**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 №3

Дисциплина: Компьютерная графика

Тема: «Аффинные преобразования на плоскости»

Выполнила: ст. группы ПВ-31

Донцов А.А.

Проверил: Осипов О. В.

Белгород 2019

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

**Вывод формул:**

Я использовал формулу поворота матрицы

Вид:

Эту матрицу необходимо умножить на вектор точки, тогда она повернется относительно начала координат на угол angle

**Код программы:**

Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <Qvector>

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

timer = new QTimer();

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

timer->start(15);

}

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

{

delete ui;

}

void **drawSun**(QPainter \*painter, QPointF center, int radius){

painter->setPen(QPen(Qt::yellow, 5, Qt::SolidLine, Qt::RoundCap));

painter->setBrush(Qt::yellow);

painter->drawEllipse(center, radius, radius);

}

void **drawStar**(QPainter \*painter, QPoint center, int kof){

QPoint a1(center);

QPoint a2(center.x() - 5, center.y() + 10);

QPoint a3(center.x() + 5, center.y() + 10);

QPoint a4(center.x() - 16, center.y() + 10);

QPoint a5(center.x() + 16, center.y() + 10);

QPoint a6(center.x() - 10, center.y() + 18);

QPoint a7(center.x() + 10, center.y() + 18);

QPoint a8(center.x() - 15, center.y() + 30);

QPoint a9(center.x() + 15, center.y() + 30);

QPoint a10(center.x(), center.y() + 22);

QVector<QPoint> a(8);

a[0] = a1;

a[1] = a2;

a[2] = a3;

a[3] = a4;

a[4] = a5;

a[5] = a6;

a[6] = a7;

a[7] = a10;

QPolygon pol(a);

painter->setBrush(Qt::yellow);

painter->drawPolygon(pol);

QVector<QPoint> ab(3);

ab[0] = a6;

ab[1] = a8;

ab[2] = a10;

QPolygon pol1(ab);

painter->drawPolygon(pol1);

QVector<QPoint> ac(3);

ac[0] = a7;

ac[1] = a9;

ac[2] = a10;

QPolygon pol2(ac);

painter->drawPolygon(pol2);

painter->setPen(QPen(Qt::black, 3, Qt::SolidLine, Qt::RoundCap));

painter->setBrush(Qt::NoBrush);

painter->drawLine(a1,a2);

painter->drawLine(a1,a3);

painter->drawLine(a2,a4);

painter->drawLine(a3,a5);

painter->drawLine(a4,a6);

painter->drawLine(a5,a7);

painter->drawLine(a6,a8);

painter->drawLine(a7,a9);

painter->drawLine(a8,a10);

painter->drawLine(a9,a10);

}

double angle2 = 0;

void **drawPlanet**(QPainter \*painter, QPointF center, int radius, double angle, double sizePlanet, QColor color1,QColor color2 ,QColor color3, bool flag1, bool flag2){

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

painter->setBrush(Qt::NoBrush);

painter->drawEllipse(center, radius, radius);

Vector2D planet(radius, 0);

planet = Matr::RotatMatrix(angle) \* planet;

QPointF pl(planet.getX()+center.x(), planet.getY()+center.y());

QLinearGradient gradient(pl.x() - sizePlanet, pl.y()- sizePlanet, pl.x() + sizePlanet, pl.y() + sizePlanet);

gradient.setColorAt(0,color1);

gradient.setColorAt(0.5, color2);

gradient.setColorAt(1, color3);

if(flag1 && flag2)

drawPlanet(painter, pl, sizePlanet/2 + sizePlanet, angle2, sizePlanet/3, Qt::gray, Qt::gray,Qt::gray, false, false);

painter->setBrush(gradient);

painter->drawEllipse(pl, sizePlanet, sizePlanet);

}

double angle[8] = {0,20,40,60,80,180,120,140};

double speed[8] = {1,1.25,0.5,0.9,1.5,1,1.4,0.3};

double sizePl[8] = {10,15,30,8,19,20,5,20};

int n = 8;

QColor color1[8] = {Qt::red, Qt::yellow, Qt::green, Qt::black, Qt::blue, Qt::red, Qt::white, Qt::gray};

QColor color2[8] = {Qt::black, Qt::blue, Qt::red, Qt::yellow, Qt::green, Qt::red, Qt::white, Qt::gray};

QColor color3[8] = { Qt::red, Qt::red, Qt::yellow, Qt::green, Qt::black, Qt::blue, Qt::white, Qt::gray};

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

{

QPainter painter(this);

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

int sizeSun;

double kof;

double k;

if(width() < height()) {

sizeSun = 0.05 \* width();

kof = 0.05 \* width();

k = width()/400.0;

}

else {

sizeSun = 0.05 \* height();

kof = 0.05 \* height();

k = height()/400.0;

}

drawSun(&painter,center, sizeSun);

//drawStar(&painter, QPoint(center.x() - sizeSun\*8,40), kof);

int index = sizeSun + kof;

for(int i = 0; i < n; i++){

if(i == 2)

drawPlanet(&painter, center, index, angle[i], sizePl[i]\*k, color1[i] ,color2[i] ,color3[i], true, true);

else

drawPlanet(&painter, center, index, angle[i], sizePl[i]\*k, color1[i] ,color2[i] ,color3[i], false, false);

index += kof;

}

}

void MainWindow::**timerUpdate**()

{

for(int i = 0; i < n; i++){

angle[i] += speed[i];

angle2 += 0.2;

}

repaint();

}

# Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPainter>

#include <QTimer>

#include "matr.h"

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

explicit **MainWindow**(QWidget \*parent = 0);

~***MainWindow***();

private:

Ui::MainWindow \*ui;

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

QTimer\* timer;

private slots:

void **timerUpdate**();

};

#endif // MAINWINDOW\_H

Matr.cpp

#include "matr.h"

#include <QtMath>

#include <QPoint>

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

Matr R;

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

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

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

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

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

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

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

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

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

return R;

}

Vector2D Matr::operator\*(const Vector2D& V)const{

Vector2D R(0,0);

R[0] = M[0][0] \* V[0] + M[0][1] \* V[1] + M[0][2] \* V[2];

R[1] = M[1][0] \* V[0] + M[1][1] \* V[1] + M[1][2] \* V[2];

R[2] = M[2][0] \* V[0] + M[2][1] \* V[1] + M[2][2] \* V[2];

return R;

}

Matr Matr::TransportMatrix(double dx, double dy){

Matr R;

R[0][0] = 1; R[0][1] = 0; R[0][2] = dx;

R[1][0] = 0; R[1][1] = 1; R[1][2] = dy;

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

return R;

}

Matr Matr::RotatMatrix(double angle){

Matr R;

angle = angle\*M\_PI/180;

R[0][0] = cos(angle); R[0][1] = sin(angle); R[0][2] = 0;

R[1][0] = -sin(angle); R[1][1] = cos(angle); R[1][2] = 0;

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

return R;

}

Matr Matr::ReflectionMatrixX(){

Matr R;

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

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

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

return R;

}

Matr Matr::ReflectionMatrixY(){

Matr R;

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

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

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

return R;

}

Matr Matr::ScaleMatrix(double sx, double sy){

Matr R;

R[0][0] = sx; R[0][1] = 0; R[0][2] = 0;

R[1][0] = 0; R[1][1] = sy; R[1][2] = 0;

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

return R;

}

Matr.h

#ifndef MATR\_H

#define MATR\_H

#include <QWidget>

class Vector2D

{

double V[3];

public:

Vector2D(double x, double y){

V[0] = x;

V[1] = y;

V[2] = 1;

}

QPointF Point()

{

return QPointF(V[0],V[1]);

}

double getX(){

return V[0];

}

double getY(){

return V[1];

}

double& operator [] (int index){

return V[index];

}

const double& operator [] (int index)const {

return V[index];

}

};

class Matr

{

double M[3][3];

public:

Matr operator\*(const Matr& A) const;

Vector2D operator\*(const Vector2D& V) const;

static Matr TransportMatrix(double dx, double dy);//матрица перемещения

static Matr RotatMatrix(double angle);//матрица поворота

static Matr ReflectionMatrixX();//матрица отражения X

static Matr ReflectionMatrixY();//матрица отражения Y

static Matr ScaleMatrix(double kx, double ky);//матрица масштабирования

double\* operator [] (int row) {

return M[row];

}

const double\* operator [] (int row) const {

return M[row];

}

};

#endif // MATR\_H

Main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

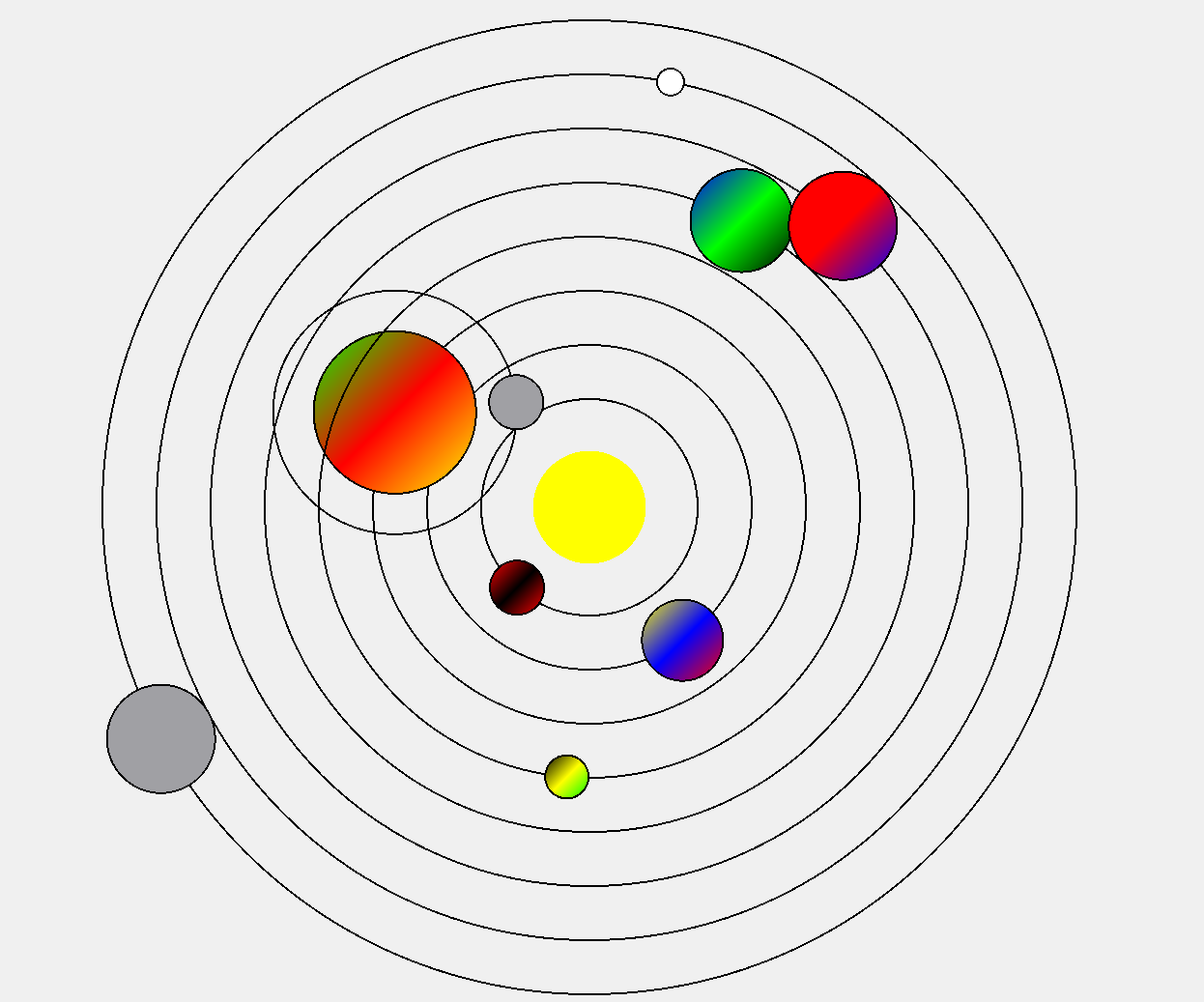
QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

**Результат работы программы:**