Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет: “Информационные технологии и прикладная математика”
Кафедра: 806 “Вычислительная математика и программирование”

Лабораторная работа №3,4,5

по курсу “Компьютерная графика”

Студент	Полей-Добронравова А.В.
Группа	М8О-307Б-18
Преподаватель	Г.С.Филиппов
Вариант	4
Дата	
Оценка	

Москва, 2020

Основы построения фотореалистичных изображений. Ознакомление с технологией OpenGL.

Задача:

(л.р. 3) Аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задаётся пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей.

(л.р. 4,5) Создать изображение с помощью OpenGL

Вариант 4

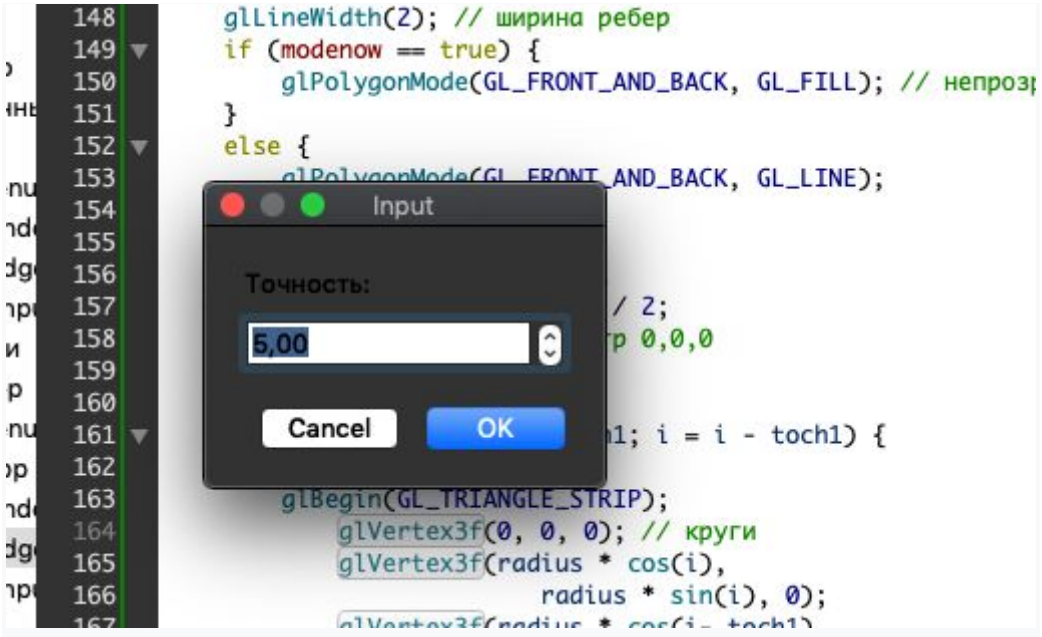
Полушарие

Описание

Программа написана на языке C++ в QTCreator, используется OpenGL. Создано два класса:

1) mainwindow - главное окно приложения;
2) myglwidget - виджет для отрисовки многогранника.

Аппроксимация передается пользователем через диалоговое окно в начале работы программы:



Под точностью аппроксимации полушария подразумевается количество ярусов боковин по оси Z (от экватора полушария до его полюса), нарисованных как четырехугольники; так же удвоенное число будет количеством треугольников днищ ярусов (аппроксимированных кругов). Более понятно это будет отображено в примерах работы.

в myglwidget.cpp аппроксимация вычисляетсявнутри MyGLWidget::draw() как

```
double toch1 = M_PI / toch;
```

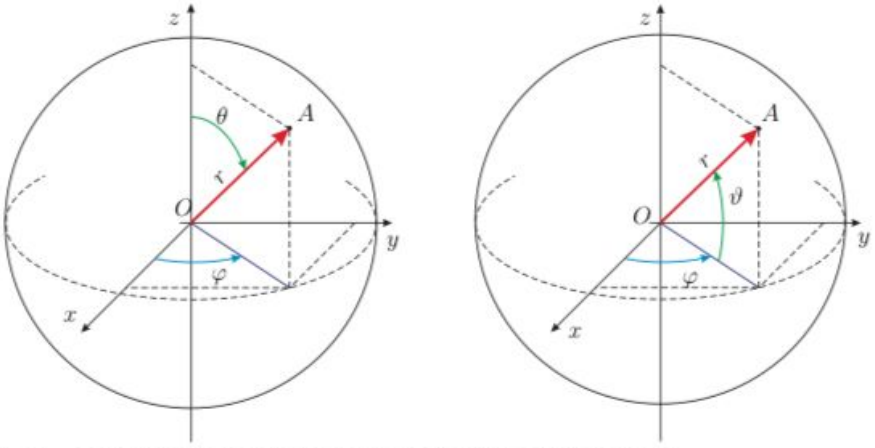
double toch2 = M_PI / toch / 2; где toch - число, задаваемое пользователем, toch1 - количество треугольников в основании, toch2 - количество ярусов по оси Z.
Далее идет отрисовка всех плоскостей многогранника.

```
for(i = 2 * M_PI; i >= toch1; i = i - toch1) {
      qglColor(Qt::red);
      glBegin(GL_TRIANGLE_STRIP);
         glVertex3f(0, 0, 0); // круги
         glVertex3f(radius * cos(i),
                    radius * sin(i), 0);
         glVertex3f(radius * cos(i- toch1),
                    radius * sin(i- toch1), 0);
      glEnd();
  } 
```

это нижнее основание полушария. Выбирается цвет ручки, и с режимом GL_TRIANGLE_STRIP рисуются связанные друг с другом треугольники.

Чтобы нарисовать другие части, проще всего использовать сферические координаты

3.5. **Сферическая система координат в пространстве.**



(r, θ, φ) — сферические координаты точки A. Формулы перехода:

$$\begin{cases} x = r\cos\varphi\sin\theta, \\ y = r\sin\varphi\sin\theta, \\ z = r\cos\theta. \end{cases}$$

Диапазоны изменения значений координат:

$$0 \leqslant r < +\infty,$$
$$0 \leqslant \theta \leqslant \pi,$$
$$0 \leqslant \varphi < 2\pi \pmod{2\pi}.$$

Географические координаты — вариант сферических.
(r, ϑ, φ) — географические координаты точки A. Формулы перехода:

$$\begin{cases} x = r\cos\varphi\cos\vartheta, \\ y = r\sin\varphi\cos\vartheta, \\ z = r\sin\vartheta. \end{cases}$$

Диапазоны изменения значений координат:

$$0 \leqslant r < +\infty, \qquad -\frac{\pi}{2} \leqslant \vartheta \leqslant \frac{\pi}{2}, \qquad 0 \leqslant \varphi < 2\pi \pmod{2\pi}.$$

(источник: http://matematika.phys.msu.ru/files/a_stud_gen/182/lect-1-print.pdf)

В этом цикле вычисляются значения точек для отрисовки, где шаг изменения каждого угла как раз определяется точностью аппроксимации:

```
for(i = 0; i <= 2 * M_PI-toch1; i = i + toch1) {
      for (h = 0; h <= M_PI_2; h = h + toch2) {
          }
}
```

Для отрисовки четырехугольников используется режим GL_POLYGON. Возникла проблема с незнанием того, что при отрисовке через OpenGL важен порядок точек, которые образуют фигуру для отрисовки. Порядок должен идти против часовой стрелки.
Так же у меня возникала проблема с тем, что если перед отрисовкой новой

фигуры я не объявляла заново цвет отрисовки через qglColor(Qt::red); у меня вообще не рисовалась эта фигура, или рисовалась неправильно.

В моей программе можно поменять режим отрисовки, нажав на кнопку “change mode”, чтобы увидеть контур всех отрисованных плоских фигур, а не заливку полностью.

Код

main.cpp

```
#include <QApplication>
#include <QDesktopWidget>

#include "mainwindow.h"

int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  MainWindow window;
  window.resize(window.sizeHint());
  window.show();
  return app.exec();
}
```

mainwindow.h

```
#ifndef MAIN_WINDOW_H
#define MAIN_WINDOW_H

#include <QWidget>
#include <QSlider>
#include <QMainWindow>
#include <QPushButton>
#include "myglwidget.h"

namespace Ui {
  class MainWindow;
}
class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  explicit MainWindow(QWidget *parent = 0);
  ~MainWindow();
private:
  Ui::MainWindow *ui;
  MyGLWidget* widget;
  QSlider *rotXSlider;
```

```
    QSlider *rotYSlider;
    QSlider *rotZSlider;
    QSlider * NSlider;
    QSlider *ScaleSlider;
    QPushButton* mode;
private slots:
    void sl1();
};
#endif
```

mainwindow.cpp

```
#include <QtWidgets>
#include <QGLWidget>
#include <QDebug>
#include <QInputDialog>
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "myglwidget.h"

QSlider* createSlider()
{
    QSlider *slider = new QSlider(Qt::Vertical);
    slider->setRange(0, 360 * 16);
    slider->setSingleStep(16);
    slider->setPageStep(15 * 16);
    slider->setTickInterval(15 * 16);
    slider->setTickPosition(QSlider::TicksRight);
    return slider;
}

QSlider* createSlider1()
{
    QSlider *slider = new QSlider(Qt::Vertical);
    slider->setRange(6, 36);
    slider->setSingleStep(1);
    slider->setPageStep(1);
    slider->setTickInterval(1);
    slider->setTickPosition(QSlider::TicksRight);
    return slider;
}

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    widget = new MyGLWidget();

    rotXSlider = createSlider();
    rotYSlider = createSlider();
```

```
    rotZSlider = createSlider();
    ScaleSlider = createSlider1();

    connect(widget, SIGNAL(xRotationChanged(int)), rotXSlider, SLOT(setValue(int)));
    connect(rotXSlider, SIGNAL(valueChanged(int)), widget, SLOT(setXRotation(int)));

    connect(widget, SIGNAL(yRotationChanged(int)), rotYSlider, SLOT(setValue(int)));
    connect(rotYSlider, SIGNAL(valueChanged(int)), widget, SLOT(setYRotation(int)));

    connect(widget, SIGNAL(zRotationChanged(int)), rotZSlider, SLOT(setValue(int)));
    connect(rotZSlider, SIGNAL(valueChanged(int)), widget, SLOT(setZRotation(int)));

    connect(widget, SIGNAL(ScaleChanged(int)), ScaleSlider, SLOT(setValue(int)));
    connect(ScaleSlider, SIGNAL(valueChanged(int)), widget, SLOT(setScale(int)));
    QGridLayout *mainLayout = new QGridLayout;
    QWidget *centralWidget = new QWidget(this);

    mode = new QPushButton("change mode");
    connect(mode, SIGNAL(clicked()), SLOT(sl1()));

    bool ok;
    widget->toch = QInputDialog::getDouble(this, tr("Input"),
                                           tr("Точность:"), 5, -10000, 10000, 2, &ok,
                                           Qt::WindowFlags());
        mainLayout->addWidget(widget);
        mainLayout->addWidget(mode,2,0);
        mainLayout->addWidget(rotXSlider, 0, 1);
        mainLayout->addWidget(rotYSlider,0,2);
        mainLayout->addWidget(rotZSlider,0,3);

        //mainLayout->addWidget(NSlider);
        mainLayout->addWidget(ScaleSlider,0,4);
        centralWidget->setLayout(mainLayout);
        setCentralWidget(centralWidget);
}

void MainWindow::sl1() {
    if (widget->modenow == true) {
        widget->modenow = false;
    }
    else {
        widget->modenow = true;
    }
    widget->updateGL();
}

MainWindow::~MainWindow()
{
    delete rotXSlider;
    delete rotYSlider;
```

```
    delete rotZSlider;
    delete NSlider;
    delete ScaleSlider;
    delete widget;
}
```

myglwigjet.h

```
#ifndef MYGLWIDGET_H
#define MYGLWIDGET_H

#include <QGLWidget>

class MyGLWidget : public QGLWidget
{
    Q_OBJECT
public:
    explicit MyGLWidget(QWidget *parent = 0);
    bool modenow;
    double scale;
    double toch;
    void draw();
    void resizeGL(int width, int height);
protected:
    void initializeGL();
    void paintGL();
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *event);

public slots:
    // slots for xyz-rotation slider
    void setXRotation(int angle);
    void setYRotation(int angle);
    void setZRotation(int angle);
    //void setN(int value);
    void setScale(int value);

signals:
    // signaling rotation from mouse movement
    void xRotationChanged(int angle);
    void yRotationChanged(int angle);
    void zRotationChanged(int angle);
    //void NChanged(int value);
    void ScaleChanged(int value);
    void clicked();

private:
    QSize minimumSizeHint() const;
    QSize sizeHint() const;
    int xRot;
```

```
    int yRot;
    int zRot;
    //int N;
    QPoint lastPos;
};

#endif
```

myglwigjet.cpp

```
#include <QtWidgets>
#include <QtOpenGL>
#include <iostream>
#include <QDebug>
#include <QtMath>

#include "myglwidget.h"
#include "light.h"

MyGLWidget::MyGLWidget(QWidget *parent)
    : QGLWidget(QGLFormat(QGL::SampleBuffers), parent)
{
    // начальные параметры
    xRot = yRot = zRot = 0;
    modenow = true;
    scale = 0.06;
}

QSize MyGLWidget::minimumSizeHint() const
{
    return QSize(50, 50);
}

QSize MyGLWidget::sizeHint() const
{
    return QSize(400, 400);
}

// обнуление периода
static void qNormalizeAngle(int &angle)
{
    while (angle < 0)
        angle += 360;
    while (angle > 360)
        angle -= 360;
}

// поворот меша на угол angle, относительно оси oX
void MyGLWidget::setXRotation(int angle)
{
```

```
    qNormalizeAngle(angle);
    if (angle != xRot) {
        xRot = angle;
        emit xRotationChanged(angle);
        updateGL();
    }
}

// поворот меша на угол angle, относительно оси oY
void MyGLWidget::setYRotation(int angle)
{
    qNormalizeAngle(angle);
    if (angle != yRot) {
        yRot = angle;
        emit yRotationChanged(angle);
        updateGL();
    }
}

// поворот меша на угол angle, относительно оси oZ
void MyGLWidget::setZRotation(int angle)
{
    qNormalizeAngle(angle);
    if (angle != zRot) {
        zRot = angle;
        emit zRotationChanged(angle);
        updateGL();
    }
}

// задание масштаба
void MyGLWidget::setScale(int value)
{
    scale = value / 100.0;
    emit ScaleChanged(value);
    updateGL();
}

// инициализация OpenGL
void MyGLWidget::initializeGL()
{
    qglClearColor(Qt::black);

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glShadeModel(GL_SMOOTH);
}

// функция отрисовки
void MyGLWidget::paintGL()
```

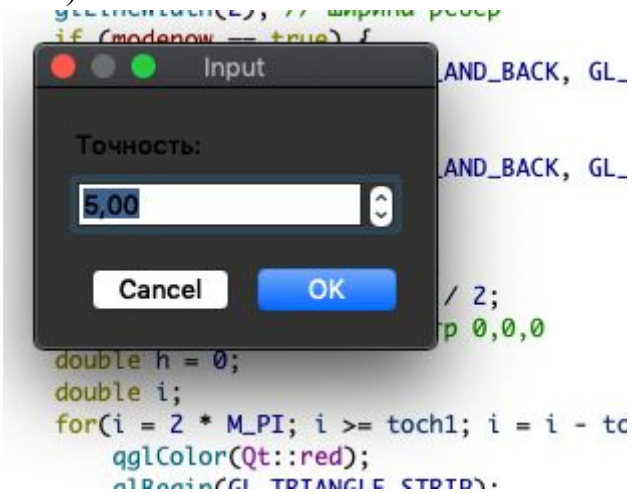
```
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glTranslatef(0.0, 0.0, -10.0);
  glRotatef(xRot, 1.0, 0.0, 0.0);
  glRotatef(yRot, 0.0, 1.0, 0.0);
  glRotatef(zRot, 0.0, 0.0, 1.0);
  glScaled(scale,scale,scale);
  draw();
}

// параметры окна отрисовки
void MyGLWidget::resizeGL(int width, int height)
{
  int side = qMin(width, height);
  glViewport((width - side) / 2, (height - side) / 2, side, side);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
#ifdef QT_OPENGL_ES_1
  glOrthof(-2, +2, -2, +2, 1.0, 15.0);
#else
  glOrtho(-2, +2, -2, +2, 1.0, 15.0);
#endif
  glMatrixMode(GL_MODELVIEW);
}

// нажатие на кнопки мыши
void MyGLWidget::mousePressEvent(QMouseEvent *event)
{
  lastPos = event->pos();
}

// перемещение мыши
void MyGLWidget::mouseMoveEvent(QMouseEvent *event)
{
  int dx = event->x() - lastPos.x();
  int dy = event->y() - lastPos.y();

  if (event->buttons() & Qt::LeftButton) {
     setXRotation(xRot + dy);
     setYRotation(yRot + dx);
  }
  if (event->buttons() & Qt::RightButton) {
     setXRotation(xRot + dy);
     setZRotation(zRot + dx);
  }
  lastPos = event->pos();
}
```

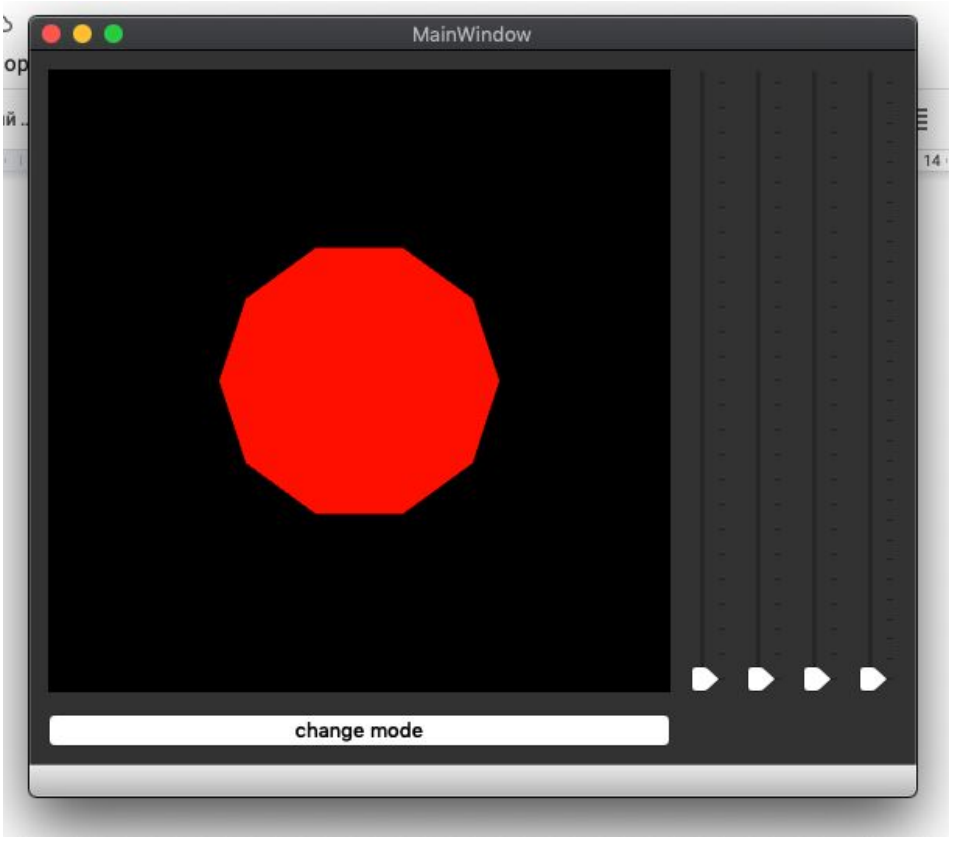
```
void MyGLWidget::draw()
{
  glLineWidth(2); // ширина ребер
  if (modenow == true) {
     glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); // непрозрачность граней
  }
  else {
     glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  }

  double toch1 = M_PI / toch;
  double toch2 = M_PI / toch / 2;
  double radius = 15; // центр 0,0,0
  double h = 0;
  double i;
  for(i = 2 * M_PI; i >= toch1; i = i - toch1) {
     qglColor(Qt::red);
     glBegin(GL_TRIANGLE_STRIP);
        glVertex3f(0, 0, 0); // круги
        glVertex3f(radius * cos(i),
                   radius * sin(i), 0);
        glVertex3f(radius * cos(i- toch1),
                   radius * sin(i- toch1), 0);
     glEnd();
  }
  for(i = 0; i <= 2 * M_PI-toch1; i = i + toch1) {
     for (h = 0; h <= M_PI_2; h = h + toch2) {
           if (h+ toch2 <= M_PI_2) { //края
              qglColor(Qt::red);
              glBegin(GL_POLYGON);
              glVertex3f(radius * cos(i) * cos(h),
                         radius * sin(i) * cos(h), radius * sin(h));
              glVertex3f(radius * cos(i+ toch1) * cos(h),
                         radius * sin(i+ toch1) * cos(h), radius * sin(h));
              glVertex3f(radius * cos(i+ toch1) * cos(h+ toch2),
                         radius * sin(i+ toch1) * cos(h+ toch2), radius * sin(h+ toch2));
              glVertex3f(radius * cos(i) * cos(h+ toch2),
                         radius * sin(i) * cos(h+ toch2), radius * sin(h+ toch2));
              glEnd();
           }
           else {
               qglColor(Qt::red);
               glBegin(GL_TRIANGLES);
               glVertex3f(0,0, radius * sin(h+toch2));
               glVertex3f(radius * cos(i) * cos(h),
                          radius * sin(i) * cos(h), radius * sin(h));
               glVertex3f(radius * cos(i+ toch1) * cos(h),
                          radius * sin(i+ toch1) * cos(h), radius * sin(h));
               glEnd();
           }
```

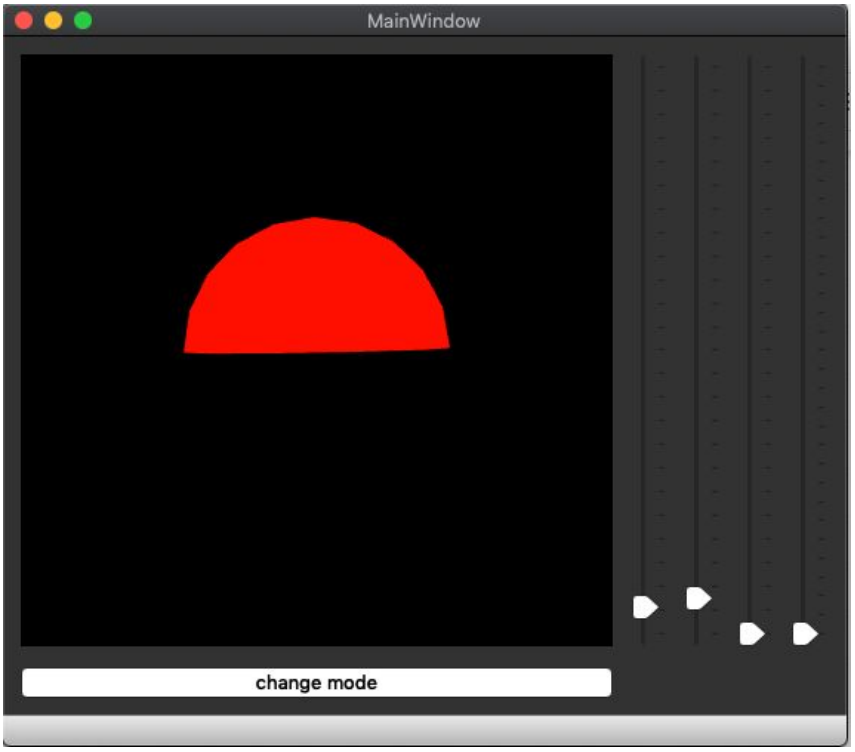
```
    }
  }
}
```

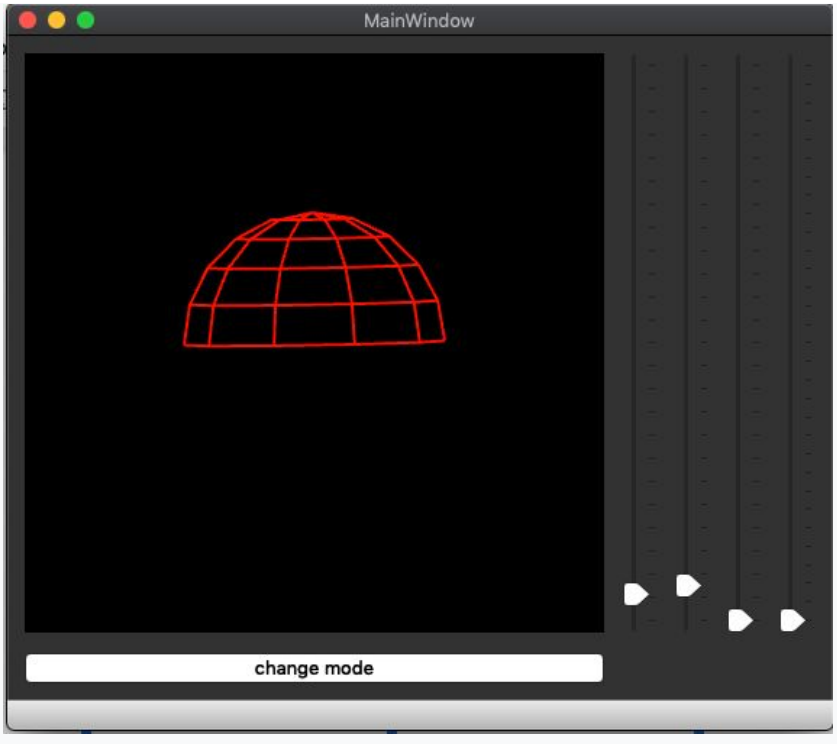
Пример работы

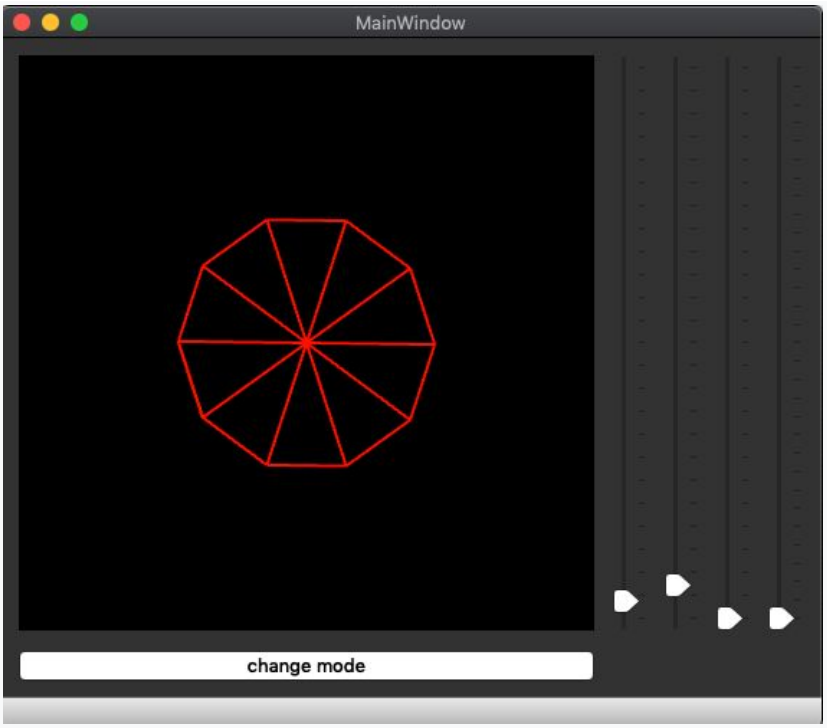
1) точность = 5





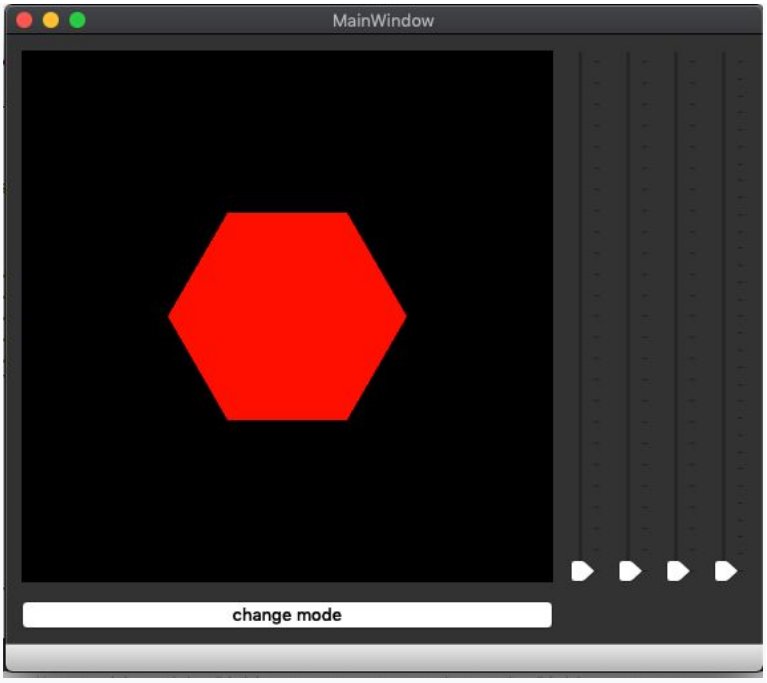

MainWindow
change mode

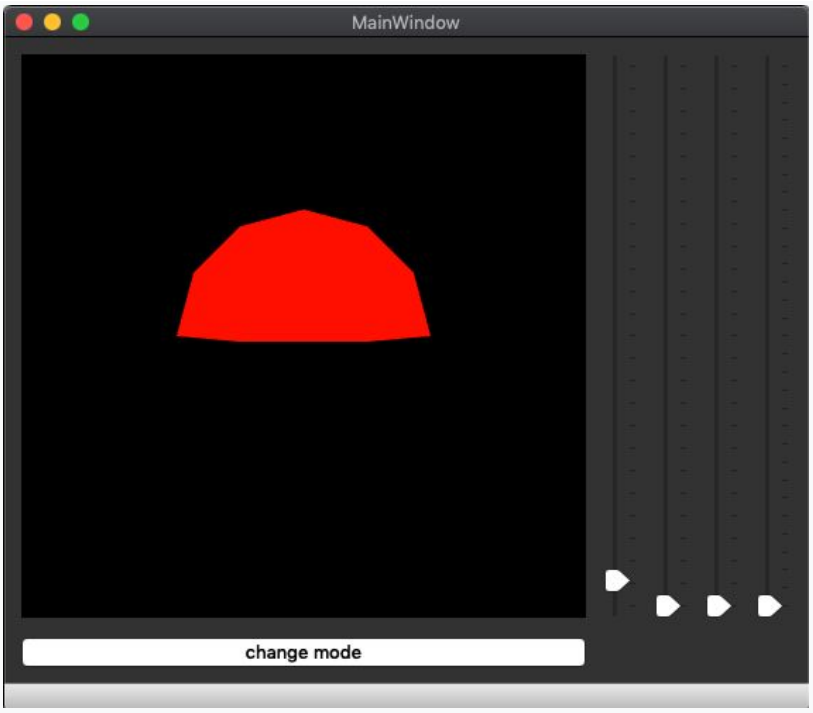

MainWindow
change mode

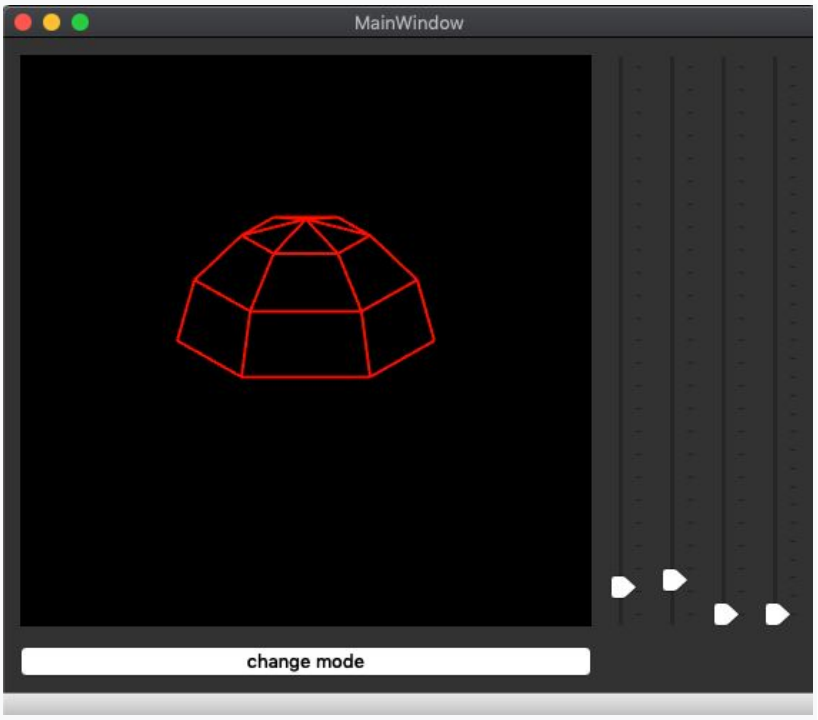


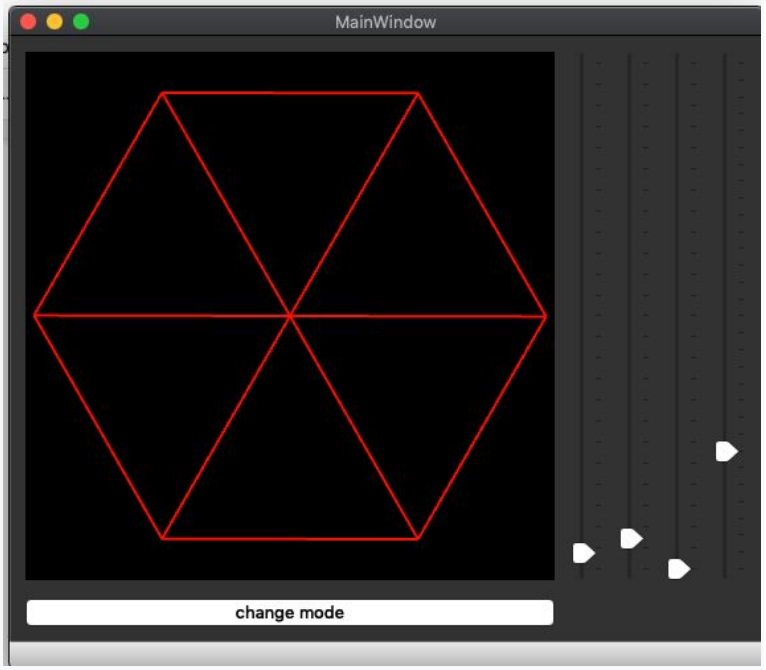
В основании 10 треугольников, 5 ярусов сбоку.

2) точность = 3




MainWindow
change mode


MainWindow
change mode



Вывод

Так как основной графический примитив - точка, аппроксимация объектов вручную помогает понимать, каким образом отрисовывается не аппроксимированный объект. Вернее, у “обычного” шара или полушария, например, это точность аппроксимации такого значения, что глаз не видит прямых линий, и считает фигуру округлой.