GDXWaker

Jonathan Hagberg* Anton Risberg Alakla†

Lund University
Sweden

Abstract

In this project we achieved a graphics design similar to that of The Legend of Zelda: Wind Waker. The feeling was achieved with a toon shader on the character and a bloom effect that is heavily used for the updated version of the game.

1 Introduction

The idea of toon shading was interesting to us and Wind Waker is a great example of a AAA game that relies on toon shading. The games aesthetics was heavily criticized when it was released with people complaining that the game was designed for children, but now it is viewed as one of the best looking and most enjoyable game in the Zelda franchise.

For these reasons, as well as sentimentality, we decided to replicate the look of that game as much as possible.

2 Algorithms

The first thing we did was go out looking for 3D models from the game. We found .obj-files containing both the first island in the game, the sea and Link himself. We tried loading them into RenderChimp but ran in to many different issues. After a bit of struggling we decided to switch to a different engine; LibGDX. Its a open source Java based game engine. The 3D part of LibGDX is quite new and still under heavy development, but it worked for our needs.

We used the Link-model with the online tool Mixamo to get a fully working skeleton and animations. Mixamo produces a .fbx file which we could import to 3ds Max and apply the textures that was needed. The .fbx file was then converted to a LibGDX-friendly format using a tool called fbx-conv.

We used Bullet for our physics simulations. The island physics model was achieved by copying the triangles from the .obj file to a triangle mesh shape, meaning no approximations. If you see a triangle on screen, you can collide with it. This should enable very accurate physics simulations.

The physics model for the player was approximated with a small ball with its bottom edge at the players feet. The rotation of the ball is ignored to avoid having the player spin around. We then apply forces to the ball to move around.

The actual toon shading was by far the easiest part of the project. It was gotten for free when applying shadows to Link.

The bloom effect was added by using post-processing of the pixels and LibGDX have an existing filter for bloom.

We added deferred shading in the same style as Assignment 2 (Sponza). The scene only has one light source; the sun, and the shadows should have the same angle wherever you are, so we had to do some changes. Firstly; the suns depth buffer is rendered with an Orthographic camera, since we want to simulate the sun being really far away (straight, parallel sun rays). Secondly, the suns position is set to a constant plus the players position each frame. That way the sun is always the same (relatively short) distance from the player and we dont need to have an enormous resolution on the frame buffer in order to properly outline the player. By removing the concept of angular and distance falloff from assignment 2 (Sponza) we got a toon-shaded look. Either an object is fully lit or it is fully shaded.

The rendering is done in five stages:

- Render a depth buffer from the point of view of the sun
- Render a depth buffer from the point of view of the players camera
- Render the standard view
- Render shadows on top of the standard view
- Add bloom effect

And finally the texture is rendered to the screen. When we first did this the shadows were projected onto the ocean surface, instead of the ground. We never managed to figure out why, but we worked around the problem by moving the ocean to the players feet in stage 1 and 2. Its not pretty (code wise), but cheating is what computer graphics is all about so its ok!

Before we had shadows we used the following method to get toon shading. The toon shading was achieved by taking the dot product between the light vector and the normal of the character object. If the dot product is lower than a constant (in our case 0.3), the color of the pixel only get half of the contribution from the diffuse texture. In the game the edge between dark and light isnt sharp. Therefore we smooth out the edge between 0.3 and 0.375 which gives a less sharp and a more gradient edge.

3 Results

The finished product runs surprisingly well (performance wise). We thought that modelling each triangle in the physics model might eat up the CPU but it works. We were also concerned that LibGDX might not be the best choice for a 3D library since it is written in Java, but all the computationally heavy tasks are executed in native libraries so it turned out ok.

4 Discussion

To achieve a style similar to Wind Waker a lot of work on the 3d-models had to be done. This took valuable time that otherwise could be used for improving the shaders.

Since the wind waker assets arent officially public, there were some issues rendering them. Mainly; the ocean. The ocean is animated in Wind Waker. There are multiple textures that are blended together and moved back and forth on the beaches. Nintendo probably has a file somewhere describing the properties of the beaches (location, angle etc), but we didnt have that. By rearranging the triangles in the obj file we managed to get a somewhat decently looking water edge, but sadly no animation.

We noticed that triangle meshes in Bullet physics dont always work well. When we built a triangle mesh from Link it didnt collide with the island mesh. It still collided with test shapes, e.g. boxes. That was the reason we used a simple sphere-collider on Link.

*e-mail: ada09jha@student.lth.se
†e-mail: ada09ari@student.lth.se

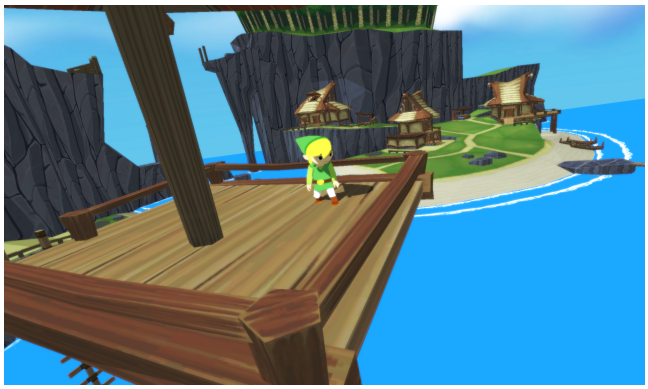

Figure 1: Overview of the starting island.

Figure 2: Top: our application. Bottom: GameCube version of Wind Waker (note: bloom is only present in the Wii U version).

There was also the issue with Links eyes. One of the features of the game is that Links eyes are animated and they always look at something interesting. Either a monster, a door or the way you need to go to progress further in the game. The mouth was also animated for when he talked, shouted etc. This could be seen in the obj file as link had multiple meshes on top of each other on the eyes and mouth, which werent textured properly so we couldnt use them. Nintendo probably had some logic for showing/hiding & texturing the different meshes dynamically, but we dont. We had to remove all but one of the meshes and then texture them ourselves. It now looks quite near to the game except for the animations.