科目	计算机图形学
学号	16340294
姓名	张星
邮箱	dukestar@qq.com

Basic

题目一

实验要求

> 使用BresenHam算法(只使用integer arithmetic)画一个三角形边框：input为三个2D点；output三条直线（要求图元只能用 GL_POINTS ，不能使用其他，比如 GL_LINES 等）。

实验思路

- 主要函数：

```
void drawTriangle();

void bresenHamLine(int x0, int y0, int x1, int y1, int pointNum, int deltaX, int
deltaY);

void drawPoint(unsigned int VAO);
```

- *drawTriangle()*：

  该函数循环比较三个输入点，判断x1是否等于x2，然后进行分步处理。

  当x1==x2时，最为简单，无需调用bresenHamLine函数，然后将y1置为小的值，y2置为大的值。直接将y从y1累加到y2即可。当不等时，比较deltaX与deltaY，这样的目的是看斜率的绝对值是否大于1，以便确定在bresenHam算法中，以哪个轴为基准。比较之后，就可以算出总共需要多少次迭代，然后进行bresenHam算法运算。我在本次实验中，将每个x和y点分别存入了一个数组中，最后将两个数组整合，然后使用 *drawPoint()* 函数描点即可。

  - x1==x2：

```
if (x[i % 3] == x[(i + 1) % 3]) {
    //make sure the startPos is smaller than the endPos.
    int y0 = y[i % 3], y1 = y[(i + 1) % 3];
    if (y[i % 3] > y[(i + 1) % 3]) {
        y0 = y[(i + 1) % 3];
        y1 = y[i % 3];
    }
    pointNum = y1 - y0 + 1;
    xPos[0] = x[i % 3];
    yPos[0] = y0;
```

```
    for (int j = 1; j < pointNum; j++) {
        xPos[j] = xPos[j - 1];
        yPos[j] = yPos[j - 1] + 1;
    }
}
```

- x1!=x2:

```
int deltaX = x[(i + 1) % 3] - x[i % 3], deltaY = y[(i + 1) % 3] - y[i % 3];
if (abs(deltaX) > abs(deltaY)) {
    pointNum = abs(deltaX) + 1;
}
else {
    pointNum = abs(deltaY) + 1;
}
bresenHamLine(x[i % 3], y[i % 3], x[(i + 1) % 3], y[(i + 1) % 3], pointNum,
abs(deltaX), abs(deltaY));
```

- 整合输出：index为总共需要输出的点，需要作为glDrawArrays()的参数。

```
for (int j = 0; j < pointNum; j++) {
    Trivertices[index + j * 2] = (float)xPos[j] / width;
    Trivertices[index + j * 2 + 1] = (float)yPos[j] / height;
}
index += pointNum * 2;
```

- *bresenHamLine()*：

  在编写这个函数的时候，我遇到了一些问题。首先是按照老师课上所讲，假设deltaX大于deltaY，即斜率小于1，然后计算的时候，发现其他情况无法处理。接着又处理deltaY大于deltaX的情况，即斜率大于1，然后以y作为基准，进行逐步加。但是后面又发现，若是斜率为负数，因变量是随着自变量逐步递减的，所以后面又添加了变量，先判断斜率是否为负数，若是，则step为-1，否则为1，顺利完成。

```
if (x1 > x0) {
    stepX = 1;
}
else {
    stepX = -1;
}
if (y1 > y0) {
    stepY = 1;
}
else {
    stepY = -1;
}
//if the gradient is bigger than 1, gradient = 1
if (deltaY > deltaX) {
    p = 2 * deltaX - 2 * deltaY;
    gradient = 1;
}
else {
    p = 2 * deltaY - 2 * deltaX;
```

```
    gradient = 0;
}
```

- *drawPoint()*：

  在 *drawTriangle()* 中进行VAO和VBO的绑定，然后调用该函数即可。

```
unsigned int VAO, VBO;
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(Trivertices), Trivertices, GL_STATIC_DRAW);
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);

glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);

drawPoint(VAO);

glDeleteVertexArrays(1, &VAO);
glDeleteBuffers(1, &VBO);
```

```
void BresenHam::drawPoint(unsigned int VAO) {
    glBindVertexArray(VAO);
    glDrawArrays(GL_POINTS, 0, pixelNum);
}
```

题目二

实验要求

> 使用Bresenham算法(只使用integer arithmetic)画一个圆：input为一个2D点(圆心)、一个integer半径；output为一个圆。

实验思路

- 主要函数：

```
void drawCircle();
int bresenHamCircle(int, int, int);
void drawPoint(unsigned int);
```

- *drawCircle()*：

  调用 *bresenHamCircle()* 进行画圆，然后进行输出。

```
int pixelNum;
pixelNum = bresenHamCircle(centerX, centerY, r);
for (int i = 0; i < pixelNum; i++) {
    Circlevertices[i * 2] = xPos[i];
    Circlevertices[i * 2 + 1] = yPos[i];
}
this->pixelNum = pixelNum;
```

- *bresenHamCircle(int, int, int)*：

  本题一开始我看了ppt，没有什么思路，于是参考了网上的算法。由于圆的高度对称性，所以只需要计算八分之一圆弧即可，然后利用对称即可完成，在本次实验中，我挑选了(0,R)到(x,x)这一段圆弧进行结算。画圆的关键点在于x+1后，y时候要-1。具体衡量准则是：

  > $$x^2 + y^2 - R^2$$

  若该公式大于0，则说明点落在了圆弧外，则y需要减一，否则不变。

  但是上述公式还不够，因为BresenHam的特点是迭代性，所以需要找到一个变量d。在每次迭代前，需要计算下一个点是否会落在圆外，下一点设为(x+1, y-0.5)，因为根据我的计算，若直接使用(x+1, y)，则肯定会落在圆外，我们需要找的是贴近圆弧的一点，若该点套用公式：

  > $$(x+1)^2 + (y-0.5)^2 - R^2 > 0$$

  则说明圆弧离y-1更近，y需要-1，否则圆弧离y更近，不变即可。

  两种情况d的公式不同，按照公式迭代，化简即可。

```
float temp1 = (float)x1 / 500, temp2 = (float)y1 / 500;
xPos[index] = temp1 + x;
yPos[index++] = temp2 + y;
xPos[index] = x - temp1;
yPos[index++] = y - temp2;
xPos[index] = temp1 + x;
yPos[index++] = y - temp2;
xPos[index] = x - temp1;
yPos[index++] = temp2 + y;

xPos[index] = x + temp2;
yPos[index++] = temp1 + y;
xPos[index] = x - temp2;
yPos[index++] = y - temp1;
xPos[index] = temp2 + x;
yPos[index++] = y - temp1;
xPos[index] = x - temp2;
yPos[index++] = temp1 + y;
if (d < 0)
{
    d = d + 2 * x1 + 3;
}
else
{
    d = d + 2 * (x1 - y1) + 5;
    y1--;
}
```

```
x1++;
```

题目三

实验要求

> 在GUI在添加菜单栏，可以选择是三角形边框还是圆，以及能调整圆的大小(圆心固定即可)。

实验思路

- 主要函数：

```
void drawCircle(int);
int bresenHamCircle(int, int, int);
void drawPoint(unsigned int);
```

- *drawCircle(int)*：

  本题与上一题差别不大，函数重载了另一个函数，将传入参数作为圆的半径调用另一个函数画圆即可。

- *ImGui*：

  使用的API与上次作业没有差别，关于调整圆的半径，我没与找到相关的API，所以还是使用了上次的ColorEdit3，将其第一个参数作为圆的半径，但是由于颜色的最大值为255，所以需要按比例计算，输入值/255然后再乘半径。

```
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();

ImGui::Begin("Edit Radius", &ImGui, ImGuiWindowFlags_MenuBar);
ImGui::ColorEdit3("Radius", (float*)&TriangleColor);
ImGui::Checkbox("Triangle", &showTriangle);
ImGui::Checkbox("Circle", &showCircle);
ImGui::Checkbox("ChangeRadius", &changeRadius);
ImGui::Checkbox("fillTriangle", &fillTriangle);
ImGui::End();
glClearColor(1.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
if (showTriangle) {
    m.drawTriangle();
}
if (showCircle) {
    m.drawCircle();
    changeRadius = false;
}
if (changeRadius) {
    m.drawCircle((int)(TriangleColor.x*width/2));
    showCircle = false;
}
```

Bonus

实验要求

> 使用三角形光栅转换算法，用和背景不同的颜色，填充你的三角形。

实验思路

- 主要函数：

```
void fillTriangle();
void drawPoint(unsigned int);
```

- *fillTriangle()* ：

  使用了 **Edge Equations** 的方法，首先根据三个顶点计算出三个方程的A,B,C参数，然后再取三个顶点中最大的x,y以及最小的x,y，分别作为top,left,bottom,right，这样可以减少遍历的点数目。再将范围内的点带入方程，若都大于0，则说明在三角形内，渲染。由于此题涉及的点数过多，我不能以数组的方式存储，并且太浪费空间，所以每个点都调用一次 *drawPoint()* 函数。

```
top = (*max_element(y, y + 3));
bottom = (*min_element(y, y + 3));
left = (*min_element(x, x + 3));
right = (*max_element(x, x + 3));
for (int i = 0; i < 3; i++) {
    A[i] = y[(i + 1) % 3] - y[i % 3];
    B[i] = -(x[(i + 1) % 3] - x[i % 3]);
    C[i] = -(A[i] * x[i] + B[i] * y[i]);
}
for (int i = left; i <= right; i++) {
    for (int j = top; j >= bottom; j--) {
        bool in = true;
        for (int k = 0; k < 3; k++) {
            if (A[k] * i + B[k] * j + C[k] < 0) {
                in = false;
                break;
            }
        }
        if (in) {
            float arr[2] = { 0 };
            arr[0] = (float)i / width;
            arr[1] = (float)j / height;
            unsigned int VAO, VBO;
            glGenBuffers(1, &VBO);
            glBindBuffer(GL_ARRAY_BUFFER, VBO);
            glBufferData(GL_ARRAY_BUFFER, sizeof(arr), arr, GL_STATIC_DRAW);
            glGenVertexArrays(1, &VAO);
            glBindVertexArray(VAO);

            glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float),
(void*)0);
            glEnableVertexAttribArray(0);
```

```
            glBindBuffer(GL_ARRAY_BUFFER, 0);
            glBindVertexArray(0);

            drawPoint(VAO);
            glDeleteVertexArrays(1, &VAO);
            glDeleteBuffers(1, &VBO);
        }
    }
}
```

实验结果

见result.gif。

Tips：首先要输入三角形的所有顶点以及圆的圆心及半径，才能够进行下一步操作。