МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по курсовой работе

по дисциплине «Компьютерная графика»

Тема: «Визуализация 3D объекта»

Студент гр. 6381 _____ Фиалковский М.С.

Студент гр. 6381 _____ Афийчук И.И.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург

2019

Задание.

Для выполнения задания необходимо создать управляемую сцену (фотореалистичность желательна). Необходимо на объекты (усеченные конусы) натянуть текстуру окружающей среды.

Теоретические сведения.

Для выполнения постановленной задачи используется современный подход к разработке приложений с использованием библиотеки OpenGL. В нем мы сами специфицируем некоторые этапы графического конвейера, разрабатывая так называемые шейдерные программы, выполняющиеся непосредственно на GPU.

Для создания эффекта присутствия в окружении был применён метод кубических карт. Вся сцена помещается внутрь большого куба, на грани которого натягиваются специальные текстуры. Этот куб движется вместе с перемещением камеры, что позволяет камере не выходить за пределы локации.

Ход работы.

В данной работе нужно сгенерировать правильную фигуру по заданию (подобие бочки со шляпкой), разработать несколько вершинных и фрагментных шейдеров, правильно определить работу с матрицами преобразования, расположить камеру и управлять ей, предусмотреть интерактивное изменения параметров вывода сцены.

Сборка и компиляция программы осуществляется с помощью утилиты cmake и компилятора gcc из пакета MinGW. Графический интерфейс выполнен с помощью библиотеки FLTK. В качестве обертки над библиотекой OpenGL, используется GLEW. FLTK обеспечивает создание контекста и имеет интерфейс для обращения к некоторым командам OpenGL. Для матричных вычислений используется библиотека GLM. Для загрузки текстур используется библиотека SOLID.

Создание требуемой фигуры состоит из нескольких этапов. Первый – создание некоторого примитива. В нашем случаем им будет цилиндр, который состоит из точек в верхней и нижней окружностях. Ввиду очевидных ограничений, требуется генерировать некоторую аппроксимацию окружности треугольниками, поэтому вводится некоторую константу степени аппроксимации времени компиляции trianglesNum.

Генерируем массив вершин каждого из кругов. Помимо координат вершин нам также требуется генерировать текстурные координаты и векторы нормалей.

```
    float angle = 0.0f;
    float angleStepSize = 360 / trianglesNum;
    float stepTextureCoord = 1.0f / trianglesNum;
   for (int i = 1; i < pointsCircleNum; i++, angle += angleStepSize){
        auto newXBottom = radiusDown * glm::cos(glm::radians(angle)),
             newZBottom = radiusDown * glm::sin(glm::radians(angle)),
             newXTop = radiusUp * glm::cos(glm::radians(angle)),
             newZTop = radiusUp * glm::sin(glm::radians(angle));

        // points for down circle
        vertices_cone[i*elementsPerVert+0] = newXBottom; // x;
        vertices_cone[i*elementsPerVert+1] = centerDown.y;  // y
        vertices_cone[i*elementsPerVert+2] = newZBottom; // z
        vertices_cone[i*elementsPerVert+3]  =  stepTextureCoord  *  i;  //  x
texture coord;
        vertices_cone[i*elementsPerVert+4] = 0.0f; // y texture coord
        glm::vec3 normalVec {0,-1,0};
        normalVec = glm::normalize(normalVec);
        vertices_cone[i*elementsPerVert+5] = normalVec.x;
        vertices_cone[i*elementsPerVert+6] = normalVec.y;
        vertices_cone[i*elementsPerVert+7] = normalVec.z;
        // point for upper circle
        …
 }
```

Далее нам нужно задать порядок обработки вершин для эффективной и корректной отрисовки двух кругов и боковой поверхности цилиндра. Для этого создаётся 3 Element Buffer Object для хранения последовательностей номеров вершин. Рассмотрим на примере нижнего круга:

```
    glBindVertexArray(VAO_cone[0]);
    glBindBuffer(GL_ARRAY_BUFFER, VBO_cone[0]);
```

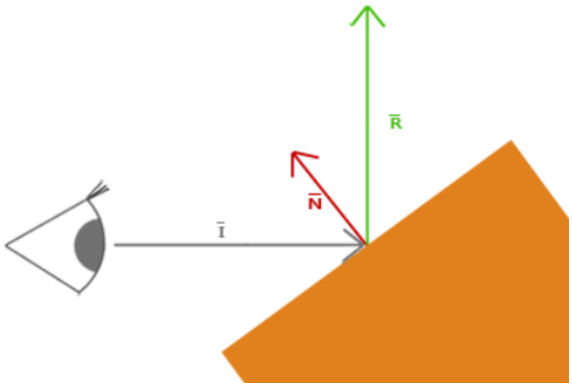
```
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices_cone), vertices_cone,
GL_STATIC_DRAW);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO_cone[0]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices_cone), indices_cone,
GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, elementsPerVert *
sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, elementsPerVert *
sizeof(GLfloat), (GLvoid*)(3 * sizeof(float)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, elementsPerVert *
sizeof(GLfloat), (GLvoid*)(5 * sizeof(float)));
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
```

Аналогичным образом создадим оставшиеся EBO и ещё 2 EBO для вывода граней черным цветом. Забегая вперед можно посмотреть, каким образом созданные буферы используются. В коде ниже происходит эта самая отрисовка:

```
void Cylinder::draw(GLuint figureTexture){
    glBindVertexArray(VAO_cone[0]);
    glBindTexture(GL_TEXTURE_CUBE_MAP, figureTexture);
    glDrawElements(GL_TRIANGLE_FAN, trianglesNum + 2, GL_UNSIGNED_INT, 0);

    glBindVertexArray(VAO_cone[1]);
    glBindTexture(GL_TEXTURE_CUBE_MAP, figureTexture);
    glDrawElements(GL_TRIANGLE_FAN, trianglesNum + 2, GL_UNSIGNED_INT, 0);

    glBindVertexArray(VAO_cone[2]);
    glBindTexture(GL_TEXTURE_CUBE_MAP, figureTexture);
    glDrawElements(GL_TRIANGLE_STRIP, trianglesNum*2+2, GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
}
void Cylinder::drawContour(){
    glBindVertexArray(VAO_cone[3]);
    glDrawElements(GL_LINES, trianglesNum * 2, GL_UNSIGNED_INT, 0);

    glBindVertexArray(VAO_cone[4]);
    glDrawElements(GL_LINES, trianglesNum * 2, GL_UNSIGNED_INT, 0);

    glBindVertexArray(0);
}
```

Теперь можно приступить ко второму этапу: создание фигуры из примитива. Требуемая бочка со шляпкой состоит из трёх цилиндров: нижний (самый большой), средний (меньший по радиусу первого) и верхний (с верхним нулевым радиусом, что создает конус). Для объединения этих примитивов создадим новый класс *SceneShape,* в который поместим эти три примитива, и определим методы для удобной работы с фигурой.

Для эффекта прозрачности нашей фигуры будем натягивать на неё текстуру окружающей среды. Для этого нужны нормали N и позиция камеры с вектором направления I.



Требуемый вектор R вычисляется во фрагментом шейдере, где сразу и происходит так называемая выборка из текстуры для получения нужного цвета обрабатываемого фрагмента. Код шейдеров:

Фрагментный:

```
#version 330 core
out vec4 FragColor;
in vec3 Normal;
in vec3 Position;
in vec3 Color;

uniform vec3 cameraPos;
uniform samplerCube skybox;

void main(){
    vec3 I = normalize(cameraPos - Position);
    vec3 R = reflect(-I, normalize(Normal));
    FragColor = vec4(texture(skybox, R).rgb, 1.0);
    FragColor = vec4(FragColor.r + Color.r/255, FragColor.g + Color.g/255,
FragColor.b + Color.b/255, 1.0);
}
```

Вершинный:

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTextCoord;
layout (location = 2) in vec3 aNormal;
out vec3 Normal;
out vec3 Position;
out vec3 Color;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform vec3 aColor;

void main() {
    Normal = mat3(transpose(inverse(model))) * aNormal;
    Position = vec3(model * vec4(aPos, 1.0));
    Position = vec3(Position.x, 1.0 - Position.y, Position.z);
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    Color = aColor;
}
```

Также в программе предусмотрено изменения параметров вывода, можно менять: отображения мировой системы координат, отображение нормалей для каждой вершины и изменение цвета материала фигуры. Данная возможность реализована через работу с объектом состояния State и дополнительной uniform переменной aColor в шейдере для задания цвета.

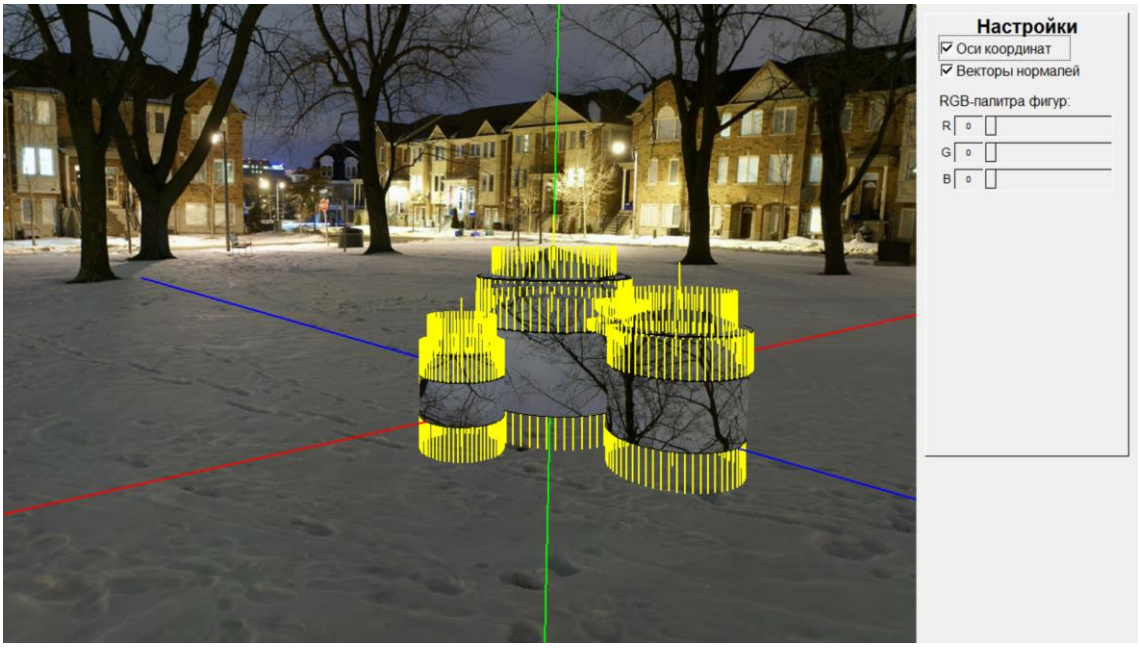
Для отрисовки нормали используется геометрический шейдер.

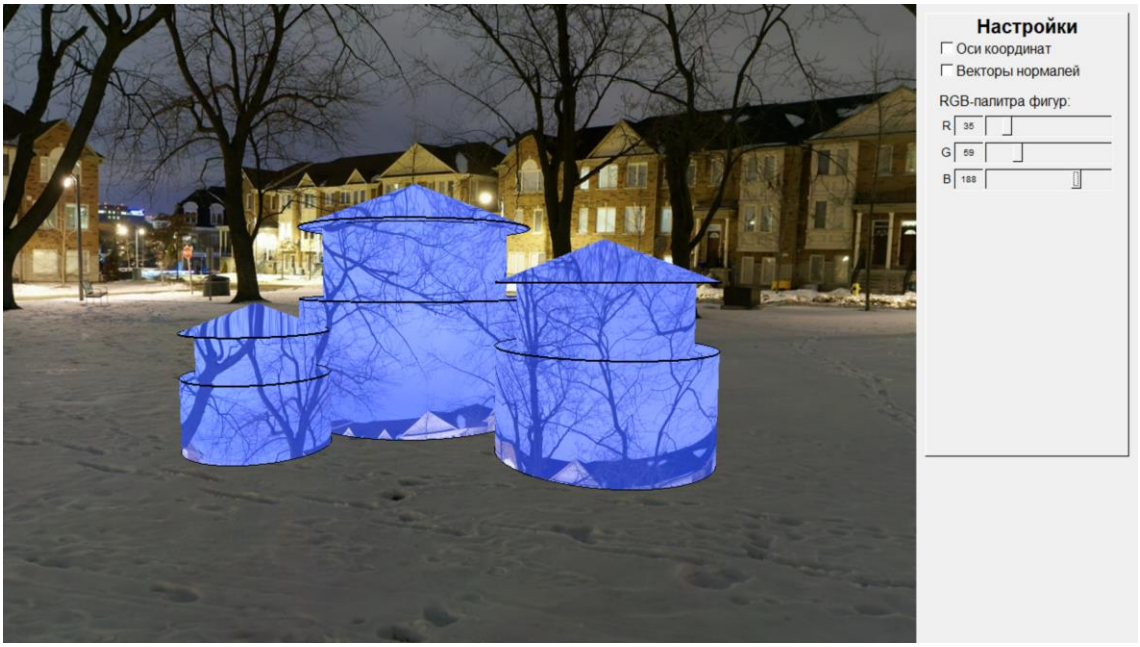
Загрузка текстур, генерация вершин, чтение, компиляция и сборка шейдеров происходит один раз – при запуске программы.

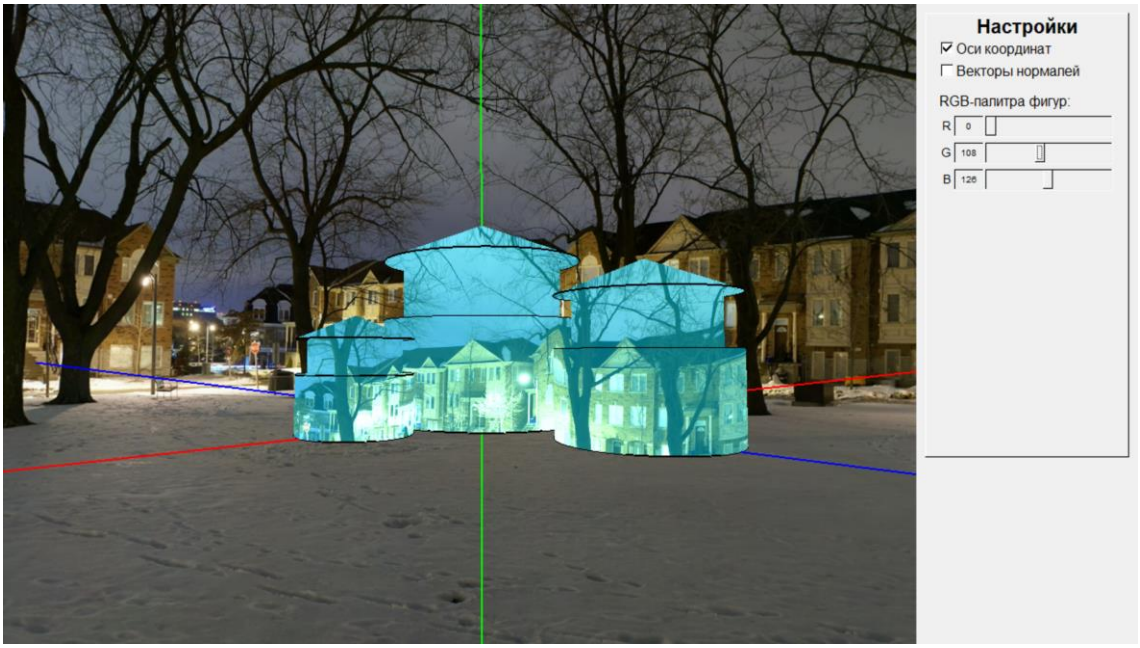
Тестирование.

Справа от области отрисовки находятся элементы UI, с помощью которых можно изменять параметры системы.

Вслед за движением мыши по области отрисовки меняется позиция направление камеры. Клавиши WASD управляют движением камеры по осям X и Z.







Вывод.

В процессе выполнения курсовой работы была разработана программа, моделирующая заданную объёмную фигуру с требуемым характеристиками. При выполнении работы были приобретены навыки работы с графическим конвейером и шейдернымм программам из графической библиотеки OpenGL.

Исходный код класса Cylinder:

```
 void Cylinder::setParams(glm::vec3 o_1, GLfloat c_1, glm::vec3 o_2, GLfloat c_2){
     centerDown = o_1;
     centerUp = o_2;
     radiusDown = c_1;
     radiusUp = c_2;
 }

 void Cylinder::loadBuffer(){
     int pointsCircleNum = trianglesNum + 1; // +1 for center point

     constexpr GLint elementsPerVert = 8;
     float vertices_cone[pointsCircleNum * (elementsPerVert) * 2]; // 3 for xyz + 2 for texture coords + 3
for normals, 2 for two circles
     // down center
     vertices_cone[0] = centerDown.x;
     vertices_cone[1] = centerDown.y;
     vertices_cone[2] = centerDown.z;
     vertices_cone[3] = 0.5f;
     vertices_cone[4] = 0.0f;
     vertices_cone[5] = 0.0f; // normal
     vertices_cone[6] = -1.0f; // normal
     vertices_cone[7] = 0.0f; // normal
     // up center
     vertices_cone[pointsCircleNum * elementsPerVert + 0] = centerUp.x;
     vertices_cone[pointsCircleNum * elementsPerVert + 1] = centerUp.y;
     vertices_cone[pointsCircleNum * elementsPerVert + 2] = centerUp.z;
     vertices_cone[pointsCircleNum * elementsPerVert + 3] = 0.5f;
     vertices_cone[pointsCircleNum * elementsPerVert + 4] = 1.0f;
     vertices_cone[pointsCircleNum * elementsPerVert + 5] = 0.0f; // normal
     vertices_cone[pointsCircleNum * elementsPerVert + 6] = 1.0f; // normal
     vertices_cone[pointsCircleNum * elementsPerVert + 7] = 0.0f; // normal
     // +3 - texture coordinate x
     // +4 - texture coordinate y

     float angle = 0.0f;
     float angleStepSize = 360 / trianglesNum;
     float stepTextureCoord = 1.0f / trianglesNum;

     for (int i = 1; i < pointsCircleNum; i++, angle += angleStepSize){
         auto newXBottom = radiusDown * glm::cos(glm::radians(angle)),
              newZBottom = radiusDown * glm::sin(glm::radians(angle)),
              newXTop = radiusUp * glm::cos(glm::radians(angle)),
              newZTop = radiusUp * glm::sin(glm::radians(angle));

         // points for down circle
         vertices_cone[i*elementsPerVert+0] = newXBottom; // x;
         vertices_cone[i*elementsPerVert+1] = centerDown.y;  // y
         vertices_cone[i*elementsPerVert+2] = newZBottom; // z
         vertices_cone[i*elementsPerVert+3] = stepTextureCoord * i; // x texture coord;
         vertices_cone[i*elementsPerVert+4] = 0.0f; // y texture coord
         glm::vec3 normalVec {0,-1,0};
         normalVec = glm::normalize(normalVec);
         vertices_cone[i*elementsPerVert+5] = normalVec.x;
         vertices_cone[i*elementsPerVert+6] = normalVec.y;
         vertices_cone[i*elementsPerVert+7] = normalVec.z;

         // points for up circle
         vertices_cone[i*elementsPerVert+0 + pointsCircleNum*elementsPerVert] = newXTop; // x;
         vertices_cone[i*elementsPerVert+1 + pointsCircleNum*elementsPerVert] = centerUp.y;  // y
         vertices_cone[i*elementsPerVert+2 + pointsCircleNum*elementsPerVert] = newZTop; // z
         vertices_cone[i*elementsPerVert+3 + pointsCircleNum*elementsPerVert] = stepTextureCoord * i; // x
texture coord;
         vertices_cone[i*elementsPerVert+4 + pointsCircleNum*elementsPerVert] = 1.0f; // y texture coord
         normalVec =  {0,1,0};
         normalVec = glm::normalize(normalVec);
```

```
            vertices_cone[i*elementsPerVert+5 + pointsCircleNum*elementsPerVert] = normalVec.x;
            vertices_cone[i*elementsPerVert+6 + pointsCircleNum*elementsPerVert] = normalVec.y;
            vertices_cone[i*elementsPerVert+7 + pointsCircleNum*elementsPerVert] = normalVec.z;
        }
        GLuint indices_cone[(trianglesNum + 2)];
        for (int i = 0; i < trianglesNum + 2; i++){
            indices_cone[i] = i;
            // indices_cone[i + trianglesNum + 2] = i + pointsCircleNum*3;
        }
        indices_cone[trianglesNum + 2 - 1] = indices_cone[1];

        GLuint indices_facet[trianglesNum*2 + 2]; // грань
        for (int i = 0; i < trianglesNum; i++){
            indices_facet[i*2] = i + trianglesNum + 2;
            indices_facet[i*2 + 1] = i + 1;
        }
        indices_facet[trianglesNum*2] = indices_facet[0];
        indices_facet[trianglesNum*2+1] = indices_facet[1];

        GLuint indices_contour[trianglesNum*2] = {};
        for (int i = 0; i < trianglesNum; i++){
            indices_contour[i*2] = i + 1;
            indices_contour[i*2 + 1] = i + 2;
        }
        indices_contour[trianglesNum*2-1] = indices_contour[0];

        // down cirlce + up circle + facet + contour down + contour up
        glGenVertexArrays(5, VAO_cone);
        glGenBuffers(5, VBO_cone);
        glGenBuffers(5, EBO_cone);

        glBindVertexArray(VAO_cone[0]);
        glBindBuffer(GL_ARRAY_BUFFER, VBO_cone[0]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices_cone), vertices_cone, GL_STATIC_DRAW);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO_cone[0]);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices_cone), indices_cone, GL_STATIC_DRAW);
        glEnableVertexAttribArray(0);
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)0);
        glEnableVertexAttribArray(1);
        glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(3 *
sizeof(float)));
        glEnableVertexAttribArray(2);
        glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(5 *
sizeof(float)));
        glBindBuffer(GL_ARRAY_BUFFER, 0);
        glBindVertexArray(0);

        glBindVertexArray(VAO_cone[1]);
        glBindBuffer(GL_ARRAY_BUFFER, VBO_cone[1]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices_cone), vertices_cone + pointsCircleNum * elementsPerVert,
GL_STATIC_DRAW);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO_cone[1]);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices_cone), indices_cone, GL_STATIC_DRAW);
        glEnableVertexAttribArray(0);
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)0);
        glEnableVertexAttribArray(1);
        glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(3 *
sizeof(float)));
        glEnableVertexAttribArray(2);
        glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(5 *
sizeof(float)));
        glBindBuffer(GL_ARRAY_BUFFER, 0);
        glBindVertexArray(0);

        glBindVertexArray(VAO_cone[2]);
        glBindBuffer(GL_ARRAY_BUFFER, VBO_cone[2]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices_cone), vertices_cone, GL_STATIC_DRAW);
```

```
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO_cone[2]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices_facet), indices_facet, GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(3 *
sizeof(float)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(5 *
sizeof(float)));
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);

    glBindVertexArray(VAO_cone[3]);
    glBindBuffer(GL_ARRAY_BUFFER, VBO_cone[3]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices_cone), vertices_cone, GL_STATIC_DRAW);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO_cone[3]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices_contour), indices_contour, GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(3 *
sizeof(float)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(5 *
sizeof(float)));
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);

    glBindVertexArray(VAO_cone[4]);
    glBindBuffer(GL_ARRAY_BUFFER, VBO_cone[4]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices_cone), vertices_cone + pointsCircleNum * elementsPerVert,
GL_STATIC_DRAW);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO_cone[4]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices_contour), indices_contour, GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(3 *
sizeof(float)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, elementsPerVert * sizeof(GLfloat), (GLvoid*)(5 *
sizeof(float)));
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}

void Cylinder::draw(GLuint figureTexture){
    glBindVertexArray(VAO_cone[0]);
    glBindTexture(GL_TEXTURE_CUBE_MAP, figureTexture);
    glDrawElements(GL_TRIANGLE_FAN, trianglesNum + 2, GL_UNSIGNED_INT, 0);

    glBindVertexArray(VAO_cone[1]);
    glBindTexture(GL_TEXTURE_CUBE_MAP, figureTexture);
    glDrawElements(GL_TRIANGLE_FAN, trianglesNum + 2, GL_UNSIGNED_INT, 0);

    glBindVertexArray(VAO_cone[2]);
    glBindTexture(GL_TEXTURE_CUBE_MAP, figureTexture);
    glDrawElements(GL_TRIANGLE_STRIP, trianglesNum*2+2, GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
}

void Cylinder::drawContour(){
    glBindVertexArray(VAO_cone[3]);
    glDrawElements(GL_LINES, trianglesNum * 2, GL_UNSIGNED_INT, 0);

    glBindVertexArray(VAO_cone[4]);
```

```
    glDrawElements(GL_LINES, trianglesNum * 2, GL_UNSIGNED_INT, 0);

    glBindVertexArray(0);
}

SceneShape::SceneShape(glm::vec3 center, GLfloat scale):
   downCenter(center), scaleFactor(scale)
{
    c_1.setParams(glm::vec3{0.0f, -1.0f, 0.0f}, 1.0f, glm::vec3{0.0f, 0.0f, 0.0f}, 1.0f);
    c_2.setParams(glm::vec3{0.0f, 0.0f, 0.0f}, 0.80f, glm::vec3{0.0f, 0.6f, 0.0f}, 0.80f);
    c_3.setParams(glm::vec3{0.0f, 0.6f, 0.0f}, 1.00f, glm::vec3{0.0f, 1.0f, 0.0f}, 0.0f);
}

void SceneShape::loadBuffer(){
    c_1.loadBuffer();
    c_2.loadBuffer();
    c_3.loadBuffer();
}

void SceneShape::draw(GLuint figureTexture){
    c_1.draw(figureTexture);
    c_2.draw(figureTexture);
    c_3.draw(figureTexture);
}

void SceneShape::drawContour(){
    c_1.drawContour();
    c_2.drawContour();
    c_3.drawContour();
}
```

Исходный код класса SimpleGL3Window:

```
SimpleGL3Window::SimpleGL3Window(State* ptr, int x, int y, int w, int h) :  Fl_Gl_Window(x, y, w, h){
    statePtr = ptr;
    mode(FL_RGB8 | FL_DOUBLE | FL_OPENGL3);
    screenWidth = this->w();
    screenHeight = this->h();
}

void SimpleGL3Window::draw(void) {
    glEnable(GL_DEPTH_TEST);
    shaderProgramFigures.readAndCompile("Shaders/vertex_figures.shader",
"Shaders/fragment_figures.shader");
    shaderProgramAxes.readAndCompile("Shaders/vertex_axes.shader", "Shaders/fragment.shader");
    shaderProgramSkyBox.readAndCompile("Shaders/vertex_map.shader", "Shaders/fragment_map.shader");
    shaderNormals.readAndCompile("Shaders/vertex_normal.shader",           "Shaders/fragment_normal.shader",
"Shaders/geometry_normal.shader");
    shaderContour.readAndCompile("Shaders/vertex_contour.shader", "Shaders/fragment_contour.shader");
    shaderProgramSkyBox.Use();
    shaderProgramSkyBox.setInt("skybox", 0);
    shaderProgramFigures.Use();
    shaderProgramFigures.setInt("skybox", 0);
    loadBuffers();
    Do_Movement();

    // glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glm::mat4 model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(0.0f, 1.4f, 0.0f));
    model = glm::scale(model, glm::vec3(1.4f, 1.4f, 1.4f));
    glm::mat4 view = camera.GetViewMatrix();
    glm::mat4 projection = glm::mat4(1.0f);
    projection = glm::perspective(camera.Zoom, (GLfloat)screenWidth / (GLfloat)screenHeight, 0.1f, 50.0f);
    auto color = statePtr->getRGB();
```

```
    shaderProgramFigures.setVec3("aColor", glm::vec3{color[0], color[1], color[2]});

    drawFigure(shape_1, model, view, projection, camera.Position);
    drawContour(shape_1, model, view, projection);
    if (statePtr->getIsNormalsDrawn())
        drawNormals(shape_1, model, view, projection);

    model = glm::translate(glm::mat4(1.0f), glm::vec3(2.1f, 0.7f, 0.0f));
    model = glm::scale(model, glm::vec3(0.7f, 0.7f, 0.7f));
    drawFigure(shape_2, model, view, projection, camera.Position);
    drawContour(shape_2, model, view, projection);
    if (statePtr->getIsNormalsDrawn())
        drawNormals(shape_2, model, view, projection);

    model = glm::translate(glm::mat4(1.0f), glm::vec3(-2.1f, 0.7f, 0.0f));
    model = glm::scale(model, glm::vec3(0.7f, 0.7f, 0.7f));
    drawFigure(shape_3, model, view, projection, camera.Position);
    drawContour(shape_3, model, view, projection);
    if (statePtr->getIsNormalsDrawn())
        drawNormals(shape_3, model, view, projection);

    model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 1.0f, 2.6f));
    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
    drawFigure(shape_4, model, view, projection, camera.Position);
    drawContour(shape_4, model, view, projection);
    if (statePtr->getIsNormalsDrawn())
        drawNormals(shape_4, model, view, projection);

    // axes
    if (statePtr->getIsAxesDrawn()){
        shaderProgramAxes.Use();
        glm::mat4 modelAxe = glm::scale(glm::mat4(1.0f), glm::vec3(100, 100, 100));
        glm::mat4 viewAxe = camera.GetViewMatrix();
        glm::mat4     projectionAxe     =     glm::perspective(camera.Zoom,     (GLfloat)screenWidth     /
(GLfloat)screenHeight, 0.1f, 50.0f);

        shaderProgramAxes.setMat4("model", modelAxe);
        shaderProgramAxes.setMat4("view", viewAxe);
        shaderProgramAxes.setMat4("projection", projectionAxe);

        glBindVertexArray(VAO_axes);
        glLineWidth(2.0f);
        glDrawArrays(GL_LINES, 0, 6*3);
        glBindVertexArray(0);
    }

    // draw skybox as last
    glDepthFunc(GL_LEQUAL);  // change depth function so depth test passes when values are equal to depth
buffer's content
    shaderProgramSkyBox.Use();
    model = glm::mat4(1.0f);
    view = camera.GetViewMatrix();
    projection  =  glm::perspective(glm::radians(camera.Zoom),  (float)screenWidth  /  (float)screenHeight,
0.1f, 100.0f);
    view = glm::mat4(glm::mat3(camera.GetViewMatrix())); // remove translation from the view matrix
    shaderProgramSkyBox.setMat4("view", view);
    shaderProgramSkyBox.setMat4("projection", projection);
    // skybox cube
    glBindVertexArray(skyboxVAO);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
    glDrawArrays(GL_TRIANGLES, 0, 36);
    glBindVertexArray(0);
    glDepthFunc(GL_LESS); // set depth function back to default
}
```

```
  void SimpleGL3Window::drawNormals(SceneShape &shape, const glm::mat4 &model, const glm::mat4 &view, const
glm::mat4 &projection){
      shaderNormals.Use();
      shaderNormals.setMat4("projection", projection);
      shaderNormals.setMat4("view", view);
      shaderNormals.setMat4("model", model);
      shape.draw(cubemapTexture);
  }

  void SimpleGL3Window::drawContour(SceneShape &shape, const glm::mat4 &model, const glm::mat4 &view, const
glm::mat4 &projection){
      shaderContour.Use();
      shaderContour.setMat4("projection", projection);
      shaderContour.setMat4("view", view);
      shaderContour.setMat4("model", model);
      shape.drawContour();
  }

  void SimpleGL3Window::drawFigure(SceneShape &shape, const glm::mat4 &model, const glm::mat4 &view, const
glm::mat4 &projection, const glm::vec3 &cameraPos){
      shaderProgramFigures.Use();
      shaderProgramFigures.setMat4("model", model);
      shaderProgramFigures.setMat4("view", view);
      shaderProgramFigures.setMat4("projection", projection);
      shaderProgramFigures.setVec3("cameraPos", camera.Position);
      shape.draw(cubemapTexture);
  }

  void SimpleGL3Window::Do_Movement() {
      // Camera controls
      if(wasd[0])
          camera.ProcessKeyboard(FORWARD);
      if(wasd[2])
          camera.ProcessKeyboard(BACKWARD);
      if(wasd[1])
          camera.ProcessKeyboard(LEFT);
      if(wasd[3])
          camera.ProcessKeyboard(RIGHT);
  }

  void SimpleGL3Window::loadBuffers(){
      static bool isLoaded = false;
      if (isLoaded)
          return;
      shape_1.loadBuffer();
      shape_2.loadBuffer();
      shape_3.loadBuffer();
      shape_4.loadBuffer();

      float vertices_axes[] = {
          -1.0,  0.0,  0.0, 1.0,  0.0,  0.0,
           1.0,  0.0,  0.0, 1.0,  0.0,  0.0,
           0.0, -1.0,  0.0, 0.0,  1.0,  0.0,
           0.0,  1.0,  0.0, 0.0,  1.0,  0.0,
           0.0,  0.0, -1.0, 0.0,  0.0,  1.0,
           0.0,  0.0,  1.0, 0.0,  0.0,  1.0,
      };
      glGenVertexArrays(1, &VAO_axes);
      glBindVertexArray(VAO_axes);
      glGenBuffers(1, &VBO_axes);
      glBindBuffer(GL_ARRAY_BUFFER, VBO_axes);
      glBufferData(GL_ARRAY_BUFFER, sizeof(vertices_axes), vertices_axes, GL_STATIC_DRAW);
      glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
      glEnableVertexAttribArray(0);
      glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)(3* sizeof(GLfloat)));
      glEnableVertexAttribArray(1);
      glBindBuffer(GL_ARRAY_BUFFER, 0);
```

```
    glBindVertexArray(0);

    float skyboxVertices[] = {
    // positions
    -1.0f,  1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,
     1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

     1.0f, -1.0f, -1.0f,
     1.0f, -1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

    -1.0f,  1.0f, -1.0f,
     1.0f,  1.0f, -1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
     1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
     1.0f, -1.0f,  1.0f
    };
    // skybox VAO
    glGenVertexArrays(1, &skyboxVAO);
    glGenBuffers(1, &skyboxVBO);
    glBindVertexArray(skyboxVAO);
    glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);

    std::vector<std::string> faces {
        "Resources/right.jpg",
        "Resources/left.jpg",
        "Resources/top.jpg",
        "Resources/bottom.jpg",
        "Resources/front.jpg",
        "Resources/back.jpg"
    };
    cubemapTexture = loadCubemap(faces);
    isLoaded = true;
}
```

```
void SimpleGL3Window::update() {
    redraw();
}

unsigned int SimpleGL3Window::loadCubemap(const std::vector<std::string> &faces) {
    unsigned int textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);

    int width, height, nrChannels;
    for (unsigned int i = 0; i < faces.size(); i++){
        unsigned char *data = SOIL_load_image(faces[i].c_str(), &width, &height, &nrChannels, 0);
        if (data) {
            glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i,
                         0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
            SOIL_free_image_data(data);
        }
        else {
            std::cout << "Cubemap texture failed to load at path: " << faces[i] << std::endl;
            SOIL_free_image_data(data);
        }
    }
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
    return textureID;
}
```

Исходный код геометрического шейдера для нормалей:

```
#version 330 core
layout (triangles) in;
layout (line_strip, max_vertices = 6) out;

in VS_OUT {
    vec3 normal;
} gs_in[];

const float MAGNITUDE = 1.0;

void GenerateLine(int index){
    gl_Position = gl_in[index].gl_Position;
    EmitVertex();
    gl_Position = gl_in[index].gl_Position + vec4(gs_in[index].normal, 0.0) * MAGNITUDE;
    EmitVertex();
    EndPrimitive();
}

void main() {
    GenerateLine(0); // first vertex normal
    GenerateLine(1); // second vertex normal
    GenerateLine(2); // third vertex normal
}
```

Исходный код вершинного шейдера для нормалей:

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 2) in vec3 aNormal;

out VS_OUT {
    vec3 normal;
} vs_out;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;
```

```
void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    mat3 normalMatrix = mat3(transpose(inverse(view * model)));
    vs_out.normal = normalize(vec3(projection * vec4(normalMatrix * aNormal, 0.0)));
}
```

Исходный код фрагментного шейдера для нормалей:

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0, 1.0, 0.0, 1.0);
}
```

Исходный код вершинного шейдера для кубической карты:

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

out vec3 TexCoords;

uniform mat4 projection;
uniform mat4 view;

void main()
{
    TexCoords = aPos;
    vec4 pos = projection * view * vec4(aPos, 1.0);
    gl_Position = pos.xyww;
}
```