CS6060- Test 1

Max Thrun

Fall 2013

Part A

All components in my model are based off a unit cube which is shown below. To achieve each shape I simply scale the cube to the appropriate size. The cube extends from `(0,0,0)` to `(1,1,1)`. A shader is used to draw a grid over the cube with a spacing of 1 unit which allows us to visualize the size of the object more easily. Note that while it *looks* like multiple cubes are being drawn for each object it is just a single stretched cube. Each object is represented by a different color to help distinguish them from each other. Note that *all* objects are represented by cubes including the lens and baseboard hinge. This was done so that we only have to create the vertices for a cube and was deemed appropriate as this assignment is more about demonstrating the matrices required to build our model. The matrices would remain exactly the same had we used proper shapes for all components with the only difference being in the vertex buffer.

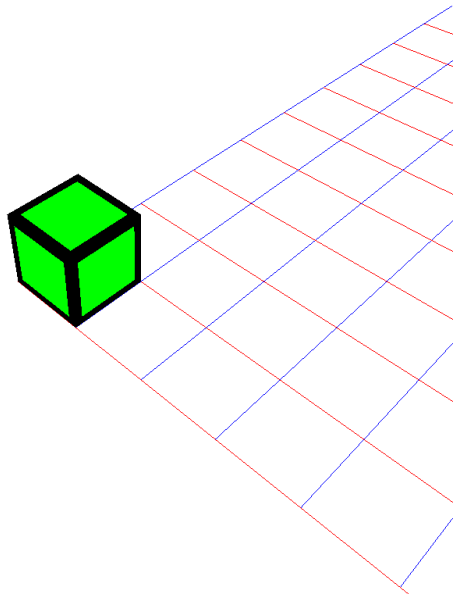

Figure 1: Unit Cube

The general format for each operation in the following sections is as follows:

$$component\_space\_matrix = translation * scale * identity$$

Note that multiplication is performed right to left in order to ensure that we scale before we translate or rotate.

The following sections show the matrices for each component in their respective object coordinate spaces.

Camera

The matrices required to assemble the camera in its local object space are shown below.

$$cam\_hing\_mcs = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$cam\_body\_mcs = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & -1.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$cam\_lens\_mcs = \begin{bmatrix} 1 & 0 & 0 & 1.5 \\ 0 & 1 & 0 & -4 \\ 0 & 0 & 1 & -0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The camera assembly in its local object space is visualized below.

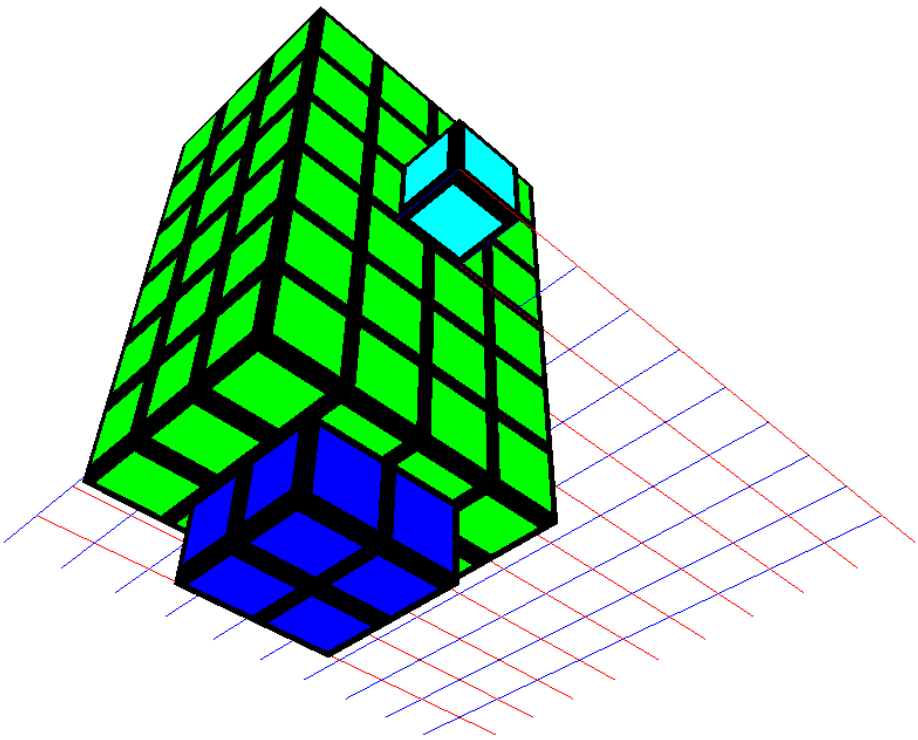

Figure 2: Camera Assembly in MCS space (viewing from bottom)

Shaft

The matrices required to assemble the shaft in its local object space are shown below.

$$shaft\_hing\_scs = \begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 15 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shaft\_body\_scs = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The shaft assembly in its local object space is visualized below.

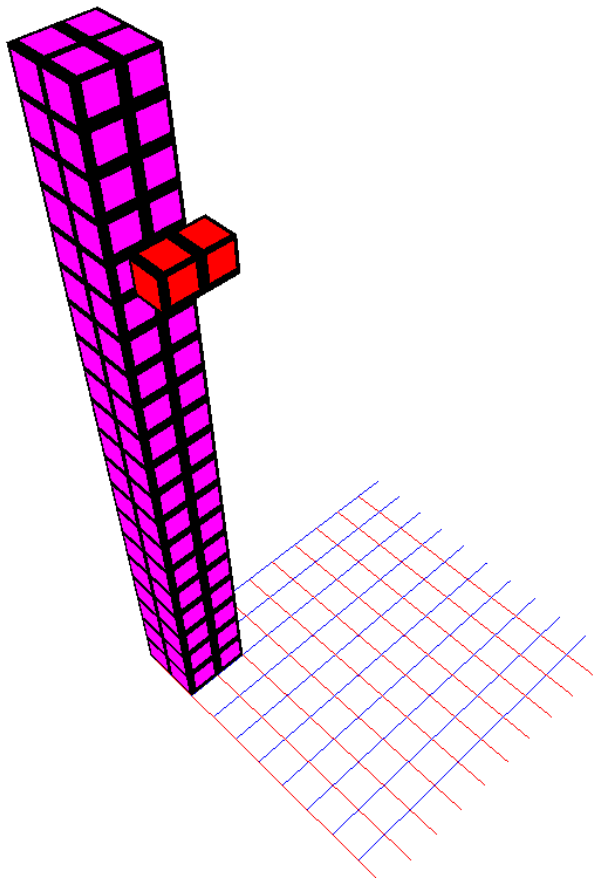

Figure 3: Shaft Assembly in SCS space

Baseboard

The matrices required to assemble the baseboard in its local object space are shown below.

$$base\_hing\_bcs = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$base\_body\_bcs = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 12 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The baseboard assembly in its local object space is visualized below.

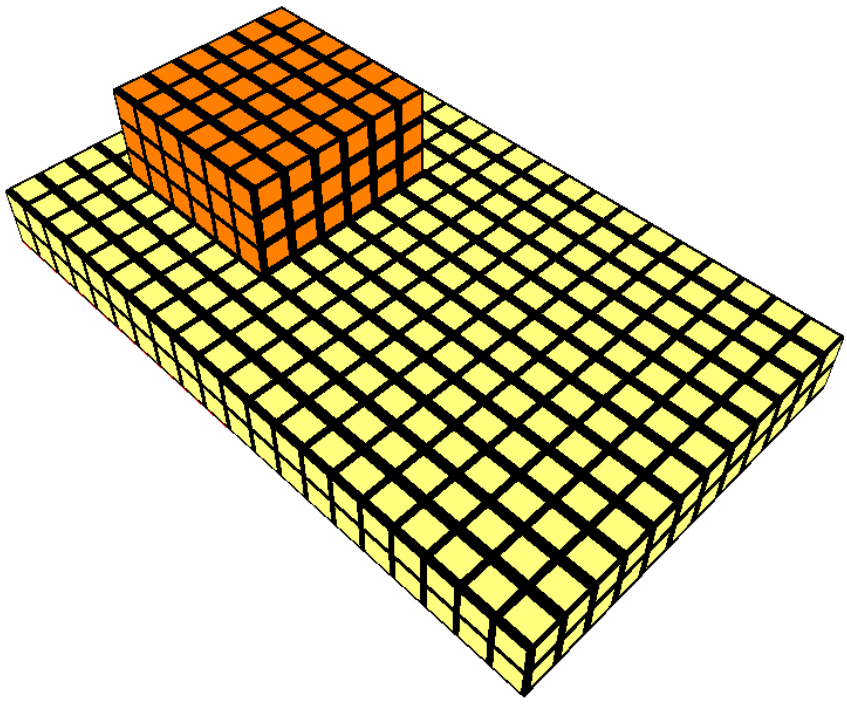

Figure 4: Base Assembly in BCS space

Desk

The matrices required to assemble the desk in its local object space are shown below.

$$desk\_top = \begin{bmatrix}1&0&0&0\\0&1&0&28\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}30&0&0&0\\0&2&0&0\\0&0&48&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$desk\_leg\_1 = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}4&0&0&0\\0&28&0&0\\0&0&4&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$desk\_leg\_2 = \begin{bmatrix}1&0&0&26\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}4&0&0&0\\0&28&0&0\\0&0&4&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$desk\_leg\_3 = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&44\\0&0&0&1\end{bmatrix}\begin{bmatrix}4&0&0&0\\0&28&0&0\\0&0&4&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$desk\_leg\_4 = \begin{bmatrix}1&0&0&26\\0&1&0&0\\0&0&1&44\\0&0&0&1\end{bmatrix}\begin{bmatrix}4&0&0&0\\0&28&0&0\\0&0&4&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

The desk assembly in its local object space is visualized below.

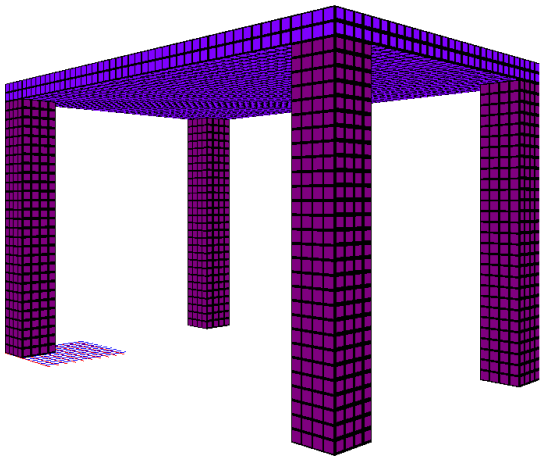

Figure 5: Desk Assembly in DCS space

Part B

The matrices required to assemble the doc-cam in its local object space are shown below.

$$base\_ocs = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$base\_hing\_ocs = base\_ocs * base\_hing\_bcs$$

$$base\_body\_ocs = base\_ocs * base\_body\_bcs$$

$$shaft\_ocs = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} * base\_hing\_ocs * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shaft\_hing\_ocs = shaft\_ocs * shaft\_hing\_scs$$

$$shaft\_body\_ocs = shaft\_ocs * shaft\_hing\_scs$$

$$cam\_ocs = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * shaft\_hing\_ocs * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$cam\_hing\_ocs = cam\_ocs * cam\_hing\_mcs$$

$$cam\_body\_ocs = cam\_ocs * cam\_body\_mcs$$

$$cam\_lens\_ocs = cam\_ocs * cam\_lens\_mcs$$

The doc-cam assembly in its local object space is visualized below.

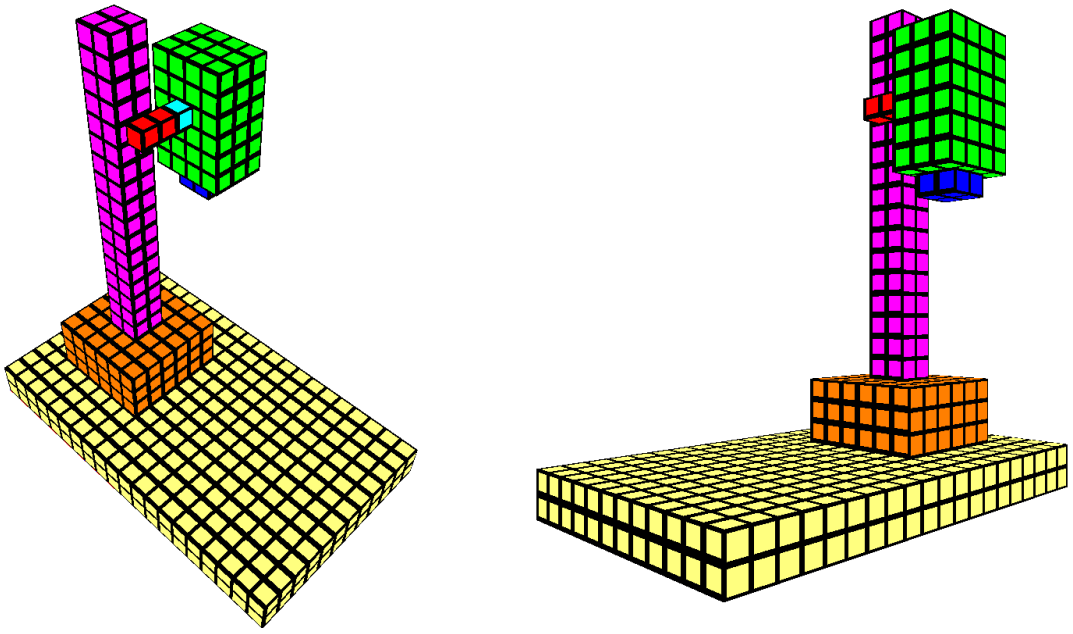

Figure 6: Doc-cam Assembly in OCS space

Part C

The matrices required to assemble the complete model in world space are shown below.

$$wcs = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$rcs\_wcs = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * wcs$$

$$dcs\_wcs = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * rcs\_wcs$$

$$ocs\_wcs = \begin{bmatrix} 1 & 0 & 0 & 9 \\ 0 & 1 & 0 & 30 \\ 0 & 0 & 1 & 16 \\ 0 & 0 & 0 & 1 \end{bmatrix} * dcs\_wcs$$

$$desk\_top\_wcs = dcs\_wcs * desk\_top\_dcs$$

$$desk\_leg\_[1-4] = dcs\_wcs * desk\_leg\_[1-4]\_dcs$$

$$base\_hing\_wcs = ocs\_wcs * base\_hing\_ocs$$

$$base\_body\_wcs = ocs\_wcs * base\_body\_ocs$$

$$shaft\_hing\_wcs = ocs\_wcs * shaft\_hing\_ocs$$

$$shaft\_body\_wcs = ocs\_wcs * shaft\_body\_ocs$$

$$cam\_hing\_wcs = ocs\_wcs * cam\_hing\_ocs$$

$$cam\_body\_wcs = ocs\_wcs * cam\_body\_ocs$$

$$cam\_lens\_wcs = ocs\_wcs * cam\_lens\_ocs$$

The completed assembly in world space is visualized below.

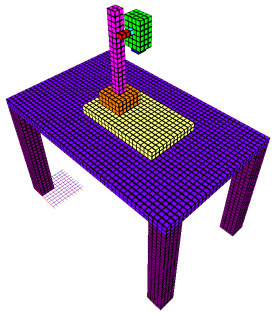

Figure 7: Full Assembly in WCS space

Using a simple shader which darkens each face depending on the direction of its normal we can achieve a simple ‘smoothed’ version which is shown below:

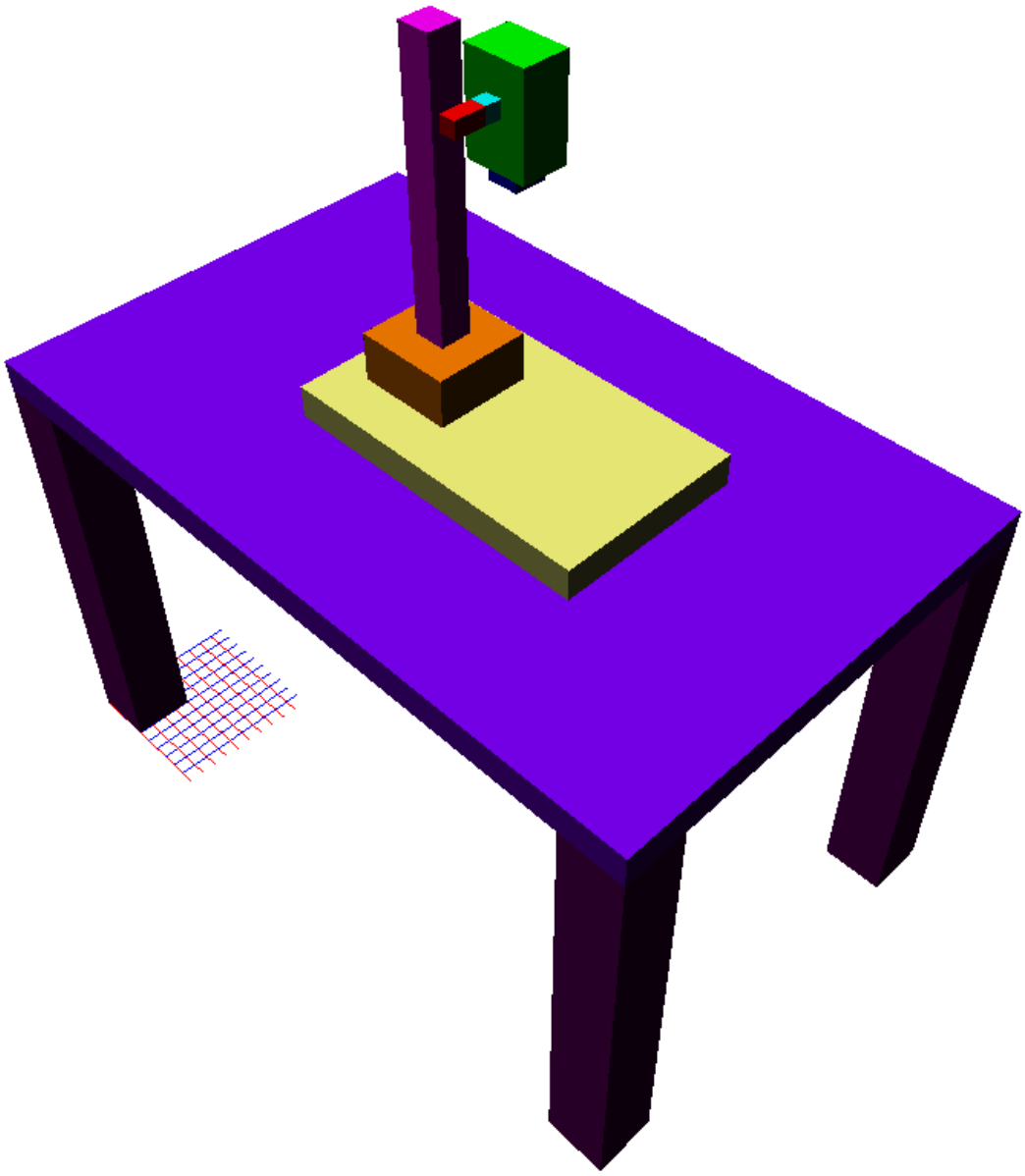

Figure 8: Full Assembly in WCS space (smoothed)

In order to tilt the shaft and camera we simply modify the matrices for their respective coordinate systems to include a rotation matrix. The matrices shown below are the same as those shown in Part B but with the inclusion of a rotation matrix.

$$shaft\_ocs = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} * base\_hing\_ocs * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 30^\circ & -\sin 30^\circ & 0 \\ 0 & \sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$cam\_ocs = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * shaft\_hing\_ocs * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos -45^\circ & -\sin -45^\circ & 0 \\ 0 & \sin -45^\circ & \cos -45^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

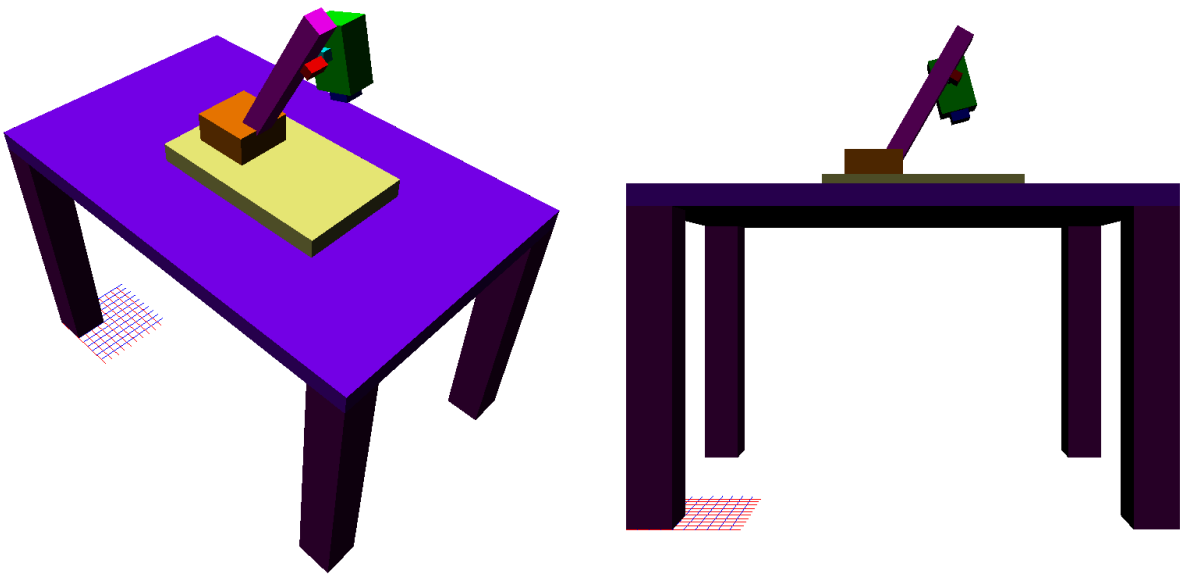

Figure 9: Doc-cam Assembly in WCS space with rotated shaft and camera

Part D

The full source code used to implement this project is shown below. The matrix definition and operations start on line 128.

```
1   #define GLEW_STATIC
2   #include <vector>
3   #include <GL/glew.h>
4   #include <SFML/Graphics.hpp>
5   #include <SFML/OpenGL.hpp>
6   #include <iostream>
7   #include <ctime>
8   #include <glm/glm.hpp>
9   #include <glm/gtc/matrix_transform.hpp>
10  #include <glm/gtx/rotate_vector.hpp>
11  #include <glm/gtc/type_ptr.hpp>
12  #include "shader.h"
13
14  GLint cube_model_id = -1;
15  GLint cube_view_id = -1;
16  GLint cube_proj_id = -1;
17  GLint cube_color_id =-1;
18
19  glm::mat4 view;
20  glm::mat4 projection;
21
22  void draw_cube(glm::mat4 cube_model, glm::vec3 color)
23  {
24      glUniformMatrix4fv(cube_model_id, 1, GL_FALSE, glm::value_ptr(cube_model));
25      glUniformMatrix4fv(cube_view_id, 1, GL_FALSE, glm::value_ptr(view));
26      glUniformMatrix4fv(cube_proj_id, 1, GL_FALSE, glm::value_ptr(projection));
27      glUniform3f(cube_color_id, color.x, color.y, color.z);
28      glDrawArrays(GL_QUADS, 0, 24);
29  }
30
31  int main()
32  {
33      sf::ContextSettings contextSettings;
34      contextSettings.depthBits = 32;
35      contextSettings.antialiasingLevel = 8;
36
37      sf::RenderWindow window(sf::VideoMode(800, 800), "Test 1", sf::Style::Default, contextSettings);
38      window.setVerticalSyncEnabled(true);
39
40      window.setActive();
41
42      glewExperimental = GL_TRUE;
43      glewInit();
44
45      //
46      // Setup the grid
47      //
48
49      #define GRID_SIZE   10
50
51      std::vector<float> default_verts;
52
53      for(int k = 0; k<GRID_SIZE; k++) {
54          // start of x line / color / end of x line / color
55          default_verts.push_back(k); default_verts.push_back(0); default_verts.push_back(0);
56          default_verts.push_back(1); default_verts.push_back(0); default_verts.push_back(0);
57          default_verts.push_back(k); default_verts.push_back(0); default_verts.push_back(GRID_SIZE);
58          default_verts.push_back(1); default_verts.push_back(0); default_verts.push_back(0);
59          // start of z line / color / end of z line / color
60          default_verts.push_back(0); default_verts.push_back(0); default_verts.push_back(k);
61          default_verts.push_back(0); default_verts.push_back(0); default_verts.push_back(1);
62          default_verts.push_back(GRID_SIZE); default_verts.push_back(0); default_verts.push_back(k);
63          default_verts.push_back(0); default_verts.push_back(0); default_verts.push_back(1);
64      }
65
66      GLuint default_vao;
67      glGenVertexArrays(1, &default_vao);
68      glBindVertexArray(default_vao);
69
70      GLuint default_vbo;
71      glGenBuffers(1, &default_vbo);
72      glBindBuffer(GL_ARRAY_BUFFER, default_vbo);
73      glBufferData(GL_ARRAY_BUFFER, default_verts.size()*sizeof(float), default_verts.data(), GL_STATIC_DRAW);
74
75      GLuint default_shader_id = LoadShaders("default_vert.glsl", "default_frag.glsl");
76      GLint default_mvp_id = glGetUniformLocation(default_shader_id, "mvp");
77      GLint default_posAttrib = glGetAttribLocation(default_shader_id, "position");
78      GLint default_colAttrib = glGetAttribLocation(default_shader_id, "color");
79      glEnableVertexAttribArray(default_posAttrib);
80      glEnableVertexAttribArray(default_colAttrib);
81      glVertexAttribPointer(default_posAttrib, 3, GL_FLOAT, GL_FALSE, 6*sizeof(GLfloat), 0);
```

```
82      glVertexAttribPointer(default_colAttrib, 3, GL_FLOAT, GL_FALSE, 6*sizeof(GLfloat), (void*)(3*sizeof(GLfloat)));
83
84      //
85      // Setup the cube
86      //
87
88      GLfloat cube_verts[] = {
89          // positions
90          0.0f,0.0f,0.0f, 1.0f,0.0f,0.0f, 1.0f,1.0f,0.0f, 0.0f,1.0f,0.0f, //front
91          0.0f,0.0f,1.0f, 1.0f,0.0f,1.0f, 1.0f,1.0f,1.0f, 0.0f,1.0f,1.0f, //back
92          0.0f,0.0f,0.0f, 0.0f,0.0f,1.0f, 0.0f,1.0f,1.0f, 0.0f,1.0f,0.0f, //left
93          1.0f,0.0f,0.0f, 1.0f,0.0f,1.0f, 1.0f,1.0f,1.0f, 1.0f,1.0f,0.0f, //right
94          0.0f,0.0f,0.0f, 1.0f,0.0f,0.0f, 1.0f,0.0f,1.0f, 0.0f,0.0f,1.0f, //bottom
95          0.0f,1.0f,0.0f, 1.0f,1.0f,0.0f, 1.0f,1.0f,1.0f, 0.0f,1.0f,1.0f, //top
96          // normals
97           0.0f, 0.0f,-1.0f,   0.0f, 0.0f,-1.0f,  0.0f, 0.0f,-1.0f,  0.0f, 0.0f,-1.0f,
98           0.0f, 0.0f, 1.0f,   0.0f, 0.0f, 1.0f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f, 1.0f,
99          -1.0f, 0.0f, 0.0f,  -1.0f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
100          1.0f, 0.0f, 0.0f,   1.0f, 0.0f, 0.0f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f, 0.0f,
101          0.0f,-1.0f, 0.0f,   0.0f,-1.0f, 0.0f,  0.0f,-1.0f, 0.0f,  0.0f,-1.0f, 0.0f,
102          0.0f, 1.0f, 0.0f,   0.0f, 1.0f, 0.0f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f, 0.0f
103     };
104
105     GLuint cube_vao;
106     glGenVertexArrays(1, &cube_vao);
107     glBindVertexArray(cube_vao);
108
109     GLuint cube_vbo;
110     glGenBuffers(1, &cube_vbo);
111     glBindBuffer(GL_ARRAY_BUFFER, cube_vbo);
112     glBufferData(GL_ARRAY_BUFFER, sizeof(cube_verts), cube_verts, GL_STATIC_DRAW);
113
114     GLuint cube_shader_id = LoadShaders("cube_vert.glsl", "cube_frag.glsl");
115     cube_model_id = glGetUniformLocation(cube_shader_id, "model");
116     cube_view_id = glGetUniformLocation(cube_shader_id, "view");
117     cube_proj_id = glGetUniformLocation(cube_shader_id, "proj");
118     cube_color_id = glGetUniformLocation(cube_shader_id, "color");
119     GLint cube_posAttrib = glGetAttribLocation(cube_shader_id, "position");
120     GLint cube_norAttrib = glGetAttribLocation(cube_shader_id, "normal");
121     glEnableVertexAttribArray(cube_posAttrib);
122     glEnableVertexAttribArray(cube_norAttrib);
123     glVertexAttribPointer(cube_posAttrib, 3, GL_FLOAT, GL_FALSE, 0, 0);
124     glVertexAttribPointer(cube_norAttrib, 3, GL_FLOAT, GL_FALSE, 0, (void*)(3*4*6*sizeof(GLfloat)));
125
126
127     // scale the unit cube to create all the parts
128     glm::mat4 cam_hing    = glm::scale(glm::mat4(1.0f), glm::vec3( 1.0f,  1.0f,  1.0f));
129     glm::mat4 cam_body    = glm::scale(glm::mat4(1.0f), glm::vec3( 3.0f,  6.0f,  4.0f));
130     glm::mat4 cam_lens    = glm::scale(glm::mat4(1.0f), glm::vec3( 2.0f,  1.0f,  2.0f));
131     glm::mat4 shaft_hing  = glm::scale(glm::mat4(1.0f), glm::vec3( 2.0f,  1.0f,  1.0f));
132     glm::mat4 shaft_body  = glm::scale(glm::mat4(1.0f), glm::vec3( 2.0f, 20.0f,  2.0f));
133     glm::mat4 base_hing   = glm::scale(glm::mat4(1.0f), glm::vec3( 6.0f,  3.0f,  6.0f));
134     glm::mat4 base_body   = glm::scale(glm::mat4(1.0f), glm::vec3(12.0f,  2.0f, 20.0f));
135     glm::mat4 desk_top    = glm::scale(glm::mat4(1.0f), glm::vec3(30.0f,  2.0f, 48.0f));
136     glm::mat4 desk_leg_1  = glm::scale(glm::mat4(1.0f), glm::vec3( 4.0f, 28.0f,  4.0f));
137     glm::mat4 desk_leg_2  = glm::scale(glm::mat4(1.0f), glm::vec3( 4.0f, 28.0f,  4.0f));
138     glm::mat4 desk_leg_3  = glm::scale(glm::mat4(1.0f), glm::vec3( 4.0f, 28.0f,  4.0f));
139     glm::mat4 desk_leg_4  = glm::scale(glm::mat4(1.0f), glm::vec3( 4.0f, 28.0f,  4.0f));
140
141     // define the translations to assemble each part in their local object coordinate system
142     glm::mat4 cam_hing_mcs   = glm::translate(glm::mat4(1.0f), glm::vec3( 0.0f,  0.0f,  0.0f));
143     glm::mat4 cam_body_mcs   = glm::translate(glm::mat4(1.0f), glm::vec3( 1.0f, -3.0f, -1.5f));
144     glm::mat4 cam_lens_mcs   = glm::translate(glm::mat4(1.0f), glm::vec3( 1.5f, -4.0f, -0.5f));
145     glm::mat4 shaft_hing_scs = glm::translate(glm::mat4(1.0f), glm::vec3( 0.5f, 15.0f,  2.0f));
146     glm::mat4 shaft_body_scs = glm::translate(glm::mat4(1.0f), glm::vec3( 0.0f,  0.0f,  0.0f));
147     glm::mat4 base_body_bcs  = glm::translate(glm::mat4(1.0f), glm::vec3( 0.0f,  0.0f,  0.0f));
148     glm::mat4 base_hing_bcs  = glm::translate(glm::mat4(1.0f), glm::vec3( 3.0f,  2.0f,  2.0f));
149     glm::mat4 desk_top_dcs   = glm::translate(glm::mat4(1.0f), glm::vec3( 0.0f, 28.0f,  0.0f));
150     glm::mat4 desk_leg_1_dcs = glm::translate(glm::mat4(1.0f), glm::vec3( 0.0f,  0.0f,  0.0f));
151     glm::mat4 desk_leg_2_dcs = glm::translate(glm::mat4(1.0f), glm::vec3(26.0f,  0.0f,  0.0f));
152     glm::mat4 desk_leg_3_dcs = glm::translate(glm::mat4(1.0f), glm::vec3( 0.0f,  0.0f, 44.0f));
153     glm::mat4 desk_leg_4_dcs = glm::translate(glm::mat4(1.0f), glm::vec3(26.0f,  0.0f, 44.0f));
154
155     // define the translations to assemble the different systems into each other
156     glm::mat4 wcs     = glm::mat4(1.0f);
157     glm::mat4 rcs_wcs = glm::translate(wcs,     glm::vec3(0.0f,  0.0f, 0.0f));
158     glm::mat4 dcs_wcs = glm::translate(rcs_wcs, glm::vec3(0.0f,  0.0f, 0.0f));
159     glm::mat4 ocs_wcs = glm::translate(dcs_wcs, glm::vec3(9.0f, 30.0f, 16.0f));
160
161     // place the desk in world space
162     glm::mat4 desk_top_wcs   = dcs_wcs * desk_top_dcs;
163     glm::mat4 desk_leg_1_wcs = dcs_wcs * desk_leg_1_dcs;
164     glm::mat4 desk_leg_2_wcs = dcs_wcs * desk_leg_2_dcs;
165     glm::mat4 desk_leg_3_wcs = dcs_wcs * desk_leg_3_dcs;
166     glm::mat4 desk_leg_4_wcs = dcs_wcs * desk_leg_4_dcs;
167
168     // define the translations to assemble each component group into the OCS
169     glm::mat4 base_ocs = glm::mat4(1.0f);
170     glm::mat4 base_hing_ocs = base_ocs * base_hing_bcs;
171     glm::mat4 base_body_ocs = base_ocs * base_body_bcs;
172
173     glm::mat4 shaft_ocs = glm::mat4(1.0f);
```

```
174             //shaft_ocs = glm::rotate(shaft_ocs, 30.0f, glm::vec3(1,0,0));
175             shaft_ocs = base_hing_ocs * shaft_ocs;
176             shaft_ocs = glm::translate(shaft_ocs, glm::vec3(2.0f, 0.0f, 2.0f));
177     glm::mat4 shaft_hing_ocs = shaft_ocs * shaft_hing_scs;
178     glm::mat4 shaft_body_ocs = shaft_ocs * shaft_body_scs;
179
180     glm::mat4 cam_ocs = glm::mat4(1.0f);
181             //cam_ocs = glm::rotate(cam_ocs, -45.0f, glm::vec3(1,0,0));
182             cam_ocs = shaft_hing_ocs * cam_ocs;
183             cam_ocs = glm::translate(cam_ocs, glm::vec3(2.0f, 0.0f, 0.0f));
184     glm::mat4 cam_lens_ocs = cam_ocs * cam_lens_mcs;
185     glm::mat4 cam_body_ocs = cam_ocs * cam_body_mcs;
186     glm::mat4 cam_hing_ocs = cam_ocs * cam_hing_mcs;
187
188     // define the translations to assemble the OCS in world space (WCS)
189     glm::mat4 base_hing_wcs  = ocs_wcs * base_hing_ocs;
190     glm::mat4 base_body_wcs  = ocs_wcs * base_body_ocs;
191     glm::mat4 shaft_hing_wcs = ocs_wcs * shaft_hing_ocs;
192     glm::mat4 shaft_body_wcs = ocs_wcs * shaft_body_ocs;
193     glm::mat4 cam_lens_wcs   = ocs_wcs * cam_lens_ocs;
194     glm::mat4 cam_body_wcs   = ocs_wcs * cam_body_ocs;
195     glm::mat4 cam_hing_wcs   = ocs_wcs * cam_hing_ocs;
196
197
198     //
199     // Setup the projection and view matrices
200     //
201
202     // simple camera
203     float camera_dist = 80.0f;
204     float camera_angle_v = 0.0f;
205     float camera_angle_h = 0.0f;
206     glm::vec3 camera_target(15.0f, 25.0f, 24.0f);
207
208     projection = glm::perspective(45.0f, 1.0f, 0.1f, 1000.0f);
209
210     int poly_mode = 0;
211
212     glEnable(GL_DEPTH_TEST);
213     glDepthFunc(GL_LESS);
214
215     while (window.isOpen())
216     {
217         sf::Event event;
218         while (window.pollEvent(event))
219         {
220             // Close window : exit
221             if (event.type == sf::Event::Closed)
222                 window.close();
223
224             if (event.type == sf::Event::KeyPressed) {
225                 switch(event.key.code) {
226                     case sf::Keyboard::Escape:
227                         window.close();
228                         break;
229                     case sf::Keyboard::P:
230                         poly_mode = !poly_mode;
231                         break;
232                     case sf::Keyboard::Left:
233                         camera_angle_h -= (M_PI / 4);
234                         break;
235                     case sf::Keyboard::Right:
236                         camera_angle_h += (M_PI/ 4);
237                         break;
238                     case sf::Keyboard::Up:
239                         camera_angle_v += (M_PI / 4);
240                         break;
241                     case sf::Keyboard::Down:
242                         camera_angle_v -= (M_PI / 4);
243                         break;
244                     case sf::Keyboard::PageUp:
245                         camera_dist *= 0.9f;
246                         break;
247                     case sf::Keyboard::PageDown:
248                         camera_dist *= 1.1f;
249                         break;
250                     case sf::Keyboard::Space:
251                         camera_target = glm::vec3(15.0f, 25.0f, 24.0f);
252                         break;
253                     case sf::Keyboard::C:
254                         camera_target = glm::vec3(0.0f, 0.0f, 0.0f);
255                         break;
256                     case sf::Keyboard::A:
257                         camera_target.x -= 1;
258                         break;
259                     case sf::Keyboard::D:
260                         camera_target.x += 1;
261                         break;
262                     case sf::Keyboard::S:
263                         camera_target.z -= 1;
264                         break;
265                     case sf::Keyboard::W:
```

```
266                         camera_target.z += 1;
267                         break;
268                     case sf::Keyboard::E:
269                         camera_target.y -= 1;
270                         break;
271                     case sf::Keyboard::Q:
272                         camera_target.y += 1;
273                         break;
274                 }
275             }
276         }
277 
278         glm::vec3 camera_direction(
279                 cos(camera_angle_v) * sin(camera_angle_h),
280                 sin(camera_angle_v),
281                 cos(camera_angle_v) * cos(camera_angle_h)
282                 );
283 
284         view = glm::lookAt(
285                 (camera_dist * camera_direction) + camera_target, // position
286                 camera_target,        // target
287                 glm::vec3(0,1,0)         // up
288                 );
289 
290         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
291         glClearColor(1.0, 1.0, 1.0, 0.0);
292 
293         if (poly_mode) {
294             glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
295             glLineWidth(2);
296         } else {
297             glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
298             glLineWidth(1);
299         }
300 
301         //
302         // Draw the body
303         //
304 
305         glUseProgram(cube_shader_id);
306         glBindVertexArray(cube_vao);
307         glBindBuffer(GL_ARRAY_BUFFER, cube_vbo);
308 
309         // define each components final model matrix (mm)
310         glm::mat4 cam_lens_mm    = cam_lens_wcs    * cam_lens;
311         glm::mat4 cam_body_mm    = cam_body_wcs    * cam_body;
312         glm::mat4 cam_hing_mm    = cam_hing_wcs    * cam_hing;
313         glm::mat4 shaft_hing_mm = shaft_hing_wcs * shaft_hing;
314         glm::mat4 shaft_body_mm = shaft_body_wcs * shaft_body;
315         glm::mat4 base_hing_mm   = base_hing_wcs  * base_hing;
316         glm::mat4 base_body_mm   = base_body_wcs  * base_body;
317         glm::mat4 desk_top_mm    = desk_top_wcs    * desk_top;
318         glm::mat4 desk_leg_1_mm = desk_leg_1_wcs * desk_leg_1;
319         glm::mat4 desk_leg_2_mm = desk_leg_2_wcs * desk_leg_2;
320         glm::mat4 desk_leg_3_mm = desk_leg_3_wcs * desk_leg_3;
321         glm::mat4 desk_leg_4_mm = desk_leg_4_wcs * desk_leg_4;
322 
323         draw_cube(cam_lens_mm    , glm::vec3(0.0f, 0.0f, 1.0f));
324         draw_cube(cam_body_mm    , glm::vec3(0.0f, 1.0f, 0.0f));
325         draw_cube(cam_hing_mm    , glm::vec3(0.0f, 1.0f, 1.0f));
326         draw_cube(shaft_hing_mm , glm::vec3(1.0f, 0.0f, 0.0f));
327         draw_cube(shaft_body_mm , glm::vec3(1.0f, 0.0f, 1.0f));
328         draw_cube(base_hing_mm   , glm::vec3(1.0f, 0.5f, 0.0f));
329         draw_cube(base_body_mm   , glm::vec3(1.0f, 1.0f, 0.5f));
330         draw_cube(desk_top_mm    , glm::vec3(0.5f, 0.0f, 1.0f));
331         draw_cube(desk_leg_1_mm , glm::vec3(0.5f, 0.0f, 0.5f));
332         draw_cube(desk_leg_2_mm , glm::vec3(0.5f, 0.0f, 0.5f));
333         draw_cube(desk_leg_3_mm , glm::vec3(0.5f, 0.0f, 0.5f));
334         draw_cube(desk_leg_4_mm , glm::vec3(0.5f, 0.0f, 0.5f));
335 
336         //
337         // Draw the grid
338         //
339 
340         glUseProgram(default_shader_id);
341         glBindVertexArray(default_vao);
342         glBindBuffer(GL_ARRAY_BUFFER, default_vbo);
343 
344         glm::mat4 mvp = projection * view * glm::mat4(1.0f);
345         glUniformMatrix4fv(default_mvp_id, 1, GL_FALSE, glm::value_ptr(mvp));
346 
347         glDrawArrays(GL_LINES, 0, default_verts.size()/6);
348 
349         window.display();
350     }
351 
352     return 0;
353 }
```

Listing 1: Main Program

```
#version 330 core

uniform mat4 model;
uniform mat4 view;
uniform mat4 proj;

in vec3 position;
in vec3 normal;

out vec3 Normal;
out vec3 frag_pos;

void main(){
    frag_pos = vec3(model * vec4(position, 0)).zxy;
    Normal = normal;
        gl_Position = proj * view * model * vec4(position,1);
}
```

Listing 2: Cube Vertex Shader

```
#version 150

in vec3 frag_pos;
in vec3 Normal;
uniform vec3 color;
out vec3 out_color;

void main()
{
    vec3 light_dir = vec3(-0.3,0.9,0.1);
    float intensity = dot(light_dir, normalize(Normal));

    vec3 f = fract(frag_pos);

    if ((f.x < 0.1 || f.x > 0.9) && (f.y < 0.1 || f.y > 0.9) ||
        (f.x < 0.1 || f.x > 0.9) && (f.z < 0.1 || f.z > 0.9) ||
        (f.z < 0.1 || f.z > 0.9) && (f.y < 0.1 || f.y > 0.9)) {
        out_color = vec3(0);
    } else {
        out_color = color;
    }

    //out_color = color * intensity;
}
```

Listing 3: Cube Fragment Shader

```
#version 150

in vec3 position;
in vec3 color;

uniform mat4 mvp;

out vec3 Color;

void main()
{
    Color = color;
    gl_Position = mvp * vec4(position, 1);
}
```

Listing 4: Default Vertex Shader

```
#version 150

in vec3 Color;
out vec4 out_color;

void main()
{
    out_color = vec4(Color, 1.0);
}
```

Listing 5: Default Fragment Shader