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 № 3

Аффинные преобразования на плоскости

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

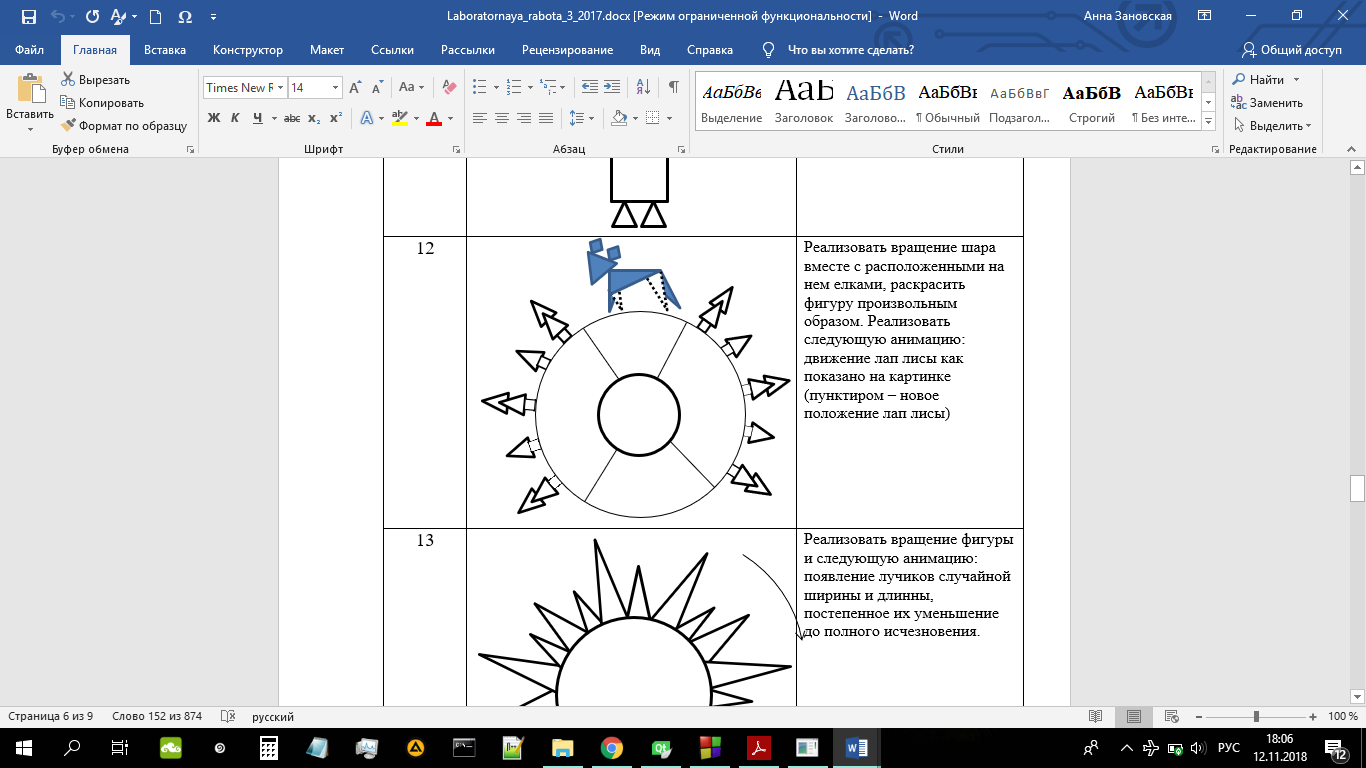
Белгород

2018

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

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 таблицы №1.



Текст программы

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPainter>

#include <QPointF>

#include <QLine>

#include <QPen>

#include <QRectF>

#include <QTimer>

#include "vector.h"

#include "matrix.h"

#include "time.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private:

Ui::MainWindow \*ui;

QTimer \*timer;

void *paintEvent*(QPaintEvent\* event);

double angle;

QPolygonF foxLeft[8];

QPolygonF foxRight[8];

private slots:

void timerUpd();

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

timer = new QTimer();

connect(timer,SIGNAL (timeout()), this, SLOT(timerUpd()));

timer->start(47);

angle = 0;

}

MainWindow::~*MainWindow*()

{

delete ui;

}

void MainWindow::timerUpd()

{

angle++;

repaint();

}

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

{

QPainter painter(this);// Создаём объект-живописец

QPointF lt(width()/2,0);

QPointF lb(width()/2,height());

painter.drawLine(lt,lb);

//координаты центра и радиуса окружности

QPointF center1(width()/4, height()/2);

double Radius1, Radius2;

if (center1.x()<center1.y()){

Radius1 = 0.5\*(center1.x()-10); Radius2 = 0.25\*(center1.x()-10);

}

else {

Radius1 = 0.5\*(center1.y()-10); Radius2 = 0.25\*(center1.y()-10);

}

Matrix T1 = Matrix::Translate(-lt.x(),0);

Matrix R1 = Matrix::ReflectionY();

Matrix T2 = Matrix::Translate(lt.x(),0);

Matrix M1, M3=T2\*R1\*T1;

M1.ZeroMatr();

M1=T2\*R1;

M1=M1\*T1;//матрица для отражения относительно центральной вертикали

Vector v(center1.x(),center1.y());

v=M1\*v;

QPointF center2(v.getX(),v.getY());

double k;

QColor dGreen (17,100,50);

QColor brown (102, 51, 0);

painter.setBrush(Qt::white);

int t = -angle-45;

QRect rect = QRect(center1.x()-Radius1, center1.y()-Radius1, Radius1\*2, Radius1\*2);

painter.setPen(QPen(Qt::black, 1, Qt::SolidLine, Qt::FlatCap));

painter.drawPie(rect, 16\*t, 16\*90);

painter.drawPie(rect, 16\*(t+90), 16\*90);

painter.drawPie(rect, 16\*(t+180), 16\*90);

painter.drawPie(rect, 16\*(t+270), 16\*90);

t \*= -1;

rect = QRect(center2.x()-Radius1, center2.y()-Radius1, Radius1\*2, Radius1\*2);

painter.drawPie(rect, 16\*t, 16\*90);

painter.drawPie(rect, 16\*(t+90), 16\*90);

painter.drawPie(rect, 16\*(t+180), 16\*90);

painter.drawPie(rect, 16\*(t+270), 16\*90);

painter.setBrush(Qt::darkGray);

painter.drawEllipse(center1,Radius2,Radius2);

painter.drawEllipse(center2,Radius2,Radius2);

double hTree = Radius1/5;

painter.setBrush(brown);

k = -42;

QPolygonF arrTree[10];

for (int i = 0; i < 5; i++)

{

QPolygonF tree;

tree << QPointF(center1.x() + Radius1 \* cos((k) \* M\_PI / 180), center1.y() + Radius1 \* sin((k) \* M\_PI / 180)) <<

QPointF(center1.x() + (hTree + Radius1) \* cos((k) \* M\_PI / 180), center1.y() + (hTree + Radius1) \* sin((k) \* M\_PI / 180)) <<

QPointF(center1.x() + (hTree + Radius1) \* cos((k+5) \* M\_PI / 180), center1.y() + (hTree + Radius1) \* sin((k+5) \* M\_PI / 180))<<

QPointF(center1.x() + Radius1 \* cos((k+5) \* M\_PI / 180), center1.y() + Radius1 \* sin((k+5)\* M\_PI / 180));

arrTree[i] = tree;

k+=20;

}

k = 137;

for (int i = 0; i < 5; i++)

{

QPolygonF tree;

tree << QPointF(center1.x() + Radius1 \* cos((k) \* M\_PI / 180), center1.y() + Radius1 \* sin((k) \* M\_PI / 180)) <<

QPointF(center1.x() + (hTree + Radius1) \* cos((k) \* M\_PI / 180), center1.y() + (hTree + Radius1) \* sin((k) \* M\_PI / 180)) <<

QPointF(center1.x() + (hTree + Radius1) \* cos((k+5) \* M\_PI / 180), center1.y() + (hTree + Radius1) \* sin((k+5) \* M\_PI / 180))<<

QPointF(center1.x() + Radius1 \* cos((k+5) \* M\_PI / 180), center1.y() + Radius1 \* sin((k+5)\* M\_PI / 180));

arrTree[i+5] = tree;

k+=20;

}

painter.setBrush(dGreen);

k = -45;

QPolygonF arrList[15];

int count = 0;

for(int i = 0; i < 5; i++)

{

QPolygonF tr;

tr << QPointF(center1.x() + (hTree + Radius1) \* cos((k) \* M\_PI / 180), center1.y() + (hTree + Radius1) \* sin((k) \* M\_PI / 180)) <<

QPointF(center1.x() + (hTree\*2.5 + Radius1) \* cos((k+6) \* M\_PI / 180), center1.y() + (hTree\*2.5 + Radius1) \* sin((k+6) \* M\_PI / 180))<<

QPointF(center1.x() + (hTree + Radius1) \* cos((k+12) \* M\_PI / 180), center1.y() + (hTree + Radius1) \* sin((k+12)\* M\_PI / 180));

arrList[count] = tr;

count++;

if ((int)i%2 == 0)

{

QPolygonF htr;

htr << QPointF(center1.x() + (hTree\*1.75 + Radius1) \* cos((k+1) \* M\_PI / 180), center1.y() + (hTree\*1.75 + Radius1) \* sin((k+1) \* M\_PI / 180)) <<

QPointF(center1.x() + (hTree\*3 + Radius1) \* cos((k+6) \* M\_PI / 180), center1.y() + (hTree\*3 + Radius1) \* sin((k+6) \* M\_PI / 180))<<

QPointF(center1.x() + (hTree\*1.75 + Radius1) \* cos((k+11) \* M\_PI / 180), center1.y() + (hTree\*1.75 + Radius1) \* sin((k+11)\* M\_PI / 180));

arrList[count] = htr;

count++;

}

k+=20;

}

k = 134;

for(int i = 0; i < 5; i++)

{

QPolygonF tr;

tr << QPointF(center1.x() + (hTree + Radius1) \* cos((k) \* M\_PI / 180), center1.y() + (hTree + Radius1) \* sin((k) \* M\_PI / 180)) <<

QPointF(center1.x() + (hTree\*2.5 + Radius1) \* cos((k+6) \* M\_PI / 180), center1.y() + (hTree\*2.5 + Radius1) \* sin((k+6) \* M\_PI / 180))<<

QPointF(center1.x() + (hTree + Radius1) \* cos((k+12) \* M\_PI / 180), center1.y() + (hTree + Radius1) \* sin((k+12)\* M\_PI / 180));

arrList[count] = tr;

count++;

if ((int)i%2 == 1)

{

QPolygonF htr;

htr << QPointF(center1.x() + (hTree\*1.75 + Radius1) \* cos((k+1) \* M\_PI / 180), center1.y() + (hTree\*1.75 + Radius1) \* sin((k+1) \* M\_PI / 180)) <<

QPointF(center1.x() + (hTree\*3 + Radius1) \* cos((k+6) \* M\_PI / 180), center1.y() + (hTree\*3 + Radius1) \* sin((k+6) \* M\_PI / 180))<<

QPointF(center1.x() + (hTree\*1.75 + Radius1) \* cos((k+11) \* M\_PI / 180), center1.y() + (hTree\*1.75 + Radius1) \* sin((k+11)\* M\_PI / 180));

arrList[count] = htr;

count++;

}

k+=20;

}

QColor fire(250,138,48);

QPolygonF foxLeft[8], foxRight[8]; //0 – тело, 1 – голова, 2-3 ушки

foxLeft[0] << QPointF(center1.x() - Radius2/2.5, center1.y() - Radius1\*1.15) << QPointF(center1.x() + Radius2/2, center1.y() - Radius1\*1.25) << QPointF(center1.x() - Radius2/3, center1.y() - Radius1\*1.35);

foxLeft[1] << QPointF(center1.x() - Radius2/1.8, center1.y() - Radius1\*1.4) << QPointF(center1.x() - Radius2/5, center1.y() - Radius1\*1.4) << QPointF(center1.x() - Radius2/2.3, center1.y() - Radius1\*1.25);

foxLeft[2] << QPointF(center1.x() - Radius2/1.7, center1.y() - Radius1\*1.44) << QPointF(center1.x() - Radius2/1.9, center1.y() - Radius1\*1.48) << QPointF(center1.x() - Radius2/2.1, center1.y() - Radius1\*1.44) << QPointF(center1.x() - Radius2/1.9, center1.y() - Radius1\*1.4);

double wHead = (Radius2/1.8 + Radius2/5)/2;

foxLeft[3] << QPointF(center1.x() - Radius2/1.7 + wHead\*0.7, center1.y() - Radius1\*1.44) << QPointF(center1.x() - Radius2/1.9 + wHead\*0.7, center1.y() - Radius1\*1.48) << QPointF(center1.x() - Radius2/2.1 + wHead\*0.7, center1.y() - Radius1\*1.44) << QPointF(center1.x() - Radius2/1.9 + wHead\*0.7, center1.y() - Radius1\*1.4); //buttom

foxLeft[4]<< QPointF(center1.x() - Radius2/2.5, center1.y() - Radius1\*1.15) << QPointF(center1.x() - Radius2/4, center1.y() - Radius1\*1.2) << QPointF(center1.x() - Radius2/2.5, center1.y() - Radius1);

foxLeft[5]<< QPointF(center1.x() + Radius2/3.5, center1.y() - Radius1\*1.25) << QPointF(center1.x() + Radius2/2, center1.y() - Radius1\*1.25) << QPointF(center1.x() + Radius2/2.5, center1.y() - Radius1);

for(int i=0; i<15; i++)

{

T1 = Matrix::Translate(-center1.x(),-center1.y());

T2 = Matrix::Translate(center1.x(),center1.y());

QPolygonF tr2, tr3, tr4, tr5; tr2.clear();

for(int j=0; j<3; j++)

{

Matrix M = Matrix::Translate(center1.x(), center1.y())

\* Matrix::Rotate(angle)

\* Matrix::Translate(-center1.x(), -center1.y());

v.setX(arrList[i][j].x()); v.setY(arrList[i][j].y());

v = M \* v; tr2 << QPointF(v.getX(),v.getY());

v = M3 \* v; tr3 << QPointF(v.getX(),v.getY());

}

if (i < 10)

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

{

Matrix M = Matrix::Translate(center1.x(), center1.y())

\* Matrix::Rotate(angle\*0.5)

\* Matrix::Translate(-center1.x(), -center1.y());

v.setX(arrTree[i][j].x()); v.setY(arrTree[i][j].y());

v = M\*v; tr4 << QPointF(v.getX(),v.getY());

v = M3 \* v; tr5 << QPointF(v.getX(),v.getY());

}

painter.setBrush(dGreen);

painter.drawPolygon(tr2);

painter.drawPolygon(tr3);

painter.setBrush(brown);

painter.drawPolygon(tr4);

painter.drawPolygon(tr5);

}

for(int j=0, i = 4; j<3; j++)

{

Matrix M = Matrix::Translate(center1.x() - Radius2/2.5, center1.y() - Radius1\*1.15)\* Matrix::Rotate(-20)\* Matrix::Translate(-(center1.x() - Radius2/2.5), -(center1.y() - Radius1\*1.15));

v.setX(foxLeft[i][j].x()); v.setY(foxLeft[i][j].y());

v = M \* v;

painter.setBrush(fire);

foxLeft[6] << QPointF(v.getX(),v.getY()); //передняя слева

v = M3 \* v;

foxRight[6] << QPointF(v.getX(),v.getY()); //передняя справа

}

for(int j=0, i = 5; j<3; j++)

{

Matrix M = Matrix::Translate(center1.x() + Radius2/2, center1.y() - Radius1\*1.25)\* Matrix::Rotate(15)\* Matrix::Translate(-(center1.x() + Radius2/2), -(center1.y() - Radius1\*1.25));

v.setX(foxLeft[i][j].x()); v.setY(foxLeft[i][j].y());

v = M \* v;

painter.setBrush(fire);

foxLeft[7] << QPointF(v.getX(),v.getY()); //задняя слева

v = M3 \* v;

foxRight[7] << QPointF(v.getX(),v.getY()); //задняя справа

}

if ((int)angle % 20 >= 10)

{

for (int i = 6; i <= 7; i++){

painter.drawPolygon(foxLeft[i]);

painter.drawPolygon(foxRight[i]);

}

}

else

{

for (int i = 4; i <= 5; i++)

for(int j=0; j<3; j++)

{

v.setX(foxLeft[i][j].x());

v.setY(foxLeft[i][j].y());

v = M1 \* v;

foxRight[i] << QPointF(v.getX(),v.getY());

}

painter.drawPolygon(foxLeft[4]);

painter.drawPolygon(foxLeft[5]);

painter.drawPolygon(foxRight[4]);

painter.drawPolygon(foxRight[5]);

}

painter.setBrush(fire);

painter.drawPolygon(foxLeft[0]);

painter.drawPolygon(foxLeft[1]);

painter.drawPolygon(foxLeft[2]);

painter.drawPolygon(foxLeft[3]);

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

{

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

{

v.setX(foxLeft[i][j].x());

v.setY(foxLeft[i][j].y());

v = M1 \* v;

foxRight[i] << QPointF(v.getX(),v.getY());

}

painter.drawPolygon(foxRight[i]);

}

}

**matrix.h**

#ifndef MATRIX\_H

#define MATRIX\_H

#include "math.h"

#include "vector.h"

class Matrix

{

public:

double M[3][3];

Matrix();

Matrix operator\*(const Matrix& A);

Vector operator\*(const Vector& V);

static Matrix Rotate(double angle)

{

Matrix R;

R.M[0][0] = cos(angle\*M\_PI/180);

R.M[0][1] = -sin(angle\*M\_PI/180);

R.M[0][2] = 0;

R.M[1][0] = sin(angle\*M\_PI/180);

R.M[1][1] = cos(angle\*M\_PI/180);

R.M[1][2] = 0;

R.M[2][0] = 0;

R.M[2][1] = 0;

R.M[2][2] = 1;

return R;

}

static Matrix Scale(double a, double b)

{

Matrix S;

S.M[0][0] = a; S.M[0][1] = 0; S.M[0][1] = 0;

S.M[1][0] = 0; S.M[1][1] = b; S.M[1][2] = 0;

S.M[2][0] = 0; S.M[2][1] = 0; S.M[2][2] = 1;

return S;

}

static Matrix ReflectionX(){

Matrix R;

R.M[0][0] = 1; R.M[0][1] = 0; R.M[0][2] = 0;

R.M[1][0] = 0; R.M[1][1] = -1; R.M[1][2] = 0;

R.M[2][0] = 1; R.M[2][1] = 0; R.M[2][2] = 1;

return R;

}

static Matrix ReflectionY(){

Matrix R;

R.M[0][0] = -1; R.M[0][1] = 0; R.M[0][2] = 0;

R.M[1][0] = 0; R.M[1][1] = 1; R.M[1][2] = 0;

R.M[2][0] = 1; R.M[2][1] = 0; R.M[2][2] = 1;

return R;

}

static Matrix Translate(double dx, double dy){

Matrix T;

T.M[0][0] = 1; T.M[0][1] = 0; T.M[0][2] = dx;

T.M[1][0] = 0; T.M[1][1] = 1; T.M[1][2] = dy;

T.M[2][0] = 0; T.M[2][1] = 0; T.M[2][2] = 1;

return T;

}

void ZeroMatr(){

for (int i=0;i<3;i++)

for (int j=0;j<3;j++)

M[i][j]=0;

}

};

#endif // MATRIX\_H

**matrix.cpp**

#include "matrix.h"

Matrix::Matrix() {

}

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

Matrix res;

res.ZeroMatr();

res.M[0][0] = M[0][0]\*A.M[0][0] + M[0][1]\*A.M[1][0] + M[0][2]\*A.M[2][0];

res.M[0][1] = M[0][0]\*A.M[0][1] + M[0][1]\*A.M[1][1] + M[0][2]\*A.M[2][1];

res.M[0][2] = M[0][0]\*A.M[0][2] + M[0][1]\*A.M[1][2] + M[0][2]\*A.M[2][2];

res.M[1][0] = M[1][0]\*A.M[0][0] + M[1][1]\*A.M[1][0] + M[1][2]\*A.M[2][0];

res.M[1][1] = M[1][0]\*A.M[0][1] + M[1][1]\*A.M[1][1] + M[1][2]\*A.M[2][1];

res.M[1][2] = M[1][0]\*A.M[0][2] + M[1][1]\*A.M[1][2] + M[1][2]\*A.M[2][2];

res.M[2][0] = M[2][0]\*A.M[0][0] + M[2][1]\*A.M[1][0] + M[2][2]\*A.M[2][0];

res.M[2][1] = M[2][0]\*A.M[0][1] + M[2][1]\*A.M[1][1] + M[2][2]\*A.M[2][1];

res.M[2][2] = M[2][0]\*A.M[0][2] + M[2][1]\*A.M[1][2] + M[2][2]\*A.M[2][2];

return res;

}

Vector Matrix:: operator\*(const Vector& V) {

Vector res;

res.ZeroVec();

res.x+=M[0][0]\*V.x+M[0][1]\*V.y+M[0][2]\*V.h;

res.y+=M[1][0]\*V.x+M[1][1]\*V.y+M[1][2]\*V.h;

res.h+=M[2][0]\*V.x+M[2][1]\*V.y+M[2][2]\*V.h;

return res;

}

**vector.h**

#ifndef VECTOR\_H

#define VECTOR\_H

class Vector

{

public:

double x,y,h;

void ZeroVec();

Vector(double x, double y);

Vector();

double getX();

double getY();

void setX(double X);

void setY(double Y);

};

#endif // VECTOR\_H

**vector.cpp**

#include "vector.h"

Vector :: Vector(double v\_x, double v\_y)

{

x = v\_x;

y = v\_y;

h = 1;

}

Vector :: Vector(){

}

void Vector :: ZeroVec(){

x = 0; y = 0; h = 0;

}

double Vector :: getX(){

return x;

}

double Vector :: getY(){

return y;

}

void Vector :: setX(double X){

x = X; h = 1;

}

void Vector :: setY(double Y){

y = Y; h = 1;

}

Результат работы программы

