Introduction

CPT205 Assignment 1 is a two-dimensional modelling project, which requires a combination of the theoretical knowledge such as polygon, viewing, transformation, etc. learned in the first half of the semester and OpenGL and C++ programming skills. The goal of the assignment is to create a greeting card to celebrate the 15th anniversary of XJTLU using a combination of the above skills.

There are four main elements in my design, which are:

- Balls
- Fireworks
- Ball Matrix
- Oscilloscopes

Each element is encapsulated as an object. This report will then describe the design thinking and implementation of each element separately.

Ball Object

Ball object is the basic unit of the more complex elements: fireworks and ball matrix. As needed, each ball has the following main properties:

- Center coordinate
- Radius
- Velocity (x and y direction) (Pixels / s)
- Color (in HSV color space)
- Luminance (opacity between 0 and 1)

The reason I chose HSV color space is it matches the human eye's perception of color. According to Figure 1, the hue, saturation and lightness can be adjusted independently without affecting the perception of other channels. So, the ball objects storage HSV colors, and convert to RGB values (implemented in “color.cpp”) when the balls need to be drawn.

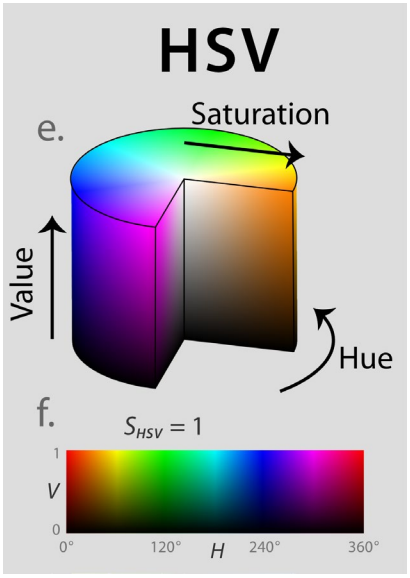

Figure 1. HSV Color Space (Wikipedia)

Each ball is actually a polygon with a sampling number of vertices since OpenGL has no method to draw a circle. In the beginning, I set the `samplingNumber = 600`, and found lagging when drawing a large number of balls. Finally, I set the number is 12 for fireworks, and found that it still looks like

a circle and the frame is smoother.

I draw a new frame in the period of 16ms, so the frequency is about 60Hz. When drawing a new frame, the `nextState()` method should be called to update the position and lightness of the balls. The current velocity vector is first obtained by integrating the gravitational acceleration and air resistance. The acceleration vector caused by air resistance is calculated by:

$$\boldsymbol{accel}_{air-resistance} = k|\boldsymbol{v}|^2(\frac{\boldsymbol{v}}{|\boldsymbol{v}|})$$

where k decides the magnitude of the air resistance. Then the current position is obtained by integrating the velocity. Finally, the lightness of the ball is updated:

$$Lightness(t) = a^t$$

where a is randomly between 0.2 and 0.8 for fireworks, and t is in seconds.

Firework Object

Briefly, the fireworks object achieves the fireworks explosion effect by creating a lot of (typically 360) ball objects with the same starting position and similar color hue, but with many random properties. Here lists the properties of the balls it generates and indicate the application of the random function:

- **Initial Positions**: Same point.
- **Initial Direction of Velocity**: Equidistant distribution around 360°.
- **Initial Magnitude of Velocity:** Random between [0 ~ `maxSpeed` of firework].
- **Color:** `centerHue` (in degree) plus random between -10 and 10 degrees.
- **Luminance Decay Factor:** Random between 0.2 and 0.8.

The variables `maxSpeed` and `centerHue` mentioned above are two properties of each distinct firework. These two properties are also randomly chosen when a new firework is generating. Because of the existence of the air resistance, the max speed of the firework actually decides the size of the firework. This is the same as in reality. The variable `centerHue` can be considered as the theme color of this firework, and each ball has a random difference between its own color and the theme color.

Each firework object contains a vector to storage the ball objects. When the brightness of a ball is too low, it is marked with a lazy delete tag and the ball is not drawn after that. All fireworks objects are maintained through a list, as we always insert new fireworks at the end of the list, sequentially traverse and draw all fireworks, and then delete those that are not visible.

Ball Matrix

The Ball Matrix occupies a fairly dominant position in the frame. The inspiration for this design is from the visualization of my researching work about synaptic devices and brain-like computing. In this paper we propose a new kind of synaptic device which can be used to replace the weighted connections in neural networks. Figure 2 shows the weights of the synaptic devices during the training process of an image classification neural network.

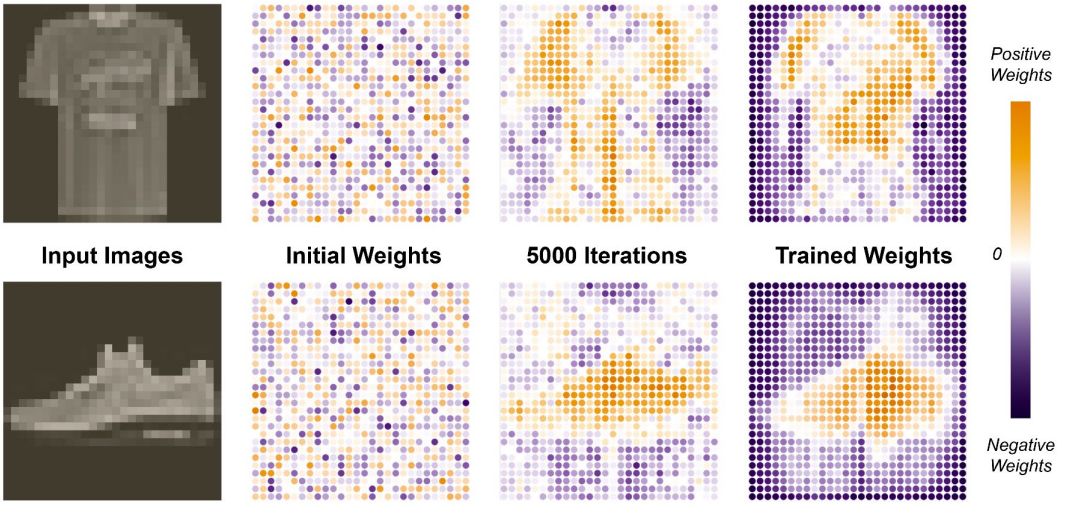

Figure.2 Inspiration for the Design of Ball Matrix

Based on the visualization I did in this paper; I introduced a similar design in this assignment. I created a ball matrix which consists 36 rows time 100 columns of ball objects. Each ball has a weight attribute in range from -1 to 1. Positive weights are displayed in orange, while negative ones are colored purple. Balls which have zero weights are invisible. The transparency of each blob is gradual, with the center being completely opaque and the edges being the most transparent, so that the center looks bright and the edges dark.

In order to create an effect with dynamics, I defined some weight matrices as the target weights. The weight matrices of the target are patterns and may contain text patterns. These weights are stored in the `matrixWeights.h` file and can be compiled into the source code by including it, which is a common practice in embedded systems development. After the target weights are defined, `weightUpdate` function is used to converge the weights to the target weights. This function is inspired from the PID control, while the target weight acts as the setpoint, and current weight acts as the feedback. The speed of convergence is random, so the balls in the matrix can be seen to form the target pattern with varying speeds.

Oscilloscope

2021 is the 15th anniversary of our university, here 15 is exactly $2^4 - 1$, in other words, numbers between 0 and 15 can be represented by 4-bit binary values, or a 4-bit counter. Figure 3 shows a timing diagram of a 4-bit counter.

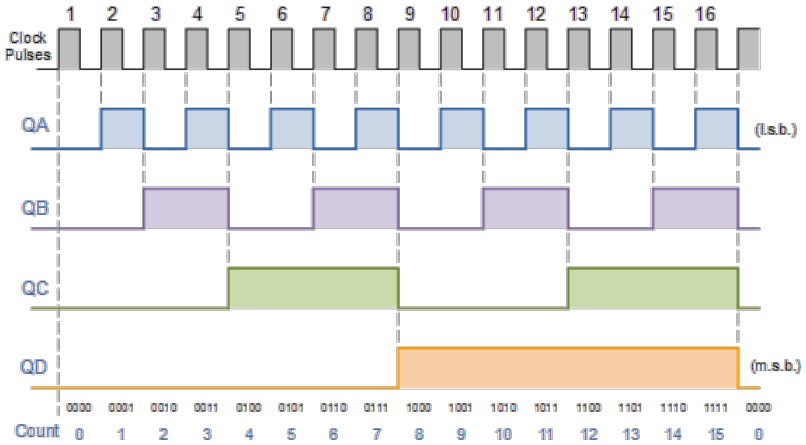

Figure.3 Timing Diagram of 4-Bit Counter

I created the oscilloscope object in order to display the waveform plots of the four channels. Each oscilloscope has a data buffer (length 500) to storage the history values. A circular queue was implemented to reach $O(1)$ time complexity when push back new value or pop a history value. Afterwards, the waveform is drawn by `GL_LINE_STRIP` method.

Square wave is discontinuous and looks too straight. To increase the dynamic, I use Fourier Series to approximate the square wave. Fourier series can represent almost any signal as a weighting sum of an infinite number of sinusoidal functions. Figure 4 shows this approximation.

In my approximation, the four channels use the same signal period, which is the length of time from the start of the program to the fireworks bloom. Since this period is very long, I chose $k = 160$, which means I added from the fundamental component to the 160th harmonic component.

Here is the end of this report.

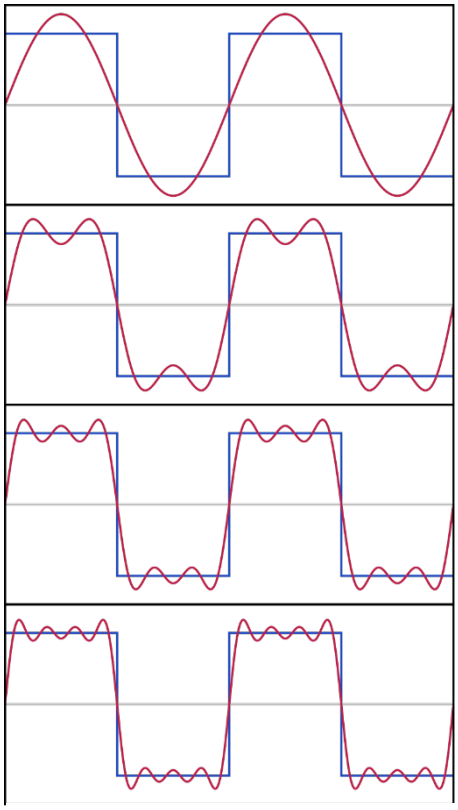

Figure 4. The first four partial sums of the Fourier series for a square wave