Computer Graphics: Assignment 2

Akash 2021009

1. Program the arrow keys on your keyboard to change the camera view of the scene.

(a) The camera center should change as follows:
- Left and right arrows move the camera along -X and +X of the camera axes.
- Down and up arrows move the camera along the -Y and +Y of the camera axes.
- Shift + down/up arrows move the camera along the -Z and +Z of the camera axes. This will give the effect of Zoom out/in.

Keep the look-at point to be the world space origin (0,0,0) and the camera up-vector to the world vector (0, 1, 0). Do not forget to update the viewing matrix after every change in camera center. Note that arrow keys should change camera center coordinates in the camera space (and not in the world space), which will result in a camera motion on a sphere around the origin. The glm::lookAt() in setupViewTransformation() requires camera center in world space.

Ans: One of the posible way can be; camera should follow the path of sphere around the cuboid and it's coordinates update as per parametric equation of sphere while pressing arrow keys.

Parametric equation of sphere:

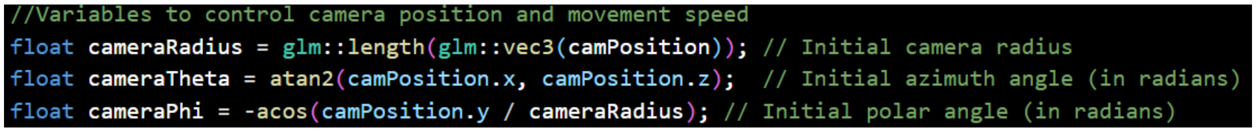
x(phi, theta) = r*sin(phi)*cos(theta))
z(phi, theta) = r*sin(phi)*sin(theta)
y(phi, theta) = r*cos(phi)

Where r is radius of sphere and phi is angle of x,z plane with y axis and theta is angle between x and z plane.

Here phi can vary from [0,π] and theta can vary from [0, 2*π].

Now we can convert vector equation to parametric equation and vise-versa.

We can convert glm::vec4 of camPosition to cameraRadius, cameraPhi and cameraTheta as shown below:



```
//Variables to control camera position and movement speed
float cameraRadius = glm::length(glm::vec3(camPosition)); // Initial camera radius
float cameraTheta = atan2(camPosition.x, camPosition.z);  // Initial azimuth angle (in radians)
float cameraPhi = -acos(camPosition.y / cameraRadius); // Initial polar angle (in radians)
```

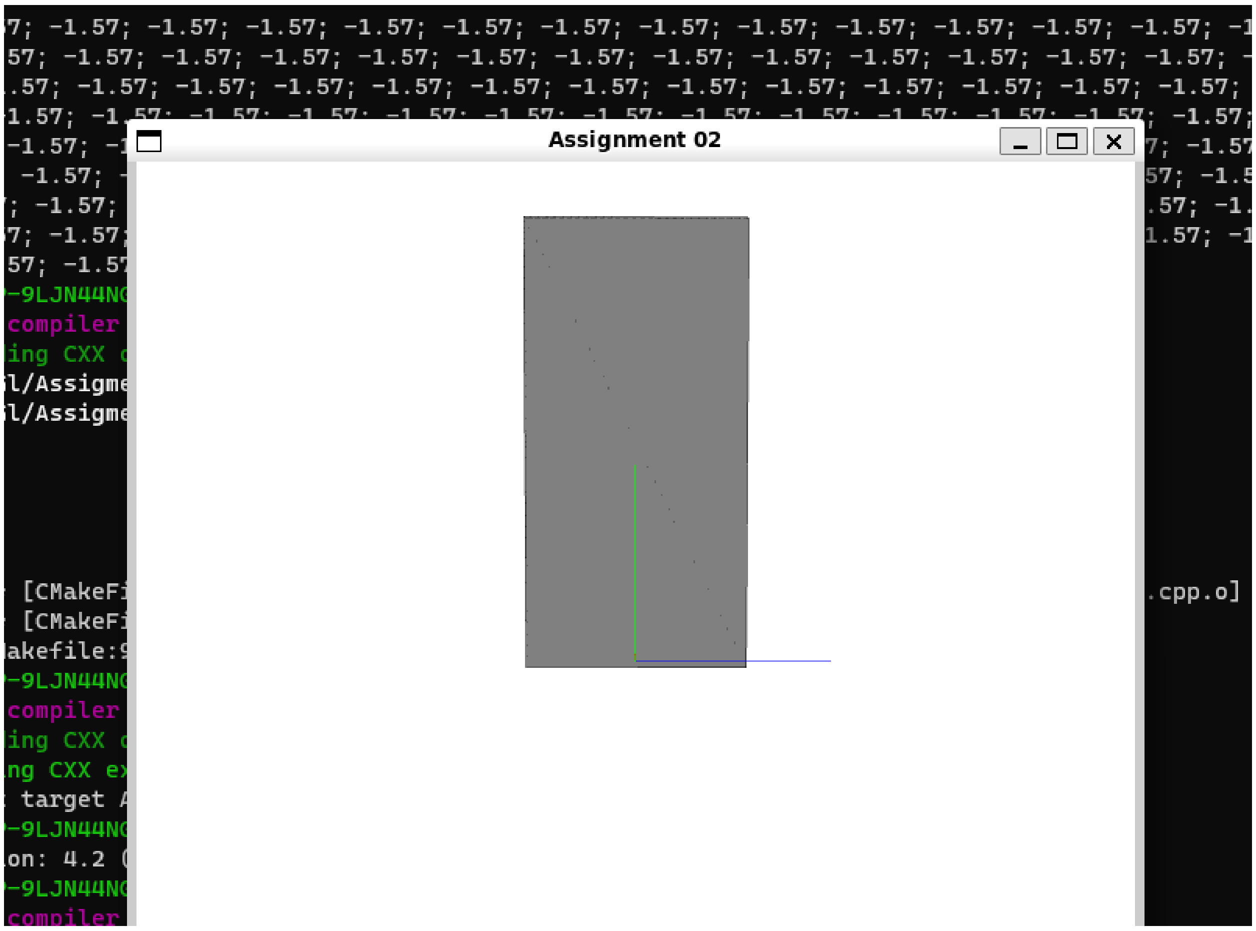
Now we can zoom in and out by increasing and decreasing value of cameraRadius respectively. To orbit around the x axis we can increase/decrease value of cameraTheta and to orbit around y axis we can in increase/decrease phi(within the range of 0 to π). For updating all these values we can simply map the keys respectively in Get Key Presses section of provided code.
Now we can update glm::vec4 via formula explained above:

```
// Calculate new camera position in Cartesian coordinates as per updated cameraRadius,cameraPhi and cameraTheta using parametric equation of sphere
        camPosition.x = cameraRadius * sin(cameraPhi) * cos(cameraTheta);
        camPosition.y = cameraRadius * cos(cameraPhi);
        camPosition.z = cameraRadius * sin(cameraPhi) * sin(cameraTheta);
```

Call setupViewTransformation(shaderProgram) to update viewing matrix.

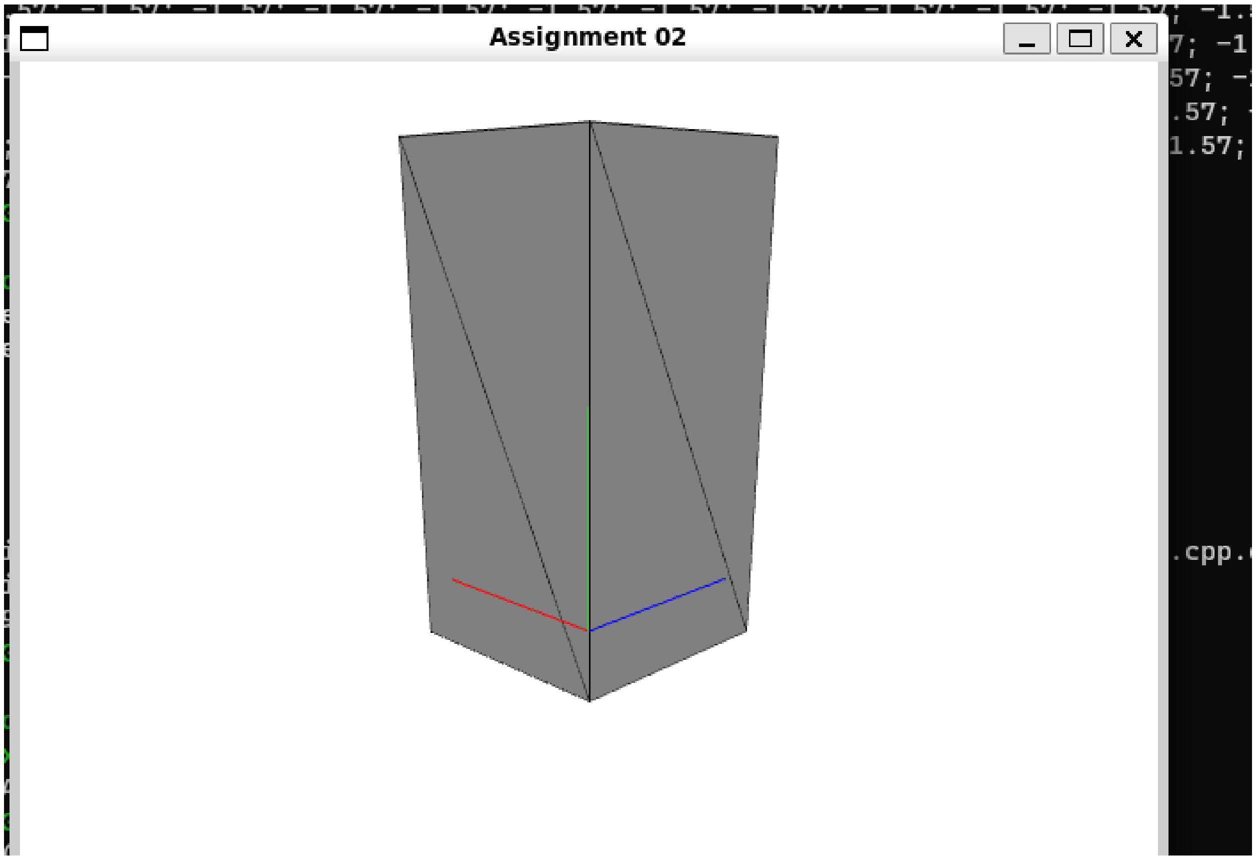
(b) Move the camera to specific positions and generate one-point perspective, two-point perspective, and the two three-point perspective views (bird's eye view and rat's eye view).

One-point perspective view:

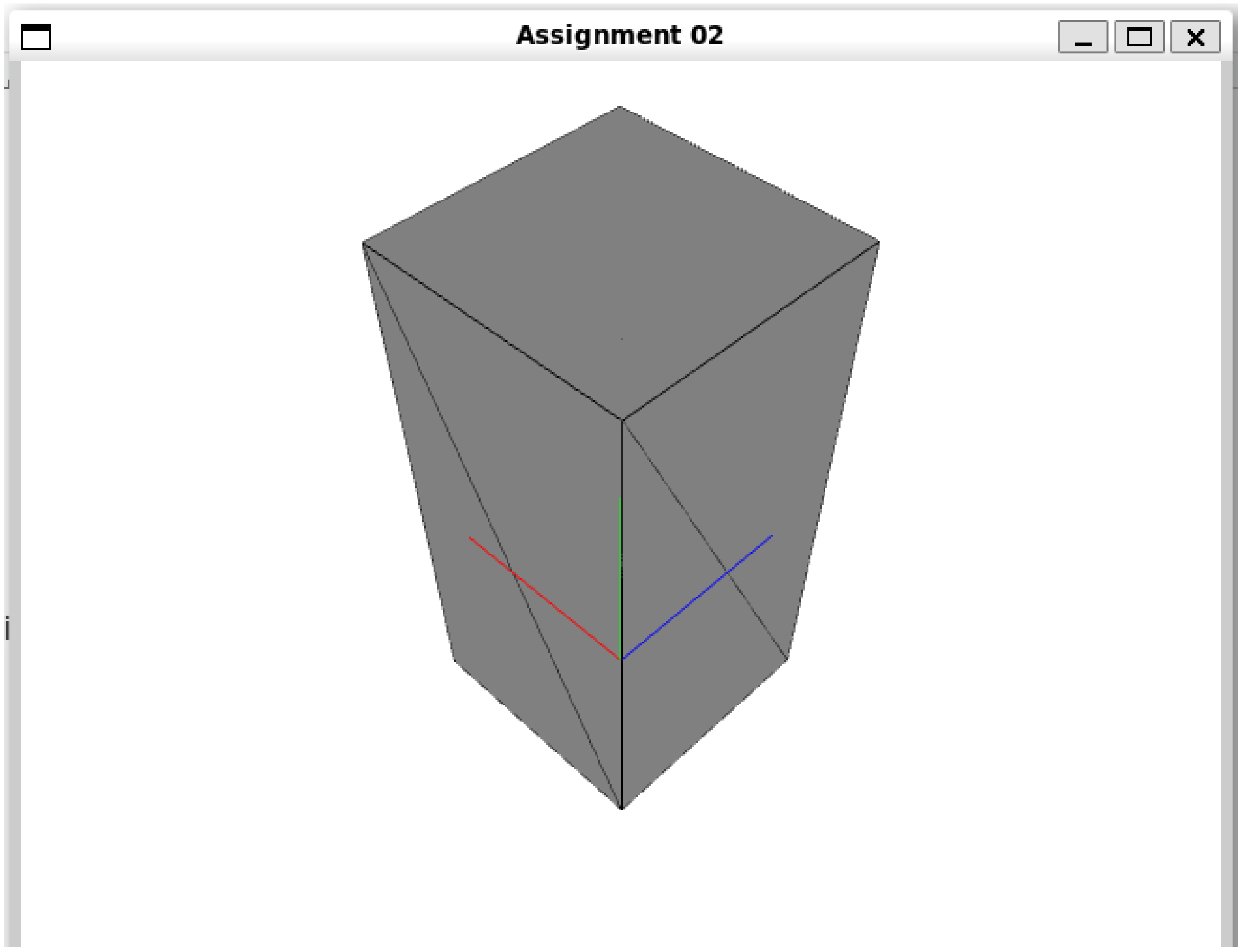


```
7; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1
57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -
.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57;
1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57; -1.57;
-1.57; -1                                                                  7; -1.57
 -1.57; -                                                                  57; -1.5
; -1.57;                                                                   .57; -1.
7; -1.57;                                                                  1.57; -1
57; -1.57
-9LJN44NC
compiler
ing CXX c
l/Assigme
l/Assigme




 [CMakeFi                                                                  .cpp.o]
 [CMakeFi
akefile:9
-9LJN44NC
compiler
ing CXX c
ng CXX ex
 target A
-9LJN44NC
on: 4.2 (
-9LJN44NC
compiler
```

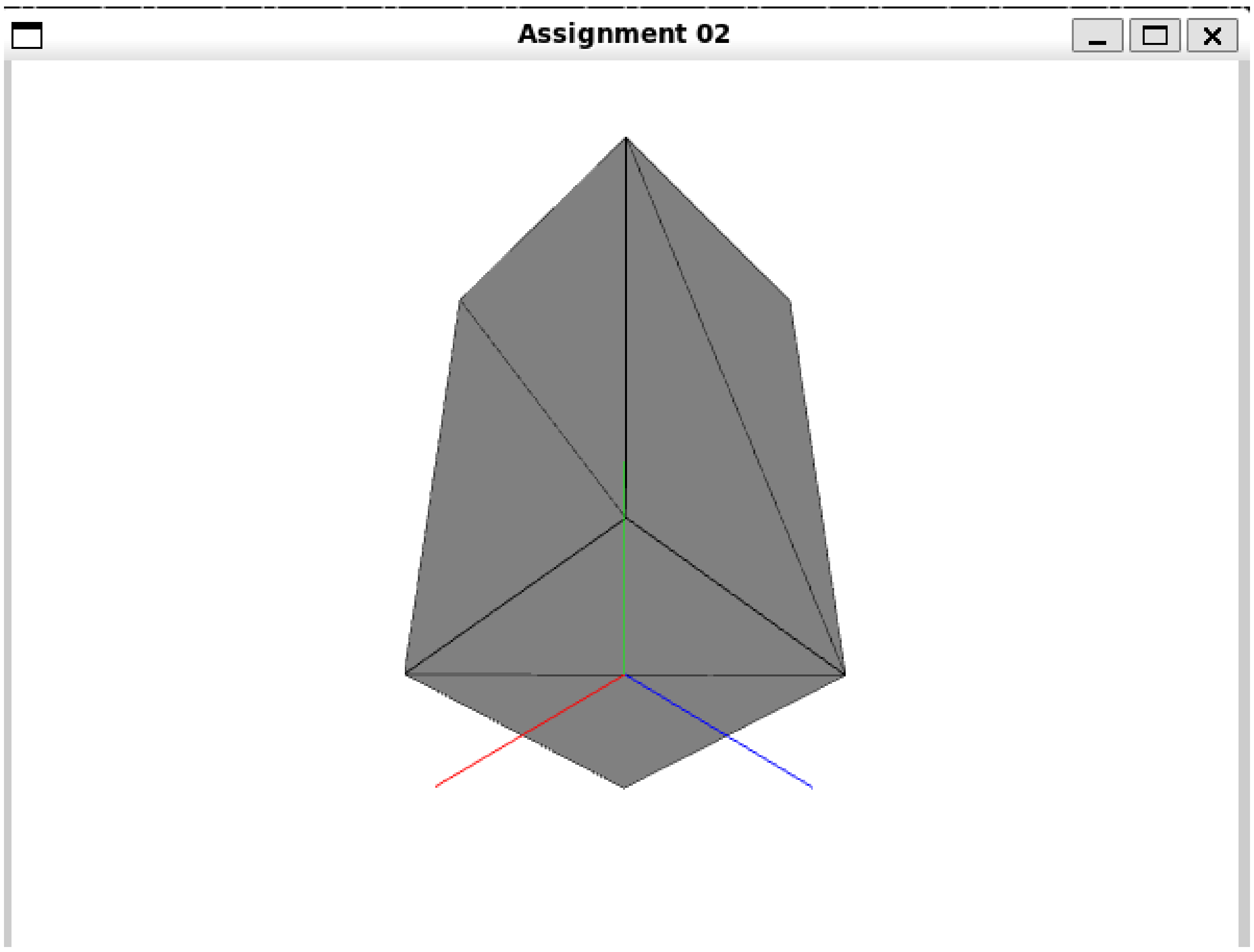
Two-point perspective view:



Three-point perspective view(bird eye):



Three-point perspective view(rat eye):



2. The given program performs a perspective projection using GLMs perspective() function. Keep the arrow key movement from the previous question and

1. Program the 'o'/'O' and the 'p'/'P' keys on the keyboard to switch between orthographic and perspective projections respectively. You can use 1 ImGui::IsKeyPressed() function to detect keyboard keys.

Ans:

Firstly we need to update setupProjectionTransformation function so that we can pass a parameter which will generate orthographic view or perspective view as per provided value.

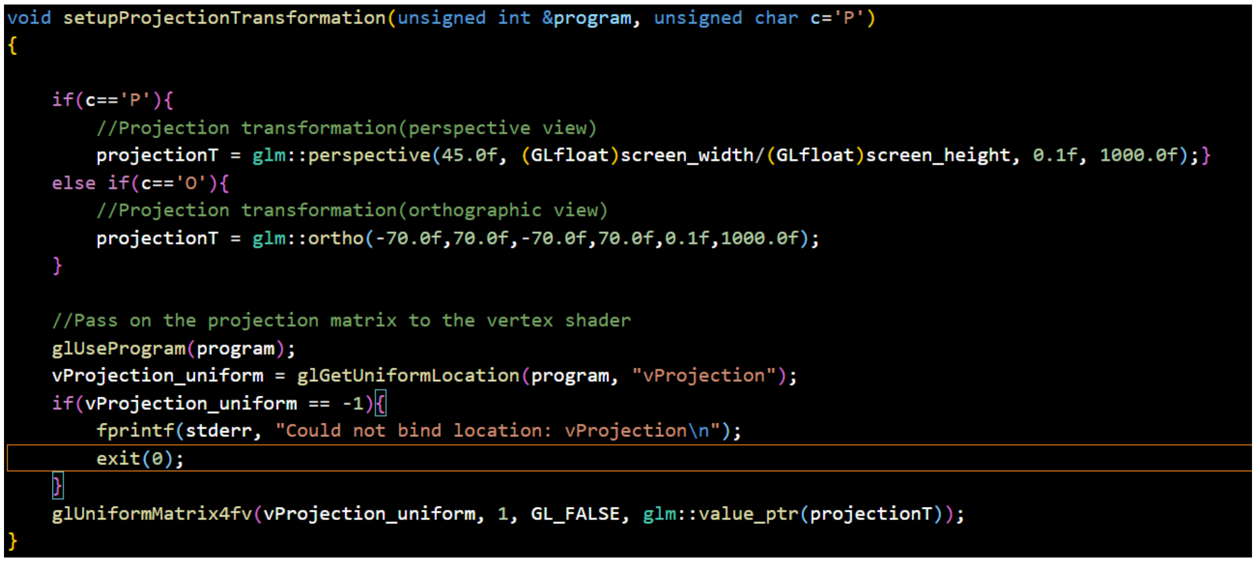
update void setupProjectionTransformation(unsigned int &) to void setupProjectionTransformation(unsigned int &,unsigned char);

The char variable 'c' will generate projection; if c=='P' it will generate perspective view and if c=='O' it will generate orthographic view.

To generate orthogonal view we can use function: glm::ortho(left_boundary, right_boundary, top_boundary, bottom_boundary, near_clipping, far_clipping);

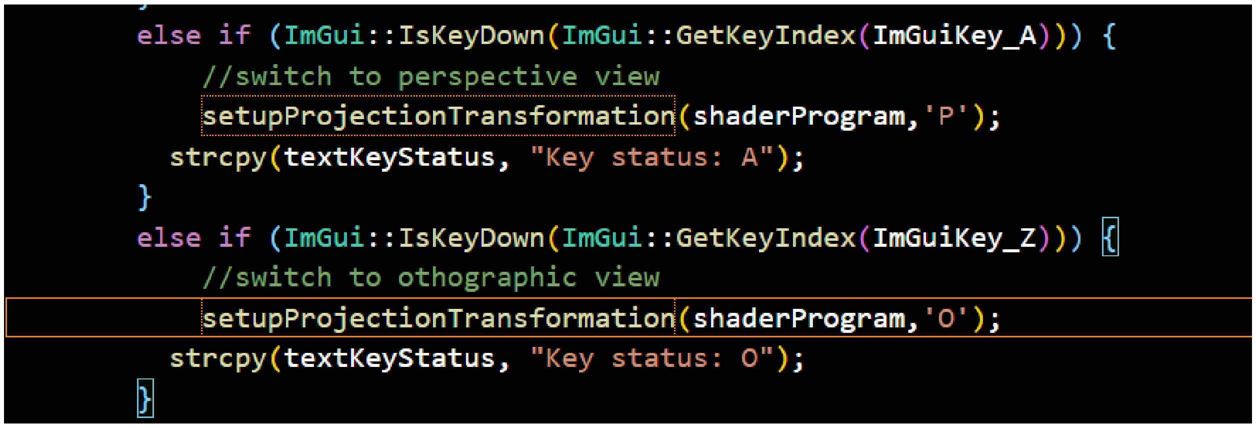
To generate perspective view we can use function: glm::perspective(field_of_view_angle, aspect_ratio, near_plane, far_plane);

Final updated setupProjectionTransformation function will be look like:



```
void setupProjectionTransformation(unsigned int &program, unsigned char c='P')
{

    if(c=='P'){
        //Projection transformation(perspective view)
        projectionT = glm::perspective(45.0f, (GLfloat)screen_width/(GLfloat)screen_height, 0.1f, 1000.0f);}
    else if(c=='O'){
        //Projection transformation(orthographic view)
        projectionT = glm::ortho(-70.0f,70.0f,-70.0f,70.0f,0.1f,1000.0f);
    }

    //Pass on the projection matrix to the vertex shader
    glUseProgram(program);
    vProjection_uniform = glGetUniformLocation(program, "vProjection");
    if(vProjection_uniform == -1){
        fprintf(stderr, "Could not bind location: vProjection\n");
        exit(0);
    }
    glUniformMatrix4fv(vProjection_uniform, 1, GL_FALSE, glm::value_ptr(projectionT));
}
```

Now we can simply map keys to switch between orthographic and perspective view in Get key Presses section of provided code.



```
        else if (ImGui::IsKeyDown(ImGui::GetKeyIndex(ImGuiKey_A))) {
            //switch to perspective view
            setupProjectionTransformation(shaderProgram,'P');
          strcpy(textKeyStatus, "Key status: A");
        }
        else if (ImGui::IsKeyDown(ImGui::GetKeyIndex(ImGuiKey_Z))) {
            //switch to othographic view
            setupProjectionTransformation(shaderProgram,'O');
          strcpy(textKeyStatus, "Key status: O");
        }
```

Here, I'm using key A to switch in Perspective view and key Z to switch in Orthographic view.

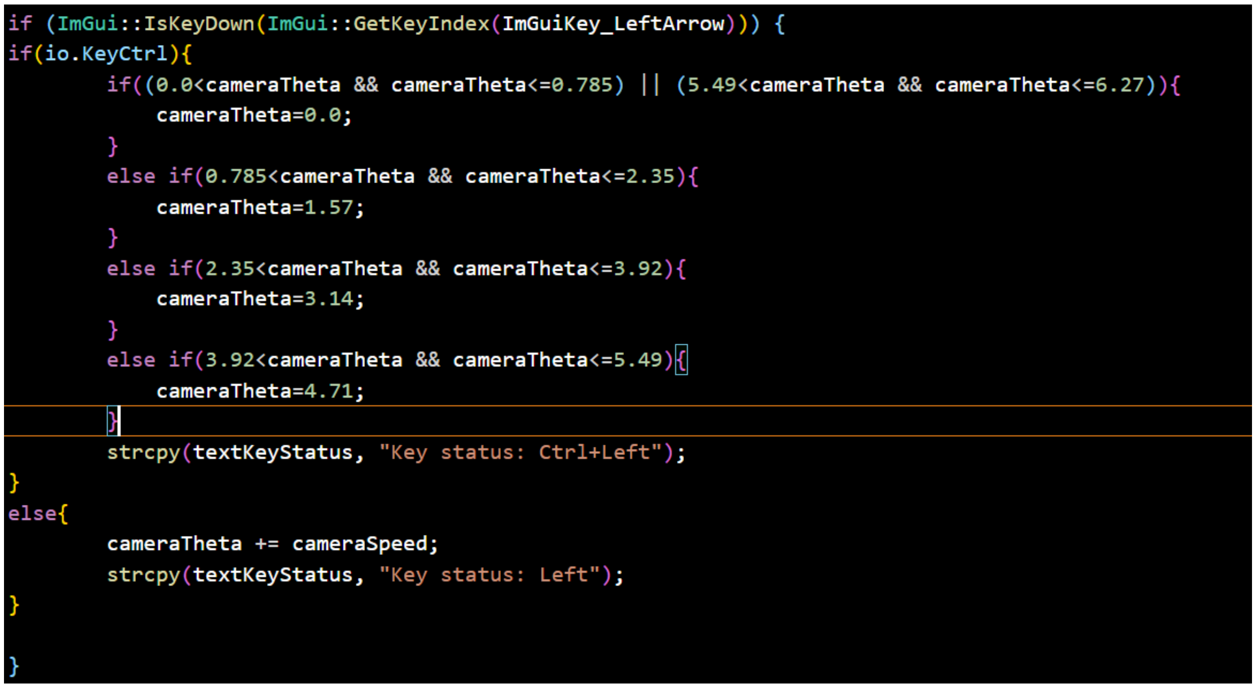
2. Using arrow keys, move the camera to specific positions to generate top view, front elevation, and side elevation. Use a modifier key on your keyboard like Ctrl (control) to snap camera center to appropriate axes/lines in order to accurately generate such views.

To move camera on Top View we can simply make cameraPhi(explained in Q1) to 0.0 when up key presses as this variable is use to orbit in y direction(When cameraPhi is 0.0 camera position will be perpendicular to y axis)

To get snap to front elevated view and side elevated view we can snap value of cameraTheta to closest axis(axis which comes towards us from screen) while pressing left/right arrow keys.
For that; as we know cameraTheta moves from in a circular orbit from 0 to 2π, we can see which value from 0, π/2, π and 3π/4 is closest to current cameraTheta value(as on these values will give side or front view) and snap cameraTheta to closest value.

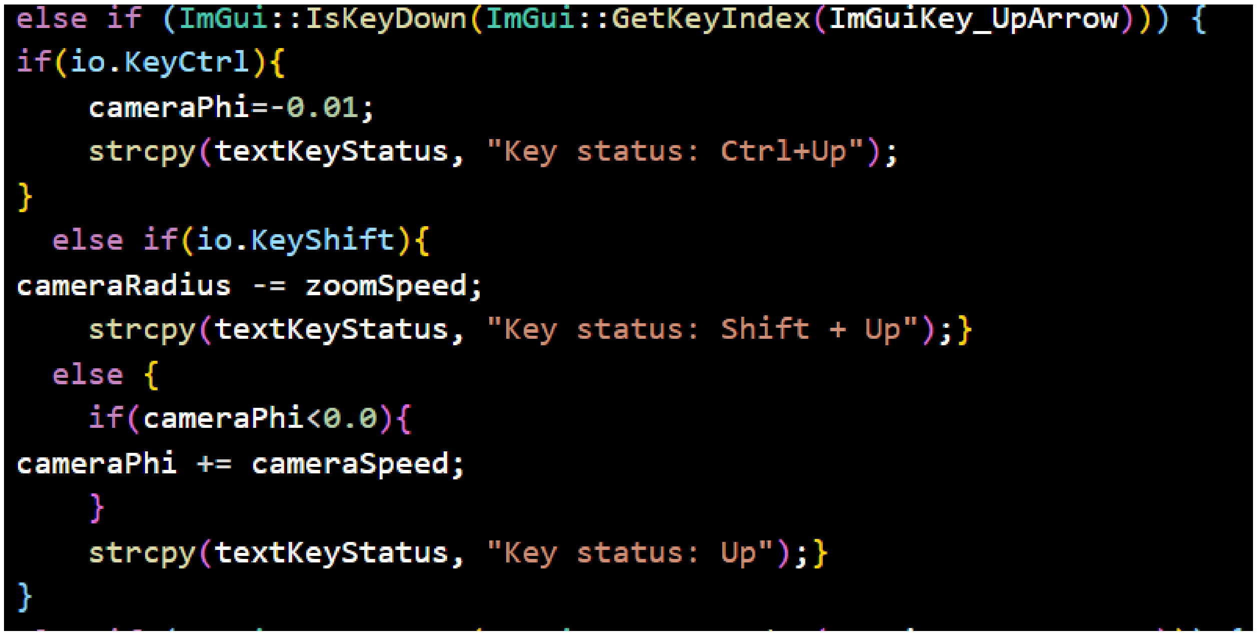
We can simply map this logic to desired keys in Get Key Presses section of provided code.

The explained logic will be look like this in code:
for side and front elevated view



```
if (ImGui::IsKeyDown(ImGui::GetKeyIndex(ImGuiKey_LeftArrow))) {
if(io.KeyCtrl){
        if((0.0<cameraTheta && cameraTheta<=0.785) || (5.49<cameraTheta && cameraTheta<=6.27)){
            cameraTheta=0.0;
        }
        else if(0.785<cameraTheta && cameraTheta<=2.35){
            cameraTheta=1.57;
        }
        else if(2.35<cameraTheta && cameraTheta<=3.92){
            cameraTheta=3.14;
        }
        else if(3.92<cameraTheta && cameraTheta<=5.49){
            cameraTheta=4.71;
        }
        strcpy(textKeyStatus, "Key status: Ctrl+Left");
}
else{
        cameraTheta += cameraSpeed;
        strcpy(textKeyStatus, "Key status: Left");
}

}
```

for top view



```
else if (ImGui::IsKeyDown(ImGui::GetKeyIndex(ImGuiKey_UpArrow))) {
if(io.KeyCtrl){
    cameraPhi=-0.01;
    strcpy(textKeyStatus, "Key status: Ctrl+Up");
}
  else if(io.KeyShift){
cameraRadius -= zoomSpeed;
    strcpy(textKeyStatus, "Key status: Shift + Up");}
  else {
    if(cameraPhi<0.0){
cameraPhi += cameraSpeed;
    }
    strcpy(textKeyStatus, "Key status: Up");}
}
```

3. Find the inverse of the rigid body transformation. Show all of your steps.
[R   t]
[0 0 0 1]

where is a 3 x 3 rotation matrix and is a 3-vector.

Ans:
A rigid body transformation consists of a rotation matrix, denoted as "R," and a translation vector, denoted as "t." This transformation is typically represented as a 4x4 matrix in homogeneous coordinates like this:
[R   t]
[0 0 0 1]

Let's go through the steps to find the inverse:

Step 1:
Let A=[R   t]
     [000 1]

As we know, $AA^{-1}=I$.
We can use this property to find A^{-1}.

Step 2:

Assume A^{-1}=[x y]
[000 1]

AA^{-1}=[R t] [x y] = [R.x R.y+t]
[000 1] [000 1] [000 1]

Step 3:
Now from step 1, we know:
R.x =I and R.y+t must be zero vector.

By R.x =I, x=R^{-1}=R^{T} (as R is orthogonal matrix so it's inverse is its transpose)

By R.y+t=0; R.y=-t

Which is y=-R^{-1}t=-R^{T}t(as R is orthogonal matrix so it's inverse is its transpose)

Step 4:
Now we know inverse of rigid body matrix(From step 2); A^{-1}=[x y] = [R^{T} -R^{T}t]
[000 1] [000 1]

—--