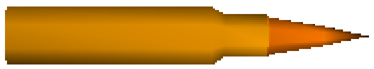
EECE 478: Project Report

Scorched Earth 3D



Submitted to:

Dr. Lee Iverson
Assistant Professor
Department of Electrical and Computer Engineering
University of British Columbia

On:

April 14, 2008

By:

Feltis, Lane Parker	(64045040)
Teymory, Neema	(89501042)
Wong, Andrew Cheun Ho	(19814037)
Yin, Ying	(75821041)

Table of Contents

Table of Figures

1 Introduction

Scorched Earth 3D is bigger, more powerful, more explosive, more fun and one full dimension better than the classic original computer game Scorched Earth. With state of the art computer graphics, realistic tank models, multiple level designs, intense wind, new-age music, mind-blowing sound effects and any gravity imaginable, this multiplayer game is sure to capture the interest of all gamers. In Scorched Earth 3D, a predetermined number of players on a randomly generated terrain take turns in launching attacks against one another until only one remains.

Throughout this paper we will discuss the game features implemented in this version of Scorched Earth 3D. Most game-play decisions are based on the original 2D Scorched Earth. A primary goal of the project is to remain true to the original concept while extending the game into the third dimension. The Requirements section provides motivation for each game feature and the rationale for each decision. In the following section, the main Project Modules and their implementations are described. Since one of the main focuses of the game was to improve the original 2D Scorched Earth into a counterpart 3D version, some of the advanced graphical techniques used by the game engine are described. Finally, the conclusion discusses features that were not included in the current version, the list of currently known bugs and any upgrades that are planned for future versions.

2 Requirements

The original Scorched Earth inspired the creation of this game and thus has great influence on the implemented game features. Although Scorched Earth has an amazingly large number of features, options and advanced game-play, we were unable to incorporate all of these great original aspects due to limited time; however we did implement the ones that we thought were most critical. The implemented features are described in detail below.

2.1 Turn-Based with Limited Mobility

There were in reality only two options for the game-play—turn-based or real-time. Although most games nowadays are real-time, the original Scorched Earth was turn-based with the ability to move the tank a marginal distance on any given turn. We felt that this was one of the key features to the tactical game-play of the original game; therefore, we decided to implement the game-play with a similar paradigm. Likewise, the player perspective is 3rd person to stay true to the original.

2.2 Multiplayer

Although the original Scorched Earth has a maximum of 8 players, the game is able to provide support for only up to 4 players. The reason for this is two-fold. Firstly, our tank model is relatively complex, requiring a fair amount of graphical processing power on a typical laptop graphics card. Secondly, adding tanks to the game quadratically increases the number of calculations performed by physics engine during collision detection. If we were to exceed our current limit of 4 players, the game frame rate would take a potential performance hit. We could have redesigned the tanks to be less detailed to allow for more players; however, with more players a larger terrain is generally needed so the game is still playable (it is not very fun if everyone is right next to each other). But, our terrain is also expensive to render; if we made our terrain twice the size and resolution (to maintain terrain detail) once again, our frame rate would a significant performance hit. If we were to lower the detail of both, perhaps we could have added more players, but the amount of time needed to change and retest the system exceeded the amount of time that remained when the idea was suggested. Also, the reduced quality of the game may have made the game visually unappealing, negatively affecting the user experience.

2.3 Random Terrain Generation

One of the best features of original Scorched Earth was that even though every level was made of green dirt, it was always randomly generated and thus, each game was different because the level was never the same. We decided to take it one step further by keeping the random terrain generation, but also adding different types of levels (mountainous, hilly, islands, and flat) to maximize the lasting appeal of the game.

2.4 Dynamic Environment

Another great aspect of the original Scorched Earth game, aside from the random terrain generation, is the dynamic environment. By changing the wind every turn or by being able to change gravity settings, players are able to play the same game in a completely different manner. There are countless possibilities of environments since the wind's speed and direction change every turn (in a random fashion) and the gravity can be altered before a match. The main purpose of this was again to maximize the lasting appeal by making the game feel different and more importantly difficult to master. Like the original, Scorched Earth 3D provides these options for altering wind and gravity effects.

2.5 Sound Effects

As is true with any game, the music and sound effects can either make it or break it. When the original game was made, sound effects existed for explosions, bullet motion, and general game-play; however, they were very simplistic in nature. Scorched Earth 3D boasts WAV and MP3 support for a rich music and sound experience.

2.6 User-Friendly Controls

The original Scorched Earth had minimal, easy-to-use controls. It provided controls to move forward, backward, raise and lower the barrel, increase and decrease firepower, change weapons and fire weapons. We thought that we should maintain this simplicity as much as possible because a game that is simple to play will appeal to a larger audience. Scorched Earth 3D provides the same set of controls, with the additional ability to turn the tank body and head independently (change its heading and barrel rotation) as well as camera controls.

2.7 Heads-Up Display

In the original Scorched Earth, a simple information bar is used to provide the user with various game data, such as current velocity, wind direction, and selected weapon. In Scorched Earth 3D, this concept is taken one step further by using a player heads-up display. The heads-up display provides various types of information to the user, including the current wind speed and direction, opponent positions through the radar, weapon power, and amount of remaining health and fuel. The heads-up display is designed in a way to impede on the current player's view the least amount possible.

3 Software Design

Scorched Earth 3D has been written entirely in standard C++ using the OpenGL API. Two additional support libraries were used to provide code portability: the SDL and FMOD libraries, for graphic and sound support respectively.

The game engine uses a variable frame rate rendering technique to take advantage of higher end graphics systems while at the same time maintaining a consistent gaming experience to users with different levels of hardware performance. Figure 1 below shows the high level relationships between software packages of the Scorched Earth 3D implementation. For the sake of brevity, only several of these packages will be discussed in detail. The package names are descriptive of their function; for example, the Physics package contains the physics engine and the Terrain package contains the terrain engine. For additional details, please refer to Section 4 as well as the project source code—all source is fully commented.

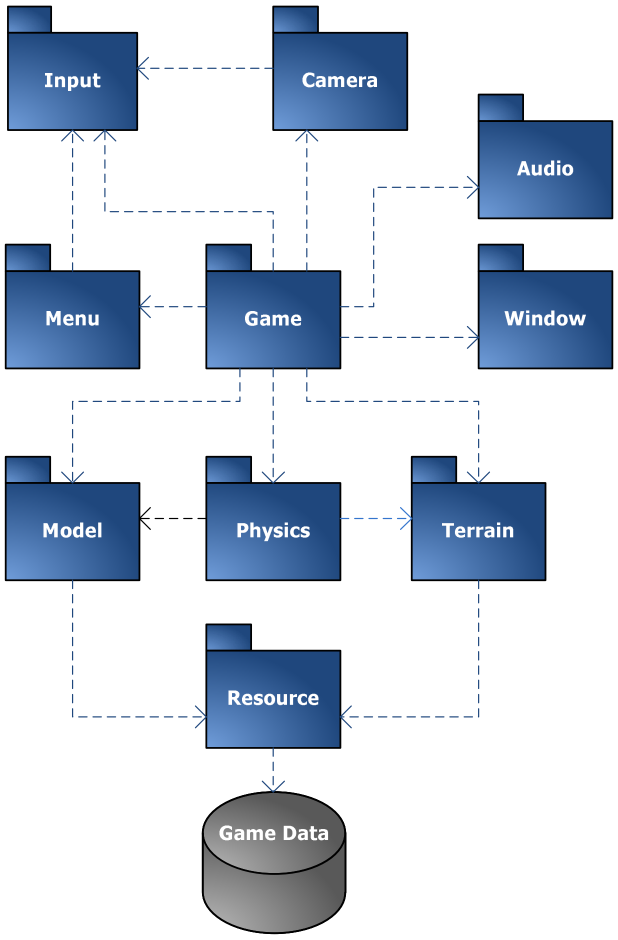


Figure 1 – Software Package Relationships

3.1 Input Package

The input package provides an abstraction for game controls to the rest of the program. By using the SDL, the input package provides infrastructure to allow game classes to easily sample keyboard and mouse data through the game controls abstraction. Services are also provided for event listeners of both the keyboard and mouse inputs.

3.2 Window Package

The window package is a simple abstraction that hides the details related to the created rendering video screen. Windows of various sizes can easily be created, and either set to display in full screen or windowed mode. The window package makes use of SDL to provide this support.

3.3 Resource Package

The game data is stored on disk and includes textures, models, sound effects, and MP3 music files. The Resource package provides uniform access to this data for the entire program. This allowed us to buffer the effects of file changes and updates throughout development. All textures are pre-cached into video memory at game startup to reduce loading times later in the game.

3.4 Game Package

The game package is the heart of the Scorched Earth 3D implementation. It makes use of nearly all other packages and controls game state transitions and logic. In its simplest form, the entire game is a state machine with two possible types of states—*control* and *animate* states. In a control state, certain game operations can be performed by the user through interacting with the mouse and keyboard; in other words, the user is in control of the game in these states. In an animate state, no control is granted to the user while some form of animation is played (bullet firing for example).

Each state—whether a control or animate state—is defined by two unique routines. The routines are its *logic* and *render* routines. The state logic routine is used to update and necessary game state information based on inputs or game events, while the state render routine is used to render the entire game in its current state.

The game engine then runs in a loop that continually executes the current state's logic routine followed by its render routine. This implementation is simple, and therefore very easy to manage. Because of the turn based nature of the game and the limited number of game states, this was the implementation of choice.

4 Project Modules

In this section a discussion of the primary modules of the game engine are provided. The game engine is broken into six main categories:

- Sound
- Heads-Up Display
- Game Menu
- Terrain Generation
- Physics Engine
- Game Models

The reason the game is broken up in this way is because each of these modules can be developed independently; each module does not need to know about any other module to operate correctly. The sound module controls any sounds played in the game. The heads-up display controls any player data that needs to be displayed to the user. The game menu controls the input settings desired by the user. The terrain generation controls the environment the tanks will battle in. The physics engine controls how objects behave in the game and finally the game models are used to manage the rendering of game objects.

4.1 Audio

Scorched Earth 3D sets an atmosphere of intensity with looping background music tracks during the menu screen and game play. A degree of realism is added on top of this by associating appropriate sound effects to various actions such as tank movement, bullet firing and explosions. Sound support in Scorched Earth 3D is accomplished through two separate frameworks — one for playing simple sound effects (.wav files) and another for playing background music in the form of compressed audio (.mp3 files).

4.1.1 Sound Effects

The framework for sound effects in Scorched Earth 3D is implemented as a wrapper around the SDL audio class. The wrapper increases the modularity of the SDL class by defining sound sample and sound engine objects. Each sound sample (a .wav file) that is loaded can be assigned its own maximum volume and attenuation, thereby allowing multiple sound effects to play simultaneously at seemingly different distances from the user. A higher level singleton class — the sound engine — is then responsible for initializing, mixing and keeping track of all sound samples as well as setting the universal game volume.

4.1.2 Music

Given the inherent complexity of compressed audio decoding, the task of playing background music is delegated to a popular third-party audio library called FMOD, which is featured in games such as Guitar Hero III, World of Warcraft and BioShock [4]. A wrapper class around FMOD exists to handle multiple music samples, similar to that for sound effects but slightly simpler since background music only needs to be played and stopped and does not need to be attenuated.

4.2 Heads-Up Display

In Scorched Earth 3D, each player has his/her own colour-coded heads-up display that reveals information about the player's tank and the surrounding environment. The heads-up display is rendered as a collection of distinct components, much like how an aviation dashboard is assembled. Each component is defined as a series of OpenGL calls in a display list, which is created (using glNewList/glEndList) during initialization of each heads-up display object and executed each time the heads-up display is updated. A screenshot of the heads-up display can be found in Appendix D.

The majority of the heads-up display components are relatively simple: the corner overlays are disks with a specified number of slices; the wind direction indicator is a three-dimensional arrow model that is rotated as necessary; and the fuel and health meters are created using overlapping quadrilaterals. Any relevant text-based data is also written to the screen with the same font renderer used in the game menu and other parts of the game.

The real-time radar, however, consists of multiple subcomponents and requires some more intricate logic to achieve its functionality. First, a circular radar frame is drawn using a specified radius. To make the radar useful to the user, the coordinate system is rotated so that the camera direction is aligned with the radar's positive y-axis prior to any drawing any radar content. Next, a texture of the terrain is generated using the generated terrain's elevation map and mapped to a simple quadrilateral. Before drawing, this terrain texture is translated as necessary so that it is centred on the current player's tank. To show only the terrain within the radar's scanning range, this texture is essentially "cropped" using a circular stencil with a radius one pixel smaller than that of the radar frame. Functionally, this is accomplished by writing the coordinates of a disk of the desired size into the stencil buffer, preventing these values from being modified by calling glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP) and enabling stencil testing.

Dots representing opponent tanks are then created for the radar display. Dots are only drawn to the screen if their position lies within a certain angle behind the animated radar sweeper. To determine the placement of these dots, each tank position is projected down to the x-y plane and its distance from the current player's tank is scaled according to the ratio between the radar's scanning range and the radius of the radar display. Finally, the radar sweeper is drawn at an angle that is regularly updated according to an update time-step. The fading effect of the sweeper trail is achieved by using a blue-to-black gradient bitmap image with its black portion blended away through alpha blending.

4.3 Game Menu

An important element for all games is a menu that allows users to change game settings. The Scorched Earth 3D Game Menu allows users to change the number of players, the player data (name and color), the terrain type (mountainous, hilly, islands or flat), the wind type (none, constant or changing) and the gravity value. The Game Menu's main purpose is to provide a simple, easy-to-use interface so that a user may adjust the game to their desired settings. In future versions, if any option is added (i.e. computer players or multiple weapons) the Game Menu would need to be updated to allow the user to adjust these options as well. The settings that the user decides upon are fixed for the match once the user selects the "Start" button. Other modules use the information stored by the menu to change or initialize game states appropriately.

4.4 Terrain Generation

The Scorched Earth 3D game provides four distinct classifications of terrain to provide players with a choice for their battleground. These terrain types are classified as follows:

- Flat terrain
- Hill terrain
- Island terrain
- Mountain terrain

Since the terrain is meant to be an open battleground for tanks, a simple height map model was adopted for terrain representation. The terrain is described as a grid of points, where each point has a height associated with it. All grid points are normalized to the range $0 < (x, y, z) < 1$. The normalized grid can then be scaled and its points joined together in a series of triangles to produce the final graphical representation. The terrain generation is managed exclusively by the game Terrain Engine. Figure 2 below provides a sample height map that the Terrain Engine may produce.

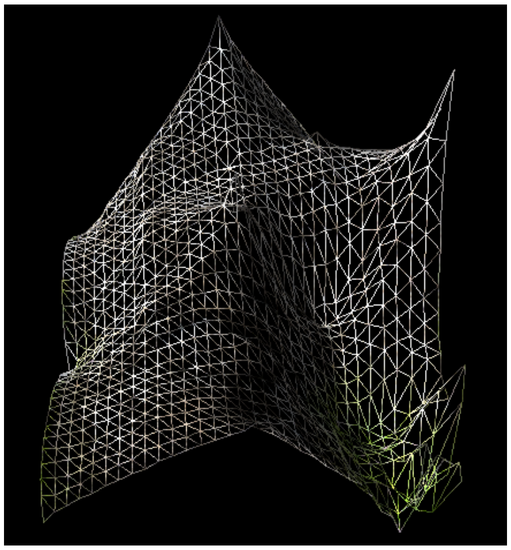


Figure 2 – Mountain Height Map Generated Using Terrain Engine

Because the choice of terrain is limited to four possibilities, an important requirement for Scorched Earth 3D is that no two terrains be identical. Whenever a terrain is selected for battle, a *new* unique height map must be generated by the Terrain Engine. The only exception to this case is for the flat terrain, since it has no variation in its height map. To provide this functionality, the Terrain Engine employs two different height map generation algorithms: *Parabola Superposition* and *Diamond Square.* Additionally, since a height map is not known *a priori*, the Terrain Engine procedurally generates a new texture each generation for realistic looking environments. These features are described in more details below.

4.4.1 Parabola Superposition

Parabola Superposition, as its name suggests, uses the parabola equation to generate a height map. This algorithm is suited for creating rounded height maps, and as such is used by the Terrain Engine to generate both the hill and island terrains. This algorithm is adapted from the Robot Frog web tutorial [1]. In essence, the algorithm has four steps:

1. Choose a random x, y, and z
2. Create an inverted parabola using choices in Step 1 as the origin
3. For each point in the height map calculate its height using the parabola equation
4. If the height in Step 3 is greater than zero, add its value to the current height at the point
5. Repeat Steps 1-4 for as many iterations as desired

The equation used to create the inverted parabola in Step 2 is as follows:

$$height = z^2 - \left((x - x_0)^2 + (y - y_0)^2\right) \quad [1]$$

Where the values x, y, and z are chosen in Step 1 and the values x_0 and y_0 are the height map positions used in Step 3. By varying the choice for z in Step 1, the Terrain Engine can produce a wide variety of different hilly terrains. Figure 3 below shows the height map generated for varying algorithm iterations.

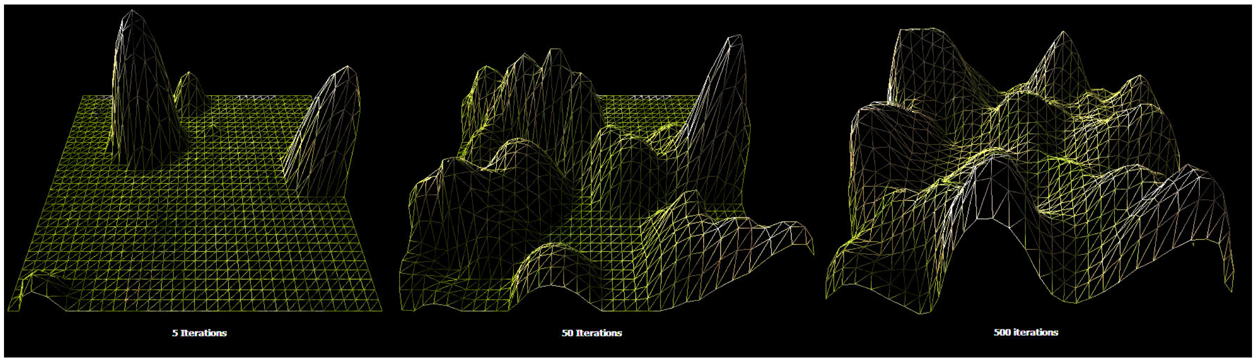


Figure 3 – Height Maps Created with Parabola Superposition for 5, 50 and 500 Iterations

4.4.2 Diamond Square

The Diamond Square algorithm is a recursive fractal generation algorithm. This algorithm is best suited for creating more jagged and rocky types of terrain, and as such is used by the Terrain Engine to generate the mountain terrain. The two major recursive steps of this algorithm are how it earned its name—the so called *diamond* and *square* steps.

In the diamond step, the heights at four corners of a square are averaged to produce a height at the center of the square. Then, a random value is added to this average [2]. This step is called the diamond step, because once done, the points "touched" by the algorithm produce a set of diamond shapes.

In the square step, similar to the diamond step, the heights at four corners of a diamond are averaged to produce a height at the center of the diamond and then a random value is added to this average [2]. Once done, the points "touched" by the algorithm once again produce a set of squares.

An iteration of the algorithm consists of performing both the diamond step and square step once. After each of the iterations, the random value mentioned above is reduced by a fixed percentage (typically 50%) [2]. This can be thought of as decay in variability—In earlier iterations heights vary greatly, while in later iterations they begin to smooth out. The recursion terminates once the grid points cannot be further broken down into smaller diamonds or squares. Figure 4 below shows the height map created by the algorithm using a different number of iterations.

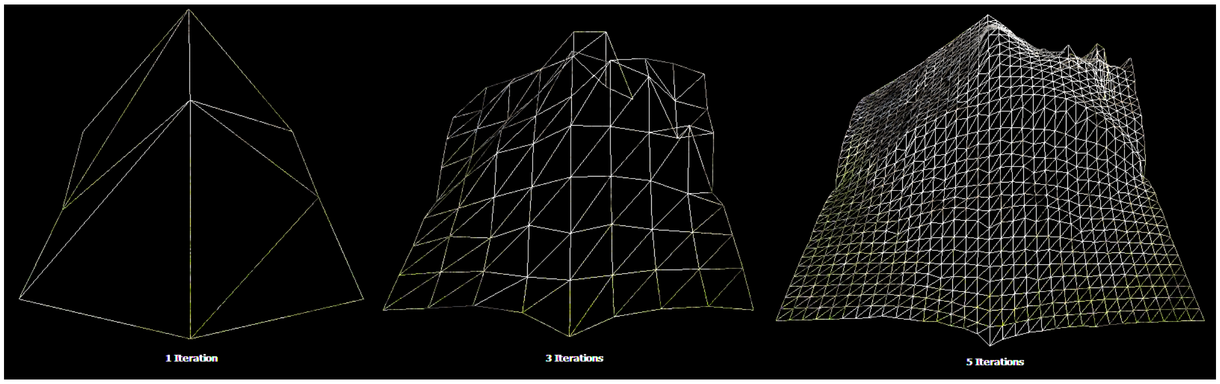


Figure 4 – Height Maps Created with Diamond Square for 1, 3 and 5 Iterations

Notice that the each iteration shows a different number of grid points, in practice however, the total number of points are pre-calculated and then iterated upon rather then generating new points as they become necessary. The Terrain Engine limits the Diamond Square algorithm to five iterations for performance considerations.

4.4.3 Texture Generation

A critical component to realistic looking terrain is the texture. For example, the tips of a mountain should appear snowy, while its valleys should appear grassy. It is then evident that the terrain texture is dependent on its height map. For this reason, the Terrain Engine procedurally generates a new texture each time a terrain height map is generated.

The generated terrain texture is a function two things: texture layer components and a terrain height map. The height map is used to select and mix texture layers to produce the final texture which is a combination of all its texture layers. Figure 5 below provides an intuitive illustration of this process.

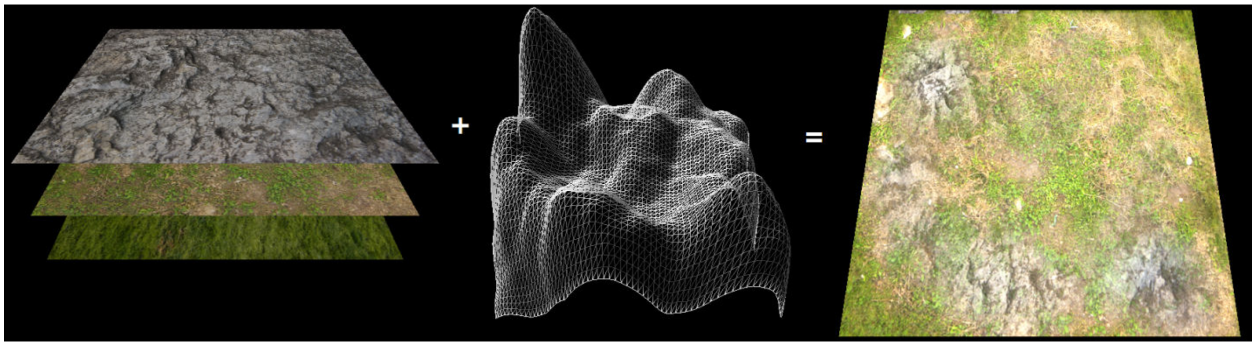


Figure 5 – Terrain Texture Generation Process

The height at each pixel is used to determine the contribution of each layer to the final texture. Since the texture is a much higher resolution than the terrain height map, linear interpolation of the height map is used to extract pixel heights during texture generation. If there are *N* texture layers, then the contribution *C* of a given layer *L* at the pixel with height *H* is given by the following expression [2]:

$$C = 1 - \left|\frac{L}{N} - H\right| \cdot N \qquad H, C \in [0,1]$$

The percentage of contribution is clamped to the range [0, 1]. Intuitively, *C* is the percentage of color that texture layer *L* should contribute to the pixel with height *H.* The absolute value in the expression determines how “far away” the layer is from the height. Multiplying by *N* causes the second term to quickly grow when the layer is far away from the height, forcing its contribution to zero.

4.5 Physics Engine

The physics engine was built from scratch without using any external libraries. It is responsible for the dynamic simulation and collision detection in the game. The code for the physics engine can be separated into three main parts: one part for dynamic simulation; one for collision detection; and one for mathematics support such as vector, matrix, transformation and quaternion calculations. All rotations are specified using quaternions.

4.5.1 Dynamic Simulation

The *DynamicsWorld* class is used to represent the world and manage all the rigid bodies in the world for physical simulation. Rigid bodies include the bullets and tanks. The world transformation of every rigid body is updated by the physics engine every frame according to the time elapsed between consecutive frames.

4.5.1.1 Projectile Motion of Bullet

The projectile motion of the bullet is affected by the acceleration due to gravity and the wind effect. If the wind velocity is zero, the only force on the bullet is the gravitational force, and the air resistance is ignored. However, if the wind velocity is not zero, the effect of wind on the bullet is modeled as a drag force due to air. The drag force is in the direction opposite to the relative velocity of the bullet and wind. The drag force is calculated using the following formula:

$$\boldsymbol{F}_{1}drag = -1/2 \cdot density_{1}fluid \cdot coeff_{1}drag \cdot [\![area]\!]_{1}(cross\ sectional) \cdot |,v - relative.| \cdot v_{1}relative \tag{1}$$

where $v_{relative} = v_{bullet} - v_{wind}$.

The resultant acceleration (**a**) of the bullet is a vector combination of acceleration due to gravity and acceleration due to the drag force. The displacement (**d**) and new velocity (**v**) is calculated in each time step (t) according to the kinematics formulae:

$$\boldsymbol{d} = \boldsymbol{u}t + \frac{\boldsymbol{a}t^2}{2} \tag{2}$$

where $\boldsymbol{u}$ is the velocity in the previous frame and

$$v = u + at \tag{3}$$

Then the new position of the bullet is updated and checked against the terrain boundary. The bullet is always rotated in the same direction of its current velocity.

4.5.1.2 Tank Motion

The motion of the tank is more complicated due to the constraints imposed by the terrain. The tank should always adhere to the surface of the terrain, and this is made more difficult on hilly or mountainous terrain. No simple dynamics can solve this problem easily. The approach taken in the physics engine is to separate the position and rotation calculations. The new position is calculated first and the rotation is updated according to the terrain heights at the new position.

In every simulation step of time *t*, the tank's new displacement and velocity are calculated according to its linear velocity and acceleration due to gravity using Equation (2) and (3). The tank's new position in the world is then updated. The terrain heights at the center and other four points (front, back, left and right, see Figure 6) of the tank are evaluated.

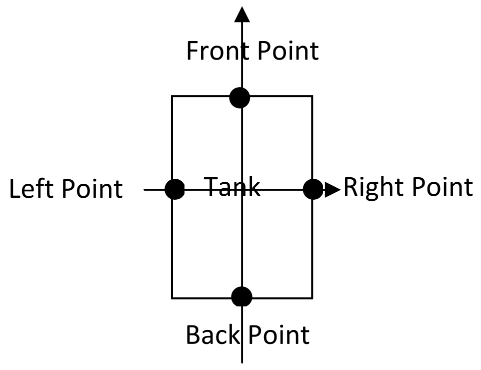


Figure 6 – Tank Collision Shape

The actual height of the tank is computed as the maximum of three values:

1. Its current height at the new position (maybe airborne)
2. The terrain height at its center
3. The average terrain height based on front height and back height (when the tank is on a concave section of the terrain)

The rotation of the tank has two components: rotation (R_1) of its major axis (front point – back point) and rotation (R_2) of its minor axis (right point – left point). The major axis in the tank's original object frame is the x-axis and the minor axis is the y-axis. The overall rotation is R_2 x R_1. All rotations are specified using quaternions. It is important to check boundary cases where the axis of rotation or angle of rotation is zero to avoid division by zero. If rotation is zero, the quaternion is set to be $q = (1, \vec{0})$ and updating the rotation will have no effect.

Please see Figure 8 and Figure 9 in Appendix D for screen shots of the tank on terrain.

4.5.2 Collision Detection

In every time step, after physical simulation, every object in the *DynamicsWorld* is checked against every other object for possible collisions. Two types of collision are considered: sphere box collision and point terrain collision.

4.5.2.1 Sphere Box Collision

The bullet is modeled as a sphere centered at the tip of the bullet. Before explosion, the collision shape of the bullet is a point with radius 0. After explosion, the collision shape of the bullet is a sphere with radius equal to the radius of the explosion. In this way, the tank can be hit by the explosion even if it is not directly hit by the bullet. The tank is modeled as a box.

To do accurate box collision detection, it is important to test the collision with an axis-aligned box. The box in its original object frame is axis-aligned and centered at the origin. Hence, for the collision detection, it is only necessary to transform the sphere to the box's frame. Let the world transformation of tank (box) be T_{tank} and the world transformation of the bullet (sphere) be T_{bullet}. Then the transformation of the bullet into the tank's frame is given by $T_{tank}^{-1} \times T_{bullet}$.

Once the bullet is transformed into the tank object frame, the remaining calculations are simple. The dimensions of the box are inflated by the radius of the sphere. Then the center of the sphere is checked to see whether it lies inside the inflated box by checking its x, y, z values against the length, width and height of the box respectively.

4.5.2.2 Point Terrain Collision

Point terrain collision detection is used for determining whether the bullet has hit the terrain or not. The position of the point in the world is evaluated. Its height is then checked against the height of the terrain at that x, y coordinate. If the height of the point (bullet) is smaller or equal to the terrain, then a collision has occurred.

4.6 Game Models

Game models are a convenient way of packaging material and vertex data for an object that needs to be repeatedly drawn. Scorched Earth 3D makes use of seven different models:

- Tank model
- Bullet model
- Arrow model
- Sun Moon Box model
- Cloud Box model

- Night Box model
- Day Box model

The tank model is used to draw the player tanks on the battlefield and manage animated player motions, the bullet model is used to draw the tank firing weapon, the arrow model is used to draw the heads up display wind indicator, and the collection of sky boxes are used to create realistic environmental effects. The sky boxes are collected together as models primarily because of the texture and materials management support offered by model objects.

To manage model data, a custom model format called the Simple Model Format (SMF) was developed and implemented using MAXScript. Using this approach, models could then be created using 3ds Max and easily exported into SMF format. The SMF format extracts only essential 3ds model information required by the Scorched Earth 3D game engine, providing the advantage that SMF files are compact in size. As an added advantage, SMF files are also formatted in plain ASCII text and are therefore easy to read and understand. This simplifies the process of creating specially designed classes for parsing and interpretation within the game engine. For a detailed description of the SMF format see Appendix C.

5 Advanced Features

This section highlights some of the more advanced components used by the Scorched Earth 3D game engine. Each of the advanced features is briefly explained and examples relating the game engine are provided.

5.1 Portability

The entire Scorched Earth 3D project has been developed in Standard C++ and uses only two external libraries which both provide cross platform compatibility. These libraries are SDL for graphics support and FMOD for sound support. The current build of the project compiles and runs using the Microsoft Visual C++, Apple Xcode, and Cygwin g++ platforms. All necessary libraries and makefiles are distributed with the project so it can easily be recompiled and tested on any of the above platforms.

5.2 Custom Model Format

As mentioned in Section 3.6, the Scorched Earth 3D engine uses a completely proprietary modeling format designed specifically for use by Scorched Earth 3D. The modeling format takes advantage of the powerful modeling tool 3ds Max. Models can rapidly be developed using 3ds Max, exported to SMF, and immediately used by the Scorched Earth 3D game engine.

5.3 Game Control Console

For advanced debugging support and real time game configuration, the Scorched Earth 3D engine provides an in game control console that can be used to control or modify nearly any aspect of the game. From frame rate control to player color details, the control console improves the debugging and game configuration process by allowing developers to "try" different things very quickly without the need for recompiling, or even restarting the executable. The control console can be accessed by pressing the tilde (~) key.

5.4 View Frustum Culling

Maintaining a high frame rate is critical to the player's game experience. A technique to improve frame rate is to avoid sending data to the graphics pipeline when it is not visible. View frustum culling is a simple test that avoids rendering objects that do not appear within the current viewing volume. The Scorched Earth 3D game engine provides a complete infrastructure to support this type of test. All tank players and fired bullets are tested against the current camera's view frustum before any rendering takes place.

5.5 Translucent Objects

To make game scenes more visually appealing, a variety of blending techniques are used to create transparent object effects. The current Scorched Earth 3D game version provides transparent cloud and related skybox effects, translucent water effects in the island terrain, semi-transparent heads-up display, and alpha decaying explosions. Also, while not strictly a part of the game, the Game Control Console also makes use of blending to appear translucent while it is being displayed.

5.6 Level of Detail

The Scorched Earth 3D engine makes use of standard OpenGL texture mipmaps to display textures at different levels of detail. This technique prevents artifacts due to minification as well Moiré effects.

5.7 Procedural Modeling

As mentioned in Section 3.4, the Scorched Earth 3D engine procedurally generates terrain at runtime as well as the terrain's matching height map texture—no two generated terrains are identically. Two different terrain generating algorithms are used depending on the type of terrain chosen to produce the terrain height maps. For complete details, see Section 3.4.

5.8 Physics Simulation

As mentioned in Section 3.5, the Scorched Earth 3D engine supports physics simulation through the physics engine. The physics engine is used to simulate projectile motion, gravity effects, as well wind drag effects. The physics simulation also includes collision detection for bullets and tanks.

5.9 Collision Detection

The Scorched Earth 3D engine performs a variety of rigid body collision detection tests. By using bounding box principles, the physics engine is able to efficiently test collisions between bullets and tanks, bullets and terrain, and tanks and terrain. The collision detection system is also responsible for maintaining a world boundary where tanks and bullets may not cross.

5.10 Music and Sound Effects

As previously discussed in Section 3.1, the Scorched Earth 3D engine provides extensive support for audio. The sound engine allows the playback of WAV files in either one-time or looped playback, while the music engine provides support for the playback of MP3 compressed audio files. The current version of Scorched Earth 3D has support for the following sound effect events:

- Tank idling
- Tank motion
- Tank barrel movement
- Tank/Terrain collision
- Bullet/Tank collision
- Bullet firing
- Bullet explosion

and the following music events:

- Loading/Title/Main menu music
- Game play background music

6 Known Bugs

The current version contains several minor bugs that have not yet been resolved:

1. A bullet's damage can only be applied to one target.
2. The game may crash on the OSX platform if there are more than 3 players (works fine on PC).
3. In some rare cases, the tank turns upside down momentarily. This is due to the fact that rotation of the tank does not consider the up vector. An additional check is necessary to avoid this.
4. Tanks can be moved while they are airborne during the beginning of a round (it is not clear whether this should be considered a bug or not).
5. In some rare cases, the bullet following camera mode faces the rear of the bullet rather than its front (although this is a nice effect, it was unintentional, and is therefore considered a bug).

These bugs are all minor and easily resolved in the next version of the game; currently, they do not have any adverse affects on game play since they are very subtle and more often than not they appear to be the intentional behavior of the game.

7 Conclusion

Both unit testing and integration testing were done to ensure the correct functionality of the game. Individual test drivers are used to test different modules. For example, the sound module, the terrain generation module, the physics engine module and the game models module were all tested separately before being integrated together. The final game was also tested extensively under various conditions: different platforms, different number of players, different terrains, and different gravity and wind parameters. The advanced features mentioned in Section 4 were also tested.

The alpha version of the game has successfully met all the requirements of the project listed in Section 2. This list is in accordance with the ones that were mentioned in the proposal. Due to time constraints, the optional features proposed were not implemented. Game AI was not implemented because it was not crucial to the ability to play the game. It is also unclear whether a user would like to play against a computer which has a clear advantage in calculating the correct angle for shooting; on the other hand, if the AI is made to be intentionally unintelligent, the user may not enjoy the game either. Advanced weapon types and network play also were not included; however, they will be implemented in the next version.

Overall, Scorched Earth 3D is a strategic game that is fun to play with up to three other friends. Comparing with the original 2D version of the game, Scorched Earth 3D has more appealing graphics, more realistic physics simulation, and a much richer sound experience.

Appendix A: Multimedia References

This appendix provides a comprehensive list of every texture and sound file used in this project. The resource pathname is provided and its source either by description or URL.

1. data/audio/menu_bg.mp3
   - http://gh.ffshrine.org/song/6587/5
2. data/audio/gameplay_bg.mp3
   - http://downloads.khinsider.com/game-soundtracks/album/starcraft-game-rip
3. data/audio/explosion.wav
   - http://simplythebest.net/sounds/WAV/sound_effects_WAV/miscellaneous_wavs_2.html
4. data/audio/tank_hit.wav
5. data/audio/tank_shot.wav
   - http://www.a1freesoundeffects.com/weapons.html
6. data/audio/victory.wav
   - http://www.allmusiclibrary.com/free_sound_effects.php
7. data/audio/tank_idle.wav
   - http://home.csumb.edu/c/contospaul/world/samples.full/Samples11:2f05/Tightdigital/Reals/Generator.wav
8. data/audio/tank_head.wav
   - http://www.pacdv.com/sounds/machine_sounds.html
9. data/explosion/round.bmp
   - http://images.google.ca/imgres?imgurl=http://www.webdesign.org/img_articles/7474/explosion-slice_26.gif&imgrefurl=http://www.webdesign.org/web/photoshop/special-effects/creating-a-planet-explosion.7474.html&h=464&w=390&sz=73&hl=en&start=17&um=1&tbnid=vBllpq__P2tIKM:&tbnh=128&tbnw=108&prev=/images%3Fq%3Dexplosion%2Btexture%26um%3D1%26hl%3Den%26sa%3DN
10. data/font/arial.tga
11. data/font/tahoma_bold.tga
    - Generated with LMNOpc Bitmap Font Builder

12. data/icon/ScorchedEarth3D.ico
    - Generated with Visual Studio icon editor
13. data/icon/ScorchedIcon.bmp
    - Generated with Adobe Photoshop, based on original image from http://www1.istockphoto.com/file_thumbview_approve/4415674/2/istockphoto_4415674_flame_symbols.jpg
14. data/model/ekg.tga
15. data/model/arrow.bmp
16. data/model/radar.bmp
17. data/model/tread.bmp
18. data/model/night.bmp
19. data/model/clouds.tga
20. data/model/moon.tga
21. data/model/sun.tga
    - Generated with Adobe Photoshop
22. data/model/camo.bmp
    - http://www.texturemaker.com/incoming/Camouflage_Resampled.jpg
23. data/model/sun_moon_box.smf
24. data/model/cloud_box.smf
25. data/model/night_box.smf
26. data/model/day_box.smf
27. data/model/bullet.smf
28. data/model/arrow.smf
29. data/model/tank.smf
    - Models created with 3ds Max (export to SMF format)
30. data/terrain/grid.tga
    - Generated with Adobe Photoshop
31. data/skybox/*
    - http://www.hazelwhorley.com/skyboxtex2_bitmaps.html

32. data/terrain/dirt.bmp

- http://www.cgtextures.com/texview.php?id=635&s=S&PHPSESSID=a573f75b839777379cbbce406f0ac156

33. data/terrain/grass.bmp

- http://www.cgtextures.com/texview.php?id=10748&s=S&PHPSESSID=a573f75b839777379cbbce406f0ac156

34. data/terrain/grass_rock.bmp

- http://www.texturewarehouse.com/gallery/index.php?action=showpic&cat=6&pic=230

35. data/terrain/motherboard.bmp

- http://www.cgtextures.com/texview.php?id=20358&s=S&PHPSESSID=2511796d93e02256df0749f92cb76889

36. data/terrain/rock.bmp

- http://www.cgtextures.com/getfile.php/3063/l/RockLayered0033_1_L.jpg?PHPSESSID=a573f75b839777379cbbce406f0ac156

37. data/terrain/sand.bmp

- http://www.cgtextures.com/texview.php?id=11705&s=S&PHPSESSID=19a4a4b6e0b16e6db07be2cf6b4a08f7

38. data/terrain/seabed.bmp

- http://www.cgtextures.com/texview.php?id=5252&s=S&PHPSESSID=19a4a4b6e0b16e6db07be2cf6b4a08f7

39. data/terrain/snow.bmp

- http://www.cgtextures.com/texview.php?id=9878&s=S&PHPSESSID=6359b905f13261d8284c8ac95f887a3b

40. data/terrain/water.bmp

- http://backgroundsarchive.com/tiles/sample.php?backgroundID=1174

Appendix B: Game Controls

The following table is a summary of game controls. SDL supports multi-key capture, so camera controls can be combined (for example, 'Shift + s + Left Arrow' will zoom out and orbit left at the same time) and tank/tank barrel controls can also be combined.

Action		Key Combination
Global	Quit Round/Game	Esc
	Show/Hide Console	` (~)
Camera	Zoom In	Shift + w
	Zoom Out	Shift + s
	Orbit Left	Shift + Left Arrow
	Orbit Right	Shift + Right Arrow
	Orbit Up	Shift + Up Arrow
	Orbit Down	Shift + Down Arrow
Tank	Move Forward	Up Arrow
	Move Backward	Down Arrow
	Turn Left	Left Arrow
	Turn Right	Right Arrow
	Fire Cannon	Spacebar
Tank Barrel	Raise	w
	Lower	s
	Turn Left	a
	Turn Right	d
	Increase Firing Velocity	=
	Decrease Firing Velocity	- (+)

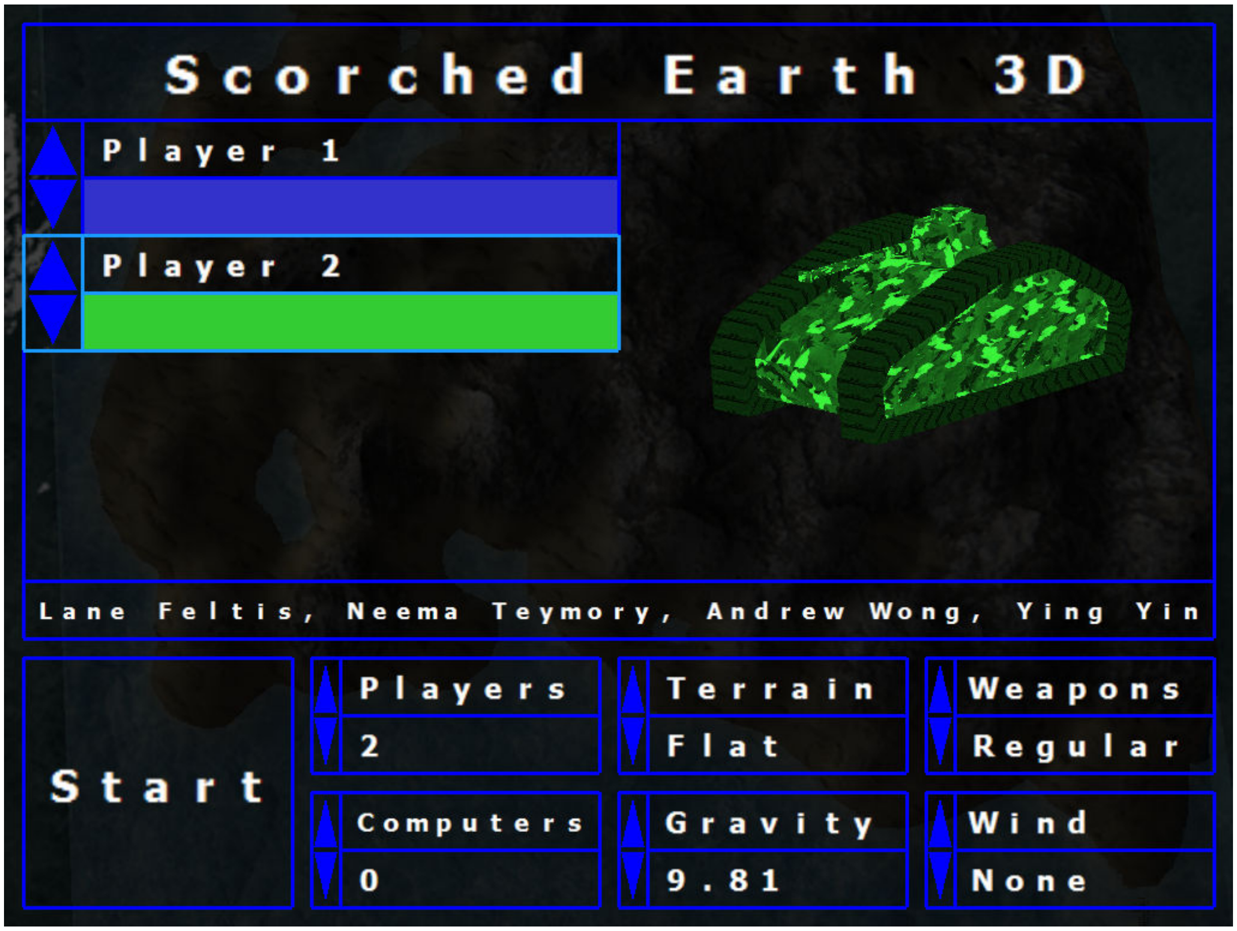
Appendix C: The SMF Format

The following is a table description of the Simple Model Format (SMF) v2.0

Line	Data	Type	Example
1	Version string	1 string	"Simple Model File v2.0"
2	Number of objects	1 float	10
3	Object name	1 string	"Arms"
4	Texture map name	1 float	"armsTex.bmp"
5	Ambient material	3 floats	0.5 0.6 0.5
6	Diffuse material	3 floats	0.5 0.6 0.5
7	Emissive material	3 floats	0.5 0.6 0.5
8	Specular material	3 floats	0.5 0.6 0.5
9	Glossiness	1 float	10
10	NV – number of vertices	1 float	127
11	<vert1> <norm1> <texc1>	3+3+2 floats	1.0 2.0 1.0 0.0 1.0 0.0 1.0 0.5
.	.	.	.
.	.	.	.
12+NV	<vertNV> <normNV> <texcNV>	3+3+2 floats	2.0 3.0 2.0 0.0 0.6 1.0 0.5 0.6
13+NV	NF – number of faces	1 float	333
14+NV	<face1 vertex indices>	3 floats	1.0 22.0 56.0
.	.	.	.
.	.	.	.
15+NV+NF	<faceNF vertex indices>	3 floats	10.0 20.0 51.0
16+NV+NF	Next object	1 string	"Legs"
.	.	.	.
.	.	.	.

Appendix D: Game Screenshots

Provided are several in game screenshots illustrating various components of the game.


Scorched Earth 3D

Player 1

Player 2

Lane Feltis, Neema Teymory, Andrew Wong, Ying Yin

Players: 2
Computers: 0
Terrain: Flat
Gravity: 9.81
Weapons: Regular
Wind: None

Figure 7 – Starting a New Game

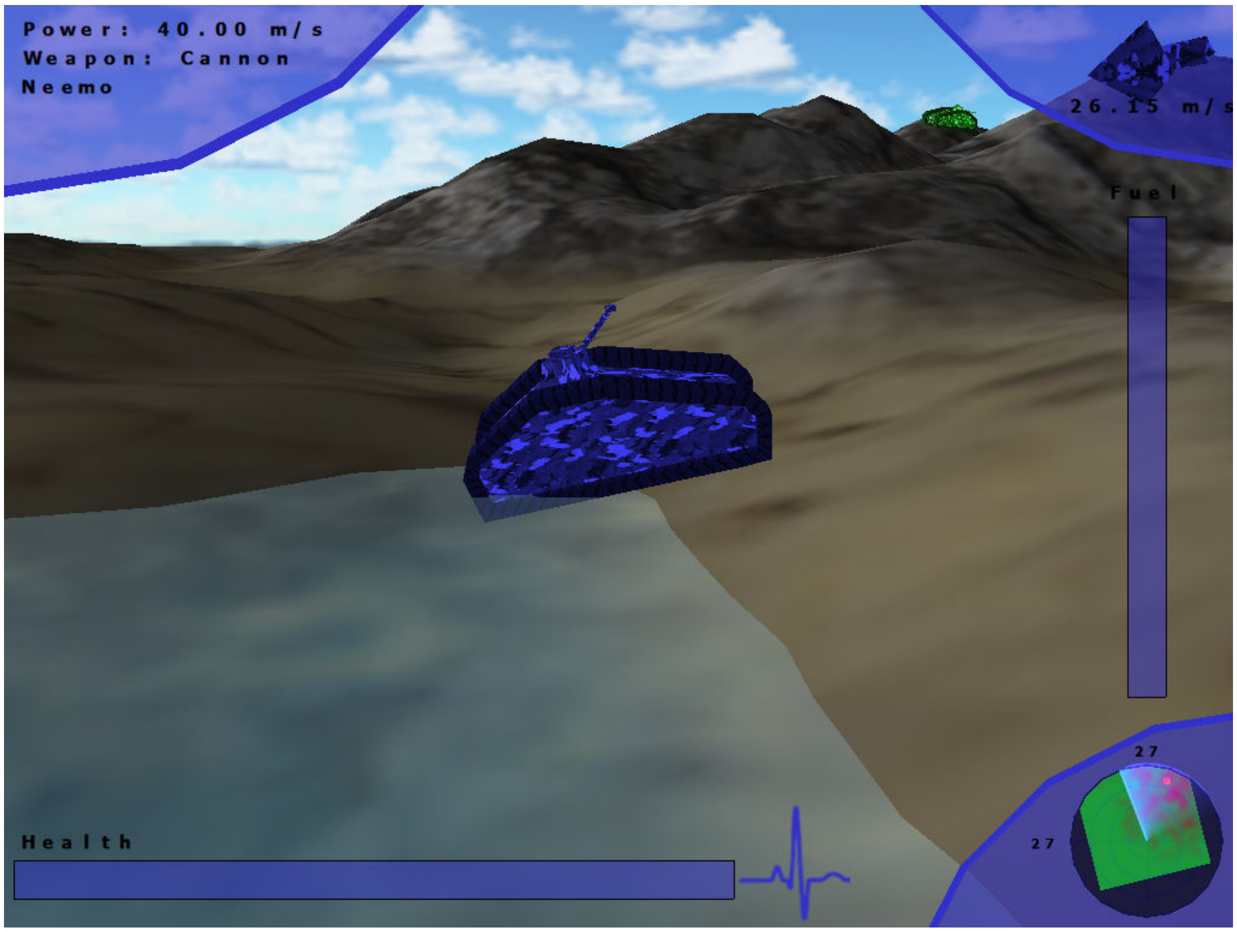


Figure 8 – A Battle for the Islands

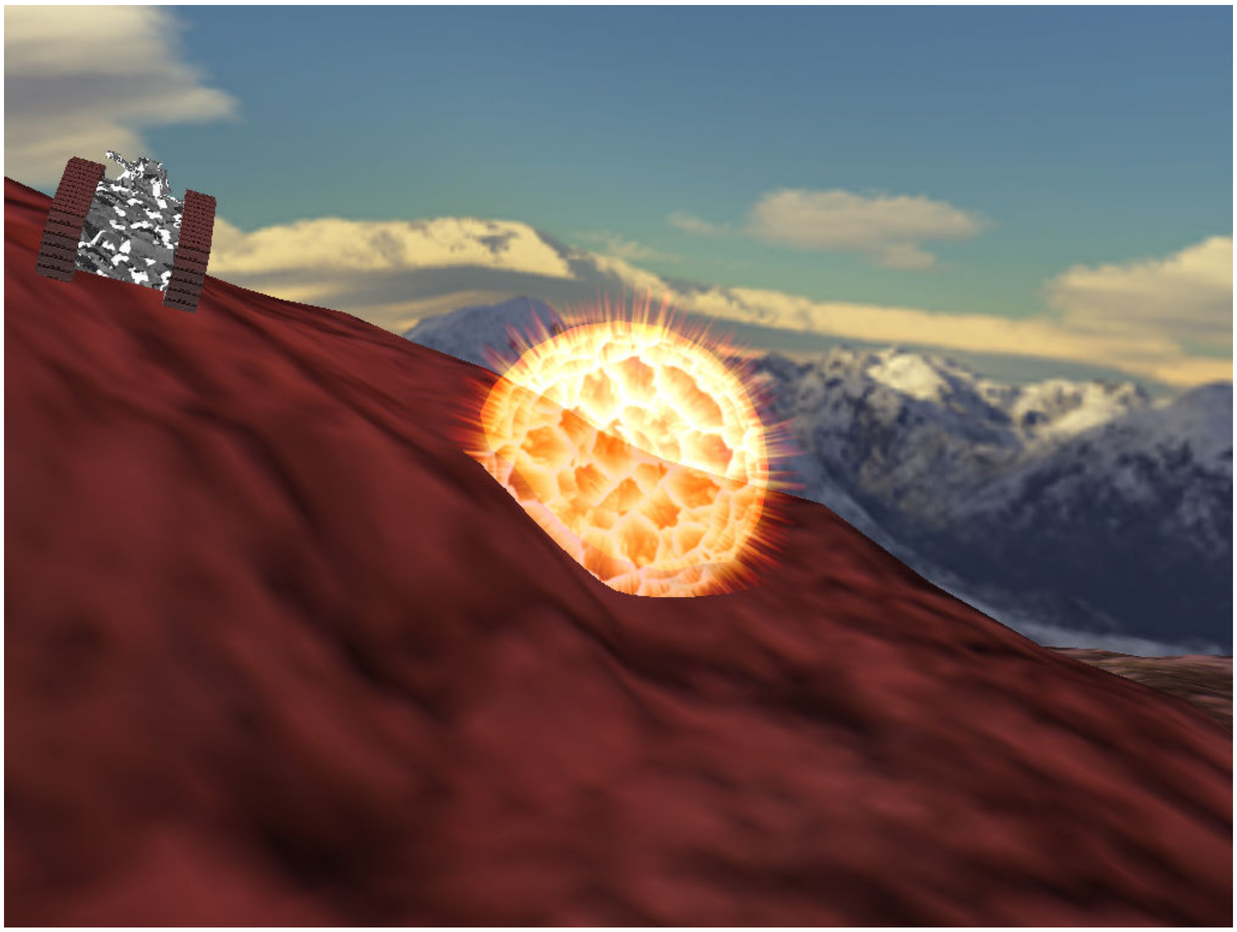


Figure 9 – Too Close for Comfort

References

[1] B. Nystrom. (2005). *Terrain Generation Tutorial*. Retrieved Jan, 2008, from Robot Frog: 3d. Web site: http://www.robot-frog.com/3d/hills/index.html

[2] P. Martz. (1997). *Generating Random Fractal Terrain*. Retrieved Jan, 2008, from Game Programmer. Web site: http://www.robot-frog.com/3d/hills/index.html

[3] T. Franke. (2001). *Terrain Texture Generation*. Retrieved Jan, 2008, from Flip Code. Web site: http://www.flipcode.com/archives/Terrain_Texture_Generation.shtml

[4] *FMOD Music & Sound Effects System*. (2008). Retrieved Mar, 2008. Firelight Technologies Pty, Ltd. Web site: http://www.fmod.org