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 №4

по дисциплине: Компьютерная графика

тема: «Аффинные преобразования в пространстве»

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;

explicit Widget(QWidget \*parent = 0);

~*Widget*();

private slots:

void on\_comboBox\_currentIndexChanged(int index);

private:

Ui::Widget \*ui;

void *paintEvent*(QPaintEvent \*event);

void *keyPressEvent*(QKeyEvent \*event);

void on\_lineEdit\_returnPressed();

void on\_lineEdit\_textChanged(const QString &arg1);

void *wheelEvent*(QWheelEvent\* );

};

#endif // WIDGET\_H

widget.cpp

#include "widget.h"

#include "ui\_widget.h"

#include <QDebug>

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Widget)

{

projection = 0;

angleX = 0;

angleY = 0;

angleZ = 0;

ui->setupUi(this);

amorph3d = Amorph3D();

objObject = ObjObject(false);

mouseWheel = 3;

}

Widget::~*Widget*()

{

delete ui;

}

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

QPainter painter(this);

ui->comboBox->move(width()/2+5, height()/2+5);

painter.setPen(Qt::black);

painter.drawLine(QPoint(width()/2, 5), QPoint(width()/2, height()-5));

painter.drawLine(QPoint(5, height()/2), QPoint(width()-5, height()/2));

int coef = (std::min(width()/10, height()/10)) - 20;

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

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

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

tmp\_v = amorph3d.extention(tmp\_v, coef, coef, coef);

tmp\_v = amorph3d.translate(tmp\_v, width()/4, height()/4, 0);

tmp\_v = amorph3d.ortho(tmp\_v);

vertecs.append(tmp\_v);

}

for (auto p: objObject.polygons){

QPolygon poly = QPolygon();

for (auto nv: p) poly.append(vertecs[nv].toPoint());

painter.drawPolygon(poly);

}

vertecs.clear();

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

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

tmp\_v = amorph3d.rotateY(tmp\_v, M\_PI/2);

tmp\_v = amorph3d.extention(tmp\_v, coef, coef, coef);

tmp\_v = amorph3d.translate(tmp\_v, 3\*width()/4, height()/4, 0);

tmp\_v = amorph3d.ortho(tmp\_v);

vertecs.append(tmp\_v);

}

for (auto p: objObject.polygons){

QPolygon poly = QPolygon();

for (auto nv: p) poly.append(vertecs[nv].toPoint());

painter.drawPolygon(poly);

}

vertecs.clear();

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

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

tmp\_v = amorph3d.rotateX(tmp\_v, M\_PI/2);

tmp\_v = amorph3d.extention(tmp\_v, coef, coef, coef);

tmp\_v = amorph3d.translate(tmp\_v, width()/4, 3\*height()/4, 0);

tmp\_v = amorph3d.ortho(tmp\_v);

vertecs.append(tmp\_v);

}

for (auto p: objObject.polygons){

QPolygon poly = QPolygon();

for (auto nv: p) poly.append(vertecs[nv].toPoint());

painter.drawPolygon(poly);

}

vertecs.clear();

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

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

tmp\_v = amorph3d.rotateX(tmp\_v, angleX);

tmp\_v = amorph3d.rotateY(tmp\_v, angleY);

tmp\_v = amorph3d.rotateZ(tmp\_v, angleZ);

switch (projection) {

case 0:

tmp\_v = amorph3d.cabinetProj(tmp\_v);

break;

case 1:

tmp\_v = amorph3d.freeProj(tmp\_v);

break;

case 2:

tmp\_v = amorph3d.centralProj(tmp\_v, 5);

break;

case 3:

tmp\_v = amorph3d.ortho(tmp\_v);

break;

}

tmp\_v = amorph3d.extention(tmp\_v, coef, coef, coef);

tmp\_v = amorph3d.translate(tmp\_v, 3\*width()/4, 3\*height()/4, 0);

vertecs.append(tmp\_v);

}

for (auto p: objObject.polygons){

QPolygon poly = QPolygon();

for (auto nv: p) poly.append(vertecs[nv].toPoint());

painter.drawPolygon(poly);

}

vertecs.clear();

}

void Widget::*keyPressEvent*(QKeyEvent \*event){

switch (event->key()) {

case Qt::Key\_Left:{

if (event->modifiers()==Qt::AltModifier) angleZ += -0.1;

else if (event->modifiers()==Qt::ControlModifier) angleX += -0.1;

else angleY += -0.1;

break;

}

case Qt::Key\_Right:{

if (event->modifiers()==Qt::AltModifier) angleZ += 0.1;

else if (event->modifiers()==Qt::ControlModifier) angleX += 0.1;

else angleY += 0.1;

break;

}

default:

break;

}

repaint();

}

// Обработчик события прокрутки колеса мыши

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

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

else mouseWheel -= 1;

if (mouseWheel > 360) {

mouseWheel = 360;

return;

}

if (mouseWheel < 3) {

mouseWheel = 3;

return;

}

objObject.rebuildObject((int)mouseWheel);

repaint(); // Обновляем окно

}

void Widget::on\_comboBox\_currentIndexChanged(int index){

projection = index;

repaint();

}

objects3d.h

#ifndef OBJECTS3D\_H

#define OBJECTS3D\_H

#include <QString>

#include <QVector>

#include <QVector3D>

#include <QPolygon>

#include <QFile>

#include <QTextStream>

#include <QDebug>

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 rebuildObject(int angles){

if ((angles < 3) || (angles > 360)) return;

points.clear();

polygons.clear();

double h = 3.0;

double r = 1.5;

double coef = 1.2;

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

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

QVector3D p2 = p1 \* coef;

QVector3D p3 = p1 + QVector3D(0, h, 0);

QVector3D p4 = p2 + QVector3D(0, h, 0);

points.append(p0);

points.append(p1);

points.append(p2);

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(0, h, 0);

QVector3D p8 = p6 + QVector3D(0, h, 0);

points.append(p5);

points.append(p6);

points.append(p7);

points.append(p8);

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

vect.append(0);

vect.append(i\*4+2);

vect.append((i-1)\*4+2);

polygons.append(vect);

vect.clear();

vect.append(i\*4+2);

vect.append((i-1)\*4+2);

vect.append((i-1)\*4+4);

vect.append(i\*4+4);

polygons.append(vect);

vect.clear();

vect.append(i\*4+1);

vect.append((i-1)\*4+1);

vect.append((i-1)\*4+3);

vect.append(i\*4+3);

polygons.append(vect);

vect.clear();

vect.append(i\*4+3);

vect.append(i\*4+4);

vect.append((i-1)\*4+4);

vect.append((i-1)\*4+3);

polygons.append(vect);

p1 = p5; p2 = p6; p3 = p7; p4 = p8;

//break;

}

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

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

points[i] \*= -1;

}

}

};

#endif // OBJECTS3D\_H

amorph3d.h

#ifndef AMORPH3D\_H

#define AMORPH3D\_H

#include <QMatrix4x4>

#include <QVector3D>

#include <QVector4D>

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();

}