LINKÖPINGS UNIVERSITET

Procedural game made in WebGL
TNM084 - Procedural methods for pictures

Anders Nord - andno922@student.liu.se
August 19, 2013

1 INTRODUCTION

This report will discuss using a procedural simplex noise function [3] in WebGL and implement it in a game.

Simplex noise is a version comparable to perlin noise and was designed to be less computational than the original version of noise. A great advantage with procedural functions is that they are calculated mathematically and appear very random. For a game, this can come in handy in many different ways.

1.1 GAME IDEA

The idea for the game is pretty simple. The z-axis of the ground is depending on the value obtained from the noise function. This creates a wave-looking surface. When the surface is below a certain value, it changes colour to blue and gives damage to the player. The player is supposed to watch out for the blue "water" as seen in fig 1.1. The other objective is to get as far north as possible, before running out of life.

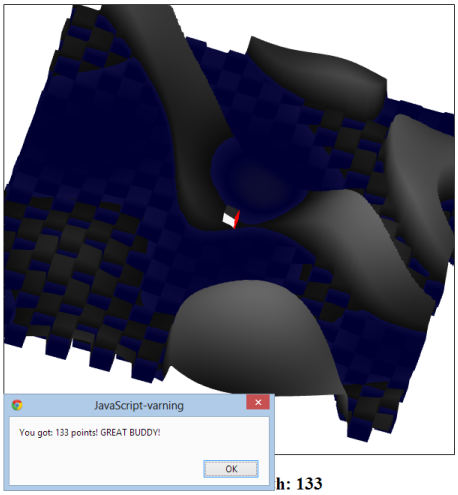

Figure 1.1: The player gets hit by the water and looses the game

So the game was implemented in WebGL which means shaders. WebGL communicates with the graphics-card through OpenGL ES 2.0-standard shaders. To access these in html, javascript is being utilized. The matrix library used was glMatrix v0.9.5 [4].

The surface is created from a number of calls to the noise-function. All with different inputs. As seen in [1, p. 4 & 7] there are a few regular procedural patterns. The procedural pattern in this game is a mix of a couple of these. The squares that can be seen in the water, as in fig 1.1, 2.1 and 2.2, is basically only for visual reasons. This is achieved by only adding the extra noise-function-calls after it is known that the first noise-value is below a certain value, or in this case, sea-level.

The noise-function most commonly uses the texture-coordinates and some form of variable affected by time as input. This creates different values all over the canvas. To address the anti-aliasing that can appear, the built in smoothstep-function is being used. This is basically an interpolation method, see [1, p. 5] or section 1.3 in [1] for a more detailed explanation.

Two different shaders was created. One for the player and one for the surface. The player-shader takes care of rendering and moving the player(cube) accordingly to how the surface is moving in the z-axis. The surface shader renders the surface according to a few different calls to the noise-function, as mentioned earlier.

To be able to account for a hit, the scene is first rendered to a texture, by using a FBO(Frame Buffer Object), from above. See fig 2.1. The textures centre, with a kernel according to the size of the cube(player), is then being checked for the colour red. If any pixel in this area is red, it means that the player is below the surface, and is supposed to loose health. See fig 2.1

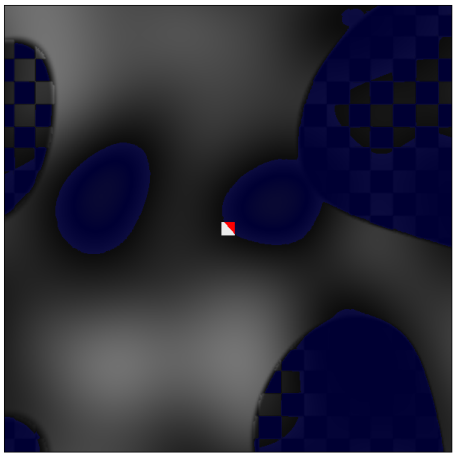

Figure 2.1: The texture rendered from the FBO. Scene seen from above.

The scene is then rendered again after unbinding the FBO. This time the view is being rotated so a more 3D-like-view is shown to the player. See fig 2.2.

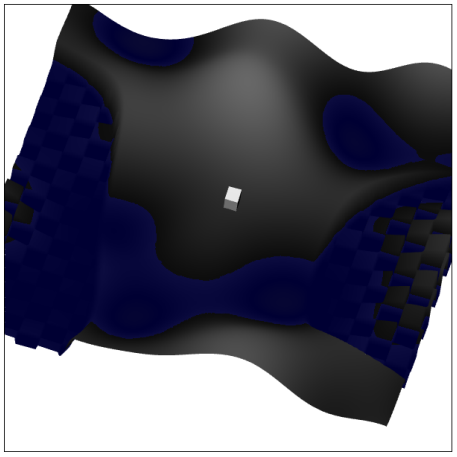

Figure 2.2: The game-view

The shading of the surface is only depending on its value i the z-axis. The player on the

other hand, is being shaded depending on a light-source. The light is located above the cubes starting position. The colour value of the cube is then calculated based on the dot product between its normals and the direction of the lightsource. This is done for every pixel, in the fragment shader and is called pixelshading. See fig 2.3

Figure 2.3: The player shaded by pixelshading

2.1 Computer specs

The project were run on this machine:

Processor: Intel Core i7 3610QM 2,3 GHz
Memory : 8 GB of DDR3 1600 MHz SDRAM
Graphics card: Intel(R) HD Graphics 4000 that has a graphics memory of 1792 MB
OS: Windows 8 64-bit

3 Discussion

It has been a fun experience working with a procedural pattern. It creates new possibilities and forces you to think outside the box. They can be very helpful in the sense that textures might not be needed necessary. This could solve problems that exist in games when using textures. But it does take some computational power so there will always be a decision to be made.

Something that would have been pleasant to add to the game is blurring the picture when loosing health. The more you loose, the blurrier it gets. This might be added in future versions.

References

[1] Stefan Gustavson, `http://webstaff.itn.liu.se/~stegu/TNM084-2012/proceduraltextures.pdf`.

[2] Stefan Gustavson, `http://webstaff.itn.liu.se/~stegu/TNM084-2012/webgl/shadertutorial.html`.

[3] Ashima Arts & Stefan Gustavson, `https://github.com/ashima/webgl-noise`.

[4] Matrix library, `https://code.google.com/p/glmatrix/wiki/Usage`.