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：

6. 三维几何变换

实验目的及要求：

选一种方法

实验内容（方法和步骤）：

把一个长方体的 x, y 坐标沿原点缩小为 0.3 倍并沿 z 轴旋转 $-45°$。

这个实验中比较有趣的地方在于增加了 camera transformation $\mathbf{M}_{cam}$，camera coordinate 围绕长方体中心旋转，运行中可以在各个方位观察长方体及三维变换后的长方体的形态。

目录结构：

```
trans3d/
|-- main.cc
|-- makefile
|-- shader.cc
|-- shader.hh
|-- trivial.frag
|-- trivial.vert
```

camera transformation 在运行中不断变化，因此在 `trivial.vert` 中声明 camera transformation matrix **`uniform mat4 M_cam`**，点的位置向量在 orthographic transformation 之前先进行 camera transformation 以得到正确的视角。

另外为了方便，orthographic transformation $\mathbf{M}_{orth}$ 也变化了，新的 camera space 为

$$\left[-\frac{windowWidth-1}{2}, \frac{windowWidth-1}{2}\right] \times \left[-\frac{windowHeight-1}{2}, \frac{windowHeight-1}{2}\right] \times [-2000, 0]$$

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

uniform mat4 M;
```

```
uniform bool should;
uniform mat4 M_cam;

const int windowWidth = 1920;
const int windowHeight = 1028;
const float depth = 2000;
const float l = - (windowWidth - 1) / 2.0f;
const float r = (windowWidth - 1) / 2.0f;
const float b = - (windowHeight - 1) / 2.0f;
const float t = (windowHeight - 1) / 2.0f;
const float n = 0;
const float f = -depth;
const mat4 M_ortho_proj = mat4(
    vec4(2.0f/(r - l),0.0f,0.0f,0.0f),
    vec4(0.0f,2.0f/(t - b),0.0f,0.0f),
    vec4(0.0f,0.0f,2.0f/(n - f),0.0f),
    vec4((r + l)/(l - r),(t + b)/(b - t),(n + f)/(f - n),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 * M_cam * gl_Position;
}
```

code 1: `trivial.vert`

`main.cc` 中先使用 `gen_M` 得到变换矩阵 `M`，设置 vertex shader 中 `M` 为 `M`。在主循环中每次循环调用 `gen_M_cam` 得到新的 camera transformation matrix `M_cam`，使视角不断变化。

长方体只要 8 个顶点表示，但 OpenGL 中没有 primitive 绘制长方体和矩形，需要绘制多个直线或三角形组成长方体。这个实验选择直线连接各个顶点，顶点对在 Element Array Buffer(EBO) 中存放。绘制直线时，调用 `glDrawElements`。

为了方便，原始长方体中心在 $\vec{o} = (0, 0, 0)$，占据空间 $[-300, 300] \times [-250, 250] \times [-600, -100]$。camera coordinate frame 的原点 $\vec{e}$ 以半径 500 绕 $(0, 200, 0)$ 垂直于 y 轴旋转，视线始终朝向长方体中心，即 $\vec{o}$。旋转中心的 y 坐标不在长方体的 y 中点是为了观察到立体的效果。

canonical coordiante frame 记为 $\langle \vec{o}, (\vec{x}, \vec{y}, \vec{z}) \rangle$，camera coordiante frame 记为 $\langle \vec{e}, (\vec{u}, \vec{v}, \vec{w}) \rangle$。依照惯例，坐标系都是右手系，camera coordiante frame 视线方向为 $-\vec{w}$。

要将 canonical coordiantes 用 camera coordiantes 表示，需要对位置向量左乘 canonical-to-basis matrix:

$$\mathbf{M}_{cam} = \begin{bmatrix} \vec{u} & \vec{v} & \vec{w} & \vec{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

视点 $\vec{e} = (0, 200, 0)$。确定 view-up vector $\vec{t}$ 后，$(\vec{u}, \vec{v}, \vec{w})$ 可以由观察方向 $\vec{g}$ 和 $\vec{t}$ 构造：

$$\begin{aligned} \vec{w} &= -\frac{\vec{g}}{\|\vec{g}\|}, \\ \vec{u} &= \frac{\vec{t} \times \vec{w}}{\|\vec{t} \times \vec{w}\|}, \\ \vec{v} &= \vec{w} \times \vec{u}. \end{aligned}$$

三维几何变换矩阵 $\mathbf{M}$ 是简单的矩阵，和二维几何变换实验中的矩阵相同。在三维中的效果是 x, y 坐标沿长方体中心缩小为 0.3 倍，再沿 z 轴旋转 $-45°$。

这些都在 `main.cc` 中：

```
#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>

using namespace std;
using namespace gl;

GLFWwindow *initWindow();
glm::mat4 gen_M(const glm::vec3 &center);
```

```
20  glm::mat4 gen_M_cam(const glm::vec3 &center);
21
22  const unsigned windowWidth(1920);
23  const unsigned windowHeight(1028);
24
25  int main()
26  {
27      auto wd = initWindow();
28      glbinding::initialize(glfwGetProcAddress);
29      glEnable(GL_DEPTH_TEST);
30
31      shader prog("trivial.vert","trivial.frag");
32
33      const GLfloat
        ↪  left(-300),right(300),bottom(-250),top(250),near(-100),far(-600);
34      const GLfloat vert[] = {
35          left,bottom,near,
36          right,bottom,near,
37          right,top,near,
38          left,top,near,
39
40          left,bottom,far,
41          right,bottom,far,
42          right,top,far,
43          left,top,far,
44      };
45      auto center = glm::vec3((left+right)/2,(top+bottom)/2,(near+far)/2);
46      const GLuint indices[] = {
47          0,1,1,2,2,3,3,0,    //near plane
48          4,5,5,6,6,7,7,4,    //far plane
49          0,4,7,3,    //left plane
50          1,5,6,2        //right plane
51      };
52
53      GLuint vao;
54      glGenVertexArrays(1, &vao);
55      glBindVertexArray(vao);
56
57      GLuint vbo;
```

```
58      glGenBuffers(1,&vbo);
59      glBindBuffer(GL_ARRAY_BUFFER, vbo);
60      glBufferData(GL_ARRAY_BUFFER, sizeof(vert), vert, GL_STATIC_DRAW);
61
62      GLuint ebo;
63      glGenBuffers(1, &ebo);
64      glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo);
65      glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
        ↪  GL_STATIC_DRAW);
66
67      glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3*sizeof(GL_FLOAT),(void
        ↪  *)0);
68      glEnableVertexAttribArray(0);
69
70      //generate transformation matrix to be tested
71      const auto M = gen_M(center);
72
73      //updating uniform requires using program first
74      prog.use();
75      prog.setMat4("M", M);
76
77      //unbind
78      glBindBuffer(GL_ARRAY_BUFFER, 0);
79      glBindVertexArray(0);
80
81      glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
82      while(!glfwWindowShouldClose(wd))
83      {
84          if (glfwGetKey(wd, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
85              glfwSetWindowShouldClose(wd, true);
86          }
87
88          glClearColor(0.5, 0.5, 0.5, 1);
89          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
90
91          //get new camera transformation
92          const auto M_cam = gen_M_cam(center);
93
94          prog.use();
```

```
95          prog.setMat4("M_cam", M_cam);
96
97          //orig. cuboid
98          prog.setBool("should", false);
99          glBindVertexArray(vao);
100         //draw lines according to indices of points
101         glDrawElements(GL_LINES, 24, GL_UNSIGNED_INT, 0);
102
103         //cuboid transformed by M
104         prog.setBool("should", true);
105         glBindVertexArray(vao);
106         glDrawElements(GL_LINES, 24, GL_UNSIGNED_INT, 0);
107
108         glfwSwapBuffers(wd);
109         glfwPollEvents();
110     }
111
112     glDeleteVertexArrays(1, &vao);
113     glDeleteBuffers(1, &vbo);
114
115     glfwTerminate();
116 }
117 glm::mat4 gen_M(const glm::vec3 &center)
118 {
119     //scale x & y coord. down to 0.3 then rotate -45.0 deg. along z axis
120     const auto x_shift = center.x;
121     const auto y_shift = center.y;
122     const float translation[] = {
123         1.0f,0.0f,0.0f,0.0f,
124         0.0f,1.0f,0.0f,0.0f,
125         0.0f,0.0f,1.0f,0.0f,
126         -x_shift,-y_shift,0.0f,1.0f
127     };
128     auto M = glm::make_mat4(translation);
129
130     const float angle = glm::radians(-45.0);
131     const float rotation[] = {
132         cos(angle),sin(angle),0.0f,0.0f,
133         -sin(angle),cos(angle),0.0f,0.0f,
```

```
134         0.0f,0.0f,1.0f,0.0f,
135         0.0f,0.0f,0.0f,1.0f
136     };
137     M = glm::make_mat4(rotation) * M;
138 
139     const float x_scale(0.3);
140     const float y_scale(0.3);
141     const float scaling[] = {
142         x_scale,0.0f,0.0f,0.0f,
143         0.0f,y_scale,0.0f,0.0f,
144         0.0f,0.0f,1.0f,0.0f,
145         0.0f,0.0f,0.0f,1.0f
146     };
147     M = glm::make_mat4(scaling) * M;
148 
149     const float reverse_translation[] = {
150         1.0f,0.0f,0.0f,0.0f,
151         0.0f,1.0f,0.0f,0.0f,
152         0.0f,0.0f,1.0f,0.0f,
153         x_shift,y_shift,0.0f,1.0f
154     };
155     return glm::make_mat4(reverse_translation) * M;
156 }
157 glm::mat4 gen_M_cam(const glm::vec3 &center)
158 {
159     //eye position
160     const float radius = 500;
161     const auto time = glfwGetTime();
162     //eye rotates along a circle whose plane is perpendicular to y axis
163     const auto e = glm::vec3(
164         center.x + radius*sin(time),
165         center.y + 200,
166         center.z + radius*cos(time)
167     );
168     //gaze direction
169     const auto target = center;
170     const auto g = target - e;    //always gaze at the center of the cuboid
171     //view-up vector
172     const auto t = glm::vec3(0,1,0);
```

```
173     //construct (u⃗,v⃗,w⃗) basis
174     const auto w = - glm::normalize(g);
175     const auto u = glm::normalize(glm::cross(t, w));
176     const auto v = glm::cross(w, u);
177     //matrix to align (u⃗,v⃗,w⃗) to (x⃗,y⃗,z⃗)
178     glm::mat4 M_cam_rot(
179         glm::vec4(u,0),    //first col
180         glm::vec4(v,0),    //second col
181         glm::vec4(w,0),    //third col
182         glm::vec4(0,0,0,1)    //translation col
183     );
184     M_cam_rot = glm::transpose(M_cam_rot);
185     //matrix to translate e⃗ to o⃗
186     const glm::mat4 M_cam_transl(
187         glm::vec4(1,0,0,0),
188         glm::vec4(0,1,0,0),
189         glm::vec4(0,0,1,0),
190         glm::vec4(-e,1)
191     );
192     //construct M_cam
193     return M_cam_rot * M_cam_transl;
194 }
195 GLFWwindow *initWindow()
196 {
197     if (!glfwInit()) {
198         std::cerr << "init failed." << std::endl;
199     }
200     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
201     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
202     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
203 
204     auto w = glfwCreateWindow(windowWidth, windowHeight, "tAsK", nullptr,
      ↪  nullptr);
205     if (!w) {
206         std::cerr << "window creation failed" << std::endl;
207         exit(-1);
208     }
209     glfwMakeContextCurrent(w);
210 
```

```
211     return w;
212 }
```

code 2: `main.cc`

实验结果与分析:

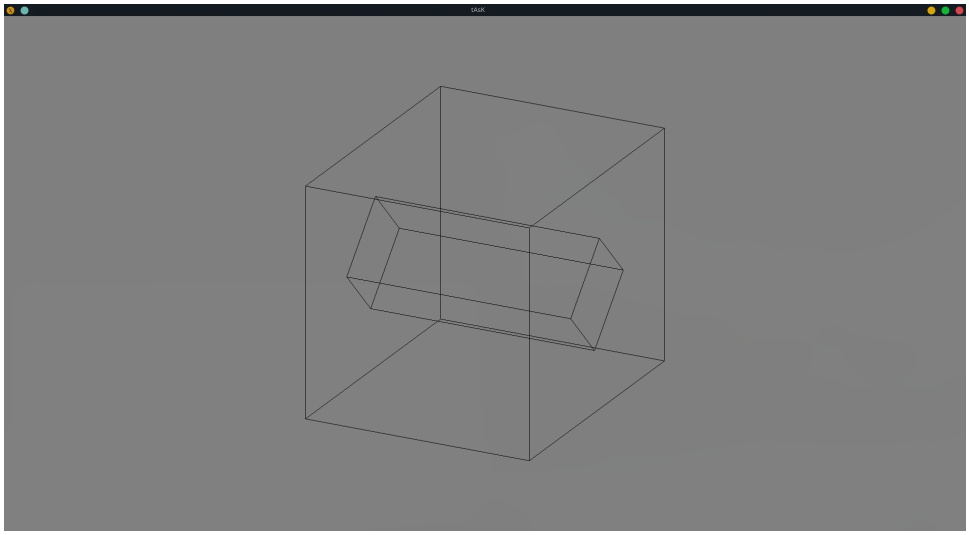

图 1: 其中一个视角

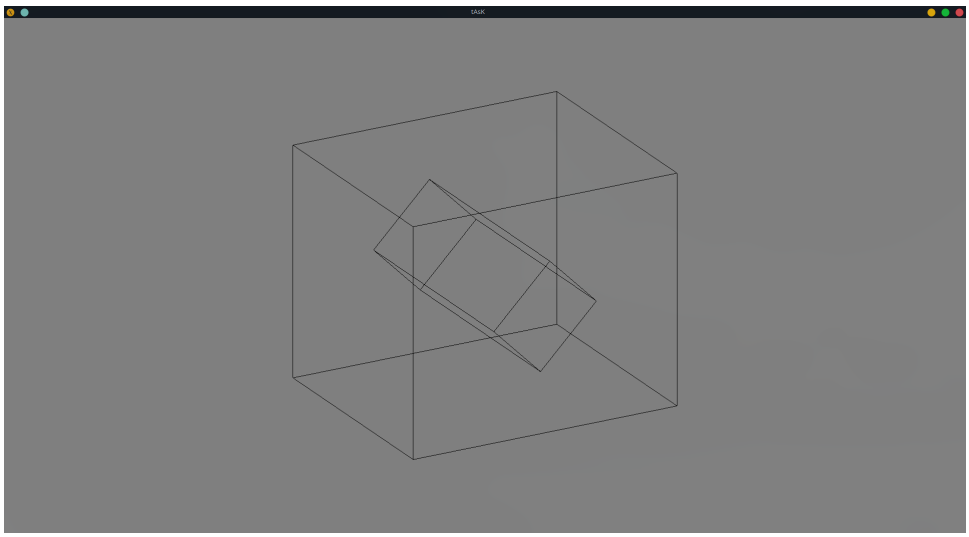

图 2: 另一个视角

成绩: **批阅教师签名**: **年 月 日**