МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №5

по дисциплине: Компьютерная графика

тема: «Алгоритмы удаления невидимых поверхностей»

Выполнил: ст. группы ВТ-32

Воскобойников И. С.

Проверил: Гибкин Ю. С.

Белгород 2020 г.

Цель работы: получение навыков использования аффинных преобразований в пространстве и создание графического приложения с использованием GDI в среде Qt Creator для визуализации простейших трёхмерных объектов.

Порядок выполнения работы

1. Разработать алгоритм и составить программу для построения на экране трёхмерных изображений в соответствии с номером варианта. В качестве исходных данных взять указанные в таблице №1.

Требования к программе

1. Окно поделить на 4 части одинаковые части:
   1. На верхней левой части должна отображаться фронтальная проекция (вид спереди);
   2. Правая верхняя часть – профильная проекция (вид сбоку);
   3. Левая нижняя часть должна отображать вид сверху (горизонтальную проекцию);
   4. На правой нижней части должна отображаться проекция, вид которой выбирает пользователь: центральная, косоугольная кабинетная, косоугольная свободная, параллельная, ортографическая.

widget.h

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QWidget>

#include "objects3d.h"

#include "amorph3d.h"

#include <QPainter>

#include <QKeyEvent>

namespace Ui {

class Widget;

}

class Widget : public QWidget

{

Q\_OBJECT

public:

double angleX;

double angleY;

double angleZ;

int mouseWheel;

int projection;

ObjObject objObject;

Amorph3D amorph3d;

QPoint mousePos;

QMatrix4x4 rotateMatrix;

explicit Widget(QWidget \*parent = 0);

QPoint selectP;

int c;

~*Widget*();

private slots:

void on\_spinBox\_valueChanged(int arg1);

void on\_verticalSlider\_sliderMoved(int position);

private:

Ui::Widget \*ui;

void *paintEvent*(QPaintEvent \*event);

void *wheelEvent*(QWheelEvent\* );

void *mouseMoveEvent*(QMouseEvent \*mouseEvent);

void *mouseReleaseEvent*(QMouseEvent \*event);

};

#endif // WIDGET\_H

widget.cpp

#include "widget.h"

#include "ui\_widget.h"

#include <QDebug>

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Widget)

{

mousePos = QPoint(0, 0);

ui->setupUi(this);

rotateMatrix = QMatrix4x4(

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

amorph3d = Amorph3D();

objObject = ObjObject(false);

mouseWheel = -1;

selectP = QPoint(0, 0);

c = 5;

}

Widget::~*Widget*()

{

delete ui;

}

void Widget::*paintEvent*(QPaintEvent \*event){

QPainter painter(this);

int coef = ((std::min(width()/4, height()/4))) - 20;

QVector<QVector3D> vertecs = QVector<QVector3D>();

for (int i = 0; i < objObject.points.size(); i++){

QVector3D tmp\_v = objObject.points[i];

tmp\_v = tmp\_v \* rotateMatrix;

tmp\_v += QVector3D(0, 0, mouseWheel);

vertecs.append(tmp\_v);

}

QColor selectColor = QColor(255, 125, 125);

QVector<QPair<QVector<QVector3D>, QColor>> polysBeforeModify = objObject.getSortedPolygons(vertecs);

QVector<QPair<QPolygon, QColor>> newPolygons = QVector<QPair<QPolygon, QColor>>();

bool fst = true;

for (int j = polysBeforeModify.size() - 1; j >= 0; j--){

QPolygon poly = QPolygon();

QColor col = polysBeforeModify[j].second;

for (int i=0; i < polysBeforeModify[j].first.size(); i++){

QVector3D v = amorph3d.centralProj(polysBeforeModify[j].first[i], c);

v = amorph3d.extention(v, coef, coef, coef);

v = amorph3d.translate(v, width()/2, height()/2, 0);

poly.append(v.toPoint());

}

if (fst && objObject.pointInPoly(poly, selectP)){ col = selectColor; fst = false;}

newPolygons.push\_front(QPair<QPolygon, QColor>(poly, col));

}

for (int j = 0; j < newPolygons.size(); j++){

painter.setBrush(newPolygons[j].second);

painter.drawPolygon(newPolygons[j].first);

}

vertecs.clear();

}

// Обработчик события прокрутки колеса мыши

void Widget::*wheelEvent*(QWheelEvent\* wheelevent){

if (wheelevent->delta() < 0) mouseWheel += 1;

else mouseWheel -= 1;

if (mouseWheel > 1) mouseWheel = 1;

repaint(); // Обновляем окно

}

void Widget::*mouseMoveEvent*(QMouseEvent \*mouseEvent)

{

if (mouseEvent->buttons() == Qt::RightButton){

QPoint dp = mouseEvent->pos() - mousePos;

rotateMatrix.rotate(dp.y(), 1, 0, 0);

rotateMatrix.rotate(-dp.x(), 0, 1, 0);

mousePos = mouseEvent->pos();

repaint();

}

}

void Widget::*mouseReleaseEvent*(QMouseEvent \*event){

if(event->button() == Qt::LeftButton){

QPoint dp = event->pos();

QPoint windowP = dp - QPoint(width()/2, height()/2);

if ((abs(windowP.x()) < width()/2) && (abs(windowP.y()) < height()/2)){

selectP = dp;

repaint();

}

}

}

void Widget::on\_spinBox\_valueChanged(int arg1)

{

objObject.rebuildObject(arg1);

repaint();

}

void Widget::on\_verticalSlider\_sliderMoved(int position)

{

c = position;

repaint();

}

objects3d.h

#ifndef OBJECTS3D\_H

#define OBJECTS3D\_H

#include <QString>

#include <QVector>

#include <QVector3D>

#include <QPolygon>

#include <QFile>

#include <QTextStream>

#include <QDebug>

#include <QtAlgorithms>

#include <QPair>

#include <QColor>

#include <amorph3d.h>

class ObjFilesReader {

public:

// сбор списка типа, соответствующего метке из строк файла fileName с меткой mark

template <class T>

static QVector<QVector<T>> parseByMark(QString fileName, bool is\_v=true) {

QFile fileObj(fileName);

fileObj.*open*(QIODevice::ReadOnly);

char mark = 'f';

if (is\_v) mark = 'v';

QVector<QVector<T>> result = QVector<QVector<T>>();

while (!fileObj.*atEnd*()) {

QString str = fileObj.readLine();

if ((str.size() >= 2) && (str[0] == mark)) {

str[0] = ' ';

QString elements[3];

QTextStream stream(&str);

QVector<T> newElement = QVector<T>();

while (!stream.atEnd()){

QString el;

stream >> el;

if (el == "") continue;

if (is\_v) newElement.append(el.toDouble());

else newElement.append(el.toInt());

}

result.push\_back(newElement);

}

}

fileObj.*close*();

return result;

}

};

class ObjObject {

public:

QVector<QVector3D> points;

QVector<QVector<int>> polygons;

QString file;

// конструктор

ObjObject(bool read\_file=true, QString fileName = "D:/cg\_labs/lab\_4/test.obj") {

points = QVector<QVector3D>();

polygons = QVector<QVector<int>>();

if (read\_file){

file = fileName;

for (auto v: ObjFilesReader::parseByMark<double>(file, true)){

QVector3D vs = QVector3D(v[0], v[1], v[2]);

points.append(vs);

}

polygons = ObjFilesReader::parseByMark<int>(file, false);

for (int i=0; i < polygons.size(); i++)

for (int j=0; j < polygons[i].size(); j++){

polygons[i][j]--;

}

} else rebuildObject(6);

}

QVector3D getPointOnCircleByParams(QVector3D c, int r, int i, int n){

return QVector3D(c.x() + r \* cos(2 \* M\_PI \* i / n),

c.y(),

c.z() + r \* sin(2 \* M\_PI \* i / n));

}

void createPoly4(int a1, int a2, int a3, int a4, bool is\_3=false){

QVector<int> poly = QVector<int>();

poly.append(a1);

poly.append(a2);

poly.append(a3);

if (!is\_3) poly.append(a4);

polygons.append(poly);

}

QVector<QPair<QVector<QVector3D>, QColor>> getSortedPolygons(QVector<QVector3D> &vertecs) {

QVector<QPair<QVector<QVector3D>, QColor>> new\_poly = QVector<QPair<QVector<QVector3D>, QColor>>();

for (int i = 0; i < polygons.size(); i++) {

QVector<QVector3D> poly = QVector<QVector3D>();

for (int j=0; j < polygons[i].size(); j++){

poly.append(vertecs[polygons[i][j]]);

}

new\_poly.append(QPair<QVector<QVector3D>,QColor>(

poly,

QColor(0, (150 + 30\*i) % 255, 255)

//QColor((50\*i) % 255, (150 + 30\*i) % 255, (20\*i + 124) % 200)

));

}

qSort(new\_poly.begin(), new\_poly.end(),

[](QPair<QVector<QVector3D>, QColor>& a, QPair<QVector<QVector3D>, QColor>& b){

return getZForPoly(a.first) < getZForPoly(b.first);

});

return new\_poly;

}

static double getZForPoly(QVector<QVector3D> &poly){

QVector3D rez = QVector3D(0., 0., 0.);

for (int i = 0; i < poly.size(); i++)

rez += poly[i];

return rez.z() / ((double)(poly.size()));

}

void rebuildObject(int angles){

points.clear();

polygons.clear();

double h = 0.25;

double r = 1.5;

double coef = 1.2;

int layers = 8;

int inStep = layers \* 2 + 2;

QVector3D p0 = QVector3D(0.0, 0.0, 0.0);

QVector3D p1 = getPointOnCircleByParams(p0, r, 0, angles);

QVector3D p2 = p1 \* coef;

QVector3D p3 = p1;

QVector3D p4 = p2;

points.append(p0);

points.append(p1);

points.append(p2);

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

p3 = p3 + QVector3D(0, h, 0);

p4 = p4 + QVector3D(0, h, 0);

points.append(p3);

points.append(p4);

}

for (int i=1; i <= angles; i++){

QVector3D p5 = getPointOnCircleByParams(p0, r, i, angles);

QVector3D p6 = p5 \* coef;

QVector3D p7 = p5;

QVector3D p8 = p6;

points.append(p5);

points.append(p6);

createPoly4(0, (i-1)\*inStep+1, i\*inStep+1, 0, true);

createPoly4((i-1)\*inStep+1, (i-1)\*inStep+2, i\*inStep+2,i\*inStep+1);

for (int l=1; l <= layers; l++){

p7 = p7 + QVector3D(0, h, 0);

p8 = p8 + QVector3D(0, h, 0);

points.append(p7);

points.append(p8);

createPoly4((i-1)\*inStep+2\*l, (i-1)\*inStep+2\*(l+1), i\*inStep+2\*(l+1), i\*inStep+2\*l);

createPoly4((i-1)\*inStep+2\*l-1, (i-1)\*inStep+2\*(l+1)-1, i\*inStep+2\*(l+1)-1, i\*inStep+2\*l-1);

}

createPoly4(i\*inStep+2\*(layers+1)-1, i\*inStep+2\*(layers+1),

(i-1)\*inStep+2\*(layers+1), (i-1)\*inStep+2\*(layers+1)-1);

}

for (int i=0; i < points.size(); i++){

points[i] -= QVector3D(0, h \* (layers/2), 0);

points[i] \*= -1;

}

}

bool pointInPoly(QPolygon poly, QPoint p){

Amorph3D amorph = Amorph3D();

QPoint a0 = poly[0] - p;

qInfo() << poly[0] << p << a0;

QPoint b;

int countBig = 0;

for (int i = 0; i < poly.size() - 1; i++){

QPoint a = poly[i] - p;

b = poly[i+1] - p;

if (a.x()\*b.y() - a.y()\*b.x() > 0.) countBig++; else countBig--;

}

if (a0.x()\*b.y() - a0.y()\*b.x() > 0.) countBig--; else countBig++;

return abs(countBig) == poly.size();

}

};

#endif // OBJECTS3D\_H

amorph3d.h

#ifndef AMORPH3D\_H

#define AMORPH3D\_H

#include <QMatrix4x4>

#include <QVector3D>

#include <QVector4D>

#include <QVector>

#include <QDebug>

#include <math.h>

class Amorph3D

{

public:

Amorph3D();

QVector3D doMult(QMatrix4x4 m, QVector3D v, bool is\_central=false);

QVector3D translate(QVector3D v, double dx, double dy, double dz);

QVector3D rotateX(QVector3D v, double angle);

QVector3D rotateY(QVector3D v, double angle);

QVector3D rotateZ(QVector3D v, double angle);

QVector3D reflectXY(QVector3D v);

QVector3D reflectYZ(QVector3D v);

QVector3D reflectXZ(QVector3D v);

QVector3D extention(QVector3D v, double kx, double ky, double kz);

QVector3D ortho(QVector3D v);

QVector3D freeProj(QVector3D v);

QVector3D cabinetProj(QVector3D v);

QVector3D centralProj(QVector3D v, double z0);

};

#endif // AMORPH3D\_H

amorph3d.cpp

#include "amorph3d.h"

Amorph3D::Amorph3D(){}

QVector3D Amorph3D::doMult(QMatrix4x4 m, QVector3D v, bool is\_central){

QVector4D vec = m \* QVector4D(v, 1.);

if (is\_central) return (vec / vec[3]).toVector3D();

return vec.toVector3D();

}

QVector3D Amorph3D::translate(QVector3D v, double dx, double dy, double dz){

QMatrix4x4 m(1., 0., 0., dx,

0., 1., 0., dy,

0., 0., 1., dz,

0., 0., 0., 1.);

return doMult(m, v);

}

QVector3D Amorph3D::rotateX(QVector3D v, double angle){

QMatrix4x4 m(1., 0. , 0. , 0.,

0., cos(angle), -sin(angle), 0.,

0., sin(angle), cos(angle) , 0.,

0., 0. , 0. , 1.);

return doMult(m, v);

}

QVector3D Amorph3D::rotateY(QVector3D v, double angle){

QMatrix4x4 m(cos(angle) , 0., sin(angle), 0.,

0. , 1., 0. , 0.,

-sin(angle), 0., cos(angle), 0.,

0. , 0., 0. , 1.);

return doMult(m, v);

}

QVector3D Amorph3D::rotateZ(QVector3D v, double angle){

QMatrix4x4 m(cos(angle), -sin(angle), 0., 0.,

sin(angle), cos(angle) , 0., 0.,

0. , 0. , 1., 0.,

0. , 0. , 0., 1.);

return doMult(m, v);

}

QVector3D Amorph3D::reflectXY(QVector3D v){

QMatrix4x4 m(1., 0., 0., 0.,

0., 1., 0. ,0.,

0., 0., -1., 0.,

0., 0., 0., 1.);

return doMult(m, v);

}

QVector3D Amorph3D::reflectYZ(QVector3D v){

QMatrix4x4 m(-1., 0., 0., 0.,

0. , 1., 0., 0.,

0. , 0., 1., 0.,

0. , 0., 0., 1.);

return doMult(m, v);

}

QVector3D Amorph3D::reflectXZ(QVector3D v){

QMatrix4x4 m(1., 0., 0., 0.,

0., -1., 0., 0.,

0., 0., 1., 0.,

0., 0., 0., 1.);

return doMult(m, v);

}

QVector3D Amorph3D::extention(QVector3D v, double kx, double ky, double kz){

QMatrix4x4 m(kx, 0. , 0. , 0.,

0. , ky, 0. , 0.,

0. , 0. , kz, 0.,

0. , 0. , 0. , 1.);

return doMult(m, v);

}

QVector3D Amorph3D::ortho(QVector3D v){

QMatrix4x4 m(1., 0., 0., 0.,

0., 1., 0., 0.,

0., 0., 0., 0.,

0., 0., 0., 1.);

return doMult(m, v);

}

QVector3D Amorph3D::freeProj(QVector3D v){

QMatrix4x4 m(1., 0., cos(M\_PI/4.), 0.,

0., 1., cos(M\_PI/4.), 0.,

0., 0., 0., 0.,

0., 0., 0., 1.);

return doMult(m, v);

}

QVector3D Amorph3D::cabinetProj(QVector3D v){

QMatrix4x4 m(1., 0., cos(M\_PI/4.)/2., 0.,

0., 1., sin(M\_PI/4.)/2., 0.,

0., 0., 0., 0.,

0., 0., 0., 1.);

return doMult(m, v);

}

QVector3D Amorph3D::centralProj(QVector3D v, double z0){

QMatrix4x4 m(1., 0., 0., 0.,

0., 1., 0., 0.,

0., 0., 0., 0.,

0., 0.,-1./z0,1.);

return doMult(m, v, true);

}

main.cpp

#include "widget.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

Widget w;

w.show();

return a.exec();

}