Developing a Whale Shooter: Terminator Land

Results of a practical course at the Chair for Computer Graphics and Multimedia
(RWTH Aachen University, Germany)

Arun Ravi* Ercan Yalvac† Hongrui Deng‡ Sergei Krestianskov§ Weiyue Wang¶

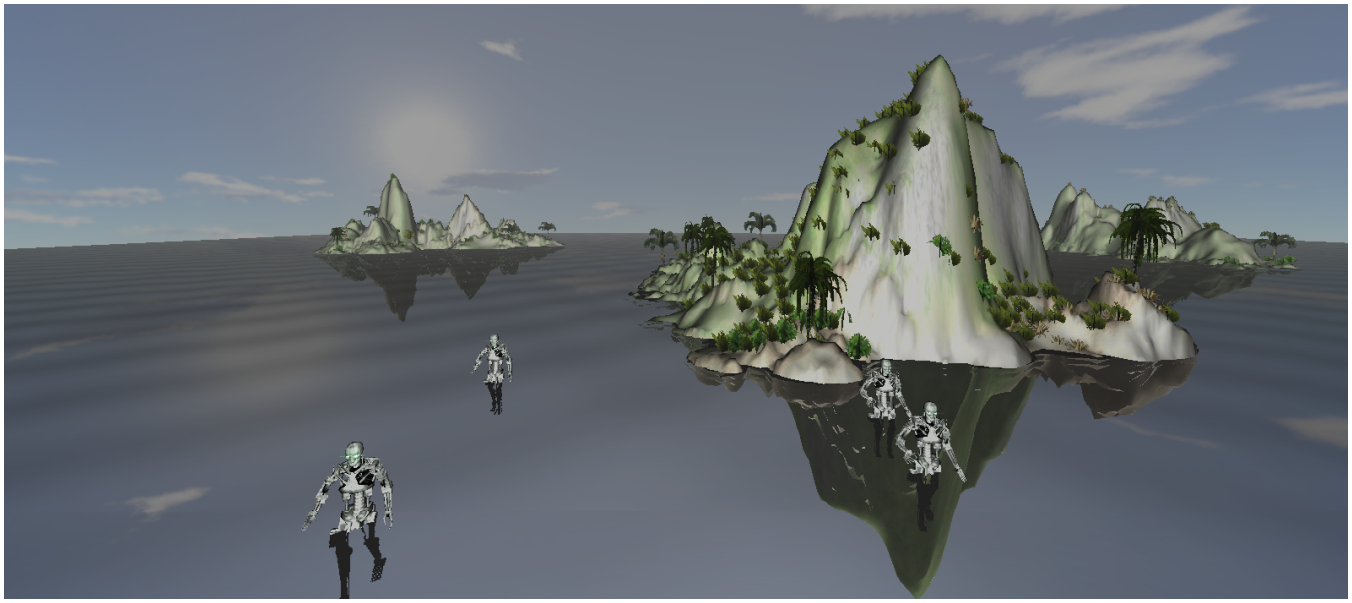

Figure 1: *A screenshot of Terminator Land.*

Abstract

This paper briefly describes the development of a whale/first-person shooter called *Terminator Land* (see Figure 1). The idea of the game was basically to create something close to a First Person Shooter that would include the fun of shooting, the challenge of aggressive enemies and the excitement of controller free gameplay. This report very briefly describes the development process as well as the graphical methodologies implemented in the game.

Keywords: game programming, whale shooter

1 Introduction

We began with the idea that our character would be stranded on an island filled with undead creatures and the only way out would be to shoot through them. The first choice for the game type was a first person shooter to offer the player full control over his/her characters movements. After some discussion however, we decided that a whale shooter would be appropriate form for the game. This would let us restrict the amount of terrain to be generated. Additionally, we were also keen on using the *Kinect* from Microsoft as a controller for our game. Our initial categorization of the primary work packages for the game was as follows:

- *Content creation*
- *Physical interactions*
- *Graphical effects*
- *Interface design*
- *Game control*

Subtasks were then created in these sections and divided among our team for development. A brief descriptions of the subtasks completed under each category is the sections ahead.

2 Content Creation

- *Terrain generation* [4] The terrain in our game is represented as a set of several islands and a plain bottom. To create a mesh, Blender together with a height-map picture and a high-resolution texture were used. The heightmap and the texture were generated manually with GIMP.
- *Water Generation* [4] In the beginning, water mesh was created in Blender and waves were made using sin-function in a vertex shader. Later on, a mesh of water was manually created to allow for more realistic graphical (reflection and refraction) and physical (waves) effects.
- *Skydome* [2] The idea was to basically create a giant sphere along with a suitable sky texture and place it in the environment so that it appears to cover the open area above the ground. The sphere was created in Blender and the normals inverted so that it faces inward towards the ground.
- *Grass* [5] Each basic unit off grass was implemented as a bush consisting of six billboards. Two billboards are combined to

*arun.ravi@rwth-aachen.de
†ercan.yalvac@rwth-aachen.de
‡hongrui.deng@rwth-aachen.de
§sergei.krestianskov@rwth-aachen.de
¶weiyue.wang@rwth-aachen.de

create a double face quad and three such pairs are arranged at relative angles of 120 degrees.

- *Trees* [3] The tree model was created by using billboard techniques. Each tree contains 6 slices of billboard, each of which was created by using Blender. Moreover, the textures are randomly attached to each tree, so that the whole layout of trees will be more natural, and appear differently with new launch of the game. Complete 3D-models of trees were also used, but too much geometry significantly slowed down the game speed.

- *Enemies* [1] The basic model of the enemy is a Terminator T-800 mesh which was obtained from the internet along with a texture. The Artificial Intelligence for the enemy was programmed only so that the enemy would simply walk towards the player and periodically fire a Shearing Death Ray from its eyes.

- *The Minigun* [1] The model was hand-crafted from scratch so as to allow the barrels to rotate to give the player a realistic feel of holding a minigun. The weapon has basically been textured with uniform colors giving it a very diffuse feel.

3 Physical Interaction

- *Collision detection for ammunition* [1] The idea here was to do a sphere-cylinder collision detection. The enemy was approximated by a cylinder and the ammunition by a sphere. This was used to trigger the explosion effect.

- *Waves on water* [4] The water mesh was created manually and it was possible to manipulate vertex positions, so that the height field fluids equation could be applied. Waves are generated by regularly applying some force at some positions on the water grid.

- *Movement of grass* [5] In order to do a realistic animation, the trigonometric functions (sine and cosine) were utilized. For this calculation we also need to consider the position to be changed and the current time. In this project only the upper parts of the grass objects were waved. coordinates.

4 Graphical Effects

- *Real-time glow* [1] The idea was to impart glow to only certain objects in the world. Ultimately, this has been limited to ammunition pellets shot from the players weapon, the enemys eyes and the enemys attacks. The idea was to just apply a gaussian blur to glow sources and append that to the original image [James and O'Rorke 2007].

- *Muzzle flash* [1] The idea here was just to give a more realistic feel of firing ammunition from the weapon. This was implemented using a simple quad billboard and dynamic switching between textures on the billboard to give the effect of firing (see Figure 2).

- *Explosion* [4] This effect was achieved with a billboard created in a geometry shader along with multiple textures to give the effect of an animated explosion (see Figure 2).

- *Atmospheric Fog* [2] This effect was achieved by simply blending pixel colors with a gray color value depending on the depth value at that pixel. Additionally, the fog was programmed to vary exponentially with changing depth to give a more realistic effect.

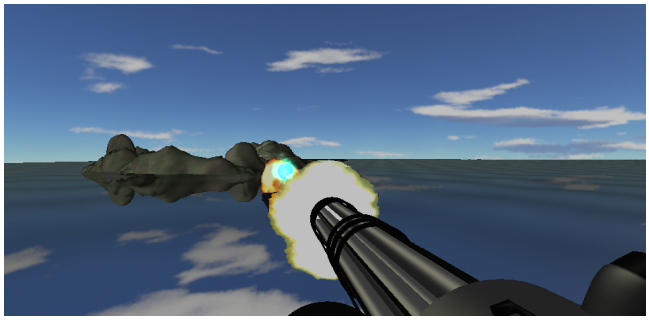

Figure 2: *Example of muzzle flash and explosion.*

- *Rain* [4] This effect was implemented using a geometry shader that created minute camera aligned triangles based on a grid of points externally created. The motion of rain was controlled by velocity dependent displacement of points in the vertex shader. Additionally the points are randomly distributed and located only within the field of view.

- *Reflections on water* [4] Deferred shading was used to implement this effect. First, the terrain mesh was reflected against water plane and rendered to a separate FBO. Then, when rendering water, the texture of that FBO was used to look up texels. We were able to produce perturbation on the water surface by using normals from the water mesh to shift texel coordinates. The normal map of the water mesh is dynamically updated every frame [Sousa 2005].

5 Interface Design

- *Heads up display* [1] This consists of a simple green health bar at the bottom left of the screen and a kinect indicator icon on top of it. The kinect indicator icon was kept just to inform the player of his/her tracking status when playing with the kinect.

6 Game Control

- *Natural interaction* [1, 2] When developing the game, we were very keen on bringing the element of the natural interaction into it. We have done this by integrating the Kinect with our game using the OpenNI and NITE packages. With this we present the player with a completely controller free approach towards gaming. We obtained basic code from the NITE examples and the OpenFlipper framework and added game-specific functionality on top of it.

- *Character Movement* [3] Since this is a whale shooter game, and we didnt set any storyline, so the path to move is initialized as a large circle around a mountain. However, player can also change the moving model from fixed path to freely moving by pressing certain button, vice versa.

References

JAMES, G., AND O'RORKE, J., 2007. Real-time glow.

SOUSA, T., 2005. Generic refraction simulation.

[1] Done by Arun
[2] Done by Ercan
[3] Done by Hongrui
[4] Done by Sergei
[5] Done by Weiyue