МИНИСТЕРСТВО 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)

Лабораторная работа №2
дисциплина «Компьютераная графика »
по теме «Аффинные преобразования на плоскости»

Выполнил: студент группы ВТ-31 Макаров Д.С.
Проверил: Осипов О.В.

Белгород 2019

Лабораторная работа №2

«Аффинные преобразования на плоскости»

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

Вариант 9

Требования к программе.

1. Разработать модуль для выполнения аффинных преобразований на плоскости с помощью матриц. В модуле должны быть реализованы перегруженные операции действия с матрицами (умножение), с векторами и матрицами (умножение вектора-строки на матрицу), конструкторы различных матриц (переноса, масштабирования, переноса, отражения).
2. В программе должна быть предусмотрена возможность ввода пользователем исходных данных (из правой колонки таблицы №1).
3. Разбить окно на 2 равные части. В левой части должна выводиться основная анимация, в правой части её отражение относительно вертикальной линии, проходящей через центр окна.
4. Изображение должно масштабироваться по центру левой и правой части окна с отступом 10 пикселей от границ и вертикальной линии и реагировать на изменение размера окна (см. пример проекта lab_1_CSharp).
5. Раскрасить (залить) примитивы (круги, многоугольники и др.) по собственному усмотрению.

Ход работы

Содержание отчёта 1. Название темы. 2. Цель работы. 3. Постановка задачи. 4. Вывод необходимых формул для построения изображения. Указать какие матрицы используются и в какой последовательности они умножаются для реализации анимации. 5. Текст программы. 6. Результат работы программы (снимки экрана).

Размеры всех фигур зависимы от наибольшей стороны экрана.

Преобразования используемые для построения изображения. 1. Полигоны материков. 1. Афинное преобразование машстабирования. 2. Афинное преобразование отражения. 3. Афинное преобразование перемещения. 2. Полигоны птицы. 1. Афинное преобразование поворота. 2. Афинное преобразование отражения. 3. Афинное преобразование перемещения.
4. Афинное преобразование машстабирования.

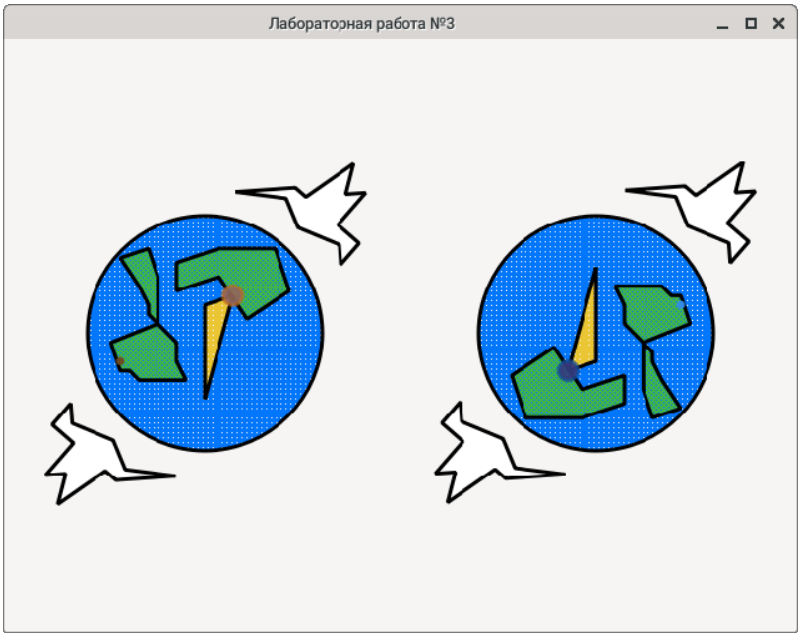

Рис. 1: Пример работы программы

Приложение

Содержимое файла main.cpp

```
#include <QtWidgets>
#include "Draw.hpp"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    DrawArea* drawArea = new DrawArea;
    drawArea->show();
    return a.exec();
}
```

Содержимое файла Matrix3x3.cpp

```
#include "Matrix3x3.hpp"

Vectorx3::Vectorx3(){
    this->vector[0] = 0;
    this->vector[1] = 0;
    this->vector[2] = 0;
};
Vectorx3::Vectorx3(qreal e1,qreal e2, qreal e3){
    this->vector[0] = e1;
    this->vector[1] = e2;
    this->vector[2] = e3;
};
Vectorx3::Vectorx3(QVector<qreal> vect){
    this->vector[0] = vect[0];
    this->vector[1] = vect[1];
    this->vector[2] = vect[2];
};
QVector<qreal> Vectorx3::qVector(){
    return QVector<qreal> {
        this->vector[0],
        this->vector[1],
        this->vector[2]
    };
};

Vectorx3::Vectorx3(QPointF point){
    this->vector[0]=point.x();
    this->vector[1]=point.y();
    this->vector[2]=1;
};


void Vectorx3::fromQVector(QVector<qreal> vectx3){
    this->vector[0] = vectx3[0];
    this->vector[1] = vectx3[1];
    this->vector[2] = vectx3[2];
};
void Vectorx3::fromQReal(qreal e1,qreal e2,qreal e3){
    this->vector[0] = e1;
    this->vector[1] = e2;
    this->vector[2] = e3;
};

qreal Vectorx3::x(){
```

```
    return this->vector[0];
};
qreal Vectorx3::y(){
    return this->vector[1];
};
qreal Vectorx3::k(){
    return this->vector[2];
};

QPointF Vectorx3::qPointF(){
    return QPointF(x(),y());
};
void Vectorx3::fromQPointF(QPointF point){
    Vectorx3(point.x(),point.y(),1);
};

Matrix3x3 mirrorMatrix3x3(bool mirrorX=false,bool mirrorY=false){
    qreal xMul = 1,yMul = 1;
    if(mirrorX) xMul = -1;
    if(mirrorY) yMul = -1;
    return Matrix3x3(
        xMul,0,0,
        0,yMul,0,
        0,0,1
    );
};
Matrix3x3 rotateMatrix3x3(qreal alpha=0){
    return Matrix3x3(
        qCos(alpha),-qSin(alpha),0,
        qSin(alpha),qCos(alpha),0,
        0,0,1
    );
};
Matrix3x3 scaleMatrix3x3(qreal scaleX=1,qreal scaleY=1){
    return Matrix3x3(
        scaleX,0,0,
        0,scaleY,0,
        0,0,1
    );
};
Matrix3x3 moveMatrix3x3(qreal deltaX=0,qreal deltaY=0){
    return Matrix3x3(
        1,0,deltaX,
        0,1,deltaY,
        0,0,1
    );
};

Matrix3x3::Matrix3x3(qreal e1,qreal e2,qreal e3,
            qreal e4,qreal e5,qreal e6,
            qreal e7,qreal e8,qreal e9)
{
    matrix[0][0] = e1;
    matrix[0][1] = e2;
    matrix[0][2] = e3;
    matrix[1][0] = e4;
    matrix[1][1] = e5;
    matrix[1][2] = e6;
    matrix[2][0] = e7;
    matrix[2][1] = e8;
```

```
    matrix[2][2] = e9;
};
```

Содержимое файла Draw.cpp

```
#include "Draw.hpp"

void DrawArea::paintEvent(QPaintEvent* event){
    //проверка полей
    if (this->width() < 40 || this->height() < 40) return;

    bool newFlash = (QRandomGenerator::global()->generate() % 100) < 3;


    QPainter *painter = new QPainter(this);
    //установка параметров от таймера
    qreal angle = to_rad(state%360);

    //антиалиасинг
    painter->setRenderHint(QPainter::HighQualityAntialiasing);

    //установка радиусов
    if(this->height()>this->width()){
        mainRadius = this->width()/5;
    }
    else{
        mainRadius = this->height()/5;
    };
    qreal outRadius = mainRadius+mainRadius/5;
    qreal helpPointRadius = outRadius/5;


    QPointF center(0,0); // center(this->width()/2,this->height()/2)
    QPointF mirrorCenter(0,0);
    center = movePoint(center,this->width()/4,this->height()/2);
    mirrorCenter = movePoint(mirrorCenter,3*this->width()/4,this->height()/2);

    QPolygonF bird;
    QPolygonF mirrorBird;
    bird<<(QPointF(0,0))<<(QPointF(3,1))<<(QPointF(4,0))<<(QPointF(4,2))
        <<(QPointF(2,2))<<(QPointF(2,5))<<(QPointF(1,4))<<(QPointF(0,6))
        <<(QPointF(-1,2))<<(QPointF(-2,2))<<(QPointF(-5,-1))<<(QPointF(-2,1));
    bird = scaleQPolygonF(bird,helpPointRadius/2,helpPointRadius/2);
    bird = mirrorQPolygonF(bird,0,1);
    bird = moveQPolygonF(bird,1,-outRadius);
    bird = rotateQPolygonF(bird,angle);
    mirrorBird = bird;
    mirrorBird = mirrorQPolygonF(mirrorBird,1,1);
    mirrorBird = moveQPolygonF(mirrorBird,3*this->width()/4,this->height()/2);
    bird = moveQPolygonF(bird,this->width()/4,this->height()/2);

    //Задание на защиту
    QPolygonF birdTask;
    QPolygonF mirrorBirdTask;
    birdTask<<(QPointF(0,0))<<(QPointF(3,1))<<(QPointF(4,0))<<(QPointF(4,2))
        <<(QPointF(2,2))<<(QPointF(2,5))<<(QPointF(1,4))<<(QPointF(0,6))
        <<(QPointF(-1,2))<<(QPointF(-2,2))<<(QPointF(-5,-1))<<(QPointF(-2,1));
    birdTask = scaleQPolygonF(birdTask,helpPointRadius/2,helpPointRadius/2);
    birdTask = mirrorQPolygonF(birdTask,1,0);
    birdTask = moveQPolygonF(birdTask,1,outRadius);
```

```
birdTask = rotateQPolygonF(birdTask,angle);
mirrorBirdTask = birdTask;
mirrorBirdTask = mirrorQPolygonF(mirrorBirdTask,1,1);
mirrorBirdTask = moveQPolygonF(mirrorBirdTask,3*this->width()/4,this->height()/2);
birdTask = moveQPolygonF(birdTask,this->width()/4,this->height()/2);



QPolygonF land1;
QPolygonF mirrorLand1;
land1<<QPointF(2,-3)<<QPointF(2,-5)<<QPointF(-1,-6)
     <<QPointF(-5,-6)<<QPointF(-6,-3)<<QPointF(-3,-1)
     <<QPointF(-3,-1)<<QPointF(-1,-4);
land1 = scaleQPolygonF(land1,helpPointRadius/2,helpPointRadius/2);
land1 = mirrorQPolygonF(land1,1,0);
mirrorLand1 = land1;
mirrorLand1 = mirrorQPolygonF(mirrorLand1,1,1);
mirrorLand1 = moveQPolygonF(mirrorLand1,3*this->width()/4,this->height()/2);
land1 = moveQPolygonF(land1,this->width()/4,this->height()/2);

QPolygonF land2;
QPolygonF mirrorLand2;
land2<<QPointF(0,7)<<QPointF(0,-3)<<QPointF(3,-4);
land2 = scaleQPolygonF(land2,helpPointRadius/3,helpPointRadius/3);
mirrorLand2 = land2;
mirrorLand2 = mirrorQPolygonF(mirrorLand2,1,1);
mirrorLand2 = moveQPolygonF(mirrorLand2,3*this->width()/4,this->height()/2);
land2 = moveQPolygonF(land2,this->width()/4,this->height()/2);

QPolygonF land3;
QPolygonF mirrorLand3;
land3<<QPointF(5,-6)<<QPointF(6,-9)<<QPointF(9,-8)
     <<QPointF(7,-5)<<QPointF(6,-3)<<QPointF(6,-2)
     <<QPointF(5,-1);
land3 = scaleQPolygonF(land3,helpPointRadius/3,helpPointRadius/3);
land3 = mirrorQPolygonF(land3,1,0);
mirrorLand3 = land3;
mirrorLand3 = mirrorQPolygonF(mirrorLand3,1,1);
mirrorLand3 = moveQPolygonF(mirrorLand3,3*this->width()/4,this->height()/2);
land3 = moveQPolygonF(land3,this->width()/4,this->height()/2);

QPolygonF land4;
QPolygonF mirrorLand4;
land4<<QPointF(5,-1)<<QPointF(3,1)<<QPointF(3,3)<<QPointF(2,5)
     <<QPointF(7,5)<<QPointF(8,4)<<QPointF(9,4)<<QPointF(10,1)
     <<QPointF(5,-1);
land4 = scaleQPolygonF(land4,helpPointRadius/3,helpPointRadius/3);
land4 = mirrorQPolygonF(land4,1,0);
mirrorLand4 = land4;
mirrorLand4 = mirrorQPolygonF(mirrorLand4,1,1);
mirrorLand4 = moveQPolygonF(mirrorLand4,3*this->width()/4,this->height()/2);
land4 = moveQPolygonF(land4,this->width()/4,this->height()/2);

QPolygonF flashPoints;
QPolygonF mirrorFlashPoints;
flashPoints<<QPointF(1,-4)<<QPointF(-5,-3)<<QPointF(-3,-4)<<QPointF(-3,-2)
           <<QPointF(7,0)<<QPointF(4,6)<<QPointF(9,3)<<QPointF(2,2);
flashPoints = scaleQPolygonF(flashPoints,helpPointRadius/3,helpPointRadius/3);
flashPoints = mirrorQPolygonF(flashPoints,1,0);
mirrorFlashPoints = flashPoints;
```

```
    mirrorFlashPoints = mirrorQPolygonF(mirrorFlashPoints,1,1);
    mirrorFlashPoints = moveQPolygonF(mirrorFlashPoints,3*this->width()/4,this->height()/2);
    flashPoints = moveQPolygonF(flashPoints,this->width()/4,this->height()/2);

    if(newFlash){
        int flashPos = (QRandomGenerator::global()->generate() % flashPoints.size());
        flashList.append(Flash(QPointF(flashPoints[flashPos])));
        mirrorFlashList.append(Flash(QPointF(mirrorFlashPoints[flashPos])));
    }


    //кисть для обводки
    QPen strokePen = QPen(QColor(0,0,0));
    strokePen.setWidth(3);
    //кисть воды
    QBrush waterBrush = QBrush(QColor(3,119,252),Qt::Dense1Pattern);
    QBrush whiteBrush = QBrush(QColor(255,255,255));
    QBrush greenBrush = QBrush(QColor(64, 179, 77),Qt::Dense2Pattern);
    QBrush desertBrush = QBrush(QColor(235, 198, 52),Qt::Dense1Pattern);

    painter->setPen(strokePen);
    painter->setBrush(waterBrush);
    painter->drawEllipse(getCircleArea(center,mainRadius));
    painter->drawEllipse(getCircleArea(mirrorCenter,mainRadius));

    painter->setBrush(whiteBrush);
    painter->drawPolygon(bird);
    painter->drawPolygon(birdTask);
    painter->drawPolygon(mirrorBird);
    painter->drawPolygon(mirrorBirdTask);

    painter->setBrush(greenBrush);
    painter->drawPolygon(land4);
    painter->drawPolygon(land3);
    painter->drawPolygon(land1);
    painter->drawPolygon(mirrorLand4);
    painter->drawPolygon(mirrorLand3);
    painter->drawPolygon(mirrorLand1);

    painter->setBrush(desertBrush);
    painter->drawPolygon(mirrorLand2);
    painter->drawPolygon(land2);

    for(auto i = flashList.begin();i!=flashList.end();i++){
        (*i)(painter,0);
        if((*i).state() == 0) flashList.erase(i);
    };
    for(auto i = mirrorFlashList.begin();i!=mirrorFlashList.end();i++){
        (*i)(painter,0);
        if((*i).state() == 0) mirrorFlashList.erase(i);
    }
    painter->end();
};

QPointF movePoint(QPointF point,qreal deltaX,qreal deltaY){
    return (moveMatrix3x3(deltaX,deltaY)*Vectorx3(point)).qPointF();
};
QPointF rotatePoint(QPointF point,qreal angle){
    return (rotateMatrix3x3(angle)*Vectorx3(point)).qPointF();
};
```

```
QPointF scalePoint(QPointF point,qreal scaleX,qreal scaleY){
    return (scaleMatrix3x3(scaleX,scaleY)*Vectorx3(point)).qPointF();
};
QPointF mirrorPoint(QPointF point,bool x,bool y){
    return (mirrorMatrix3x3(x,y)*Vectorx3(point)).qPointF();
};

QRectF getCircleArea(QPointF center,qreal radius){
    QRectF area = QRectF(center.x() - radius, center.y() - radius, radius*2, radius*2);
    return area;
};

void DrawArea::animate(){
    state++;
    repaint();
};

DrawArea::DrawArea(){
        QTimer* timer = new QTimer;
        QWidget::connect(timer,SIGNAL(timeout()),this,SLOT(animate()));
        timer->start(1);
        this->setWindowTitle("Лабораторная работа №3");
};

qreal to_rad(qreal angle){
    return angle*M_PI/180;
};

QPolygonF rotateQPolygonF(QPolygonF poly,qreal angle){
    QPolygonF result;
    for(auto i=poly.begin();i!=poly.end();i++){
        result<<rotatePoint(*i,angle);
    };
    return result;
};

QPolygonF moveQPolygonF(QPolygonF poly,qreal deltaX,qreal deltaY){
    QPolygonF result;
    for(auto i=poly.begin();i!=poly.end();i++){
        result<<movePoint(*i,deltaX,deltaY);
    };
    return result;
};
QPolygonF mirrorQPolygonF(QPolygonF poly,bool x,bool y){
    QPolygonF result;
    for(auto i=poly.begin();i!=poly.end();i++){
        result<<mirrorPoint(*i,x,y);
    };
    return result;
};
QPolygonF scaleQPolygonF(QPolygonF poly,qreal scaleX,qreal scaleY){
    QPolygonF result;
    for(auto i=poly.begin();i!=poly.end();i++){
        result<<scalePoint(*i,scaleX,scaleY);
    };
    return result;
};
```

Содержимое файла Draw.hpp

```
#include <QtWidgets>
```

```
#include "Matrix3x3.hpp"

#pragma once

static qreal mainRadius;

QPointF movePoint(QPointF point,qreal deltaX,qreal deltaY);
QPointF rotatePoint(QPointF point,qreal angle);
QPointF scalePoint(QPointF point,qreal scaleX,qreal scaleY);
QPointF mirrorPoint(QPointF point,bool x,bool y);

QPolygonF rotateQPolygonF(QPolygonF poly,qreal angle);
QPolygonF moveQPolygonF(QPolygonF poly,qreal deltaX,qreal deltaY);
QPolygonF mirrorQPolygonF(QPolygonF poly,bool x,bool y);
QPolygonF scaleQPolygonF(QPolygonF poly,qreal scaleX,qreal scaleY);

QRectF getCircleArea(QPointF center,qreal radius);
bool pointInPolygon(QPolygonF poly,QPointF point);
qreal to_rad(qreal angle);

class Flash{
    private:
        int _state;
        QPointF _point;
        int red;
        int green;
        int blue;
        int scale;
    public:
        Flash(QPointF point){
            _point = point;
            _state = 100;
            red = (QRandomGenerator::global()->generate())%255;
            blue = (QRandomGenerator::global()->generate())%255;
            green = (QRandomGenerator::global()->generate())%255;
        };
        QPointF point(){return _point;};
        int state(){return _state;};
        void operator() (QPainter* painter,bool mirror){
            if(_state != 0){
                QColor color(red,green,blue,255-_state);
                scale = _state*mainRadius*0.001;
                QBrush flashBrush = QBrush(color);
                QPen flashPen = QPen(color);
                flashPen.setWidth(3);
                QPointF drawPoint(0,0);

                if(mirror){
                    drawPoint = movePoint(drawPoint,-_point.x(),_point.y());
                    drawPoint = mirrorPoint(drawPoint,0,1);
                    drawPoint = movePoint(drawPoint,2*_point.x(),1);
                }else{
                    drawPoint = movePoint(drawPoint,_point.x(),_point.y());
                };

                QBrush tempBrush = painter->brush();
                QPen tempPen = painter->pen();
                painter->setBrush(flashBrush);
                painter->setPen(flashPen);
                painter->drawEllipse(getCircleArea(drawPoint,scale));
```

```
                painter->setBrush(tempBrush);
                painter->setPen(tempPen);

                _state--;
            };
        };
};

class DrawArea:public QWidget{
Q_OBJECT
public:
    private:
        unsigned state = 0;
        QList<Flash> flashList;
        QList<Flash> mirrorFlashList;
    public slots:
        void animate();
    public:
    DrawArea();
    void paintEvent(QPaintEvent* event);
};
```

Содержимое файла Matrix3x3.hpp

```
#include <QtWidgets>
#include <QtMath>
class Vectorx3{
    private:
        qreal vector[3];
    public:
        Vectorx3();
        Vectorx3(qreal e1,qreal e2, qreal e3);
        Vectorx3(QVector<qreal> vect);
        Vectorx3(QPointF point);
        QVector<qreal> qVector();
        QPointF qPointF();
        void fromQPointF(QPointF point);
        void fromQVector(QVector<qreal> vectx3);
        void fromQReal(qreal e1,qreal e2,qreal e3);
        qreal operator [](int index){
            return vector[index];
        };
        qreal x();
        qreal y();
        qreal k();
};

class Matrix3x3{
    private:
        qreal matrix[3][3];
    public:
        Matrix3x3();
        Matrix3x3(qreal e1,qreal e2,qreal e3,
                  qreal e4,qreal e5,qreal e6,
                  qreal e7,qreal e8,qreal e9);

        QVector<qreal> operator *(QVector<qreal> m2){
            return QVector<qreal>({
                this->matrix[0][0]*m2[0]+this->matrix[0][1]*m2[1]+this->matrix[0][2]*m2[2],
                this->matrix[1][0]*m2[0]+this->matrix[1][1]*m2[1]+this->matrix[1][2]*m2[2],
```

```
                this->matrix[2][0]*m2[0]+this->matrix[2][1]*m2[1]+this->matrix[2][2]*m2[2]
            });
        };

        Vectorx3 operator *(Vectorx3 m2){
            return Vectorx3(
                this->matrix[0][0]*m2[0]+this->matrix[0][1]*m2[1]+this->matrix[0][2]*m2[2],
                this->matrix[1][0]*m2[0]+this->matrix[1][1]*m2[1]+this->matrix[1][2]*m2[2],
                this->matrix[2][0]*m2[0]+this->matrix[2][1]*m2[1]+this->matrix[2][2]*m2[2]
            );
        };

        qreal* operator [](int index){
            return matrix[index];
        };
};


Matrix3x3 mirrorMatrix3x3(bool mirrorX,bool mirrorY);
Matrix3x3 rotateMatrix3x3(qreal alpha);
Matrix3x3 scaleMatrix3x3(qreal scaleX,qreal scaleY);
Matrix3x3 moveMatrix3x3(qreal deltaX,qreal deltaY);
```