# ECE 4122/6122 Final Project: Sonic 3D Map
## Custom Class & OpenGL
(300 pts)
*Due*: Tuesday Dec 6th, 2022 by 11:59 PM



Write a C++ application that uses a custom class with OpenGL and ASSIMP to load and display the attached OBJ file (***map_static.obj***) with multiple meshes. You do not need to implement any reflections, but your lighting and camera view should be similar to the image above.

1.  (180 pts) You need to create an application using a custom class called ECE_OBJ_Loader to load and display all the textured meshes within the file. Your class needs to have the following public methods:

    o  bool loadFile(const std::string &filename);
    o  void renderMeshes();

2.  (30 pts) The camera view should always point towards the center of the model.
3.  (30 pts) Pressing the up/down arrow keys should zoom in and out.
4.  (30 pts) Pressing the left/right arrow keys rotate either the camera view or the model left and right.
5.  (30 pts) Pressing escape key ends the application.

You are free to use third party libraries such as https://freeimage.sourceforge.io/ for loading the textures. Other than ASSIMP you cannot use a third part library to load/render the OBJ file.

<u>**Extra Credit (up to 20 pts) (TA's discretion)**</u>

Place the rings, sonic, jump pad, Dr Egghead, and ball along the model as shown above.

# Turn-in Instructions:

Two methods:
1.
   a. Upload a video of your application running and demonstrate the requirements above.
   b. Put <u>all the code files you created</u> into a zip file called ***FinalProject.zip*** and upload to canvas.
2.
   a. The TAs will build and run your code.
   b. You can use the tutorial09_Assimp example from class to develop your code. Place your new source code files in a subfolder called *code*. Once you have finished your development, zip the tutorial09_Assimp folder with your changes into a zip file called ***FinalProject.zip*** and upload to canvas.

# Grading Rubric

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA's will **randomly** look through this set of "perfect-output" programs to look for other elements of meeting the lab requirements. The table below shows typical deductions that could occur.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

| Element | Percentage Deduction | Details |
|---|---|---|
| Does Not Compile | 30% | Code does not compile on PACE-ICE! |
| Does Not Match Output | Up to 100% | The code compiles but does not produce the correct outputs. See point values above. |
| 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 – 2 * H | H = number of hours (ceiling function) passed deadline |

# 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 (section)
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.