МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Компьютерная графика»
Тема: Исследование математических методов представления и преобразования графических объектов на плоскости и в пространстве

Студент гр. 9308	____________________	Степовик В.С. Соболев М.С.
Преподаватель	____________________	Матвеева И.В.

Санкт-Петербург
2022

Оглавление

Цель работы

Исследовать математические методы представления и преобразования графических объектов на плоскости и в пространстве.

Задание

Сформировать отрезок, проведенный из произвольно расположенной точки на плоскости к заданной окружности, определив предварительно координаты точки касания. Необходимо предусмотреть возможность редактирования положения точки и параметры окружности.

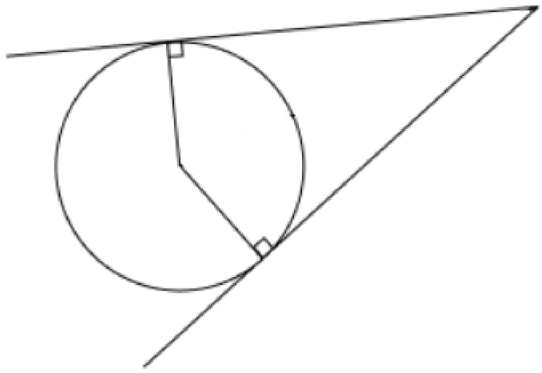
Используемые ресурсы

Для выполнения лабораторной работы использовался язык C++ с использованием фреймворка Qt.

Основные теоретические положения

Данную работу поделим на две теоретические составляющие: отрисовка графических объектов и формулы, задающие положение объектов.

Для изображения окружности и касательных к ней из любой точки на плоскости, нам потребуются, соответственно, изображение окружности, точки и отрезков (касательных и радиусов, перпендикулярным им):



Ввиду того, что объекты строятся в плоской системе координат (а, значит, каждая точка определяется двумя переменными – x,y), задающие объекты уравнения и алгоритмы отрисовки выглядят следующим образом:

Точка:

Точка на плоскости рисуется очень просто: при помощи библиотечной функции drawPoint() фреймворка Qt, с передача в неё необходимых координат как параметров (на скрине так же присутствует смещение)

```
// ---------- sDraw ----------

void sPoint::sdraw(QPainter& qp)
{
    qp.drawPoint(x() + origin.getX(), y() + origin.getY());
}

```

Отрезок:

$$AB = (x_2 - x_1; y_2 - y_1)$$

Длинна задаётся уравнением на плоскости: $L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, где $(x_1 ; y_1), (x_2; y_2)$ – концы отрезка.

Алгоритм отрисовки отрезка заключается в определении положения концов отрезка относительно друг друга и поточечного смещения от одного конца к другому.

Реализация вышеописанного алгоритма на языке C++ с использованием библиотек фреймворка Qt:

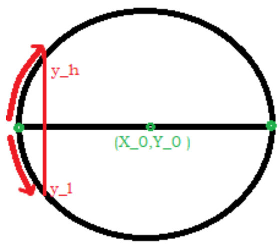
```cpp

void sLine::sdraw(QPainter& qp)
{
    int x1 = p1.x();
    int x2 = p2.x();
    int y1 = p1.y();
    int y2 = p2.y();

    int deltaX = abs(x2 - x1);
    int deltaY = abs(y2 - y1);
    int signX = x1 < x2 ? 1 : -1;
    int signY = y1 < y2 ? 1 : -1;
    int error = deltaX - deltaY;
    qp.drawPoint(x2 + origin.getX(), y2 + origin.getY());

    while (x1 != x2 || y1 != y2)
    {
        qp.drawPoint(x1 + origin.getX(), y1 + origin.getY());
        int error2 = error * 2;

        if (error2 > -deltaY)
        {
            error = error - deltaY;
            x1 = x1 + signX;
        }
        if (error2 < deltaX)
        {
            error = error + deltaX;
            y1 = y1 + signY;
        }
    }
}

```

Проблемы не найдены.

Окружность:

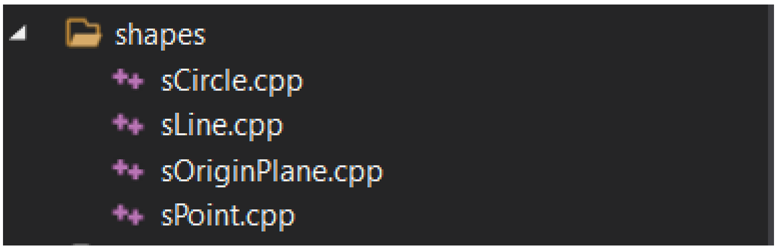
Задаётся уравнением на плоскости $(x - x_0)^2 + (y - y_0)^2 = R^2$, где R – радиус, а $(x_0; y_0)$ – координаты центра окружности.

По сути, алгоритм вывода окружности на экран представляет собой пошаговую отрисовку её верхней и нижней полуокружностей от $x = x_0 - r$ до $x = x_0 + r$, при том на каждом шаге вычисляются y_h и y_l, после чего получаем две точки верхней и нижней полуокружностей соответственно: (x; y_h), (x; y_l):



```cpp
void sCircle::sdraw(QPainter& local_qpainter)
{
    int x_0 = point.x(); // x coordinate
    int y_0 = point.y(); // y coodrinate
    // counting y coordinates for putting points (from start x - r to end x + r)
    for (int x = x_0 - r; x <= x_0 + r; x++)
    {
        double horde = sqrt((double)(r*r - (x - x_0)*(x - x_0))); // horde from begit to circle
        double y_new = y_0 + horde; // adding + horde to draw
        y_new = y_new < 0 ? y_new - 0.5 : y_new + 0.5; // counting nearest integer value
        int y_1 = (int)(y_new); // making an integer value

        y_new =  y_0 - horde; // adding - horde to draw
        y_new = y_new < 0 ? y_new - 0.5 : y_new + 0.5; // counting nearest integer value
        int y_2 = (int)(y_new); // making an integer value

        local_qpainter.drawPoint(x + origin.getX(), y_1 + origin.getY()); // drawing point up
        local_qpainter.drawPoint(x + origin.getX(), y_2 + origin.getY()); // drawing point down
    }
}
```

Представленные выше алгоритмы являются методами соответствующих классов:



Теперь выведем формулу, по которой будут определятся точки касания.

Для этого воспользуемся следующими данными: т.к. точки касания лежат на окружности, а сами касательные перпендикулярны радиусам, то справедлива система уравнений, состоящая из уравнения окружности и скалярного произведения векторов, определяющих радиус и касательную.

$$\begin{cases} (x - x_0)^2 + (y - y_0)^2 = R^2 \\ (x - x_0)(x_1 - x) + (y - y_0)(y_1 - y) = 0 \end{cases},$$

Где r – радиус, $(x_0; y_0)$ – координаты центра окружности, $(x_1; y_1)$ – точка, от которой проводим касательную.

Ниже представленно решение системы уравнений сведённой к квадратному уравнению относительно x:

$$(x-x_0)^2+(y-y_0)^2=r^2 \Rightarrow y=\sqrt{r^2-(x-x_0)^2}+y_0 \qquad (x-x_0)^2=x^2-2x_0x+x_0^2$$

$$(x-x_0)(x_1-x)+(y-y_0)(y_1-y)=0 \Rightarrow -x^2+x_0x+x_1x-x_0x_1-y^2+y_0y+y_1y-y_0y_1=0$$

$$\Rightarrow -x(x-x_0-x_1)-y(y-y_0-y_1)-x_0x_1-y_0y_1=0$$

$$-x(x-x_0-x_1)-x_0x_1-y_0y_1=(y+y_0-y_0)(y-y_0-y_1)$$

$$-x(x-x_0-x_1)-x_0x_1-y_0y_1=\left(\sqrt{r^2-(x-x_0)^2}+y_0\right)\left(\sqrt{r^2-(x-x_0)^2}-y_1\right)$$

$$-x(x-x_0-x_1)-x_0x_1-y_0y_1=r^2-(x-x_0)^2+(y_0-y_1)\left(\sqrt{r^2-(x-x_0)^2}\right)-y_0y_1$$

$$\cancel{-x^2}+x_0x+x_1x-x_0x_1-\cancel{y_0y_1}=r^2-\cancel{x^2}+2x_0x-x_0^2-\cancel{y_0y_1}+(y_0-y_1)\sqrt{r^2-(x-x_0)^2}$$

$$\left(\frac{x_1x-x_0x-x_0x_1-r^2+x_0^2}{y_0-y_1}\right)^2=r^2-(x-x_0)^2$$

$$\frac{\left(x(x_1-x_0)-(r^2+x_0x_1-x_0^2)\right)^2}{(y_0-y_1)^2}=r^2-x^2+2x_0x-x_0^2$$

$$x^2(x_1-x_0)^2-2(x_1-x_0)(r^2+x_0x_1-x_0^2)x+(r^2+x_0x_1-x_0^2)^2=-(y_0-y_1)^2x^2+2(y_0-y_1)^2x_0x-(y_0-y_1)^2(x_0^2-r^2)$$

$$\left((x_1-x_0)^2+(y_0-y_1)^2\right)x^2-2\left((x_1-x_0)(r^2+x_0x_1-x_0^2)+(y_0-y_1)^2x_0\right)x+\left((r^2+x_0x_1-x_0^2)^2+(y_0-y_1)^2(x_0^2-r^2)\right)=0$$

Таким образом, алгоритм нахождения точек касания сводится к подсчёту коэфицентов и решении квадратного уравнения с полседующей подстановкой результата в формулу $y=\sqrt{r^2+\,(x\,-\,x_0)^2}\,+y_0$.

Реализация :

```
sOriginPlane cb(0, 0);

    sPoint xo1 = sPoint(x01, y01, cb);
    sPoint xo2 = sPoint(x02, y02, cb);

    sPoint line11 = sPoint(0, 0, cb);
    sPoint line12 = sPoint(0, 0, cb);
    sPoint line21 = sPoint(0, 0, cb);
    sPoint line22 = sPoint(0, 0, cb);
    sPoint line31 = sPoint(0, 0, cb);
```

```
    sPoint line32 = sPoint(0, 0, cb);
    sPoint line41 = sPoint(0, 0, cb);
    sPoint line42 = sPoint(0, 0, cb);

    float r = r1;
    float x_0 = x01; // center of circle
    float y_0 = y01; // center of circle
    float x_1 = x02;
    float y_1 = y02;
    float point_1[] = { 0,0 };
    float point_2[] = { 0,0 };
    float a = (x_1 - x_0) * (x_1 - x_0) + (y_0 - y_1) * (y_0 - y_1);
    float b = -2 * ((x_1 - x_0) * (r * r + x_0 * x_1 - x_0 * x_0) + x_0 * (y_0 - y_1) * (y_0
- y_1));
    float c = (r * r + x_0 * x_1 - x_0 * x_0) * (r * r + x_0 * x_1 - x_0 * x_0) + (y_0 -
y_1) * (y_0 - y_1) * (x_0 * x_0 - r * r);
    float D = b * b - 4 * a * c;

    if (D > 0)
    {
        // x0 -- circle
        // x1 -- dot
        float y_h, y_l;
        point_1[0] = (sqrt(D) - b)/(2*a);
        y_h = y_0 + sqrt( r*r - (point_1[0] - x_0)*(point_1[0] - x_0) ); // y high -- "+"
square root
        y_l = y_0 - sqrt( r*r - (point_1[0] - x_0)*(point_1[0] - x_0) ); // y low -- "-""
square root

        if(((point_1[0] - x_0)*(x_1 - point_1[0]) + (y_l - y_0)*(y_1 - y_l))==0)
        {
            point_1[1] = y_l;
        }
        else if (((point_1[0] - x_0)*(x_1 - point_1[0]) + (y_h - y_0)*(y_1 - y_h))==0)
        {
            point_1[1] = y_h;
        }
        else // abs (number module) is necessary
        {
            if (abs(((point_1[0] - x_0)*(x_1 - point_1[0]) + (y_l - y_0)*(y_1 - y_l)))
                >= abs(((point_1[0] - x_0)*(x_1 - point_1[0]) + (y_h - y_0)*(y_1 - y_h))))
// abs -- absolute
            {
                point_1[1] = y_h; // if y_l >= y_h => setting y_h (measurement error is
less)
            }
            else // final else
            {
                point_1[1] = y_l;
            }
        }

        point_2[0] = (-sqrt(D) - b)/(2*a);
        y_h = y_0 + sqrt( r*r - (point_2[0] - x_0)*(point_2[0] - x_0) );
        y_l = y_0 - sqrt( r*r - (point_2[0] - x_0)*(point_2[0] - x_0) );
        if(((point_2[0] - x_0)*(x_1 - point_2[0]) + (y_l - y_0)*(y_1 - y_l))==0)
        {
            point_2[1] = y_l;
        }
        else if (((point_2[0] - x_0)*(x_1 - point_2[0]) + (y_h - y_0)*(y_1 - y_h))==0)
        {
            point_2[1] = y_h;
        }
        else // abs (number module) is necessary
```

```
        {
            if (abs(((point_2[0] - x_0)*(x_1 - point_2[0]) + (y_l - y_0)*(y_1 - y_l)))
                >= abs(((point_2[0] - x_0)*(x_1 - point_2[0]) + (y_h - y_0)*(y_1 - y_h))))
            {
                point_2[1] = y_h; // if y_l >= y_h => setting y_h (measurement error is
less)
            }
            else // final else
            {
                point_2[1] = y_l;
            }
        }
    }
    else if (D == 0)
    {
        point_1[0] = -b / (2 * a);
        point_1[1] = y_0 + sqrt(r * r - (point_1[0] - x_0) * (point_1[0] - x_0));
        point_2[0] = point_1[0];
        point_2[1] = y_0 - sqrt(r * r - (point_1[0] - x_0) * (point_1[0] - x_0));
    }

    // setting values for sending to other function

    float res[4] = { point_1[0],point_1[1],point_2[0],point_2[1] };
    line21.setX(point_1[0]);
    line21.setY(point_1[1]);
    line22.setX(x02);
    line22.setY(y02);
    line31.setX(point_2[0]);
    line31.setY(point_2[1]);
    line32.setX(x02);
    line32.setY(y02);
```

Пример работы программы

Пример работы программы представлен на рисунках ниже:

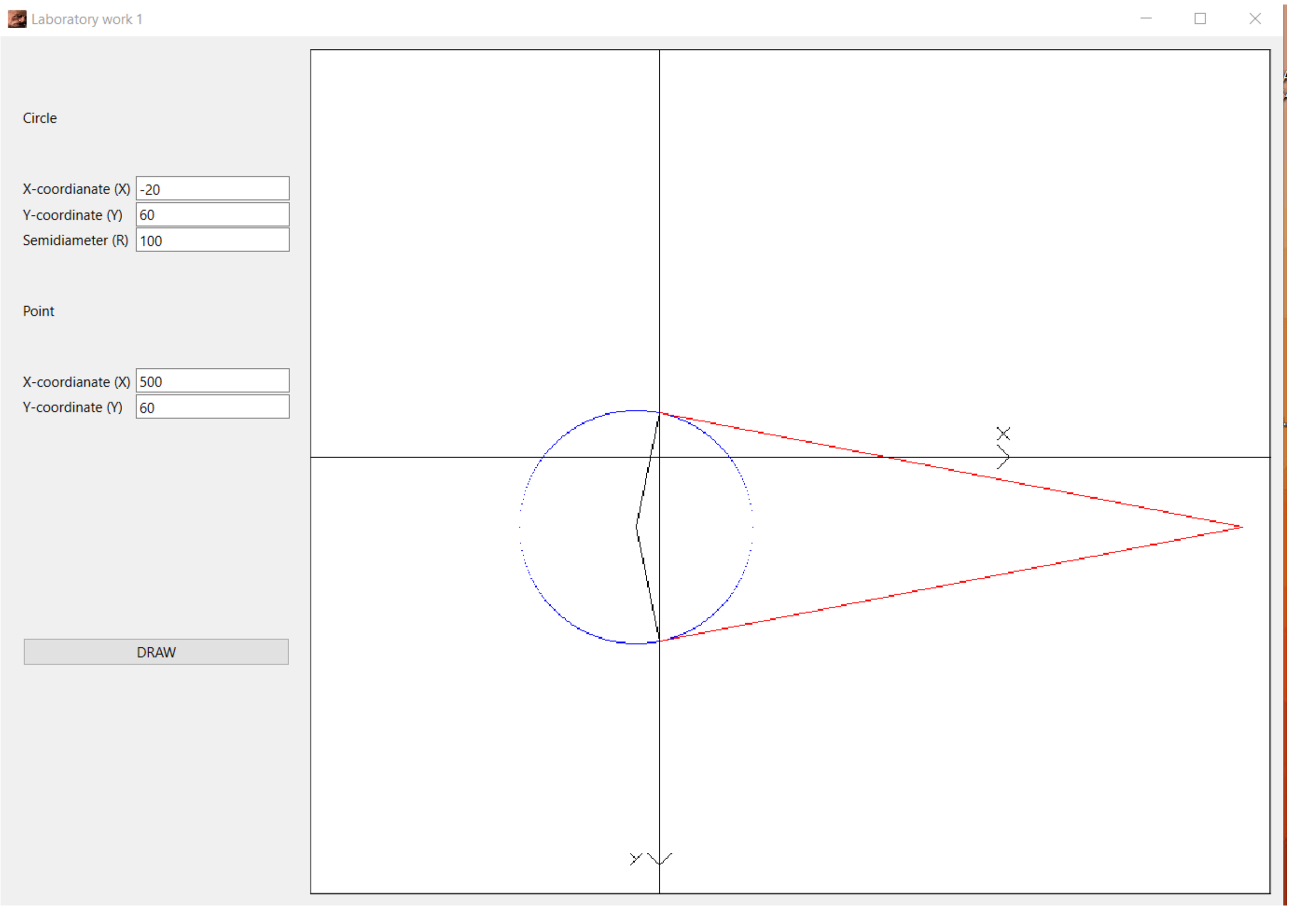


Рисунок 1. Начальное окно

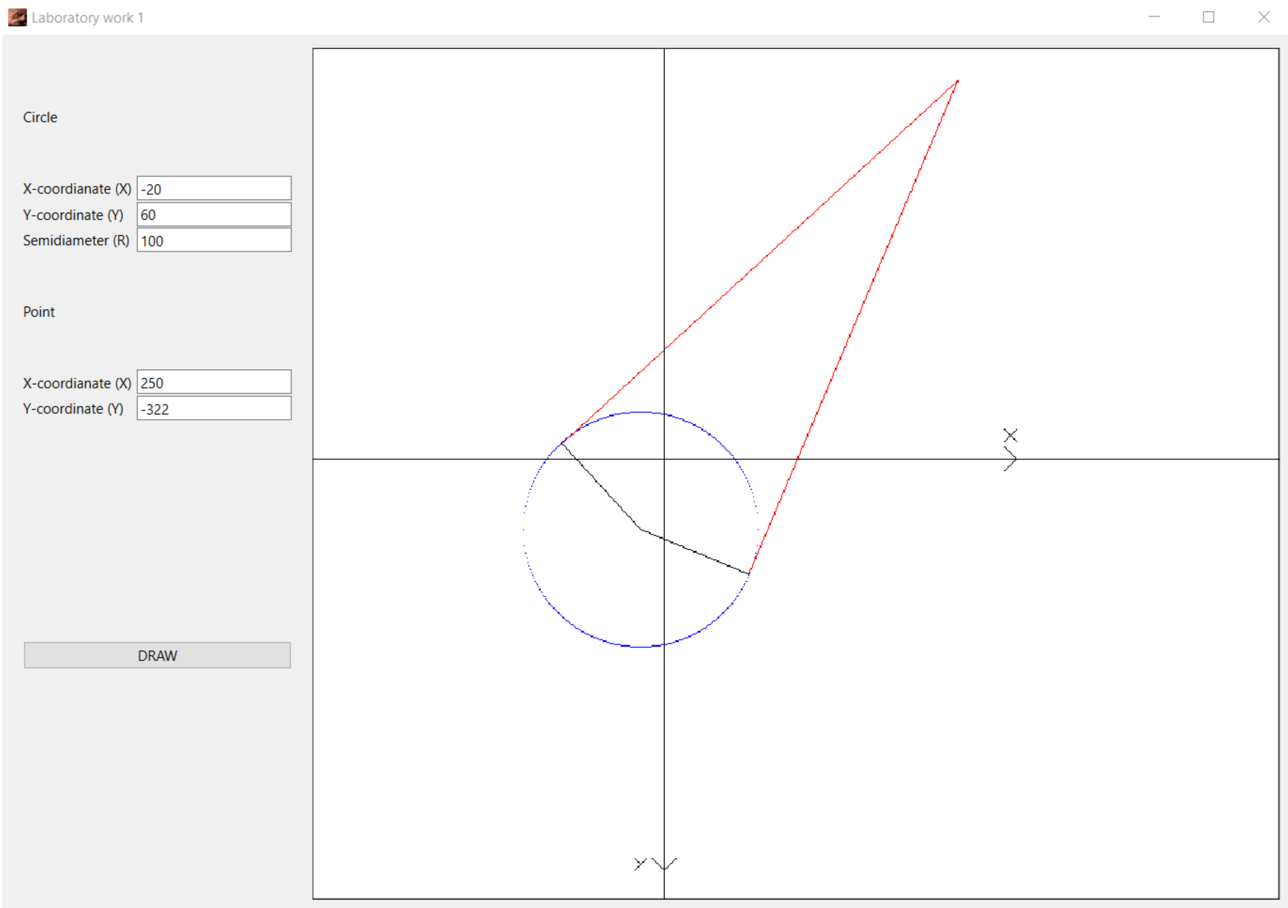


Рисунок 2. Результат изменения координат точки

Вывод

При выполнении лабораторной работы изучены базовые преобразования графических объектов на плоскости. В частности, реализован механизм отрисовки касательной из любой точки плоскости к окружности различного диаметра.