DirectXTK-based Terrain Generator Graphics Application

# Abstract

Terrain is one of the essential parts of game that makes up virtual world. In order to generate diverse and reasonable terrains, many games use unique world generation algorithms. This application has (1) applied fractal noise to generate low-poly terrain mesh; (2) implemented ray check to adjust terrain shape; (3) created custom shader to achieve multiple biomes and post-processing effects; (4) integrated ImGui for convenient parameter adjustment; (5) optimised user experience with more parameters to control the generating and processing. With adjusting parameters, this application can generate random terrain with three different biomes as well as a mini game.

# Introduction

Game industry have paid high efforts on design and art rather than programming. In other words, the game content taking a huge part of making games. This has resulted in very low profits for producing content-oriented games, and producers may also face risk of underperformance in the gaming market (Shaker, Togelius and Nelson, 2016). Procedural Content Generation (PCG) can possibly overcome this challenge for its contribution to increase the efficiency of content production for designers and artists. This application implemented DirectXTK-based terrain generator using PCG technique. The gameplay of mini game is to find red balls and use mouse to collect them.

* 1. Perlin Noise

This application used perlin noise as pseudo-random function for procedural generation. As lecture has introduced, the ideally noise that Perlin (1985) introduced is suit for this task because of its three features:

* + - Statistical invariance under rotation.
    - A narrow bandpass limit in frequency.
    - Statistical invariance under translation.

A simple function to generate this type of noise was introduced in his article. Perlin noise, including classic noise and simplex noise, uses a permutation table to calculate every point gradient and uses 3𝑡2 − 2𝑡3 (in classical) and 6𝑡5 − 15𝑡4 + 10𝑡3 (in simplex) as interpolation formula to gain nature and efficient results (Perlin, 2002). For those integer points, use mapped table value directly. For those non-integer points, use

adjacent points’ gradient to interpolate a value. This can be easily extended to high dimensions. This application use 3D version to generate terrain, which provides rich variety of terrain shape.

Perlin noise is useful in computing fire, cloud, and other complex texture. Since each point can be computed independently, GPU technology can be used to accelerate the generation.

* 1. Fractal Noise

In order to add more details to results, multiple samples at constant change frequencies are combined to create self-similar shapes which are known as fractal. This summation of noise frequencies is called fractal noise or pink noise.

There are four concepts in fractal noise which can be adjust in application:

* + - Frequency means the frequency at which peaks and troughs alternate.
    - Octaves means the number of addition times of successive noise function.
    - Lacunarity is the coefficient of each successive noise frequency change.
    - Persistence is the coefficient of each successive noise amplitude change.
  1. Post-processing

Post-processing is generally used in 3D rendering and video production. More and more engines, such as Unity and Unreal, support or implement advanced post-processing to improve graphic performance. In 3D game rendering, post-processing technology mainly exists as part of the shader-controlled pipeline. Scene is first rendered to a

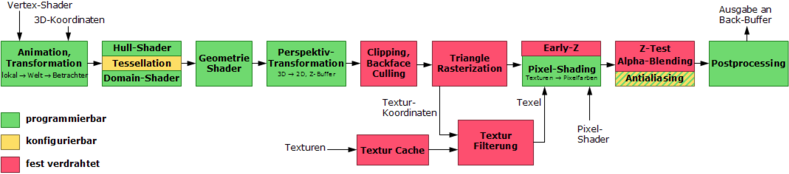


Figure 1. Programmable 3d-pipeline (Wikipedia, 2011).

texture buffer in GPU. This render texture then go through multiple shader passes before displaying it to the screen.

This application implemented bloom effect introduced on the lecture. Bloom is commonly used in combination with high dynamic range rendering (HDRR). This application has six parameters to control the bloom effects separately.

* 1. Ray Intersection

Ray intersection is an essential technique in game interaction, especially in physics system. Through the ray intersection of the object bounding box, it can be judged whether the objects overlap, and make corresponding changes: collision or adhesion. Ray intersection is also used in other part. For example, robots use infrared and other

sensing equipment to avoid obstacles. Similarly, path navigation and pathfinding in games can use rays to achieve even obstacle avoidance and path correction. The visible area of the player in the scene can also be realized by the ray intersection.

This application uses ray intersection to achieve picking ball and edit terrain actions.

* 1. Dear ImGui

Dear ImGui is a bloat-free graphical user interface library for C++. It is fast, portable, renderer agnostic and self-contained (no external dependencies). (ocornut, 2019)

ImGui makes it easy to create clean sub-windows and control the variables, which is useful for developers to create control interfaces. The control interfaces can be used to display status of application parameters and can also be used to debug. Interfaces in this application are used to control states and change parameters.

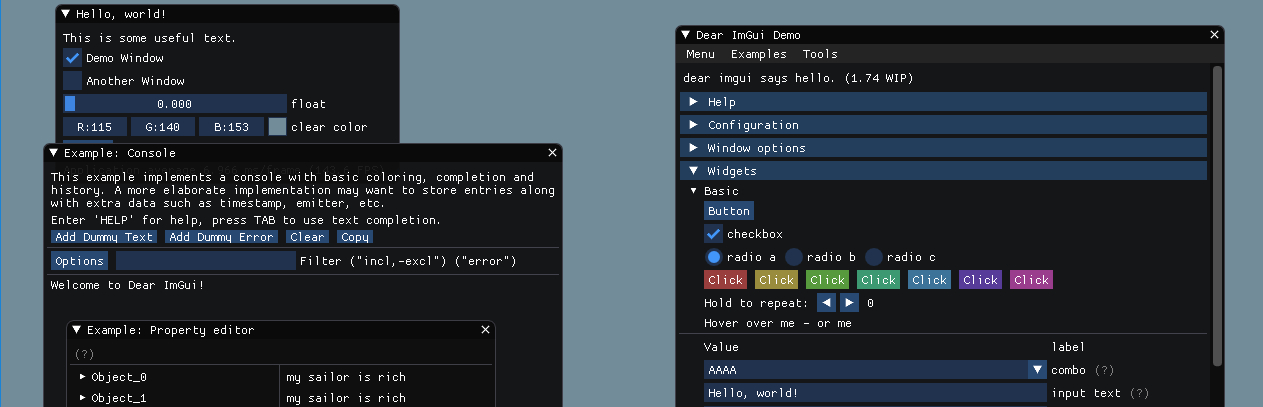


Figure 2. ImGui demo windows.

* 1. Control Guide
     + Press “W” and “S” to move forward and backward based on viewpoint.
     + Press “A” and “D” to move left and right horizontally.
     + Press “Q” and “E” to move up and down vertically.
     + Press left mouse button to collect balls in game mode.
     + Press right mouse button and drag to move viewpoint.
     + Press “Esc” to quit application.
     + Press “Alt” + “Enter” to switch screen mode between window mode and full screen mode.

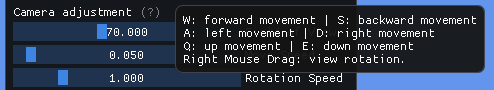


Figure 3. Camera control tool tips.

# Methods

Fractal noise, multiple biomes and GUI control were implemented in the last version. In order to complete the gameplay, making this application interactable, ray check is implemented in *Game* class. The intersection of rays can be used to detect the interaction between the rays emitted from a specific point and the environment. In addition, three pixel shaders are implemented to achieve post-processing effects. The realization of the bloom effect through four rendering processes. At the end, more polish works such as input and components optimise are done to improve experience.

* 1. Application Composition

The application was developed with visual studio 2017 (v141) using DirectXTK (ver. 2019.5.31.1) packet. It also integrated ImGui for GUI interaction. Application are developed based on last version submitted. Scripts structure are showed in Figure 4.

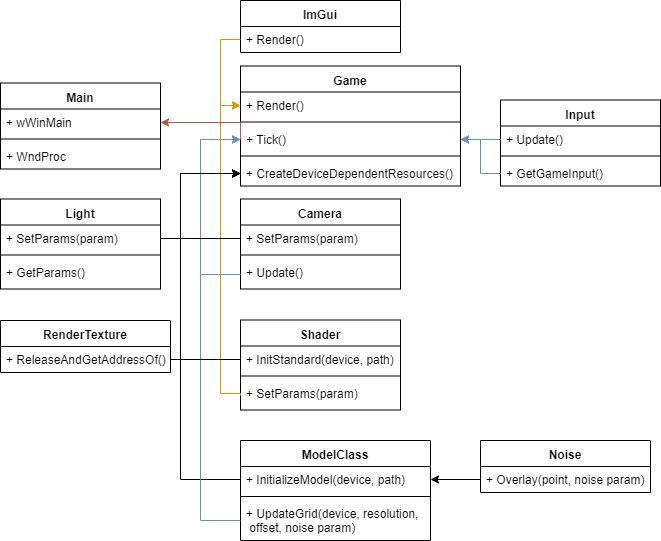


Figure 4. Application class structure.

* 1. Implementation of Interactions

Ray intersection consists of two parts, the ray and the measured area. The main difficulty lies in the transformation of the coordinate system.

The interactive way of the mini game in this application is to collect small balls with a mouse click. Thus, the ray has the transmitter as the camera and the receiver as the ball. Therefore, the rays first need to be converted from the camera clip coordinates to the world coordinates. Then check whether the ray intersection with target object.

Ideally, the ray test function could be integrated into camera class or be in the physics system. This application put this function in the main class for convenience, because the three conversion matrices can be accessed directly in the main class

*RayTest* function is called in *Update* function when ray test is active. In function, two point were created in clip coordinates. Their world position can be obtained through a convenient function provided by *DirectX::SimpleMath.* Using world matrix, view matrix and projection matrix, viewport of camera can unproject clip points back to world position. The direction in world coordinates is determined with near and far point. Then use the ray function to check whether the intersection occurs. The results will be used to decided which event will be triggered.

At the same time, *m\_gameInputCommands* was adjusted in *Input* class to support interaction. Now it can handle left mouse click and record mouse position.

* 1. Implementation of post-processing

To implement post-processing, the application needs to have corresponding pixel shaders and create corresponding resources and call them in the rendering pipeline.

* + 1. Implementation of Three Pixel Shaders

In the *BloomExtract* shader, the downsampling work is completed, pixels that are above a certain threshold are screened out and renormalized to 0 to 1.

In the *GaussianBlur* shader, each point is sampled multiple times based on the Gaussian blur parameters that have been generated in *Game* class. Instead of using fixed weight and offset in sampling (rastertek, no date), the parameters are calculated in struct *VS\_BLUR\_PARAMETERS* based on screen pixel. This allows Gaussian blur to have a consistent effect in different pixel images. The weight coefficient is obtained by the formula of a Gaussian function in one dimension:

𝐺(𝑥) =

1

√2𝜋𝜎2

𝑥2

𝑒−2𝜎2 ,

From a signalling perspective, the Gaussian blur formula is a low-pass filter. It can be used to remove burrs in the image and make the image soft. So it can be combined with highlights to form a bloom effect.

In the *BloomCombine* shader, sampling the two textures and performing saturation processing, correcting the color value of the base according to the value of bloom, and finally combining the values of the two as the output.

* + 1. Resources for shader and rendering

Each shader is loaded in *CreateDeviceDependentResources* function as well as parameter buffers. Meanwhile, the corresponding texture is created in *CreatePostProcessingResources* function for multi-level rendering and debugging. The address of the parameter cache is also bound in this function.

* + 1. Implementation of four render passes

The post-processing function is called after the monitor window is drawn.

After the first rendering, the screenshot is saved in the resource of m\_FirstRenderPass. Then through downsampling, horizontal blur, vertical blur and merge processing, to obtain the final bloom effect. Each step is stored with a corresponding texture, which plays a key role in the debugging process.

* 1. Polish

An additional automatic check box has been added to toggle the automatic update of terrain generation. Reduce the impact on parameter adjustment when automatically refreshing the terrain.

Added additional height adjustment parameters, separating the binding of biome and height. Allows the generator to generate more types of terrain, such as basins.

Added top view window background instead of black background.

Added manual editing of terrain in terrain editing mode, which can adjust the height of terrain locally.

Fixed the bug that the terrain will not be updated synchronously when changing biome parameters.

# Results

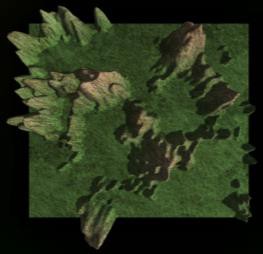
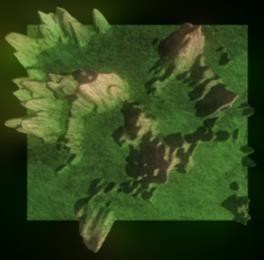
Figure 5 to Figure 7 Shows the final picture under different parameters. Figure 5 and Figure 6 illustrate the threshold can separate color areas well. Figure 7 illustrate the relationship between formula parameters and blur degree. Figure 8 shows the textures of each shader pass.

Figure 5. Final texture of the variety bloom threshold from 0 to 1.

Figure 6. Texture of bloom threshold at 0 and 0.25 without base texture.



Figure 7. Texture of blur amount at 4 and 2 without base texture.

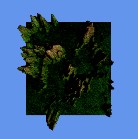


Figure 8. Texture of each pass. (Downsample -> HorizontalBlur -> VerticalBlur -> Combine)

# Discussion

* 1. Implications

These results hold three implications.

Firstly, gaussian blur is a classic blur function of the Photoshop, and is also widely used in video editing, post-processing rendering and other scenes. Through the processing of the threshold, application can effectively and conveniently control the processing of colour areas under specific conditions.

Secondly, by adjusting the blur parameters, it is very convenient to create a focused effect or scene depth effect when the lens transitions.

Thirdly, Multiple passes generally can get good rendering results, but the corresponding increase in GPU overhead. Through multi-step rendering, game makers can use three- dimensional models to create pixel animations, greatly saving development costs (Vasseur, 2018).

* 1. Limitations

This application has three main limitations.

Still did not complete the water simulation. It was originally planned to use multiple sine functions to simulate the flow of water in the shader, but because the management

and method expansion of the shader class were not well completed, the program could not correctly handle the lighting and water surface materials. The uneven flow of the water surface can also be achieved with Perlin noise.

The efficiency of terrain reconstruction and adjustment is low. Since each update involves the creation and destruction of a large amount of memory, a more reasonable cache and object pool should be used to save memory overhead.

With the development of the application, the game class becomes gradually bloated, which leads to higher coupling of some functions. Need to do further decoupling and structural separation of applications.

# Conclusion

This application aims to build a terrain generator using DirectXTK functionalities. Last version has implemented fractal noise, convert the output values to mesh, render mesh with gradient colour, decorate with environment elements and integrate ImGui to interact. The new version optimizes some functions of the old version and adds post- processing, ray detection functions and a mini game. Experiment has explored the cost and performance of build a procedure content application. At the same time, it deepens the experience of multi-rendering pipeline and writing shader.

# Reference

Claudette, G. (no date) Stone-texture-1 [fig.]. Available at:

[https://www.publi](http://www.publicdomainpictures.net/en/view-)c[domainpictures.net/en/view-](http://www.publicdomainpictures.net/en/view-) image.php?image=194996&picture=stone-texture-1 (Accessed: 28 April 2020).

Claudette, G. (no date) Papier-a-texture-deau [fig.]. Available at:

[https://www.publi](http://www.publicdomainpictures.net/en/view-)c[domainpictures.net/en/view-](http://www.publicdomainpictures.net/en/view-) image.php?image=292732&picture=water-texture-paper (Accessed: 28 April 2020).

Hummel, R. A., Kimia, B. and Zucker, S. W. (1987) ‘Deblurring gaussian blur’. Elsevier (Computer Vision, Graphics, and Image Processing, 38), 38(1), pp. 66–80.

Luna, F. (2012) Introduction to 3D game programming with DirectX 11. Stylus Publishing, LLC.

Ocornut (2020) Dear ImGui. Available at: https://github.com/ocornut/imgui (Accessed: 28 April 2020).

Osaka, Z. (2020) Stylized tree 3D model [Model]. Available at: [https://www.turbosquid.com/3d](http://www.turbosquid.com/3d-models/stylized-tree-3d-model-1374946)-[models/stylized-tree-3d-model-1374946](http://www.turbosquid.com/3d-models/stylized-tree-3d-model-1374946) (Accessed:

28 April 2020).

Perlin, K. (1985) An image synthesizer. ACM Siggraph Computer Graphics, 19(3), pp. 287-296.

Perlin, K. (2002) Improving Noise. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002) 21(3), pp. 681–682.

Rastertek (no date) Tutorial 36: Blur. Available at: <http://www.rastertek.com/dx11tut36.html>(Accessed: 28 April 2020).

Shaker, N., Togelius, J. and Nelson, M. J. (2016) Procedural Content Generation in Games. 1st ed.

Vasseur, T. (2018) Art Design Deep Dive: Using a 3D pipeline for 2D animation in Dead Cells. Available at:

[https://www.gamasutra.com/view/news/313026/Art\_Design\_Deep\_Dive\_Using\_a\_3D](https://www.gamasutra.com/view/news/313026/Art_Design_Deep_Dive_Using_a_3D_pipeline_for_2D_animation_in_Dead_Cells.php)

[\_pipeline\_for\_2D\_animation\_in\_Dead\_Cells.php](https://www.gamasutra.com/view/news/313026/Art_Design_Deep_Dive_Using_a_3D_pipeline_for_2D_animation_in_Dead_Cells.php) (Accessed: 28 April 2020).

Walbourn, C. (2018) PostProcess. Available at: <https://github.com/microsoft/DirectXTK/wiki/PostProcess>(Accessed: 28 April 2020).

Wikipedia (2011) Programmable 3d-pipeline [fig.]. Available at: https://en.wikipedia.org/wiki/File:3D-Pipeline.png (Accessed: 28 April 2020).