**Coms 331** **Assignment 9** **Fall 2019**

## Introduction

This assignment is the first in a series of assignments that will build the canal boat application. This is also the first assignment that renders a 3-dimensional object. To keep it simple, we will not include any lighting effects. Those will come in subsequent assignments. Thus, the rendered scene will not look 3-dimensional, but we have to start somewhere.

The object that will will create and render is the two side walls of the canal. Each one will be a long, thin rectangular box. The two walls are identical (actually mirror images of each other).

## The User Interface

If the user resizes the window, the scene will remain centered (same look point and eye point). That will happen automatically. If the user presses the left or right arrow keys, the eye point will rotate left or right, respectively (`eye_yaw`). If the user presses the up or down arrow keys, the eye point will rotate up or down (`eye_pitch`). If the user presses the plus (`+`) or minus (`-`) keys, the eye point will move toward or away from the look point (`eye_dist`).

By moving the eye point around, we may gain a sense of the depth of the object.

## Moving the Eye Point

It is easy to move the eye point. When the left or right arrow keys are pressed, increment or decrement `eye_yaw` by a small amount. When the up or down arrow keys are pressed, increment or decrement `eye_pitch` by a small amount. When the wheel is scrolled, multiply or divide `eye_dist` by a factor close to `1.0f`. Then call `setView()` to create a new view matrix and pass it to the shaders.

## The `setView()` Function

The `setView()` function will be given. It first computes the location of the eye point by using the yaw $\theta$, pitch $\phi$, and distance $d$ values with the formulas

$$\begin{aligned} x &= d\cos\phi\sin\theta, \\ y &= d\sin\phi, \\ z &= d\cos\phi\cos\theta. \end{aligned}$$

Then it passes the eye point, the look point, and the up vector to the `lookAt()` function, which returns the view matrix. The view matrix is passes over to the shader. This function must be called everytime the eye point is moved.

## The Shaders

The shaders will be pass-through shaders. Only the vertex shader will apply the view, model, and projection matrices to the vertex and it will apply the normal matrix to the normal vector (optional for now).

## Program Design

Create a `Wall` class with member functions

```
Wall();
void create();
void draw() const;
```

and with a single data member

```
int m_num_quads;
```

Use a struct `VertexData3D` defined as

```
struct VertexData3D
{
    vec3 pos;
    vec3 color;
    vec3 norm;
}
```

In the `create()` function, assign values to a dynamically allocated array, giving the $x$-, $y$-, and $z$-coordinates of each vertex of the wall (box), the RGB vector for the color (light gray), and the $x$-, $y$-, and $z$-coordinates of the surface normals. The box will have 6 sides (or 5 if you choose to leave off the bottom). The surface normals will point in the positive and negative directions of the three axes, depending on which face they are associated with. Define each face of the wall as a quadrilateral.

The length, width, and height of the wall will be defined by global constants `WALL_LEN`, `WALL_WID`, and `WALL_HGT`. Use realistic values. The wall must be longer than a canal boat, but not much longer, and canal boats were pretty long. The differential in water levels above and below the canal lock was typically 8 feet, so the wall must be higher than 8 feet. The width is the thickness of the wall, enough to keep the wall from collapsing. These constants should be defined globally and then used in the `create()` function. That way, you can change them later without needing to change the function.

At the end of the `create()` function, be sure to delete the array. It will be in the GPU's memory. We do not need to keep in in main memory.

There is also the space between the two walls, `LOCK_WID`, which is just a wee bit wider than the width of a canal boat. Typically, there was a 6-inch clearance on each side of the canal boat.

The `draw()` function will use a `for` loop limited by number of quadrilaterals (`m_num_quads`), which should be either 5 or 6. The value of `m_num_quads` should be calculated with the statement

```
m_num_quads = sizeof(wall_data) / sizeof(VertexData3D) / 4;
```

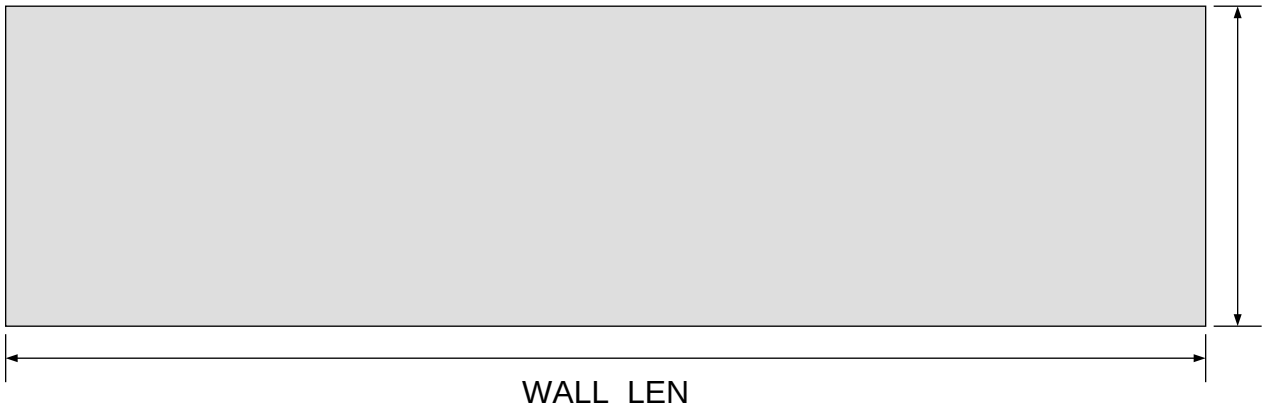
The `draw()` function will include

```
for (int i = 0; i < m_num_quads; i++)
    glDrawArrays(GL_TRIANGLE_FAN, 4 * i, 4);
```

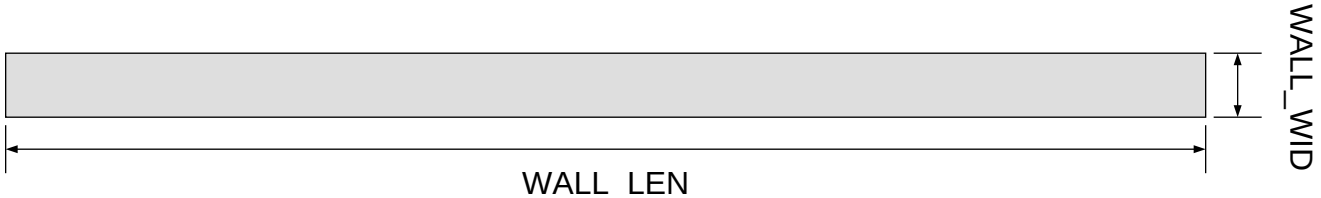
along with a few other things, like moving the object from model coordinates to world coordinates.

## Due Date

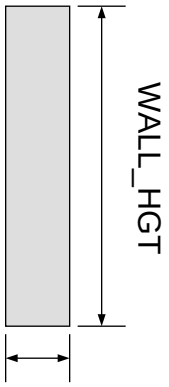
This assignment will not be collected, but you should finish it by Wednesday, October 16.




Side view




Top view




End view


Top view


End view