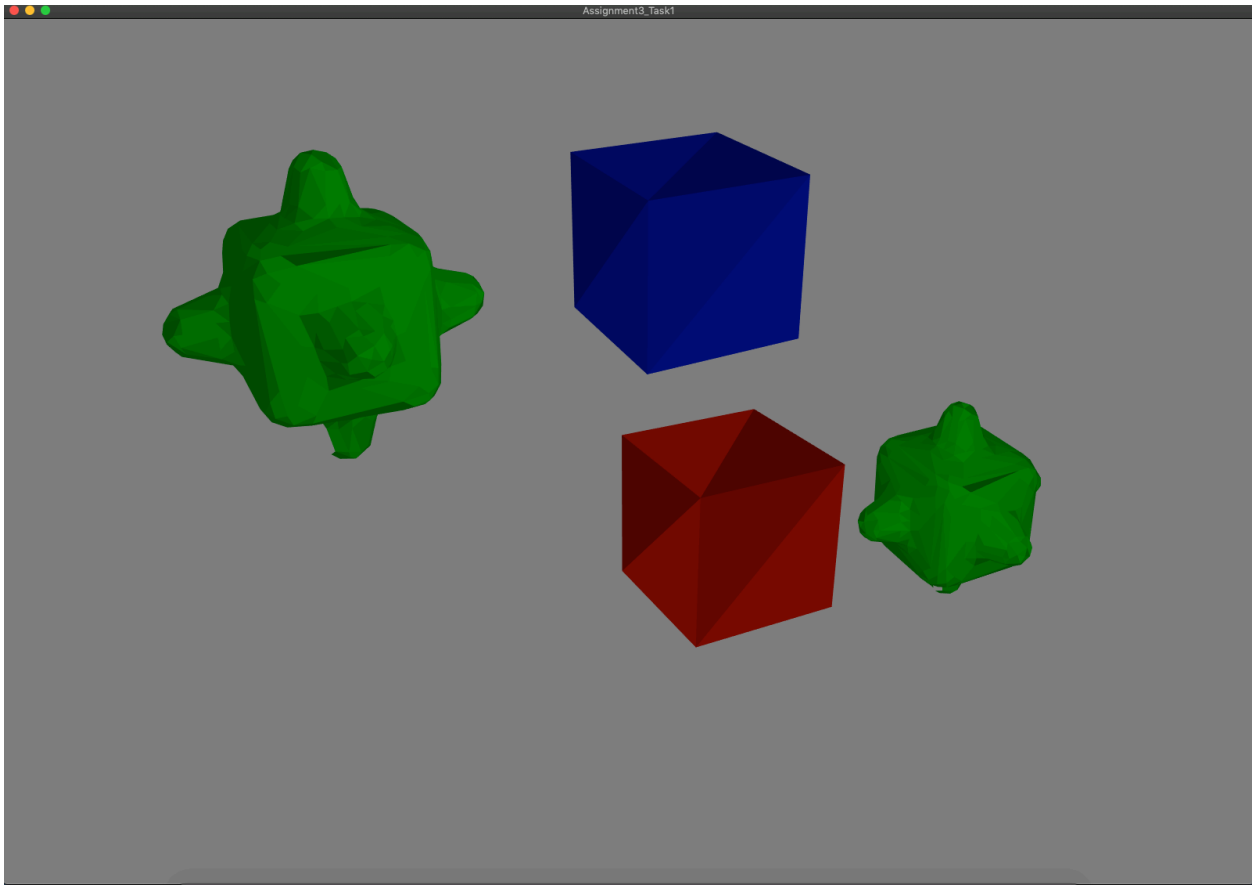

Figure 10: Flat shading is enabled

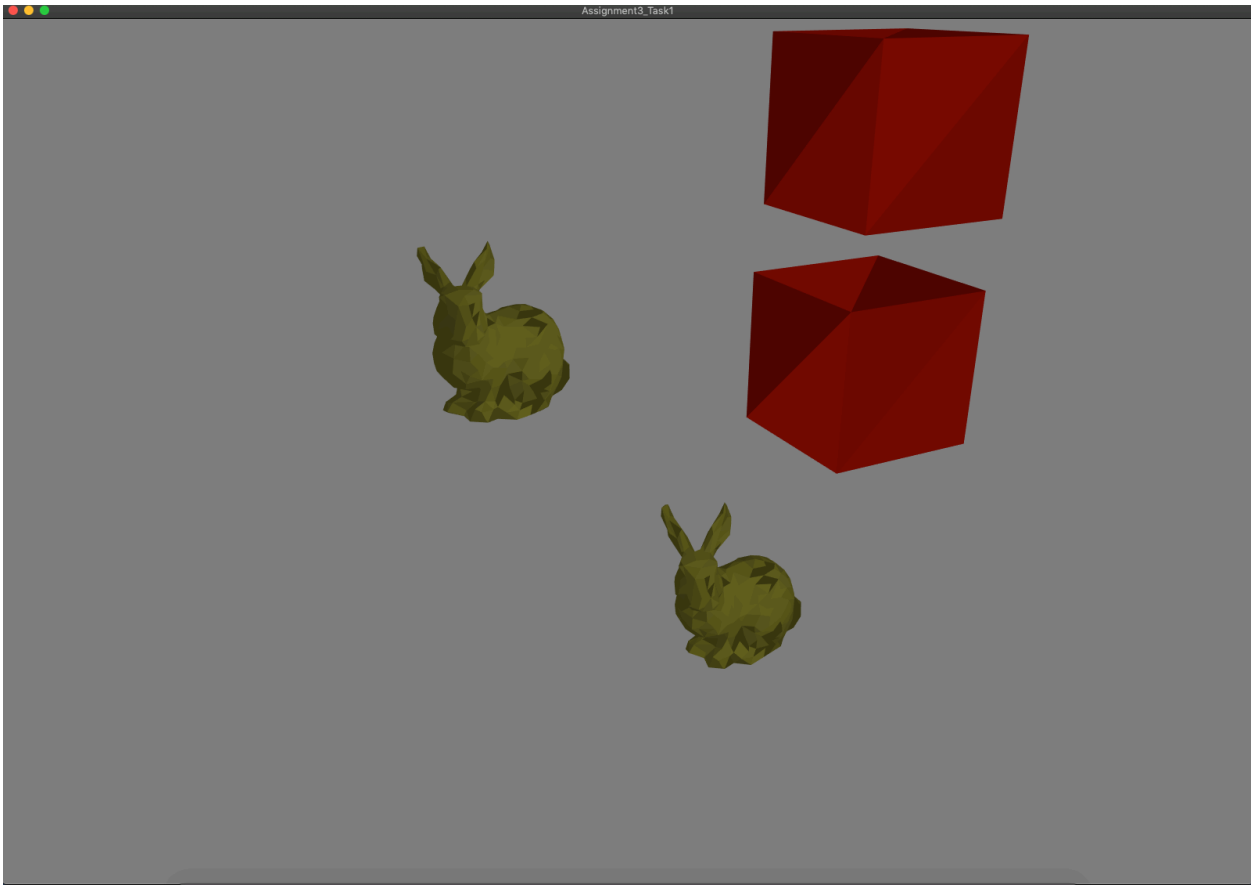

Figure 11: Flat shading is enabled

Figure 12: By default, Phong shading is enabled

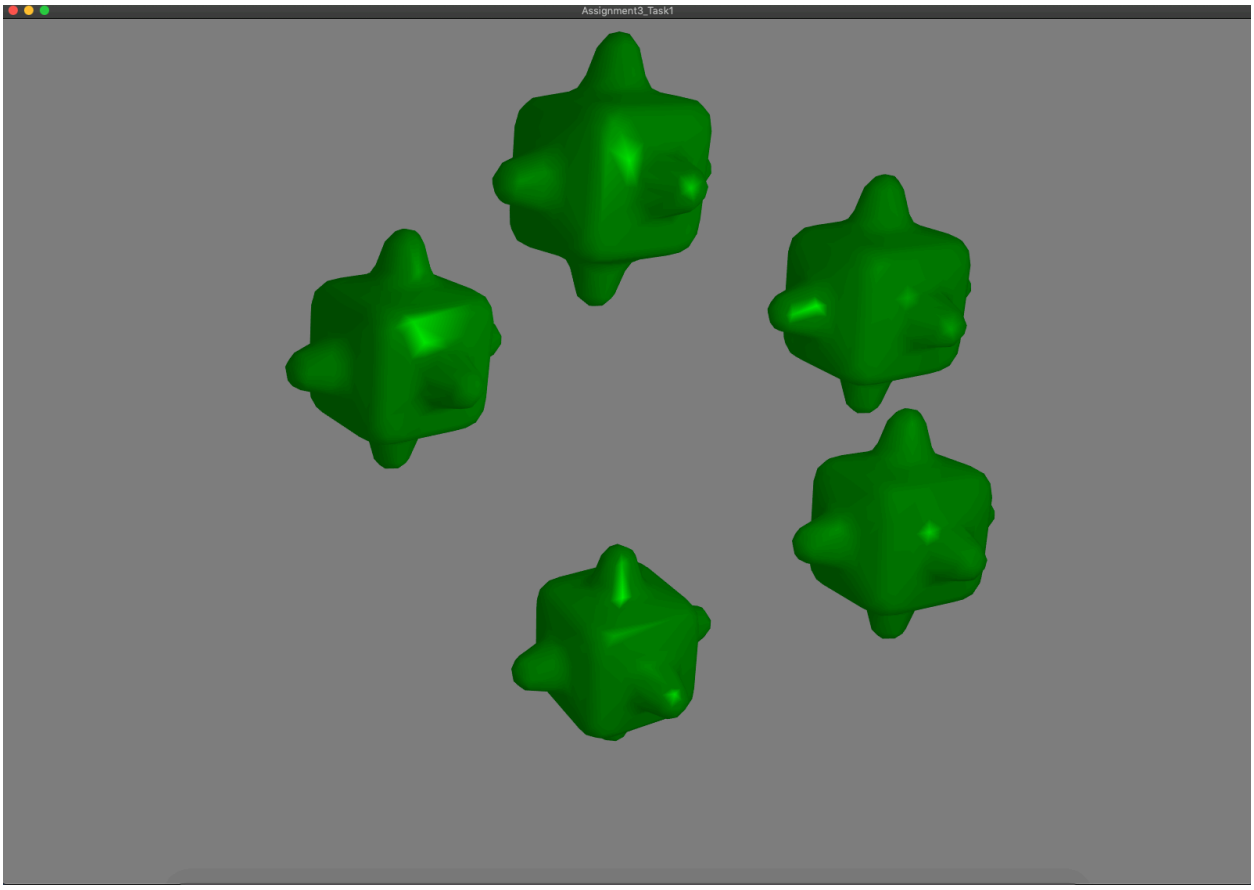

Figure 13: By default, Phong shading is enabled

Task3: Camera control

Implementation:

1. Use Arrow keys to move the camera. Left and Right arrow keys will move the camera left and right respectively and Up and Down arrow keys will move the camera to the top and bottom of the scene.
2. Use key K to enable Perspective projection and key L to enable Orthographic projection.

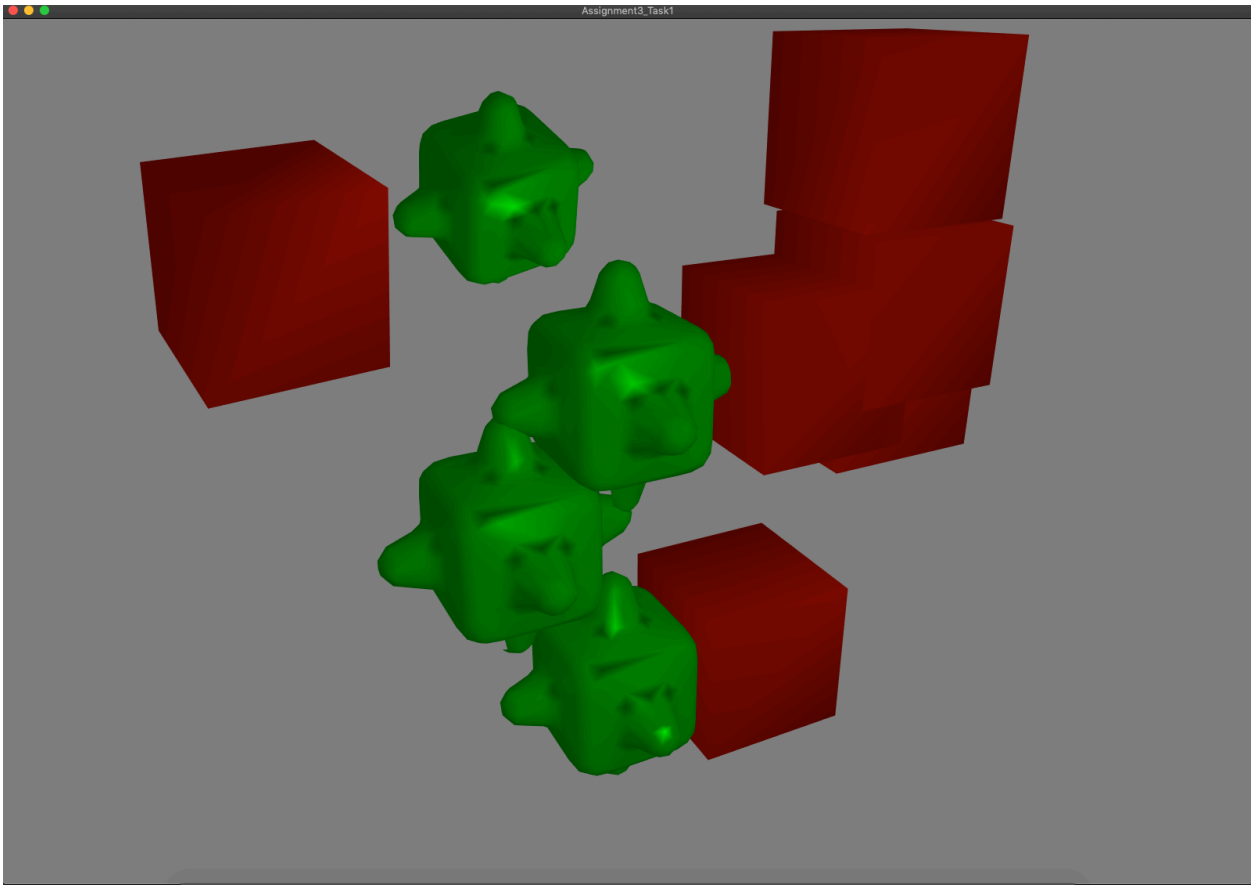

Figure 14: Left arrow is used to move the camera to the left

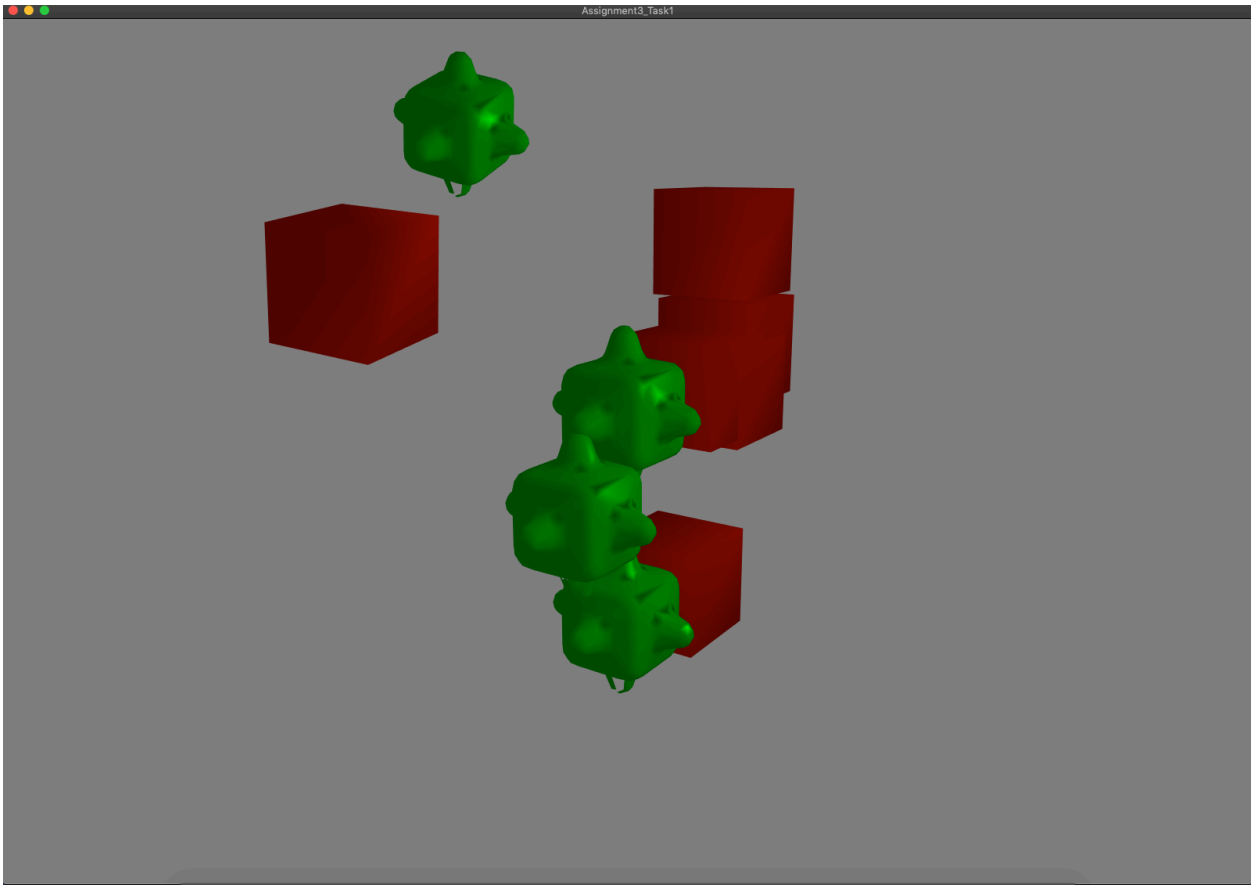

Figure 15: Left arrow is used few more times to further move the camera to the left

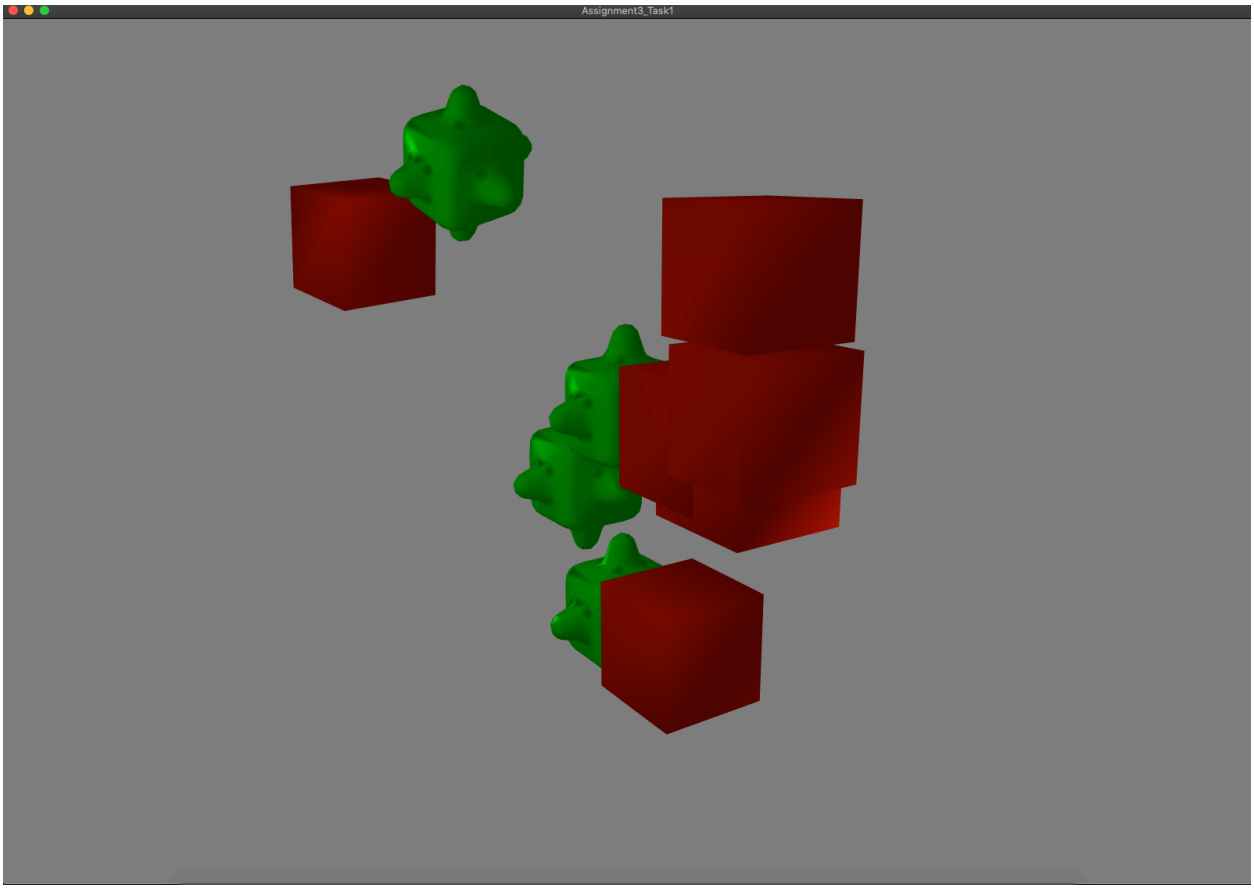

Figure 16: Right arrow is used to move the camera to the right from the original position

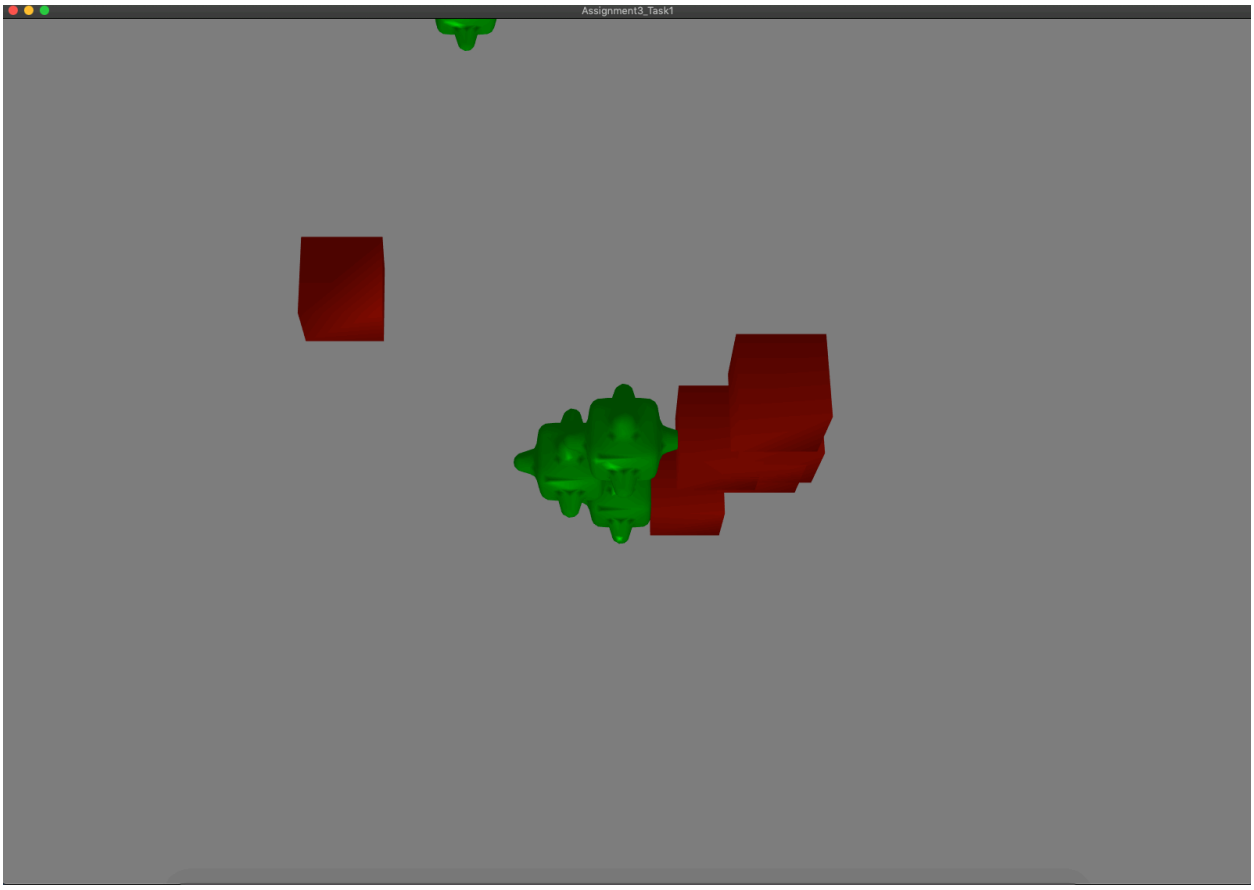

Figure 17: Up arrow is used to move the camera to the top view

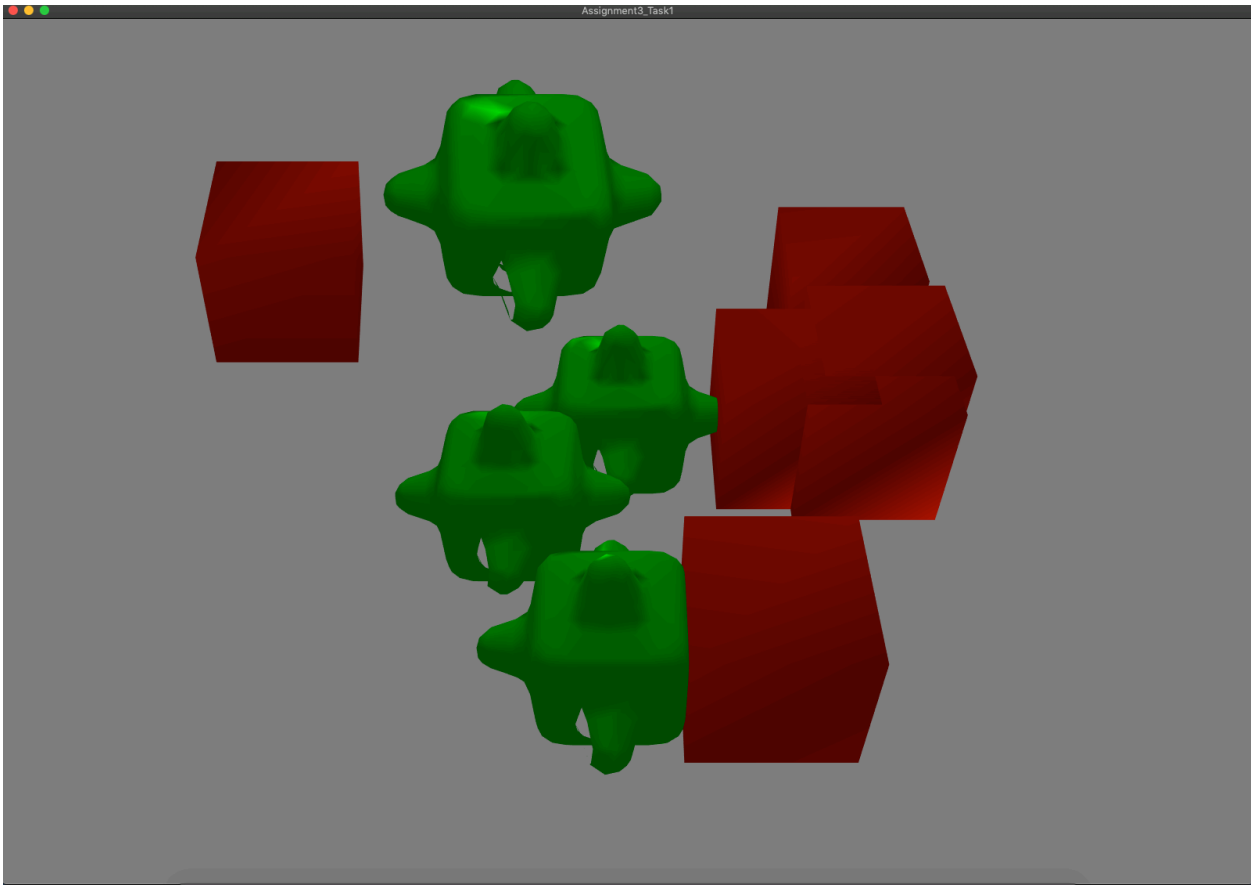

Figure 18: Down arrow is used to move the camera to the bottom view

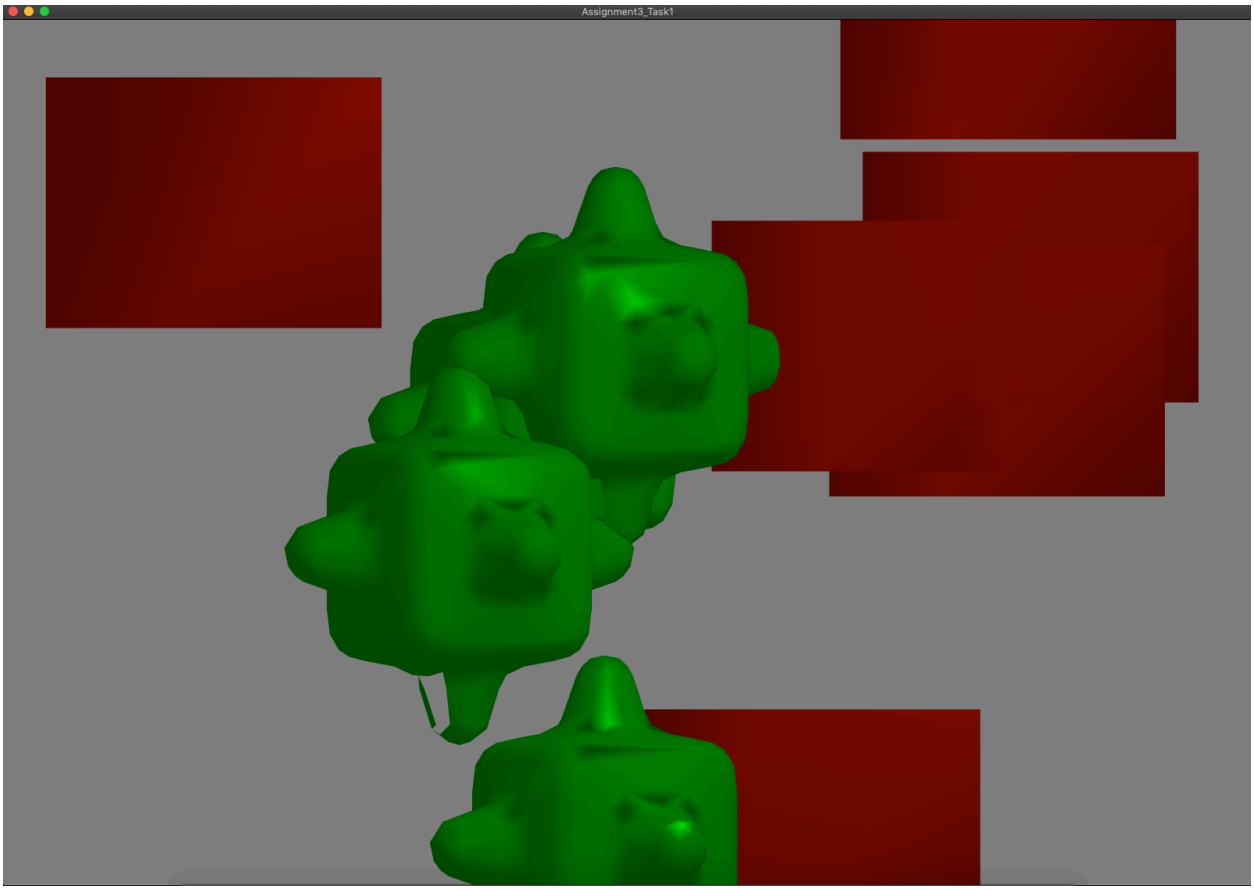

Figure 19: Orthographic projection/camera view

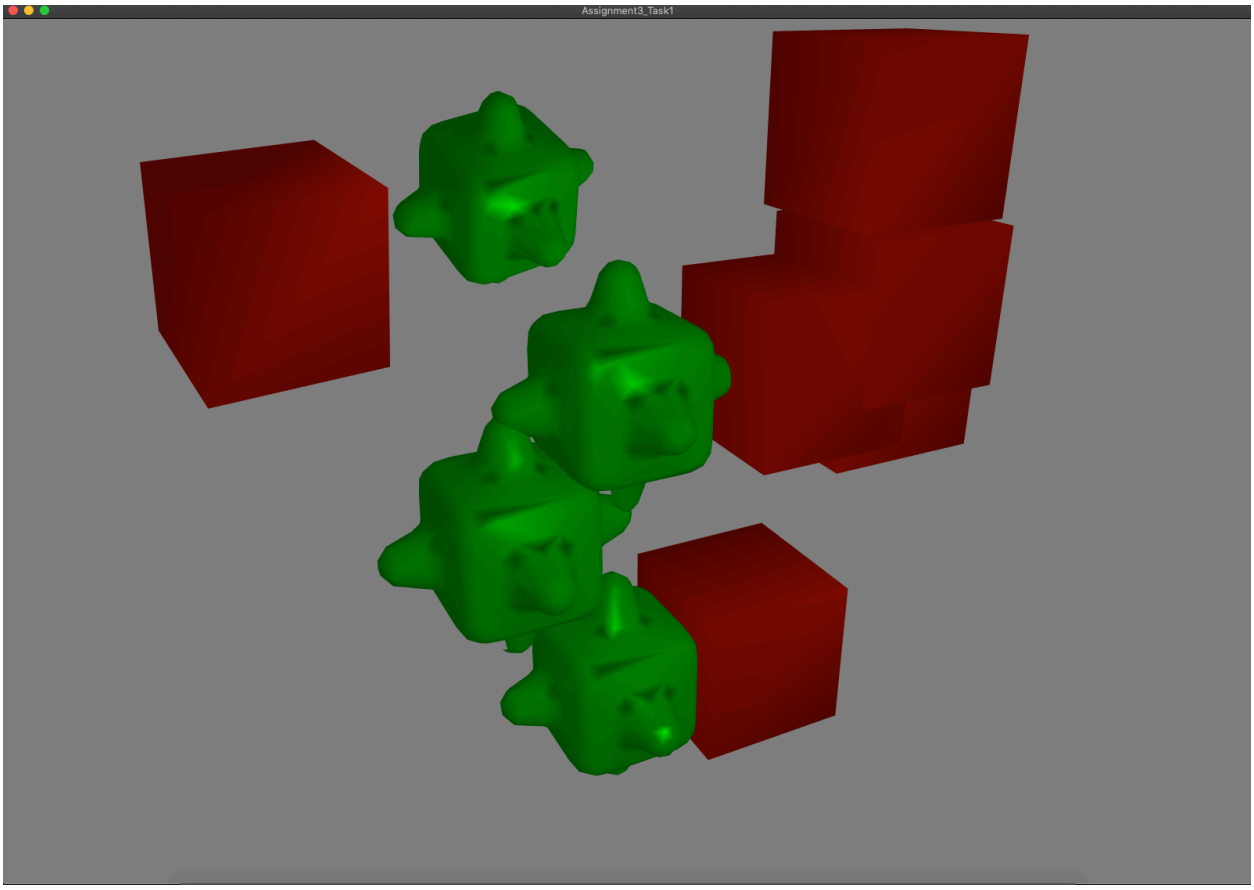

Figure 20: By default, the application is in the Perspective projection/camera view