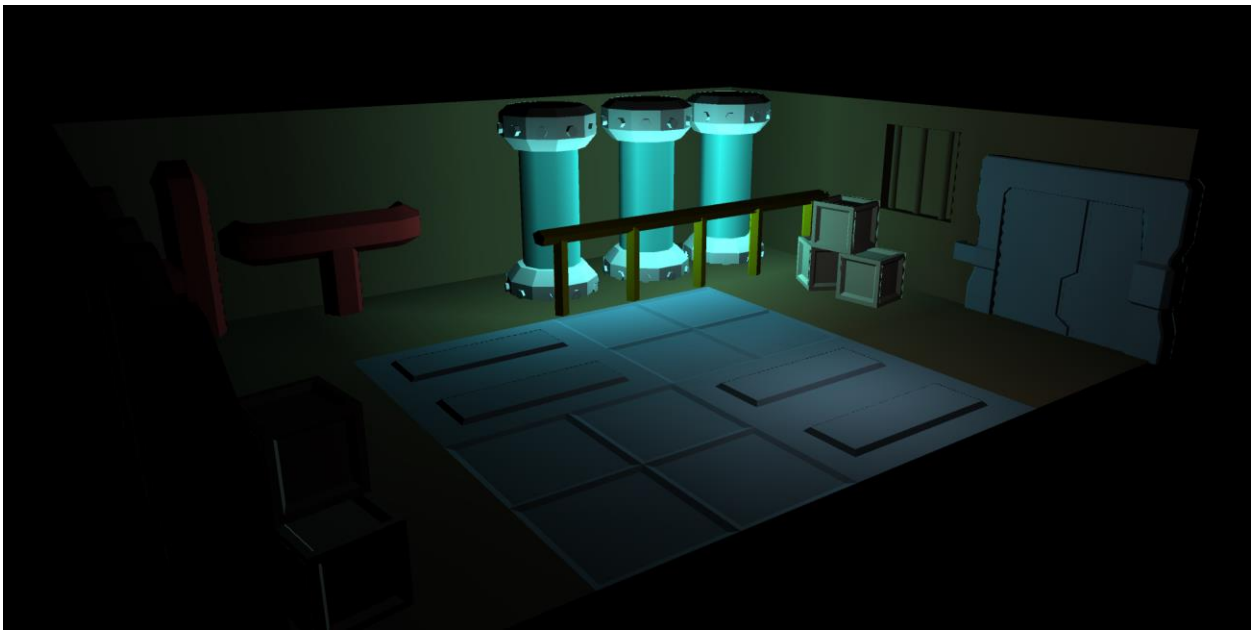
Project 5 – 3D Scene Loading



Overview

In this project you are going to load and ultimately render data that is read from several scene files. A "scene" is just a group of data necessary to render a particular thing—this could be a level in a video game, a menu screen, or maybe a shot from an animated movie.

Description

You are going to do some simple file I/O operations to read data from several scene files. These files will contain:

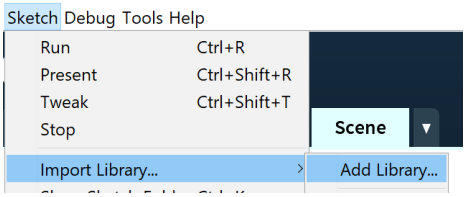
1. The names of mesh files to load (and use to create PShape objects)
2. The world position of those meshes, and the color of each PShape
3. The location and color of various point lights

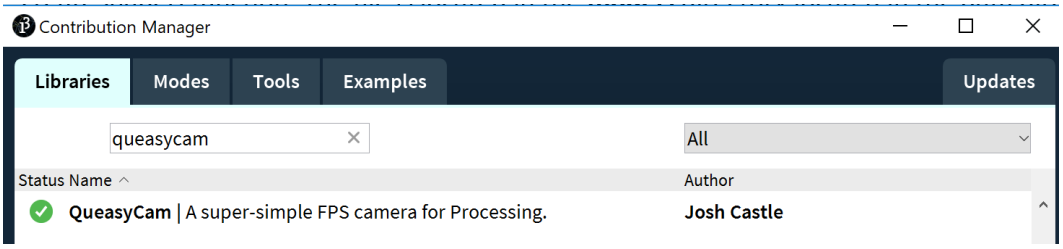
The shell of the program is going to be set up for you, you're just going to handle the file loading and rendering part.

After creating (and saving) a Processing sketch, create a folder called **data** and, from the assignment's .zip file, place **models** and **scenes** in there. Also, you can use the provided program template to get started.

Camera

The camera in this program is already implemented; it's a library for Processing called QueasyCam. Click on Sketch->Import Library->Add Library… and then search for queasycam. Click Install, then you're all set.





The sample code for this project already contains the references to this library, you just need to make sure you've installed it. It's a simple FPS (first person shooter) camera, and the controls are the following:

Right-Mouse button – click and hold to "turn on" the camera, allowing movement and changing the camera's orientation
WASD keys – move forward, backward, left and right
QE keys – move the camera up and down

Warmup

For simple programs you can hard-code values that you need such as file names, positions of objects, etc. When there is a lot of data, however, it makes sense to store and load that data from a file. The details of the scene files are described later in this document. For

Here's a warmup for this assignment. If you open up **scenes/testfile.txt**, it will look like this:

```
text,-300,0,75,255,255,255
-150,100,50,200,255,200
```

The first line in the file represents 3 pieces of information:

1. The name of a model that you are going to load. This is a .OBJ file that you will load using createShape() function. The file is located in the **models** directory located in the data directory of your project, so the full path will you will open is **"models/text.obj"** (Processing looks in the data directory by default, so you can omit that).
2. A X,Y,Z location where to draw the model that was loaded
3. An RGB color

The second line is 2 pieces of information about the lighting in the scene:

1. The XYZ position of a point light
2. The RGB color of a point light

You should write a simple test to read that data from the file, and using the values contained within, load a shape, draw it in the correct position with the appropriate light set. You'll need the createShape() function for loading, and then the shape() and pointlight() function to render. You'll definitely know if you've done it correctly; you should see a 3D shape rendered with lighting, like the following image:

Hello World, scene edition

Reading External Files

File I/O isn't terribly complicated, but Processing makes it even easier with the `BufferedReader` class.

(Processing reference of BufferedReader)

With this class you'll open a file and read it one line at a time to extract data from the file. After you've gotten one or more lines, what you do with that data is up to you.

```
BufferedReader reader = createReader("someFileToOpen.ext");

try
{
    // Example: Get an entire line from the file
    // What you do with this line after is up to you
    line = reader.readLine();

    // Get the next 4 lines, one at a time
    for (int i = 0; i < 4; i++)
        line = reader.readLine();
}
catch (IOException e)
{
    // In case something went wrong during the process
    e.printStackTrace();
}
```

After getting a string from the file, splitting it into separate pieces can be done with the .split function, passing it a comma as a delimiter. The function returns an array of strings, which you can then convert one at a time to whatever type of data you need. For example, if you had a string with a single integer and a float separated by a comma, you could do this:

```
String example = "104,3.14";
String twoValues[] = example.split(",");
int valueOne = Integer.parseInt(twoValues[0]);
float valueTwo = Float.parsefloat(twoValues[1]);
```

Scene File Format

The scene files that you will be loading in this assignment will have the following format:

1. An RGB color for the background color of the scene
2. A number indicating how many meshes are referenced in the scene
3. For each of those meshes, you will have to read:
    a. A mesh name – a string that is the name of the .obj file you have to load and later draw
    b. 3 floats for the x, y, z position of the object
    c. 3 integers representing the color of the mesh in RGB format
4. A number representing how many lights are in the scene
5. For each light, you will read
    a. A position (x, y, z value)
    b. 3 bytes for the color (r, g, b) value

Lights

([Processing reference of pointLight])

For this project you will use a type of light called a **point light**—a simple light with a position and a color. In Processing these are very easy to use, with the **pointLight()** function. This function takes 6 parameters: the first three are for the r,g,b color of the light, and the next three are for x,y,z position of the light. Any lights that you want to when rendering something must be specified before drawing the object, so the steps would be something like this:

```
pointLight(rgb,xyz); // Set a light
pointLight(rgb,xyz); // Set a second light
//etc…
pushMatrix();
translate(200, 40, -23);
box();
popMatrix();
sphere();
shape(); // Draw objects using previously set lights
```

Data Storage and Drawing Shapes

You should at a minimum create a Scene class in order to store and draw all the data for one scene. That class will need to store information about:

Multiple PShapes and their respective positions
Multiple lights, including their positions and colors
The background color

How you choose to store this is up to you, but you might find it helpful to create classes for things like a meshes and lights.

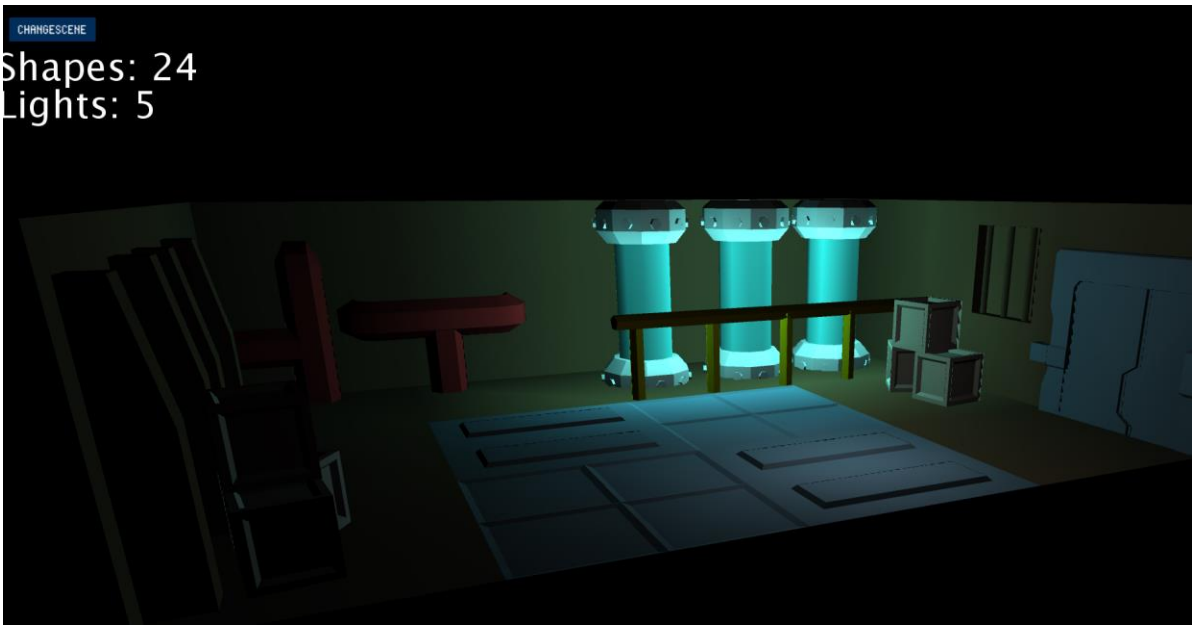
You should be able to switch between the two scenes using the ControlP5 button in the top corner of the program. Each scene should be self-contained, and a call to the DrawScene() function from within the main draw() of your program should show everything.

Because each PShape will have its own translation, remember to use pushMatrix() and popMatrix() to ensure proper transformations.

Final Results

After you've loaded the scene(s), they should look like this:

Scene 1



Scene 2

Submissions

Create a .zip file with any code files you created for this project (in Processing they are files with the extension .pde), and name the file ***LastName.FirstName.Project5.zip***. Submit the .zip file on the Canvas page for Project 5.

Tips

- Load the test file first to make sure you can get a mesh and a light loaded from a source other than hard-coding it

Grading

Item	Description	Maximum Points
Scene models loaded and rendered	All models present in a scene, in the correct position	60
Lights loaded and used	All lights loaded and used	60
Object-oriented code	Scene class implemented	40
Scene cycling	Can switch between two different scenes	40
	Total	200