Procedural Generation Project Report

Zihang Jiao
School of Design and Informatics
Abertay University
DUNDEE, DD1 1HG, UK

1. Introduction

In this project, I implemented an application that creates and renders a 3D scene with examples of procedural content generation techniques. There are three main aspects of the project:
- Procedural generation methods: Cellular Automata
- Mouse click detection: Ray picking
- Post-processing effect: Reflection effect

In this report, I aim to provide the reader with an understanding and justification for making decisions. This report should also be a starting point for further development and reflect on the achievements and difficulties throughout the project's development.

2. Control

A. Movement

The user can press "W" and "S" to move forwards and backward; Press "A" and "D" to rotate the camera to the left and right; Press "Z" and "X" to rotate the camera upwards and downwards. Press "Left Shift" to raise the camera upwards and press "Left Ctrl' to low the camera downwards.

B. Cellular Automata

The user can press "R" to initialize the alive status of the cells; Press "C" to execute the island generation algorithm. The corresponding parameters of island generation can be adjusted in the imGui table; Press "V" to execute Conway's Game of Life;

C. Environment Control

The user can click the meteorite to make it start rotating/stop rotating. The user can change the normalized RGB value in the imGui table to change the water color in the scene.

3. Application Design

The project is based on the lab code and Rastertek framework.

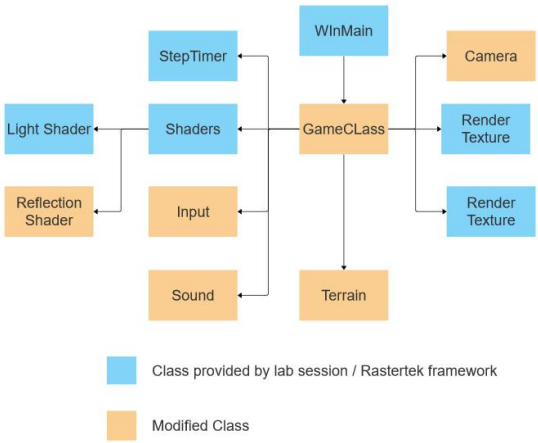

Figure 1: Class structure

Function of Class and Modifications

Class name	Function of Class	Modified
WinMain	Generate window, switch window size	None
GameClass	Controls the main logic of the program. Call function from other classes to read input, load resources, and render the scene.	Add functions about ray picking (mouse clicking detection). Set reflection stuff in the render function
StepTimer	Count for time	None
Light Shader	Set light effect	None
Reflection Shader	Set reflection effect	Build the shader, add additional reflection buffers, and load additional textures to the pixel shader
Input	Read user input and pass to game class	Define user input about cellular automata, mouse-clicking, and camera

		rotation on the x-axis.
Camera	Calculate camera view matrix	Add function to calculate the reflection camera view matrix.
Render Texture	Creating textures	None
Model Class	Read and Render obj models	Change the code to allow reading data that contains two triangle faces in a line
Device Resources	Allocate and destroy resources, and communicate with the hardware.	None
Sound	Read and Load Audio data, output the sound information to corresponding playback device	Create Primary and Secondary buffers. Load sound information into the buffers, output the sound.
Terrain	Initialize, generate and render the Terrain	Adding functions about Cellular Automata

4. Procedural Generation

4.1 Structure of Terrain

A struct called HeightMap represents the terrain, which contains the x, y, and z position, textures, and normals of specific coordinates. An array of HeightMap can be used to represent the whole terrain.

4.2 Cellular Automaton

A cellular automaton is a collection of cells on a grid of specified shape, the state of the cell is either alive or dead. The state of each cell evolves through several discrete time steps according to a set of rules based on the states of neighboring cells. The whole process can be divided into three parts:
- Initializing the alive state of every cell
- Calculate the number of nearest neighbors
- Change the alive status of each cell based on the number of alive neighbors

For alive status initialization, a function was used to loop through every cell and generate a random number from 0 to 100 for each cell. The user inputs an AliveProb parameter in the imGui window.

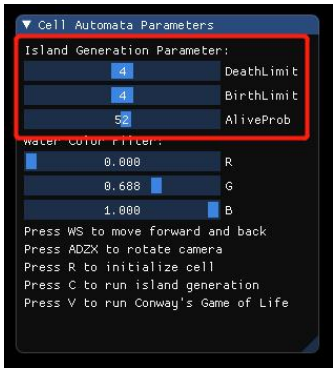

Figure 2 : Island generation parameters

If the randomly generated number is less than the user input AliveProb, the cell is set to be alive. Otherwise will be set to dead. The alive status of cells will be stored in an array called AliveMap, which has the same size as HeightMap.

After the active status is initialized, we need to count the number of neighbors in each cell. There is a maximum of 8 neighbors in total. The method will go through each of them and check how many of them are alive. In this project, we assume the border of the HeightMap consists of live cells. In this case, the cell at the border or corner is considered to be surrounded by more alive cells.

The alive status of each cell is updated every time step based on the number of their neighbors. We used two methods in the project: the island generation and Conway's game of life.

The rules of island generation:
1. If a cell is alive and has less than DeathLimit(4) alive neighbors, it becomes dead.
2. If a cell is alive and has more than DeathLimit(4) alive neighbors, it remains alive.
3. If a cell is dead and has more than BirthLimit(4) alive neighbors, it dies.
4. If a cell is dead and has less than BirthLimit(4) alive neighbors, it remains dead.

Where the DeathLimit and BirthLimit are numbers set by the user in the imGui window, they have a default value of 4, which can show a relatively good performance in dividing the map into several parts. In our project, the alive cell will be represented by the texture of water, and the texture of the ice will represent the dead cell.

In this case, the larger BirthLimit and larger DeathLimit show more ice area.

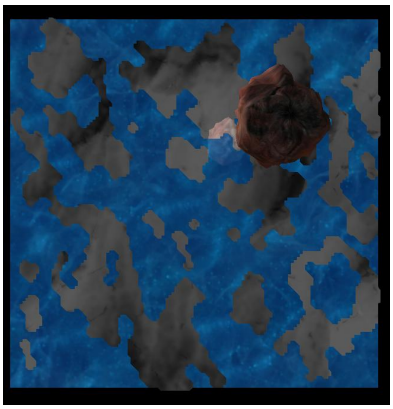

Figure 3 :Example of island generation with BirthLimit = 4 and DeathLimit = 4.

Conway's Game of Life is a cellular automaton rule devised by British mathematician John Conway in 1970. After giving the input to the cells, the player can observe how the cells evolve.

The rules of Conway's game of life:
1. If a cell is alive and has less than 2 alive neighbors, it becomes dead, as if by underpopulation.
2. If a cell is alive and has 2 or 3 alive neighbors, it remains alive, as if by the next generation.
3. If a cell is dead and has more than 3 alive neighbors, it becomes dead, as if by overpopulation.
4. If a cell is dead and has 3 alive neighbors, it remains dead, as if by reproduction.

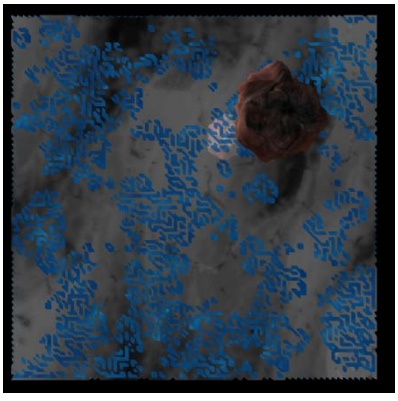

Figure 4: Example of Conway's Life Game

To avoid messing up the neighbor count of other cells, we created a temporary array to store the renewed alive state instead of directly changing the cell's state while it was still looping. Then assign the value of the aliveMap by this temporary array after finishing looping.

5. Ray Picking

In this project, I loaded a meteorite model into the scene and achieved the effect that the meteorite will start rotating/become stationary whenever you click the model. A technique called ray picking is used.
Ray picking is a method that checks ray intersection with an obstacle.
- Function of the ray: Origin + Direction
Where the origin point of the ray is the camera's position, and the direction of the ray is (User clicking point - Origin point).

Mouse. x and Mouse. y functions are used to get the user to click point in the two-dimensional window. The problem is that the point on the window that the user clicks on is two-dimensional, but the scene is three-dimensional. Thus a method is needed to transform our selection point back to three dimensional.

A multiplication of sequences: World Matric * View Matrix * Projection Matrix can be used to map a point from two-dimensional space to three-dimensional space. On the other hand, if we want to transfer the point back to three-dimensional space, we can multiply the point by the inverse of each of the matrices in sequence.

Clicking Point in three dimensional space = Clicking Point in two dimensional space * Inverse(Projection Matrix) * Inverse(View Matrix) * Inverse(World Matrix)

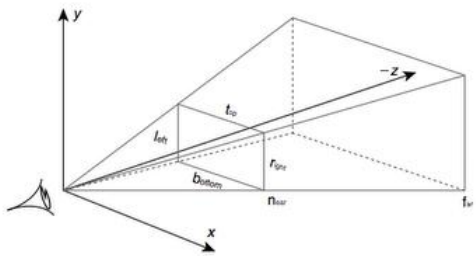

Figure 5 : The rectangle in the middle represents the window that the user clicks on, we can build a ray based on the eye's position (camera) and the point clicked in three-dimensional space.

After calculating the point in three-dimensional space, we can use the point and original point to calculate the ray function. To test for the intersection between the ray with the meteorite model, I used the function BoundingSphere() to create a bonding sphere at the position of the meteorite model. After that, BondingSphere.Intersects(origin, direction, distance) function can be used to check for the intersection of the ray and the sphere.

Every time the user clicks the stationary meteorite, it will start rotation by the local x-axis. The rotation is created by CreateRotationX(total_seconds), where total_seconds are accumulated by the timer.GetElapsedSeconds() every frame. If the rotating meteorite model is clicked, the total_second will stop increasing, and the rotation will thus stop.

6. Render To Texture Reflection

I implemented the reflection effect based on the render to texture technique.
In real life, we see the reflection image as the light reflected by the object's surface and passed to the human eyes(camera). If we want to make a reflection effect, we can imagine another camera on the other side of the reflection surface; in the rest of the paragraph, I will call the imaginary camera a mirror camera. The line connecting the two cameras is perpendicular to the reflection surface. If we want to plot the picture of reflection, we will find the position of the mirror camera and plot the graph from its point of view. Then we can use render to texture to render this graph at the ground surface to show the effect.

We need to create a reflection view matrix for the mirror camera compared to the ordinary view matrix. The y-coordinate of the two cameras needs to be symmetric by the reflection surface. So the position of the mirror camera is the position of the camera.y + (reflection surface .y * 2.0). The camera's orientation on the y axis also needs to be the negation of the camera's orientation (As our camera can rotate on the x-axis). After calculating the position and orientation of the mirror camera, we use the CreateLookAt() function to create the reflection view matrix base on the position and orientation of the mirror camera.

To pass the reflection corresponding information to the vertex shader, I created a new buffer in the ReflectionShaderClass. The buffer contains two pieces of information. The first is the reflection view matrix, which we previously calculated. The second one is a normalized four-dimensional vector that includes the user input color. We will use this input color to change the color of the water.

In the vertex shader, the reflected object's position needs to be calculated:
Reflect Position = input.position* worldMatrix* reflectViewM atrix * projectionMatrix
After the reflection positions are calculated, they will be passed to the pixel shader.

In the pixel shader, we first need to calculate the texture coordinate of the reflected model. We divide the x and y coordinates by w to get the tu and tv coordinates in the range -1 to 1. To map the range from 0 to 1, we need to divide the coordinate by 2, plus 0.5.After calculating texture coordinates, we need to assign corresponding texture colors. There are three kinds of textures taken into the pixel shader in this project, the texture of meteorite surface, the texture of ice, and the texture of water. The ice and water are original textures that represent the texture of the ground. The meteorite surface texture represents the texture of the thing being reflected. To make the reflected color effect, we use the lerp() function, a linear interpolation function, on the color of the original texture and reflected texture to create a blending effect.

In previous cell automation, we will set the y coordinate of the alive cell to -2.0 and set it to 0 if it is dead. We want to use the ice texture to represent those dead cells and the water texture to represent the alive ones. Thus, we pass the y coordinate of the input vertex position to the reflection pixel shader and set a boundary of -1.0. Every vertex with a height higher than -1.0 will be set to the color of ice, and the lower ones will be set to the color of the water. We also want the color of the water to change by the user's input color. Thus, if the height is less than -1.0, we will call the lerp() function again to produce a further mixture of colors.

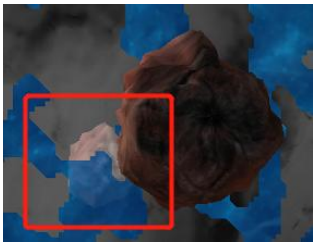

Figure 6 :Reflection in ice and water

We will create a RenterTexture object and draw the meteorite in the reflection image at this step. To make the reflection effect stable in both windowed mode and fullscreen mode, we need to create a new render to texture object with the current window size every time the size of the window changes. After that, we will set this RenderTextre as the texture of the ground. Then the floor will be able to show the corresponding reflections.

7. Conclusion

In this project, I built cellular automata algorithm and made a corresponding visualization effect. During the project's implementation, I had the opportunity to figure out the nature of the graphics problem and delve into the most optimized way to solve them. I also gained a deeper understanding of shaders. It is a great exercise for me. I hope this can be a foundation for my future study in the graphics area.

8. REREFENCE

Generate Random Caves Using Cellular Automata. Available at: <https://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664> .

Procedural Level Generation in Games using a Cellular Automaton. Available at: <https://www.raywenderlich.com/2425-procedural-level-generation-in-games-using-a-cellular-automaton-part-1>.

Procedural Dungeon Generation: Cellular Automata. Available at: https://blog.jrheard.com/procedural-dungeon-generation-cellular-automata

The Cellular Automation Method for Cave Generation. Available at: https://jeremykun.com/2012/07/29/the-cellular-automaton-method-for-cave-generation/

DirectX 10/11 – Basic Shader Reflection – Automatic Input Layout Creation. Available at: <https://takinginitiative.wordpress.com/2011/12/11/directx-1011-basic-shader-reflection-automatic-input-layout-creation/>

Reflection-Rastertek tutorials. Available at: <Tutorial 27: Reflection (rastertek.com)> .

Picking-Rastertek tutorials. Available at: <http://www.rastertek.com/dx11tut47.html> .

Picking tutorial. Available at: <https://www.braynzarsoft.net/viewtutorial/q16390-20-picking>

Picking tutorial. Available at: <https://www.youtube.com/watch?v=zbzCZeaWrsM>