

Battle City 3D
CS488 Project Winter 2012

Sheng Liu
Student ID: 20273200
WatIM ID: s43liu
Software Engineering, University of Waterloo

April 2, 2012

Final Project:

Purpose :

The purpose of this project is to create a 3D arcade tank game that mimic the core game mechanism of the classic NES(Nintendo Entertainment System) tank game *Battle City* while adding 3D features to the game using OpenGL.

Statement :

Published in 1985 by Namco, *Battle City* is a multi-directional shooter video game on NES(later ported to other platforms). The goal of the game is to fairly simple, the player, controlling a tank, must destroy all enemy tanks and protect the base from destroyed. This project would recreate the game of *Battle City* with 3D graphics and also adding new features to the game to increase playability.

In order to recreate the game in 3D, multiple 3D graphic technology would be used, such as 3D texture mapping, 3D modeling, 3D clipping, 3D collision detection, reflection and 3D lighting. Artificial Intelligence would be used to control the enemy tanks and make the game more challenging.

I believe this project is very interesting and challenging as it excercises many concepts and technics taught in class and in perticular, 3D collision detection. As there would be many tanks and bullets moving in this game, there would be many calculation to detect if the bullet has hit the obstacles or base, if the tank is allowed to drive in a given direction or if a tank is been destroyed by a bullet. As there are many moving objects in the game, the really challenging part is making the collision detection efficient so we can have enough CPU power to update the game frame by frame.

For a game with n objects, the n th object need to detect collision with the rest $n - 1$ objects, the $(n - 1)$ th object need to detect collision with the rest $n - 2$ objects and so on. Hence the total number of detection is in the $O(n!)$. During the game, we need to do detections on each frame, so it is crucial to implement 3D collision detection efficiently for the game to play smoothly and actually be playable.

By completing this project, I would have a through understanding of 3D texture mapping, 3D modeling, 3D clipping, 3D collision detection, reflection and 3D lighting using OpenGL.

Communication :

Input: Level map documents

Interaction: Mouse and Keyboard

Output: none

Modules :

I have used a modified version of scene lua loader from assignment 4 so that I can use the extended primitives while still been able to get the parsed scene node tree instead of running the script to generate a image file.

Technical Outline :

The Game

The Game Board

The board of the game is a square with a 26 by 26 grid. The top left corner of the board is labeled as $(0,0)$ where as the bottom right is labeled as $(25,25)$.

The Obstacle

An obstacle in the game is made of either brick or steel. Obstacle can come in 2x2, 2x1 or 1x2 size by default. When the brick obstacles are hit by a bullet, it is destroyed by 1 grid size. For example, if a 2x2 brick obstacle is hit by a bullet from north, the obstacle became a 1x2 obstacle instead. Hitting it again will completely destroy the obstacle. On the other hand, obstacle made by steel is indestructible.

The Base

The Base is an object located at the center bottom of the map occupying a 2 by 2 grid size. It only takes one hit to destroy the base. When the base is destroyed by either an enemy tank or by the player (friendly fire), the game is over. No obstacle can be placed on top of the base.

The Tank

The tank is a 2x2 grid size block that can be moved by AI or player on the game board. Tank can only move to block that is not occupied by other tanks or obstacles. Based on the type of the tank, the tank moves at different speed. Tanks can also shoot bullets with the same direction as the tank is facing. The speed at which each tank can shoot the bullets depends on the type of the tank. At the beginning of the game, player's tank appears next to the base facing north while enemy tanks appear at three spawn areas facing south.

The Spawn Area

The spawn area is a 2x2 grid size block at which the enemy tanks appear on the game board. There are three spawn areas on the game board: top left, top right and top center of the map. No obstacle can be placed on top of the spawn area.

The Land

The land on the game board determines how tanks move. When on default land, the tanks move according to the control of the AI or player; when on icy land, the tanks continue to move in the direction when the tanks enter the land and stop only when the tanks hit an obstacle or are no longer on the ice; when on the forest land, the tank is hidden in the trees and cannot be seen by other tanks, but can still shoot or get hit by bullets; tanks can not move onto sea land, but can shoot a bullet through the land.

Winning

There will be 20 enemies spawned in each level, when the player destroys all enemy tanks while the base is not destroyed, the player wins the level.

Losing

Each player has three lives to begin the game, when a player's tank gets hit, the player loses one life and respawns in the initial location. If the player loses all the lives or the base is destroyed, the game is over and the player loses.

Movement

When the tank moves, the movement must stop on the location where the tank aligns to a 2x2 grid on the board. Moving half way is forbidden. It is not possible to stop or change direction while the tank is not aligned to a 2x2 grid.

The Bullet

A bullet covers a 1x2 grid size on the game board and moves at a given speed. When the bullet collides with a tank, the base or a brick obstacle both the bullet and the object are destroyed. When the bullet collides with a steel obstacle, the bullet is destroyed.

Modelling

The game board can be represented by a 26 by 26 matrix with each value represent the obstacle and land information. In particular:

- 0: default land
- 1: brick
- 2: steel
- 3: sea
- 4: icy land
- 5: forest

The tanks can be modeled by using a Lua script. Part of assignment 4 Lua script interface can be reused to build the model. However, this may be very costly and require a lot CPU time to redraw all tanks on each frame. A minimal tank model is needed to ensure the game can run smoothly.

The View Mode

The game need to support three view mode.

Strategy Mode

This mode closely resemble the traditional game. The camera is placed a a fixed location on top of the game board looking downwards, the movement of the player tank does not change the camera angle.

Following Mode

This mode is sometimes called third person view in some shooting games, the camera is fixed at a location relative to the player tank. The camera is placed at the back of the tank above the game board in order to allow player see the full player tank.

Ground Mode

This mode is sometimes called first person view in some shooting games. The camera is placed inside the tank the player controles. The player should see the game as if he or she is seating inside the tank.

The Interface

The interface of the program have a menu bar and a viewing area. The menu bar will have an **Application** menu, a **Mode** menu and a **Level** menu. In addition, when the graphics display is resized the aspect ratio should be properly maintained – the rendering of the game might change size, but its proportions should not become distorted.

Application Menu

The **Application** menu should have the following items:

New Game – Start a new game from level 1. Keyboard shortcut **N**.

Reset Game – Start a new game from current level. Keyboard shortcut **R**.

Quit – Terminate the program. Keyboard shortcut **Q**.

Mode Menu

The **Mode** menu should have the following radiobutton selections:

Strategy Mode – Top bird view of the game. Keyboard shortcut **1**.

Following Mode – Third person view of the player tank. Keyboard shortcut **2**.

Ground Mode – First person view through player tank. Keyboard shortcut **3**.

Level Menu

The **Level** menu should have radiobutton selections for all the levels.

Control

The player can control the player tank move arrow keys \leftarrow , \rightarrow , \uparrow , \downarrow . When in Strategy Mode, if the key pressed is not in the same direction as the current direction of the tank, the tank rotate to the new direction, otherwise the tank move in that direction by 1 grid. When in following mode or ground mode, \leftarrow rotate the tank 90 degree to the left; \rightarrow rotate the tank 90 degree to the right; \uparrow move the tank foward by one grid.

To fire bullets in current direction, the space bar is used. Based on the type of the tank, the tank can only shoot another bullets after a given time.

Organization :

All files will be found in the subdirectories under the A5 directory. The subdirectories and files are indicated in the following.

src contains all the code that are needed to compile the application.

levels contains all the maps for different game levels.

textures contains all the image files used by the program.

Documentation :

The file README gives a brief installation guide. This document is a report for this project. The file proposal.pdf is the project proposal.

Implementation :

Data Structures

To represent the game board in my final project, I used a level object to keep track of the game board information. I used a two dimensional integer array to represent the land information. This information can be later used to simplify the hit detection process significantly. I also created a two dimensional pointer array to keep track of the corresponding scene nodes created for each tile. This allow me to easily modify the game board when brick tile is been hit.

Image Loading

I implemented an image loading class utilizing the existing image utility from assignment 4. The actual implementation is in texture.hpp and texture.cpp. The overall steps are follows:

bf Step 1 Reading the image files using the image utility from assignment 4

bf Step 2 Convert the image data from float point representation ranging from 0.0 to 1.0 into integer representation ranging from 0 to 255

bf Step 3 Create a bind slot for each image in order to OpenGL to load the image into memory for later use.

bf Step 4 Actually loading the image data into the corresponding slot and return the corresponding bind handle.

Texture Mapping

Based on the previous implementation, now I can map images onto shapes, and cube in particular. After researching on the OpenGL built-in function of creating a cube, glutSolidCube(), I found out that sadly it does not support texture mapping. So instead I wrote my own cube implementation. The way of creating a texture mapped cube is to use six quad shapes to form a cube and map texture onto these quad shapes individually. The actual implementation is in primitive.cpp and primitive.hpp.

Manual :

The program was compiled and runs on machines inside graphics lab. The main executable is called battle_city. It must be executed in the same directory as the textures and levels folders are located. I assume that valid level files for first two levels are present in the levels folder and all the necessary image files for texture mapping is present in the textures folder.

The user can use left mouse to change the view point around. The menu is also used to select different levels and different game mode. The arrow keys are used to control the tank.

The user can control the player tank move arrow keys $\leftarrow, \rightarrow, \uparrow, \downarrow$. When in Strategy Mode, if the key pressed is not in the same direction as the current direction of the tank, the tank rotate to the new direction, otherwise the tank move in that direction by 1 grid. When in following mode or ground

mode, \leftarrow rotate the tank 90 degree to the left; \rightarrow rotate the tank 90 degree to the right; \uparrow move the tank forward by one grid.

The user can use the menu to select different levels to play. The user can also use the menu to switch between different view modes as Strategy Mode (Top bird view of the game), Following Mode (Third person view of the player tank) or Ground Mode (First person view through player tank).

To fire bullets in current direction, the space bar is used. Based on the type of the tank, the tank can only shoot another bullets after a given time.

Sources :

In src. Executing command make under src directory will compile my project and produce an executable named battle_city.

Executable :

My executable is named battle_city.

Data Files :

Textures are located under textures folder and level maps are located under levels folder.

Bibliography :

Hearn, Donard, Baker, M.Pauline, *Computer Graphics* Prentice Hall(1994)

Martin John Baker, *3D Theory - Collision Detection*, "3D World Simulation", Web(2011)

Ericson, Christer, *Real-time Collision Detection* Elsevier(2005)

Lin, Ming C, *Efficient Collision Detection for Animation and Robotics* (thesis). University of California, Berkeley.(1993)

Abdallah Rababah, CS 488/688 Course Notes, University of Waterloo(2012)

Objectives:

Full UserID:_____ Student ID:_____

- **1:** Tank model is complete and displays correct. The direction in which a tank is facing can be easily identified.
- **2:** Board map is parsed correctly and displayed correctly with strategy mode.
- **3:** Texture mapping for obstacle, base, land and tank is complete.
- **4:** Implement fake surroundings around the game board
- **5:** Tanks cannot move into obstacles or each other. Bullet is implemented. Collision detection is needed correctly to detect hit against other tanks, base and obstacles.
- **6:** Space partition is implemented to improve performance of hit tests.
- **7:** Reflection on icy land and sea land is implemented.
- **8:** Artificial Intelligence is implemented for enemy tanks to allow simple movements and firing.
- **9:** Game mechanic is implemented.
- **10:** Skybox is used to give a distant background.

Declartion:

I have read the statements regarding cheating in the CS488/688 course handouts. I affirm with my signature that I have worked out my own solution to this assignment, and the code I am handing in is my own.

Signature: