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: Прикладная математика и информатика

**Лабораторная работа №8**

**по курсу: «Алгоритмы компьютерной графики»**

Выполнил:

Студент группы ИУ9-42Б

Дельман А.Д.

Проверил:

Цалкович П. А.

1. **Условия задачи**

Изображение выглядит как текст

Автоматически созданное описание  
Лабораторная работа 6

1. **Основная теория**

* **glutInit** - инициализация GLUT
* **glutInitDisplayMode** - задает начальный режим отображения
* **glutInitWindowSize** - устанавливает начальный размер окна
* **glutInitDisplayMode** - устанавливает начальный режим отображения
* **glutCreateWindow** - создает окно
* **glutDisplayFunc** - создает функцию обратного вызова для текущего окна (устанавливает функцию, ответственную за перерисовку окна)
* **glutSpecialFunc** - устанавливает функцию, которая будет обрабатывать специальные нажатия клавиатуры
* **glutMainLoop** - запускает бесконечный GLUT цикл
* **glClear** - очищает буферы до заданных значений
* **glutSwapBuffers** - меняет местами буферы текущего окна при двойной буферизации
* **glutPostRedisplay** - помечает текущее окно как требующее повторной отрисовки
* **glLoadIdentity** - заменяет текущую матрицу на единичную
* **glTranslatef** - умножает текущую матрицу на матрицу переноса  
  **glScalef** - умножает текущую матрицу на матрицу масштабирования
* **glRotatef** - умножает текущую матрицу на матрицу поворота
* **glPolygonMode** - устанавливает режим отображения многоугольников
* **glPushMatrix**/**glPopMatrix** - push и pop текущего стека матриц
* **glGenTextures –** создает массив имен текстур
* **glBindTexture -** выбирает указанную текстуру как активную для наложения ее на объекты
* **glTexEnvf -** функция задания параметров наложения текстур
* **glTexCoord2f -** устанавливает текстурные координаты для точек или примитивов
* **glutTimerFunc -** устанавливает таймер
* **glMaterialfv -** устанавливает свойства материала
* **stb\_image -** библиотека, позволяющая загружать изображение и файла
* **glewInit** - инициализация GLEW
* **attribute** - глобальные переменные, которые могут изменяться для каждой вершины, которые передаются из приложения OpenGL в вершинные шейдеры.
* **uniform** - глобальные переменные, которые могут изменяться для каждого примитива
* **varying** – используется для передачи данных между вершинным шейдером и фрагментным шейдером
* **glEnableVertexAttribArray**, **glDisableVertexAttribArray** - Включение или отключение универсального массива атрибутов вершин
* **glGetAttribLocation**, **glGetUniformLocation** - возвращает местоположение attribute/uniform переменной
* **glCreateShader -** создает пустой объект шейдера и возвращает ненулевое значение, с помощью которого на него можно ссылаться
* **glShaderSource -** заменяет исходный код в объекте шейдера
* **glCompileShader -** компилирует исходный код, который был сохранен в объекте шейдера
* **glCreateProgram -** создает пустой программный объект и возвращает ненулевое значение, с помощью которого на него можно ссылаться
* **glAttachShader -** присоединяет объект шейдера к объекту программы
* **glLinkProgram -** связывает программный объект
* **glUseProgram -** устанавливает программный объект как часть текущего состояния визуализации
* **glGenBuffers** - получение идентификатора буфера
* **glBindBuffer** - выбор активного буфера
* **glBufferData** - передача данных в буфер

1. **Практическая реализация**

#include <gl\glew.h>

#include <gl\glut.h>

#define STB\_IMAGE\_IMPLEMENTATION

#include "stb\_image.h";

#define  PI  3.14159265359

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

string loadFile(string fileName) {

    ifstream file(fileName);

    string s;

    getline(file, s, '\0');

    file.close();

    return s;

}

GLuint compileShaders(string fileName, GLenum type){

    string s = loadFile(fileName);

    const char\* shaderCode = s.c\_str();

    GLuint shaderId = glCreateShader(type);

    glShaderSource(shaderId, 1, &shaderCode, NULL);

    glCompileShader(shaderId);

    GLint compileStatus;

    glGetShaderiv(shaderId, GL\_COMPILE\_STATUS, &compileStatus);

    if (!compileStatus) {

        int length;

        glGetShaderiv(shaderId, GL\_INFO\_LOG\_LENGTH, &length);

        char\* cMessage = new char[length];

        glGetShaderInfoLog(shaderId, length, &length, cMessage);

        std::cout << "Cannot Compile Shader: " << cMessage;

        delete[] cMessage;

        glDeleteShader(shaderId);

        return 0;

    }

    return shaderId;

    return 0;

}

struct point {

    float x, y, z;

    point() : x(0), y(0), z(0) {};

    point(float \_x, float \_y, float \_z) : x(\_x), y(\_y), z(\_z) {};

    friend std::ostream& operator<< (std::ostream& out, const point& p);

} a[10], b[10];

std::ostream& operator<< (std::ostream& out, const point& p) {

    out << "Point(" << p.x << ", " << p.y << ", " << p.z << ")";

    return out;

}

point count\_normal(point p1, point p2, point p3) {

    point a, b, n;

    GLfloat l;

    a.x = p2.x - p1.x;

    a.y = p2.y - p1.y;

    a.z = p2.z - p1.z;

    b.x = p3.x - p1.x;

    b.y = p3.y - p1.y;

    b.z = p3.z - p1.z;

    n.x = (a.y \* b.z) - (a.z \* b.y);

    n.y = (a.z \* b.x) - (a.x \* b.z);

    n.z = (a.x \* b.y) - (a.y \* b.x);

    l = sqrt(n.x \* n.x + n.y \* n.y + n.z \* n.z);

    n.x /= l;

    n.y /= l;

    n.z /= l;

    return n;

}

GLuint  texture[2];

GLvoid LoadGLTextures() {

    int x, y, n;

    unsigned char\* data = stbi\_load("3.bmp", &x, &y, &n, 0);

    glGenTextures(1, &texture[0]);

    glBindTexture(GL\_TEXTURE\_2D, texture[0]);

    glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

    glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

    glTexImage2D(GL\_TEXTURE\_2D, 0, 3, x, y, 0,

        n == 3 ? GL\_RGB : GL\_RGBA, GL\_UNSIGNED\_BYTE, data);

    stbi\_image\_free(data);

    data = stbi\_load("1.bmp", &x, &y, &n, 0);

    glGenTextures(1, &texture[1]);

    glBindTexture(GL\_TEXTURE\_2D, texture[1]);

    glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

    glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

    glTexImage2D(GL\_TEXTURE\_2D, 0, 3, x, y, 0,

        n == 3 ? GL\_RGB : GL\_RGBA, GL\_UNSIGNED\_BYTE, data);

    stbi\_image\_free(data);

}

double rotate\_x = 0, rotate\_y = 0;

unsigned int n = 9;

float radius = 0;

float zprism = 4;

float light\_pos0[] = { 2, 4, 2, 1 };

float light\_pos1[] = { 2, -4, 2, 1 };

float h = 0;

float zfloor = -2;

int state = -1;

float g = 0.002;

float acceleration = g, v = 0;

float eyeX = 6, eyeY = 0, eyeZ = 0.5;

float minz = 1e10;

GLuint programId, vboId1, vboId2, vboId3;

float texcoord1[20], texcoord2[8 \* 9];

void initShader() {

    GLuint vShaderId = compileShaders("vertexShader.txt", GL\_VERTEX\_SHADER);

    GLuint fShaderId = compileShaders("fragmentShader.txt", GL\_FRAGMENT\_SHADER);

    programId = glCreateProgram();

    glAttachShader(programId, vShaderId);

    glAttachShader(programId, fShaderId);

    glLinkProgram(programId);

    float vertices1[30];

    for (int i = 0; i < 10; i++) {

        vertices1[3 \* i] = a[i].x;

        vertices1[3 \* i + 1] = a[i].y;

        vertices1[3 \* i + 2] = a[i].z;

        texcoord1[2 \* i] = 0.5 + a[i].x / radius \* 0.5;

        texcoord1[2 \* i + 1] = 0.5 + a[i].y / radius \* 0.5;

    }

    float vertices2[30];

    for (int i = 0; i < 10; i++) {

        vertices2[3 \* i] = b[i].x;

        vertices2[3 \* i + 1] = b[i].y;

        vertices2[3 \* i + 2] = b[i].z;

    }

    float vertices3[12 \* 9];

    for (int i = 0; i < n; i++) {

        vertices3[12 \* i] = a[i].x; vertices3[12 \* i + 1] = a[i].y; vertices3[12 \* i + 2] = a[i].z;

        vertices3[12 \* i + 3] = b[i].x; vertices3[12 \* i + 4] = b[i].y; vertices3[12 \* i + 5] = b[i].z;

        vertices3[12 \* i + 6] = b[i+1].x; vertices3[12 \* i + 7] = b[i+1].y; vertices3[12 \* i + 8] = b[i+1].z;

        vertices3[12 \* i + 9] = a[i+1].x; vertices3[12 \* i + 10] = a[i+1].y; vertices3[12 \* i + 11] = a[i+1].z;

        texcoord2[8 \* i] = 0; texcoord2[8 \* i + 1] = 0;

        texcoord2[8 \* i + 2] = 1; texcoord2[8 \* i + 3] = 0;

        texcoord2[8 \* i + 4] = 1; texcoord2[8 \* i + 5] = 1 \* sin(PI / n);

        texcoord2[8 \* i + 6] = 0; texcoord2[8 \* i + 7] = 1 \* sin(PI / n);

    }

    glGenBuffers(1, &vboId1);

    glBindBuffer(GL\_ARRAY\_BUFFER, vboId1);

    glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices1), vertices1, GL\_STATIC\_DRAW);

    glBindBuffer(GL\_ARRAY\_BUFFER, 0);

    glGenBuffers(1, &vboId2);

    glBindBuffer(GL\_ARRAY\_BUFFER, vboId2);

    glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices2), vertices2, GL\_STATIC\_DRAW);

    glBindBuffer(GL\_ARRAY\_BUFFER, 0);

    glGenBuffers(1, &vboId3);

    glBindBuffer(GL\_ARRAY\_BUFFER, vboId3);

    glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices3), vertices3, GL\_STATIC\_DRAW);

    glBindBuffer(GL\_ARRAY\_BUFFER, 0);

}

void calculate(float r) {

    radius = r;

    float segment = 2 \* PI / n;

    for (int i = 0; i < n; i++) {

        float x = r \* cos(i \* segment);

        float y = r \* sin(i \* segment);

        a[i].x = b[i].x = x;

        a[i].y = b[i].y = y;

        a[i].z = r;

        b[i].z = -r;

    };

    a[9] = a[0];

    b[9] = b[0];

    minz = 1e10;

    float cosa = cos(rotate\_x \* PI / 180), cosb = cos(rotate\_y \* PI / 180), sina = sin(rotate\_x \* PI / 180), sinb = sin(rotate\_y \* PI / 180);

    for (int i = 0; i < n; i++) {

        minz = min(minz, -sinb \* a[i].x + cosb \* (sina \* a[i].y + cosa \* a[i].z));

    }

    for (int i = 0; i < n; i++) {

        minz = min(minz, -sinb \* a[i].x + cosb \* (sina \* a[i].y + cosa \* (a[i].z - 2 \* radius)));

    }

}

void timer(int) {

    glutPostRedisplay();

    glutTimerFunc(1000 / 60, timer, 0);

    h += v \* state;

    v += acceleration;

    if (h < zfloor - minz - zprism) {

        h = zfloor - minz - zprism;

        acceleration \*= -1;

        state \*= -1;

    }

    if (v == 0) {

        h = 0;

        acceleration \*= -1;

        state \*= -1;

    }

}

void show() {

    glUseProgram(programId);

    float vertices[] = {

                -3, -5, zfloor,

                3, -5, zfloor,

                3, 5, zfloor,

                -3, 5, zfloor

    };

    GLuint vboId;

    glGenBuffers(1, &vboId);

    glBindBuffer(GL\_ARRAY\_BUFFER, vboId);

    glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices, GL\_STATIC\_DRAW);

    glBindBuffer(GL\_ARRAY\_BUFFER, 0);

    GLuint posAttributePosition = glGetAttribLocation(programId, "position");

    glBindBuffer(GL\_ARRAY\_BUFFER, vboId);

    glVertexAttribPointer(posAttributePosition, 3, GL\_FLOAT, 0, 0, NULL);

    glBindBuffer(GL\_ARRAY\_BUFFER, 0);

    GLuint normalUniformLocation = glGetUniformLocation(programId, "normal");

    glUniform3f(normalUniformLocation, 0.0f, 0.0f, 1.0f);

    GLuint lightPositionUniformLocation = glGetUniformLocation(programId, "lightPosition");

    glUniform3f(lightPositionUniformLocation, 0.0f, 0.0f, -1.0f);

    glActiveTexture(GL\_TEXTURE0);

    glBindTexture(GL\_TEXTURE\_2D, texture[1]);

    GLuint t = glGetUniformLocation(programId, "texA");

    glUniform1i(t, 0);

    float texCoords[] = { 1,0, 1,1, 0,1, 0,0 };

    GLuint tc = glGetAttribLocation(programId, "myTexCoord");

    glVertexAttribPointer(tc, 2, GL\_FLOAT, 0, 0, texCoords);

    glEnableVertexAttribArray(posAttributePosition);

    glEnableVertexAttribArray(tc);

    glDrawArrays(GL\_QUADS, 0, 4);

    glDisableVertexAttribArray(tc);

    glDisableVertexAttribArray(posAttributePosition);

}

void prism() {

    glUseProgram(programId);

    glActiveTexture(GL\_TEXTURE0);

    glBindTexture(GL\_TEXTURE\_2D, texture[0]);

    GLuint t = glGetUniformLocation(programId, "texA");

    glUniform1i(t, 0);

    GLuint posAttributePosition = glGetAttribLocation(programId, "position");

    GLuint colorUniformPosition = glGetUniformLocation(programId, "color");

    GLuint tc = glGetAttribLocation(programId, "myTexCoord");

    glUniform3f(colorUniformPosition, 0.0f, 0.0f, 0.5f);

    glEnableVertexAttribArray(posAttributePosition);

    glEnableVertexAttribArray(tc);

    glVertexAttribPointer(tc, 2, GL\_FLOAT, 0, 0, texcoord1);

    glBindBuffer(GL\_ARRAY\_BUFFER, vboId1);

    glVertexAttribPointer(posAttributePosition, 3, GL\_FLOAT, 0, 0, NULL);

    glBindBuffer(GL\_ARRAY\_BUFFER, 0);

    glDrawArrays(GL\_POLYGON, 0, 10);

    glVertexAttribPointer(tc, 2, GL\_FLOAT, 0, 0, texcoord1);

    glBindBuffer(GL\_ARRAY\_BUFFER, vboId2);

    glVertexAttribPointer(posAttributePosition, 3, GL\_FLOAT, 0, 0, NULL);

    glBindBuffer(GL\_ARRAY\_BUFFER, 0);

    glDrawArrays(GL\_POLYGON, 0, 10);

    glVertexAttribPointer(tc, 2, GL\_FLOAT, 0, 0, texcoord2);

    glBindBuffer(GL\_ARRAY\_BUFFER, vboId3);

    glVertexAttribPointer(posAttributePosition, 3, GL\_FLOAT, 0, 0, NULL);

    glBindBuffer(GL\_ARRAY\_BUFFER, 0);

    for (int i = 0; i < n; i++) {

        glDrawArrays(GL\_QUADS, i \* 4, i \* 4 + 4);

    }

    glDisableVertexAttribArray(tc);

    glDisableVertexAttribArray(posAttributePosition);

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

    glLoadIdentity();

    gluLookAt(eyeX, eyeY, eyeZ,

        0, 0, 0,

        0, 0, 1);

    glLightfv(GL\_LIGHT0, GL\_POSITION, light\_pos0);

    glLightfv(GL\_LIGHT1, GL\_POSITION, light\_pos1);

    glColor3d(1, 1, 1);

    glBindTexture(GL\_TEXTURE\_2D, 0);

    show();

    glTranslatef(0, 0, h + zprism);

    glRotatef(15, 0, 0, 1);

    glRotatef(rotate\_x, 1, 0, 0);

    glRotatef(rotate\_y, 0, 1, 0);

    prism();

    glutSwapBuffers();

}

void normalKeyFunction(unsigned char key, int x, int y) {

    switch (key) {

    case 'r':

        h = 0;

        acceleration = g;

        v = 0;

        state = -1;

        break;

    case 'a':

        eyeY -= 0.1;

        if (eyeY < -3) eyeY = -3;

        break;

    case 'd':

        eyeY += 0.1;

        if (eyeY > 3) eyeY = 3;

        break;

    case 's':

        eyeZ -= 0.1;

        if (eyeZ < -3) eyeZ = -3;

        break;

    case 'w':

        eyeZ += 0.1;

        if (eyeZ > 3) eyeZ = 3;

        break;

    }

    glutPostRedisplay();

}

void specialKeyFunction(int key, int x, int y) {

    switch (key) {

    case GLUT\_KEY\_LEFT:

        rotate\_x -= 15;

        break;

    case GLUT\_KEY\_RIGHT:

        rotate\_x += 15;

        break;

    case GLUT\_KEY\_DOWN:

        rotate\_y += 15;

        break;

    case GLUT\_KEY\_UP:

        rotate\_y -= 15;

        break;

    }

    minz = 1e10;

    float cosa = cos(rotate\_x \* PI / 180), cosb = cos(rotate\_y \* PI / 180), sina = sin(rotate\_x \* PI / 180), sinb = sin(rotate\_y \* PI / 180);

    for (int i = 0; i < n; i++) {

        minz = min(minz, -sinb \* a[i].x + cosb \* (sina \* a[i].y + cosa \* a[i].z));

    }

    for (int i = 0; i < n; i++) {

        minz = min(minz, -sinb \* a[i].x + cosb \* (sina \* a[i].y + cosa \* (a[i].z - 2 \* radius)));

    }

    glutPostRedisplay();

}

void initGL() {

    LoadGLTextures();

    glDepthFunc(GL\_LESS);

    glEnable(GL\_DEPTH\_TEST);

    glMatrixMode(GL\_MODELVIEW);

    calculate(1);

}

void reshape(int w, int h) {

    glViewport(0, 0, w, h);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluPerspective(90, w / h, 1, 100);

    glMatrixMode(GL\_MODELVIEW);

}

int main(int argc, char\*\* argv) {

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_DEPTH | GLUT\_RGBA);

    glutInitWindowPosition(100, 100);

    glutInitWindowSize(500, 500);

    glutCreateWindow("test");

    initGL();

    glutReshapeFunc(reshape);

    glutSpecialFunc(specialKeyFunction);

    glutKeyboardFunc(normalKeyFunction);

    glutTimerFunc(0, timer, 0);

    glewInit();

    initShader();

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

Вершинный шейдер

#version 110

in vec3 position;

attribute vec2 myTexCoord;

varying vec2 texCoord;

uniform vec3 normal;

uniform vec3 lightPosition;

varying float diff;

void main() {

    gl\_Position = gl\_ModelViewProjectionMatrix \* vec4(position, 1.0);

    texCoord = myTexCoord;

    vec3 FragPos = vec3(gl\_ModelViewMatrix \* vec4(position, 1.0));

    vec3 lightDir = normalize(lightPosition - FragPos);

    diff = dot(lightDir, normal);

}

Фрагментный шейдер

#version 150

uniform sampler2D texA;

varying vec2 texCoord;

varying float diff;

void main() {

    vec3 color = texture2D(texA, texCoord).xyz;

    vec3 lightColor = vec3(0.4, 0.8, 0.9);

    vec3 result = diff \* lightColor \* color \* 0.5;

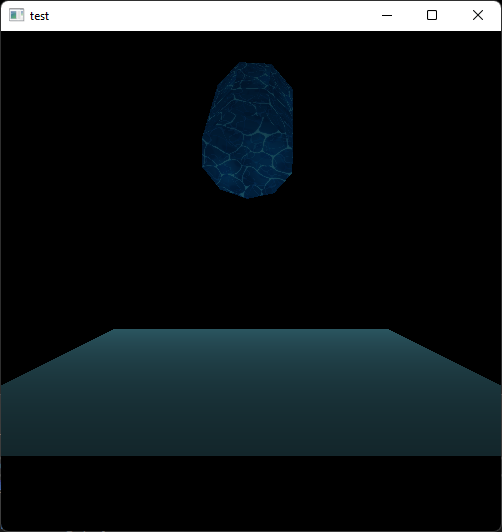
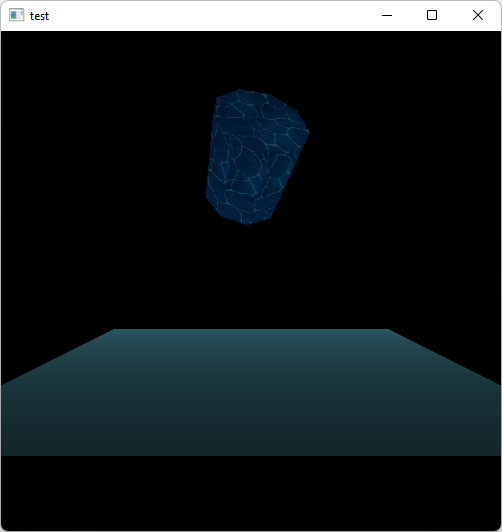
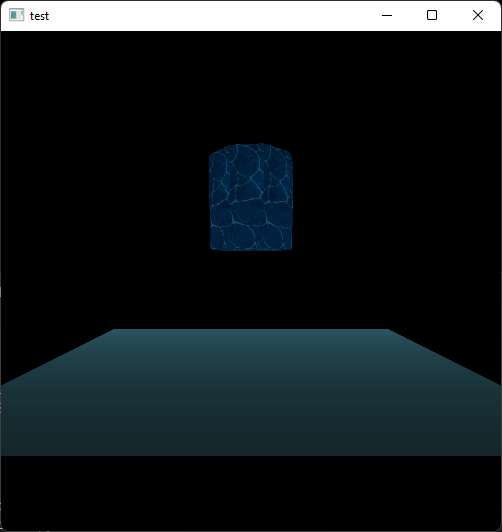
    gl\_FragColor = vec4(result, 1);

}

1. **Заключение**

Определена правильная призма с наложенной текстурой и горизонтальная поверхность. Реализована анимация равноускоренного падения при условии абсолютно упругого соударения. При помощи клавиш влево, вправо, вниз, вверх можно вращать призму.

Отображение объектов, текстурирование и свет реализованы через шейдеры.

Изображение выглядит как текст

Автоматически созданное описание