SPL-1 Project Report, 2019

Hello World Game Engine

SE 305: Software Project Lab 1

Submitted by

Syed Yusuf Ahmed

BSSE Roll No. : 1013

BSSE Session: 2017-18

Supervised by

Md. Saeed Siddik

Designation: Lecturer

Institute of Information Technology

Institute of Information Technology

University of Dhaka

[29-05-2019]

Table of Contents

1. Introduction

It is truly fascinating to think that it takes millions of lines of codes to draw a single pixel on screen. For a game developer, it would be a horrible decision to code each of those millions of lines to each individual pixels that he wants to draw on the screen. To avoid this problem, developers use game engines, which are sophisticated software-development environments designed specifically designed to enhance code reusability and visualization.

Inspired by popular game engines like Unity, my project is a raw coded, 3d game engine. Its purpose is to allow users to create a simple 3d game quickly, without taking the hassle of writing a lot of code. The software features 3d model loading and creating with a drap-and-drop interface, complete with first person camera, lighting, collision detection and response, saving and loading projects and exporting projects as separate jar files.

1.1. Background Study

Necessary background study for this project can be broadly classified into three stages. 3D models, projection and physics.

1.1.1 3d models:

One of the standard and most convenient forms of storing a 3D model is in an “obj” file. Each line in the file contains vertex information of a triangle. Joining may such triangles create a 3D object.

```
# Triangle.obj

v  0.0  0.0  0.0
v  0.0  1.0  0.0
v  1.0  0.0  0.0

f  1  2  3
```

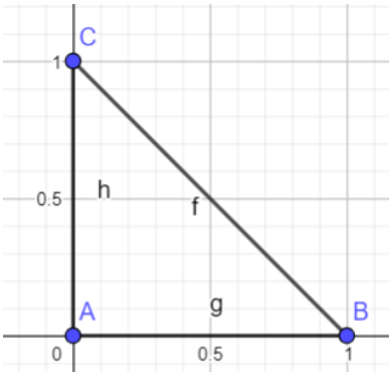


Fig 1: Simplest obj file and its corresponding triangle.

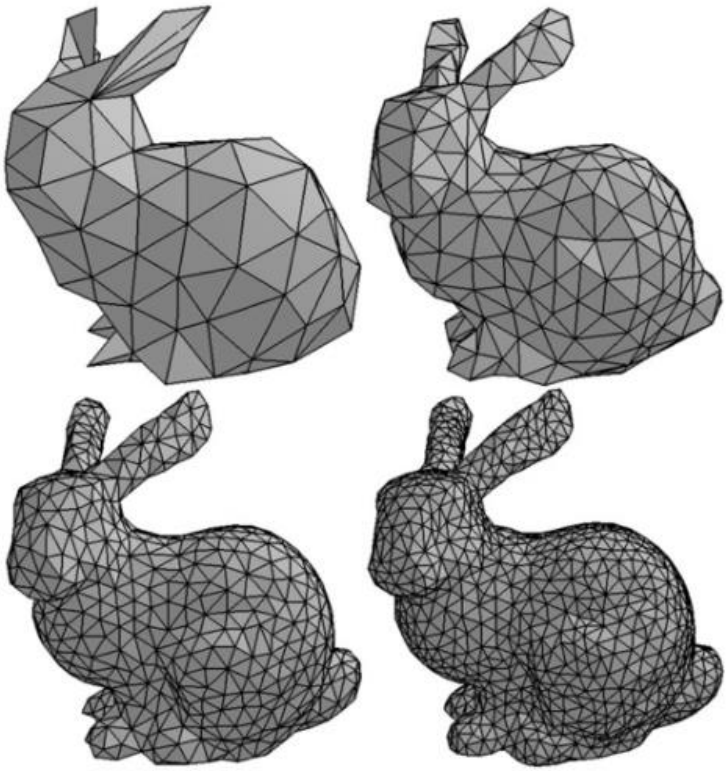


Fig 2: A 3D model of a rabbit created with increasing number of triangles.
http://giovannire.blogspot.com/2014/05/mesh-3d-printer.html

1.1.2. Projection:

These 3D models have to be draw on a 2D screen. But the drawing must be in such a way that it looks like 3D to the viewer. This is illustrated in figure 3, where the further away the object is, the more it converges to one point, as shown by the red lines. This is called perspective projection.

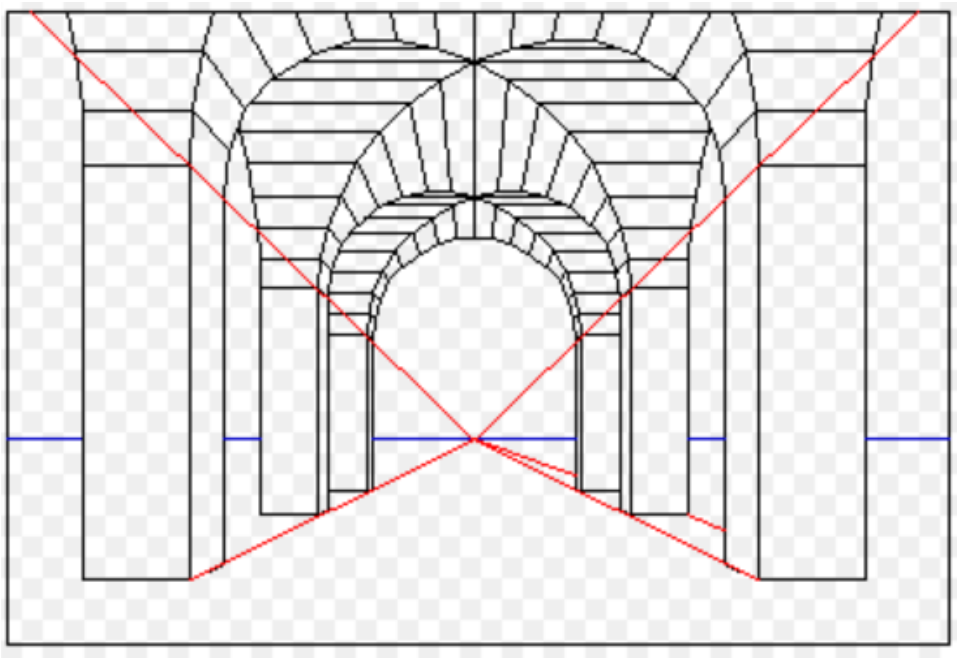


Fig 3: Perspective projection in action.

1.1.3. Physics:

Now that the objects are in world space, it is very common for them to interact with each other physically. Any such collisions must be detected and resolved. If the collision is resolved by simply displacing the objects away, it is called static resolution. On the other hand, dynamic resolution take into account other physics phenomena like momentum, mass, co-efficient of restitution, etc.

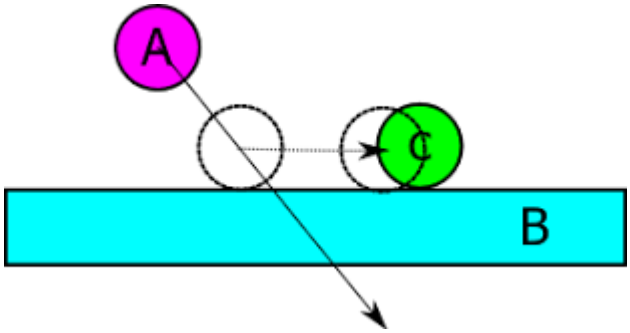


Fig 4: An example of proper collision detection and response.

Firstly, the objects are divided and wrapped into smaller boxes. Then unnecessary divisions are merged using **hierarchical clustering.**

Collision is then detected using the **separating axis theorem(SAT)**. This is where the objects are tested for overlap on each of their edges.

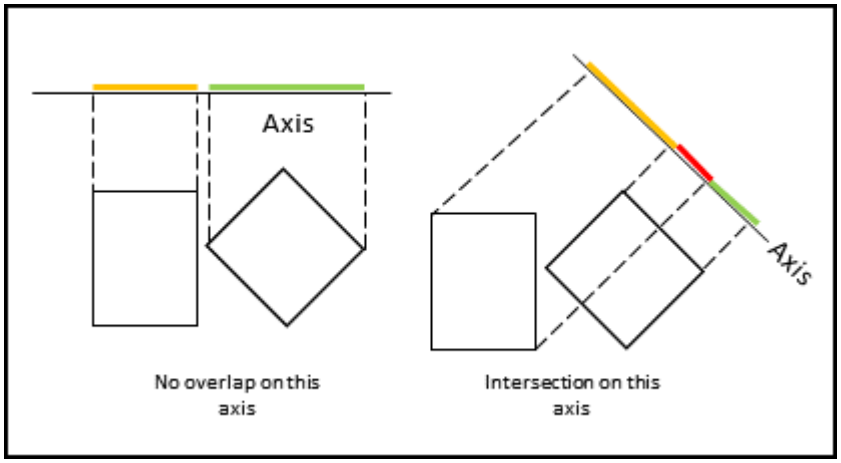


Fig 5: SAT

1.2. Challenges

- Debugging

  This has been one of the biggest challenge for this project. Debugging is always difficult, but what makes this specially challenging for my project is the large number of data set. It has been a common case where a few triangles of a model consisting of say, 10,000 triangles is not showing properly on the screen. In such cases, it is not possible to manually review the states of all those triangles even with a debugger.

  Its solution lies in being able to come up with other models which show the same bug but have a more manageable number of triangles to work with.

- Triangle clipping

  After introducing a moveable camera into the mix, many triangles fell partially outside the field of vision of the camera. Such triangles has to be clipped into parts whereby one or more parts fall into the field of vision and the others don't.

Fig 6: Before and after a triangle is clipped. (The white region shows space in the camera field of view).

- Caching
  Every frame requires a lot of calculations of each triangles and many times, the calculations give same answers as those of previous frames. Such cases are identified using an “if-modified-since” time stamp that is always updated if a setter is called.
- Dynamic class loading (scripting)
  My game engine must allow users to write and test their written code that will be run with the code of the game engine. For this, class are loaded from disk during runtime. But java assumes that the definition of class remains same during its lifetime and will not reload class from disk until the program closes. This was a problem as users may alter and run scripts without having to restart the engine.
  Its resolution includes accessing the command line during runtime to recompile and re-run the user’s created project as a completely different application.
- Accurate collision detection and response
  The naïve form of collision detection between two objects would be to test each triangle of obect1 with every triangle of object2. This needs n*m calculations every frame, making it practically useless. Every other collision detection algorithms were either too elaborate to implement in the time-frame of spl1, or were incomplete and under ongoing research. So I finally came up with my own way of detecting collision, where each object is approximated using a number of boxes without compromising accuracy much. For collision response, I calculated the overlap between the two objects and displaced one by the amount of overlap to resolve collision.

- Manual serialization
  In order to strictly avoid the use of library functions where possible, I manually wrote methods to write and read method attributes from disk. This proved to be difficult for very large classes as any changes to this class would require the read and write functions to change as well.

2. Project Overview

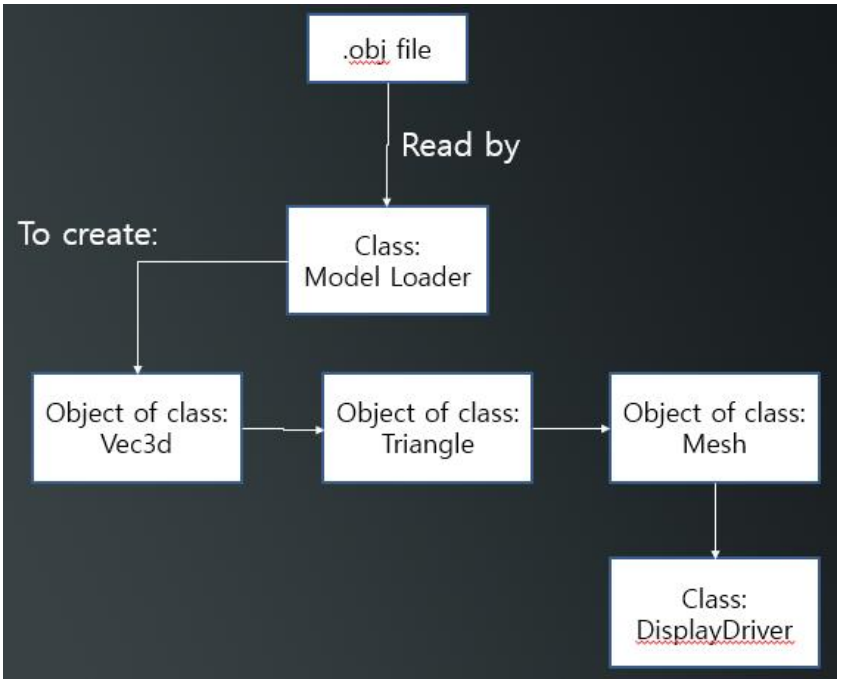


Fig 7: Class diagram (1)

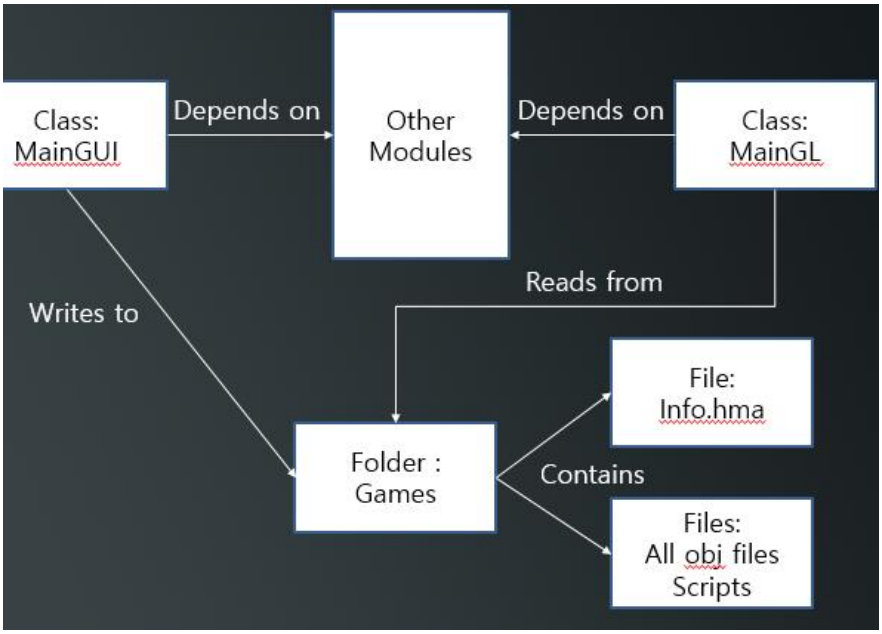


Fig 8: class diagram (2)

Other modules include:

- Input module: To take and process user inputs. User inputs include change in mouse cursor position and/or keyboard presses. This module contains the class KeyboardHandler that processes both of the said inputs for changes like change in camera position and/or camera direction.
- Scripting module: It is impossible to define every possible behavior of objects beforehand. Thus scripting allows users to write their own code that will be run with my own codes. Each script is dynamically loaded from disk during runtime and their methods are run by a master class named MasterScript.
- Physics module: This module handles collision detection and response. It contains classes
  Obb: Object bounding box. This is a box approximation of an object.
  ObjectSliceAndMerge: A class that contains all the utility function of physics engine like methods to divide object in boxes, wrap boxes around objects, etc.
  Collider: this class has the responsibility of checking whether two objects collide. For this, objects were approximated by boxes to reduce the number of required calculations without compromising much accuracy. The results are shown below:

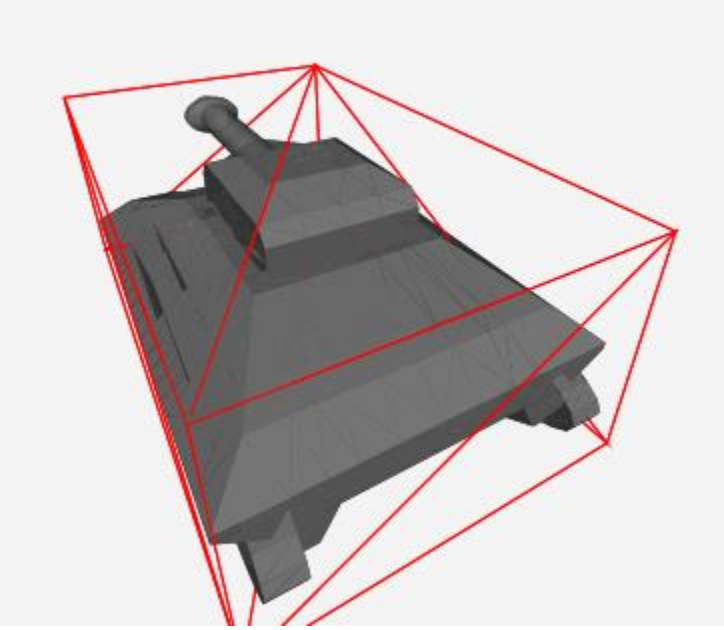


Fig 9: number of boxes used for approximation =1

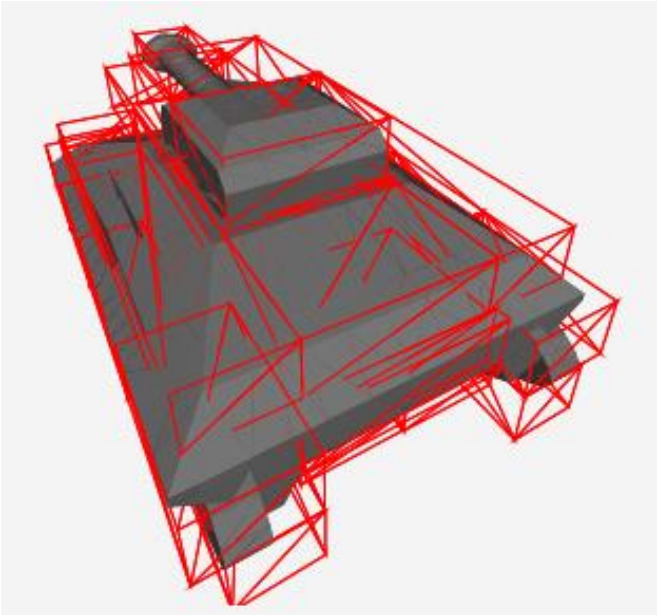


Fig 10: number of boxes used for approximation =35

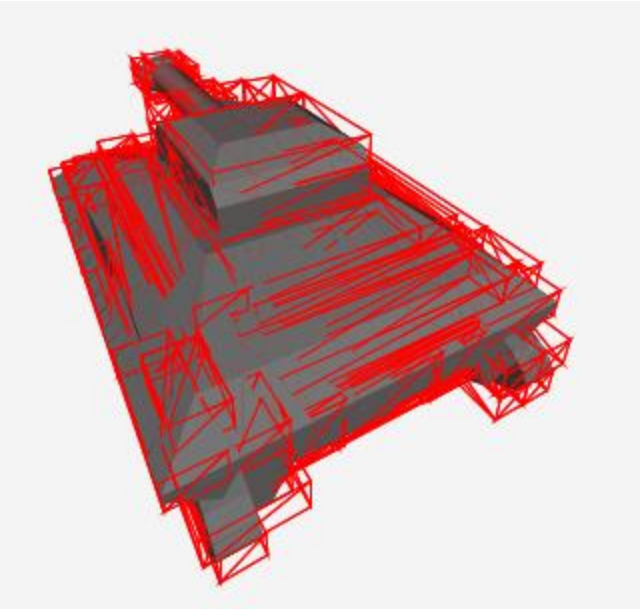


Fig 11: number of boxes used for approximation =208

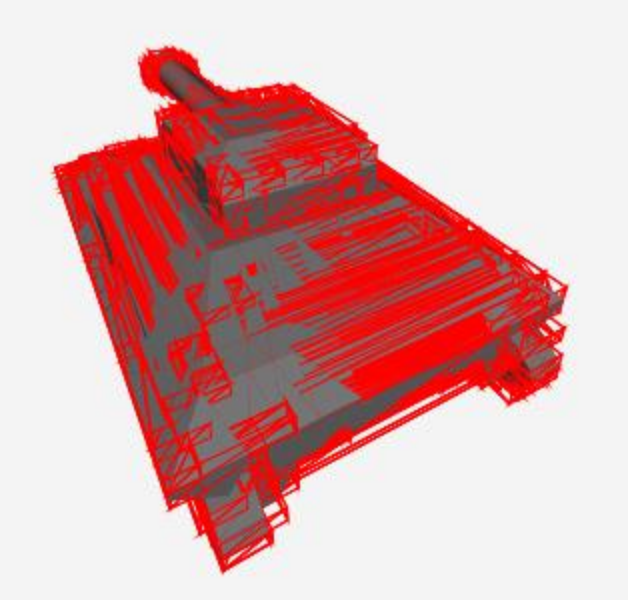


Fig 12: number of boxes used for approximation = 883

3. User Manual

3.1. Requirements:

- Java version: 10

3.2. Usage:

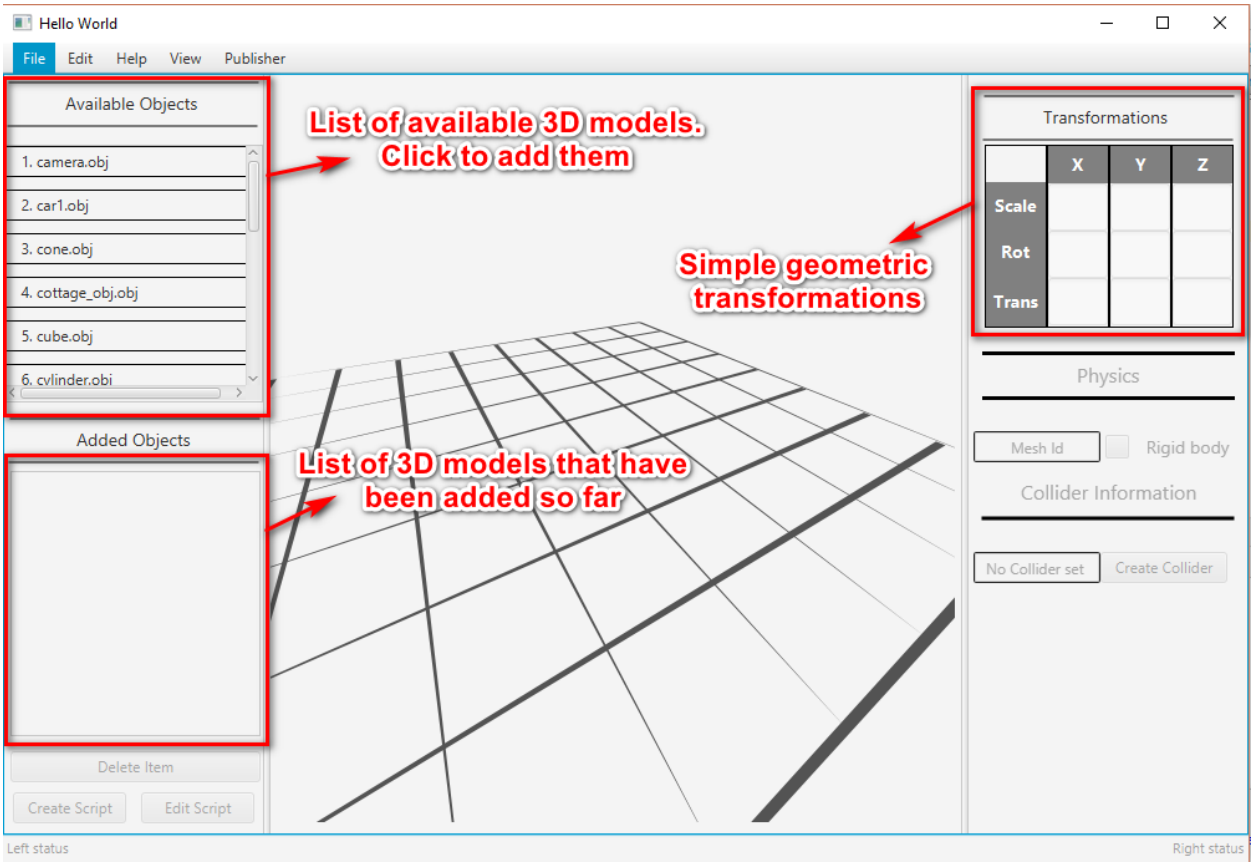
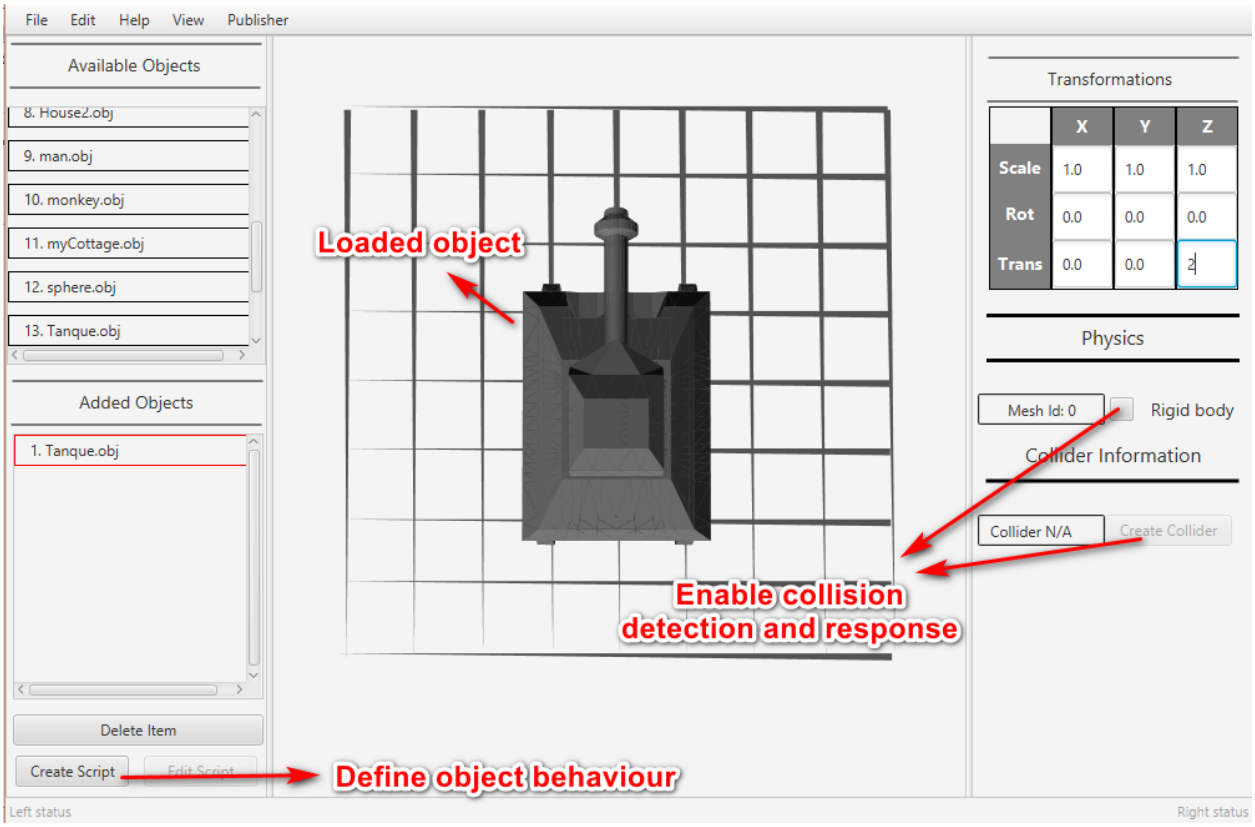
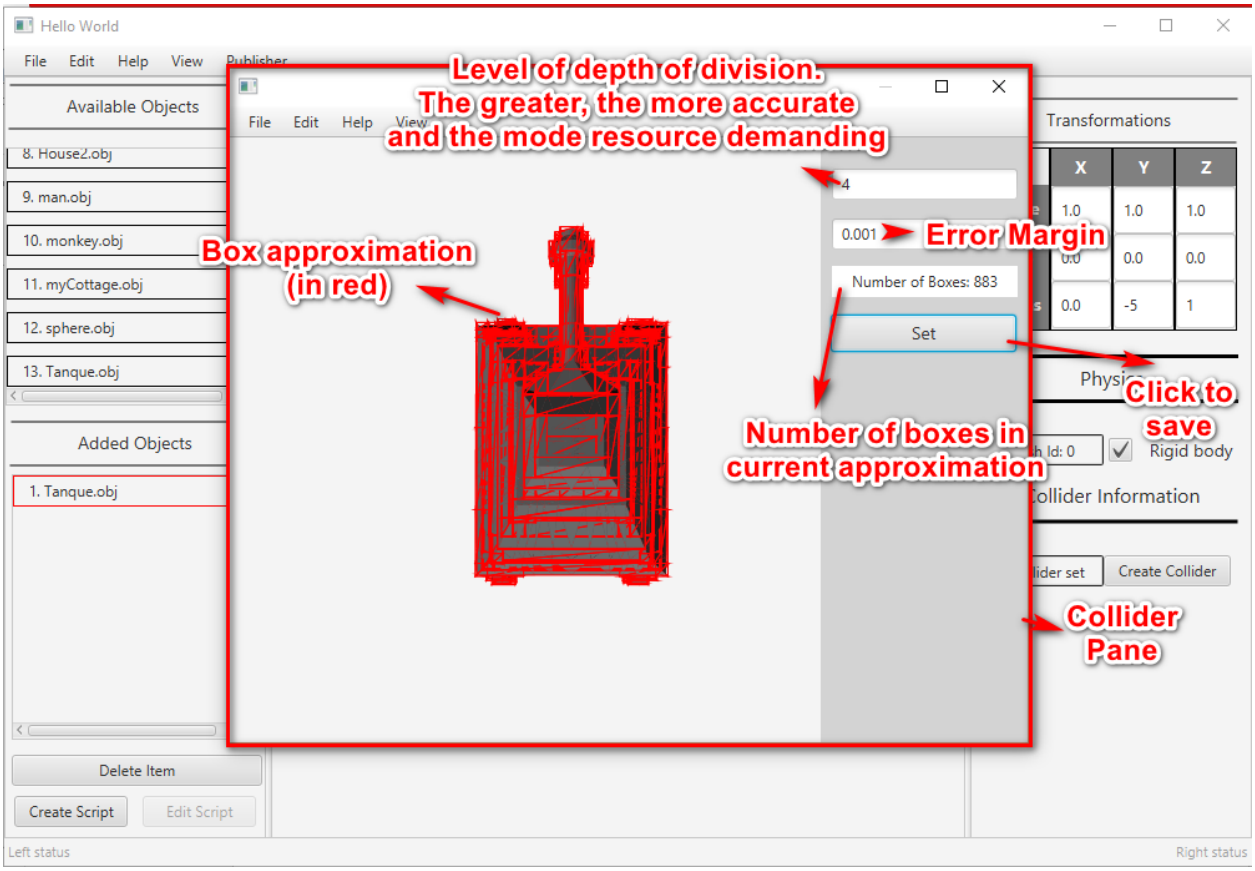


Fig 13: Screenshot


File Edit Help View Publisher
Available Objects
8. House2.obj
9. man.obj
10. monkey.obj
11. myCottage.obj
12. sphere.obj
13. Tanque.obj
Added Objects
1. Tanque.obj
Delete Item
Create Script
Edit Script
Loaded object
Enable collision detection and response
Define object behaviour
Transformations
X Y Z
Scale 1.0 1.0 1.0
Rot 0.0 0.0 0.0
Trans 0.0 0.0 2
Physics
Mesh Id: 0
Rigid body
Collider Information
Collider N/A
Create Collider
Left status
Right status


Hello World
File Edit Help View Publisher
Available Objects
8. House2.obj
9. man.obj
10. monkey.obj
11. myCottage.obj
12. sphere.obj
13. Tanque.obj
Added Objects
1. Tanque.obj
Delete Item
Create Script
Edit Script
Level of depth of division.
The greater, the more accurate
and the mode resource demanding
Box approximation
(in red)
4
0.001
Error Margin
Number of Boxes: 883
Set
Number of boxes in current approximation
Click to save
Collider Pane
Transformations
Rigid body
Create Collider
Left status
Right status

3.3. Menu:

- To save: File -> save or File -> save as
- To load: File -> load
- To edit lighting effects: Edit -> Lighting
- To test project : Publisher -> Run in release mode
- To publish game: Publisher -> Publish game

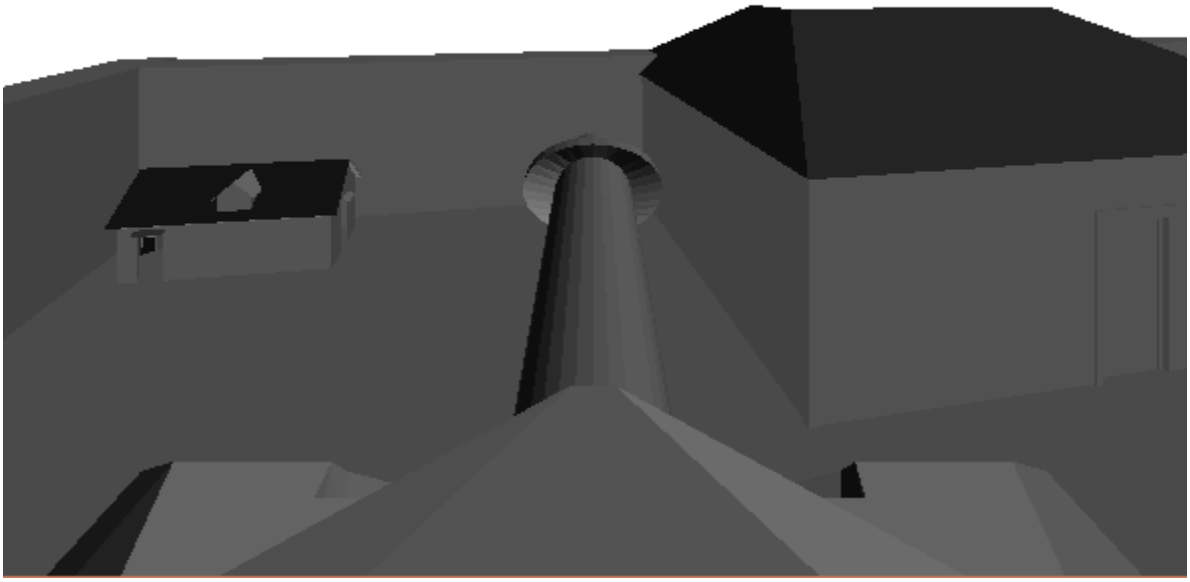


Figure 16: Sample game screenshot

4. Conclusion

Looking back at the beginning of the semester I am confident that I have come a long way toward improvement. I had very little experience in developing and handle large codes. This SPL project has challenged my limits. I am happy that I could complete this challenge successfully and has taken the project to a state of progress that I had hoped at the beginning. I have learnt a lot about the field of 3D graphics, games and the mathematics behind them. I feel that my critical thinking has also improved a lot. Hence I see myself a much better developer/programmer than that I was at the start of this semester.

References

https://en.wikipedia.org/wiki/Game_engine retrieved on 29/5/19

https://en.wikipedia.org/wiki/Hyperplane_separation_theorem retrieved on 29/5/19

https://en.wikipedia.org/wiki/Collision_response retrieved on 29/5/19

https://en.wikipedia.org/wiki/3D_projection retrieved on 29/5/19