**МИНИСТЕРСТВА 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

Дисциплина: Компьютерная графика

Тема: «Аффинные преобразования в пространстве»

Выполнила: ст. группы ПВ-31

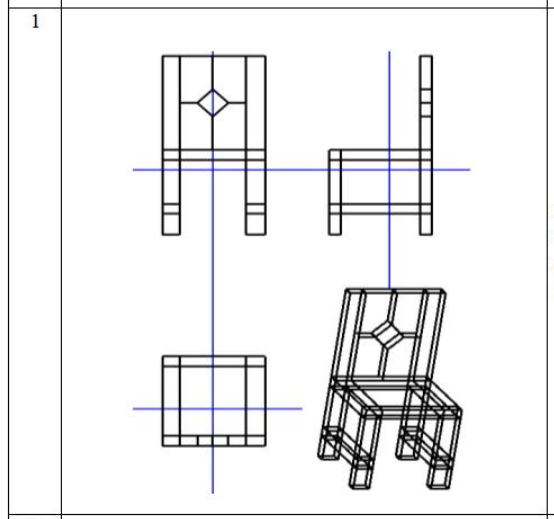
Донцов А.А.

Проверил: Осипов О. В.

Белгород 2019

**Цель работы:** разработка библиотеки для работы с матрицами размера 4\*4 и различными видами проекции, разработка графического приложения для визуализации 3х мерной проекции.

**Вариант 1**



Main

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

Figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include <mainwindow.h>

#include <QVector>

#include <QVector3D>

#include <QMatrix4x4>

#include "matr\_pr.h"

using namespace std;

class **Figure**{

public:

QVector <QVector3D> Points;

**Figure**();

void **set\_part**(double centerX, double centerY, double centerZ, double x, double y, double z, int angle);

void **draw**(QPainter \*painter);

~**Figure**() {};

};

#endif // FIGURE\_H

Figure.cpp

#include "figure.h"

Figure::**Figure**()

{

set\_part(-1.25,-2,0, 2.5,3,0.5, 0);

set\_part(-1.25,-1.25,0, 0.5,0.5,0.5, 45);

set\_part(-1.25,1,0, 2.5,0.5,2.5, 0);

set\_part(-1.25,1,0, 0.5,2,0.5, 0);

set\_part(0.75,1,0, 0.5,2,0.5, 0);

set\_part(-1.25,1,2, 0.5,2,0.5, 0);

set\_part(0.75,1,2, 0.5,2,0.5, 0);

}

void Figure::**set\_part**(double centerX, double centerY, double centerZ, double x, double y, double z, int angle) //задать часть фигуры

{

QMatrix4x4 m;

m.rotate(angle, QVector3D(0,0,1));

Points.push\_back(m\*QVector3D(centerX + x,centerY + y,centerZ));//x,y,0

Points.push\_back(m\*QVector3D(centerX + x,centerY,centerZ));//x,0,0

Points.push\_back(m\*QVector3D(centerX,centerY,centerZ));//0,0,0

Points.push\_back(m\*QVector3D(centerX,centerY + y,centerZ));//0,y,0

Points.push\_back(m\*QVector3D(centerX + x,centerY + y,centerZ + z));//x,y,0

Points.push\_back(m\*QVector3D(centerX + x,centerY,centerZ + z));//x,0,0

Points.push\_back(m\*QVector3D(centerX,centerY,centerZ + z));//0,0,0

Points.push\_back(m\*QVector3D(centerX,centerY + y,centerZ + z));//0,y,0

}

void Figure::**draw**(QPainter \*painter)

{

painter->setBrush(Qt::NoBrush);

painter->setPen(QPen(Qt::black, 2, Qt::SolidLine, Qt::RoundCap));

QVector<QPolygon> vect;

for(int i = 0, p = 0; i < 50; i += 4, p += 1)

{

QPolygon t;

for(int j = 0; j < 4; j++)

t << QPoint(Points[i + j].x(), Points[i + j].y());

vect.push\_back(t);

//соединительные грани

if (p % 2 == 0)

{

vect.push\_back(QPolygon() << QPoint(Points[i + 5].x(), Points[i + 5].y()) << QPoint(Points[i + 1].x(), Points[i + 1].y()) << QPoint(Points[i + 2].x(), Points[i + 2].y()) << QPoint(Points[i + 6].x(), Points[i + 6].y()));

vect.push\_back(QPolygon() << QPoint(Points[i + 4].x(), Points[i + 4].y()) << QPoint(Points[i].x(), Points[i].y()) << QPoint(Points[i + 3].x(), Points[i + 3].y()) << QPoint(Points[i + 7].x(), Points[i + 7].y()));

vect.push\_back(QPolygon() << QPoint(Points[i + 5].x(), Points[i + 5].y()) << QPoint(Points[i + 1].x(), Points[i + 1].y()) << QPoint(Points[i].x(), Points[i].y()) << QPoint(Points[i + 4].x(), Points[i + 4].y()));

vect.push\_back(QPolygon() << QPoint(Points[i + 6].x(), Points[i + 6].y()) << QPoint(Points[i + 2].x(), Points[i + 2].y()) << QPoint(Points[i + 3].x(), Points[i + 3].y()) << QPoint(Points[i + 7].x(), Points[i + 7].y()));

}

}

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

painter->drawPolygon(vect[i]);

}

}

Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QMouseEvent>

#include <QPainter>

#include "figure.h"

#include "matr\_pr.h"

QT\_BEGIN\_NAMESPACE

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

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

private slots:

void **newProection\_pressed**();

public:

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

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

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

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

QMatrix4x4 R1;

int wheelCoeficient;

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

~***MainWindow***();

private:

Ui::MainWindow \*ui;

//Повороты по осям

float xAxisRotation;

float yAxisRotation;

int numberProjection;

//Координаты предыдущей точки курсора мыши

QPointF previousPoint;

};

#endif // MAINWINDOW\_H

Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <math.h>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

setMinimumWidth(320);

numberProjection = 0;

xAxisRotation = 0;

yAxisRotation = 0;

wheelCoeficient = 0;

R1 = QMatrix4x4();

ui->setupUi(this);

ui->ProectionBox->addItem("Центральная");

ui->ProectionBox->addItem("Ортографическая");

ui->ProectionBox->addItem("Кабинетная");

ui->ProectionBox->addItem("Свободная");

ui->ProectionBox->addItem("Триметрическая");

connect(ui->ProectionBox, SIGNAL(currentIndexChanged(int)), this, SLOT(newProection\_pressed()));

}

void MainWindow::***paintEvent***(QPaintEvent \*event)

{

QPainter painter(this);

painter.setPen((QPen(Qt::black, 2, Qt::SolidLine, Qt::FlatCap)));

painter.drawLine(width()/2, 0, width()/2, height());

painter.drawLine(0, height()/2, width(), height()/2);

double scale = 1;

if (width()/2 < height())

scale = width()/20.0;

else

scale = height()/10.0;

//вид сверху

Figure fg;

QMatrix4x4 matr;

matr.translate(width() / 4, 3 \* height() / 4, 0);

matr.scale(scale, scale, scale);

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

{

QVector3D V = {fg.Points[i].x(), fg.Points[i].y(), fg.Points[i].z()};

V = matr \* V;

fg.Points[i] = QVector3D(V.x(), V.y(), V.z());

}

fg.draw(&painter);

//вид спереди

fg = Figure();

matr = QMatrix4x4();

matr.translate(width() / 4, 1.2 \* height() / 4, 0);

matr.scale(scale, scale, scale);

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

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

{

QVector3D V = {fg.Points[i].x(), fg.Points[i].y(), fg.Points[i].z()};

V = matr \* V;

fg.Points[i] = QVector3D(V.x(), V.y(), V.z());

}

fg.draw(&painter);

//вид сбоку

fg = Figure();

matr = QMatrix4x4();

matr.translate(3\*width()/4, 1.2 \* height() / 4, 0);

matr.scale(scale, scale, scale);

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

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

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

{

QVector3D V = {fg.Points[i].x(), fg.Points[i].y(), fg.Points[i].z()};

V = matr \* V;

fg.Points[i] = QVector3D(V.x(), V.y(), V.z());

}

fg.draw(&painter);

//проекции

fg = Figure();

//центральная проекция

if (numberProjection == 0)

{

QMatrix4x4 matr;

matr.translate(3 \* width() / 4, 3 \* height() / 4, 0);

matr = matr \* Matrix::prj\_c(scale\*15);

matr.translate(0, 0, wheelCoeficient);

matr = matr \* R1.transposed();

matr.scale(scale, scale, scale);

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

{

QVector3D V(fg.Points[i].x(), fg.Points[i].y(), fg.Points[i].z());

V = matr \* V;

fg.Points[i] = QVector3D(V.x(), V.y(), V.z());

}

}

else

//параллельная проекция

{

QMatrix4x4 matr;

matr.translate(3 \* width() / 4, 3 \* height() / 4, 0);

matr = matr \* Matrix::prj(numberProjection);

matr.rotate(yAxisRotation, QVector3D(0,1,0));

matr.rotate(xAxisRotation, QVector3D(1,0,0));

matr.scale(scale,scale,scale);

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

{

QVector3D V(fg.Points[i].x(), fg.Points[i].y(), fg.Points[i].z());

V = matr \* V;

fg.Points[i] = QVector3D(V.x(), V.y(), V.z());

}

}

fg.draw(&painter);

}

void MainWindow::**newProection\_pressed**()

{

numberProjection = ui->ProectionBox->currentIndex();

repaint();

}

//Нажатие мыши

void MainWindow::***mousePressEvent***(QMouseEvent \*event)

{

previousPoint = event->windowPos();

}

//Смещение мыши относительно нажатия

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

{

float dh = (event->y() - previousPoint.y());

float dw = (event->x() - previousPoint.x());

R1.rotate(-dw, QVector3D(0,1,0));

R1.rotate(dh, QVector3D(1,0,0));

//Обновление данных о предыдущей координате

previousPoint = event->windowPos();

repaint();

}

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

int wheelCoeficientPlus = event -> delta() / 4;

wheelCoeficient += wheelCoeficientPlus;

repaint();

}

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

{

delete ui;

}

Matr\_pr.h

#ifndef MATRIX\_H

#define MATRIX\_H

#include <math.h>

#include <QMatrix4x4>

class **Matrix** {

public:

static QMatrix4x4 **prj**(int t);

static QMatrix4x4 **prj\_c**(double h);

};

#endif // MATRIX\_H

Matr\_pr.cpp

#include "matr\_pr.h"

#include "stdio.h"

QMatrix4x4 Matrix::**prj\_c**(double h)

{

return QMatrix4x4 (1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 0, 0,

0, 0, -1/h, 1);

}

QMatrix4x4 Matrix::**prj**(int t)

{

QMatrix4x4 M;

// 1 тип - Ортографическая проекция с z = 0

if (t == 1)

M = QMatrix4x4(1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 0, 0,

0, 0, 0, 1);

/\*2 тип - Триметрическая проекция (поворот на 30 вокруг Y

поворот на 45 вокруг X и проецирование на Z = 0)\*/

if (t == 2)

M =

QMatrix4x4(sqrt(3)/2, 0, 1/2, 0,

sqrt(2)/4, sqrt(2)/2, -sqrt(6)/4, 0,

0, 0, 0, 0,

0, 0, 0, 1);

//3 тип - Косоугольная Свободная , т.к угол = pi/4

if (t == 3)

M = QMatrix4x4(1, 0, sqrt(2)/2, 0,

0, 1, sqrt(2)/2, 0,

0, 0, 0, 0,

0, 0, 0, 1);

//4 тип - Косоугольная Кабинетная , т.к угол = 1/2 \* pi/4

if (t == 4)

M = QMatrix4x4(1, 0, sqrt(2)/4, 0,

0, 1, sqrt(2)/4, 0,

0, 0, 0, 0,

0, 0, 0, 1);

return M;

}

