**Lab 1 Report**

Name: Bingjing Xu

Contact: [bingjing@kth.se](mailto:bingjing@kth.se)

* 1. **Set up the environment**

In order to set up the coding environment successfully, I firstly check the version of my operating system and Xcode.

My computer software version: OS X EI Capitan

My Xcode version: Version 7.3 (7D175)

Based on bellowed information, I followed a post on KTH social (https://www.kth.se/social/files/56fd013ff276544e9cfdccd9/SDL-Setup-Mac%2810.11.1%29-Xcode%287.0.1%29.pdf) which tells about how to set up in Mac with Xcode. It is a very detailed and hands-on introduction guiding you how to operate environment step-by-step. Since the steps are basically the same, I skipped it in my report.

After the set-up, run the project. You can see a blue window popping up which means your Xcode works well with SDL since now.

* 1. **Bilinear Interpolation of Colors**

Firstly, I tried to realize the function of linear interpolation. To do it, I wrote a function declaration as following:

**vec3 Interpolate(vec3 a, vec3 b, int i, vector<vec3>& result);**

When using the function, you have to provide color a and b which you want to interpolate, also you have to declare a vector to store the result of the interpolation for every pixel. Then, it will return the interpolated color of every pixel every time you use the function.

For details about implantation, the core code is as following:

**float DELTA\_RED = RED2 - RED1;**

**float DELTA\_GREEN = GREEN2 - GREEN1;**

**float DELTA\_BLUE = BLUE2 - BLUE1;**

**result[i].x = RED1 + (DELTA\_RED \* fraction);**

**result[i].y = GREEN1 + (DELTA\_GREEN \* fraction);**

**result[i].z = BLUE1 + (DELTA\_BLUE \* fraction);**

In principle, it is just the implementation of linear interpolation formula. And we did the interpolation correspondently to Red, Green and Blue.

Having this Interpolate function ready, then I can do bilinear interpolation in the for-loop in Draw():

**vec3 color1 = Interpolate(topLeft, topRight ,x , result\_x1);**

**vec3 color2=Interpolate(bottomRight, bottomLeft ,x ,result\_x2);**

**vec3 color = Interpolate(color1, color2, y, result);**

**PutPixelSDL( screen, x, y, color );**

In this segment of code, I tried to interpolate the color in x-axis two times and then in y-axis once. Next, use PutPixelSDL() function to draw the color of pixel.

And the result is as following:



* 1. **Starfield**

Following the instruction of Lab1, I wrote the initial\_draw( ) function to initialize static view of the star field. To see the stars showing on the screen averagely, I used the random number formula provided in the pdf to produce 1000 statistic points. However, I faced some difficulties when doing so. This formula can only produce float numbers between 0 and 1, according to the requirement of stars’ position, the range of x and y should be from -1(including -1) to +1(including +1). Thus we have to rewrite the function to produce random numbers which meets the requirement.

So we use:

**float random\_number1()**

**{**

**double m = (rand()%2?-1:1)\*((rand()%1)+(rand()%100)\*0.01);**

**return m;**

**}**

It is time to draw the stars on the screen! Again I used Draw( ) function but with some modification. Considering that we have to project t 3D objects on 2D screen, we use the transformation formula as following:

**for( size\_t s=0; s<stars.size(); ++s )**

**{**

**float f= float(SCREEN\_HEIGHT)/2;**

**float u= f\*(stars[s].x/stars[s].z)+SCREEN\_WIDTH/2;**

**float v= f\*(stars[s].y/stars[s].z)+SCREEN\_HEIGHT/2;**

**PutPixelSDL(screen, u, v, color );**

**}**

The only step left is to update the position of stars every fixed period. Through this step, the stars will start to move on the screen as animation effect.

**void Update()**

**{**

**int t2 = SDL\_GetTicks();**

**dt = float(t2-t);**

**t = t2;**

**for( int s=0; s<stars.size(); ++s )**

**{ // Add code for update of stars**

**float V = 0.00025;**

**stars[s].z= stars[s].z-V\*dt;**

**if( stars[s].z <= 0 )**

**stars[s].z += 1;**

**if( stars[s].z > 1 )**

**stars[s].z -= 1;**

**color = 0.2f \* vec3(1,1,1) / (stars[s].z\*stars[s].z);**

**}**

**}**

The idea is to update z-axis position every **dt** with a reasonable velocity. Here, pay a closer attention to V(velocity)! If the value of V is too small or too big, you just cannot see the proper animation effect. So be careful about V and try to estimate its value in advance.

The final step is to rewrite color variable which enhance the animation effect.

The result is as following:

****