Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №2
«Модельно-видовые и проективные преобразования»
по курсу: «Алгоритмы компьютерной графики»

Выполнил:
Студент группы ИУ9-31Б
Гречко Г.В.

Проверил:
Цалкович П.А.

Москва, 2023

Цели

Получение навыков совершения модельно-видовых преобразований на примере выполнения линейного преобразования с кубом.

Задачи

1. Определить куб в качестве модели объекта сцены.
2. Определить преобразования, позволяющие получить трехточечную перспективу. Для демонстрации проекции добавить в сцену куб (в стандартной ориентации, не изменяемой при модельно-видовых преобразованиях основного объекта).
3. Реализовать изменение ориентации и размеров объекта (навигацию камеры) с помощью модельно-видовых преобразований (без gluLookAt). Управление производится интерактивно с помощью клавиатуры и/или мыши.
4. Предусмотреть возможность переключения между каркасным и твердотельным отображением модели (glFrontFace/ glPolygonMode)

Основная теория

Однородные координаты

Это система координат, используемая в проективной геометрии, подобно тому, как декартовы координаты используются в евклидовой геометрии. Однородные координаты обладают тем свойством, что определяемый ими объект не меняется при умножении всех координат на одно и то же ненулевое число.

По большому счёту, однородные координаты нужны с единственной целью - чтобы при получении экранных координат точки не нужно было различать ортогональную и перспективную проекции.

Линейное преобразование (линейный оператор)

Это отображение векторного пространства в само себя, удовлетворяющее свойству линейности.

Матричное представление линейного оператора

- Координаты представляются вектором-столбцом
- Геометрическое преобразование задается матрицей, умножаемой справа на вектор-столбец координат
- Матрица композиции преобразований является произведением матриц элементарных преобразований (матрицы перемножаются в обратном порядке): операция является ассоциативной, но в общем случае некоммутативной

Виды преобразований

- общие линейные преобразования
  - w’ != 1 (проективные, необходимо «перспективное деление»)
  - прямые переходят в прямые
- аффинные преобразования
  - w’ = 1
  - сохраняется параллельность линий
  - пример: сдвиг
- преобразование подобия
  - сохраняются углы
  - пример: равномерное масштабирование
- изометрия (движение)

- сохраняются расстояния
- пример: поворот, перенос

Центральные (перспективные) проекции

Центральные проекции параллельных прямых, не параллельных плоскости проекции будут сходиться в точке схода, количество которых (для прямых, параллельных осям координат), зависит от числа координатных осей, которые пересекает плоскость проекции

Матрица центральной проекции

$$[T] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & r & 1 \end{pmatrix}$$

$$p = -1/x_c$$

$$q = -1/y_c$$

Где $r = -1/z_c$

Практическая реализация

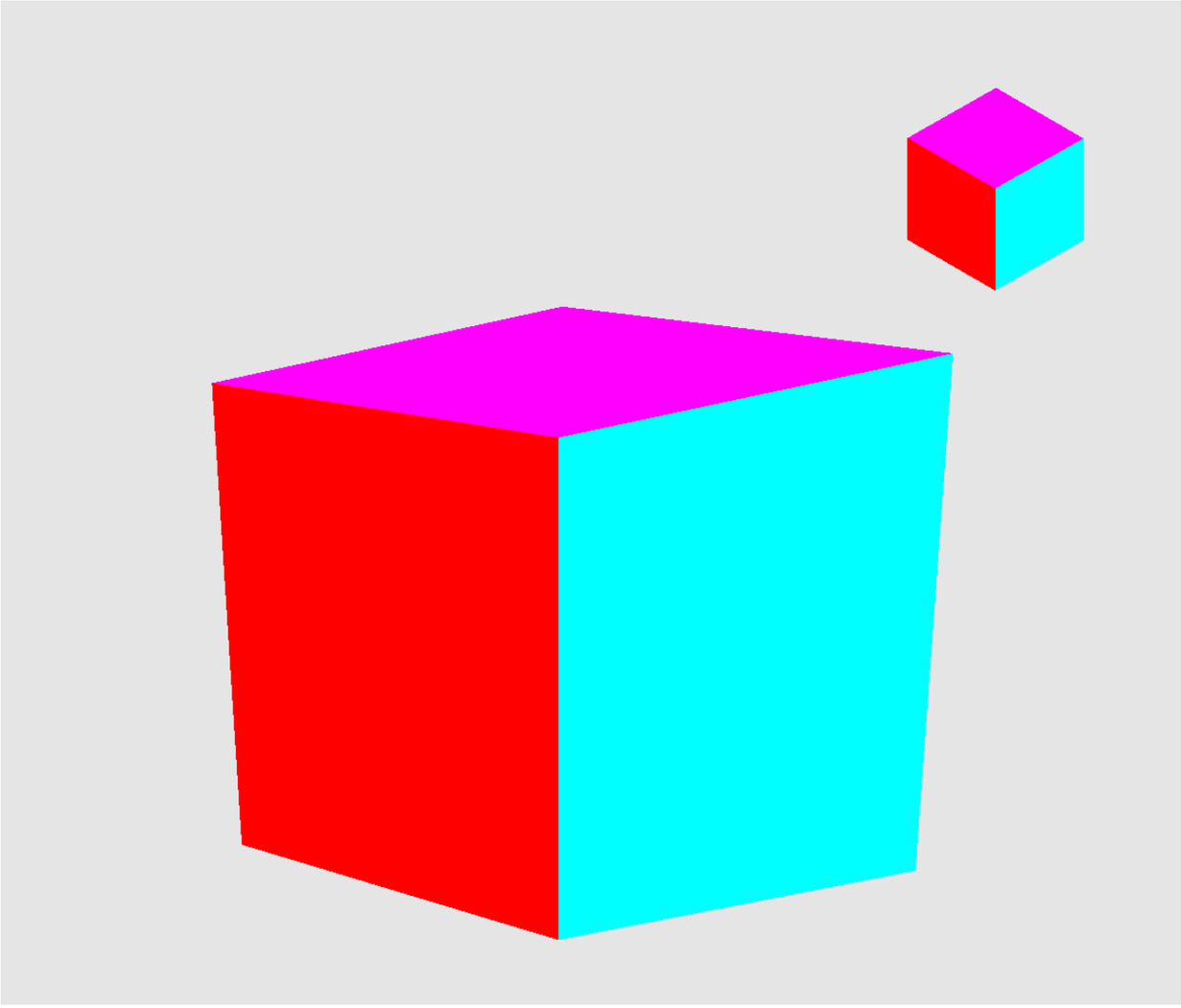
```
#include <GL/gl.h>
#include <GLFW/glfw3.h>
#include <GL/freeglut.h>

#include <iostream>
#include <cmath>

float theta = 0.61;
float phi = 0.78;

float theta1 = 0.61;
float phi1 = 0.78;
int VIEW_MODE = 4;

bool fill = false;

int x = 10;
int y = 10;
int z = -10;

using std::cos, std::sin;
```

```
22
23  void key(GLFWwindow *window, int key, int scancode, int action, int mods)
24  {
25      if (action == GLFW_PRESS)
26      {
27          if (key == GLFW_KEY_ESCAPE)
28          {
29              glfwSetWindowShouldClose(window, GL_TRUE);
30          }
31          else if (key == GLFW_KEY_UP)
32          {
33              theta1 -= 0.1;
34          }
35          else if (key == GLFW_KEY_DOWN)
36          {
37              theta1 += 0.1;
38          }
39          else if (key == GLFW_KEY_LEFT)
40          {
41              phi1 += 0.1;
42          }
43          else if (key == GLFW_KEY_RIGHT)
44          {
45              phi1 -= 0.1;
46          }
47          else if (key == GLFW_KEY_Q)
48          {
49              fill = false;
50          }
51          else if (key == GLFW_KEY_W)
52          {
53              fill = true;
54          }
55      }
56  }
57
58  void DrawCube(GLfloat size)
59  {
60      glBegin(GL_QUADS);
61      // левая грань
62      glColor3f(1.0, 0.0, 0.0);
63      glVertex3f(-size / 2, -size / 2, -size / 2);
64      glVertex3f(-size / 2, size / 2, -size / 2);
65      glVertex3f(-size / 2, size / 2, size / 2);
66      glVertex3f(-size / 2, -size / 2, size / 2);
67      // правая грань
68      glColor3f(0.0, 1.0, 0.0);
69      glVertex3f(size / 2, -size / 2, -size / 2);
70      glVertex3f(size / 2, -size / 2, size / 2);
71      glVertex3f(size / 2, size / 2, size / 2);
72      glVertex3f(size / 2, size / 2, -size / 2);
73      // нижняя грань
74      glColor3f(0.0, 0.0, 1.0);
75      glVertex3f(-size / 2, -size / 2, -size / 2);
76      glVertex3f(-size / 2, -size / 2, size / 2);
77      glVertex3f(size / 2, -size / 2, size / 2);
78      glVertex3f(size / 2, -size / 2, -size / 2);
79      // верхняя грань
80      glColor3f(1.0, 0.0, 1.0);
81      glVertex3f(-size / 2, size / 2, -size / 2);
82      glVertex3f(-size / 2, size / 2, size / 2);
83      glVertex3f(size / 2, size / 2, size / 2);
```

```
84      glVertex3f(size / 2, size / 2, -size / 2);
85      // задняя грань
86      glColor3f(1.0, 1.0, 0.0);
87      glVertex3f(-size / 2, -size / 2, -size / 2);
88      glVertex3f(size / 2, -size / 2, -size / 2);
89      glVertex3f(size / 2, size / 2, -size / 2);
90      glVertex3f(-size / 2, size / 2, -size / 2);
91      // передняя грань
92      glColor3f(0.0, 1.0, 1.0);
93      glVertex3f(-size / 2, -size / 2, size / 2);
94      glVertex3f(size / 2, -size / 2, size / 2);
95      glVertex3f(size / 2, size / 2, size / 2);
96      glVertex3f(-size / 2, size / 2, size / 2);
97
98      glEnd();
99  }
100
101 void display(GLFWwindow *window)
102 {
103     glEnable(GL_DEPTH_TEST);
104     glDepthFunc(GL_LESS);
105     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
106     glMatrixMode(GL_MODELVIEW);
107     glClearColor(0.9f, 0.9f, 0.9f, 1.0f);
108     glPushMatrix();
109     glLoadIdentity();
110
111     glPolygonMode(GL_FRONT_AND_BACK, fill ? GL_FILL : GL_LINE);
112
113     GLfloat m[4][4] = {
114         {0.87, -0.09f, 0.98f, 0.49f},
115         {0.0f, 0.98f, 0.35f, 0.17f},
116         {0.5f, 0.15f, -1.7f, -0.85f},
117         {0.0f, 0.0f, 1.0f, 2.0f}};
118
119     // GLfloat m_perspective[4][4] = {
120     //     {1.f, 0.f, 0.f, 0.f},
121     //     {0.f, 1.f, 0.f, 0.f},
122     //     {0.f, 0.f, 1.f, 0.f},
123     //     {-1 / x, -1 / y, -1 / z, 1.f}};
124
125     GLfloat m_perspective[4][4] = {
126         {1.f, 0.f, 0.f, -(1.f / x)},
127         {0.f, 1.f, 0.f, -(1.f / y)},
128         {0.f, 0.f, 1.f, -(1.f / z)},
129         {0.f, 0.f, 0.f, 1.f}};
130
131     std::cout << -(1.f / x) << std::endl;
132
133     GLfloat front_view[4][4] = {
134         {1.f, 0.f, 0.f, 0.f},
135         {0.f, 1.f, 0.f, 0.f},
136         {0.f, 0.f, -1.f, 0.f},
137         {0.f, 0.f, 0.f, 1.f}};
138
139     GLfloat side_view[4][4] = {
140         {0.f, 0.f, -1.f, 0.f},
141         {0.f, 1.f, 0.f, 0.f},
142         {-1.f, 0.f, 0.f, 0.f},
143         {0.f, 0.f, 0.f, 1.f}};
144
145     GLfloat top_view[4][4] = {
```

```
        {1.f, 0.f, 0.f, 0.f},
        {0.f, 0.f, -1.f, 0.f},
        {0.f, -1.f, 0.f, 0.f},
        {0.f, 0.f, 0.f, 1.f}};

    GLfloat m_rotate[4][4] = {
        {cos(phi1), sin(theta1) * sin(phi1), sin(phi1) * cos(theta1),
    0.f},
        {0.0f, cos(theta1), -sin(theta1), 0.f},
        {sin(phi1), -cos(phi1) * sin(theta1), -cos(phi1) * cos(theta1),
    0.f},
        {0.0f, 0.0f, 0.0f, 1.0f}};



    glMultMatrixf(&m_perspective[0][0]);
     glMultMatrixf(&m_rotate[0][0]);

    DrawCube(0.8);

    glPopMatrix();

    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

    glPushMatrix();
    GLfloat m_move[4][4] = {
        {1.0, 0.f, 0.f, 0.f},
        {0.0f, 1.f, 0.f, 0.f},
        {0.f, 0.f, 1.f, 0.f},
        {0.7f, 0.7f, 0.7f, 1.f}};

    GLfloat m_isometry[4][4] = {
        {cos(phi), sin(theta) * sin(phi), sin(phi) * cos(theta), 0.f},
        {0.0f, cos(theta), -sin(theta), 0.f},
        {sin(phi), -cos(phi) * sin(theta), -cos(phi) * cos(theta), 0.f},
        {0.0f, 0.0f, 0.0f, 1.0f}};
    glMultMatrixf(&m_move[0][0]);
    glMultMatrixf(&m_isometry[0][0]);
    DrawCube(0.2);
    glPopMatrix();
}

int main(int argc, char **argv)
{
    if (!glfwInit())
        exit(1);

    GLFWwindow *window = glfwCreateWindow(1280, 1280, "Lab 1", NULL,
    NULL);

    if (window == NULL)
    {
        glfwTerminate();
        exit(1);
    }

    glfwMakeContextCurrent(window);
    glfwSwapInterval(1);
    glfwSetKeyCallback(window, key);

    while (!glfwWindowShouldClose(window))
    {
```

```
205         display(window);
206         glfwSwapBuffers(window);
207         glfwPollEvents();
208     }
209 
210     glfwDestroyWindow(window);
211     glfwTerminate();
212     return 0;
213 
214     return 0;
215 }
```



Заключение

В ходе лабораторной работы было изучено, как выполнять модельно-видовые преобразования, в частности на разных объектах. Было изучено переключение режимов каркасного и полного отображения, а так же использование нескольних матриц для отрисовки разных объектов с разными преобразованиями.