Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №4-6
по курсу «Компьютерная графика»

Студент:	Марков А.Н.
Группа:	М80-308Б-18
Преподаватель:	Филиппов Г.С.
Оценка:	
Дата:	

Москва
2020

1. Постановка задачи.

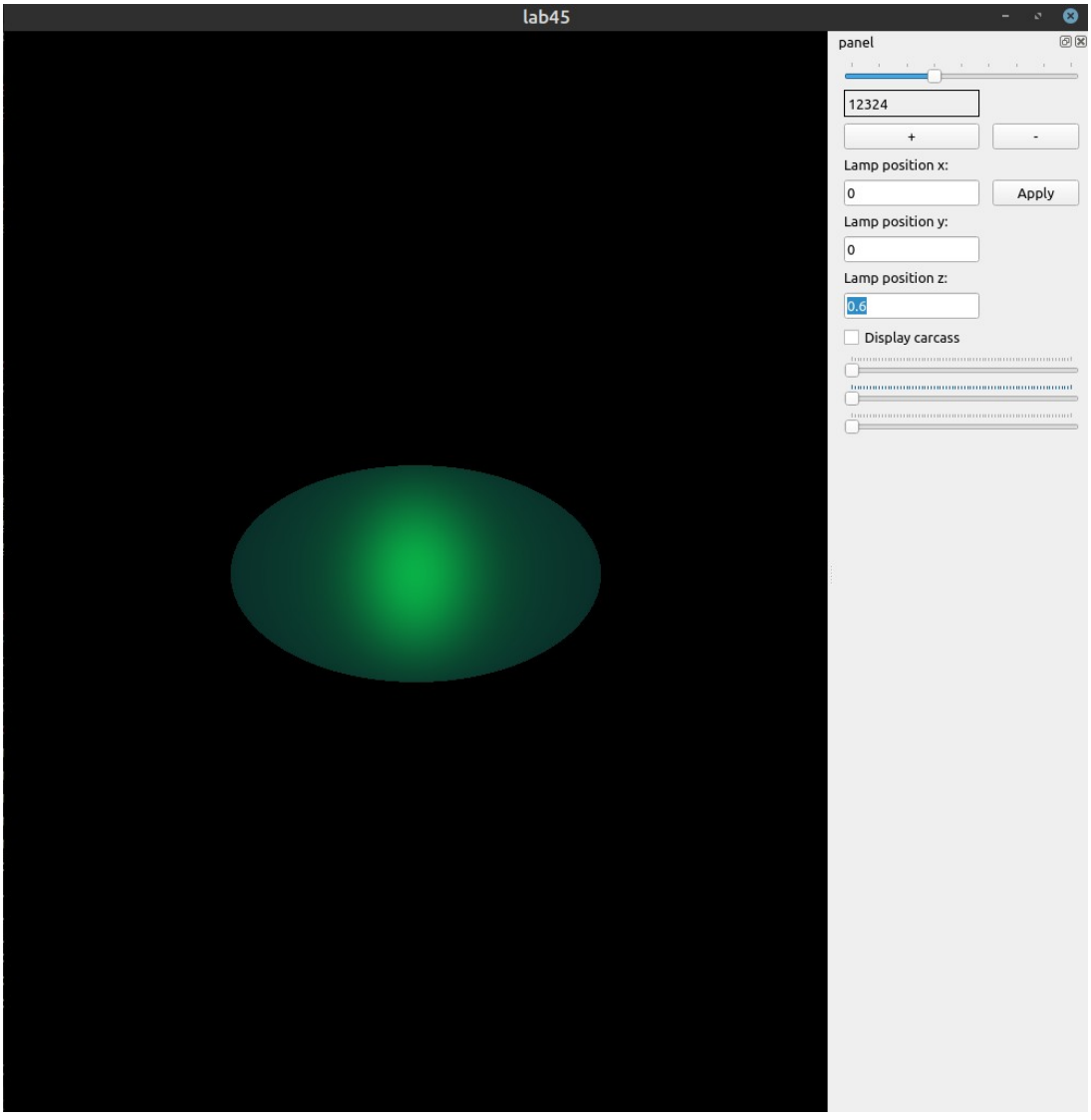
Создать графическое приложение с использованием OpenGL. Используя результаты Л.Р.№3, изобразить заданное тело (то же, что и в л.р. №3) с использованием средств OpenGL 2.1. Использовать буфер вершин. Точность аппроксимации тела задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель освещения на GLSL. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

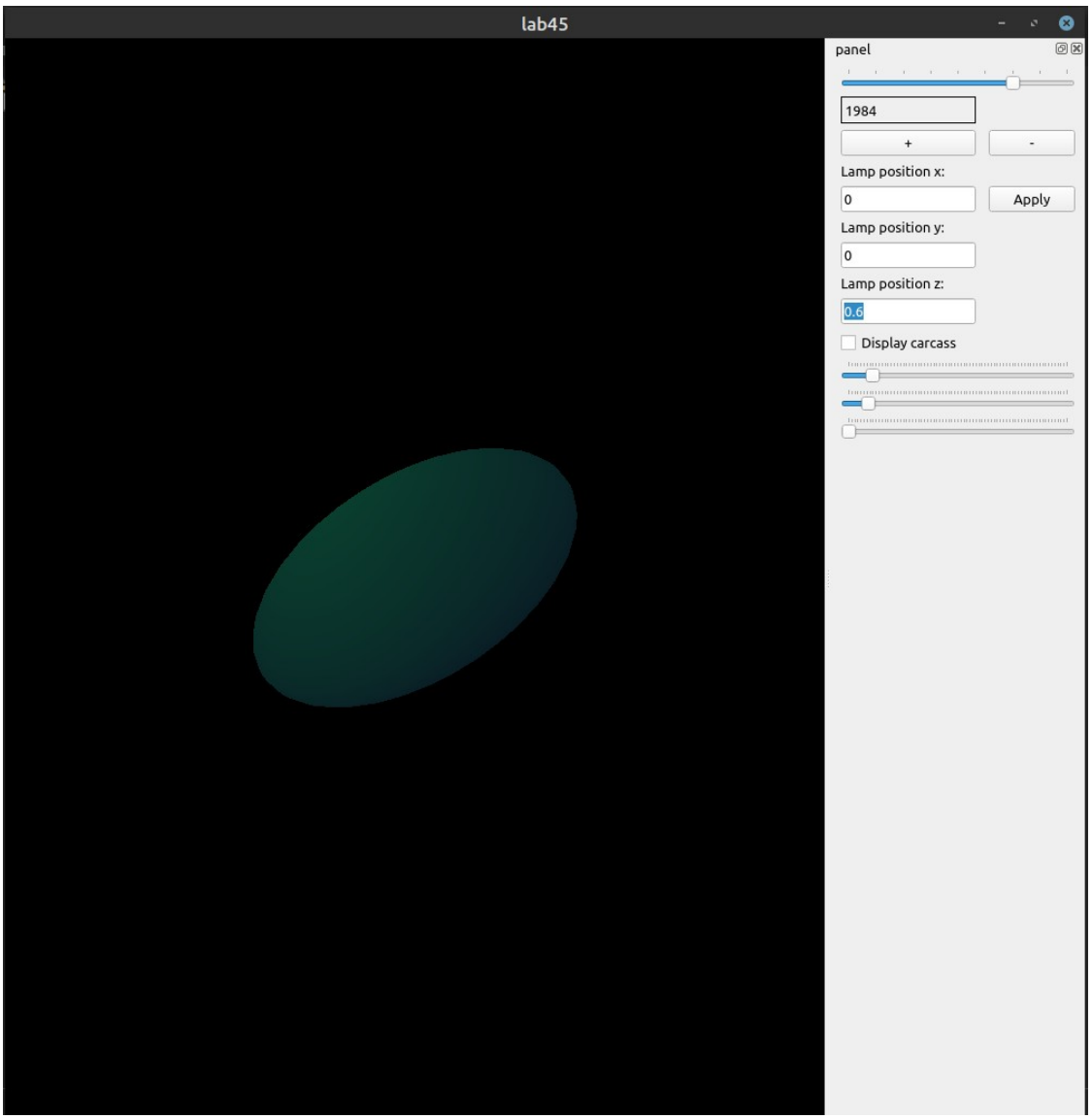
Вариант №5: Эллипсоид. Анимация. Координата X изменяется по закону X*cos(t), координата Y изменяется по закону Y = Ysin(X+t)

2. Решение задачи.

Фигура строится по аналогии с предыдущей лабораторной работой, только используя средства OpenGL. Здесь так же присутствует класс Polygon для хранения полигонов, класс glEllipsoid, представляющий фигуру эллипсоид. Такая фигура состоит из множества полигонов. Пользователь может задавать количество полигонов. Все преобразования для фигуры выполняются средствами OpenGL.

3. Демонстрация работы программы





4. Листинг программы.

```
void View::initializeGL() {
    initializeOpenGLFunctions();
    glClearColor(0.f, 0.f, 0.f, 1.f);

    glEnable(GL_DEPTH_TEST);
}

void View::resizeGL(int width, int height) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, width, height);
}

void View::paintGL() {
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    float a, b, c;
    a = fig.get_a();
    b = fig.get_b();
    c = fig.get_c();
    if (a < 1.f) {
        a = 1.;
    }
    if (b < 1.f) {
        b = 1.;
    }
    if (c < 1.f) {
        c = 1.;
    }
    glOrtho(-2. * static_cast<double>(a), 2. * static_cast<double>(a), -2. *
static_cast<double>(b),
            2. * static_cast<double>(b), -2. * static_cast<double>(c), 2. *
static_cast<double>(c));

    anim_draw();
    glDisable(GL_DEPTH_TEST);
}

void View::start_timer() {
    timer->start(100);
}

void View::stop_timer() {
    timer->stop();
}

void View::draw() {
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glRotatef(rotateX, 1.f, 0.f, 0.f);
    glRotatef(rotateY, 0.f, 1.f, 0.f);
    glRotatef(rotateZ, 0.f, 0.f, 1.f);
    glScalef(scale, scale, scale);

    if (displayCarcass) {
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

```
        glDisable(GL_LIGHTING);
    } else {
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        glEnable(GL_LIGHTING);
    }
    glEnable(GL_NORMALIZE);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,
fig.get_ambient_color());
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, fig.get_diffuse_color());
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,
fig.get_specular_color());
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, fig.get_shininess());

    float light_ambient[] = {0.f, 0.22f, 0.51f, 1.f};
    float light_diffuse[] = {0.f, 0.55f, 0.128f, 1.f};
    float light_specular[] = {0.f, 0.44f, 0.102f, 1.f};
    float light_position[] = {lightPositionX,
                              lightPositionY,
                              lightPositionZ, 1.f};

    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 128);
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.f);
//    glPopMatrix();
    glEnable(GL_CULL_FACE);
    glCullFace(GL_FRONT);

    glColor3f(1.f, 0.f, 0.f);
    for (auto polygon: fig.get_polygons()) {
        glBegin(GL_POLYGON);
            for (auto vertex: polygon.vertices) {
                glVertex3f(vertex.x(), vertex.y(), vertex.z());
            }
        glEnd();
    }
    glDisable(GL_CULL_FACE);
    glDisable(GL_LIGHT0);
    glDisable(GL_LIGHTING);
}

void View::anim_draw() {
```

```
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glRotatef(rotateX, 1.f, 0.f, 0.f);
    glRotatef(rotateY, 0.f, 1.f, 0.f);
    glRotatef(rotateZ, 0.f, 0.f, 1.f);
    glScalef(scale, scale, scale);

    if (displayCarcass) {
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        glDisable(GL_LIGHTING);
    } else {
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        glEnable(GL_LIGHTING);
    }

    glEnable(GL_NORMALIZE);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,
fig.get_ambient_color());
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, fig.get_diffuse_color());
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,
fig.get_specular_color());
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, fig.get_shininess());

    float light_ambient[] = {0.f, 0.22f, 0.51f, 1.f};
    float light_diffuse[] = {0.f, 0.55f, 0.128f, 1.f};
    float light_specular[] = {0.f, 0.44f, 0.102f, 1.f};
    float light_position[] = {lightPositionX,
                              lightPositionY,
                              lightPositionZ, 1.f};

    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 128);
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.f);

    glEnable(GL_CULL_FACE);
    glCullFace(GL_FRONT);

    glColor3f(1.f, 0.f, 0.f);
    for (auto polygon: fig.get_polygons()) {
        glBegin(GL_POLYGON);
            for (auto vertex: polygon.vertices) {
```

```
            glVertex3f(vertex.x() * static_cast<float>(cos(time)),
                    vertex.y() * static_cast<float>(sin(static_cast<double>(vertex.x()) +
time)),
                    vertex.z());
          }
       glEnd();
    }
    glDisable(GL_CULL_FACE);
    glDisable(GL_LIGHT0);
    glDisable(GL_LIGHTING);
}
```

5. Вывод.

В ходе выполнения данной лабораторной я получил опыт работы с OpenGL.