Shiyu Fan G42039555

**Lab 5: Berlin Noise**

# Main missions

Generate a perlin noise and using it to create a perlin noise map.

# Development environment

Microsoft VS 2017(Visual C++)

OpenGL (GLUT 3.7)

# Implementation steps

1. Preprocessing: Consider the set of all points in the x, y, z space (coordinates are integers), which we call the integer grid. Now we attach a pseudo-random gradient value (x, y, z) on each point of the integer lattice, which is a vector, and process it into a unit vector.

The calculation method given by perlin is as follows:

For each vertex, a vector is randomly generated (such as calling rand() in the C library, and then the subscript is fixedly mapped to another subscript, and the vertex vector corresponding to the other subscript is taken. If it is 2D or 3D, this process will be repeated 2 times. The specific process can look at the following code.

1. If (x, y, z) is on an integer lattice, then the noise here is d.
2. If (x, y, z) is not on the integer lattice, we calculate the smooth interpolation coefficient.

The specific method is as follows. Taking 2D as an example, we need to find four integer points around this point. Then we get four values, the abscissa is bx0 ~ bx1, and the ordinate is by0 ~ by1. The distance from point to bx0 on the x-axis is rx0, and the distance from to0 on the y-axis is ry0. Perlin gives a easing curve that makes linear interpolation consistent.

A gradual curve function that enables the first derivative to be continuous (initial version): s\_curve(t) ( t \* t \* (3. - 2. \* t) )

1. A continuation curve function that enables the second derivative to be continuous (updated version): s\_curve(t) ( t \* t \* t \* (6 \* t \* t - 15 \* t + 10) ).Then pass rx0 and ry0 into the easing curve respectively to get a new value sx and sy.
2. Next, do a bilinear interpolation. sx and sy are the coefficients of the first linear interpolation, but after calculating the coefficients, we also need the starting and ending points of the interpolation. Their calculation methods are as follows:

(1) Find (rx0, ry0) and the point b00 of the upper left corner of the gradient b00 to obtain the starting point u, find (dx0 + 1, ry0) and the point b10 of the upper right corner of the gradient b10, get the end point v, with uv At both ends, sx is the interpolation coefficient, and linear interpolation is performed to obtain a

(2) Find (rx0, ry0 + 1) and the point b of the gradient b01 of the lower left corner to obtain the starting point u, and find the point multiplication of (bx0 + 1, ry0 + 1) and the gradient b11 of the lower right corner to obtain the end point v. , with uv as the two ends, sx as the interpolation coefficient, do linear interpolation, get b

(3) Finally, linear interpolation of a and b, the interpolation coefficient is sy, and the final result is obtained.

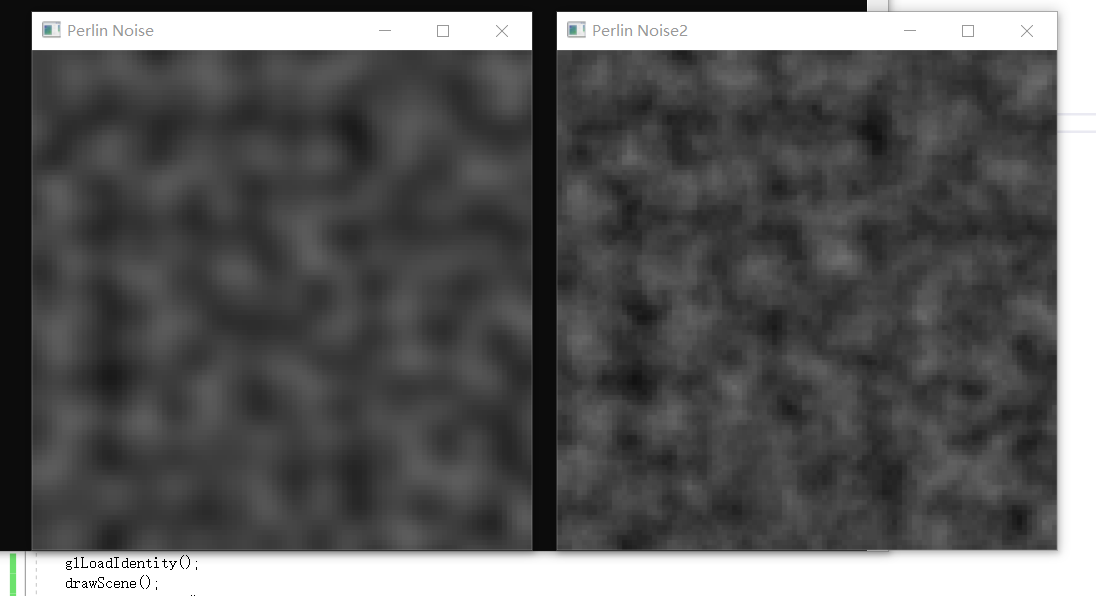
1. In the above algorithm steps, it is indifferent to interpolate the x-axis or the y-axis first.
2. The resulting Berlin noise is distributed between -1 and 1, and we can map it to the color difference we need (such as 0 ~ 255 or 0 ~ 1) to get the corresponding color. Build an edge table for a polygon. Sort all the lines in the polygon and find out the highest and lowest point. Store the edges and some vertices information in the initial edge table. These edge table should link to the bucket which represent the scanline number.

For each scanline from y=1 to y=max, calculate all the intersection point information ， and build up a new active edge table.

For each polygon，by using the edge table, get all the RGB color number and Z depth and store them in the Z-buffer.

Deal with all the front polygon by the same way. Always keep the nearest pixel color data in the Z-buffer.

# Output



# Future works

1. One problem is that i cannot larger than 4. If it is, the gray map will show up more white noise.
2. I am trying to apply this Perlin noise in Bump Mapping.