**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 3**

Тема: Основы построения

фотореалистичных изображений.

Студент: Айрапетова Евгения Ашотовна

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

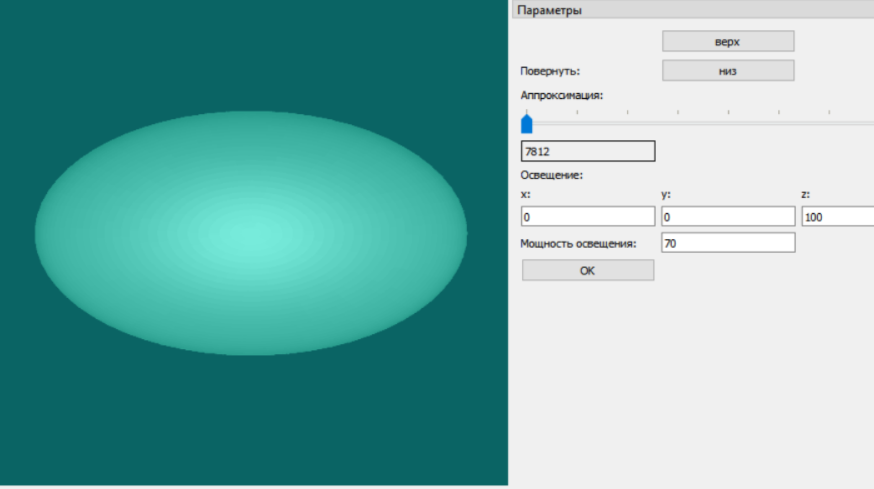
Используя результаты ЛР2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

1. Описание программы

Для решения задачи реализованы классы для построения эллипсоида, вычисления и хранения координат вершин фигуры аппроксимации, а также для хранения координат лампы и для боковой панели.

Для отрисовки многоугольника используется массив координат вершин, цвет одного многоугольника рассчитывается, как сумма базового цвета и параметров освещения в данной области.

1. Результаты выполнения программы



1. Листинг программы

Pyramid.h:

#pragma once

#include <QOpenGLWidget>

// ======================================================================

class MyPyramid : public QOpenGLWidget {

private:

GLuint m\_nPyramid;

GLfloat m\_xRotate;

GLfloat m\_yRotate;

QPoint m\_ptPosition;

protected:

virtual void initializeGL ( );

virtual void resizeGL (int nWidth, int nHeight);

virtual void paintGL ( );

virtual void mousePressEvent(QMouseEvent\* pe );

virtual void mouseMoveEvent (QMouseEvent\* pe );

GLuint createPyramid (GLfloat fSize = 1.0f );

public:

MyPyramid(QWidget\* pwgt = 0);

};

// ======================================================================

class MyPyramidProjection : public QOpenGLWidget {

private:

GLuint m\_nPyramid;

GLfloat m\_xRotate;

GLfloat m\_yRotate;

QPoint m\_ptPosition;

protected:

virtual void initializeGL ( );

virtual void resizeGL (int nWidth, int nHeight);

virtual void paintGL ( );

virtual void mousePressEvent(QMouseEvent\* pe );

virtual void mouseMoveEvent (QMouseEvent\* pe );

public:

MyPyramidProjection(QWidget\* pwgt = 0);

void draw1(int xOffset, int yOffset);

};

Pyramid.cpp:

#include <QOpenGLFunctions>

#include <QtGui>

#include <QVector>

#include <QPointF>

#include "OGLPyramid.h"

// ----------------------------------------------------------------------

MyPyramid::MyPyramid(QWidget\* pwgt/\*= 0\*/) : QOpenGLWidget(pwgt)

, m\_xRotate(0)

, m\_yRotate(0)

{

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramid::initializeGL()

{

QOpenGLFunctions\* pFunc =

QOpenGLContext::currentContext()->functions();

pFunc->glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

pFunc->glEnable(GL\_DEPTH\_TEST);

glShadeModel(GL\_FLAT);

m\_nPyramid = createPyramid(1.2f);

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramid::resizeGL(int nWidth, int nHeight)

{

glViewport(0, 0, (GLint)nWidth, (GLint)nHeight);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glFrustum(-1.0, 1.0, -1.0, 1.0, 1.0, 10.0);

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramid::paintGL()

{

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

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0, 0.0, -3.0);

glRotatef(m\_xRotate, 1.0, 0.0, 0.0);

glRotatef(m\_yRotate, 0.0, 1.0, 0.0);

glCallList(m\_nPyramid);

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramid::mousePressEvent(QMouseEvent\* pe)

{

m\_ptPosition = pe->pos();

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramid::mouseMoveEvent(QMouseEvent\* pe)

{

m\_xRotate += 180 \* (GLfloat)(pe->y() - m\_ptPosition.y()) / height();

m\_yRotate += 180 \* (GLfloat)(pe->x() - m\_ptPosition.x()) / width();

update();

m\_ptPosition = pe->pos();

}

// ----------------------------------------------------------------------

GLuint MyPyramid::createPyramid(GLfloat fSize/\*=1.0f\*/)

{

int k = 8;

QVector<QPointF> A(k);

GLuint n = glGenLists(1);

glNewList(n, GL\_COMPILE);

glBegin(GL\_POLYGON);

glColor4f(1.0f, 0.0f, 0.0f, 1.0f);

for(int i = 0; i<k; ++i){

float fAngle = 2 \* 3.14 \* i / k;

A[i].setX(fSize\*cos(fAngle));

A[i].setY(fSize\*sin(fAngle));

glVertex3f(A[i].x(), A[i].y(), 0);

}

glEnd();

glBegin(GL\_QUADS);

float r,g,b,a;

r = 0.2;

b = 0;

g = 1;

a = 1;

for (int i = 0; i< k-1; ++i){

glColor4f(r, g, b, a);

glVertex3f(A[i].x(), A[i].y(), 0.0);

glVertex3f(A[i].x()\*0.6, A[i].y()\*0.6, fSize);

glVertex3f(A[i+1].x()\*0.6, A[i+1].y()\*0.6, fSize);

glVertex3f(A[i+1].x(), A[i+1].y(), 0.0);

r+=0.3;

g-=0.2;

b+=0.3;

a-=0.1;

}

glColor4f(0.0f, 0.3f, 1.0f, 0.6f);

glVertex3f(A[7].x(), A[7].y(), 0.0);

glVertex3f(A[7].x()\*0.6, A[7].y()\*0.6, fSize);

glVertex3f(A[0].x()\*0.6, A[0].y()\*0.6, fSize);

glVertex3f(A[0].x(), A[0].y(), 0.0);

glEnd();

glBegin(GL\_POLYGON);

glColor4f(0.0f, 0.5f, 0.5f, 0.5f);

for(int i = 0; i<k; ++i){

glVertex3f(A[i].x()\*0.6, A[i].y()\*0.6, fSize);

}

glEnd();

glEndList();

return n;

}

// ----------------------------------------------------------------------

// ----------------------------------------------------------------------

// ----------------------------------------------------------------------

MyPyramidProjection::MyPyramidProjection(QWidget\* pwgt/\*= 0\*/) : QOpenGLWidget(pwgt)

, m\_xRotate(0)

, m\_yRotate(0)

{

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramidProjection::initializeGL()

{

QOpenGLFunctions\* pFunc =

QOpenGLContext::currentContext()->functions();

pFunc->glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramidProjection::resizeGL(int nWidth, int nHeight)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glViewport(0, 0, (GLint)nWidth, (GLint)nHeight);

glOrtho(0, 400, 200, 0, -1, 1);

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramidProjection::paintGL()

{

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

draw1(0, 40);

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramidProjection::mousePressEvent(QMouseEvent\* pe)

{

m\_ptPosition = pe->pos();

}

// ----------------------------------------------------------------------

/\*virtual\*/void MyPyramidProjection::mouseMoveEvent(QMouseEvent\* pe)

{

m\_xRotate += 180 \* (GLfloat)(pe->y() - m\_ptPosition.y()) / height();

m\_yRotate += 180 \* (GLfloat)(pe->x() - m\_ptPosition.x()) / width();

update();

m\_ptPosition = pe->pos();

}

void MyPyramidProjection::draw1(int xOffset, int yOffset) //offset - координаты, с которых начать отрисовку

{

int n = 8;

glPointSize(2);

glBegin(GL\_POLYGON);

glColor3f(1, 1, 1);

for (int i = 0; i < n; ++i) {

float fAngle = 2 \* 3.14 \* i / n;

int x = (int)(50 + cos(fAngle) \* 40 + xOffset);

int y = (int)(50 + sin(fAngle) \* 40 + yOffset);

glVertex2f(x, y);

}

glEnd();

xOffset+=100;

yOffset = 70;

glBegin(GL\_QUADS);

glColor3f(1, 1, 1);

glVertex2f(60+xOffset, yOffset\*2);

glVertex2f(xOffset+60\*1.6, yOffset);

glVertex2f(xOffset+60\*1.6 +xOffset, yOffset);

glVertex2f(2\*xOffset+60\*1.6 +30\*1.6, yOffset\*2);

glEnd();

}

main.cpp:

#include <QApplication>

#include "OGLPyramid.h"

// ----------------------------------------------------------------------

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

{

QApplication app(argc, argv);

MyPyramid MyPyramid;

MyPyramid.resize(200, 200);

MyPyramid.setWindowTitle("Изометрическая проекция");

MyPyramid.show();

MyPyramidProjection Projection;

Projection.setWindowTitle("Ортографические проекции");

Projection.resize(400, 200);

Projection.show();

return app.exec();

}

ВЫВОД

В процессе выполнения данной ЛР я узнала как работать с телами, а также как вводить их аппроксимацию и работать с освещением.

ЛИТЕРАТУРА

Документация по Qt: https://doc.qt.io (дата обращения 05.12.2021).