CCSI 3161 Project – Submarine Simulator

Objectives: To develop a significant OpenGL animation application.

Due date: Dec 3rd, 11:59pm.
No late submission will be accepted since the grades need to be submitted.

Hand in: Electronic submission of entire Visual C project (source, compiled, project files, etc.) using dal.ca/brightspace. Please zip up your whole project directory and submit the zip file.

Note, before beginning your project, please read:

1. The coding style guideline.
2. The policy on plagiarism.

General Comments:

In this project you DO have to handle a window re-shape. You should also use depth buffering and double buffering for your project.

Part A: Basic Scene [20 marks total]:

(i) Basic Scene [3]

In this part you will create a frame of reference.

Draw 3 lines that represents the positive unit vectors in the X, Y and Z directions, with line width = 5. Make the lines colored red, green and blue, respectively. Draw a small white ball at the origin.

Use perspective projection. Handle a window re-size properly.

The user can toggle between wire frame and solid rendering by pressing the 'u' key. The user can switch between full screen mode and windowed mode by pressing the 'f' key. Pressing 'q' quits from the application.

Hints:

1. Use glPolygonMode to switch between wire frame and solid rendering.
2. Use glutFullScreen and glutReshapeWindow/glutPositionWindow to toggle between full screen and windowed modes.
3. You can use gluSphere to make the sphere.

(ii) Submarine [8]

Read in the file 'submarine.obj'. (Courtesy of ModernGnome on Thingiverse licensed under the Creative Commons - Attribution - Share Alike license)
The model will be split into different groups. Each group will start with the line

```
g NAME_OF_GROUP
```

After this it will define the vertices of that group that look like:

```
v 0.242636 0.170825 -0.0272018
v 0.269521 0.170825 -0.0192831
v 0.269521 0.170825 0.195895
…
```

Each 'v' represents a vertex's 3D coordinates. After the vertices, it contains normals that look like:

```
vn 0.242636 0.170825 -0.0272018
vn 0.269521 0.170825 -0.0192831
vn 0.269521 0.170825 0.195895
…
```

After this it will define the faces, or triangles of the object, defining which vertices and normal to use for this face. It will be formatted as the following:

```
f v1//n1 v2//n2 v3//n3
```

The v's and n's refer to the vertices and normals of the triangle (ordered from the start of the text file). For example:

```
f 409//1  1//1  481//1
f 409//2 481//2 2//2
f 481//3 3//3 2//3
…
```

The first line above says to make a triangle out of the 409th, 1st and 481st vertices you've loaded in, and to use the 1st normal for all three vertices. It will keep defining groups in this way, declaring the group name, the vertices, normals and faces until the file ends.

You can view the .obj file in a text editor like NotePad to see this directly.

Apply a yellow material to the submarine. (An example can be seen below).

Notes:

1. You can ignore any line that doesn't start with g, v or f.
2. There are not necessarily the same number of normals in the file as there are vertices. You may need to dynamically allocate more memory as you read in the file.

(iii) Camera Control [5]

Allow the user to control the submarine's movement with the keyboard and mouse. Allow the user to control the direction of the camera/submarine movement with keyboard keys and the mouse. The W and S keys move the submarine forward and backwards.

'W'	:	Move forward
'S'	:	Move backwards
'A'	:	Move right
'D'	:	Move left

The up and down arrow keys control the vertical position of the camera/submarine:

GLUT_KEY_ UP	:	Move Up
GLUT_KEY_ DOWN	:	Move Down

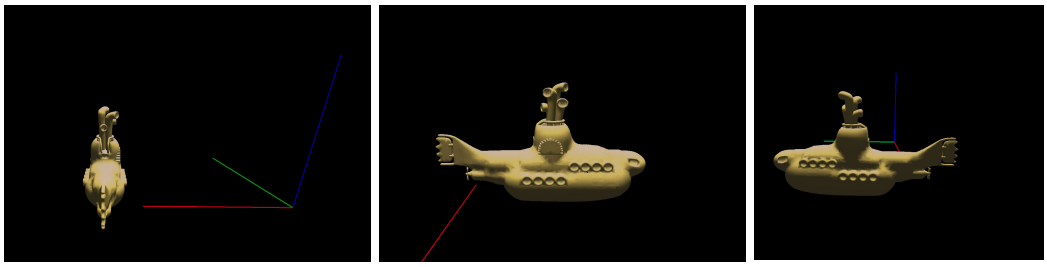
The user can hold down a key to continuously move in a direction, so you might want to use glutKeyboardUpFunc (which tells you when the user has released a key) as well as glutKeyboardFunc (which tells you when a key has been pressed).

Moving the mouse causes the camera to orbit the submarine. It will always stay the same distance to the submarine, and always be facing towards it.

mouse to the right	: rotates the camera towards the right of the submarine
mouse to the left	: rotates the camera to the left of the submarine
mouse upwards	: rotates the camera to the top of the submarine
mouse downwards	: rotates the camera to the bottom of the submarine

The mouse button does not need to be down for this to work.

Use perspective projection with gluPerspective and gluLookAt to position the camera.



Hints:

1. Use glutSpecialFunc to register a callback function for the special keys.
2. Use glutPassiveMotionFunc to handle mouse movement.

(iv) Lighting [4]

Use openGL's lighting model. Use at least one directional light source, with appropriate ambient, specular and diffuse levels. Position the light source so that it is similar to sunlight.

You need to set appropriate shininess, ambient, specular and diffuse material properties for each object. You will have set the normal for each vertex in the submarine.

Hints:

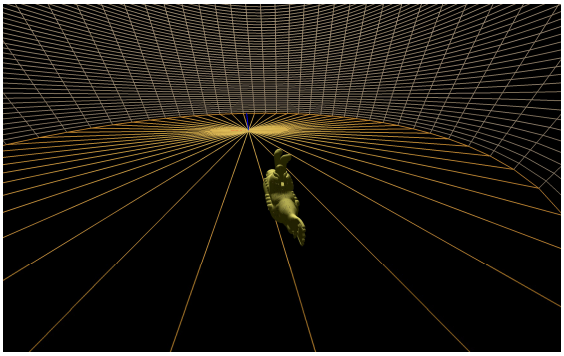
1. Set the light position after calling gluLookAt.

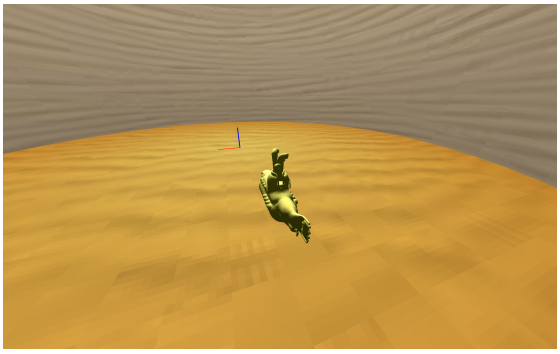
Part B: Sea, Sky and Land [20 marks total]:

(v) Sea & Sky [6]

In this section you will create a texture mapped disk, and a texture mapped cylinder. The disk lies on the ground and is texture mapped with a sand image. The cylinder stands up right and encloses the 'world'. The cylinder is texture mapped with the same sand image. The cylinder will look more like water when we apply fog in the next section.

Your scene should look like this (wire frame and solid):





The disk should be slightly bigger than the cylinder. The cylinder should sit slightly lower than the disk. This will prevent any gaps. Make sure the cylinder is large enough so that you cannot see the top of it (unless you move to the top of the scene) but not so high that the texture is overly stretched vertically.

You can either write your own functions for constructing the cylinder and the disk, or you can use the OpenGL functions that construct them for you. If you use the OpenGL functions, you will need: gluNewQuadric, gluQuadricTexture, gluCylinder and gluDisk.

Use a high emissive term for the lighting of both the sand and the water so that they are both evenly lit.

Turn texture mapping on only for texture mapped objects. Use texture mipmapping.

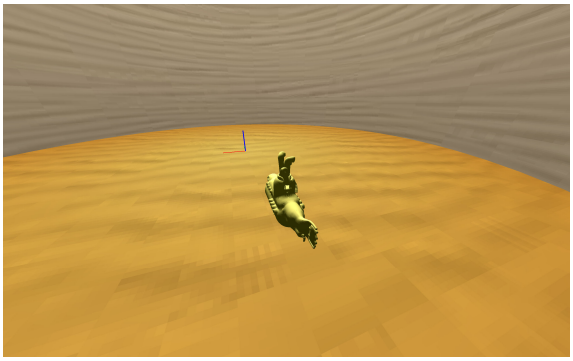
You can also (optionally) find more free textures online. But, if you are using PPM image files, you will need to convert these textures to the PPM format. You can use Paint Shop Pro to do this: www.jasc.com/products/paintshoppro/ .

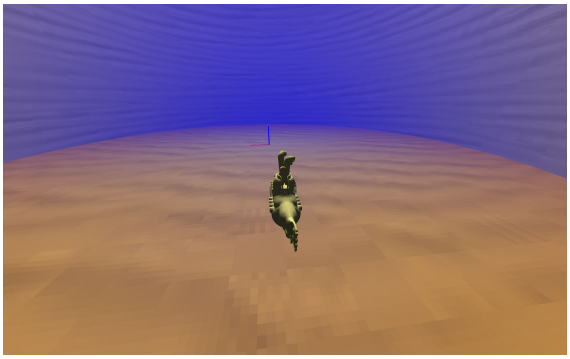
(vi) Fog [2]

We'll be using fog to mimic the effect of being underwater. Use the built-in OpenGL functions to achieve this. Make the fog blue and use an exponential fog function. Make it look like the example below.

The fog is toggled on and off with the 'b' key.

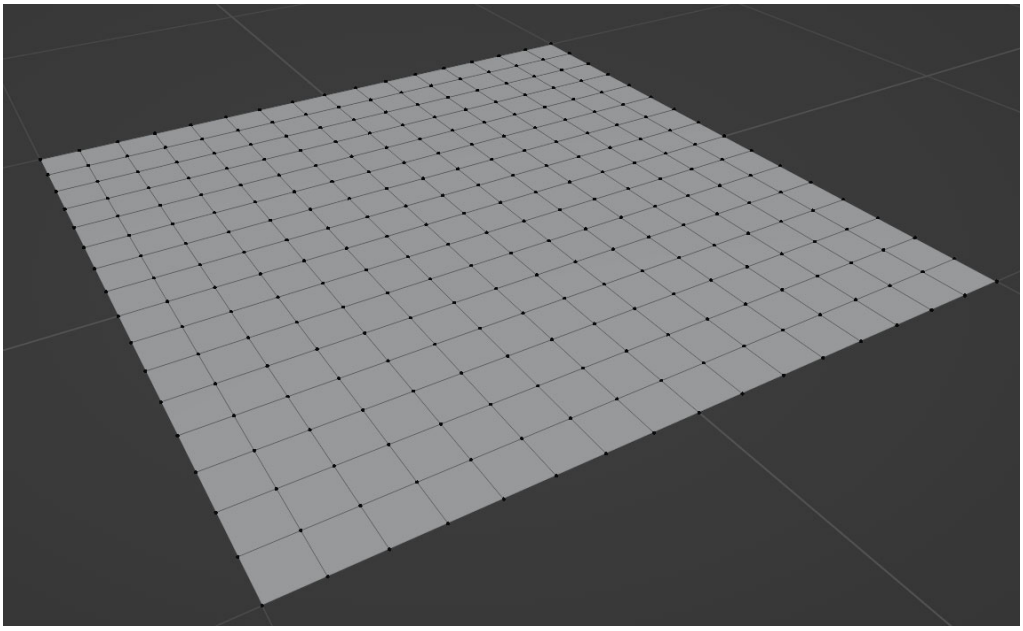
Your scene should look like this (with and without fog):





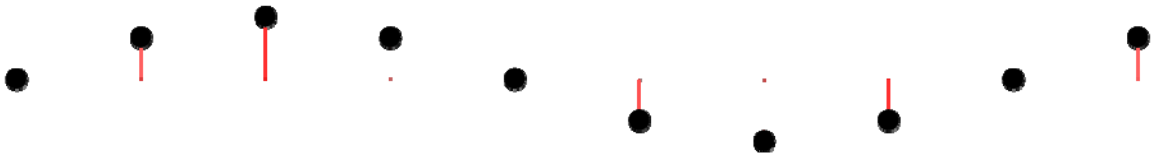
(vii) Surface Water [8]

Create surface water above the scene. The construction of the waves will follow a simple algorithm that first starts with creating a subdivided plane.



In 2D:

At each of these vertices you'll be updating the height based on a sin function.

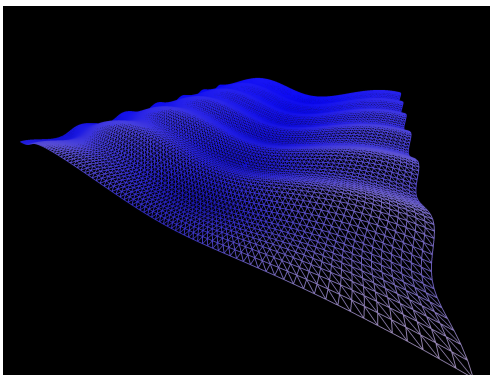


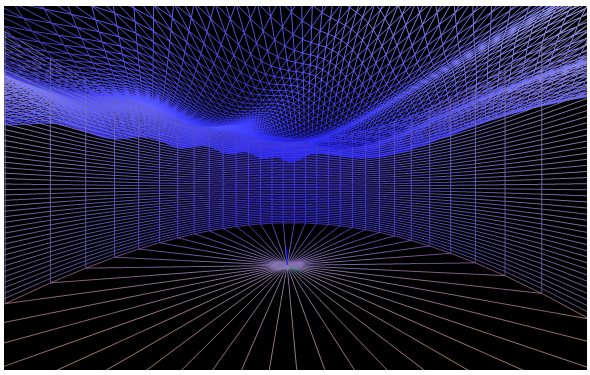
You can animate the waves by changing the input to the sin function based on position and time.

heightAtVertex = sin(valueBasedOnPosition + phase + timeValue) * waveAmplitude

The phase value will shift the wave by a certain amount, and waveAmplitude governs how tall or short the waves will be. valueBasedOnPosition tells the direction of the wave. Using the X position would have the wave travel in the X direction, X+Z would move it diagonally, etc. Try adding multiple different sin functions with differing amplitudes, directions and phases to make the waves more convincing. Place the water on top of your cylinder so it looks like it is the surface water above your scene.

Your water should look something like this (wire frame):





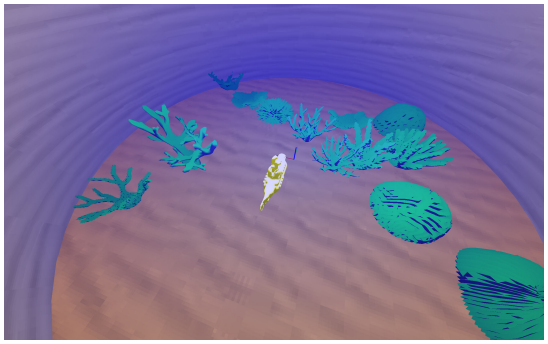
Part C: Coral and Fish [20 marks total]:

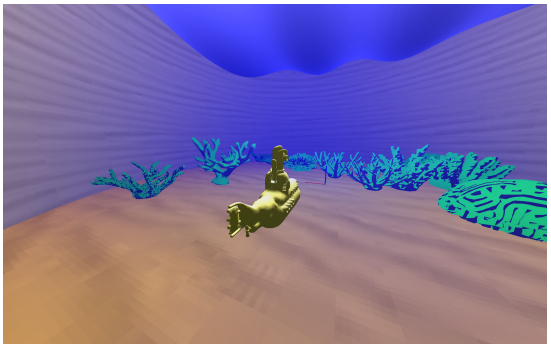
In this section you'll be adding some life to your scene through coral and moving fish.

(viii) Coral [5]

In the project download you'll find a folder of OBJ files. These files are in the same format as the submarine in Part A. Each one of these contains a model of a different type of coral. Place these randomly throughout your scene. Apply a material that colours them green.

All the coral models are courtesy of marcusnystrand on Thingiverse licensed under the Creative Commons - Attribution - Share Alike license. And have been decimated down to ~1000 triangles using Blender.

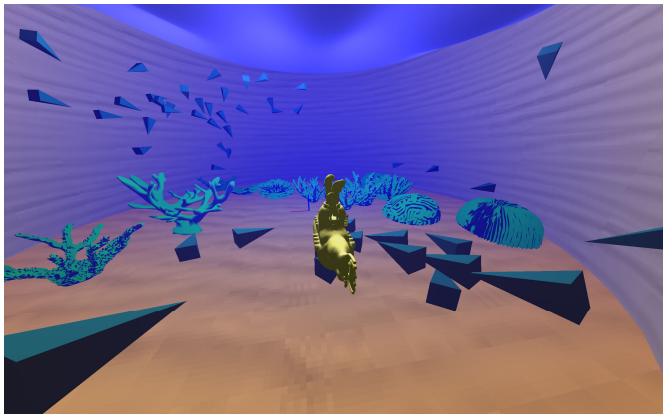




(ix) Fish [15]

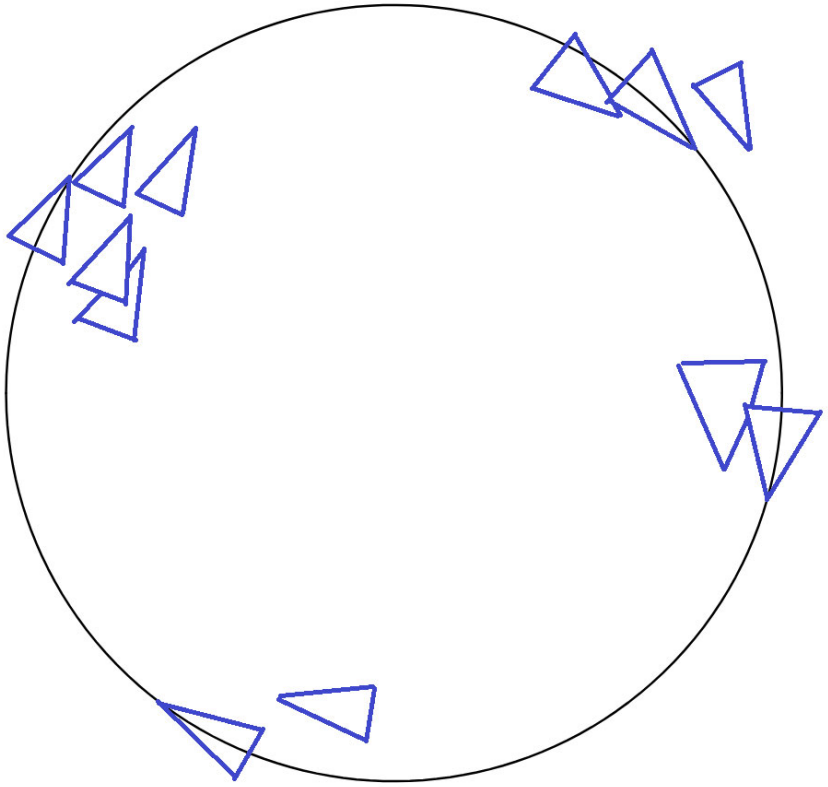
In this section you **only pick one** of three methods A, B and C with each increasing in difficulty. **Method C will give you 4 bonus marks. (This can't bring you above 100% but may fill in for other areas).**

In all methods you will need to create some fish that swim around the scene. As a fish model you can use a square base pyramid. Include at least 10 fish in the scene. (The example picture below uses method C).



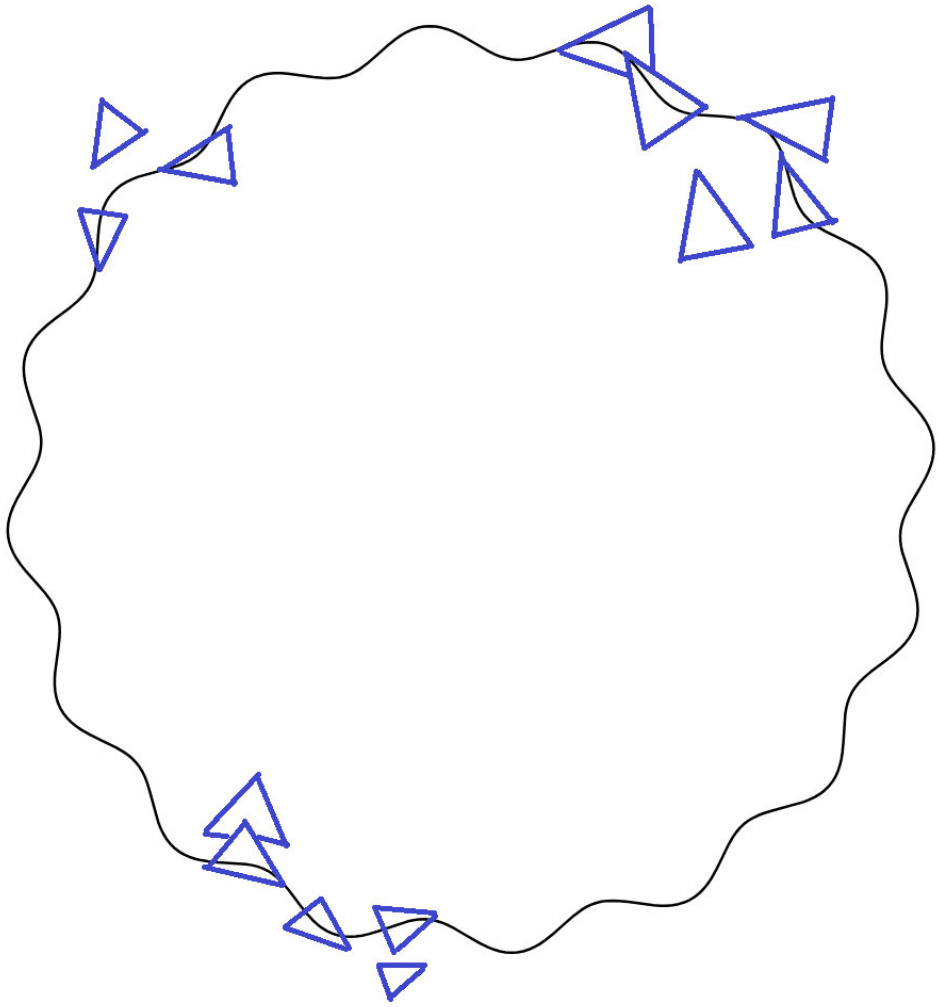
A) Circling [10/15]

Have the fish swim in a circle around the scene in small groups. From the top down, the general path they take will look like a perfect circle. Vary the height of the groups.



B) Swimming [15/15]

Have the fish flutter while moving in a circle in small groups. From the top down it will look like a wavy circle. Vary the size, and speed of the fish. Vary the height of the groups.



C) Flocking [19/15]

Extend your boid behaviour from Assignment 1 to 3D. The fish only need to avoid the “walls”, the surface water and the ground. They do not need to avoid the submarine or the coral.

Additional Enhancements [4]:

Add some additional feature(s) to your project. They will have to be something *very* special for full marks. **Add a readme file that states exactly what your inventions are and how to use them.**

Print out Keyboard controls [0]

Use printf to dump out a listing of all the keyboard key functions that you are supporting. Get the keyboard controls working.
Include what operating system you want the assignment to be marked on in the print out.