ECE 4122/6122 Final Project

(300 pts)

Section	Due Date
89313 - ECE 4122 - A	Dec 3rd, 2019 by 11:59 PM
89314 - ECE 6122 - A	Dec 3rd, 2019 by 11:59 PM
89340 - ECE 6122 - Q	Dec 3rd, 2019 by 11:59 PM
89706 - ECE 6122 - QSZ	Dec 3rd, 2019 by 11:59 PM

Notes:

You can write, debug and test parts of your code locally on your personal computer. However, the code you submit must compile and run correctly on the PACE-ICE server.

Submitting the Assignment:

See Appendix C.

Grading Rubric

AUTOMATIC GRADING POINT DEDUCTIONS :

Element	Percentage Deduction	Details
Does Not Compile	40%	Code does not compile on PACE-ICE!
Does Not Execute as expected and/or Output is wrong format	10%-90%	The code compiles but does not produce correct outputs or does not execute as expected. UAVs flight paths are incorrect. UAVs shape/color is wrong.
Clear Self-Documenting Coding Styles	10%-25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

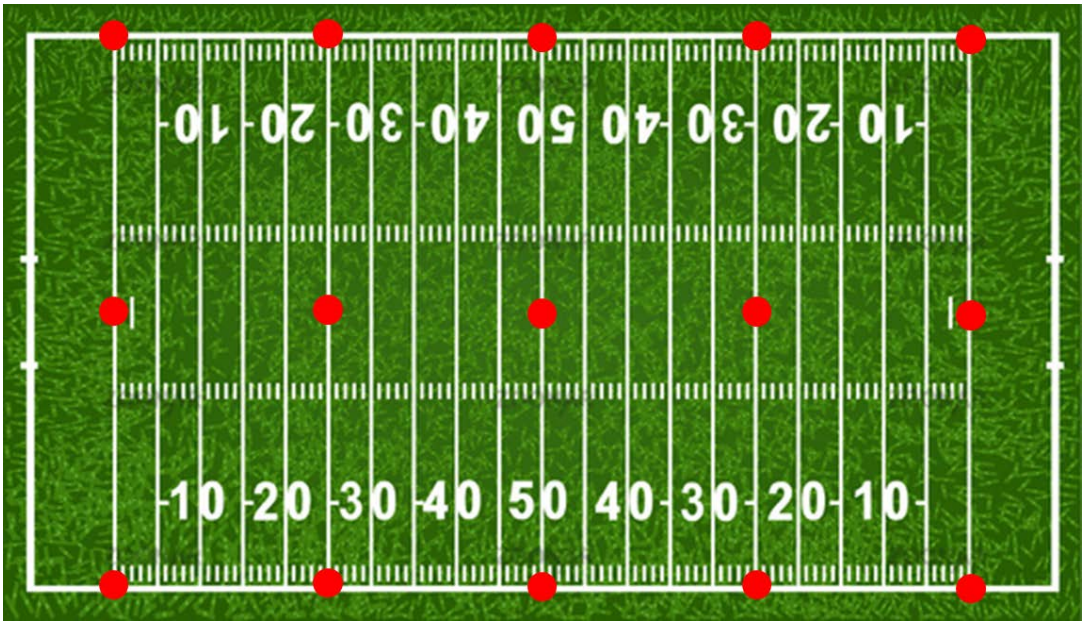
Element	Percentage Deduction	Details
Late Deduction Function	score - (20/24)*H	H = number of hours (ceiling function) passed deadline note : Sat/Sun count as one day; therefore H = 0.5*$H_{weekend}$

GaTech Buzzy Bowl

The company you work for has been selected to develop a half-time show using UAVs. You need to develop a 3D simulation using MPI and OpenGL to demo the show to the game organizers for their approval.

Below is a description of the show that you will be creating with a 3D simulation.

1) The show is made up of 15 UAVs that are placed on the football field at the 0, 25, 50, 25, 0 yard-lines as shown below by the red dots.



2) The UAVs remain on the ground for 5 seconds after the beginning of the simulation.

3) After the initial 5 seconds the UAVs then launch from the ground and go towards the point (0, 0, 50 m) above the ground with a maximum velocity of 2 m/s

4) As they approach the point, (0, 0, 50 m), they began to fly in random paths along the surface of a virtual sphere of radius 10 m while attempting to maintain a speed between 2 to 10 m/s.

5) The simulation ends once all of the UAV have come within 10 m of the point, (0, 0, 50 m), and the UAVs have flown along the surface for 60 seconds.

6) Each UAV has the following
    a. Each UAV has a mass of 1 kg and is able to generate a single force vector with a total magnitude of 20 N in any direction.
    b. Each UAV it just small enough to fix in a 1-m cube bounding box.
    c. The color of your UAV is determined using the first letter of your **last** name:

Red	A,K,T
Green	B,L,U
Blue	C,M,V
cyan	D,N,W
magenta	E,0,X
yellow	F,P,Y
Red	H,Q,Z
Green	I,R
blue	J,S

    d. The shape of your UAV is determined by the first letter of your **first** name:

Sphere	A,K,T
Cube	B,L,U
Cone	C,M,V
Torus	D,N,W
Dodecahedron	E,0,X
Octahedron	F,P,Y
Tetrahedron	H,Q,Z
Icosahedron	I,R
Teapot	J,S

7) You must develop a MPI application using 16 processes. The main process (rank = 0) is responsible for rendering the 3D scene with the 15 UAVs and a green (RGB=(0,255,0)) rectangle representing the football field in a 400 x 400 window. You will get 5 extra bonus points for using a bitmap file called **ff.bmp** in the same location as the executable to apply a football field texture to the rectangle. The other 15 processes are each responsible for controlling the motion of a single UAV.

8) The main process gathers the location and velocity vector of each UAV every 100 msec and immediately broadcasts this information to all the UAV processes, and updates the 3D scene.

9) The coordinate system of the 3D simulation is define as flows: The origin is location in the center of the football field at ground level. The positive z axis points straight up, and the x axis is along the width of the field and the y axis is along the length.

10) A camera location, orientation, and field of view should be used so that the whole football field and the virtual 10m sphere is in the view.

11) The flight of the UAV is controlled by the following kinematic rules
    a. The force vector created by the UAV plus the force of gravity (10 N in the negative z direction) determine the direction of acceleration for the UAV.
    b. Use Newton's 2nd Law to determine the acceleration of the UAV in each direction.

$$F_{x,y,z} = m * a_{x,y,z}$$

    c. Use the equations of motion for constant acceleration in each direction (xyz) to determine the location and velocity of the UAV every 100 msec.

$$x = x_o + v_{xo}t + \frac{1}{2}a_x t^2$$
$$v_x = v_{xo} + a_x * t$$

    d. You can use any method you want to maintain the UAVs flight path along the surface of the 10m radius virtual sphere. One possible method to consider is to use a variation of Hooke's Law simulating a virtual spring between the surface of the sphere and the radial distance of the UAV from the center of the sphere.

$$\boldsymbol{F}_s = k(10 - D)$$

    where D is the distance from the UAV to the center of the sphere. And the direction of F_s is along the normalized vector from the UAV to the sphere center. The value of k can be varied to bind the UAV tighter to the surface.

12) If the bounding boxes of two UAVs come within 1 cm of each other an elastic collision occurs. For simplicity we are going to model the UAVs as point objects and the UAVs will just swap initial velocity vectors for the next time step:

$$v_1 = v_2$$
$$v_2 = v_1$$

ECE6122 Students:

The magnitude of the color of your UAV should oscillate between full and half color throughout the simulation. The RGB values of your color should decrease by one every 20 time steps until it reaches 128 and then increase by one every 20 time steps until it reach 255. Example below:

Red : (255,0,0), (254,0,0)…(128,0,0), (129,0,0)…(255,0,0)

Compiling and running your code:

Compile your code using

```
>module load mesa gcc mvapich2
```

```
>mpic++ *.cpp –lGLU –lglut –std=c++11
```

To run our code you will need to create an interactive session with 16 processes either using the remote VNC setup or using an X11 server as covered in class.

In the folder where your executable is located, execute the following commands:

```
>module load mesa gcc mvapich2
```

```
>mpirun -np 16 ./name_of_your_exectable
```

In a few seconds, a window should appear displaying your OpenGL rendering.

Appendix A: Coding Standards

Indentation:

When using *if/for/while* statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: <your name>
Class: ECE4122 or ECE6122
Last Date Modified: <date>

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.

Appendix B: Submitting Assignment

Step-By-Step Submitting a Tar.gz

PLEASE NOTE: All the following instructions are run on PACE. If you are struggling with any of the steps, please come see the TAs during office hours. Better to start and test early than wait until the last minute.

Below is a step-by-step tutorial on how to create the .tgz file for the Final Project for ECE 4122/6122. Additionally, if you do not use a header file for a solution, you do not need to submit a header file. Please adjust these instructions accordingly. The tarball should contain a folder with your buzzid containing a subfolder for the problem with the corresponding .cpp, .h files, and any texture files.

Please make sure that the problem is in a subfolder with the naming conventions:
<FirstName_LastName>_FinalProj

Create a subdirectory named *buzzid* in the current working directory by executing the following command in the shell:
$ mkdir *buzzed*

Create a text file named **manifest.**

Please make sure that the **manifest** file is a plain text file and not a rich text file. Microsoft word will generate a rich text file. The easiest method to create a plain text file is to use a command line editor such as vi, vim, or nano.

(If you want a tutorial, these can be useful: vi: http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Editors/ViIntro.html, vim: https://openvim.com/, nano: https://www.howtogeek.com/howto/42980/the-beginners-guide-to-nano-the-linux-command-line-text-editor/)

Populate the **manifest** file with the following:

buzzid/<FirstName_LastName>_ FinalProj /*.cpp
buzzid/<FirstName_LastName>_ FinalProj /*.h
buzzid/<FirstName_LastName>_ FinalProj /*.bmp

(***PLEASE NOTE***: this is subject to change with depending on your class section. The wildcard (*) character will grab all the .cpp and .h files you generated for each problem.)

Now you need to construct the correct file structure for the submission. This means that you need to put all the homework files into your *buzzid* folder. Execute the following lines in the shell:

$ cp -a <FirstName_LastName>_* *buzzid*

Many people have had issue with this step. You need to make sure the naming conventions are consistent to avoid potential problems.

It is now time to tarball your submission. If all the other steps have gone smoothly execute the following command:

$ tar -zcvf *buzzid*-FinalProj.tgz $(cat manifest)

You can now check the new tgz file just generated with the following command:

$ tar -ztvf *buzzid*-FinalProj.tgz