DIRECTX SCENE REPORT

Coursework report for AG1101A, University of Abertay, Dundee

Richa Sachdeva
M.Sc. Computer Games Technology
Student Id. – 1304869
15th January 2014

Introduction

Computer graphics are central to any game developed and from a mere pixel representation in games like Pong, Breakout to almost real, life-like representation in games like Last of Us, GTA5, they have come a long way. This in part was made possible owing to the tremendous improvement in computer hardware, which not only facilitated advancement in computer graphics but also popularizes them and made them indispensable for the video games. Also, because of the popularity, many computer hardware companies started supporting different graphic libraries. One such library is DirectX, which is a collection of application programming interfaces (API's) that helps in managing graphics on Microsoft platform and is a component of Microsoft Windows. Direct3D is the 3D graphics API within DirectX and is the library used to create video games and manage other graphic related applications.

The coursework associated with the module 'Programming for Games' required programming a 3D scene using DirectX. Some of the features included in the scene are: -

- Loading and rendering models from text files
- World, view and projection transformations for the models rendered in the scene
- Ambient and diffuse lighting
- Use of different textures for model and particle system
- Camera control using keyboard
- Frames per second counter

In addition to above features, other features included in the scene are: -

- Direct3D Sound
- Particle System - Shader generated
- 2D texture rendering for start up screen
- Bill boarding and Instancing for the particle system
- Sky-dome
- Reflection
- Translate Shader to rotate Spheres

Rest of the report discusses the above-mentioned features in detail. It also discusses the code structure, strategies used and the lessons learnt by undertaking this module

Features

The 3D scene as a requirement for the coursework was developed with the intent of learning features of DirectX library and becoming familiar with different aspects of it. As the library is huge, emphasis was laid on learning the core features first. With the foundation laid in the current semester, further advanced concepts can be learned and added in the scene in the next semester.

The scene represents a “Checker’s world”, and has been constructed using a single texture to give a unique character to the scene. Also, as the Sky Dome uses the same texture, it gives the feeling of being inside a box, which was done to give an idea of a closed space. The scene consists of a Sky Dome and inside is present a reflective floor that reflects the spheres hanging at a certain height. While the entire scene represents a closed space, the reflection in the limited space gives idea not only of the ambient light, but also of the relative distance between the floor and the dome. Moreover, the scene makes full use of programmer’s creative vision, which in this case is not limited by the ‘reflections’ in the real world.

As the scene is abstract it can be extended and suited for different situations. The first attempt was to make a floor with variable, dancing lights, but once implemented it wasn’t coming out as desired. Thus, the idea was discarded and the present idea of a reflective world was implemented. Moreover, due to the abstractedness of the scene it can be easily customized to suit as per the requirements of a game.

Spheres are rendered using a simple sphere model provided in the lab tutorials and as there are multiple spheres so instead of making an array or individual model for each of the spheres, a separate class for handling the spheres and assigning position to them has been used. The position for each of the sphere is generated using the loop, to fasten the process and reduce error by hard coding the positions. The rotation of the spheres is achieved by translating the texture, instead of calling the rotation matrix. The reflection was achieved by first calculating the reflection matrix. The spheres were rendered on the floor texture using the reflection matrix and then finally the floor was rendered using the reflection shader.

Camera controls include movement in all the directions. The user can navigate left, right, forward, backward using the arrow keys, up, down using the spacebar and the control key. The W, A, S, D, Q, E keys are used to control the pitch, yaw and roll left, right respectively. Sound can be muted using M key, N key to unmute it. Escape (Esc) key is to terminate the program. Enter key is used to go from initial screen to the second screen.

As knowing about the performance of the system is essential, FPS and CPU counters were added to understand the performance of the system and how much of the resources were used. Also, knowing about the system performance helps in optimizing the system and eliminating the redundant and other resources, which prove to be expensive.

Particle system is generated using shaders and the two-particle systems in the scene were rendered using instancing. As the systems differ only in the position, it was better to use instancing in which only a single vertex buffer is used. It remains cached on the video card and is modified, rendered per instance; the information for each instance is present in the other buffer and is passed onto the vertex buffer (www.rastertek.com). Particle system is also bill boarded, so the particles always face the user based upon the user's position in the world.

The scene also uses a dome as a background, and has been rendered with the same texture, only with different dimension. Instead of using one single picture, several small dimension pictures were clubbed together, increasing the detailing and the clarity of the dome's texture. The texture was taken from the website (http://sammigurl61190.deviantart.com) and is free to use for academic purpose. The sky dome texture was created using Microsoft Paint tool.

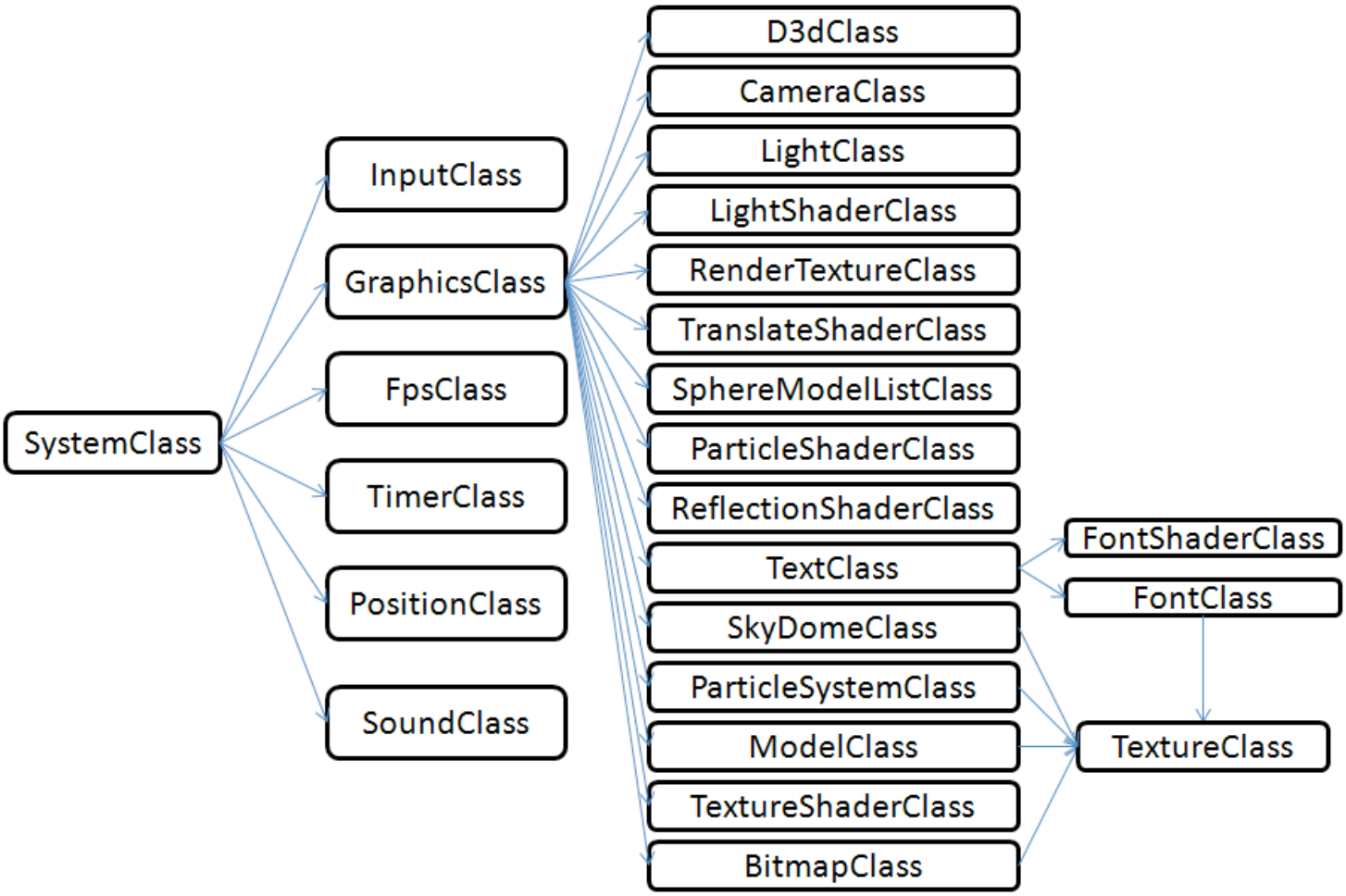
Direct Sound is used to implement the sound in the scene. The background music is played in loops and using the keystrokes M, user can mute it and using the key N, can unmute it. Sound is in wav format and is taken from http://www.mediacollege.com/ and the sound file 'windchimes.wav' has a public domain license so it's permissible to use, without infringement of any copyright laws. The initial .wav file obtained was of lower quality and hence Audacity software (http://audacity.sourceforge.net/about/) was used to increase its quality. It's a free, open source software distributed under GNU GPL, and was used to increase the sampling rate and hence the overall quality of the sound file.

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. System file controls the various aspects of the system like displaying Cpu usage, counting Fps, rendering graphics (by passing control to GraphicsClass).

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.



Results

The scene incorporated many DirectX features and on the whole the exercise was fruitful as it did a good job of explaining basic graphic rendering concepts and the learning's will be useful in the next semester. The scene incorporated several different techniques like bill boarding, reflection, sky dome, translate shader, 2d rendering as the intent was to include as many features as possible of the DirectX programming.

Code was structured based on the basic framework provided in the labs, and while the framework was expanded to accommodate for different classes and functionality, the structure remained more or less same as very less attempt was made to incorporate inheritance, polymorphism or other features of object oriented programming. A resource effective technique would have been to club different shaders together, or to make use of inheritance by constructing a base abstract shader class, which incorporates the basic functionality of rendering the graphical object. The other shaders like texture, lighting, translation can extend from the base class and add their functionality and override the base class functions. But due to time constraints, and inherent difficulty, which the shaders possess, this wasn't implemented.

Fps count was around 79~80 which is acceptable but low, given the complexity associated with the scene. One reason can be the possible use of translate shader on spheres instead of calling the Api function for rotation, as a direct function call is certainly faster than the shader method. Also, a generic model class from which different models like the one that can be instanced, based on the number of models used in the scene could have been constructed. The more specialized model classes can then inherit from the base class and be more object oriented in their approach.

Managing time and having a long-term vision are important. A clear goal of being able to program a game at the end of the module would have helped the programmer in having a different outlook about the module. But lessons are learnt; hope it will be helpful in the future. On whole, the exercise was fruitful as it did explain and familiarized the programmer with the DirectX library and several of its features.

Conclusion

The initial framework provided in the labs was a good structure to begin with. Though it took some time to get familiarized with the framework, but once understood, it became quite easy to add new features and experiment with the different options available in the DirectX library. It did a good job to understand the basic graphic concepts, but somehow a correlation between the DirectX graphics Api and the C++ language wasn't been made, as many features of the language were left untouched. It can be the result of programmer's limited experience with the language. Also, there was less concentration on programming games and more on programming, rather rendering graphics on the scene. Guess that was essential to reduce the complexity for the beginner's.

Planning and time management are equally important as lack of planning results in trying different things without any concrete idea of how the final scene should be. Time management is essential as without setting deadlines, being able to understand the DirectX Api would be difficult and if left for last minute, then nothing much will be gained out of this exercise.

On the whole, the module taught the basic concepts of Direct3D well. It made the programmer familiar with the graphics framework. Also, as the graphic concepts are similar across other graphic libraries like OpenGl, SDL, the learning's from this module can be used there as well as in the other graphical applications.

References

- Bett, M. (2013) *AG1101A Lecture Notes,* University of Abertay, Dundee
- Rastertek.com (2013) *RasterTek – DirectX 10 and DirectX 11 Tutorials.* [Online available at:] http://www.rastertek.com/ [Accessed: 13 January 2014].
- DirectX SDK Documentations and Samples
- Luna, F. *INTRODUCTION TO 3D GAME PROGRAMMING WITH DIRECTX 11,* 2012, Mercury Learning and Information LLC, Canada
- Deviantart.com (2013) *The Checkered Floor* [online available at:] http://sammigurl61190.deviantart.com/art/Tileable-Checkered-Floor-7456673 [Accessed: 13 January 2014]
- Mediacollege.com (2013) *Wav Sound* [online available at:] http://www.mediacollege.com/ [Accessed: 13 January 2014]
- Audacity.sourceforge.net (2013) *Audacity Software* [online available at:] http://audacity.sourceforge.net/about/ [Accessed: 13 January 2014]

Screen shot of the scene