**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИИ**  
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”

по теме “Аффинные преобразования на плоскости ”

Выполнил: ст. группы ПВ-31  
Ковалев Павел Александрович

Проверил: Осипов О.В.

**Белгород 2020**

**Вариант 7**

**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.

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.

Результат работы программы (снимки экрана).

**Задача**

k-количество оборотов первого вала

Реализовать вращение валов и следующую анимацию: движение по конвейеру красных шаров. Движение вверх вниз пресса с надавливанием на шары. Когда пресс нажимает на шар, то он попадает в воронку, уменьшается в размере и продолжает движение вниз.

**Ход работы**

Используемые формулы в построении фигуры.

**Поворот** точки вокруг начала координат на угол .

**Перенос** точки вдоль вектора .

**Масштабирование** относительно начала координат на величины .

Матрицы:

Матрица поворота,масштабирование, отражение и перехода.

В классе circle используются: масштабирование и перехода.

В классе stamp используются: перехода.

**Приложения:**

Содержимое файла mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPainter>

#include <QTimer>

#include "matrix.h"

#include "vector.h"

#include "circle.h"

#include "stamp.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

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

void **drawStamp**(QPainter \* painter);

void **drawConv**(QPainter \* painter);

void **drawLowerCircles**(QPainter \* painter);

void **drawMainCircles**(QPainter \* painter);

*private*:

Ui::MainWindow \*ui;

QTimer \*timer;

int tickSkip;

int radius;

int angle;

int scaledCircleWidth;

Vector center;

QPolygon conv, ltr, trt;

Stamp stamp;

QVector <Circle \*> circles;

QRect lowerCircles[3], circleStartPos, circleReadyPos;

*private* slots:

void **animate**();

};

#endif *//* *MAINWINDOW\_H*

Содержимое файла mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QtDebug>

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

*delete* ui;

}

void MainWindow::**animate**(){

update();

}

MainWindow::**MainWindow**(QWidget \*parent):QMainWindow(parent), ui(*new* Ui::MainWindow){

ui->setupUi(*this*);

*this*->setMinimumSize(1200, 600);

*this*->setMaximumSize(1200, 600);

angle = 0;

timer = *new* QTimer(*this*);

timer->setInterval(1000/360);

*//timer->setInterval(1000/4);*

connect(timer, SIGNAL(timeout()), *this*, SLOT(animate()));

timer->start();

radius = *this*->height() / 20;

scaledCircleWidth = radius;

center = Vector(*this*->width() / 4, *this*->height() / 2);

tickSkip = 3;

conv.push\_back(QPoint(0,center.y()));

conv.push\_back(QPoint(0,center.y() - (radius \* 2)));

conv.push\_back(QPoint(radius \* 2,center.y() - (radius \* 2)));

conv.push\_back(QPoint(radius \* 2,center.y()));

conv.push\_back(QPoint(*this*->width() / 3,center.y()));

conv.push\_back(QPoint(*this*->width() / 3,center.y() + radius));

conv.push\_back(QPoint(0,center.y() + radius));

ltr.push\_back(QPoint(-50, -50));

ltr.push\_back(QPoint(-50, 50));

ltr.push\_back(QPoint(50, 50));

stamp = Stamp(QPoint(*this*->width() / 3, center.y() + radius), QPoint(*this*->width() / 3 + radius \* 2, 0));

lowerCircles[0] = QRect(*this*->width() / 4 - radius, center.y() + radius, radius, radius);

QPoint tr = QPoint(lowerCircles[0].center().x() - radius / 8, lowerCircles[0].center().y() + radius / 8);

QPoint bl = QPoint(tr.x() - 2 \* radius, tr.y() + 2 \* radius);

QPoint tl = QPoint(bl.x(), tr.y());

QPoint br = QPoint(tr.x(), bl.y());

lowerCircles[1] = QRect(tl, br);

tl = QPoint(tl.x() - 1.5 \* radius, tl.y() + 1.5 \*radius);

br = QPoint(br.x() - 1.5 \*radius, br.y() + 1.5 \*radius);

lowerCircles[2] = QRect(tl, br);

tl = QPoint(0, center.y());

br = QPoint(radius \* 2, center.y() - radius \* 2);

circleStartPos = QRect(tl, br);

tl = QPoint(*this*->width() / 3 , center.y() -2 \* radius);

br = QPoint(tl.x() + 2 \* radius , tl.y() + 2 \* radius);

circleReadyPos = QRect(tl, br);

Circle \* firstCircle = *new* Circle(circleStartPos);

circles.push\_back(firstCircle);

}

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

QPainter painter(*this*);

painter.setBrush(Qt::*black*);

drawStamp(&painter);

drawLowerCircles(&painter);

drawMainCircles(&painter);

drawConv(&painter);

painter.setBrush(Qt::*black*);

painter.end();

}

void MainWindow::**drawConv**(QPainter \* painter){

painter->setPen(Qt::*black*);

painter->drawPolygon(conv);

}

void MainWindow::**drawLowerCircles**(QPainter \* painter){

QBrush oldBrush = painter->brush();

*if* (angle < 360) angle++; *else* angle = 0;

int f = 1;

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

f \*= -1;

painter->setBrush(Qt::*red*);

painter->drawPie(lowerCircles[i], (0 + angle) \* 16 \* f, 90 \* 16);

painter->setBrush(Qt::*blue*);

painter->drawPie(lowerCircles[i], (90 + angle) \* 16 \* f, 90 \* 16);

painter->setBrush(Qt::*red*);

painter->drawPie(lowerCircles[i], (180 + angle) \* 16 \* f, 90 \* 16);

painter->setBrush(Qt::*blue*);

painter->drawPie(lowerCircles[i], (270 + angle) \* 16 \* f, 90 \* 16);

}

painter->setBrush(oldBrush);

}

void MainWindow::**drawMainCircles**(QPainter \*painter){

QBrush oldBrush = painter->brush();

bool newCircleNeeded = *false*;

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

*if* ((circles[i]->getRect().bottomLeft().x() == circleReadyPos.bottomLeft().x()) && !circles[i]->isReady()){

circles[i]->setDirection(0);

circles[i]->setReadyPos(1);

}

circles[i]->move();

*//Определяем,* *нужно* *ли* *уменьшать* *шар*

*if* (circles[i]->isReady() && circles[i]->getWidth() != scaledCircleWidth){

*if* (!tickSkip){

circles[i]->scale();

tickSkip = 10;

}

tickSkip--;

}

painter->setBrush(Qt::*red*);

painter->drawEllipse(circles[i]->getRect());

}

*if* (circles[circles.size() - 1]->getRect().bottomLeft().y() == 335){

Circle \* tmpCircle = *new* Circle(circleStartPos);

circles.push\_back(tmpCircle);

newCircleNeeded = *false*;

}

*//Если* *очередной* *шар* *достиг* *нижней* *границы* *экрана* *-* *удалить*

*if* (circles[circles.size() - 1]->getRect().bottomRight().y() == 600) {

*delete* circles[circles.size() - 1];

circles.pop\_front();

circles.resize(circles.size() - 2);

}

painter->setBrush(oldBrush);

}

void MainWindow::**drawStamp**(QPainter \* painter){

QBrush oldBrush = painter->brush();

painter->setBrush(Qt::*black*);

*//Сдивигаем* *штамп*

stamp.move();

*if* (stamp.getLowerPos() == -240 || stamp.getLowerPos() == 0) stamp.changeDirection();

painter->drawRect(stamp.getRect());

painter->setBrush(oldBrush);

}

Содержимое файла stamp.h

#ifndef STAMP\_H

#define STAMP\_H

#include <QPolygon>

#include <matrix.h>

*class* **Stamp**{

QRect s;

int direction;

int translation;

*public*:

**Stamp**();

**Stamp**(QRect newStamp):s(newStamp), direction(1), translation(0){}

**Stamp**(QPoint tl, QPoint br);

void **move**();

void **changeDirection**();

void **setDirection**(int newDirection);

int **getLowerPos**();

int **getTranslation**();

QRect **getRect**();

};

#endif *//* *STAMP\_H*

Содержимое файла stamp.cpp

#include "stamp.h"

#include <QDebug>

Stamp::**Stamp**(){

direction = -1;

translation = -1;

}

Stamp::**Stamp**(QPoint tl, QPoint br){

s = QRect(tl, br);

direction = -1;

translation = -1;

}

void Stamp::**move**(){

Vector tmp, tmp1;

tmp = Matrix::translationMatrix(0, translation) \* Vector(s.topLeft().x(), s.topLeft().y());

tmp1= Matrix::translationMatrix(0, translation) \* Vector(s.bottomRight().x(), s.bottomRight().y());

s = QRect(tmp.getPoint(), tmp1.getPoint());

}

int Stamp::**getLowerPos**(){

*return* s.bottomLeft().y();

}

QRect Stamp::**getRect**(){

*return* s;

}

void Stamp::**changeDirection**(){

direction \*= -1;

translation \*= -1;

}

void Stamp::**setDirection**(int newDirection){

direction = newDirection;

}

int Stamp::**getTranslation**(){

*return* translation;

}

Содержимое файла circle.h

#ifndef CIRCLE\_H

#define CIRCLE\_H

#include <QRect>

#include <QPainter>

#include <vector.h>

#include <matrix.h>

*class* **Circle**{

QRect rect;

int direction;

int inc;

bool reachedReadyPos;

bool scalingNeeded;

*public*:

**Circle**();

**Circle**(QRect r);

~**Circle**(){}

QRect **getRect**();

*//0* *-* *вдоль* *оси* *х,* *1* *-* *вдоль* *оси* *у;* *inc* *-* *знак* *приращения,* *для* *отражения* *-1:*

void **move**();

void **draw**(QPainter \* painter);

void **scale**();

int **getDirection**();

int **getInc**();

int **getWidth**();

void **setReadyPos**(bool);

void **setDirection**(int newDirection);

void **setInc**(int newInc);

bool **isReady**();

bool **isScalingNeeded**();

};

#endif *//* *CIRCLE\_H*

Содержимое файла circke.cpp

#include "circle.h"

#include "QDebug"

Circle::**Circle**(){

rect = QRect();

direction = 1;

inc = 1;

reachedReadyPos = 0;

}

Circle::**Circle**(QRect r){

rect = r;

direction = 1;

inc = 1;

reachedReadyPos = 0;

}

QRect Circle::**getRect**(){

*return* rect;

}

void Circle::**move**(){

QPoint tl, br;

*if* (direction){

tl = QPoint(rect.topLeft().x() + 1 \* inc, rect.topLeft().y());

br = QPoint(rect.bottomRight().x() + 1 \* inc, rect.bottomRight().y());

rect = QRect(tl, br);

} *else* {

tl = QPoint(rect.topLeft().x(), rect.topLeft().y() + 1 \* inc);

br = QPoint(rect.bottomRight().x(), rect.bottomRight().y() + 1 \* inc);

rect = QRect(tl, br);

}

}

void Circle::**draw**(QPainter \*painter){

QBrush prevBrush = painter->brush();

painter->setBrush(Qt::*red*);

painter->drawEllipse(rect);

painter->setBrush(prevBrush);

}

void Circle::**scale**(){

Vector tl, br;

QPoint center = rect.center();

tl = Matrix::scalingMatrix(0.9, 0.9) \* Vector(rect.topLeft().x(), rect.topLeft().y());

br = Matrix::scalingMatrix(0.9, 0.9) \* Vector(rect.bottomRight().x(), rect.bottomRight().y());

rect = QRect(tl.getPoint(), br.getPoint());

int rx, ry;

rx = center.x() - rect.center().x();

ry = center.y() - rect.center().y();

tl = Matrix::translationMatrix(rx, ry) \* tl;

br = Matrix::translationMatrix(rx, ry) \* br;

rect = QRect(tl.getPoint(), br.getPoint());

}

int Circle::**getDirection**(){

*return* direction;

}

void Circle::**setDirection**(int newDirection){

direction = newDirection;

}

int Circle::**getInc**(){

*return* inc;

}

int Circle::**getWidth**(){

*return* rect.bottomRight().x() - rect.bottomLeft().x();

}

void Circle::**setInc**(int newInc){

inc = newInc;

}

void Circle::**setReadyPos**(bool f){

reachedReadyPos = f;

}

bool Circle::**isReady**(){

*return* reachedReadyPos;

}

bool Circle::**isScalingNeeded**(){

*return* scalingNeeded;

}

**Результат работы программы:**

