课程名称：计算机图形学　　指导教师：王振武

班级：计科 19-2　　姓名：王凌峰　　学号：1910630221

实验项目名称：

1. 画直线
2. 画圆
3. 多边形填充

实验目的及要求：

选一种方法

实验内容（方法和步骤）：

说明

这些 PDF 文件没有对应的 Word 版本，因为是用 LaTeX 写的。源文件在 `tex` 文件夹。

实验中使用 OpenGL core mode，版本要求 3.3 及以上。Context creation 使用 GLFW，function loading 使用 glbinding。不使用 Visual Studio。

1　画直线

用中点法。

实际上 OpenGL 的 rasterization 是不可编程的，只能先在 CPU 生成一系列坐标，作为 vertices 输入 GPU，以 point primitive 绘制来模拟 rasterization。实验 1 和实验 2 都是这么干的。本实验中 rasterization 的模拟放在 `line.cc` 中。

目录结构：

```
line/
|-- line.cc
|-- line.hh
|-- main.cc
|-- makefile
```

```
|-- shader.cc
|-- shader.hh
|-- trivial.frag
|-- trivial.vert
```

之后的实验中只有新增或修改文件时才贴出代码，目录结构中没有给出代码的文件默认与下面贴出的代码相同。文件名标在完整代码的下方。

由于使用 OpenGL core mode，shader 需要自己编写。`trivial.vert` 和 `trivial.frag` 分别为 vertex shader 和 fragment shader。`shader` 类的作用是读取、编译 shader，与 program 连接，并对错误做必要处理。

```
#ifndef SHADER_HH
#define SHADER_HH

#include <string>
#include <glbinding/gl/gl.h>
#include <glbinding/glbinding.h>

class shader
{
    public:
        gl::GLuint id;

        shader(const char *vs_path,const char *fs_path);
        void use() const;
        void setBool(const std::string &name, bool value) const;
        void setInt(const std::string &name, int value) const;
        void setFloat(const std::string &name, float value) const;
    private:
        std::string read_file(const char *path) const;
        gl::GLuint compile(const gl::GLchar * const source,gl::GLenum type)
        ↪  const;
};

inline void shader::use() const
{
    gl::glUseProgram(id);
}

inline void shader::setBool(const std::string &name, bool value) const
```

```
{
    gl::glUniform1i(gl::glGetUniformLocation(id, name.c_str()), (int)value);
}

inline void shader::setInt(const std::string &name, int value) const
{
    gl::glUniform1i(gl::glGetUniformLocation(id, name.c_str()), value);
}

inline void shader::setFloat(const std::string &name, float value) const
{
    gl::glUniform1f(gl::glGetUniformLocation(id, name.c_str()), value);
}

#endif //SHADER_HH
```

code 1: shader.hh

shader 类从文件读取 vertex shader 和 fragment shader：

```
#include "shader.hh"
#include <iostream>
#include <string>
#include <fstream>
#include <strstream>
#include <cstdlib>

using namespace std;
using namespace gl;

shader::shader(const char *vs_path,const char *fs_path)
{
    //read source and compile
    auto vertexSource = read_file(vs_path);
    GLuint vertexShader = compile(vertexSource.c_str(), GL_VERTEX_SHADER);

    auto fragmentSource = read_file(fs_path);
    GLuint fragmentShader = compile(fragmentSource.c_str(),
    ↪  GL_FRAGMENT_SHADER);
```

```
19
20      //program
21      id = glCreateProgram();
22      glAttachShader(id, vertexShader);
23      glAttachShader(id, fragmentShader);
24      //link
25      glLinkProgram(id);
26      GLboolean success(false);
27      glGetProgramiv(id, GL_LINK_STATUS, &success);
28      if (!success) {
29          char mess[512];
30          glGetProgramInfoLog(id, 512, nullptr, mess);
31          cerr << "link error" << endl << mess << endl;
32          exit(-1);
33      }
34
35      //delete shader objects
36      glDeleteShader(vertexShader);
37      glDeleteShader(fragmentShader);
38  }
39
40  string shader::read_file(const char *path) const
41  {
42      ifstream is(path);
43      if (!is) {
44          cerr << "cannot open file " << path << endl;
45          exit(-1);
46      }
47      ostrstream code;
48      code << is.rdbuf();
49      return code.str();
50  }
51
52  GLuint shader::compile(const GLchar * const source,gl::GLenum type) const
53  {
54      GLuint id = glCreateShader(type);
55      glShaderSource(id, 1, &source, nullptr);
56      glCompileShader(id);
57
```

```
58      GLboolean success(false);
59      glGetShaderiv(id,GL_COMPILE_STATUS,&success);
60      if (!success) {
61          char mess[512];
62          glGetShaderInfoLog(id, 512, nullptr, mess);
63          cerr << "compile error\n" << mess << endl;
64          exit(-1);
65      }
66
67      return id;
68  }
```

code 2: `shader.cc`

我设定窗口大小为 1920×1028，以屏幕左下角为原点。为了将 $[0, 1919] \times [0, 1027]$ 内的点变换到 $[-1, 1]^3$，先把输入点用 homogeneous coordinate 表示，再通过 orthographic projection transformation 将其变换到 NDC。这是一个简单的 windowing transform：

$$\mathbf{M}_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{l+r}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{b+t}{b-t} \\ 0 & 0 & \frac{2}{n-f} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

l, r, b, t, n, f 分别为左、右、底、顶、近、远处在各个方向的坐标。这个实验不涉及 z 轴，n 和 f 的取值是无所谓的。为了方便分别取 0 和 -1。

$\mathbf{M}_{orth}$ 不会变化，所以直接编码在 vertex shader 内：

```
1   #version 330 core
2   layout (location = 0) in vec2 aPos;
3   const int windowWidth = 1920;
4   const int windowHeight = 1028;
5   const mat4 M_ortho_proj = mat4(
6       2.0f/(windowWidth-1),0.0f,0.0f,0.0f,
7       0.0f,2.0f/(windowHeight-1),0.0f,0.0f,
8       0.0f,0.0f,1.0f,0.0f,
9       -1.0f,-1.0f,-1.0f,1.0f
10  );
11  void main()
12  {
```

```
    gl_Position = M_ortho_proj * vec4(aPos,0.0f,1.0f);
}
```

code 3: `trivial.vert`

fragment shader 直接返回黑色：

```
#version 330 core
out vec4 color;
void main()
{
    color = vec4(0.0f,0.0f,0.0f,1.0f);
}
```

code 4: `trivial.frag`

顶点数据，即模拟 rasterization 得到的点向 GPU 的输入使用现代的 VAO 和 VBO，每个 VAO 关联一组 vertices，代表一条测试直线的 rasterization 的结果。测试覆盖了直线斜率的各种情况，共 6 条。

`main.cc` 中先进行窗口初始化和函数绑定，然后编译、连接得到 program。调用 `line_rasterizer` 得到测试线 rasterization 结果，在 GPU 中创建 buffer 并存入。在主循环中清除 color buffer，依次绑定每个 VAO 并绘制与 VAO 绑定的 VBO 中的点：

```
#include "shader.hh"
#include "line.hh"

#include <iostream>
#include <cstdlib>

#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
#include <glbinding/gl/gl.h>
#include <glbinding/glbinding.h>

using namespace std;
using namespace gl;

GLFWwindow *initWindow();
```

```
16
17 int main()
18 {
19     auto w = initWindow();
20     glbinding::initialize(glfwGetProcAddress);
21
22     shader prog("trivial.vert","trivial.frag");
23
24     size_t cnt[6];
25     GLint *lines[] = {
26         line_rasterizer(960,0,960,1028,cnt[0]),
27         line_rasterizer(0,514,1920,514,cnt[1]),
28         line_rasterizer(960,514,1200,1028,cnt[2]),
29         line_rasterizer(1920,800,960,514,cnt[3]),
30         line_rasterizer(1920,230,960,514,cnt[4]),
31         line_rasterizer(1200,0,960,514,cnt[5])
32     };
33
34     GLuint vao[6];
35     glGenVertexArrays(6, vao);
36     GLuint vbo[6];
37     glGenBuffers(6,vbo);
38
39     for (size_t i(0); i != 6; ++i) {
40         glBindVertexArray(vao[i]);
41         glBindBuffer(GL_ARRAY_BUFFER, vbo[i]);
42         glBufferData(GL_ARRAY_BUFFER, sizeof(GL_INT)*cnt[i]*2, lines[i],
            ↪  GL_STATIC_DRAW);
43
44         glVertexAttribPointer(0, 2, GL_INT, GL_FALSE,
            ↪  2*sizeof(GL_INT),static_cast<void *>(0));
45         glEnableVertexAttribArray(0);
46
47         delete lines[i];
48     }
49
50     glBindBuffer(GL_ARRAY_BUFFER, 0);
51     glBindVertexArray(0);
52
```

```
53      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
54      while(!glfwWindowShouldClose(w))
55      {
56          if (glfwGetKey(w, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
57              glfwSetWindowShouldClose(w, true);
58          }
59
60          glClearColor(0.7, 0.7, 0.7, 1);
61          glClear(GL_COLOR_BUFFER_BIT);
62
63          prog.use();
64
65          for (size_t i(0); i != 6; ++i) {
66              //bind to different sets of points of lines
67              glBindVertexArray(vao[i]);
68              //draw as points as to simulate rasterization
69              glDrawArrays(GL_POINTS, 0, cnt[i]);
70          }
71
72          glfwSwapBuffers(w);
73          glfwPollEvents();
74      }
75
76      glfwTerminate();
77  }
78  GLFWwindow *initWindow()
79  {
80      if (!glfwInit()) {
81          std::cerr << "init failed." << std::endl;
82      }
83      glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
84      glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
85      glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
86
87      auto w = glfwCreateWindow(1920, 1028, "tAsK", nullptr, nullptr);
88      if (!w) {
89          std::cerr << "window creation failed" << std::endl;
90          exit(-1);
91      }
```

```
92      glfwMakeContextCurrent(w);
93
94      return w;
95  }
```

code 5: main.cc

下面是 line.hh:

```
1   #ifndef LINE_HH
2   #define LINE_HH
3
4   #include <glbinding/gl/gl.h>
5   #include <glbinding/glbinding.h>
6
7   void mid_point(int ref_beg,int ref_end,int beg,int end,gl::GLint *points,bool
    ↪   asc,bool reversed);
8   gl::GLint *line_rasterizer(int x0,int y0,int x1,int y1,size_t &cnt, void
    ↪   f(int,int,int,int,gl::GLint*,bool,bool) = mid_point);
9   void swapp(int &a,int &b);
10
11  #endif //LINE_HH
```

code 6: line.hh

line_rasterizer 接受一个函数指针，可以传入 mid-point 或 bresenham 方法。

当 $0 < k \leq 1$ 不满足时，或可以直接给出结果 ($k = 0$ 或 $k \to \infty$)，或可以转换为 $0 < k \leq 1$ 的情形。

$k = 0$ 或 $k \to \infty$ 的情形是非常简单的。line_rasterizer 中的代码直接处理这个平凡的情形。

当 $|k| > 1$ 时，需要互换 x, y。为了结果正确，以 reversed 标记存入结果时是否需要互换坐标位置。

为了处理方便，称其中一条坐标轴为参考轴，规定处理过程中点在参考轴上的坐标只能增加。另一条坐标轴称为非参考轴，点在非参考轴的坐标变化没有限制。输入确定直线的两点后，以参考轴坐标较小的点为起点，每次将参考轴坐标增 1，计算应有的非参考轴坐标，直到参考轴坐标超过参考轴坐标较大的点的参考轴坐标。

所以需要知道非参考轴坐标如何计算。容易看出，

- 若 $k < 0$，非参考轴坐标随参考轴坐标增加而**减小或不变**
- 若 $k > 0$，非参考轴坐标随参考轴坐标增加而**增加或不变**

因此可根据 k 判断非参考轴的增减性，这个性质用 `asc` 标记。增减性不同时 d 的初始值和增量也不同，对于 $Ax + By + C = 0$ 形式的直线，

- 若 `asc==true`: $\Delta d = A + B(d < 0) \quad \text{or} \quad \Delta d = A(d \geq 0)$
- 若 `asc==false`: $\Delta d = A(d < 0) \quad \text{or} \quad \Delta d = A - B(d \geq 0)$

`line_rasterizer` 中判断 k 的值，给 `asc` 和 `reversed` 赋合适的值，传入 mid-point 或 bresenham 方法的函数。

具体处理 $0 < k < 1$ 情形的函数则根据这些参数决定 Δd 以及存入点坐标的次序。

```
#include "line.hh"
using namespace gl;

GLint *line_rasterizer(int x0,int y0,int x1,int y1,size_t &cnt, void
 ↪  f(int,int,int,int,GLint*,bool,bool))
{
    if (x0 == x1) {
        if (y1 < y0)
            swapp(y0, y1);
        cnt = y1 - y0 + 1;
        auto points = new GLint[2*cnt];
        size_t i(0);
        for (int y(y0); y <= y1; ++y) {
            points[i++] = x0;
            points[i++] = y;
        }
        return points;
    }
    else if (x0 > x1) {
        swapp(x0,x1);
        swapp(y0,y1);
    }

    bool asc(true);
    if (y1 - y0 < 0) {
        asc = false;
    }
```

```
27
28      GLint *points(nullptr);
29      if (abs(y1 - y0) <= x1 - x0) {
30          cnt = x1 - x0 + 1;
31          points = new GLint[2*cnt];
32          f(x0, x1, y0, y1, points, asc, false);
33      }
34      else {
35          cnt = abs(y1 - y0) + 1;
36          points = new GLint[2*cnt];
37          if (asc)
38              f(y0, y1, x0, x1, points, asc, true);
39          else
40              f(y1, y0, x1, x0, points, asc, true);
41      }
42      return points;
43  }
44  void swapp(int &a,int &b)
45  {
46      int tem(a);
47      a = b;
48      b = tem;
49  }
50  void mid_point(int ref_beg,int ref_end,int beg,int end,GLint *points,bool
    ↪  asc,bool reversed)
51  {
52      int A(beg - end);
53      int B(ref_end - ref_beg);
54      int C(ref_beg * end - beg * ref_end);
55      float d;
56      if (asc) {
57          d = A * (ref_beg + 1) + B * (beg + 0.5) + C;
58      }
59      else {
60          d = A * (ref_beg + 1) + B * (beg - 0.5) + C;
61      }
62
63      size_t i(0);
64      while (ref_beg <= ref_end) {
```

```
65          if (!reversed) {
66              points[i++] = ref_beg++;
67              points[i++] = beg;
68          }
69          else {
70              points[i++] = beg;
71              points[i++] = ref_beg++;
72          }
73          if (d < 0) {
74              if (asc) {
75                  ++beg;
76                  d += A + B;
77              }
78              else {
79                  d += A;
80              }
81          }
82          else {
83              if (asc)
84                  d += A;
85              else {
86                  --beg;
87                  d += A - B;
88              }
89          }
90      }
91  }
```

code 7: `line.cc`

2 画圆

用中点法。

与上个实验相同，绘制点模拟 rasterization。

目录结构：

```
circle/
|-- circle.cc
```

```
|-- circle.hh
|-- main.cc
|-- makefile
|-- shader.cc
|-- shader.hh
|-- trivial.frag
|-- trivial.vert
```

rasterization 代码在 `circle.cc` 中。

`main.cc` 和实验一类似，测试了 6 个圆。

```
#include "shader.hh"
#include "circle.hh"

#include <iostream>
#include <cstdlib>

#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
#include <glbinding/gl/gl.h>
#include <glbinding/glbinding.h>

using namespace std;
using namespace gl;

GLFWwindow *initWindow();

int main()
{
    auto w = initWindow();
    glbinding::initialize(glfwGetProcAddress);

    shader prog("trivial.vert","trivial.frag");

    size_t cnt[6];
    GLint *circles[] = {
        circle_rasterizer_mid_point(0, 0, 90, cnt[0]),
        circle_rasterizer_mid_point(0, 0, 160, cnt[1]),
        circle_rasterizer_mid_point(0, 0, 260, cnt[2]),
        circle_rasterizer_mid_point(1920, 1028, 400, cnt[3]),
```

```
30          circle_rasterizer_mid_point(600, 800, 200, cnt[4]),
31          circle_rasterizer_mid_point(960, 514, 300, cnt[5])
32      };
33
34      GLuint vao[6];
35      glGenVertexArrays(6, vao);
36      GLuint vbo[6];
37      glGenBuffers(6,vbo);
38
39      for (size_t i(0); i != 6; ++i) {
40          glBindVertexArray(vao[i]);
41          glBindBuffer(GL_ARRAY_BUFFER, vbo[i]);
42          glBufferData(GL_ARRAY_BUFFER, sizeof(GL_INT)*cnt[i]*2, circles[i],
            ↪  GL_STATIC_DRAW);
43
44          glVertexAttribPointer(0, 2, GL_INT, GL_FALSE,
            ↪  2*sizeof(GL_INT),static_cast<void *>(0));
45          glEnableVertexAttribArray(0);
46
47          delete circles[i];
48      }
49
50      glBindBuffer(GL_ARRAY_BUFFER, 0);
51      glBindVertexArray(0);
52
53      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
54      while(!glfwWindowShouldClose(w))
55      {
56          if (glfwGetKey(w, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
57              glfwSetWindowShouldClose(w, true);
58          }
59
60          glClearColor(0.7, 0.7, 0.7, 1);
61          glClear(GL_COLOR_BUFFER_BIT);
62
63          prog.use();
64
65          for (size_t i(0); i != 6; ++i) {
66              //bind to different sets of points of circles
```

```
            glBindVertexArray(vao[i]);
            //draw as points as to simulate rasterization
            glDrawArrays(GL_POINTS, 0, cnt[i]);
        }

        glfwSwapBuffers(w);
        glfwPollEvents();
    }

    glfwTerminate();
}
GLFWwindow *initWindow()
{
    if (!glfwInit()) {
        std::cerr << "init failed." << std::endl;
    }
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

    auto w = glfwCreateWindow(1920, 1028, "tAsK", nullptr, nullptr);
    if (!w) {
        std::cerr << "window creation failed" << std::endl;
        exit(-1);
    }
    glfwMakeContextCurrent(w);

    return w;
}
```

code 8: main.cc

```
#ifndef CIRCLE_HH
#define CIRCLE_HH

#include <glbinding/gl/gl.h>
#include <glbinding/glbinding.h>

gl::GLint *circle_rasterizer_mid_point(int x,int y,int r,size_t &cnt);
```

```
8
9  #endif //CIRCLE_HH
```

code 9: `circle.hh`

课本中从 $(0,R)$ 开始顺时针计算 $1/8$ 个圆，再对这部分点对称变换得到剩余部分。但 y 轴上和直线 $y=x$ 上的点沿自身所在直线对称变换后得到自身，导致冗余。所以这里对点 $(0,R)$ 单独对称变换，而 $1/8$ 圆的计算从 $(1,R)$ 或 $(1,R-1)$ 开始，到 $x \geq y$ 为止。$1/8$ 圆计算结束时若 $x=y$，还要将直线 $y=x$ 上的点沿 x 轴和 y 轴对称。

因此一个圆需要的点的数量为 $(\lfloor R/\sqrt{2} \rfloor * 8 + 4)$ 或 $(\lfloor R/\sqrt{2} \rfloor * 8 + 8)$。

对于圆心 $\vec{o}$ 不在 $(0,0)$ 的圆，计算出的任意一点位置向量加上 $\vec{o}$ 即可。

```cpp
#include "circle.hh"
#include <cmath>
using namespace gl;

GLint *circle_rasterizer_mid_point(int x0,int y0,int r,size_t &cnt)
{
#define ADD_POINT(X,Y) points[i++]=X;points[i++]=Y;
    cnt = static_cast<int>(r / pow(2, 0.5)) * 8 + 4;
    //reserve for the possible extra 4 points
    auto points = new GLint[2*(cnt+4)];

    size_t i(0);
    //when points fall on axes
    ADD_POINT(x0,y0 + r);
    ADD_POINT(x0 + r,y0);
    ADD_POINT(x0,y0 - r);
    ADD_POINT(x0 - r,y0);

    float d(4.25f-r);
    GLint x(1),y((d<=0)? r : r-1);
    //when points fall between axes
    while (x < y) {
        //original point
        ADD_POINT(x + x0,y + y0);
        //reflect along y = x
        ADD_POINT(y + x0,x + y0);
```

```
27          //along x axis
28          ADD_POINT(x + x0,-y + y0);
29          ADD_POINT(y + x0,-x + y0);
30          //along y axis
31          ADD_POINT(-x + x0,y + y0);
32          ADD_POINT(-y + x0,x + y0);
33          ADD_POINT(-x + x0,-y + y0);
34          ADD_POINT(-y + x0,-x + y0);
35          if (d <= 0) {
36              d += 2 * x + 3;
37              ++x;
38          }
39          else {
40              d += 2 * (x - y) + 5;
41              ++x;
42              --y;
43          }
44      }
45      //when the last point to calc. happens to fall on y = x
46      if (x == y) {
47          cnt += 4;
48          ADD_POINT(x + x0,y + y0);
49          ADD_POINT(x + x0,-y + y0);
50          ADD_POINT(-x + x0,y + y0);
51          ADD_POINT(-x + x0,-y + y0);
52      }
53      return points;
54  }
```

code 10: `circle.cc`

3 多边形填充

用 4-邻接的种子填充法。

目录结构：

```
filling/
|-- fill.cc
```

```
|-- fill.hh
|-- line.cc
|-- line.hh
|-- main.cc
|-- makefile
|-- point.cc
|-- point.hh
|-- shader.cc
|-- shader.hh
|-- trivial.frag
|-- trivial.vert
```

为了方便，用 point 类表示点。

```
#ifndef POINT_HH
#define POINT_HH

#include <glbinding/gl/gl.h>
#include <glbinding/glbinding.h>

class point
{
    public:
        gl::GLint x;
        gl::GLint y;
        point(int xx,int yy):x(xx),y(yy) {}
        bool operator==(const point &another) const {
            return x == another.x && y == another.y;
        }
};
bool lessf(const point &p1,const point &p2);

#endif //POINT_HH
```

code 11: point.hh

```
#include "point.hh"
bool lessf(const point &p1,const point &p2) {
```

```
    if (p1.x < p2.x) {
        return true;
    }
    else if (p1.x == p2.x) {
        return p1.y < p2.y;
    }
    return false;
}
```

code 12: point.cc

main.cc 中测试了 4 边形的填充:

```
#include "shader.hh"
#include "point.hh"
#include "fill.hh"

#include <iostream>

#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
#include <glbinding/gl/gl.h>
#include <glbinding/glbinding.h>

using namespace std;
using namespace gl;

GLFWwindow *initWindow();

int main()
{
    auto w = initWindow();
    glbinding::initialize(glfwGetProcAddress);

    shader prog("trivial.vert","trivial.frag");

    size_t cnt;
    const auto poly = fill_poly({
            point(840,600),
```

```
27              point(900,540),
28              point(840,700),
29              point(740,540)
30      },cnt);
31
32      GLuint vao;
33      glGenVertexArrays(1, &vao);
34      GLuint vbo;
35      glGenBuffers(1,&vbo);
36
37      glBindVertexArray(vao);
38      glBindBuffer(GL_ARRAY_BUFFER, vbo);
39      glBufferData(GL_ARRAY_BUFFER, sizeof(GL_INT)*cnt*2, poly,
        ↪  GL_STATIC_DRAW);
40
41      glVertexAttribPointer(0, 2, GL_INT, GL_FALSE,
        ↪  2*sizeof(GL_INT),static_cast<void *>(0));
42      glEnableVertexAttribArray(0);
43      delete [] poly;
44
45      glBindBuffer(GL_ARRAY_BUFFER, 0);
46      glBindVertexArray(0);
47
48      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
49      while(!glfwWindowShouldClose(w))
50      {
51          if (glfwGetKey(w, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
52              glfwSetWindowShouldClose(w, true);
53          }
54
55          glClearColor(0.7, 0.7, 0.7, 1);
56          glClear(GL_COLOR_BUFFER_BIT);
57
58          prog.use();
59
60          glBindVertexArray(vao);
61          //draw as points as to simulate rasterization
62          glDrawArrays(GL_POINTS, 0, cnt);
63
```

```
64          glfwSwapBuffers(w);
65          glfwPollEvents();
66      }
67
68      glfwTerminate();
69  }
70  GLFWwindow *initWindow()
71  {
72      if (!glfwInit()) {
73          std::cerr << "init failed." << std::endl;
74      }
75      glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
76      glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
77      glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
78
79      auto w = glfwCreateWindow(1920, 1028, "tAsK", nullptr, nullptr);
80      if (!w) {
81          std::cerr << "window creation failed" << std::endl;
82          exit(-1);
83      }
84      glfwMakeContextCurrent(w);
85
86      return w;
87  }
```

code 13: main.cc

```
1  #ifndef FILL_HH
2  #define FILL_HH
3
4  #include "line.hh"
5  #include "point.hh"
6
7  #include <cstdlib>
8  #include <iostream>
9  #include <set>
10 #include <deque>
11 #include <initializer_list>
12 #include <algorithm>
```

```
13
14 #include <glbinding/gl/gl.h>
15 #include <glbinding/glbinding.h>
16
17 gl::GLint *fill_poly(std::initializer_list<point> il,size_t &cnt);
18 void add_edge_points(const std::initializer_list<point>::iterator &it1, const
   ↪  std::initializer_list<point>::iterator
   ↪  &it2,std::set<point,decltype(lessf)*> &points);
19
20 #endif //FILL_HH
```

code 14: `fill.hh`

`fill_poly` 中点存在 `std::set` 中。首先把多边形每个点根据顺序两两配对，调用 `add_edge_points` 把这两点所表示的直线 rasterize。

填充起始点 $(x_0, y_0) = \frac{1}{n}\sum_{i=1}^{n}(x_i, y_i)$，$n$ 为多边形顶点个数。这保证了起始点在多边形内部，但只保证对凸多边形有效。

在 `std::set` 中的点表示已填充的点。填充过程中出栈一个点，把每个不在 `std::set` 中且不在栈中的这个点的 4-邻接点入栈，直到栈空就完成了填充。

```
1  #include "fill.hh"
2
3  using namespace gl;
4  using namespace std;
5
6  GLint *fill_poly(initializer_list<point> il,size_t &cnt)
7  {
8      if (il.size() <= 2) {
9          std::cerr << "insufficient number of points for polygon" << std::endl;
10         exit(-1);
11     }
12     set<point,decltype(lessf)*> points(lessf);
13
14     //add edge points & find a point inside
15     point start_p(il.begin()->x,il.begin()->y);
16     points.insert(point(il.begin()->x,il.begin()->y));
17     for (auto it = il.begin() + 1; it != il.end(); ++it) {
18         start_p.x += it->x;
```

```
19          start_p.y += it->y;
20          add_edge_points(it,it-1,points);
21          points.insert(point(it->x,it->y));
22      }
23      add_edge_points(il.begin(), il.end()-1, points);
24
25      start_p.x /= il.size();
26      start_p.y /= il.size();
27
28      //filling
29      deque<point> stack(1,start_p);
30      while (!stack.empty()) {
31          auto p = stack.front();
32          stack.pop_front();
33          auto retval = points.insert(p);
34          if (retval.second) {
35              //4-adjacency, counter-clockwise
36              if (find(stack.begin(), stack.end(), p) == stack.end())
37                  stack.push_front(point(p.x+1,p.y));
38              if (find(stack.begin(), stack.end(), p) == stack.end())
39                  stack.push_front(point(p.x,p.y+1));
40              if (find(stack.begin(), stack.end(), p) == stack.end())
41                  stack.push_front(point(p.x-1,p.y));
42              if (find(stack.begin(), stack.end(), p) == stack.end())
43                  stack.push_front(point(p.x,p.y-1));
44          }
45      }
46
47      //conversion to array
48      cnt = points.size();
49      auto arr = new GLint[2*cnt];
50      size_t i(0);
51      for (const auto &p : points) {
52          arr[i++] = p.x;
53          arr[i++] = p.y;
54      }
55      return arr;
56  }
```

```cpp
void add_edge_points(const initializer_list<point>::iterator &it1, const
    initializer_list<point>::iterator &it2,set<point,decltype(lessf)*>
    &points)
{
    size_t cnt;
    auto line = line_rasterizer(it1->x, it1->y, it2->x, it2->y, cnt);
    for (size_t i = 0; i < 2*cnt-2; i+=2) {
        points.insert(point(line[i],line[i+1]));
    }
    delete [] line;
}
```

code 15: `fill.cc`

实验结果与分析:

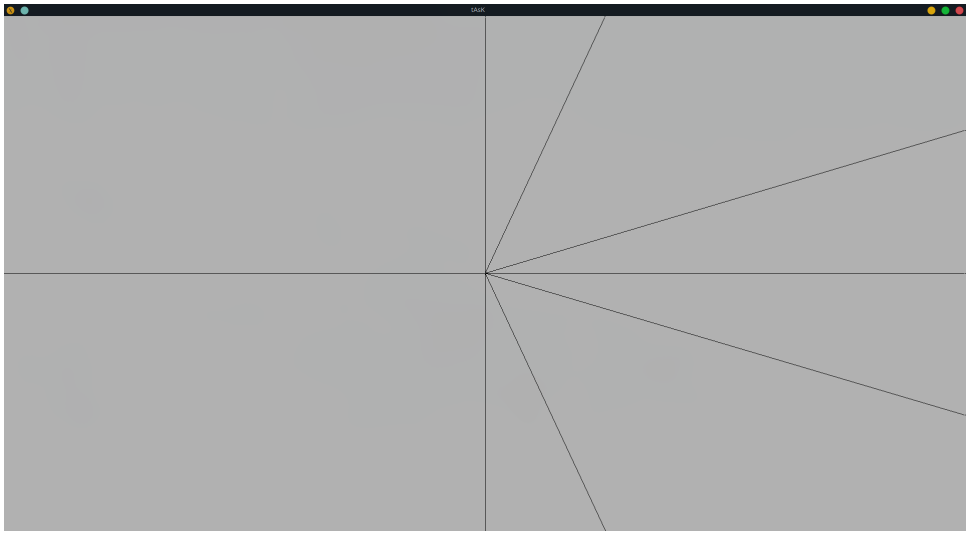

图 1: 画直线

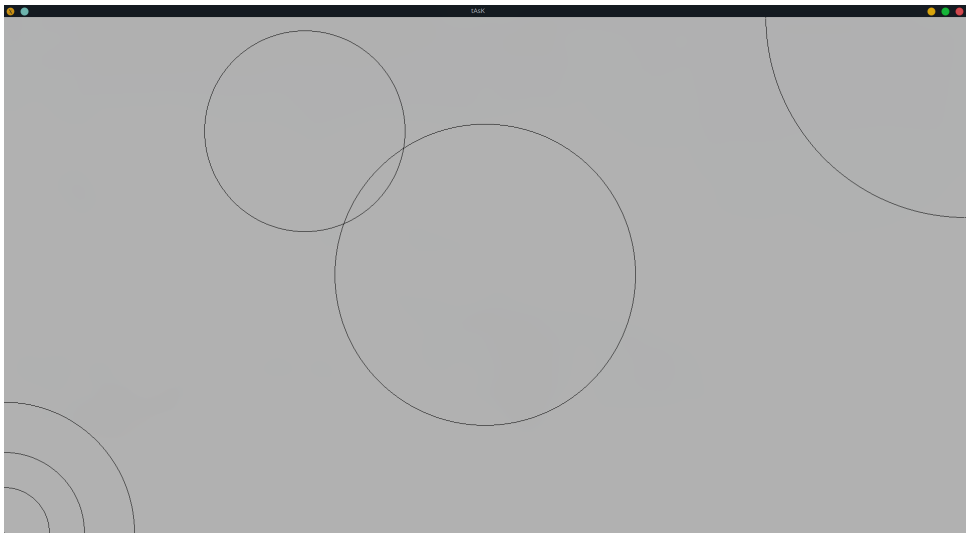

图 2: 画圆

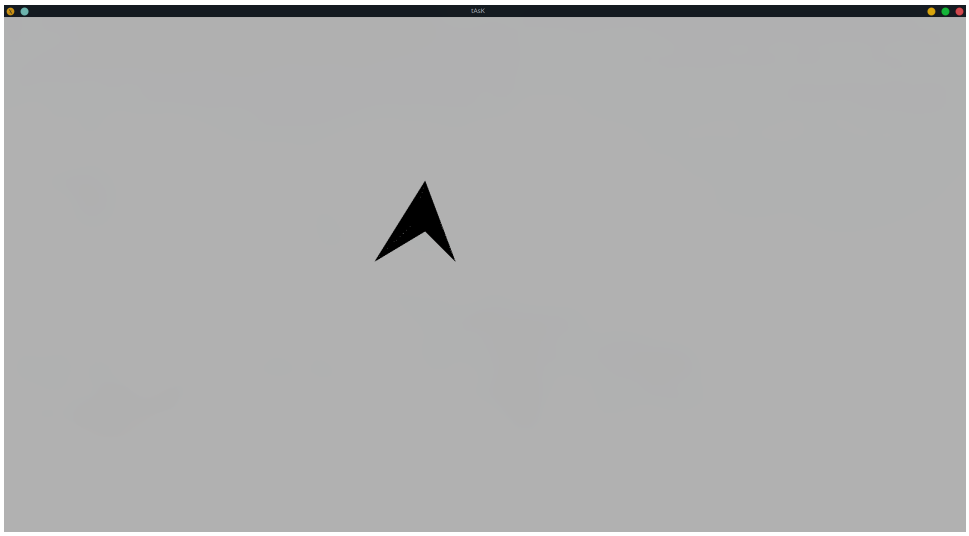

图 3: 填充

成绩:　　**批阅教师签名**:　　**年　　月　　日**