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：

4. 二维几何变换
5. 裁剪

实验目的及要求：

选一种方法

实验内容（方法和步骤）：

4　二维几何变换

把一个等腰三角形缩小为 0.3 倍并沿一条垂线上的中点旋转 $-45°$。

目录结构：

```
trans2d/
|-- main.cc
|-- makefile
|-- shader.cc
|-- shader.hh
|-- trivial.frag
|-- trivial.vert
```

矩阵变换由 GPU 完成比较合适，因此在 `trivial.vert` 中声明了 **`uniform mat4`** 类型的变量 `M`，表示转换矩阵。`main.cc` 中构造出 `M`，再通过 `gl::glUniformMatrix4fv` 设置 vertex shader 中 `M` 的值。

为了展示转换矩阵的效果，`trivial.vert` 中还声明了 **`uniform bool`** 类型的变量 `should`。`should` 为 `false` 时不乘转换矩阵。

```
#version 330 core
layout (location = 0) in vec3 aPos;

```

```
uniform mat4 M;
uniform bool should;

const int windowWidth = 1920;
const int windowHeight = 1028;
const mat4 M_ortho_proj = mat4(
    2.0f/(windowWidth-1),0.0f,0.0f,0.0f,
    0.0f,2.0f/(windowHeight-1),0.0f,0.0f,
    0.0f,0.0f,1.0f,0.0f,
    -1.0f,-1.0f,-1.0f,1.0f
);

void main()
{
    if (should)
        gl_Position = M * vec4(aPos,1.0f);
    else
        gl_Position = vec4(aPos,1.0f);
    gl_Position = M_ortho_proj * gl_Position;
}
```

code 1: `trivial.vert`

`main.cc` 中使用 glm 库完成矩阵和向量运算。在主循环里分别设置 `should` 为 `true` 和 `false` 绘制原三角形和变换后的三角形。

```
#include "shader.hh"
#include <iostream>
#include <cmath>
//disable inclusion of the development environment header
#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
//glbinding
#include <glbinding/gl/gl.h>
#include <glbinding/glbinding.h>
//glm
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
```

```
14
15 using namespace std;
16 using namespace gl;
17
18 GLFWwindow *initWindow();
19
20 const unsigned windowWidth(1920);
21 const unsigned windowHeight(1028);
22
23 int main()
24 {
25     auto w = initWindow();
26     glbinding::initialize(glfwGetProcAddress);
27
28     shader prog("trivial.vert","trivial.frag");
29
30     const GLfloat vert[] = {
31         400,300,0,
32         1520,300,0,
33         960,800,0
34     };
35
36     GLuint vao;
37     glGenVertexArrays(1, &vao);
38     glBindVertexArray(vao);
39
40     GLuint vbo;
41     glGenBuffers(1,&vbo);
42     glBindBuffer(GL_ARRAY_BUFFER, vbo);
43     glBufferData(GL_ARRAY_BUFFER, sizeof(vert), vert, GL_STATIC_DRAW);
44
45     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3*sizeof(GL_FLOAT),(void
       ↪  *)0);
46     glEnableVertexAttribArray(0);
47
48     //transformation to be tested
49     const auto x_shift = (vert[0] + vert[3])/2;
50     const auto y_shift = (vert[1] + vert[7])/2;
51     const float translation[] = {
```

```
        1.0f,0.0f,0.0f,0.0f,
        0.0f,1.0f,0.0f,0.0f,
        0.0f,0.0f,1.0f,0.0f,
        -x_shift,-y_shift,0.0f,1.0f
    };
    auto M = glm::make_mat4(translation);

    const float angle = glm::radians(-45.0);
    const float rotation[] = {
        cos(angle),sin(angle),0.0f,0.0f,
        -sin(angle),cos(angle),0.0f,0.0f,
        0.0f,0.0f,1.0f,0.0f,
        0.0f,0.0f,0.0f,1.0f
    };
    M = glm::make_mat4(rotation) * M;

    const float x_scale(0.3);
    const float y_scale(0.3);
    const float scaling[] = {
        x_scale,0.0f,0.0f,0.0f,
        0.0f,y_scale,0.0f,0.0f,
        0.0f,0.0f,1.0f,0.0f,
        0.0f,0.0f,0.0f,1.0f
    };
    M = glm::make_mat4(scaling) * M;

    const float reverse_translation[] = {
        1.0f,0.0f,0.0f,0.0f,
        0.0f,1.0f,0.0f,0.0f,
        0.0f,0.0f,1.0f,0.0f,
        x_shift,y_shift,0.0f,1.0f
    };
    M = glm::make_mat4(reverse_translation) * M;

    //updating uniform requires using program first
    prog.use();
    prog.setMat4("M", M);

    //unbind
```

```
91      glBindBuffer(GL_ARRAY_BUFFER, 0);
92      glBindVertexArray(0);
93
94      glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
95      while(!glfwWindowShouldClose(w))
96      {
97          if (glfwGetKey(w, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
98              glfwSetWindowShouldClose(w, true);
99          }
100
101         glClearColor(0.5, 0.5, 0.5, 1);
102         glClear(GL_COLOR_BUFFER_BIT);
103
104         prog.use();
105
106         //orig. triangle
107         prog.setBool("should", false);
108         glBindVertexArray(vao);
109         glDrawArrays(GL_TRIANGLES, 0, 3);
110
111         //triangle transformed by M
112         prog.setBool("should", true);
113         glBindVertexArray(vao);
114         glDrawArrays(GL_TRIANGLES, 0, 3);
115
116         glfwSwapBuffers(w);
117         glfwPollEvents();
118     }
119
120     glDeleteVertexArrays(1, &vao);
121     glDeleteBuffers(1, &vbo);
122
123     glfwTerminate();
124 }
125 GLFWwindow *initWindow()
126 {
127     if (!glfwInit()) {
128         std::cerr << "init failed." << std::endl;
129     }
```

```
130     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
131     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
132     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
133 
134     auto w = glfwCreateWindow(windowWidth, windowHeight, "tAsK", nullptr,
        ↪  nullptr);
135     if (!w) {
136         std::cerr << "window creation failed" << std::endl;
137         exit(-1);
138     }
139     glfwMakeContextCurrent(w);
140 
141     return w;
142 }
```

code 2: `main.cc`

5 裁剪

使用 Liang-Barsky 算法。

目录结构：

```
clipping
|-- clip.cc
|-- clip.hh
|-- main.cc
|-- makefile
|-- point.cc
|-- point.hh
|-- shader.cc
|-- shader.hh
|-- tree
|-- trivial.frag
|-- trivial.vert
```

`main.cc` 中画了表示窗口的矩形和一条直线的裁剪结果。

```
1 #include "shader.hh"
```

```cpp
#include "clip.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");

    const GLint rect[] = {
        600,200,
        1200,200,
        1200,800,
        600,800
    };
    const GLuint indices[] = {
        0,1,1,2,2,3,3,0
    };
    const point p1(300,300),p2(900,900);

    GLuint vao[2];
    glGenVertexArrays(2, vao);
    glBindVertexArray(vao[0]);

    GLuint vbo[2];
    glGenBuffers(2,vbo);
```

```
41      //rectangle
42      glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
43      glBufferData(GL_ARRAY_BUFFER, sizeof(rect), rect, GL_STATIC_DRAW);
44
45      GLuint ebo;
46      glGenBuffers(1, &ebo);
47      glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo);
48      glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
        ↪  GL_STATIC_DRAW);
49
50      glVertexAttribPointer(0, 2, GL_INT, GL_FALSE, 2*sizeof(GL_INT),(void *)0);
51      glEnableVertexAttribArray(0);
52      //clipping
53      const auto a = clip(point(rect[0],rect[1]), point(rect[4],rect[5]), p1,
        ↪  p2);
54      bool empty(a.size() == 0);
55      if (!empty) {
56          const GLint points[] = {
57              a[0].x,a[0].y,
58              a[1].x,a[1].y
59          };
60          glBindVertexArray(vao[1]);
61          glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
62          glBufferData(GL_ARRAY_BUFFER, sizeof(points), points,
            ↪  GL_STATIC_DRAW);
63          glVertexAttribPointer(0, 2, GL_INT, GL_FALSE, 2*sizeof(GL_INT),(void
            ↪  *)0);
64          glEnableVertexAttribArray(0);
65      }
66
67      //unbind
68      glBindBuffer(GL_ARRAY_BUFFER, 0);
69      glBindVertexArray(0);
70
71      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
72      while(!glfwWindowShouldClose(w))
73      {
74          if (glfwGetKey(w, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
75              glfwSetWindowShouldClose(w, true);
```

```
76          }
77
78          glClearColor(0.7, 0.7, 0.7, 1);
79          glClear(GL_COLOR_BUFFER_BIT);
80
81          prog.use();
82
83          glBindVertexArray(vao[0]);
84          glDrawElements(GL_LINES, 8, GL_UNSIGNED_INT, 0);
85
86          if (!empty) {
87              glBindVertexArray(vao[1]);
88              glDrawArrays(GL_LINES, 0, 2);
89          }
90
91          glfwSwapBuffers(w);
92          glfwPollEvents();
93      }
94
95      glfwTerminate();
96  }
97  GLFWwindow *initWindow()
98  {
99      if (!glfwInit()) {
100         std::cerr << "init failed." << std::endl;
101     }
102     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
103     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
104     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
105
106     auto w = glfwCreateWindow(1920, 1028, "tAsK", nullptr, nullptr);
107     if (!w) {
108         std::cerr << "window creation failed" << std::endl;
109         exit(-1);
110     }
111     glfwMakeContextCurrent(w);
112
113     return w;
114 }
```

code 3: main.cc

主要代码在 clip.cc 中。

```
#ifndef CLIP_HH
#define CLIP_HH

#include <vector>
#include "point.hh"

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

std::vector<point> clip(const point &lb,const point &rt,const point &p1,const point &p2);

#endif //CLIP_HH
```

code 4: clip.hh

```
#include "clip.hh"

using namespace std;
using namespace gl;

vector<point> clip(const point &lb,const point &rt,const point &p1,const point &p2)
{
    const auto dx = p2.x - p1.x;
    const auto dy = p2.y - p1.y;
    const GLint Ps[] = {
        -dx,dx,-dy,dy
    };
    const GLint Qs[] = {
        p1.x - lb.x,
        rt.x - p1.x,
        p1.y - lb.y,
        rt.y - p1.y
    };
```

```
19      if (Ps[0] == 0 || Ps[2] == 0) {
20          for (const auto q : Qs) {
21              if (q >= 0) {
22                  if (Ps[0] == 0) {
23                      const point p1_(Ps[0],lb.y);
24                      const point p2_(Ps[0],rt.y);
25                      return {p1_,p2_};
26                  }
27                  else {
28                      const point p1_(lb.x,Ps[2]);
29                      const point p2_(rt.x,Ps[2]);
30                      return {p1_,p2_};
31                  }
32              }
33              else {
34                  return {};
35              }
36          }
37      }
38      GLfloat u1(0),u2(1);
39      for (size_t i(0); i != 4; ++i) {
40          GLfloat temp;
41          if (Ps[i] < 0 && (temp = static_cast<GLfloat>(Qs[i]) / Ps[i]) > u1) {
42              u1 = temp;
43          }
44          else if (Ps[i] > 0 && (temp = static_cast<GLfloat>(Qs[i]) / Ps[i]) <
            ↪  u2) {
45              u2 = temp;
46          }
47      }
48      if (u1 >= u2) {
49          return {};
50      }
51      const point p1_(p1.x + dx * u1,p1.y + dy * u1);
52      const point p2_(p1.x + dx * u2,p1.y + dy * u2);
53      return {p1_,p2_};
54  }
```

code 5: clip.cc

实验结果与分析:

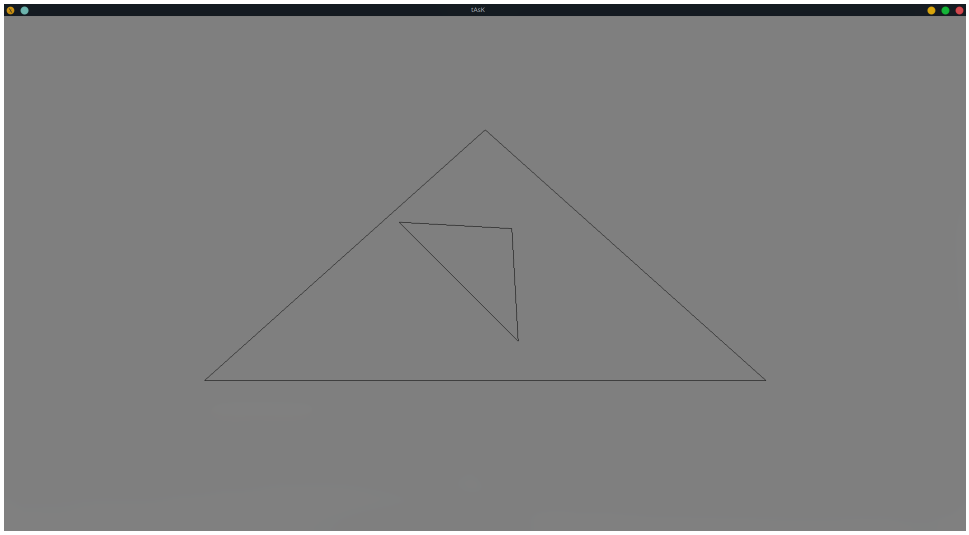

图 1: 二维几何变换

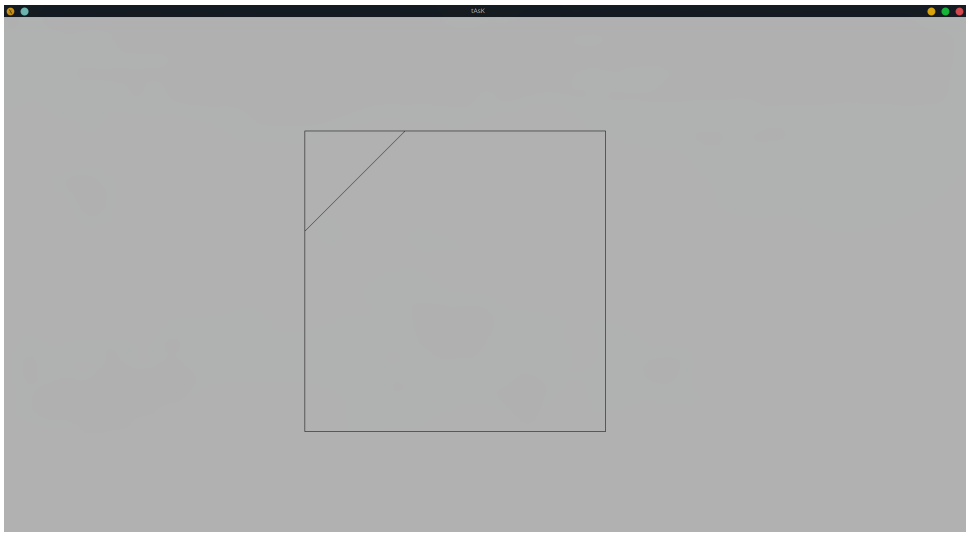

图 2: 二维裁剪

成绩: **批阅教师签名**: **年 月 日**