МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

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

Дисциплина: Компьютерная графика

Лабораторная работа № 4

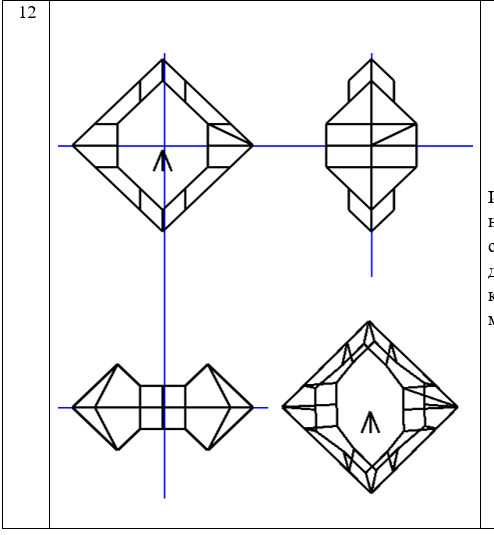
Аффинные преобразования в пространстве

|  |  |
| --- | --- |
|  | Выполнила: ст. группы ПВ-31  Зановская А.И.  Проверил: Осипов О.В. |

Белгород

2018

**Цель работы**: получение навыков построения аффинных преобразований на плоскости и написание графического приложения с использованием GDI в среде Qt Creator.



matrix.h

#ifndef MATRIX\_H

#define MATRIX\_H

#include <QPainter>

#include <vector>

class Vector3D {

public:

double x, y, z, h;

Vector3D();

Vector3D(double \_x, double \_y, double \_z, double \_h);

~Vector3D();

double X();

double Y();

double Z();

double H();

void setX(double \_x);

void setY(double \_y);

void setZ(double \_z);

void setH(double \_h);

Vector3D normalized();

};

class Line3D {

private:

Vector3D \*point1;

Vector3D \*point2;

public:

Line3D();

Line3D(Vector3D \*p1, Vector3D \*p2);

~Line3D();

void draw(QPainter &painter);

};

class Matrix3D {

public:

double M[4][4];

Matrix3D();

Matrix3D(double a11, double a12, double a13, double a14,

double a21, double a22, double a23, double a24,

double a31, double a32, double a33, double a34,

double a41, double a42, double a43, double a44);

~Matrix3D();

Matrix3D operator \*(const Matrix3D& A);

Matrix3D operator \*=(const Matrix3D& A);

Vector3D operator \*(const Vector3D& V);

Vector3D operator \*=(const Vector3D& V);

static Matrix3D rotateX(double angle);

static Matrix3D rotateY(double angle);

static Matrix3D rotateZ(double angle);

static Matrix3D reflectionX();

static Matrix3D reflectionY();

static Matrix3D reflectionZ();

static Matrix3D scale(double a, double b, double c);

static Matrix3D translate(double dx, double dy, double dz);

static Matrix3D orthoProjXY();

static Matrix3D orthoProjYZ();

static Matrix3D orthoProjZX();

static Matrix3D freeProj();

static Matrix3D cabinetProj();

static Matrix3D centralProj(double z0);

};

class Object3D {

private:

std::vector<Vector3D> points;

std::vector<Vector3D> current\_pos;

std::vector<Line3D> lines;

public:

Object3D();

~Object3D();

void draw(QPainter& painter, Matrix3D M3D);

void draw3D(QPainter& painter, Matrix3D M3D, Matrix3D T, QPointF center);

};

#endif // MATRIX\_H

matrix.cpp

#include "matrix.h"

Vector3D::Vector3D() {}

Vector3D::Vector3D(double \_x, double \_y, double \_z, double \_h) {

x = \_x; y = \_y; z = \_z; h = \_h;

}

Vector3D::~Vector3D() {}

double Vector3D::X() { return x; }

double Vector3D::Y() { return y; }

double Vector3D::Z() { return z; }

double Vector3D::H() { return h;}

void Vector3D::setX(double \_x) {

x = \_x;}

void Vector3D::setY(double \_y) {

y = \_y;}

void Vector3D::setZ(double \_z) {

z = \_z;}

void Vector3D::setH(double \_h) {

h = \_h;}

Vector3D Vector3D::normalized() {

return Vector3D(x / h, y / h, z / h, 1);}

Matrix3D::Matrix3D() {}

Matrix3D::Matrix3D(double a11, double a12, double a13, double a14,

double a21, double a22, double a23, double a24,

double a31, double a32, double a33, double a34,

double a41, double a42, double a43, double a44) {

M[0][0] = a11; M[0][1] = a12; M[0][2] = a13; M[0][3] = a14;

M[1][0] = a21; M[1][1] = a22; M[1][2] = a23; M[1][3] = a24;

M[2][0] = a31; M[2][1] = a32; M[2][2] = a33; M[2][3] = a34;

M[3][0] = a41; M[3][1] = a42; M[3][2] = a43; M[3][3] = a44;

}

Matrix3D::~Matrix3D() {}

Matrix3D Matrix3D::operator \*(const Matrix3D& A) {

return Matrix3D(M[0][0] \* A.M[0][0] + M[0][1] \* A.M[1][0] + M[0][2] \* A.M[2][0] + M[0][3] \* A.M[3][0],

M[0][0] \* A.M[0][1] + M[0][1] \* A.M[1][1] + M[0][2] \* A.M[2][1] + M[0][3] \* A.M[3][1],

M[0][0] \* A.M[0][2] + M[0][1] \* A.M[1][2] + M[0][2] \* A.M[2][2] + M[0][3] \* A.M[3][2],

M[0][0] \* A.M[0][3] + M[0][1] \* A.M[1][3] + M[0][2] \* A.M[2][3] + M[0][3] \* A.M[3][3],

M[1][0] \* A.M[0][0] + M[1][1] \* A.M[1][0] + M[1][2] \* A.M[2][0] + M[1][3] \* A.M[3][0],

M[1][0] \* A.M[0][1] + M[1][1] \* A.M[1][1] + M[1][2] \* A.M[2][1] + M[1][3] \* A.M[3][1],

M[1][0] \* A.M[0][2] + M[1][1] \* A.M[1][2] + M[1][2] \* A.M[2][2] + M[1][3] \* A.M[3][2],

M[1][0] \* A.M[0][3] + M[1][1] \* A.M[1][3] + M[1][2] \* A.M[2][3] + M[1][3] \* A.M[3][3],

M[2][0] \* A.M[0][0] + M[2][1] \* A.M[1][0] + M[2][2] \* A.M[2][0] + M[2][3] \* A.M[3][0],

M[2][0] \* A.M[0][1] + M[2][1] \* A.M[1][1] + M[2][2] \* A.M[2][1] + M[2][3] \* A.M[3][1],

M[2][0] \* A.M[0][2] + M[2][1] \* A.M[1][2] + M[2][2] \* A.M[2][2] + M[2][3] \* A.M[3][2],

M[2][0] \* A.M[0][3] + M[2][1] \* A.M[1][3] + M[2][2] \* A.M[2][3] + M[2][3] \* A.M[3][3],

M[3][0] \* A.M[0][0] + M[3][1] \* A.M[1][0] + M[3][2] \* A.M[2][0] + M[3][3] \* A.M[3][0],

M[3][0] \* A.M[0][1] + M[3][1] \* A.M[1][1] + M[3][2] \* A.M[2][1] + M[3][3] \* A.M[3][1],

M[3][0] \* A.M[0][2] + M[3][1] \* A.M[1][2] + M[3][2] \* A.M[2][2] + M[3][3] \* A.M[3][2],

M[3][0] \* A.M[0][3] + M[3][1] \* A.M[1][3] + M[3][2] \* A.M[2][3] + M[3][3] \* A.M[3][3]);

}

Matrix3D Matrix3D::operator \*=(const Matrix3D& A) {

return ((\*this) \* A);}

Vector3D Matrix3D::operator \*(const Vector3D& V) {

return Vector3D(M[0][0]\*V.x + M[0][1]\*V.y + M[0][2]\*V.z + M[0][3]\*V.h,

M[1][0]\*V.x + M[1][1]\*V.y + M[1][2]\*V.z + M[1][3]\*V.h,

M[2][0]\*V.x + M[2][1]\*V.y + M[2][2]\*V.z + M[2][3]\*V.h,

M[3][0]\*V.x + M[3][1]\*V.y + M[3][2]\*V.z + M[3][3]\*V.h);

}

Vector3D Matrix3D::operator \*=(const Vector3D& V) {

return ((\*this) \* V);}

Matrix3D Matrix3D::rotateX(double angle) {

double radAngle = angle \* M\_PI / 180;

return Matrix3D(1, 0, 0, 0,

0, cos(radAngle), sin(radAngle), 0,

0, -sin(radAngle), cos(radAngle), 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::rotateY(double angle) {

double radAngle = angle \* M\_PI / 180;

return Matrix3D(cos(radAngle), 0, -sin(radAngle), 0,

0, 1, 0, 0,

sin(radAngle), 0, cos(radAngle), 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::rotateZ(double angle) {

double radAngle = angle \* M\_PI / 180;

return Matrix3D( cos(radAngle), sin(radAngle), 0, 0,

-sin(radAngle), cos(radAngle), 0, 0,

0, 0, 1, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::reflectionX() {

return Matrix3D(-1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::reflectionY() {

return Matrix3D(1, 0, 0, 0,

0, -1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::reflectionZ() {

return Matrix3D(1, 0, 0, 0,

0, 1, 0, 0,

0, 0, -1, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::scale(double a, double b, double c) {

return Matrix3D(a, 0, 0, 0,

0, b, 0, 0,

0, 0, c, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::translate(double dx, double dy, double dz) {

return Matrix3D(1, 0, 0, dx,

0, 1, 0, dy,

0, 0, 1, dz,

0, 0, 0, 1);

}

Matrix3D Matrix3D::orthoProjXY() {

return Matrix3D(1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 0, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::orthoProjYZ() {

return Matrix3D(0, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::orthoProjZX() {

return Matrix3D(1, 0, 0, 0,

0, 0, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::freeProj() {

return Matrix3D(1, 0, cos(M\_PI / 4) / 2, 0,

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

0, 0, 0, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::cabinetProj() {

return Matrix3D(1, 0, cos(M\_PI / 4), 0,

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

0, 0, 0, 0,

0, 0, 0, 1);

}

Matrix3D Matrix3D::centralProj(double z0) {

return Matrix3D(1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 0, 0,

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

}

Line3D::Line3D() {}

Line3D::~Line3D() {}

Line3D::Line3D(Vector3D \*p1, Vector3D \*p2) {

point1 = p1; point2 = p2;}

void Line3D::draw(QPainter &painter) {

painter.drawLine(point1->X() / point1->H(), point1->Y() / point1->H(),

point2->X() / point2->H(), point2->Y() / point2->H());

}

Object3D::Object3D() {

// Задаём координаты точек

points.push\_back(Vector3D(0, 1.5, 0.5, 1)); //0

points.push\_back(Vector3D(0.5, 1, 0.5, 1));

points.push\_back(Vector3D(1, 0.5, 1, 1));

points.push\_back(Vector3D(1, 0, 1, 1));

points.push\_back(Vector3D(1, -0.5, 1, 1));

points.push\_back(Vector3D(0.5, -1, 0.5, 1));

points.push\_back(Vector3D(0, -1.5, 0.5, 1)); //0

points.push\_back(Vector3D(-0.5, -1, 0.5, 1));

points.push\_back(Vector3D(-1, -0.5, 1, 1));

points.push\_back(Vector3D(-1, 0, 1, 1));

points.push\_back(Vector3D(-1, 0.5, 1, 1));

points.push\_back(Vector3D(-0.5, 1, 0.5, 1));

points.push\_back(Vector3D(0, 1.5, -0.5, 1)); //0

points.push\_back(Vector3D(0.5, 1, -0.5, 1));

points.push\_back(Vector3D(1, 0.5, -1, 1));

points.push\_back(Vector3D(1, 0, -1, 1));

points.push\_back(Vector3D(1, -0.5, -1, 1));

points.push\_back(Vector3D(0.5, -1, -0.5, 1));

points.push\_back(Vector3D(0, -1.5, -0.5, 1)); //0

points.push\_back(Vector3D(-0.5, -1, -0.5, 1));

points.push\_back(Vector3D(-1, -0.5, -1, 1));

points.push\_back(Vector3D(-1, 0, -1, 1));

points.push\_back(Vector3D(-1, 0.5, -1, 1));

points.push\_back(Vector3D(-0.5, 1, -0.5, 1));

points.push\_back(Vector3D(0, 2, 0, 1)); //24

points.push\_back(Vector3D(0.5, 1.5, 0, 1));

points.push\_back(Vector3D(1.5, 0.5, 0, 1));

points.push\_back(Vector3D(2, 0, 0, 1));

points.push\_back(Vector3D(1.5, -0.5, 0, 1));

points.push\_back(Vector3D(0.5, -1.5, 0, 1));

points.push\_back(Vector3D(0, -2, 0, 1));

points.push\_back(Vector3D(-0.5, -1.5, 0, 1));

points.push\_back(Vector3D(-1.5, -0.5, 0, 1));

points.push\_back(Vector3D(-2, 0, 0, 1));

points.push\_back(Vector3D(-1.5, 0.5, 0, 1));

points.push\_back(Vector3D(-0.5, 1.5, 0, 1));

points.push\_back(Vector3D(0, 0, 0, 1));

points.push\_back(Vector3D(0, 0.5, 0, 1));

points.push\_back(Vector3D(-0.25, 0.5, 0, 1));

points.push\_back(Vector3D(0.25, 0.5, 0, 1));

current\_pos.resize(points.size());

for (int i = 0; i < 11; i++)

lines.push\_back(Line3D(&current\_pos[i], &current\_pos[i+1]));

lines.push\_back(Line3D(&current\_pos[0], &current\_pos[11]));

for (int i = 12; i < 23; i++)

lines.push\_back(Line3D(&current\_pos[i], &current\_pos[i+1]));

lines.push\_back(Line3D(&current\_pos[12], &current\_pos[23]));

lines.push\_back(Line3D(&current\_pos[24], &current\_pos[27]));

lines.push\_back(Line3D(&current\_pos[27], &current\_pos[30]));

lines.push\_back(Line3D(&current\_pos[30], &current\_pos[33]));

lines.push\_back(Line3D(&current\_pos[33], &current\_pos[24]));

for (int i = 24; i < 36; i++){

lines.push\_back(Line3D(&current\_pos[i], &current\_pos[i-12]));

lines.push\_back(Line3D(&current\_pos[i], &current\_pos[i-24]));

}

lines.push\_back(Line3D(&current\_pos[27], &current\_pos[16]));

for (int i = 37; i < 40; i++)

lines.push\_back(Line3D(&current\_pos[36], &current\_pos[i]));

}

Object3D::~Object3D() {}

void Object3D::draw(QPainter& painter, Matrix3D M3D) {

// Рассчитываем текущую позицию каждой точки

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

current\_pos[i] = M3D \* points[i];

// Рисуем отрезки

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

lines[i].draw(painter);

}

void Object3D::draw3D(QPainter& painter, Matrix3D M3D, Matrix3D T, QPointF center) {

// Рассчитываем текущую позицию каждой точки

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

current\_pos[i] = T \* (M3D \* points[i]).normalized();

// Рисуем отрезки

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

lines[i].draw(painter);

}

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPainter>

#include <QMouseEvent>

#include <vector>

#include <iostream>

#include "matrix.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private:

Ui::MainWindow \*ui;

double scl;

double angleX;

double angleY;

double angleZ;

int index;

QPointF mousePosition;

void *paintEvent*(QPaintEvent \*event);

void *resizeEvent*(QResizeEvent \*event);

void *mousePressEvent*(QMouseEvent \*event);

void *mouseMoveEvent*(QMouseEvent \*event);

};

#endif // MAINWINDOW\_H

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QComboBox>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

index = 0;

connect(ui->comboBox, static\_cast<void(QComboBox::\*)(int)>(&QComboBox::highlighted),

[=](int index){ this->index = index; repaint(); });

}

MainWindow::~*MainWindow*(){

delete ui;

}

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

mousePosition.setX(event->x());

mousePosition.setY(event->y());

std::cout << mousePosition.x() << " " << mousePosition.y() << std::endl;

}

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

if (event->buttons() == Qt::LeftButton && event->x() > width() / 2 && event->y() > height() / 2) {

double mouseMoveX = event->x() - mousePosition.x();

double mouseMoveY = event->y() - mousePosition.y();

angleX += mouseMoveX / 100;

angleY += mouseMoveY / 100;

angleZ += (mouseMoveY + mouseMoveX)/ 250;

repaint();

}

}

void MainWindow::*resizeEvent*(QResizeEvent \*event) {

if (width() < height())

scl = 0.05 \* width();

else

scl = 0.05 \* height();

repaint();

}

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

QPainter painter(this);

ui->comboBox->move(width()/2+1, height()/2-ui->comboBox->height()/2-1);

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

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

QPointF centerLeftUp(width() / 4, height() / 4);

QPointF centerLeftDown(width() / 4, height() \* 3 / 4);

QPointF centerRightUp(width() \* 3 / 4, height() / 4);

QPointF centerRightDown(width() \* 3 / 4, height() \* 3 / 4);

Object3D object;

Matrix3D M3D;

Matrix3D T;

// Вид верх-лево

M3D = Matrix3D::translate(centerLeftUp.x(), centerLeftUp.y(), 0)

\* Matrix3D::scale(scl, scl, scl);

object.draw(painter, M3D);

// Вид верх-право

M3D = Matrix3D::translate(centerRightUp.x(), centerRightUp.y(), 0)

\* Matrix3D::rotateY(90)

\* Matrix3D::scale(scl, scl, scl);

object.draw(painter, M3D);

// Вид лево-низ

M3D = Matrix3D::translate(centerLeftDown.x(), centerLeftDown.y(), 0)

\* Matrix3D::rotateX(90)

\* Matrix3D::scale(scl, scl, scl);

object.draw(painter, M3D);

// Вид право-низ

Matrix3D P;

switch (index) {

case 0:

P = Matrix3D::orthoProjXY(); break;

case 1:

P = Matrix3D::cabinetProj(); break;

case 2:

P = Matrix3D::freeProj(); break;

case 3:

P = Matrix3D::centralProj(150); break;

default:

break;

}

M3D = P

\* Matrix3D::rotateY(angleX)

\* Matrix3D::rotateX(angleY)

\* Matrix3D::rotateX(angleZ)

\* Matrix3D::scale(scl, scl, scl);

T = Matrix3D::translate(centerRightDown.x(), centerRightDown.y(), 0);

object.draw3D(painter, M3D, T, centerRightDown);

std::vector<Vector3D> axis;

axis.push\_back(Vector3D(0, 0, 0, 1));

axis.push\_back(Vector3D(1, 0, 0, 1));

axis.push\_back(Vector3D(0, 1, 0, 1));

axis.push\_back(Vector3D(0, 0, 1, 1));

// Поворачиваем оси

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

axis[i] = M3D \* axis[i];

// Нормализуем оси

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

axis[i] = Vector3D(axis[i].X() / axis[i].H(),

axis[i].Y() / axis[i].H(),

axis[i].Z() / axis[i].H(),

1);

}

// Перемещаем оси

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

axis[i] = T \* axis[i];

QPen pen;

pen.setWidth(2);

pen.setColor(Qt::green);

painter.setPen(pen);

painter.drawLine(axis[0].X(), axis[0].Y(), axis[1].X(), axis[1].Y());

pen.setColor(Qt::red);

painter.setPen(pen);

painter.drawLine(axis[0].X(), axis[0].Y(), axis[2].X(), axis[2].Y());

pen.setColor(Qt::blue);

painter.setPen(pen);

painter.drawLine(axis[0].X(), axis[0].Y(), axis[3].X(), axis[3].Y());

}

main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

