Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №1
по курсу «Компьютерная графика»

Студент:	Марков А.Н.
Группа:	М80-308Б-18
Преподаватель:	Филиппов Г.С.
Оценка:	
Дата:	

Москва
2020

1. Постановка задачи.

Написать и отладить программу, строящую изображение заданной замечательной кривой.

Вариант №5: x^(2/3) +y^(2/3) = a^(2/3), где

x, y — декартовы координаты, a — константа, значение которой выбирается пользователем (вводится в окне программы). a > 0.

Обеспечить автоматическое масштабирование и центрирование кривой при изменении размеров окна.

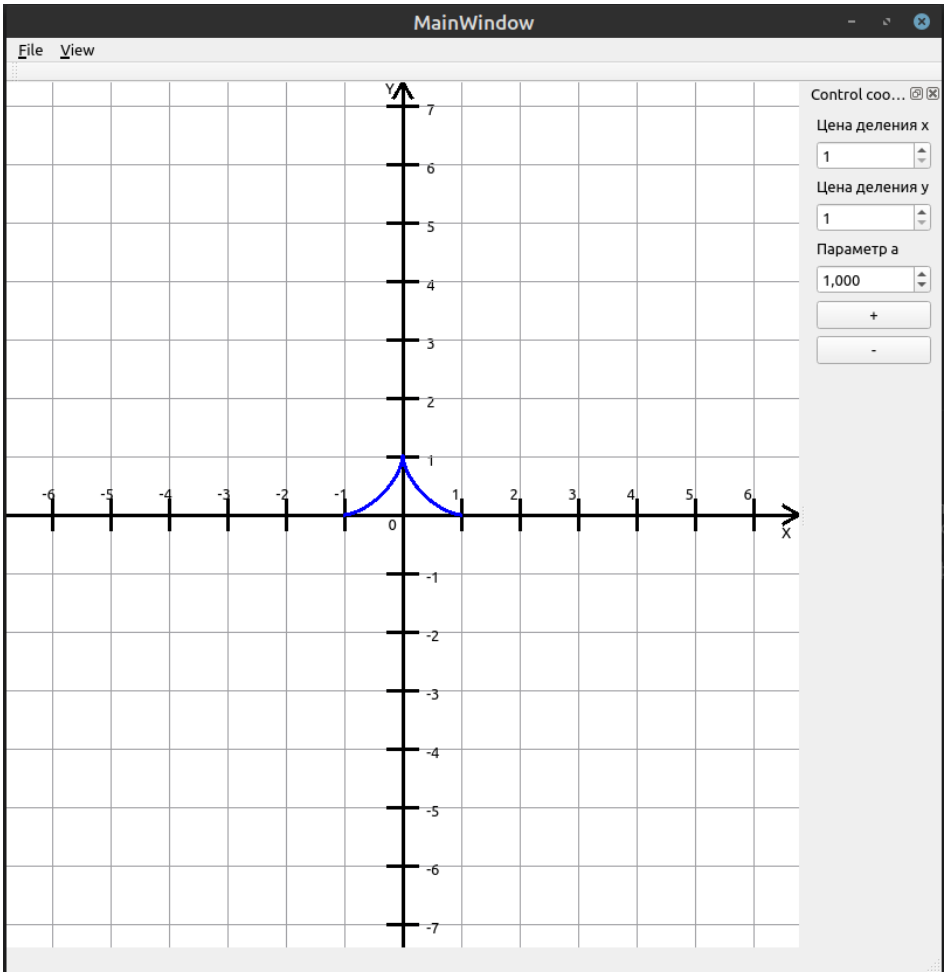
2. Решение задачи.

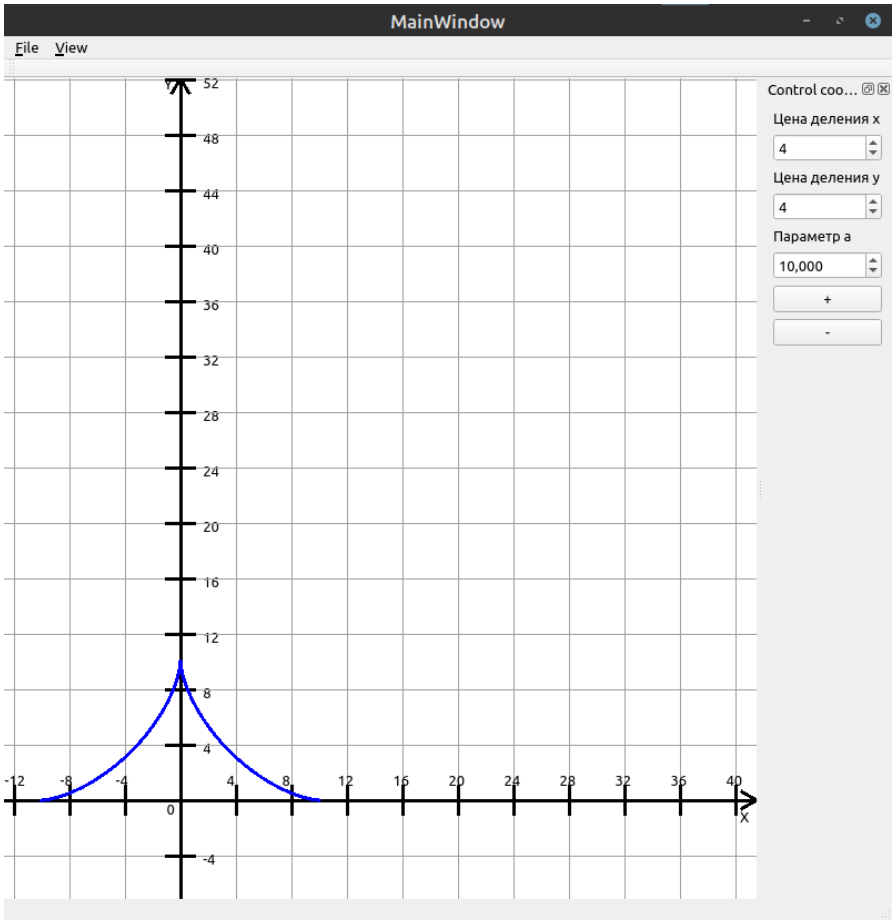
Для решения задачи я решил использовать C++ и фреймворк Qt, в котором использовал библиотеку QPainter для отрисовки точек и линий.

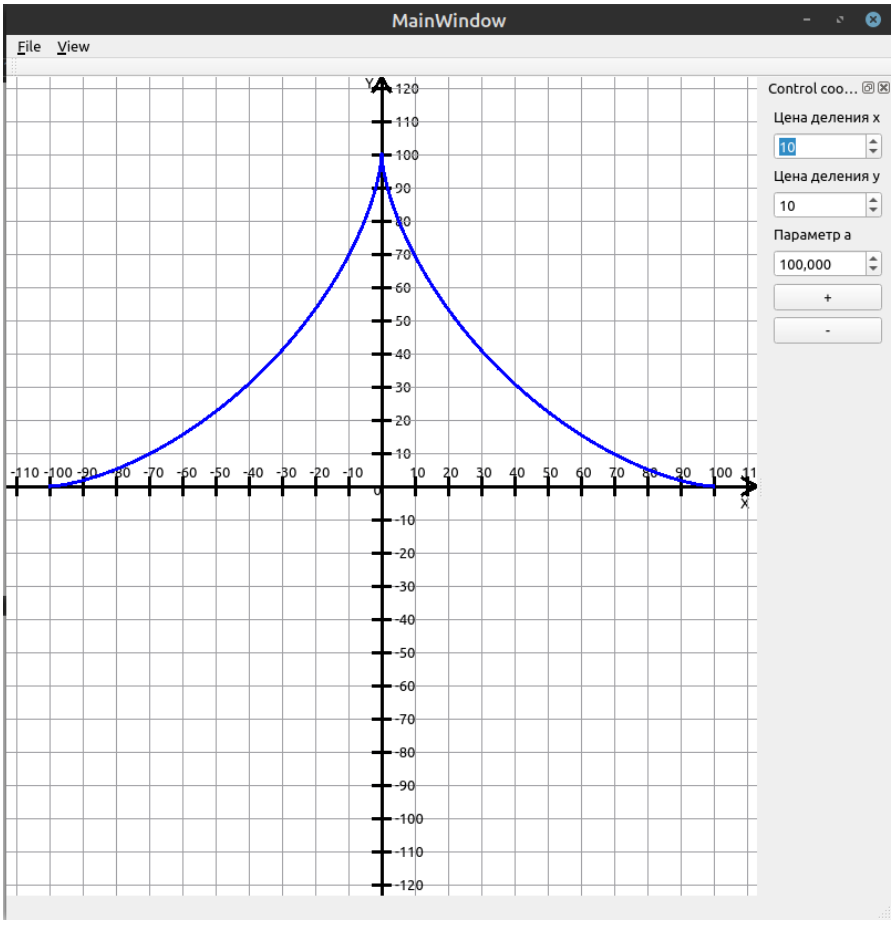
Я создал координатную плоскость, строю график, предлагаемый по умолчанию, со значением константы 1.0 и добавляю панель для ввода значений параметра, цены деления по x и цены деления по y. Эта панель изначально находится справа, но ее можно перемещать и влево, и вверх, и вниз.

При изменении размера окна вызывается функция resizeEvent(), которая вычисляет коэффициенты изменения ширины и высоты окна, а затем при учете этих коэффициентов график перерисовывается. Таким образом реализуется автоматическое масштабирование.

3. Демонстрация работы программы.




MainWindow
File View
Control coo...
Цена деления x
4
Цена деления y
4
Параметр a
10,000
+
-
Y
X


MainWindow
File View
Control coo...
Цена деления x
10
Цена деления y
10
Параметр a
100,000
+
-
Y
X

4. Листинг программы.

Здесь будут отображены только важные части кода.

```
// Функция, отвечающая за отрисовку системы координат и функции.
void view::paintEvent(QPaintEvent *) {
    if (c_s == nullptr) {
        return;
    }
    if (first) {
        center_x = width() / 2;
        center_y = height() / 2;
        first = false;
    }
    draw_cord_system(this);
    draw_function(this);
}

// Функция, которая вызывается при изменении размера окна
void view::resizeEvent(QResizeEvent *event) {
    if (event->oldSize().width() == -1 || event->oldSize().height() == -1)
        return;
    double coef_x = width() / static_cast<double>(event->oldSize().width());
    double coef_y = height() / static_cast<double>(event->oldSize().height());
    if (step_x * coef_x < 1 || step_y * coef_y < 1) {
        update();
        return;
    }
    step_x *= coef_x;
    step_y *= coef_y;
    center_x *= coef_x;
    center_y *= coef_y;
    update();
}

// Функция, предназначенная для получения координаты курсора, при нажатии
// на кнопку мыши
void view::mousePressEvent(QMouseEvent *event) {
    previousPoint = event->pos();
}


// Функция, предназначенная для перемещения графика внутри окна
void view::mouseMoveEvent(QMouseEvent *event) {
    QPointF newPoint = event->pos();
    double delta_x = newPoint.x() - previousPoint.x();
    double delta_y = newPoint.y() - previousPoint.y();
```

```
    center_x += delta_x;
    center_y += delta_y;

    previousPoint = newPoint;
    update();
}

// Прорисовка системы координат
void draw_cord_system(view *v) {
    const int div_x = v->c_s->division_x();
    const int div_y = v->c_s->division_y();
    const double pi = atan(1) * 4;

    QPainter ptr{v};
    ptr.setPen(QPen(Qt::black, 3));

    QPoint p1{0, static_cast<int>(v->center_y)};
    QPoint p2{v->width(), static_cast<int>(v->center_y)};
    const int branch_len = 15;
    // drawing axis x
    ptr.drawLine(p1, p2);

    QPointF p_branch1{static_cast<double>(v->width()), static_cast<double>(v-
>center_y)};
    QPointF p_branch2{branch_len * cos(-5 * pi / 6) + v->width(),
            branch_len * sin(-5 * pi / 6) + v->center_y};
    ptr.drawLine(p_branch1, p_branch2);
    p_branch2 = {branch_len * cos(5 * pi / 6) + v->width(),
             branch_len * sin(5 * pi / 6) + v->center_y};
    ptr.drawLine(p_branch1, p_branch2);

    // drawing axis y
    p1.setX(static_cast<int>(v->center_x));
    p1.setY(0);
    p2.setX(static_cast<int>(v->center_x));
    p2.setY(v->height());
    ptr.drawLine(p1, p2);

    p_branch1 = {static_cast<double>(v->center_x), 0};
    p_branch2 = {branch_len * cos(pi / 3) + v->center_x,
            branch_len * sin(pi / 3)};
    ptr.drawLine(p_branch1, p_branch2);
    p_branch2 = {branch_len * cos(2 * pi / 3) + v->center_x,
             branch_len * sin(2 * pi / 3)};
    ptr.drawLine(p_branch1, p_branch2);
```

```
    ptr.setPen(QPen(Qt::black, 3));
    ptr.drawText(QPointF(v->center_x - (v->step_x / 4), v->center_y + (v->step_y /
4)),
                 QString::number(0));
    ptr.drawText(v->width() - 15, static_cast<int>(v->center_y + 20), "X");
    ptr.drawText(static_cast<int>(v->center_x - 15), 10, "Y");

    // drawing grid
    p_branch1.setY(v->center_y + v->step_y / 4);
    p_branch2.setY(v->center_y - v->step_y / 4);
    p1.setY(0);
    p2.setY(v->height());
    for (int x = static_cast<int>(v->step_x), num = 0; x + v->center_x < v->width() ||
v->center_x - x > 0;
         x += v->step_x, num += div_x) {
        ptr.setPen(Qt::gray);
        p1.setX(static_cast<int>(x + v->center_x));
        p2.setX(static_cast<int>(x + v->center_x));
        ptr.drawLine(p1, p2);

        ptr.setPen(QPen(Qt::black, 3));
        p_branch1.setX(x + v->center_x);
        p_branch2.setX(x + v->center_x);
        ptr.drawLine(p_branch1, p_branch2);

        ptr.drawText(QPointF(x + v->center_x - v->step_x / 6,
static_cast<int>(p_branch1.y()) - v->step_y / 2),
                     QString::number(num + div_x));

        ptr.setPen(Qt::gray);
        p1.setX(static_cast<int>(v->center_x - x));
        p2.setX(static_cast<int>(v->center_x - x));
        ptr.drawLine(p1, p2);

        ptr.setPen(QPen(Qt::black, 3));
        p_branch1.setX(v->center_x - x);
        p_branch2.setX(v->center_x - x);
        ptr.drawLine(p_branch1, p_branch2);

        ptr.drawText(QPointF(v->center_x - x - v->step_x / 6,
static_cast<int>(p_branch1.y()) - v->step_y / 2),
                     QString::number(-1 * (num + div_x)));
    }

    p_branch1.setX(v->center_x + v->step_x / 4);
    p_branch2.setX(v->center_x - v->step_x / 4);
```

```
    p1.setX(0);
    p2.setX(v->width());
    for (int y = static_cast<int>(v->step_y), num = 0; y + v->center_y < v->height() ||
v->center_y - y > 0;
         y += v->step_y, num += div_y) {
        ptr.setPen(Qt::gray);
        p1.setY(static_cast<int>(y + v->center_y));
        p2.setY(static_cast<int>(y + v->center_y));
        ptr.drawLine(p1, p2);

        ptr.setPen(QPen(Qt::black, 3));
        p_branch1.setY(y + v->center_y);
        p_branch2.setY(y + v->center_y);
        ptr.drawLine(p_branch1, p_branch2);

        ptr.drawText(QPointF(static_cast<int>(p_branch1.x()) + v->step_x / 6, v-
>center_y - y + v->step_y / 6),
                     QString::number(num + div_y));

        ptr.setPen(Qt::gray);
        p1.setY(static_cast<int>(v->center_y - y));
        p2.setY(static_cast<int>(v->center_y - y));
        ptr.drawLine(p1, p2);

        ptr.setPen(QPen(Qt::black, 3));
        p_branch1.setY(v->center_y - y);
        p_branch2.setY(v->center_y - y);
        ptr.drawLine(p_branch1, p_branch2);

        ptr.drawText(QPointF(static_cast<int>(p_branch1.x()) + v->step_x / 6, v-
>center_y + y + v->step_y / 6),
                     QString::number(-1 * (num + div_y)));
    }
}

// Прорисовка графика функции
void draw_function(view *v) {
    QPainter ptr{v};
    ptr.setPen(QPen(Qt::blue, 3));
    const double param = pow(v->c_s->parametr(), 2. / 3.);
    const double step = 0.01;
    const double div_x = v->c_s->division_x();
    const double div_y = v->c_s->division_y();
    QPointF p1{static_cast<double>(v->center_x), v->center_y - pow(param, 3. / 2.) *
v->step_y / div_y};
```

```
    QPointF p2{};
    QPointF p3{p1};
    QPointF p4{};
    for (double x = step; param - pow(x, 2. / 3.) >= 0; x += step) {
        p2 = {x * v->step_x / div_x + v->center_x, v->center_y - pow(param - pow(x, 2.
/ 3.), 3. / 2.) * v->step_y / div_y};
        ptr.drawLine(p1, p2);
        p1 = p2;
        p4 = {-x * v->step_x / div_x + v->center_x, v->center_y - pow(param - pow(x,
2. / 3.), 3. / 2.) * v->step_y / div_y};
        ptr.drawLine(p3, p4);
        p3 = p4;
    }

}
```

Реализация боковой панели:

```
coordinate_system_panel::coordinate_system_panel(QWidget *parent) :
QWidget(parent)
{
    QLabel *value_division_x = new QLabel("Цена деления x");
    div_x = new QSpinBox;
    div_x->setRange(1, 10000);
    div_x->setSingleStep(5);
    div_x->setValue(1);

    QLabel *value_division_y = new QLabel("Цена деления y");
    div_y = new QSpinBox;
    div_y->setRange(1, 10000);
    div_y->setSingleStep(5);
    div_y->setValue(1);

    QLabel *value_param = new QLabel("Параметр a");
    param = new QDoubleSpinBox;
    param->setDecimals(3);
    param->setRange(0.001, 10000);
    param->setSingleStep(1);
    param->setValue(1);

    plus = new QPushButton("+", this);

    minus = new QPushButton("-", this);

    QVBoxLayout *lout = new QVBoxLayout;
    lout->addWidget(value_division_x);
```

```
    lout->addWidget(div_x);
    lout->addWidget(value_division_y);
    lout->addWidget(div_y);
    lout->addWidget(value_param);
    lout->addWidget(param);
    lout->addWidget(plus);
    lout->addWidget(minus);
    lout->addStretch();
    setLayout(lout);

    connect(div_x, SIGNAL(valueChanged(int)),
            this, SIGNAL(div_x_changed(int)));
    connect(div_y, SIGNAL(valueChanged(int)),
            this, SIGNAL(div_y_changed(int)));
    connect(param, SIGNAL(valueChanged(double)),
            this, SIGNAL(param_changed(double)));
    connect(plus, SIGNAL(clicked(bool)),
            this, SIGNAL(plus_scale(bool)));
    connect(minus, SIGNAL(clicked(bool)),
            this, SIGNAL(minus_scale(bool)));
}

void coordinate_system_panel::set_div_x(const int &x) {
    div_x->setValue(x);
}

void coordinate_system_panel::set_div_y(const int &y) {
    div_y->setValue(y);
}

int coordinate_system_panel::division_x() const {
    return div_x->value();
}

int coordinate_system_panel::division_y() const {
    return div_y->value();
}

double coordinate_system_panel::parametr() const {
    return param->value();
}
```

5. Вывод.

В ходе выполнения данной лабораторной работы я познакомился с фреймворком Qt, нарисовал систему координат и график заданной функции. Знания, полученные в ходе выполнения работы, несомненно понадобятся мне, потому что задачи визуализации данных встречаются довольно часто.