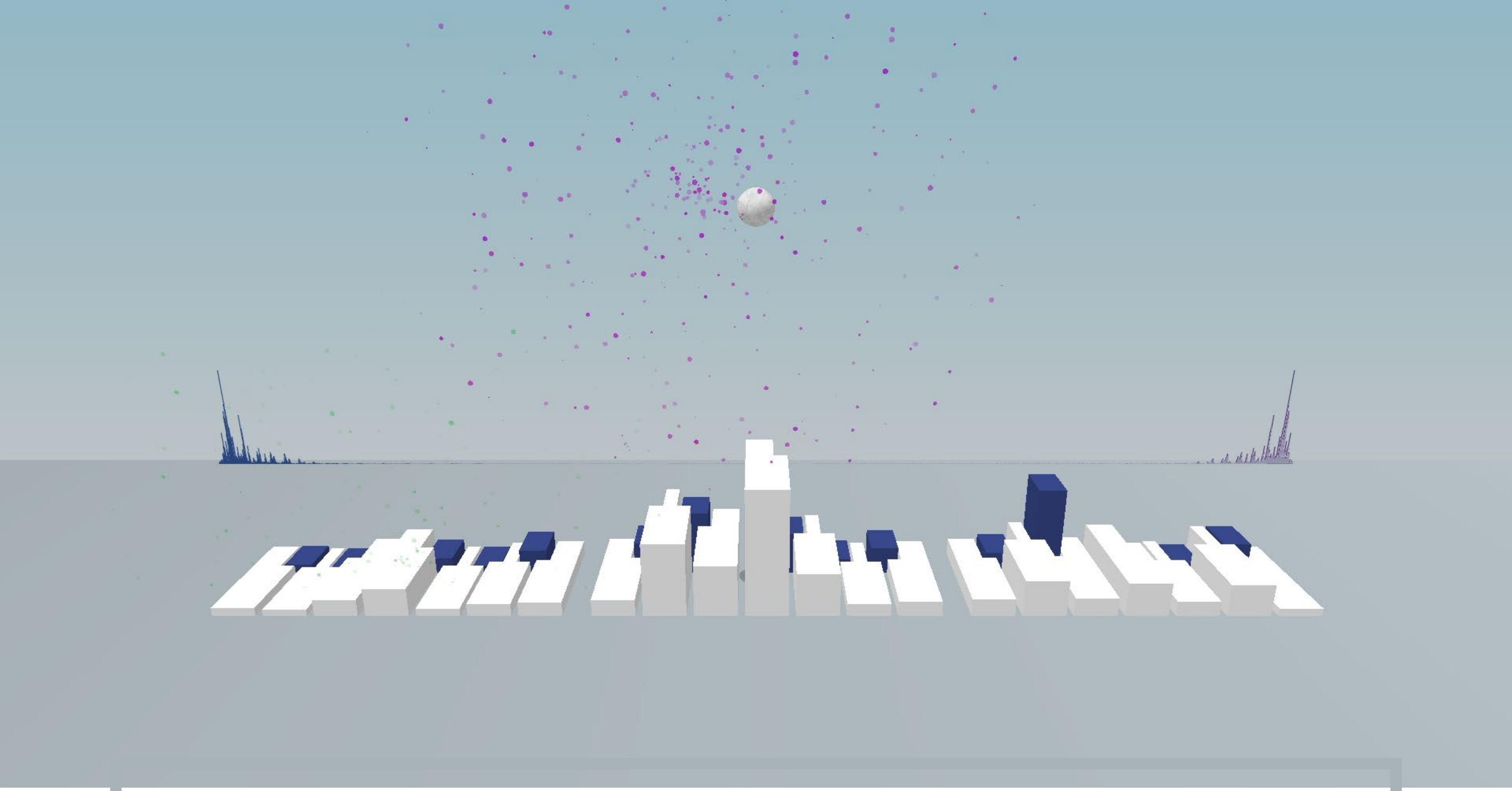

CPT205 Computer Graphics

Assignment 2: 3D Modelling Project

The Music Visualizer

Name: Jingyi Wang 王景祎
Student ID: 1929591
Programme: BEng Digital Media Technology
E-mail: Jingyi.Wang1903@student.xjtlu.edu.cn
Date: Dec. 13rd, 2021

Instruction:

Please make sure to turn on your speaker first!
Arrow keys move the snow ball, and
Space key jumps the snow ball.

- **Player Stage:** Please enjoy the music first.
- **Free Stage:** After the music is finished, you can try to interactive by:
  - **Moving** the snowball to **hit** the piano keys, or
  - **Clicking** on the piano keys with the mouse.

Xi'an Jiaotong-Liverpool University
西交利物浦大学

1. Introduction

The music visualization is typically a feature in the media player software, which can generate the animation from pieces of music and render in real time with the music as it is playing. This report purposes a quite beautiful music visualizer as the assignment of CPT205 module. The purposed music visualizer uses Fourier Transform to get the volume on different frequencies, and then uses a range of graphics techniques, such as hierarchical modelling, viewing and projection, lighting and materials, and texture mapping to render the 3D scene, piano keys, snow balls and other objects while the music is playing. The project also contains some interesting interactions between the user and snowball, the user and piano keys, and between the snowball and piano keys.

2. Audio Signal Processing

Audio signal processing is a crucial part of this assignment. Because human is very sensitive to continuous audio signals, there are quite a few details to consider in terms of audio-video synchronization and waveform generation. I will briefly introduce this part.

2.1 Music Transcription

Audio transcription is the practice of notating a piece or a sound in wave format. In simple terms, it is to get the intensity of each musical note (do, re, mi…) at the current time. First, I crop a piece of music which has 2048 samples. The Hanning window function [1] is multiplied to the corresponding samples:

$$w[n] = 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) \quad 0 \le n < N, N = 2048$$

Then the Fast Fourier Transform (FFT) [2], shown in Figure 1(a), is applied to this piece of Hanning weighted data:

$$x[k] = \sum_{n=0}^{N-1} \exp\left(-i2\pi k\frac{n}{N}\right) x[n] \quad k = 0,1,\dots,N-1$$

After that, for each piano key, I purposed a triangle filter to get the height of the piano key from the result of the Fourier Transform:

$$w[k] = \begin{cases} (k-k_L)/(k_0-k_L)\,, & k_L \le k < k_0, \\ (k-k_H)/(k_0-k_H), & k_0 \le k \le k_H, \\ 0, & otherwise \end{cases}$$

$$Height = \sum_{k=0}^{N-1} w[k]x[k]$$

In this formula, $w[k]$ is the result of FFT, and k_0, k_L, k_H are the frequency of the current musical note, lower note, higher note respectively in the transformed frequency axis. Figure 1(b) shows the purposed triangle filters.

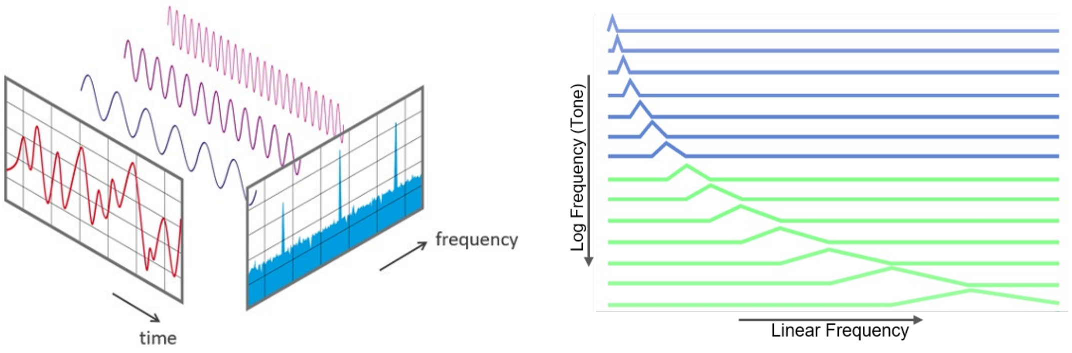

Figure 1: (a) The Fourier Transform; (b) The purposed triangle filters.

When rendering the heights of piano keys and the raw FFT results, a Time-Smoothed technique [3] is applied to the heights, to reduce the intense jumping of the piano keys:

$$\widehat{height}[t] = a \cdot height[t] + (1-a) \cdot \widehat{height}[t-1]$$
$$a = \begin{cases} 0.9, & height[t] > \widehat{height}[t-1] \\ 0.04, & height[t] \le \widehat{height}[t-1] \end{cases}$$

2.2 Audio Decoder and Player

For the decoder part, I did not use any library, but implemented a wav file loader manually (see wav.cpp/h). It only supports 16-bit-2-channel audio currently, which is the most common type. The wav file is relatively easy to read because it is uncompressed.

To be able to play music in sync, I use the Miniaudio [4] library to operate the audio device. It is quite small and in single header file. The library will call the callback function when the next piece of audio is needed. I need to copy the next specified number of frames (typically 480) to the specified address in the callback function.

3. Graphics and Rendering

This assignment uses a range of the various graphics techniques learned in this module, e.g., hierarchical modelling, viewing and projection, lighting and materials, and texture.

3.1 Hierarchical Modelling

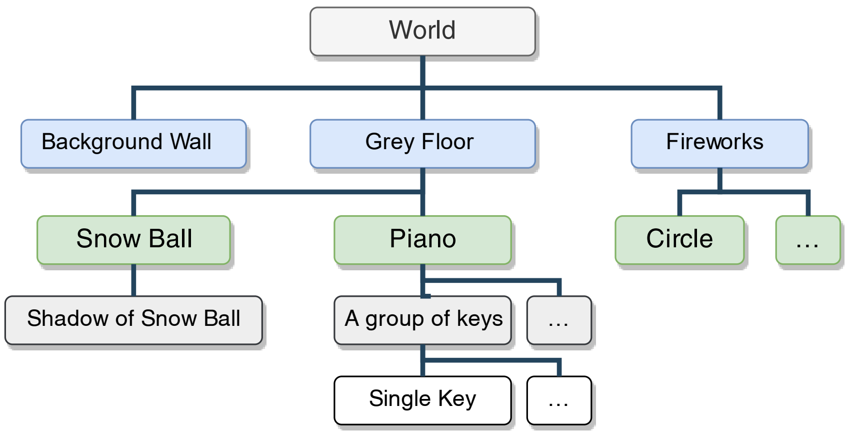

Figure 2: The hierarchical tree of this project

In order to draw each object in a hierarchical way and to perform the coordinate system transformation in a reasonable way, I use the hierarchical modeling method. Figure 2 shows the hierarchical tree of the objects. To facilitate the interaction of different objects, most of the time I'm using the floor coordinate.

3.2 Viewing and Projection

This project uses the perspective projection. To avoid the stretching or squashing of the screen when the window is resized, I call the viewport and perspective function in the reshape callback function.

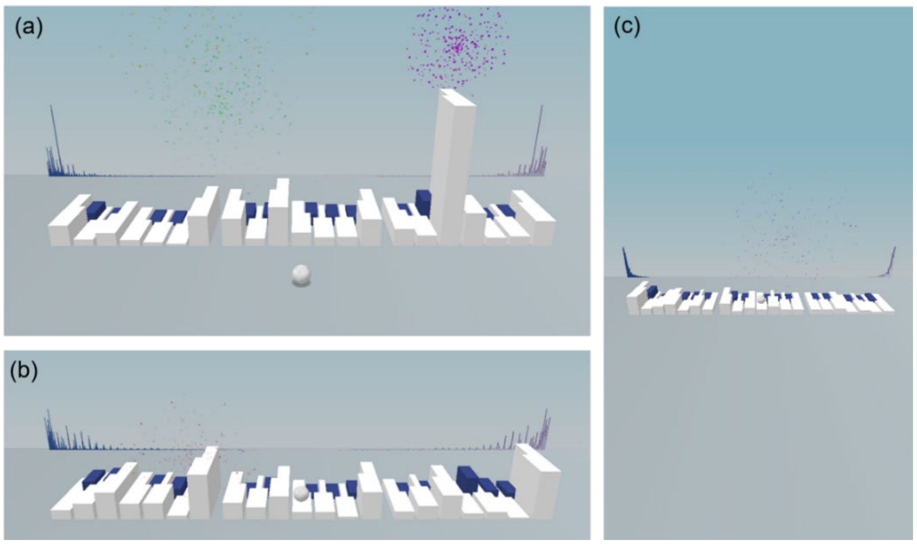

Figure 3: (a) The initial shape of window, (b) The squashed window, (c) The narrow window.

However, there was another problem. The *gluPerspective* function has a parameter *fovy*, which means the viewable angle of pitch axis is constant. But the viewable angle of yaw axis should be constant if we want to see the whole piano when the window is reshaped. So, I re-implemented the perspective function to solve this problem, as shown in Figure 3.

3.3 Geometry and Lighting

OpenGL never has a geometry of “piano key”. So, I storage the shape of a piano key by a vector of all points of the bottom surface, and a height value. When drawing the piano key, I drew the upper surface, lower surface as polygons at first, and side surfaces as quads. The problem is that the shape of white piano keys are not convex polygons so OpenGL can not properly draw. The screen shot is shown in Figure 4(a). My solution is to draw two rectangles for each upper or lower surface.

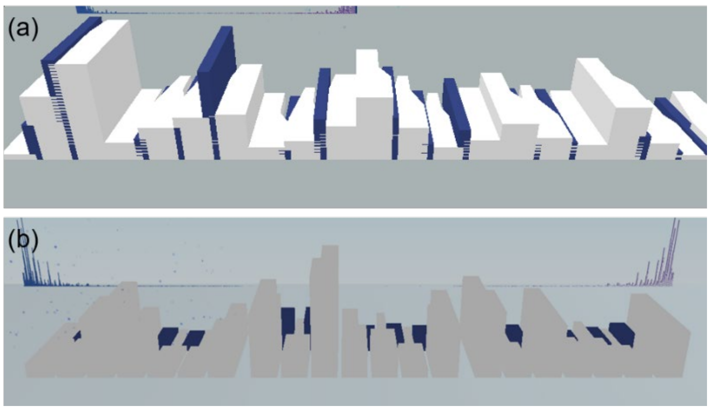

Figure 4: (a) Draw non-convex polygon with OpenGL, (b) Polygons without normal vectors.

The normal vectors of each vertex of each polygon should be correctly calculated and set, to get the correct lighting and shading effects. Figure 4(b) shows the screenshot when the normal vectors of the piano keys are not correctly set.

3.4 Texture

To get a snowball effect, a photo of a snowy field was used as the sphere's mapping. Figure 5 shows the texture and screenshots.

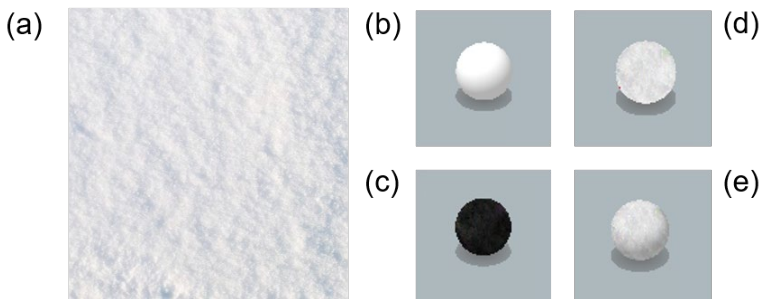

Figure 5: (a) The show texture, (b) Sphere without texture, (c) Sphere with error texture environment parameters, (d) Sphere with texture but no lighting, and (e) Final result.

4. Interaction

The interaction can be happened between human and the program or between the two objects. Figure 6 shows the main interaction methods in this project. The human player can control the snow ball by using arrow keys and space key for jumping. Through my physics engine, the ball and the piano will affect each other, such as playing sounds or changing the movement of the ball.

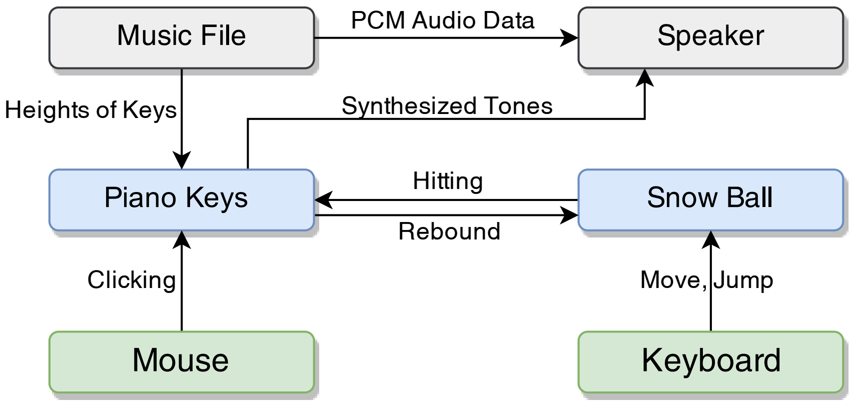

Figure 6: The interaction methods between human and serval objects

References

[1] "numpy.hanning — NumPy v1.21 Manual." https://numpy.org/doc/stable/reference/generated/numpy.hanning.html (accessed Dec. 13, 2021).
[2] "FFT in a single C-file - Stack Overflow." https://stackoverflow.com/questions/8801158/fft-in-a-single-c-file (accessed Dec. 13, 2021).
[3] "java - How to filter FFT data (for audio visualisation)?," Stack Overflow. https://stackoverflow.com/questions/20408388/how-to-filter-fft-data-for-audio-visualisation (accessed Dec. 13, 2021).
[4] "miniaudio - A single file audio playback and capture library." https://miniaud.io/ (accessed Dec. 13, 2021).