GAME PROGRAMMING FOR PC AND XBOX REPORT

Coursework report for AG1102A, University of Abertay, Dundee

Richa Sachdeva
M.Sc. Computer Games Technology
Student Id. – 1304869
14th May 2014

Introduction

Procedural generation is a widely used term in the production of media; it refers to content generated algorithmically rather than manually. Often, this means creating content on the fly rather than prior to distribution. This is often related to computer graphics applications and video game level design[Wiki]. In recent years, there have been tremendous development in the procedurally generated content as first it's not repetitive, second the options available are enormous, third it works well for platforms where resources are limited example mobile. Also, with improvements in the procedural algorithms, life like graphics can be generated, so less dependency on the artist, but then these methods are resource intensive, so a trade off is there, which should be worked around based on the given situation.

The coursework associated with the module 'Game Programming for the PC and XBox' required to construct a game that features procedural content generation. (along with another couple of requirements). Some of the features included in the game are :-

- Procedural content generation - Generating terrain, texture over terrain procedurally instead of loading from model/texture file
- Post processing in which a scene is rendered to texture, then blurred and rendered to the scene
- Simplex noise to generate procedural texture

Rest of the report discusses the features of the game and how procedural methods were integrated in the gameplay. It also discusses the code structure, strategies used and the lessons learnt by undertaking this module

Features

The game developed is a demo exhibiting usage of several procedural techniques learnt by undertaking this module. In this, the player which is depicted using a sphere, can move right/left/forward/backward using the arrow keys over the surface of the terrain and has to catch/collect the another sphere while avoiding the rolling spheres, or the enemies. The camera is fixed over the player and moves as the player moves. The player when reaches near the sphere collides with it, increasing the score, and with collision the sphere moves somewhere else, and the player has to catch it again. Presently, the game doesn't end and the player can keep on collecting the spheres as long as they like. The player can increase the level by pressing 'L' key, but only two more levels are added currently. Also, the rolling spheres, which are spread and moving across the terrain are like miniature enemies, from which the player needs to save itself and if the player collides with them, the score is reduced. Ideally or in the future, game will end if the player collides with the rolling spheres after a fix number of times (for example five). The added levels are same to the initial one, only difference is in the size of the terrain, number of enemies and walls rendered at the end.

When the game is launched, and it may take some time to load, initially a startup screen is shown which gives general game related information and on proceeding from there the game starts immediately. The startup screen is made using Photoshop.

Terrain is generated algorithmically (using Mid-Point Displacement algorithm), where based on the pixel position, the height or y coordinate value is set. For this, a 512 * 512 terrain is thought to be composed of smaller 32 * 32 size grids. In a smaller 32 * 32 grid (which is directly expanded from a 16 * 16 pattern), a peak is achieved in the center of the grid, by taking the pixel index and then seeing if it lies in the grid/2 +/- 2 range and if it does, then if the pixel is +/- 2 then make y coordinate +1. For +/- 1, add 2 to y coordinate, and at center add 3 more to the current height (which is y = 2 in current case). But as it results in peaks, which are smoothen by passing the height of the terrain through a smoothing function which takes the value of the neighbouring pixels (eight of them), find the average of the nine and apply that value to the current pixel. The peaks are the barrier to the player as they slow down the player's speed. Ideally the player shouldn't be able to climb them up, but after several attempts when the desired result wasn't obtained then speed of the player has been slowed down on the terrain.

Coloring of the terrain is also done through code instead of using a texture file. For this, color value has been passed in the index buffers based on the pixel's current position and height. Geometry of the terrain played a big role in applying color, as instead of per pixel, terrain is colored on per triangle basis, so understanding geometry and how it's arranged helped in coloring of the terrain. The main reason of doing this was maybe in future while extending the game, color can be dynamic and can change with game's time, player position or some other parameter.

Post processing was the most difficult part. Firstly, one had to think the appropriate way of integrating it in the game, second it is resource expensive, so has to keep that in mind. In post processing, a scene is initially rendered from scene to a texture, then a pixel shader operation, which is blurring in this case is applied to it, and after that it (the texture) is applied back to the screen. So, the terrain has been rendered to a texture, then a blurring operation consisting of horizontal and vertical blurring is applied to that texture, and then that texture has been rendered on the wall models, which are present at the end of the terrain, giving a reflective, blurred surface a blurred mirror kinda feel, and they also mark the end of the terrain.

For blurring, the nearest eight neighbors of the current pixel are taken and based on their distance from the current pixel, a value, or a weight is associated with each of them. Then a weighted average of the pixels is taken to find the value of the current pixel. In this, the vertical and horizontal blurring is combined together. For this, for the given pixel, given width and height, in a single pass, the blurring was performed on the given pixel, which was basically reducing the intensity of color at a particular pixel location, based on the value that comes from weighted average, is taken not just from x coordinates, but also from z coordinates (as the terrain is spread over x-z coordinate so blurring was applied over the x-z coordinate). So, in total there are eight neighbours, which carry a weighted value not just of x axis, but of z axis as well. Two reasons, first more intensive blurring, second, clubbing vertical and horizontal blurring together is faster than first applying a blur in x direction and then in z direction and also blurring shouldn't be limited by the coordinates and should include both the coordinates. A blur in the y direction wasn't applied as terrain is spread over X-Z axis and blurring using the same axis seemed natural.

Rolling spheres are present on the terrain and they are serving the role of miniature enemies. Based on whether they're moving along X axis or along Z axis, their color is decided, which also matches with terrain pattern over a certain height and was chosen for the same reason. Rolling spheres are constrained by the terrain size and move inside the given area. While, at the moment, nothing happens when the player collides with any rolling sphere apart from collision count increasing and score decreasing, in future a sound will be added on collision and game will end after a certain number of collisions. The speed of the rolling spheres is variable and increases towards the center of the terrain, while the player speed is constant throughout and when the level increases and terrain gets smaller in size, avoiding enemies become lot more challenging.

Collision detection is between spheres, so a circle to circle collision mechanism was used, in which, based on the current positions of the two spheres, distance between them is calculated and based on whether this distance lies within the radii of 2 spheres or not, collision happens or not and in this case proved to be more appropriate than the ray intersection method which was implemented initially.

The player, represented by the sphere, moves on top of the terrain and its movement is controlled by the user. The texture of the player is generated using perlin noise, in it's simplest form. To implement Perlin noise, the Hermite blending function (3t^2 - 2t^3) is used. In this, using four points in 1D, a square is formed. From these formed squares, a gradient is picked. The gradient so picked is pseudo random for noise to be repeatable and for that gradient cannot be totally random, so a fix number of gradients of same length but different direction were picked. In the current case, 8 gradients were used to generate the noise in the pixel shader. Though initially, perlin noise pixel shader implementation was done with the intent of rendering clouds on a skydome, but due to the tricky nature of the shaders and bad time management on programmer's part, that was discarded and noise was applied as a texture on the player. Also, apart from the catching sphere texture, all textures are generated programmatically, which is a great leap from last semester and this has been an immense learning experience with respect to shaders. The catch sphere has been given a distinct texture so that it is visible from a distance as well.

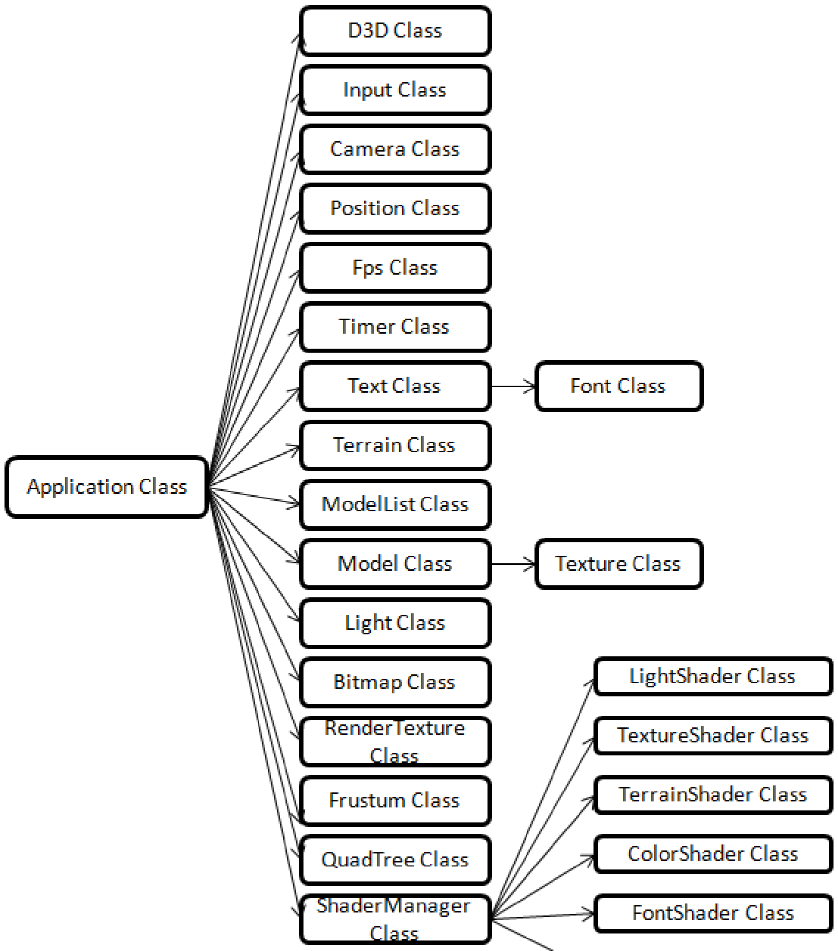
Two more levels are added to demonstrate the procedural content generation as the size of the terrain is controlled based on the level the player currently is. When the 'L' key is pressed, terrain size decreases, with it the index buffer is reinitialized with the current height and width and is the reason for the momentarily pause/hang that happens in the game. The player sphere and catch sphere also repositions themselves, but sometimes the catch sphere may disappear which is a known bug. So, the hang and the disappearance are two main bugs currently present in the game and had there been more time they definitely would have been resolved. The camera is always above the player, and player has been given left-right movement, no rotation that's why camera is stuck above the player when the player is moving backwards.

Program Structure

Code structure was based on the framework provided in the lab tutorials and the RasterTek tutorials. Code is organized in four main folders - External dependency - for external files, header files - stores all the header files (.h files), shader files - stores all the shader (pixel and vector shader) files, source files - stores all the source files (.cpp files). Control runs from the Main file to System file to Application file. Application file controls the various aspects of the application like displaying Cpu usage, counting Fps, rendering terrain, keeping tab of collisions, performing collision related function. The program is compatible with DirectX 10 and onwards.

Class Diagram

Class diagram outlines the basic code structure and demonstrates how it is organized, how the control passes from one class to another. It also tells about the classes present and how they interact.



This semester emphasis was given on understanding the framework, improving it by eliminating duplicate or redundant classes so that the code is modular and makes full use of object oriented programming features. First step in this direction was implementation of a shader manager class, which manages all the shaders present and also adds a layer of abstraction and makes application layer organized. The shader manager class stores reference to all the shader objects, instantiates them, and from the application class the correct shader function can be called using shader manager object present there.

The rolling spheres are given a separate class to manage the initialisation and rendering of the spheres, as their behavior is same it made sense to delegate a separate class to them which would take care of all the features. For this instantiation of the spheres is done in the modellist class, where the sphere models are called, assigned a position on the terrain and based on their position, color is assigned to them.

Generating, smoothing, coloring and rendering of the terrain is done in the terrain class. These three steps were taken to ensure that the application class is organized, clean, and is modular.

Fps and Timer can be clubbed into a single utility class, but due to time constraint, it wasn't done.

As terrain is huge so there is no point in rendering entire terrain every frame. For this, only the part of terrain's that's visible has been displayed and rest been culled out. For this, terrain has been divided into a number of smaller terrains based on the fix number of triangles it contain. Now for each smaller terrain, triangles belonging to each of the terrain are identified and a vertex buffer for that smaller terrain is created. So, in a way instead of using single big vertex buffer, several smaller vertex buffers are used and while viewing based on the player's location only a smaller vertex buffer is rendered instead of the larger vertex buffer and because of this speed is gained while rendering the terrain.

Results / Critical Appraisal

The game was developed with the intent to learn about procedural methods and to develop a game using DirectX. Though the initial idea of the game was very different, it was an open ended dynamically generating terrain where player can explore the terrain and based on the interaction of the player with the objects on the terrain, a fractal would be developed, much on the lines of the game "Flower", only the following leaves were to be replaced by the fractal pattern which would evolve with the time. But implementing fractals proved to be hard, due to limited knowledge of the pixel shaders.

As the coursework mentions developing a playable game, so ideas and planning right from the beginning was directed towards that, which was good as it helped in focussing and spending more time on things which would help with the same. It also provided the focus and direction which helped in organizing the game idea in better way than the last semester.

Code organization has also improved which can be attributed to familiarity with the DirectX, the existing framework and the simple structure of the game. Main application file to govern several different aspects of the application, like generating Terrain, handling and processing input, displaying game status with help of Text class, which can be thought of a simple user interface and in future with the evolution of the game can be given a separate screen, while maintaining the object oriented features. Though post processing has been implemented, it's not been aptly used in the game in current scenario and it's use is more or less abstract. It's a very powerful tool and could have been used to attain dramatic results but due to time constraints and the fact that post processing concept is difficult to understand and implement, it hasn't been used to its full context in the current game.

Procedural generation is incomplete without trying to implement a terrain with Perlin or some other variant of noise, and while initial attempts were made in the same direction, but results obtained were not satisfactory. The initial game idea of an open world terrain, with sporadic heights/peaks was difficult to attain using perlin noise as one needs an in-depth understanding of the noise. So, after several attempts idea of generating terrain using Perlin noise was discarded, and a terrain with peaks at regular distances was generated. An important lesson gained by implementing terrain not using noise function was getting the clarity on how terrains are generated and if a simple terrain can be generated than a more evolved or complex terrain can also be generated if required. In the current case for generating terrain, approach similar to mid point displacement is used. The terrain is thought to be composed of several grids, where pixel on each grid has been given a height value, which increases towards the center.

Perlin noise which was initially learnt to implement the terrain was then used to generate the clouds, and when not able to implement in way envisioned, it was used as a texture on the player's sphere surface. Though this is far from the expected outcome, but due to constant attempt to implement noise using shaders proved to be a good exercise and helped in understanding the working of shaders to a greater depth.

The coursework submitted is a demo and is far from finished and over the time many things can be improved/added. Features at top of the list that will be added, but weren't in the current submission due to time limitations are fixing the hanging of the screen when level is changed, fixing the disappearance of the catch sphere(it happens sometimes and not all the time), sound on collision of the player sphere with enemies and catch sphere(two different sounds). Ending the game after certain fix number of collisions. Stopping the player from going out of the terrain, which at present can happen. Even though the bounds are applied still camera can go backwards, so camera positioning can be improved. Also, right now levels are changed by user on will (by pressing Key L) and it can be improved so that after a certain number of points are collected, user moves to next level, which will also help in dealing with the present hanging of the screen. An improved user interface with ability to pause/restart game can be added. Adjusting camera by giving the player sphere a rotation value so that moving backwards, and forwards is just moving on the terrain, instead of moving forward and backward.

Conclusion

Undertaking this module was very beneficial as it has helped in understanding procedural generation - how terrains are generated, how procedural textures are implemented. Though these days with improve graphics, procedural textures might not be used extensively, but having information on how they can be implemented is necessary. By trying out various techniques has helped in improving the programming knowledge and has given confidence in the procedural content generation. Also, all the techniques that have been learned be it procedural terrain generation, or texture, or post processing have been used in building a game prototype, so the module has helped in understanding the process of game creation at an individual level, starting from looking for a game idea, iterating and refining the idea to an extent, and then to implement them and develop a game has been a useful and immensely important exercise. Also as the demo is far from finished and there are many things which can be added, improved but due to the time constraints weren't, but this has increased the programmer's understanding of the game development process and how attention to detail is required and important as a finishing touch takes game to another level, but such is not the case with the current demo.

The lectures were quite informative specially about the latest development with respect to the procedurally generated content, which tells about the relevance of the course and the information regarding various techniques and methods used, keeps the student motivated as well.

As it was the second semester of using DirectX, and as this time extent and scope was vast in comparison to the first semester, so it was challenging and exciting at the same time. And only due to the vastness of the module, directx capabilities have been increased multifold in comparison to the first sem and by developing a game prototype has given the programmer confidence in using directx, including its advanced features with ease and with the limitations/challenges in implementing some features have given insight on how to manage time and focussed.

References

- Bett, M. (2014) *AG1101A Lecture Notes,* University of Abertay, Dundee
- Rastertek.com (2014) *RasterTek – DirectX 10 and DirectX 11 Tutorials.* [online available at:] http://www.rastertek.com/ [Accessed : 12 May 2014].
- DirectX SDK Documentations and Samples
- Luna, F. *INTRODUCTION TO 3D GAME PROGRAMMING WITH DIRECTX 11,* 2012, Mercury Learning and Information LLC, Canada

- Simplex noise (2014) [online available at:] http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf [Accessed: 10 May 2014]

Screenshot of the game