EECE 478

Game Project

Grigoris Eglesos (37465051)

Karl Burkat

Ting Yun (86768017)

York Pan (84258045)

April 10, 2008

Abstract

In order to demonstrate the theories and skills learned in our ECE 478 class, we were split into teams and told to design a game from "scratch" using OpenGL, glut or SDL. This game must demonstrate basic understanding of the theories learned in class, and also attempt to include some of the more advanced techniques that were discussed.

This report will detail the design process from initial ideas to implementation and testing. It does not cover background material, which is simply the course material. It is recommended that the reader have some knowledge of OpenGL and computer graphics theory before reading this report.

Table of Contents

List of Abbreviations

OpenGL	Open Graphics Library
GLUT	OpenGL Utility Toolkit
SDL	Simple Direct Media Layer
TD	Tower Defense
RTS	Real Time Strategy
FPS	First Person Shooter

1.0 Introduction

Our team is comprised of the remnants of the class after all the teams were formed, so the first challenge we encountered was trying to find a genre of gaming that everybody was familiar with. Familiarity brings the advantage of a unified design vision which we would need if we were to create an original game both visually on par with the larger groups, as well as being simple and fun to play.

In this report, the major components of the game will be separated and the progress made on them will be described independently. The design phases of conception, implementation, and testing and modifications will be covered logically for each of the components. In these sections, this report will discuss success requirements for each component, the steps taken to implement the solutions, document the results of testing and solutions to problems discovered, and will analyze the results. For brevity sake, terms explained in the course material will be assumed to be understood by the reader.

As of the writing of this report, the game has not been completed, and progress is ongoing. For the demo, we will have a working game with a completed feature set, and when it is complete, there will be an addendum to this report detailing the aspects that were completed in the intervening time.

2.0 Game Mechanic

2.1 Inspirations

Initial ideas included some tower defense games[1], a Scorched Earth / Gunbound / Worms style game[2], and some simple RTS game ideas. Keeping in mind that the game must meet the requirements of integrating the third dimension into gameplay aspects somehow, we tried to modify the ideas to incorporate the third dimension either strategically or as a mechanic.

In the case of the TD game, players would need to build according to hills and valleys to try and herd the "monsters" around the map to maximize the amount of time the player's towers have to shoot at them.

The shooting game idea incorporated the 3rd dimension rather easily as there would simply be a second angle for the players to align to determine direction as well. To keep that idea from being trivial, we would set a random wind vector to throw off the player's aim. At this time, the game we had envisioned was still a turn based game, in the spirit of the games which inspired us.

We were unable to come up with a method of effectively integrating the 3rd dimension in an RTS game, so we discarded that idea.

2.2 Modification

Our idea of a 3d Scorched Earth was the most appealing to the team members so we went with that as an initial design target. For this type of game, we would need a character model of some kind, a world, some kind of collision detection, and a projectile mechanism. We spoke to Dr. Iverson and got his input on the idea and also to get a sort of green light on the project before we devoted too much time into the idea which may or may not be badly received at demo time.

With Dr. Iverson's input, we decided to eliminate the turn based aspect and go with a real time system.

2.3 Considerations

It was around this time that there was some debate as how the game would present itself, as in whether there would be an AI controlled opponent, or whether it would be online with another player. With a little research, we found some prebuilt libraries that would handle most of the net code, so we decided to rule out an AI controlled opponent in favor of player vs player interaction.

Later on in development, we were focused so heavily on other aspects of the game; we realized we had neglected the networking portion for too long. After bringing this up at a weekly meeting, we pushed network play off the schedule until we could finish building the world and have an interacting character in it. In the worst case scenario, where we did not have any time to implement networking at all, we would simply set two viewports side by side as in multiplayer console games and set the controls on the keyboard for two players.

3.0 Character Model

3.1 Concept

Originally the game we envisioned was heavily focused on the gameplay and not the characters that the players are controlling. The characters could be tanks, or soldiers, or even a kid throwing a snowball. While the actual model was not clear, the process which we would build the model was mostly certain though. We would need to create a skeleton on which we could lay a skin of our choosing.

3.2 Implementation

For the skeleton we used OpenGL and created series of lines drawn straight down, at the local origin and then rotated and translated each of them to the proper position to form a stick figure. What this provided us was both a center point for collision detection calculations, and basis on which we could overlay the skins. The separately drawn stick figure also gave us the benefit of being able to animate the limbs independently and therefore also the skin, while remaining conceptually simple.

For the skin, we had originally tried to build a model using OpenGL and GLUT primitives; however, this proved to be very time consuming and made the math fairly complex. We spoke with other groups to see where others were standing with the same issue, and we came to the conclusion that we could use an external program to create the skins we needed. We started trying to build models using 3dStudio Max and importing them into the game world. The models looked infinitely better than the ones we tried to draw using primitives and it was much faster as well. With the inclusion of some extra code, we were able to import the models into our game, but not without having to debug a nightmare of linker errors. After solving the linker error problem, the skins bound to the skeleton with no problem. The entire character moves with glTranslate and glRotate functions.

4.0 Interaction

4.1 Perspective

We did not know what kind of view we wanted at first, whether it be the classic FPS type of perspective or a third person perspective[3]. So to begin with, we created a first person camera system in GLUT, as it seemed the easiest at the time.

4.2 Movement

The initial first person[3] system used polar global coordinates with keyboard callbacks to turn and move the camera naturally. This worked quite well, given the inherent keyboard callback limitations of GLUT.

The next task was trying to get link the movement of the camera and the model. The first problem that we encountered with this task is that gluLookat moves the camera differently than glRotate does. Trying to brute force coordinate the movement of the camera and the objects did not go very well, the mathematics were complicated and hard to keep track of.

Around this time, the team decided to switch the camera perspective from first person to third person, to better show off the models we were developing in 3dStudio Max. Trying to convert the camera system to 3rd person view in polar global coordinates while trying to synchronize the movement of the model was becoming far too time consuming, so we decided to redo the entire camera system to match the skeleton drawing system. That is, to use a simplified local coordinate system instead to take care of translations and rotations. While conceptually harder to grasp at first, it pays off when trying to coordinate more complex movements at once, as the mathematics are just much simpler than compared to a polar coordinate system.

4.3 Projectiles

The first projectile system we tried to implement involved a simulated parabolic arc with user inputted variables such as lateral and vertical angle and launching force. Of course these would be worked into the controls of the game, and not as simple numbers to be inputted. The implementation got as far as the theoretical formula for velocity as a function of (arbitrarily created, for this purpose only) time and initial velocity.

Once the force vector is broken down into component vectors with the angle information, the calculations for positioning of the projectile at a given time works out like this:

$$Zposition = (Zspeed*time + (g*time^2)/2);$$

$$Xposition = Xspeed*time;$$

Where the Z position represents altitude and X position represents distance traveled from the starting position, aka the player. The vertical component of the projectile vector is under constant acceleration downwards while the horizontal component is a constant velocity. Time is simulated for the duration of this calculation. The calculation continues until the vertical position reaches the height of the plane, or if a collision is detected with an object.

It was not brought to full implementation before the time of the writing of this report.

5.0 Landscape

5.1 Intention

During preliminary planning phases, we had envisioned a procedurally generated landscape that would be pseudo random generated for every new instance of the game. The area which the player is allowed to move around and interact in would be a hilly open area dotted with trees and rocks.

5.2 Realization

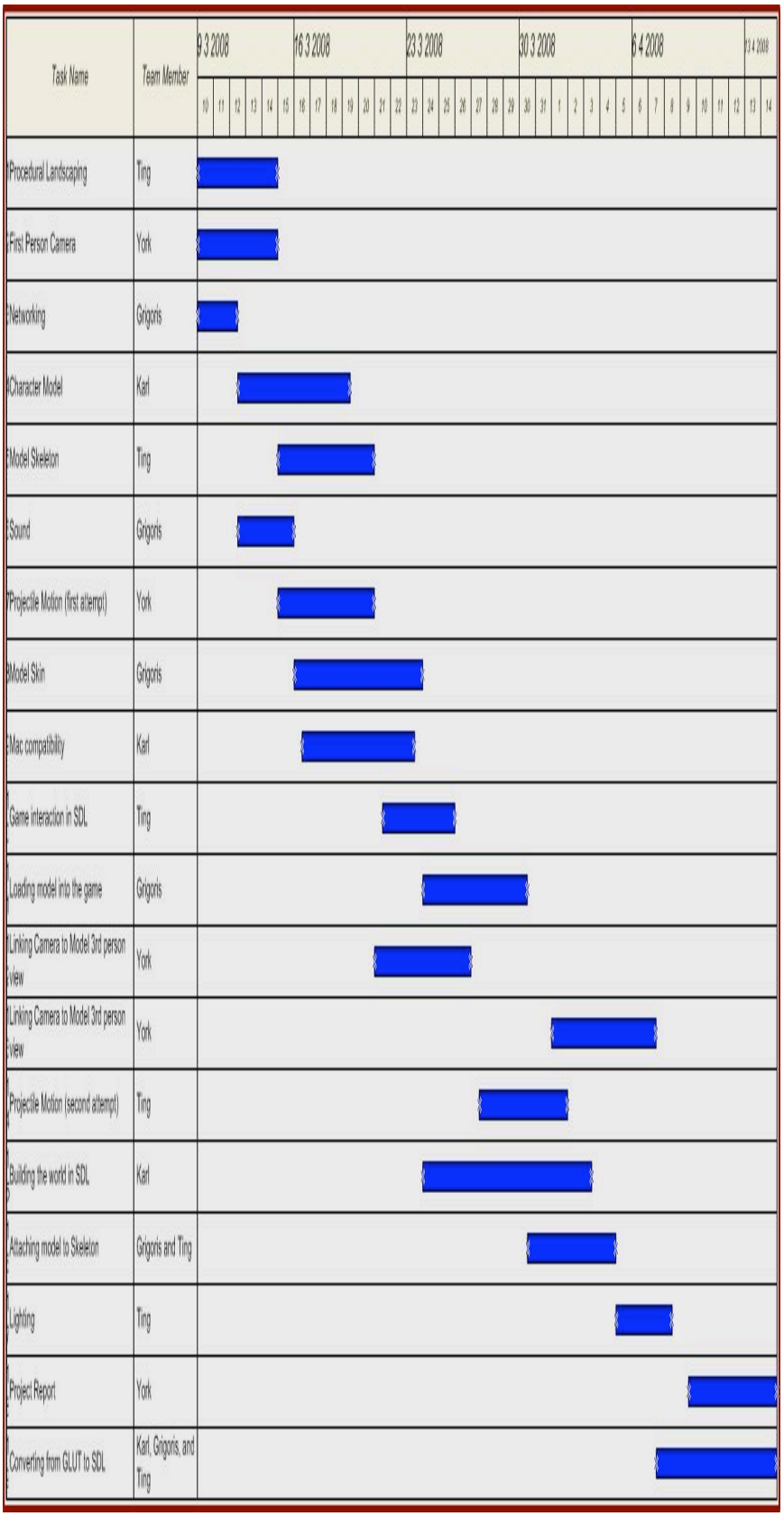
Getting the terrain generator to work properly proved to be more complicated than first anticipated. The theory works by first generating a 2 dimensional array of floating point integers, then using glTexCoord to bind the floor to those points. After having spent too much time already on a non vital feature, we decided to shelve this particular feature for now, and go with a flat terrain.

5.3 Skybox

One simple addition that makes a world of difference was the addition of a skybox. Basically it is an extremely large box which encompasses the entire playing area. A sky texture is then applied to the inside of the box so that it would give the illusion of look out over a world rather than a plane in an empty space. One handy tutorial that helped us achieve this can be located here

6.0 Work Break Down

Our group change log is the constant email communication between the group members. Every time there was a change whether it be small or large we sent an email to the group. As a result, we can go back and find the older iterations of code that we sent out, however due to the many parallel versions of the game not to mention the different fragments that we wrote, it is extremely messy to try and write a single log for it all. So rather than showing individual logs, we have amalgamated the individual progresses into a single visual representation. These dates are an approximation as the actual border between tasks is a little fuzzy. For ease of viewing, the chart will be included as a file as well as on the report.


Task Name
Team Member
9 3 2008
16 3 2008
23 3 2008
30 3 2008
6 4 2008
13 4 2008
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Procedural Landscaping
Ting
First Person Camera
York
Networking
Grigoris
Character Model
Karl
Model Skeleton
Ting
Sound
Grigoris
Projectile Motion (first attempt)
York
Model Skin
Grigoris
Mac compatibility
Karl
Game interaction in SDL
Ting
Loading model into the game
Grigoris
Linking Camera to Model 3rd person view
York
Linking Camera to Model 3rd person view
York
Projectile Motion (second attempt)
Ting
Building the world in SDL
Karl
Attaching model to Skeleton
Grigoris and Ting
Lighting
Ting
Project Report
York
Converting from GLUT to SDL
Karl, Grigoris, and Ting

Appendix

1. The premise of a tower defense game is simply to keep the monsters or "creep" that emerge from a starting gate from reaching the goal of a map. The map may be as simple as a large open area or it may be complex, but there is always at least one starting gate and one ending gate (there have been some derivations from this formula, but for the most part it has been done this way). The creep will take the shortest open path to the exit gate, so the player must build "towers" that shoot at the creep in such a way as so the creep are killed before they reach the exit. If there is no open path, the monsters will attack the nearest tower, destroying it, but otherwise, the creep will not attack. The logical strategy is to construct a maze of towers to maximize the creep path around the towers so the towers have more time to kill the creep.

2. Scorched Earth, Gunbound, and Worms are games where the player attempts to destroy the other player first. Each player controls an artillery piece or character and has control over force of projectile launch and angle of fire. The players take turns firing at each other with the idea that with each shot, their aim gets better until they are able to hit their opponent. Random elements such as wind and terrain make this task less than trivial. If the players are mobile then it adds another layer of strategy to the gameplay.

3. First person and third person perspective refers to the placement of the camera in games. In a first person perspective, the camera is placed where the character's eyes would be, giving the player the illusion of being in the game. From a story telling point of view, this perspective is used when the player's empathy for the character is prioritized above other gameplay aspects. From a simulation point of view, this perspective most closely resembles the feeling of driving a car, flying a plane, or even just the illusion of "belonging" in the game world. Third person

perspective positions the camera a small distance behind the player in the game. This can apply to many different types of characters, to use the previous example, behind a car or plane, or behind a person. The advantage this view brings is a greater awareness of the player's surroundings. As screens are only so wide, images beyond the field of view cannot be rendered at all, where as in reality, humans rely on peripheral vision. Third person perspective simulates this by showing objects and surroundings that otherwise be in the character's peripheral vision, but cannot be rendered due to physical constraints.