COMPUTER GRAPHICS PROJECT

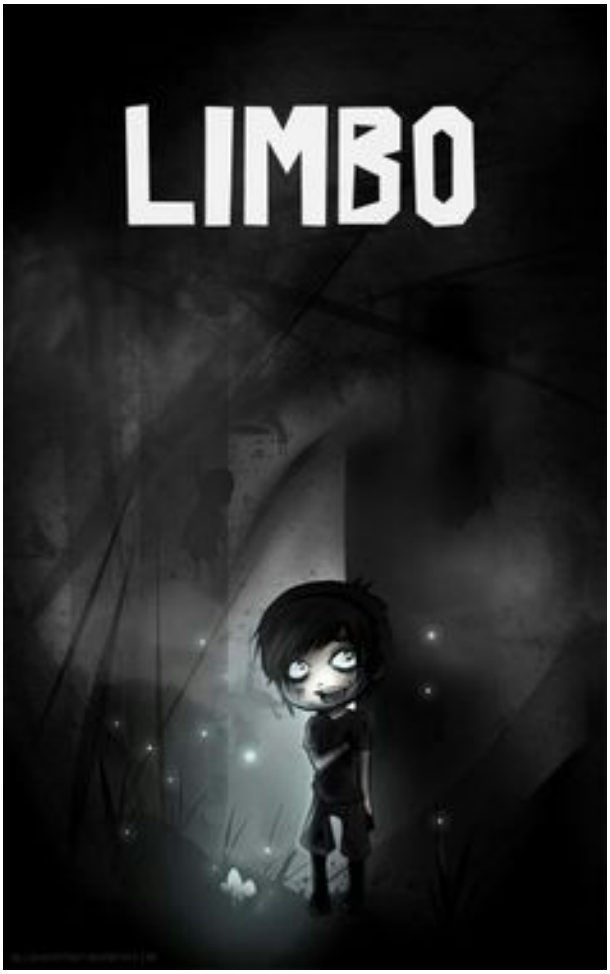
Francis Cote-Tremblay

26615287

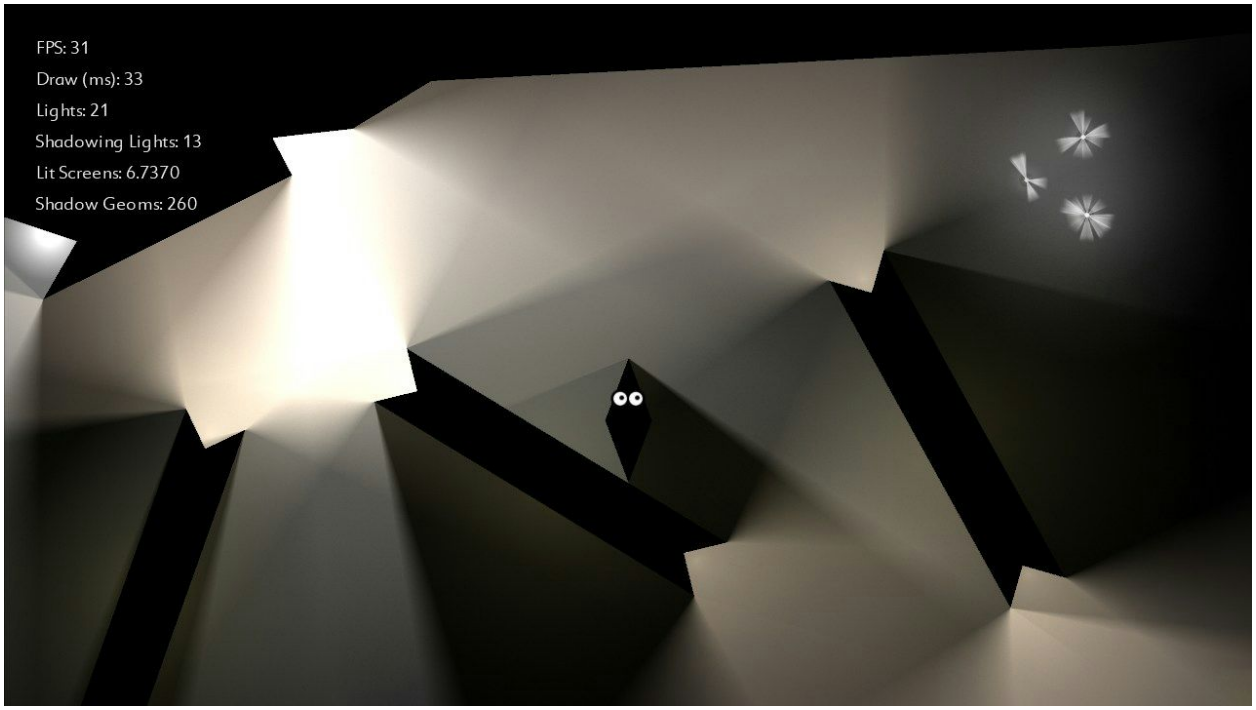
MY GRAVITATIONAL SYSTEM

I took this class mainly because I like to try indie games and compare them based on their language and graphics language. Before this class I imagined that producing video games was a fairly easy task and I did not know how much knowledge was associated to computer graphics. Throughout the semester I began to comprehend how huge is the amount of work to build a game from scratch. Because of this reason I miscalculated the amount of work needed to build a game and that is why I had to diverge to a simpler project near the deadline. My initial objective was to implement a game that would take advantage of a particular lighting engine. I wanted to build a small game with similar visuals as Limbo and Fez. What I had in mind was to create a 3D world that you could only rotate by 90 degrees therefore allowing the player to see previously hidden features of the level but still in a restrictive way. The player would have to get through some obstacle to get to the end of the level. He would have had to use the light and the world rotation to overcome obstacles. At that time I thought that the physics were handled by the graphic language and that we did not even have to

care about it. This is another reason why I changed mind about my game. I would have had to implement a physic engine and this is not in this course's scope.



You can see from the pictures above that each game have a different style. I wanted to use the geometry concept of Fez with the shade of colors of Limbo. The lighting effects in Limbo are simple but still really interesting. I wanted to have a sharp shadow effect similar to this kind of outcome:



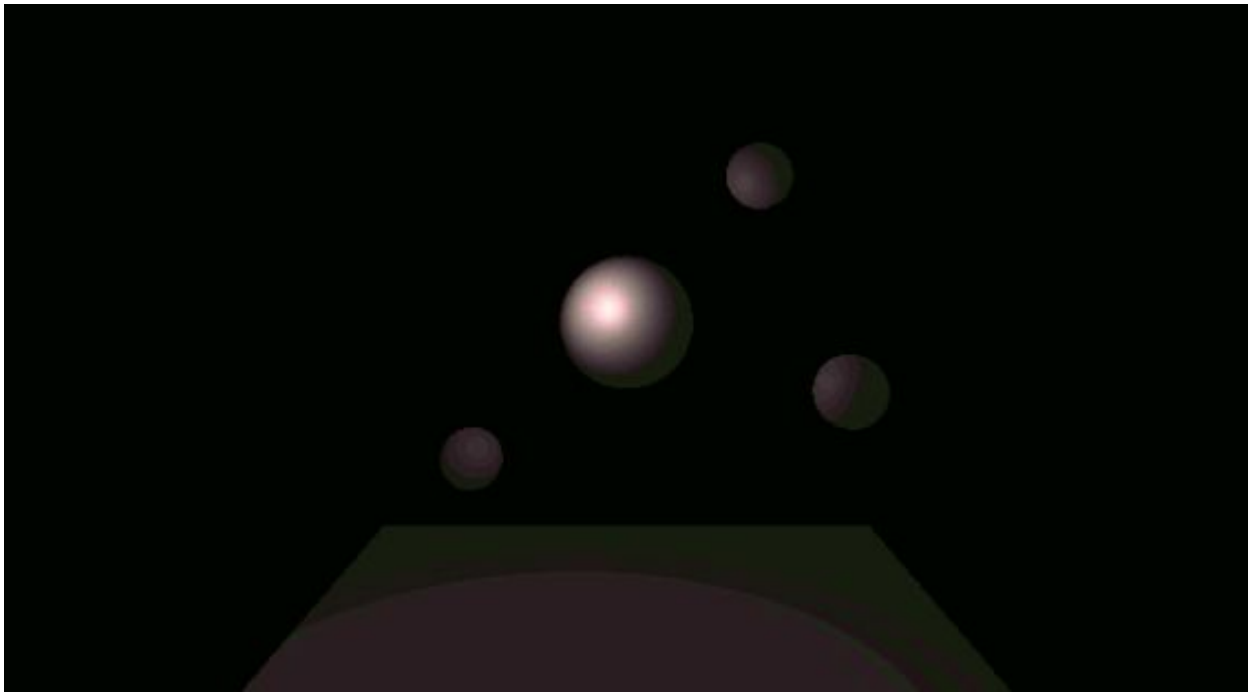
I wanted to have somehow sharp shadows will an unrealistic effect like this.

I followed this project idea until two weeks before the due date and figured out I had to change my subject to something else that would still use what I had programmed already.

I used an object oriented approach and used inheritance to write my software in an organized way. I have a total of 15 classes. I used the factory pattern and the facade pattern to keep my code structured. I am really happy about the architecture of my code. The main function did not overflow with code as I was using many objects to delegate the tasks. The most important classes of my projects are the Shape class and World class. Most of the drawing and transformation logic is contained within these classes. The World class act like a facade. Mostly everything is

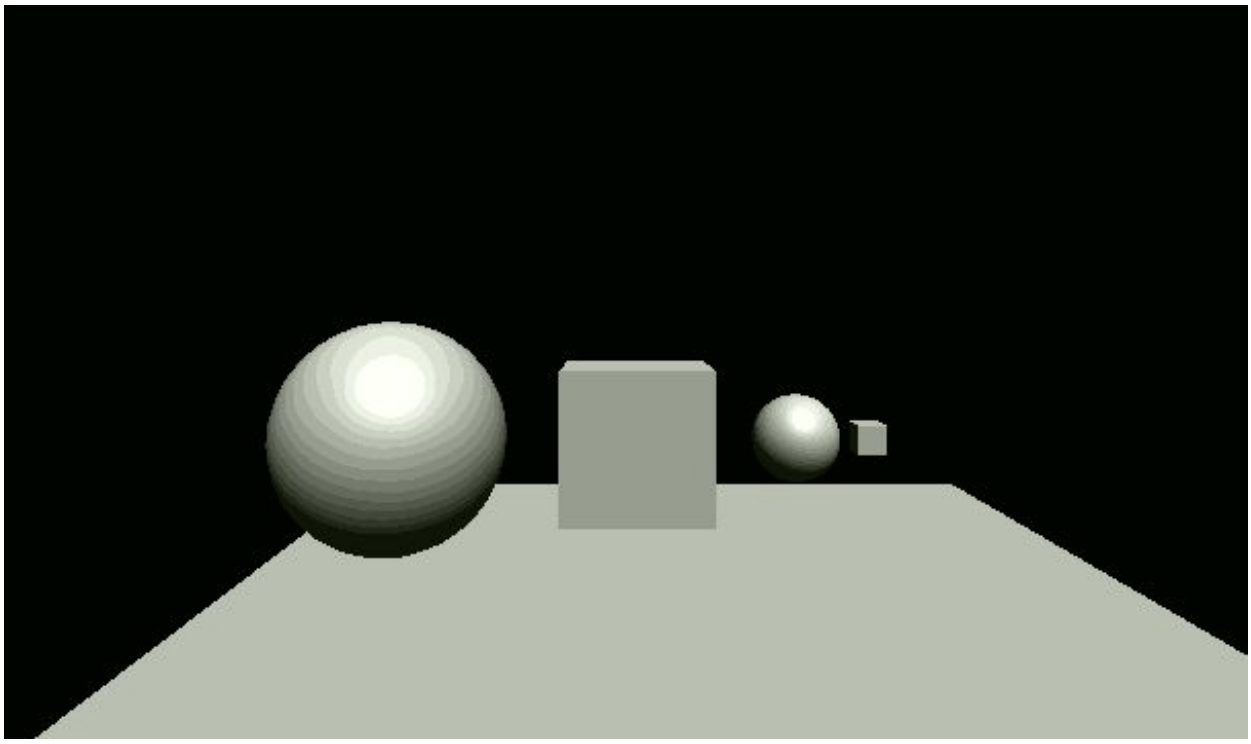
wrapped behind my World object. This way it abstracts the implementation details from the driver (the main function). The world also contains a list of shapes which can be either a cube or a sphere. I used a list of pointers of type Shape to handle both Spheres and Cubes. Although I did not use any cubes in the final version of my scene I still think that abstracting Sphere and Cube to a Shape class is a good practice. It makes the code more understandable and provide flexibility for future improvements.

I first began by implementing the light effect from tutorial 8* from opengl-tutorial.org. It is a really simple light effect that does not take into account any shadow maps. All object illuminations and shadows are independant from each other. Since at first I was making a game, I wanted my the player to be able to control the light so I mapped the mouse to determine the light position and the mouse scroll wheel to the light's position on the z axis. I used a simple approximation to estimate the coordinates from the pixel space to the object space. Here is how the first lighting I implemented looked like:



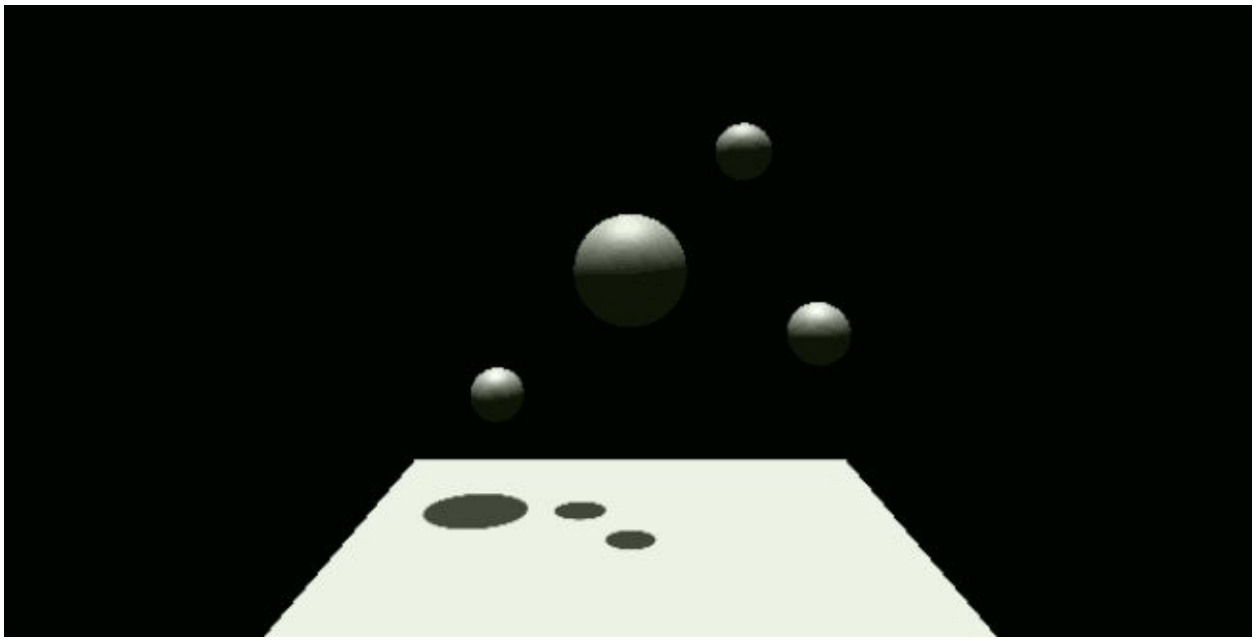
At this point I had implemented a few other things for my game such as the 90 degree world rotation. I used a timer to make the animation smooth. I also wanted to be able to build the level on runtime for debug purpose so I designed a system that could create a cube on left mouse click and a sphere on right mouse click. The size of the object is defined by how long you keep the mouse button pressed. The initial position of the object is estimated from the pixel coordinates. For example, this was

generated at runtime:



I did not know that it would provide an additional feature later on for my gravitational system. Now upon generation the new object automatically begin to orbit around the player or in the case of a gravitational system the nucleus, or sun or whatever the kind of gravitational system we are talking about. The player class was originally supposed to represent a player object but now it represent the referential point for other objects to rotate around. I left the Player class name as Player because my project does represent a general gravitational system so the core object has no name in particular but is still considered the main object in the scene, just as a Player in a game.

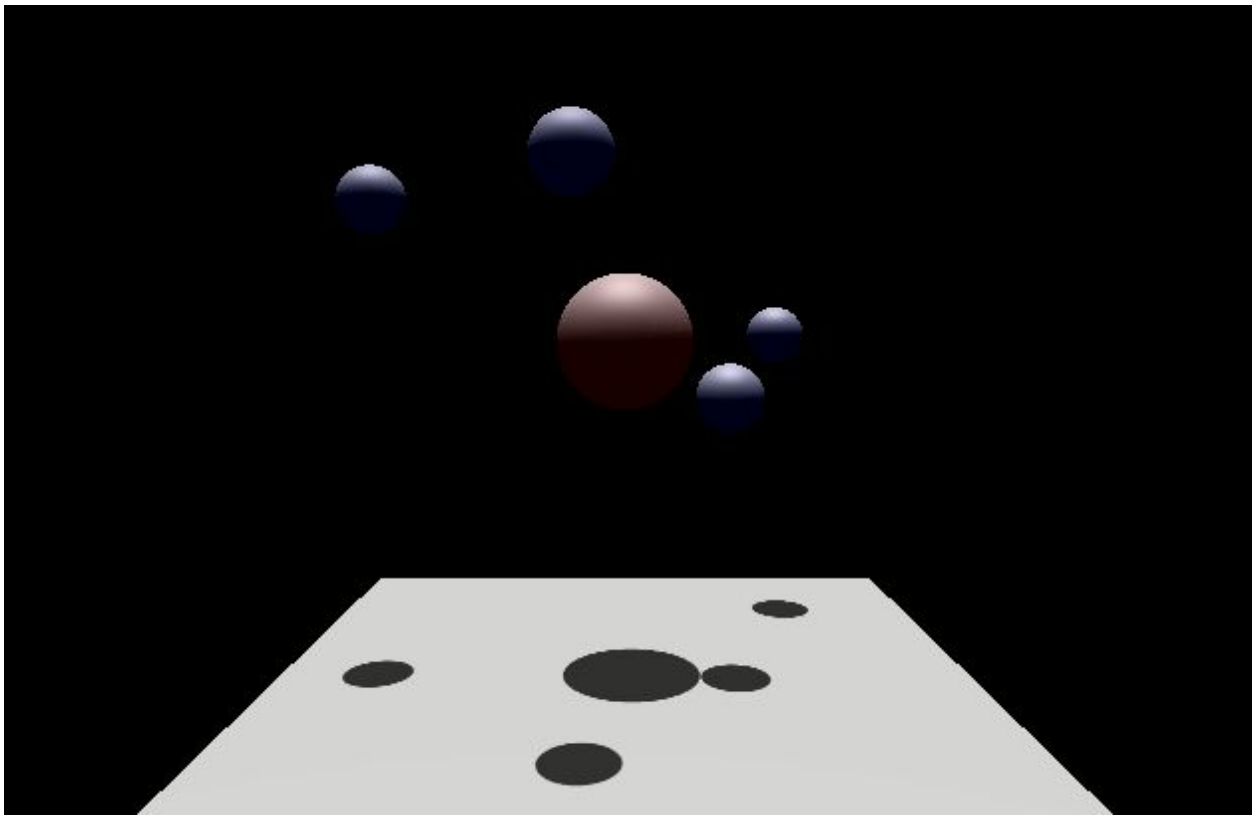
I was not satisfied with the lighting effects so I decided to learn about shadow maps and shader programming. I learned how how they works and how to use them. I based myself off of tutorial 16** from opengl-tutorial.org. It was considerably more complicated than I expected. I also had to learn about depth texture. It found that this type of lighting was really rigorous in its code compared to my first light implementation. Additionally I found that it was also rigorous in the way objects and shadows have to get through the pipeline. It happened multiple times that I was stuck because of a misplaced line of code. I consider that shadow maps are very hard to understand and implement properly and I feel like I still do not understand it completely. But after some hard work I managed to make it partially work and to obtain something. But however still now the proper thing:



As you can see, the shadows were not quite right yet. I had a hard time making the shadows work properly. For a while I was unable to make the shadow model fit the right object model. In other words some shadows were following the wrong objects and sometimes weird glitches were appearing randomly. For example there could be only one shadow even if the scene contained multiple objects. Or adding a new object in rotation would cause the shadow of an immobile object to rotate in the same way as the new object. This was due to my misunderstanding of how the OpenGL pipeline behave so it helped me improve my understanding of the pipeline. I found that the problem was that I was using the same matrix for computing all my shadows. Since all my objects have different model, those model must be applied independently to each shadow. I implemented the lighting in a way that it can be toggled back to my other lighting implementation on a keypress and freeze the scene (stop the transformations in progress). This procedure include regenerating my uniform and all my ids to fit the right shader since both lighting uses a different shader.

Since I needed to do my project to be something else than a game I realized I had already built all the resources necessary to built something similar to an atom or a planet system model. I had already created a class called SceneFactory which contains some procedures to create different scenes such as a sample game level or an atom model. I wrote a new scene with a core object and some spheres around it. I also needed to implement some sort of rotational system that would take another object as a parameter. Based on this parameter I had to compute the distance

between the two objects and use it as a radius to calculate a new sphere equation. I also had to compute the initial angle of the object in rotation based on its position from the core object to give phi and theta some initial values. Since my objects are spheres and their texture is uniform I only needed to implement the translation part. I had to dig through my old mathematics courses to implement the rotation around the nucleus. I found it really awesome that I could finally used what I learned in my mathematic courses in real life applications. I also learned that there are many effect in computer graphics that are based on physics equations.



This picture shows multiple spheres rotating around the bigger sphere in the middle. I was off track for a while because I have

not been using spherical coordinates*** for a while and could not figure out why my electrons were following weird trajectories. I did some trials and error to make it work properly in the end because I could get my finger on the exact problem. Before I began implementing this rotational system I tried to use the glm library to reach my ends but for some reasons did not succeed. I tried to use the translation then the rotation to transform my object. For the final touch I added some ambient colors to my objects and also added a random color generator for objects generated at runtime.

In conclusion I learned a lot from this course and from my project. I found out I was really interested in this subject. I was always motivated to work on my project so in the meantime I have applied and been accepted to the Industrial Experience program to find a work term in computer graphics or computer games.

SOURCES:

*http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

**http://www.opengl-tutorial.org/beginners-tutorials/tutorial-8-basic-shading/

***https://en.wikipedia.org/wiki/Spherical_coordinate_system