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)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №3

по дисциплине: Компьютерная графика

тема: «Аффинные преобразования на плоскости»

Выполнил: ст. группы ВТ-32

Воскобойников И. С.

Проверил: Гибкин Ю. С.

Белгород 2020 г.

Цель работы: получение навыков выполнения аффинных преобразований на плоскости и создание 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 таблице №1.

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.

amorph.h

#ifndef AMORPH\_H

#define AMORPH\_H

#include <QMatrix>

#include <QPointF>

class Amorph

{

QMatrix reflectMatrixOX;

QMatrix reflectMatrixOY;

public:

Amorph();

// Отражение

QPointF reflection(QPointF p, bool is\_ox=true);

// Поворот

QPointF rotate(QPointF p, int a);

// Масштабирование

QPointF resize(QPointF p, int kx, int ky);

// Перенос

QPointF moveTo(QPointF p, int dx, int dy);

};

#endif // AMORPH\_H

amorph.cpp

#include "amorph.h"

Amorph::Amorph()

{

reflectMatrixOY = QMatrix(-1, 0, 0, 1, 0, 0);

reflectMatrixOX = QMatrix(1, 0, 0, -1, 0, 0);

}

QPointF Amorph::reflection(QPointF p, bool is\_ox){

QMatrix xy = QMatrix(p.x(), 0, p.y(), 0, 1, 0);

QMatrix tmp;

if (is\_ox)

tmp = reflectMatrixOX \* xy;

else

tmp = reflectMatrixOY \* xy;

return QPointF(tmp.m11(), tmp.m21());

}

QPointF Amorph::rotate(QPointF p, int a){

QMatrix xy = QMatrix(p.x(), 0, p.y(), 0, 1, 0);

double agile = (a % 360) / 57.3;

QMatrix ma = QMatrix(cos(agile), -sin(agile), sin(agile), cos(agile), 0, 0);

ma = ma \* xy;

return QPointF(ma.m11(), ma.m21());

}

QPointF Amorph::resize(QPointF p, int kx, int ky){

QMatrix xy = QMatrix(p.x(), 0, p.y(), 0, 1, 0);

QMatrix rm = QMatrix(kx, 0, 0, ky, 0, 0);

rm = rm \* xy;

return QPointF(rm.m11(), rm.m21());

}

QPointF Amorph::moveTo(QPointF p, int dx, int dy){

// QMatrix xy = QMatrix(p.x(), 0, p.y(), 0, 1, 0);

// QMatrix m = QMatrix(1, 0, 0, 1, dx, dy);

// m = m \* xy;

// return QPointF(m.m11(), m.m21());

return QPointF(p.x() + dx, p.y() + dy);

}

bird.h

#ifndef BIRD\_H

#define BIRD\_H

#include <QPainter>

#include <QPointF>

#include "amorph.h"

class Bird

{

QPainter \*painter;

int size;

int agile;

Amorph \*amorph;

public:

Bird(QPainter \*p, Amorph \*am, int size, int a);

void draw(int r);

};

#endif // BIRD\_H

bird.cpp

#include "bird.h"

#include <QVector>

#include <QPointF>

Bird::Bird(QPainter \*p, Amorph \*am, int size, int a){

painter = p;

this->size = size;

agile = a;

amorph = am;

}

void Bird::draw(int r){

painter->setBrush(Qt::blue);

painter->setPen(Qt::darkBlue);

int pad = r \* 0.0666;

QPointF p8(0, r + pad);

QPointF p7 = amorph->moveTo(p8, -3\*size, 2\*size);

QPointF p6 = amorph->moveTo(p7, size, size);

QPointF p9 = amorph->moveTo(p8, 3\*size, size);

QPointF p10 = amorph->moveTo(p9, size, 2\*size);

QPointF p2 = amorph->moveTo(p6, pad, pad);

QPointF p1 = amorph->moveTo(p2, 2\*size, 0);

QPointF p0 = amorph->moveTo(p1, 0, 5\*size);

QPointF p3 = amorph->moveTo(p2, 0, pad);

QPointF p5 = amorph->moveTo(p3, size - pad, 1.67 \* size + pad);

QPointF p4 = amorph->moveTo(p5, -size, 3\*size);

QPointF p14 = amorph->moveTo(p6, -pad, 0.5\*pad);

QPointF p15 = amorph->moveTo(p14, -2\*size, 0);

QPointF p16 = amorph->moveTo(p15, -size, -size);

QPointF p17 = amorph->moveTo(p14, -size, -size);

QPointF p18 = amorph->moveTo(p17, -pad, -pad);

QPointF p19 = amorph->moveTo(p18, -2\*size, 0);

QPointF p20 = amorph->moveTo(p19, -size, -size);

QPointF p11 = amorph->moveTo(p10, pad, -pad);

QPointF p12 = amorph->moveTo(p11, -size, -2\*size);

QPointF p13 = amorph->moveTo(p12, 3\*size, -2\*size);

QPointF wing1[3] = {p0, p1, p2};

QPointF wing2[3] = {p3, p4, p5};

QPointF mounth[3] = {p18, p19, p20};

QPointF tail[3] = {p11, p12, p13};

QPointF head[4] = {p14, p15, p16, p17};

QPointF body[5] = {p6, p7, p8, p9, p10};

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

wing1[i] = amorph->rotate(wing1[i], agile);

wing2[i] = amorph->rotate(wing2[i], agile);

tail[i] = amorph->rotate(tail[i], agile);

mounth[i] = amorph->rotate(mounth[i], agile);

}

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

head[i] = amorph->rotate(head[i], agile);

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

body[i] = amorph->rotate(body[i], agile);

painter->drawPolygon(wing1, 3);

painter->drawPolygon(wing2, 3);

painter->drawPolygon(body, 5);

painter->drawPolygon(head, 4);

painter->drawPolygon(mounth, 3);

painter->drawPolygon(tail, 3);

}

insideobjects.h

#ifndef INSIDEOBJECTS\_H

#define INSIDEOBJECTS\_H

#include <QPointF>

#include <QPainter>

#include <QPolygonF>

#include <QVector>

class InsideObjects

{

public:

InsideObjects();

QVector<QPointF> points;

QVector<double> sizes;

void addRandomPoint(QPointF \*f1, int s1,

QPointF \*f2, int s2,

QPointF \*f3, int s3,

QPointF \*f4, int s4,

QPointF \*f5, int s5,

int r, int radius);

bool pointInPoly(QPointF p, QPointF \*arr, int s);

void draw(QPainter \*painter, int r, int side);

};

#endif // INSIDEOBJECTS\_H

insideobjects.cpp

#include "insideobjects.h"

InsideObjects::InsideObjects()

{

qsrand(9139); // Установка базового числа для отсчёта рандома в qrand

}

void InsideObjects::addRandomPoint(

QPointF \*f1, int s1, QPointF \*f2, int s2,

QPointF \*f3, int s3, QPointF \*f4, int s4,

QPointF \*f5, int s5, int r, int side){

QPointF \*arr;

int as;

switch (qrand() % 5) {

case 0:

arr = f1; as = s1; break;

case 1:

arr = f2; as = s2; break;

case 2:

arr = f3; as = s3; break;

case 3:

arr = f4; as = s4; break;

default:

arr = f5; as = s5; break;

}

double circle\_size = (double)(qrand() % 5 + 1) / side;

double x=0, y=0;

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

x += arr[i].x();

y += arr[i].y();

}

x = (x + (qrand() % 2\*side - side)) / (as \* r);

y = (y + (qrand() % 2\*side - side)) / (as \* r);

QPointF point(x, y);

sizes.append(circle\_size);

points.append(point);

}

void InsideObjects::draw(QPainter \*painter, int r, int side){

QVector<QPointF>::iterator cur;

int c = 0;

for(cur = points.begin(); cur != points.end(); cur++, c++){

QVector<double>::iterator sizes\_cur = sizes.begin() + c;

painter->setBrush(Qt::red);

QPointF point(r \* cur->x(), r \* cur->y());

painter->drawEllipse(point, \*sizes\_cur\* side, \*sizes\_cur\* side);

}

}

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPainter>

#include "bird.h"

#include <QPointF>

#include "insideobjects.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private:

Ui::MainWindow \*ui;

int agile;

InsideObjects objects;

void *paintEvent*(QPaintEvent \*event);

void *timerEvent*(QTimerEvent \*event);

};

#endif // MAINWINDOW\_H

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "amorph.h"

#include <QPolygonF>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

agile = 0;

objects = InsideObjects();

startTimer(10);

}

MainWindow::~*MainWindow*()

{

delete ui;

}

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

{

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

QColor orangeRed(240, 100, 0); // Оранжево-красный цвет

Amorph amorph = Amorph();

// Координаты центра

QPointF center = QPointF(width() \* 0.5, height() \* 0.5);

int r;

if (width() < height()) r = width() / 4;

else r = height() / 4;

// Main Circle in center

painter.drawEllipse(center, r, r);

painter.setPen(orangeRed);

painter.translate(center);

Bird bird = Bird(&painter, &amorph, r/6, agile);

bird.draw(r);

int side = r/5;

QPointF f0 = QPoint(0, 0);

QPointF f1 = amorph.moveTo(f0, 2\*side, 0);

QPointF f2 = amorph.moveTo(f0, side, 2\*side);

QPointF f10 = amorph.moveTo(f0, side, -2\*side);

QPointF f11 = amorph.moveTo(f10, side, side);

QPointF f12 = amorph.moveTo(f11, 2\*side, 0);

QPointF f13 = amorph.moveTo(f10, 2\*side, 0);

QPointF f3 = amorph.moveTo(f0, -2\*side, -1.5\*side);

QPointF f4 = amorph.moveTo(f3, side, -2\*side);

QPointF f5 = amorph.moveTo(f3, -side, -2\*side);

QPointF f6 = amorph.moveTo(f0, -2\*side, 2\*side);

QPointF f7 = amorph.moveTo(f6, -2\*side, -0.2\*side);

QPointF f8 = amorph.moveTo(f7, 3\*side, 3\*side);

QPointF f9 = amorph.moveTo(f6, side, 0.7\*side);

QPointF f14 = amorph.moveTo(f2, 2\*side, -side);

QPointF f15 = amorph.moveTo(f14, side, 0);

QPointF f16 = amorph.moveTo(f15, 0.3\*side, 0.5\*side);

QPointF f17 = amorph.moveTo(f16, -side, 2\*side);

QPointF f18 = amorph.moveTo(f14, -side, side);

QPointF fig3[3] = {f0, f1, f2};

QPointF fig1[3] = {f3, f4, f5};

QPointF fig2[4] = {f10, f11, f12, f13};

QPointF fig4[4] = {f6, f7, f8, f9};

QPointF fig5[5] = {f14, f15, f16, f17, f18};

painter.drawPolygon(fig1, 3);

painter.drawPolygon(fig2, 4);

painter.drawPolygon(fig3, 3);

painter.drawPolygon(fig4, 4);

painter.drawPolygon(fig5, 5);

if (agile % 100 == 0){

objects.addRandomPoint(fig1, 3, fig2, 4, fig3, 3, fig4, 4, fig5, 5, r, side);

}

objects.draw(&painter, r, side);

}

void MainWindow::*timerEvent*(QTimerEvent \*event){

agile += 1;

repaint();

}

main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}