Josh Hartman, Chris Spence, Nicholas Wulf

April 23, 2008

**CAP4730: Computational Structures in Computer Graphics**

**Assignment 5: Make a Game**

**Team 5 - Group Report**

Our game is named Bangerang. It takes place far into the future, in the year 3045, when a powerful alien race known as the Xelthorians arrives at Earth and challenges their finest combatant to a game of Bangerang. It is a fast paced and exciting vehicular death match were only the fastest and strongest survive. And if the humans should lose, their planet will be forfeit.

The main goal of the game is to knock your opponent out of the arena. The more damage that you take, the farther you will go flying when you are attacked, making it easier for you to get knocked out. Each car is armed with a basic tackle accelerator that can give the car a short burst of speed when activated. Don’t use it recklessly though, because every time you try to tackle and miss there will be about a one second cooldown where your car will come to a stop and not move, making you an easy target. Also remember that getting hit in the back does twice the damage as a side collision, and getting hit in the front does almost no damage at all. To add to the mayhem, missles and mines have been added to the arena to help you blast your opponent away. However, you won’t just be handed these amazing weapons. Instead you’ll need to earn them by doing cool tricks in the arena such as jumping over gaps or doing spins in the air. Only by mastering the arena will you truly be able to master Bangerang.

The controls are:

W, A, S, and D – Basic movement

Spacebar – Tackle

J – Jump

K – Fire missile

L – Lay down mine

Enter – Pull up help screen

Collision detection is performed by assuming that the cars are cylindrical in shape and that colliding objects work as springs. The players will bounce off of the ground, walls, and other cars. When a collision between the player’s car and another object is detected, the force pushing against the player’s car will be proportional to the amount of the car’s depression into the object. In this way, collisions exactly mimic the physics of a spring collision. The nice thing about this approach to collision detection, is that when it is implemented correctly, many other physical phenomenon occur without needing to be specifically programmed for. For example, cars naturally bounce off of walls and floors at a reflected angle based on the normal of the surface, and climbing up a hill gets harder the steeper that the hill is. A dampening function is also performed on the velocity of a car as it is colliding so as to create an inelastic collision. If this were not the case, then the cars would continue to bounce along the floor indefinitely.

The goal in designing vehicle movement was to simulate hovercraft that would be able to slide around the arena. In order to create the slide effect, the velocity of each car in Cartesian coordinates is calculated, manipulated, and stored, and this velocity is added on to the Cartesian position coordinates every frame. Thus, if the velocity is non-zero and remains unchanged from frame to frame, the car will just slide along the course. However, we also wanted to simulate a small amount of friction so as to make movement around the course easier. To do this, the velocity of the car is scaled down by 99.5% every frame, meaning that the car will eventually slow down if left alone.

The camera is positioned to sit behind and a little above the car so that the car as well as the playing field directly in front of the car are all visible. To do this the angle the car is pointed in as well as position of the car are used to calculate where the camera should be positioned. Once the coordinates of the camera are determined, gluLookAt() is called and is given the position of the car as the center coordinate parameters. However, in order to smooth out the camera motion, the calculated camera position is actually used as the desired position. By storing the current camera coordinates in memory, and slowly bringing these closer and closer to the desired calculated coordinates, the camera motion appears much smoother and transitions nicely from position to position. Additionally, the slight lag in the camera’s position helps to accentuate the quick movements and high speeds of the cars as they zoom around the course. Finally, a system was put in place to compensate for sloped ground so that if a car is looking downhill the camera will rise so as not to run into the ground, and if the car is facing uphill the camera will sink so that it isn’t staring straight down at the car.

In order to produce a fast and effective particle system, a little research had to be done in order to discover what current techniques are used in industry. In the end we settled on a particle system that models explosions by shooting out a shower of orange particles in spherically random directions. At first, we were going to draw the particles as simple 3D shapes so that they could be seen from any angle. However, it soon became clear that we would need to speed up our particle calculations in order to have the game run smoothly. Therefore, we decided to do a little extra work and implement a system that would create a square that was always facing the camera. In this way, the particle is easily visible from all angles and is not as computationally expensive as a 3D model. We also employed a blending effect on the particles in order to make the explosions look slightly transparent. Furthermore, the velocity of each particle is stored in memory, and every frame the y velocity component is decremented and the positions are updated, giving the effect of gravity acting on the particles.

The game supports networking for only two players right now, though it wouldn’t be too hard to increase this number to three or four in the future. The code is split into two main sections, a client and a server. The server handles all the physics and gameplay operations. Each player will run the client code, which will communicate to the server code, telling the server what keys have been pressed and receiving information such as the positions of the vehicles so that everything can be redrawn each frame.