**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИИ**  
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

Компьютерная графика

**Аффинные преобразования в пространстве**

Выполнил: ст. группы ВТ-22  
Ковалёв И. Д.

Проверил: Осипов О. В.

**Белгород 2019**

**Цель работы:**

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

**Порядок выполнения работы**

1. Разработать алгоритм и составить программу для построения на экране изображения 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);

1.2. Правая верхняя часть – профильная проекция (вид сбоку);

1.3. Левая нижняя часть должна отображать вид сверху(горизонтальную проекцию);

1.4. На правой нижней части должна отображаться проекция, вид которой выбирает пользователь: центральная, кабинетная, косоугольная, свободная, ортографическая.

Содержимое файла mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <cmath>

#include <QtMath>

#include <QPainter>

#include <QMatrix4x4>

#include <QLine>

#include <QPoint>

#include <QMouseEvent>

#include <float.h>

#include "object.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow{

Q\_OBJECT

*private*:

Cubes f;

float globalScale, ctrlMouseWheelCount, altMouseWheel;

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\* e);

void **drawLines**(QPainter & painter);

void **drawProjections**(QPainter & painter, Object & obj);

void **draw3D**(QPainter & painter, Object & obj, int view);

*private* slots:

void **on\_zoomInc\_clicked**();

void **on\_zoomDec\_clicked**();

void **on\_central\_clicked**();

void **on\_cabinet\_clicked**();

void **on\_free\_clicked**();

void **on\_ortogonal\_clicked**();

*private*:

Ui::MainWindow \*ui;

QMatrix4x4 rotationMatrix;

int view, range;

QPointF pos;

void ***mouseMoveEvent***(QMouseEvent \* event);

void ***wheelEvent***(QWheelEvent \* event);

};

#endif *//* *MAINWINDOW\_H*

Содержимое файла mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent) : QMainWindow(parent), ui(*new* Ui::MainWindow){

ui->setupUi(*this*);

range = 0;

f = Cubes(range);

globalScale = 1;

view = 0;

ui->central->setChecked(1);

}

MainWindow::~***MainWindow***(){

*delete* ui;

}

float **maxSize**(QVector3D& v){

*if* (v.x() > v.y())

*if*(v.x() > v.z()) *return* v.x();

*else* *return* v.z();

*else* *if* (v.y() > v.z()) *return* v.y();

*else* *return* v.z();

}

void MainWindow::***paintEvent***(QPaintEvent \* e){

QPainter painter(*this*);

drawLines(*painter*);

drawProjections(*painter*, f);

draw3D(*painter*, f, view);

}

void MainWindow::**drawLines**(QPainter &painter){

painter.save();

int ax = ui->tl->geometry().center().x();

*//this->width()* */* *4;*

int ay = ui->tl->geometry().center().y();

int bx = ui->tr->geometry().center().x();

int by = ui->tr->geometry().center().y();

int cx = ui->bl->geometry().center().x();

int cy = ui->bl->geometry().center().y();

painter.setPen(Qt::*blue*);

painter.drawLine(QLine(QPoint(ax, ay), QPoint(bx, by)));

painter.drawLine(QLine(QPoint(ax, ay), QPoint(cx, cy)));

painter.restore();

}

void MainWindow::**drawProjections**(QPainter &painter, Object &obj){

QVector3D center = obj.getCenter();

QVector3D size = obj.getSize();

float k = qMax(ui->br->width(), ui->br->height())/maxSize(*size*)/2\*(1+(globalScale)/100.f);

QVector <QPolygonF> polygons;

QPolygonF polygon;

*const* QVector<QVector<QVector3D>>currentObject = obj.getObject();

QVector3D v;

QMatrix4x4 M;

*//Вид* *спереди*

M.translate(ui->tl->geometry().center().x(), ui->tl->geometry().center().y());

M.scale(k, -k, k);

M.translate(-center);

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

*for*(int j = 0; j < currentObject.at(i).size(); j++){

v.setX(currentObject.at(i).at(j).x());

v.setY(currentObject.at(i).at(j).y());

v.setZ(currentObject.at(i).at(j).z());

v = M\*v;

polygon << QPointF(v.x(), v.y());

}

polygons << polygon;

polygon.clear();

}

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

painter.drawPolygon(polygons.at(i));

polygons.clear();

*//Вид* *сверху*

M.setToIdentity();

M.translate(ui->bl->geometry().center().x(), ui->bl->geometry().center().y());

M.rotate(90, QVector3D(1, 0, 0));

M.scale(k, -k, k);

M.translate(-center);

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

*for*(int j = 0; j < currentObject.at(i).size(); j++){

v.setX(currentObject.at(i).at(j).x());

v.setY(currentObject.at(i).at(j).y());

v.setZ(currentObject.at(i).at(j).z());

v = M\*v;

polygon << QPointF(v.x(), v.y());

}

polygons << polygon;

polygon.clear();

}

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

painter.drawPolygon(polygons.at(i));

polygons.clear();

*//Вид* *совзбоку*

M.setToIdentity();

M.translate(ui->tr->geometry().center().x(), ui->tr->geometry().center().y());

M.rotate(90, QVector3D(0, 1, 0));

M.scale(k, -k, k);

M.translate(-center);

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

*for*(int j = 0; j < currentObject.at(i).size(); j++){

v.setX(currentObject.at(i).at(j).x());

v.setY(currentObject.at(i).at(j).y());

v.setZ(currentObject.at(i).at(j).z());

v = M\*v;

polygon << QPointF(v.x(), v.y());

}

polygons << polygon;

polygon.clear();

}

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

painter.drawPolygon(polygons.at(i));

polygons.clear();

}

void MainWindow::**draw3D**(QPainter &painter, Object &obj, int view){

float c, k;

QVector3D size = obj.getSize();

float scale = qMax(ui->br->width(),

ui->br->height()) / maxSize(*size*) / 2 \* (1 + (globalScale) / 100.f);

QMatrix4x4 M, projectionMatrix;

*switch* (view){

*case* 0: c = 200 + ctrlMouseWheelCount;

*//Центральная* *проекция*

projectionMatrix = {1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 0, 0,

0, 0, -1/c, 1};

*break*;

*case* 1: k = qCos(M\_PI\_4)/2;

*//Косоугольная* *кабинетная* *проекция*

projectionMatrix = {1, 0, k, 0,

0, 1, k, 0,

0, 0, 0, 0,

0, 0, 0, 1};

*break*;

*case* 2: k = qCos(M\_PI\_4);

*//Косоугольная* *свободная* *проекция*

projectionMatrix = {1, 0, k, 0,

0, 1, k, 0,

0, 0, 0, 0,

0, 0, 0, 1};

*break*;

*case* 3:

*//Ортогональная* *проекция*

projectionMatrix = {1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 0, 0,

0, 0, 0, 1};

*break*;

}

M.translate(ui->br->geometry().center().x(),

ui->br->geometry().center().y(),

0);

QVector3D center = obj.getCenter();

M = M \* projectionMatrix;

*//M.translate(0,* *0,* *altMouseWheel);*

M = M \* rotationMatrix.transposed();

M.scale(scale, -scale, scale );

M.translate(-center);

QVector<QPolygonF> polygons;

QPolygonF polygon;

*const* QVector<QVector<QVector3D>> currentObject = obj.getObject();

QVector4D v;

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

*for*(int j = 0; j < currentObject.at(i).size(); j++){

v.setX(currentObject.at(i).at(j).x());

v.setY(currentObject.at(i).at(j).y());

v.setZ(currentObject.at(i).at(j).z());

v.setW(1);

v = M \* v;

polygon << QPointF(v.x()/v.w(), v.y()/v.w());

}

polygons << polygon;

polygon.clear();

}

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

painter.drawPolygon(polygons.at(i));

}

void MainWindow::**on\_zoomInc\_clicked**(){

globalScale += 10;

*this*->repaint();

}

void MainWindow::**on\_zoomDec\_clicked**(){

*if* (globalScale > 10)

globalScale -= 10;

*this*->repaint();

}

void MainWindow::**on\_central\_clicked**(){

ui->central->setChecked(1);

ui->cabinet->setChecked(0);

ui->free->setChecked(0);

ui->ortogonal->setChecked(0);

view = 0;

*this*->repaint();

}

void MainWindow::**on\_cabinet\_clicked**(){

ui->central->setChecked(0);

ui->cabinet->setChecked(1);

ui->free->setChecked(0);

ui->ortogonal->setChecked(0);

view = 1;

*this*->repaint();

}

void MainWindow::**on\_free\_clicked**(){

ui->central->setChecked(0);

ui->cabinet->setChecked(0);

ui->free->setChecked(1);

ui->ortogonal->setChecked(0);

view = 2;

*this*->repaint();

}

void MainWindow::**on\_ortogonal\_clicked**(){

ui->central->setChecked(0);

ui->cabinet->setChecked(0);

ui->free->setChecked(0);

ui->ortogonal->setChecked(1);

view = 3;

*this*->repaint();

}

void MainWindow::***mouseMoveEvent***(QMouseEvent\* event){

rotationMatrix.rotate(pos.x() - event->pos().x(), QVector3D(0,1,0));

rotationMatrix.rotate(-pos.y() + event->pos().y(), QVector3D(1, 0, 0));

pos = event->pos();

repaint();

}

void MainWindow::***wheelEvent***(QWheelEvent \*event){

*if* (event->delta() < 0){

*if* (range > 0) range--;

} *else* range++;

f = Cubes(range);

*this*->repaint();

}

Содержимое файла object.h:

#ifndef OBJECT\_H

#define OBJECT\_H

#include <QVector3D>

#include <QMatrix4x4>

#include <QtMath>

*class* **Object**{

*protected*:

QVector<QVector<QVector3D>> obj;

*public*:

**Object**() {};

QVector<QVector<QVector3D>> **getObject**();

QVector3D **getCenter**();

QVector3D **getSize**();

};

*class* **Cubes** : *public* Object{

*public*:

**Cubes**();

**Cubes**(int range);

};

#endif *//* *OBJECT\_H*

Содержимое файла object.cpp:

#include "object.h"

QVector<QVector<QVector3D>> Object::**getObject**(){

*return* obj;

}

QVector3D Object::**getCenter**(){

float x0 = 0, y0 = 0, z0 = 0, x1, y1, z1;

float n1, n = obj.size();

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

x1 = 0;

y1 = 0;

z1 = 0;

n1 = obj.at(i).size();

*for* (int j = 0; j < obj.at(i).size(); j++){

x1 += obj.at(i).at(j).x();

y1 += obj.at(i).at(j).y();

z1 += obj.at(i).at(j).z();

}

x0 += x1/n1;

y0 += y1/n1;

z0 += z1/n1;

}

*return* QVector3D(x0/n, y0/n, z0/n);

}

QVector3D Object::**getSize**(){

float min\_x = obj.at(0).at(0).x(),

max\_x = min\_x,

min\_y = obj.at(0).at(0).y(),

max\_y = min\_y,

min\_z = obj.at(0).at(0).z(),

max\_z = min\_z;

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

*for* (int j = 0; j < obj.at(i).size(); j++){

*if* (obj.at(i).at(j).x() < min\_x) min\_x = obj.at(i).at(j).x();

*if* (obj.at(i).at(j).x() > max\_x) max\_x = obj.at(i).at(j).x();

*if* (obj.at(i).at(j).y() < min\_y) min\_y = obj.at(i).at(j).y();

*if* (obj.at(i).at(j).y() > max\_y) max\_y = obj.at(i).at(j).y();

*if* (obj.at(i).at(j).z() < min\_z) min\_z = obj.at(i).at(j).z();

*if* (obj.at(i).at(j).z() > max\_z) max\_z = obj.at(i).at(j).z();

}

}

*return* QVector3D(max\_x - min\_x, max\_y - min\_y, max\_z - min\_z);

}

Cubes::**Cubes**(){

Cubes(1);

}

Cubes::**Cubes**(int range){

QVector3D A (0,0,0), B (4,0,0), C (4,4,0), D (0,4,0),

A1(0,0,4), B1(4,0,4), C1(4,4,4), D1(0,4,4);

QVector<QVector3D> tmp;

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

A = QVector3D(4, 3, 3); B = QVector3D(4, 3, 1);

C = QVector3D(5 + range, 3, 1); D = QVector3D(5 + range, 3, 3);

A1 = QVector3D(4, 1, 3); B1 = QVector3D(4, 1, 1);

C1 = QVector3D(5 + range, 1, 1); D1 = QVector3D(5 + range, 1, 3);

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

A = QVector3D(5 + range, -1, -1); B = QVector3D(9 + range, -1, -1);

C = QVector3D(9 + range, 5, -1); D = QVector3D(5 + range, 5, -1);

A1 = QVector3D(5 + range, -1, 5); B1 = QVector3D(9 + range, -1, 5);

C1 = QVector3D(9 + range, 5, 5); D1 = QVector3D(5 + range, 5, 5);

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

A = QVector3D(9 + range, 3, 3); B = QVector3D(9 + range, 3, 1);

C = QVector3D(10 + range \* 2, 3, 1); D = QVector3D(10 + range \* 2, 3, 3);

A1 = QVector3D(9 + range, 1, 3); B1 = QVector3D(9 + range, 1, 1);

C1 = QVector3D(10 + range \* 2, 1, 1); D1 = QVector3D(10 + range \* 2, 1, 3);

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

A = QVector3D(10 + range \* 2, 0, 0); B = QVector3D(14 + range \* 2, 0, 0);

C = QVector3D(14 + range \* 2, 4, 0); D = QVector3D(10 + range \* 2, 4, 0);

A1 = QVector3D(10 + range \* 2, 0, 4); B1 = QVector3D(14 + range \* 2, 0, 4);

C1 = QVector3D(14 + range \* 2, 4, 4); D1 = QVector3D(10 + range \* 2, 4, 4);

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

}

