Advanced Graphics Programming

Workshop Eight (Part B) – Camera Controls (2D)

In this workshop you will extend workshop 8 (Part A) to add camera controls.

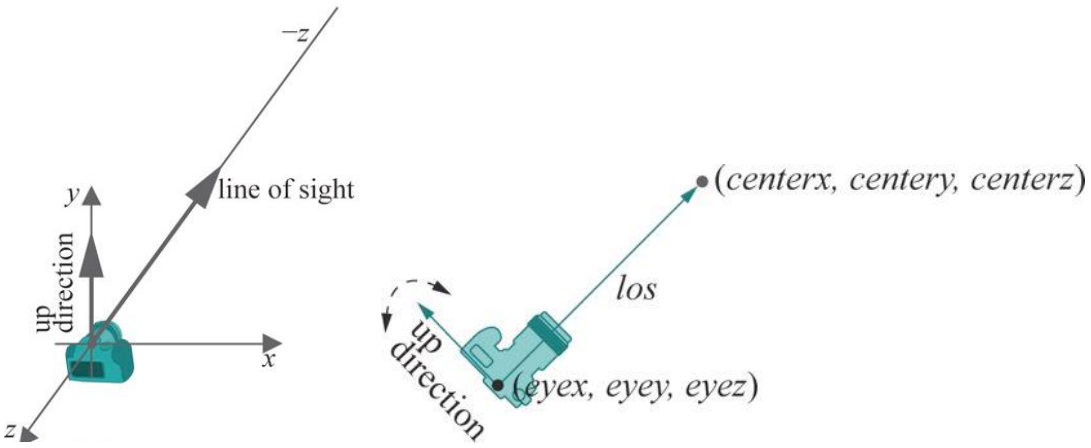
Step 1: Step up the camera for 3D viewing. Take a copy of workshop 8 (Part A) your fractal tree and remove the line for orthographic projection and then add a line for perspective projection. This can be done with the glm frustum function or the more intuitive perspective function. The glm perspective function takes four parameters: fovy, aspect, near and far. fovy is the field of view in radians, aspect ratio = width / height and near and far are the clipping planes.

```
// Obtain projection matrix uniform location and set value.
projMatLoc = glGetUniformLocation(programId, "projMat");
projMat = ortho(0.0, 100.0, 0.0, 100.0, -1.0, 1.0);
projMat = perspective(radians(60.0), (double)ASPECT, 0.1, 200.0);
glUniformMatrix4fv(projMatLoc, 1, GL_FALSE, value_ptr(projMat));
```

Even if you set the perspective projection correctly and run your code you will get an empty screen as you also need to set the camera to point at the tree as shown in the next step.

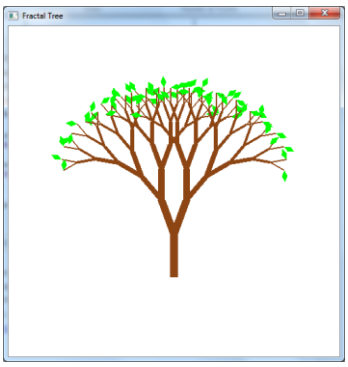
Step 2: Set the camera to look at the tree. The glm lookAt(**eye, center, up**) function simulates moving the camera to the location **eye**, pointed at **center** and rotated about the line of sight (los) to the **up** direction. **eye, center, up** are all vectors with (x,y,z) values. Add the following code to your drawScene method to set the camera to look at the tree.

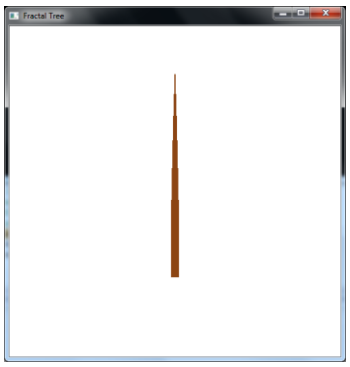
```
modelViewMat = lookAt(eye, center, up);
glUniformMatrix4fv(modelViewMatLoc, 1, GL_FALSE, value_ptr(modelViewMat));
```

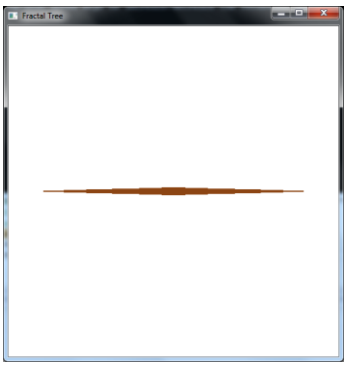


When you have set both perspective and lookAt, run your code and you should see something similar to your previous workshop, a 2D tree viewed from the front.

Step 3: Now try viewing your tree from the front, side and top by changing the lookAt parameters. It is important to understand how to move the camera so that you can debug your implementation and add keyboard controls later in this workshop.


Front view


Side View


Top View

It is useful to add keyboard shortcuts (e.g 'f', 's' and 't') for each of these views so that you can toggle between them.

The next few steps will take you through setting up a virtual camera that can be moved by the keyboard or mouse. Initially, you should select two keys to rotate the camera around the Y axis (e.g. in the XZ plane) and two keys to move the camera forward and backward in the current direction.

Step 4: Create variables that can be used to manipulate the camera direction and set them to the initial values used to view the tree from the front.

- cameraTheta = 0 : the angle of rotation around the Y axis
- **los** (0,0,-1): A **unit** vector (x,y,z) defining the line of sight. Note that the line of sight is a unit vector as this simplifies camera transformations.

Then in your drawScene function update the camera position using the lookAt function. This time you do not want to fix the camera to look at the tree but look in the direction specified by the user, so update the center as follows:

lookAt(**eye**, **eye** + **los**, **up**);

Warning: After this change only the front view will still work, the side view will be fixed in step 6 and the top view in the next workshop where you will add the ability to rotate the camera up and down.

Step 5: Now you are ready to process the keyboard input so start by moving the camera forward and backward along the line of sight. This can be done by adding or subtracting a multiple of the line of sight vector (los), remember that as los is a unit vector it has length 1.

eye.x = eye.x + los.x * speed
eye.z = eye.z + los.z * speed

Test that you can move forward and backward in your scene.

Step 6: Next add two keys to rotate the camera left and right. These keys will control the rotation of the camera around the y-axis, as this will result in movement on the XZ plane so you do not need to change the los.y. First add or subtract a small amount to the current camera theta angle depending on the direction of rotation and then the new los.x and los.z can be calculated using:

los.x = sin(cameraTheta)
los.z = -cos(cameraTheta)

Hint: In C++ cos is calcuated in radians so you will need to convert degrees to radians.

Test that you can rotate the camera to the left and right in your scene. If you want to also be able to view your tree from the side you will need to set the right amount of camera rotation (cameraTheta), when switching views.

Step 7: Add two more keys that allow you to strafe right and left. Test that you can strafe left and right in your scene.

Hint: You can strafe by using the cross product of the los vector and the up vector to get a vector pointed to the camera's right.

Gobal variables:

```
int DisplayMode = 0; //0 front view; 1 for side view; 2 for top view
float ZoomFactor = 0.0;
float cameraTheta = 0.0;
```

Part 1: Keyboard input function

```
// Keyboard input processing routine.
void keyInput(unsigned char key, int x, int y)
{
  switch(key)
  {
          case 'f':
                  ZoomFactor = 0.0;
                  DisplayMode = 0;
                  break;
          case 's':
                  ZoomFactor = 0.0;
                  DisplayMode = 1;
                  break;
          case 't':
                  ZoomFactor = 0.0;
                  DisplayMode = 2;
                  break;
          case 'o':
                  ZoomFactor = ZoomFactor + 1.0;
                  DisplayMode = 3;
                  break;
          case 'l':
                  ZoomFactor = ZoomFactor - 1.0;
                  DisplayMode = 3;
                  break;
          case 'b':  //camera rotation
                  cameraTheta = cameraTheta + 10.0;
                  DisplayMode = 4;
                  break;
          case 'n': //camera rotation
                  cameraTheta = cameraTheta - 10.0;
                  DisplayMode = 4;
                  break;
    default:
      break;
  }
  glutPostRedisplay(); // Redraw screen
}
```

Part 2: Modify modelview matrix

```
// Obtain modelview matrix uniform location and set value.
   modelViewMatLoc = glGetUniformLocation(programId, "modelViewMat");
   vec3 eye, center, up, los;

   if (DisplayMode == 0)
   {
          eye.x = 0.0; eye.y = 0.0; eye.z = 100.0; //front view
          up.x = 0.0; up.y = 1.0; up.z = 0.0; //front view
   }
   else if (DisplayMode == 1)
   {
          eye.x = 100.0; eye.y = 0.0; eye.z = 0.0; //side view
          up.x = 0.0; up.y = 1.0; up.z = 0.0; //side view
   }
   else if (DisplayMode == 2)//top view
   {
          eye.x = 0.0; eye.y = 100.0; eye.z = 0.0;  //top view
          up.x = 0.0; up.y = 0.0; up.z = 1.0; // topview
   }
   else if(DisplayMode == 3)
   {
          eye.x = 0.0; eye.y = 0.0; eye.z = 100.0; //front view
          up.x = 0.0; up.y = 1.0; up.z = 0.0; //front view
          los.x = 0.0; los.y = 0.0; los.z = -1.0;
          eye.x = eye.x + los.x * ZoomFactor;
          eye.z = eye.z + los.z * ZoomFactor;
   }
   else
   {
          eye.x = 0.0; eye.y = 0.0; eye.z = 100.0; //front view
          up.x = 0.0; up.y = 1.0; up.z = 0.0; //front view
          //los.x = 0.0; los.y = 0.0; los.z = -1.0;
          //rotation eye around the center
          los.x = eye.x*cos(radians(cameraTheta)) - eye.z*sin(radians(cameraTheta));
          los.z = eye.x*sin(radians(cameraTheta)) + eye.z*cos(radians(cameraTheta));
          eye.x = los.x;
          eye.z = los.z;
   }

   center.x = 0.0; center.y = 0.0; center.z = 0.0;
   modelViewMat = lookAt(eye, center, up);
   glUniformMatrix4fv(modelViewMatLoc, 1, GL_FALSE, value_ptr(modelViewMat));
   ////////////////////////////////////
```