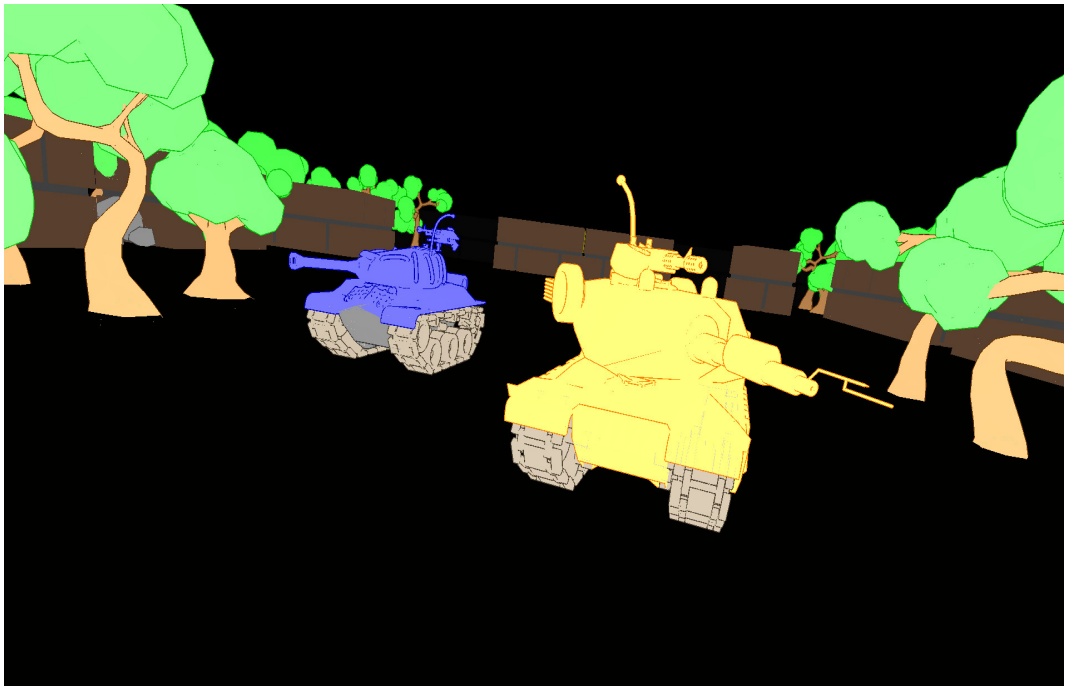
Final Project Report
TANK 2017

Behnam Saeedi
(Saeedib@oregonstate.edu)
CS550: Computer Graphics



Fall 2017

Abstract

This document is a report on the final project of the CS550 class. In this document we talk about proposal, requirements, designs, implementation, learning outcomes and cleaver workarounds that were done in order to finish the project. Furthermore, at the very end, this document provides an installation guide on how to get the game to work.

CONTENTS

1 Proposal 4

2 Process 5

2.1 Camera . . . 5

2.2 Plane . . . 6

2.3 Walls . . . 6

2.4 Importing Models . . . 7

2.5 Test Environment Setup . . . 8

2.6 Game Objects . . . 8

2.6.1 Tanks . . . 9

2.6.2 Trees . . . 9

2.6.3 walls . . . 9

2.6.4 shells . . . 9

2.7 Auto-Load of Map . . . 9

2.8 ASCII-to-Map . . . 10

2.9 Animation . . . 11

2.10 Event Based Game Logic and Physics . . . 11

2.10.1 Keyboard Input . . . 11

2.10.2 Tanks . . . 12

2.10.3 Obstacles and Destruction . . . 12

2.10.4 shells . . . 12

2.11 Particle effects . . . 13

2.12 Lighting . . . 13

2.13 Rendering . . . 13

2.14 Sound . . . 14

3 Comparison to proposal 14

3.1 Models 14
3.2 Texture for brick wall 14
3.3 Main Menu 14

4 Clever Features 15

5 Learning Outcome 15

5.1 What are Shaders 15
5.2 Different available buffers 15
5.3 OpenAL 15

6 Images 16

7 Video 18

8 Installation guide Windows: 19

8.1 Version with menu 19
8.2 Game only 19

9 Installation guide Linux 19

1 PROPOSAL

Tank 1990 was one of my favorite video games on the Nintendo entertainment systems and super Nintendo entertainment systems. One of the best things about this game was the tension and excitement of it as me and my brother used to sit next to each other and play this game against each other.
I have been working on series of 3D models of funny looking Tanks and I thought it would be a fun experience to recreate this old game with new models and in 3 dimensions.

I developed the model for the tanks and the tank shell and I found the brick pattern on Google image which I will texture into a cube and the trees are in OBJ format obtained from turbo squid. Here is a list of things that needs to be done in order to get this project to work:

- Import models from files
- Set up VBOs and push models into GPU memory
- Introduce game objects with distinct X and Y coordinates and angles
- Make objects movable as a function of time, position and angle
- Add map elements
- Create collision model
- Add lighting
- Add Shaders
- Fix Keyboard input (make it continues rather than hits, could be achieved with use of key up instead of keyboard)
- Add Tank shells
- make map elements destructible
- Create damage model
- Randomly generated map
- Costume maps needs to be read from a file
- smoke, dust and particle effects
- Support multiple players locally
- Add sounds

List of models:

1) Tank 1
2) Tank 2
3) Brick walls
4) Trees (several different models)
5) Rocks
6) Tank shell
7) Particles
8) Plane

Game engine will be capable of performing these tasks while maintaining a high FPS. The randomization of certain map elements will introduce some interesting gaming dynamics. Furthermore, creating maps by reading ASCII characters from a text file creates a unique opportunity for generating new maps. Map design could be difficult but by making it generated from file we are eliminating the process of trial and error that comes with positioning game elements in the field. Furthermore, Web interfaces could be created to generate maps. This game could also create a great basis for AI development fr my future personal projects
Each one of these tasks individually is easy to achieve and could be done in less than a day. I believe 14 days is more than enough to have a working prototype.

2 PROCESS

In this section we will cover the steps that were taken in order to create this project and get it to work. This project had several design requirements in order to match the proposal. These requirements were:

- Import models from files
- Set up VBOs and push models into GPU memory
- Introduce game objects with distinct X and Y coordinates and angles
- Make objects movable as a function of time, position and angle
- Add map elements
- Create collision model
- Add lighting
- Add Shaders
- Fix Keyboard input (make it continues rather than hits, could be acheived with use of key up instead of keyboard)
- Add Tank shells
- make map elements destructible
- Create damage model
- Randomly generated map
- Costume maps needs to be read from a file
- smoke, dust and particle effects
- Support multiple players locally
- Add sounds

Lets walk through each one of these features and see how they were implemented.

2.1 Camera

One of the important steps that needed be taken very early on in order to get this game to work was the camera angle and settings. The purpose of this game was to be a 3D version of an SNES game and in order to match

that nostalgia we needed a top-down view. However, this top-down view will limit the 3D effects that we were looking for. As it turned out the best angle was a 45 degree approach. In order to get this effect we placed the camera at up and right of the game plane.

```
#define CAMX -45;
#define CAMY 80;
float eyex = CAMX;
float eyey = CAMY;
float eyez = 0;
float targetx = 0;
float targety = 0;
float targetz = 0;
float upx = 1;
float upy = 0;
float upz = 0;
gluLookAt(eyex, eyey, eyez, targetx, targety, targetz, upx, upy, upz);
```

2.2 Plane

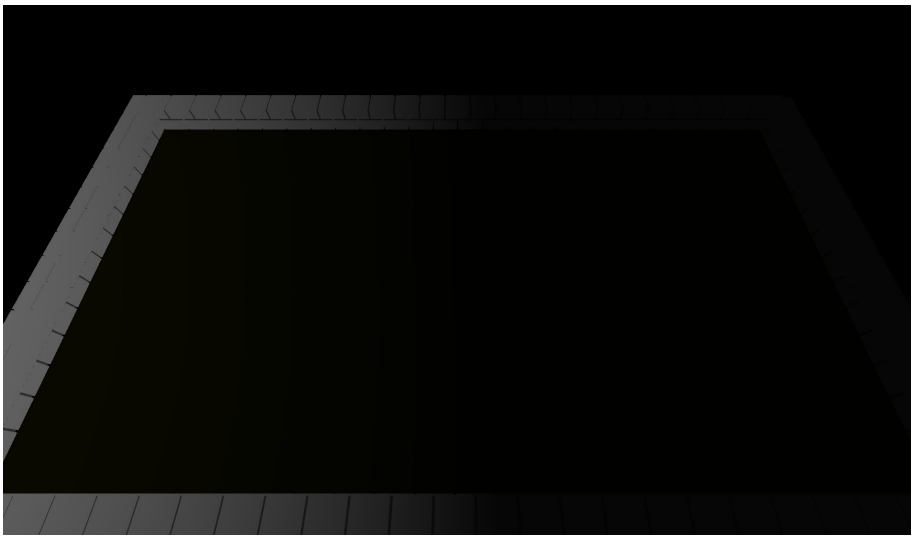
Another model that needed to be present is the plane. Plane is very simple since it is just a rectangle.So we just drew a rectangle where we needed to have game elements appear.

```
#define MAPEDGEX 40
#define MAPEDGEY 70
#define CUBESIZE 6.0
glBegin(GL_QUADS);
        glPushMatrix();
        glColor3f(0.05, 0.05, 0.0);
        glVertex3f(MAPEDGEX + CUBESIZE,0, MAPEDGEY + CUBESIZE);
        glVertex3f(MAPEDGEX + CUBESIZE, 0, -MAPEDGEY - CUBESIZE);
        glVertex3f(-MAPEDGEX - CUBESIZE, 0, -MAPEDGEY - CUBESIZE);
        glVertex3f(-MAPEDGEX - CUBESIZE, 0, MAPEDGEY + CUBESIZE);
        glPopMatrix();
        glEnd();
```

2.3 Walls

next step for this project was to specify the map perimeter. We already know the map dimensions from last section and now all we have to do is to generate the walls. Originally I wanted to use textures in order to do this part.

The texture did not look as good as I was hoping for, so instead I made my own wall. The advantage of this is we can have walls in any color we want now, which allows us to have 1 model for distributable, base and non distributable walls:



another advantage to this approach is the fact that we can now procedurally generate and place walls.

2.4 Importing Models

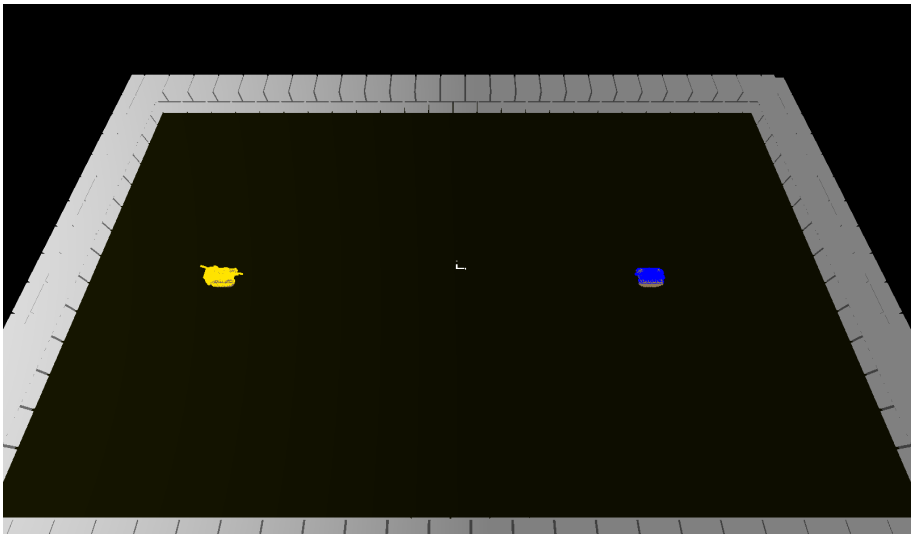
This probably was one of the most important and time consuming parts of the project. The game has series of models that needs to be converted from Autodesk 3D Max format to OBJ in order to be imported to OpenGL. This took a long time and lots of trial and error to find a compatible model with the parser that I was using. Here are a list of models that needed to be imported:

1) Tank 1
2) Tank 2
3) Brick walls
4) Trees (several different models)
5) Rocks
6) Tank shell
7) Particles
8) Plane

And here are pictures of some of this in 3D max:

2.5 Test Environment Setup

We needed a test environment setup in order to be able to test the models and continue with the project.



2.6 Game Objects

This section covers the code behind the game objects that were imported.

2.6.1 Tanks

Each tank is an individual game object with its own properties such as ammo count, smoke count, Health points and more. Each one of those needed to be declared and accessible to functions that would modify them later in the game.

2.6.2 Trees

There are 6 different trees and one rock. This creates an interesting game dynamic where there is a random element in the game. The trees will be selected out of this 7 random options. The rock will not allow movement making each round of same map's game play a bit different. Position and orientation of each one of these models needed to be set and carefully analyzed.

2.6.3 walls

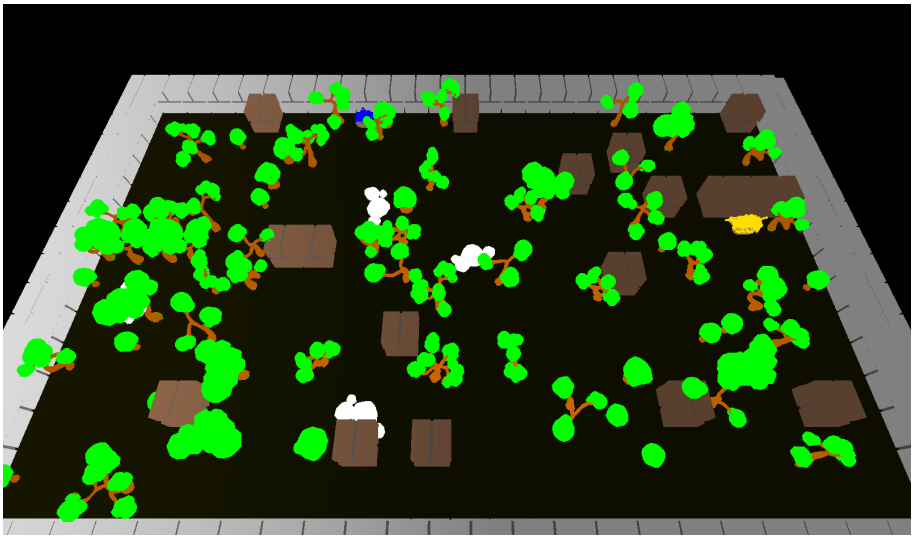
Walls have an interesting property. They are destructible but while standing they do not allow movement. Other objects can collide with walls.

2.6.4 shells

Shell has several features. Firstly, it is a completely independent object than the tanks and actually follows realistic tank battle physics. It has a range, accuracy and penetration. The penetration factor creates unique situations where the shell bounces and hits other objects. Also, the shell can destroy brick walls.

2.7 Auto-Load of Map

The next step for this project is to have a grid where we can assign what item shows up in each cell of the grid. The was the fastest way of populating the map for the game engine. In order to achieve this a game object was created and using a function we can grammatically populate the entire map. Also, map objects hold data for an element called Map Collision Model or MCM.



```
struct Map {
        float coord[24][14][3];
        float color[24][14][3];
        float angle[24][14];
        float HP[24][14];
        bool isSolid[24][14];
        bool MCM[24][14];
};
```

2.8 ASCII-to-Map

After the Auto-load functionality for the map, a problem appeared that needed immediate solution. It was very difficult to populate the maps and randomly generating them did not create any interesting patterns. In order to address this I created a parser to generate the maps.

```
$$$$$$$$$$__$$$$$$$$$$
$$_###$####__##+##$##_$$
$__##_____________##__$
___#_____$$___________
_______##$$$$##_______
##_____#_____#_____##
B#_____##_____#B
B#__A_____##_____T__#B
##_____#_____#_____##
_______##$$$$##_______
___________$$_____#___
$__##_____________##__$
$$_##$####___+###$###_$$
$$$$$$$$$___$$$$$$$$$$
```

- '_': Empty space
- **'#':** Brick walls
- **'$':** Trees
- **'B':** Base walls
- **'A':** Player 1
- **'T':** Player 2

This solution was extremely effective making it very easy to generate different maps with different features. This could create a lot of potential for some interesting symmetric and asymmetric scenarios.

2.9 Animation

The animation heavily relies on time and a multiplier. The time is used for effects that are independent of player such as the particle effects and the shells that are flying around. Those elements are time Dependant. Arrest of the elements are dependent on the speed which the player can adjust to set how fast they want their game play to be. These animations are generated through a function of either time or speed. In our engine the animations could be triggered based on 2 elements:

1) **User input:** These events are triggered when player enters a key to move the tank, turret, or fire the main gun or deploy smoke.
2) **Time:** These events are triggered when a unique time is generated or expired.

2.10 Event Based Game Logic and Physics

Next step in implementation was creation of events. Each event has a specific task and is triggered under unique set of conditions. These events dictate what happens in the next frame. They control everything from the position of elements in the screen to physical properties of tank shell ricocheting.

2.10.1 Keyboard Input

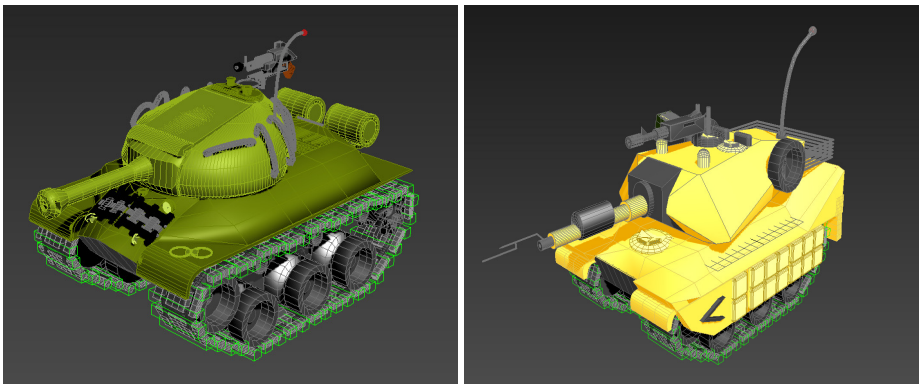
Keyboard plays a major factor in triggering of different events. Bellow are the key bounding and the events:

- **W:** Yellow tank move forward.
- **S:** Yellow tank move backwards.
- **A:** Yellow tank move left.
- **D:** Yellow tank move right.
- **Q:** Yellow tank's turret move left.
- **E:** Yellow tank's turret move right.
- **F/Space:** Yellow tank Fire.
- **C:** Yellow tank deploy smoke.
- **I/8:** Blue tank move forward.
- **K/5:** Blue tank move backwards.
- **J/4:** Blue tank move left.
- **L/6:** Blue tank move right.
- **U/7:** Blue tank's turret move left.
- **O/9:** Blue tank's turret move right.
- **H/Enter:** Blue Yellow tank Fire.
- **N/.:** Blue Yellow tank deploy smoke.
- **G:** Game restart
- **Escape:** Game exit
- **+:** Speed up game
- **-:** Speed down game

This keyboard binding posed a problem. The keyboard function was not capable of handling multiple keys at the same time. In order to solve this issue, the solution was to set up a boolean that contains all of the key triggers. This way we are not blocking the process while handling the triggers.

2.10.2 Tanks

There are several rules that follow the tank object in the game. These rules cover, tank's interaction with other tank, its position, ammunition, smoke and health points. Furthermore, tanks trigger the shell event. Both tanks have a specific speed for movement and rotation of hull and turret. These models had to be carefully analyzed and added to the game engine.

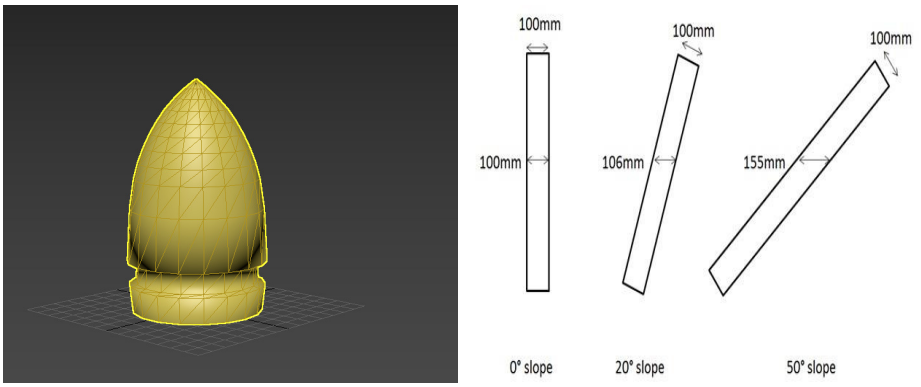


2.10.3 Obstacles and Destruction

Obstacles are generated on map load and all of their physical properties are set on load. an event for this feature is triggered if it is a wall and it gets shot. This would also trigger particle effects and the wall falls apart.

2.10.4 shells

The shell object has its own unique X and Y and Z coordinates and after the trigger it changes its behavior based on where it hits. A math model needed to be developed in order to calculate its trajectory. This will be calculated based on where it is, what angle it is moving at with respect to surface and how much damage it is going to cause based on that angle. There is also a small chance of ricocheting if the angle is too sharp. This is realistic to normal tank to tank combat in physics. If the angle of incident is too sharp, the round has to travel through thickness of the armor at an angle increasing its effective armor. This technique is actually used by tank commanders to protect their tank. This technique is known as "side-scraping".



2.11 Particle effects

Next portion to be developed was the particle effects. The particle effects include smoke, explosion and dust. They are based on time and they could be created under one of the following conditions:

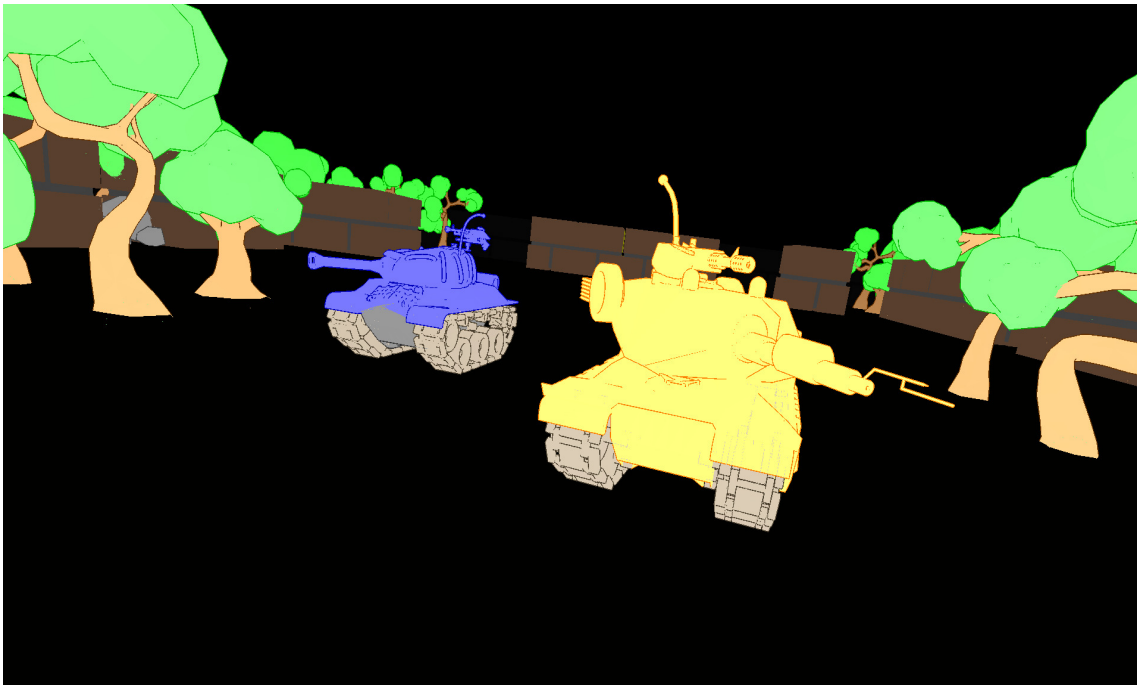
1) Destruction
2) Movement
3) Firing of a round
4) Deployment of tactical smoke grenades

2.12 Lighting

The next step was to create lighting. This was easy and is done in 2 steps. First step is using normal OpenGL lighting for dual rendering of the object (we will talk about this more in rendering). Second, is the shader lighting.

2.13 Rendering

Elements in the screen based on their type get rendered twice. This is because of the effect that I was trying to achieve.



As you can see in the picture above, the elements in the scene have a darker outline almost as if they were painted with pencil. This improves the look of the game by making small details in the model more visible. This effect is achieved by rendering the scene twice. First process of rendering uses ambient light only, giving everything a uniform color. This uniform color is in the back and seen as the outline. On top of that we are rendering with normal lighting to cover the polygonal lines of the model that was created with ambient lighting and that gives our model the uniform polished look.

2.14 Sound

Next step and last step in the development was the addition of sound. This was done using a library known as OpenAL. Unfortunately working with OpenAl was very difficult since most of the documentation referred to in tutorials were deprecated. Nonetheless, the sound is in place and the game has several different audio tracks that get triggered at different events:

1) **Main Music:** Triggered at the beginning of the game
2) **Shell firing sound:** Triggered every time a shell is fired
3) **Bounce sound:** Triggered when shell ricochets
4) **Explosion sound:** Triggered when one or both of the tanks explode.

3 COMPARISON TO PROPOSAL

3.1 Models

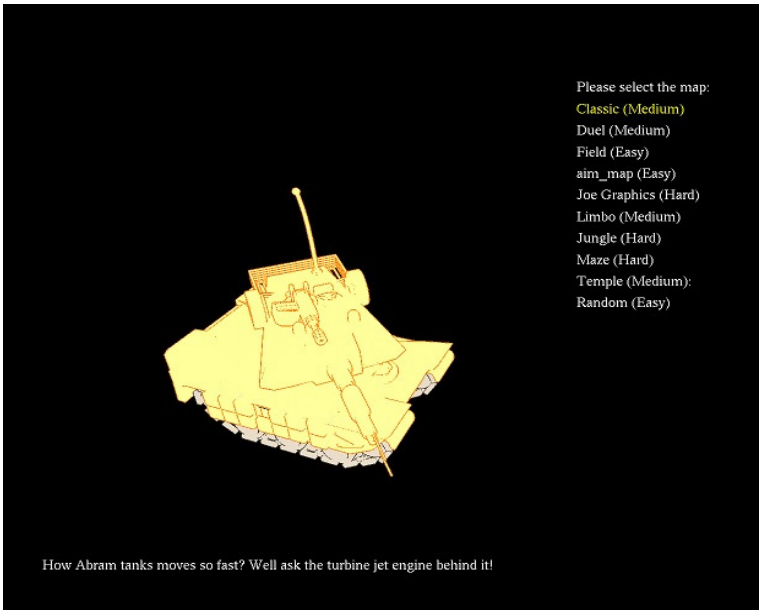
Only 2 tank models were used in the game instead of 3 since there was not a possible way to put 1 more set of controls for the player 3.

3.2 Texture for brick wall

Instead of using the texture, I create a brick wall model which significantly improved how the game looks by adding more detail to the model.

3.3 Main Menu

In addition to the game a main menu was needed in order to help user navigate different levels and start the game.



4 CLEVER FEATURES

There were several cleaver work around in this project. However, following work around are by far my favorites.

- The pencil outline effect for models
- Shell ricochet
- Improving performance with VBOs
- Map loading feature

5 LEARNING OUTCOME

This project truly put my knowledge of OpenGL to the test and forced me to learn about new features such as stencil buffer, depth buffer, order which objects are drawn at, and many more. Following subsections are some of the new things I learned through this project.

5.1 What are Shaders

A major breakthrough happened in my project once I realized that shaders are literally programs that get executed on GPU instead of CPU. This gave me the ability to take this into my advantage and significantly improve both looks and performance of my game.

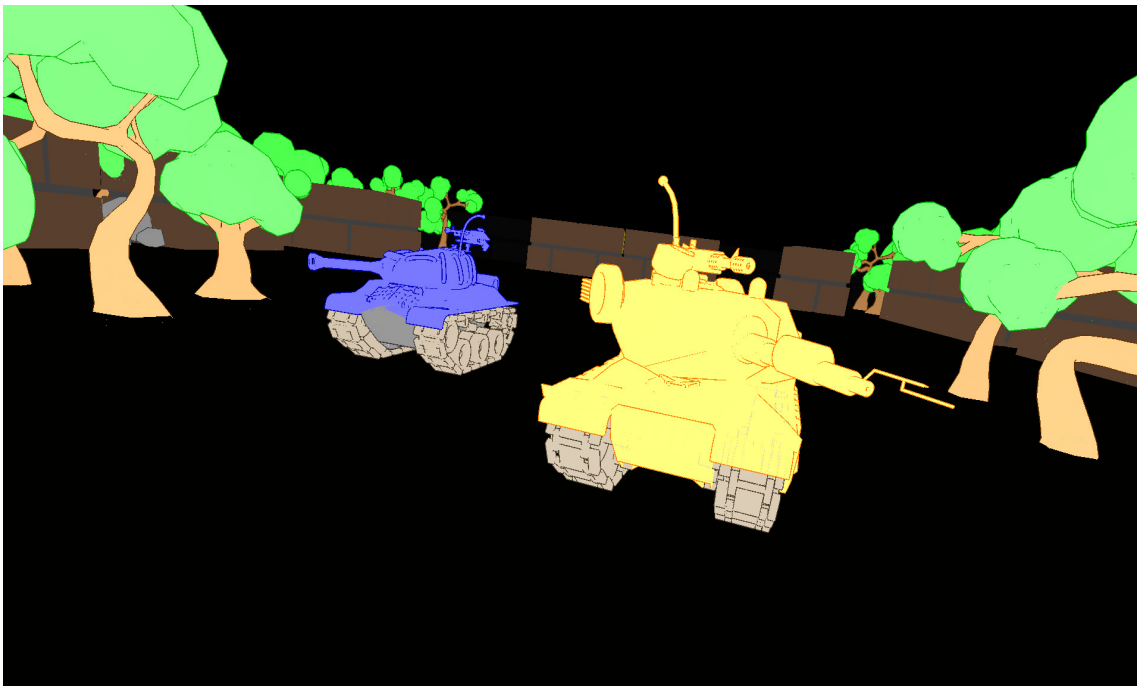
5.2 Different available buffers

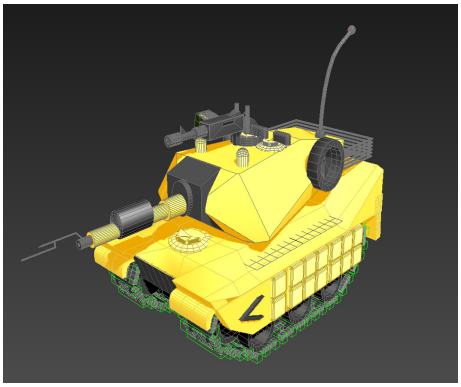
The OpenGL has several buffers that could be manipulated at developer's disposal. These buffers are the depth, color, stencil and many more. In this project we used these buffers to create some interesting visual effects.

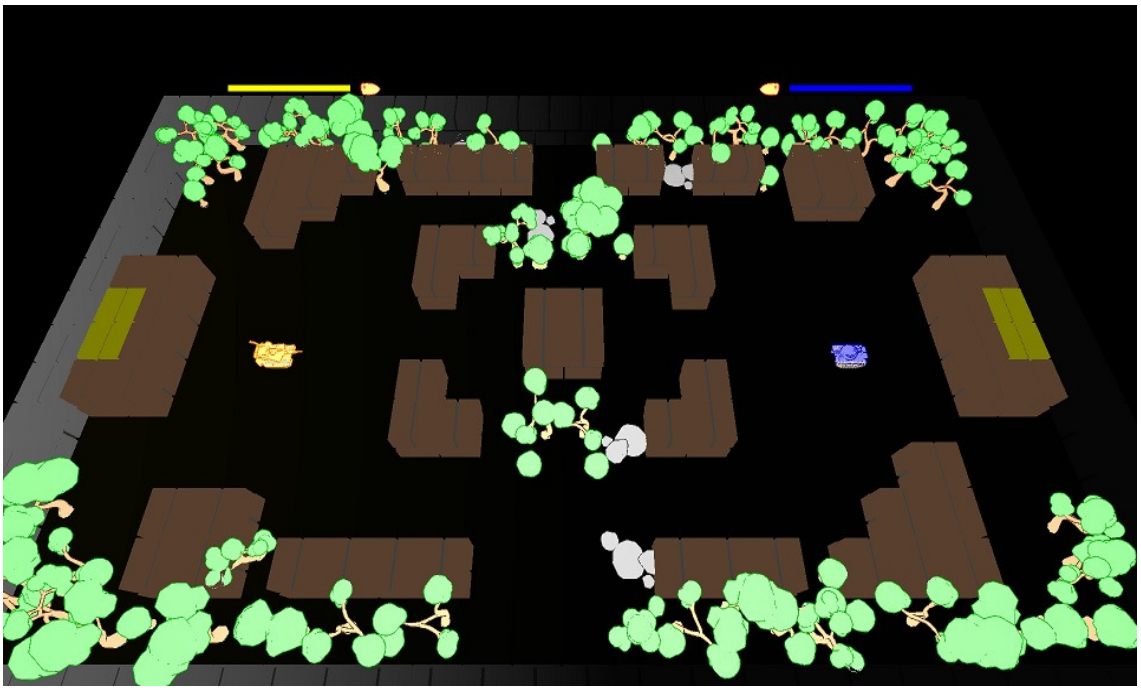
5.3 OpenAL

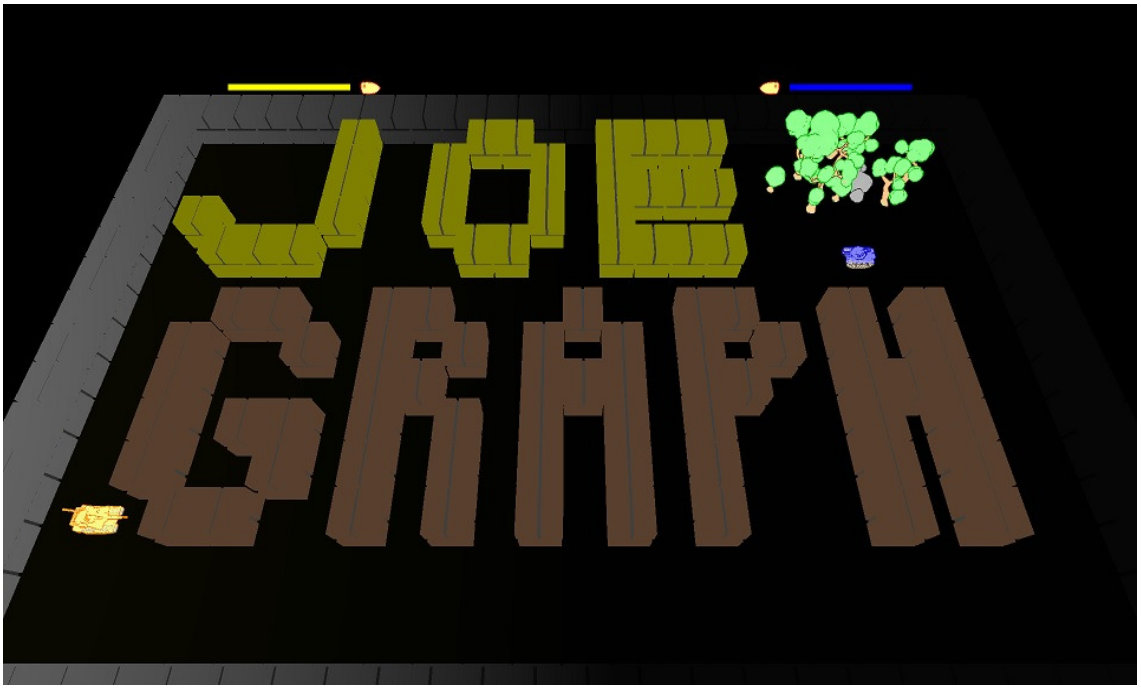
One of the tools that I learned about by using was OpenAl library which provides an extensive audio feature to C++ programs. It was very difficult to get this to work and it had a surprisingly poor documentation.

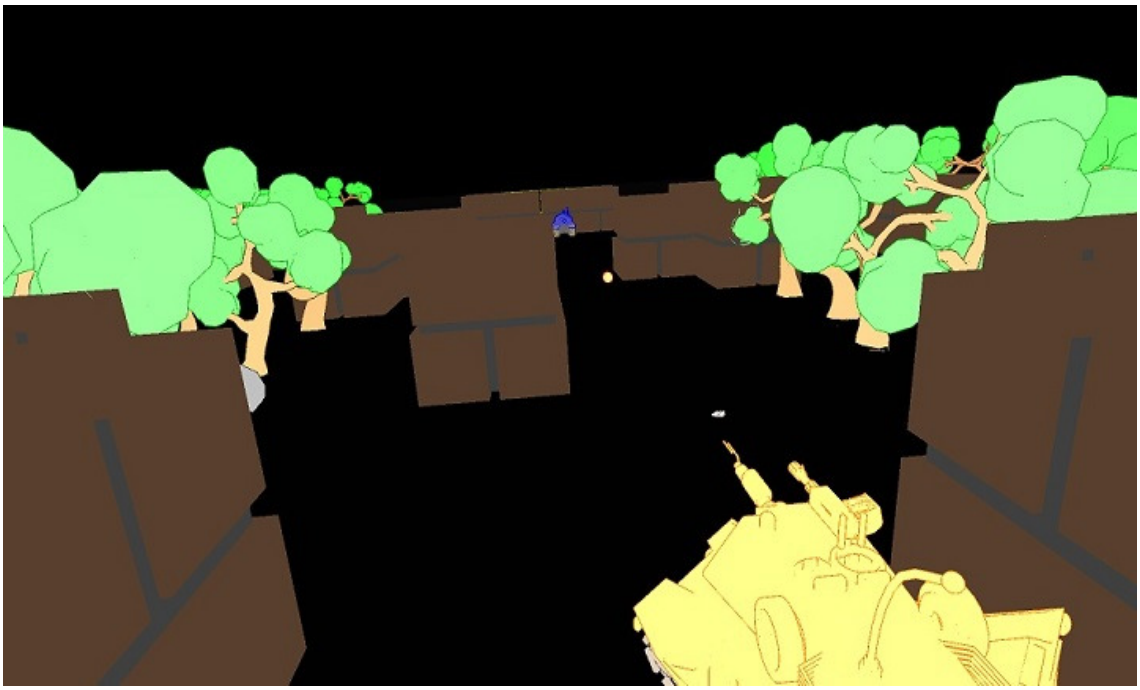
6 IMAGES

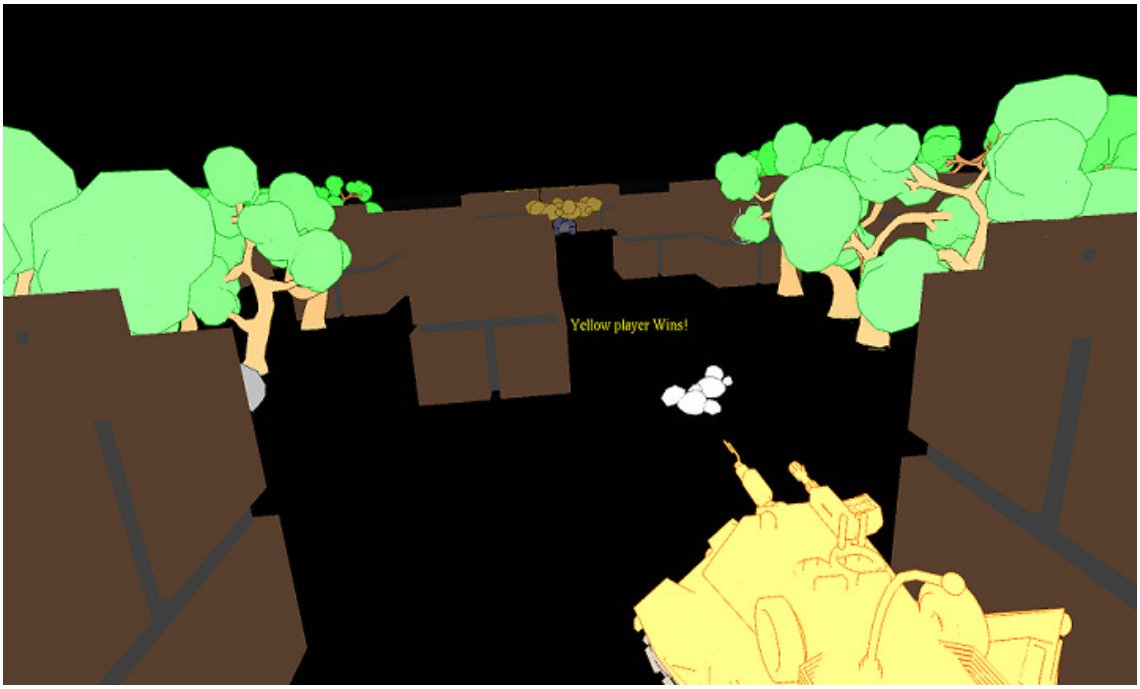







JOE
GRAPH





7 VIDEO

Link to video

8 INSTALLATION GUIDE WINDOWS:

8.1 Version with menu

1) Copy the zip file to desired destination
2) Unzip it into a folder
3) Run ”oalinst.exe” This will install necessary files for OpenAL
4) Run ”Tanks-2017”
5) The main menu will load up.
6) Use the arrow keys to navigate through the main menu (if the arrow keys did not work it means the game window is out of focus, just click anywhere on the screen)
7) After selecting the desired map, hit enter and wait for the game to load.
8) if the game buttons did not work, it means the window is out of focus, click anywhere on the screen.

8.2 Game only

1) Copy the zip file to desired destination
2) Unzip it into a folder
3) Run ”oalinst.exe” This will install necessary files for OpenAL
4) Run ”Tanks-2017”
5) Two screens will get prompted, one is a console and the other is the window
6) Click on the console and enter the number of your desired map, 1 to 5 is default.
7) The game should load in the models and start the game
8) Click on the graphics window and enjoy the game.

9 INSTALLATION GUIDE LINUX

1) Extract the folder
2) Type ”make”