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

ОТЧЕТ

по лабораторной работе №6
по дисциплине «Компьютерная графика»
Тема: «Реализация трехмерного
объекта с использованием
библиотеки OpenGL»

Студент гр. 6381 _____ Фиалковский М.С.

Студент гр. 6381 _____ Афийчук И.И.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург
2019

Задание.

Разработать программу, реализующую представление разработанного вами трехмерного рисунка, используя предложенные функции библиотеки OpenGL (матрицы видового преобразования, проецирование) и язык GLSL.

Разработанная программа должна быть пополнена возможностями остановки интерактивно различных атрибутов через вызов соответствующих элементов интерфейса пользователя, замена типа проекции, управление преобразованиями, как с помощью мыши, так и с помощью диалоговых элементов.

Общие сведения.

Для выполнения постановленной задачи используется современный подход к разработке приложений с использованием библиотеки OpenGL, описанный в прошлой лабораторной работе (использование только core-profile режим).

В данной работе нужно разработать вершинный и фрагментный шейдеры, правильно определить работу с матрицами преобразования, расположить камеру и управлять ей, предусмотреть интерактивное изменения параметров отрисовки.

Ход работы.

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.

Построений фигур происходит путем поэтапного преобразования изначально загруженных в буфер соответствующих координат. Рассмотрим код этого процесса на примере куба:

```
float vertices_cube[] = {
    // Coordinates
```

```
        -0.5,  0.5, 0.5, // N-W
         0.5,  0.5, 0.5, // N-E
         0.5, -0.5, 0.5, // S-E
        -0.5, -0.5, 0.5, // S-W

        -0.5,   0.5,  -0.5,   // N-W
         0.5,   0.5,  -0.5,   // N-E
         0.5,  -0.5,  -0.5,   // S-E
        -0.5,  -0.5,  -0.5,   // S-W
    };
    GLuint indices_cube[] = {
        0,1, 1,2, 2,3, 3,0,
        0,4, 4,5, 5,1,
        4,7, 7,6, 6,5,
        6,2, 7,3
    };
    glGenVertexArrays(1, &VAO_cube);
    glGenBuffers(1, &VBO_cube);
    glGenBuffers(1, &EBO_cube);
    glBindVertexArray(VAO_cube);
    glBindBuffer(GL_ARRAY_BUFFER, VBO_cube);
    glBufferData(GL_ARRAY_BUFFER,
                 sizeof(vertices_cube), vertices_cube, GL_STATIC_DRAW);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO_cube);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
                 sizeof(indices_cube), indices_cube, GL_STATIC_DRAW);

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                          3 * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
```

Это происходит лишь один раз – при запуске программы. Далее надобность в повторной загрузке данных отпадает, т.к. они всё время хранятся в буфере отрисовки.

Затем мы получаем данные с пользовательского интерфейса и обновляем матрицы требуемых преобразований:

```
    glm::mat4 model = glm::mat4(1.0f);
    auto [move_x, move_y, move_z] = statePtr->getXYZ(ActionType::translate);
    auto [angle_x, angle_y, angle_z] = statePtr->getXYZ(ActionType::rotate);
    auto [scale_x, scale_y, scale_z] = statePtr->getXYZ(ActionType::scale);
    model = glm::translate(model, glm::vec3(move_x, move_y, move_z));
    model = glm::rotate(model, glm::radians(angle_x), glm::vec3(1.0f, 0.0f,
0.0f));
```

```
    model = glm::rotate(model, glm::radians(angle_y), glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::rotate(model, glm::radians(angle_z), glm::vec3(0.0f, 0.0f, 1.0f));
    model = glm::scale(model, glm::vec3(scale_x, scale_y, scale_z));

    glm::mat4 view = glm::mat4(1.0f);
    view = camera.GetViewMatrix();

    glm::mat4 projection = glm::mat4(1.0f);
    if (statePtr->getProjectionType() == ProjectionType::perspective){
        projection = glm::perspective(camera.Zoom, (GLfloat)screenWidth / (GLfloat)screenHeight, 0.1f, 50.0f);
    }
    if (statePtr->getProjectionType() == ProjectionType::orthogonal){
        projection = glm::ortho(-4.0f, 4.0f, -3.0f, 3.0f, 0.1f, 100.0f );
    }
```

И передаем пересчитанные матрицы преобразования в шейдер:

```
    glUniformMatrix4fv(glGetUniformLocation(shaderProgramFigures.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(glGetUniformLocation(shaderProgramFigures.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(glGetUniformLocation(shaderProgramFigures.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
    glUniform3f(glGetUniformLocation(shaderProgramFigures.Program, "figureColor"), 1.0, 1.0f, 1.0f);
```

И рисуем куб:

```
    glLineWidth(2.0f);
    glBindVertexArray(VAO_cube);
    glDrawElements(GL_LINES, 4*6, GL_UNSIGNED_INT, 0);
```

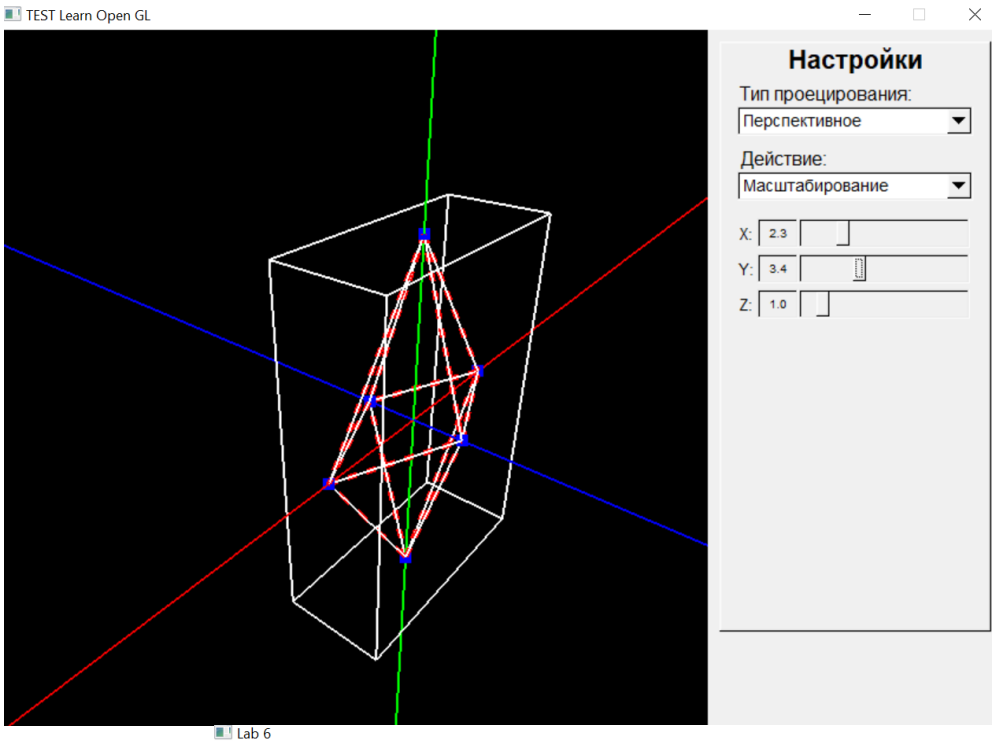
В работе используется три шейдера: два вершинных и один фрагментный. Из них затем компилируется две шейдерные программы: первый для фигур и второй для координатных осей. Переключение шейдеров происходит с помощью метода Use() соответствующего экземпляра класса.

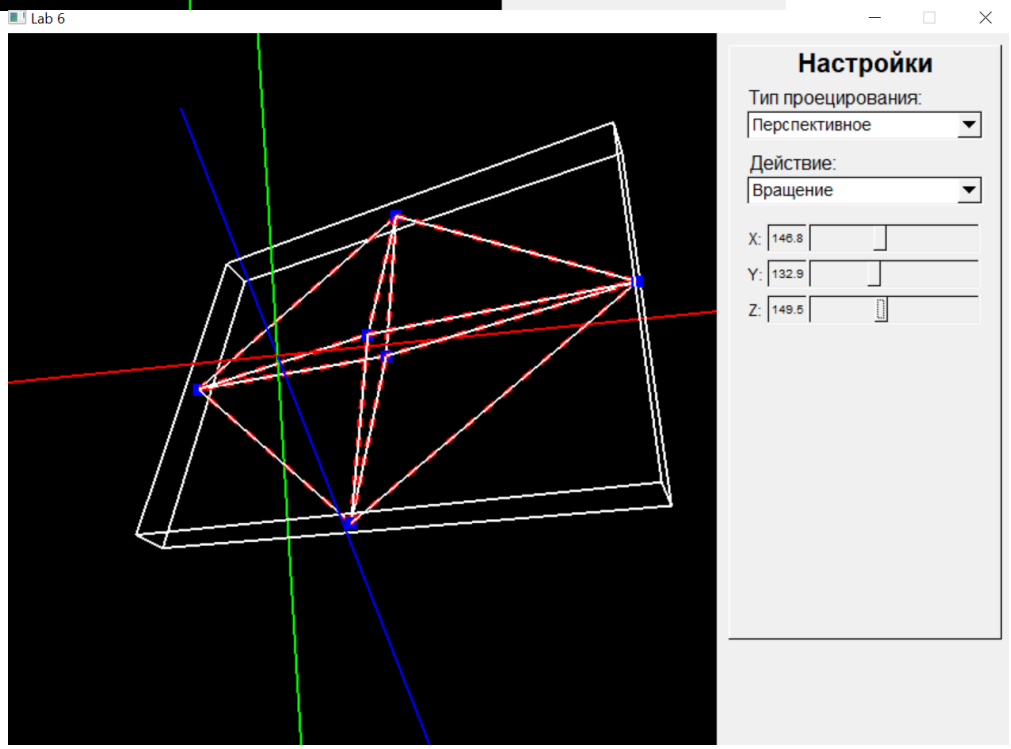
Хранение и изменение текущего состояния происходить с помощью класса промежуточного класса State.

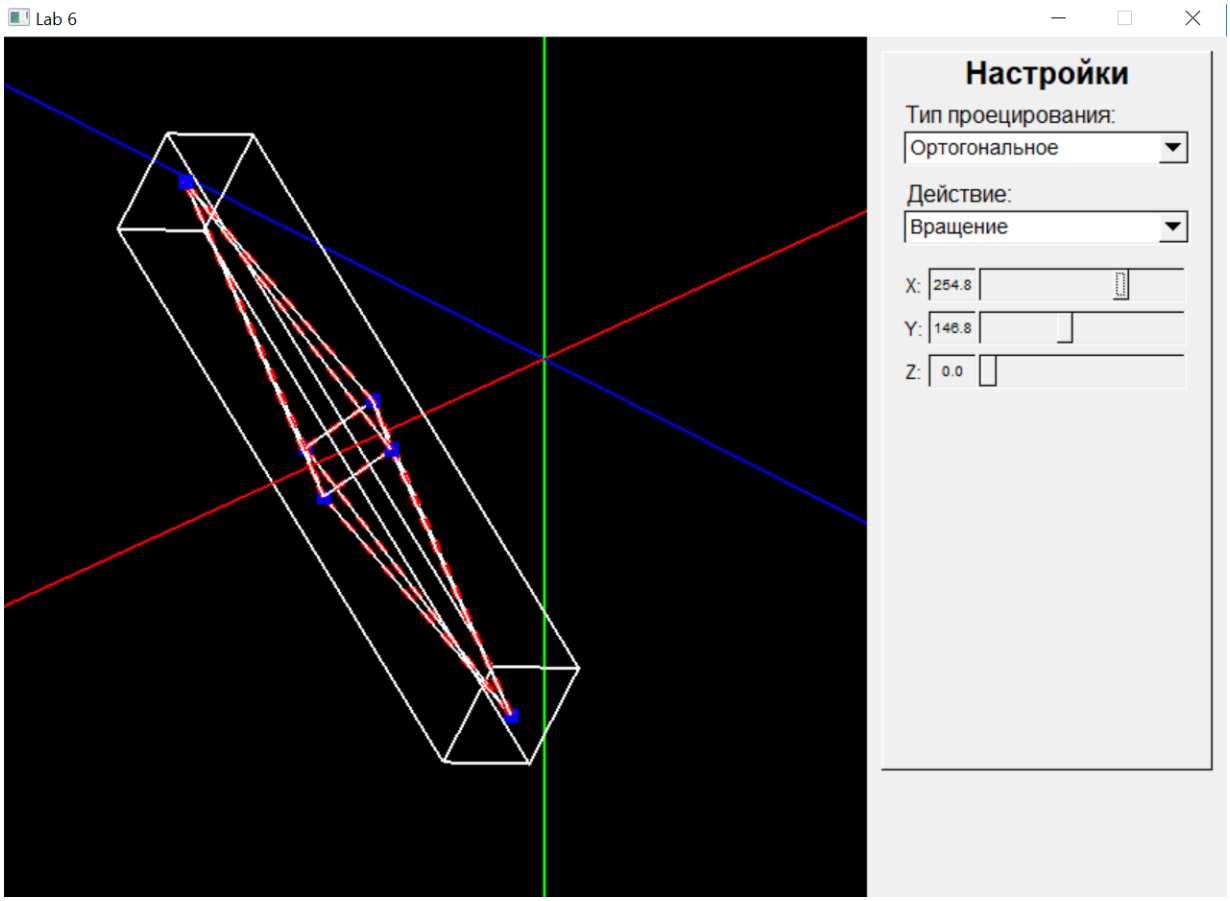
Тестирование.

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 лабораторной работы была разработана программа, моделирующая заданную объёмную фигуру. При выполнении работы были приобретены навыки работы с графическим конвейером и шейдернымм программам из графической библиотеки OpenGL.

Исходный код метода Draw:

```
    shaderProgramFigures.readAndCompile("Shaders/vertex_figures.shader",
"Shaders/fragment.shader");
    shaderProgramAxes.readAndCompile("Shaders/vertex_axes.shader",
"Shaders/fragment.shader");
    loadBuffers();
    Do_Movement();

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPointSize(10.0f);
    // glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
    shaderProgramFigures.Use();

    // Create transformations
    glm::mat4 model = glm::mat4(1.0f);
    auto [move_x, move_y, move_z] = statePtr->getXYZ(ActionType::translate);
    auto [angle_x, angle_y, angle_z] = statePtr->getXYZ(ActionType::rotate);
    auto [scale_x, scale_y, scale_z] = statePtr->getXYZ(ActionType::scale);
    model = glm::translate(model, glm::vec3(move_x, move_y, move_z));
    model = glm::rotate(model, glm::radians(angle_x), glm::vec3(1.0f, 0.0f,
0.0f));
    model = glm::rotate(model, glm::radians(angle_y), glm::vec3(0.0f, 1.0f,
0.0f));
    model = glm::rotate(model, glm::radians(angle_z), glm::vec3(0.0f, 0.0f,
1.0f));
    model = glm::scale(model, glm::vec3(scale_x, scale_y, scale_z));

    glm::mat4 view = glm::mat4(1.0f);
    view = camera.GetViewMatrix();

    glm::mat4 projection = glm::mat4(1.0f);
    if (statePtr->getProjectionType() == ProjectionType::perspective){
        projection       glm::perspective(camera.Zoom,   (GLfloat)screenWidth   /
(GLfloat)screenHeight, 0.1f, 50.0f);
    }
    if (statePtr->getProjectionType() == ProjectionType::orthogonal){
        projection = glm::ortho(-4.0f, 4.0f, -3.0f, 3.0f, 0.1f, 100.0f );
    }
    // Get their uniform location ans Pass them to the shaders
    glUniformMatrix4fv(glGetUniformLocation(shaderProgramFigures.Program,
"model"), 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(glGetUniformLocation(shaderProgramFigures.Program,
"view"), 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(glGetUniformLocation(shaderProgramFigures.Program,
"projection"), 1, GL_FALSE, glm::value_ptr(projection));
    glUniform3f(glGetUniformLocation(shaderProgramFigures.Program,
"figureColor"), 1.0, 1.0f, 1.0f);
```

```
    glLineWidth(2.0f);
    glBindVertexArray(VAO_cube);
    glDrawElements(GL_LINES, 4*6, GL_UNSIGNED_INT, 0);

    glLineWidth(2.0f);
    glBindVertexArray(VAO_octahedra);
    glDrawElements(GL_LINES, 3*2*6, GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);

    glEnable(GL_LINE_STIPPLE);
    glLineWidth(5.0f);
    glLineStipple(1, 0x00FF);
    glUniform3f(glGetUniformLocation(shaderProgramFigures.Program,
"figureColor"), 1.0, 0.0f, 0.0f);
    glBindVertexArray(VAO_octahedra);
    glDrawElements(GL_LINES, 3*2*6, GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
    glDisable(GL_LINE_STIPPLE);

    glUniform3f(glGetUniformLocation(shaderProgramFigures.Program,
"figureColor"), 0.0, 0.0f, 1.0f);
    glBindVertexArray(VAO_octahedra);
    glDrawElements(GL_POINTS, 3*2*6, GL_UNSIGNED_INT, 0);

    // Drawing axes
    shaderProgramAxes.Use();
    // Create transformations
    glm::mat4 modelAxe = glm::mat4(1.0f);
    modelAxe = glm::scale(modelAxe, glm::vec3(100, 100, 100));

    glm::mat4 viewAxe = glm::mat4(1.0f);
    viewAxe = camera.GetViewMatrix();

    glm::mat4 projectionAxe = glm::mat4(1.0f);
    if (statePtr->getProjectionType() == ProjectionType::perspective){
        projectionAxe = glm::perspective(camera.Zoom, (GLfloat)screenWidth /
(GLfloat)screenHeight, 0.1f, 50.0f);
    }
    if (statePtr->getProjectionType() == ProjectionType::orthogonal){
        projectionAxe = glm::ortho(-4.0f, 4.0f, -3.0f, 3.0f, 0.1f, 100.0f );
    }
    // Get their uniform location ans Pass them to the shaders
    glUniformMatrix4fv(glGetUniformLocation(shaderProgramAxes.Program,
"model"), 1, GL_FALSE, glm::value_ptr(modelAxe));
    glUniformMatrix4fv(glGetUniformLocation(shaderProgramAxes.Program,
"view"), 1, GL_FALSE, glm::value_ptr(viewAxe));
    glUniformMatrix4fv(glGetUniformLocation(shaderProgramAxes.Program,
"projection"), 1, GL_FALSE, glm::value_ptr(projectionAxe));
```

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