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.

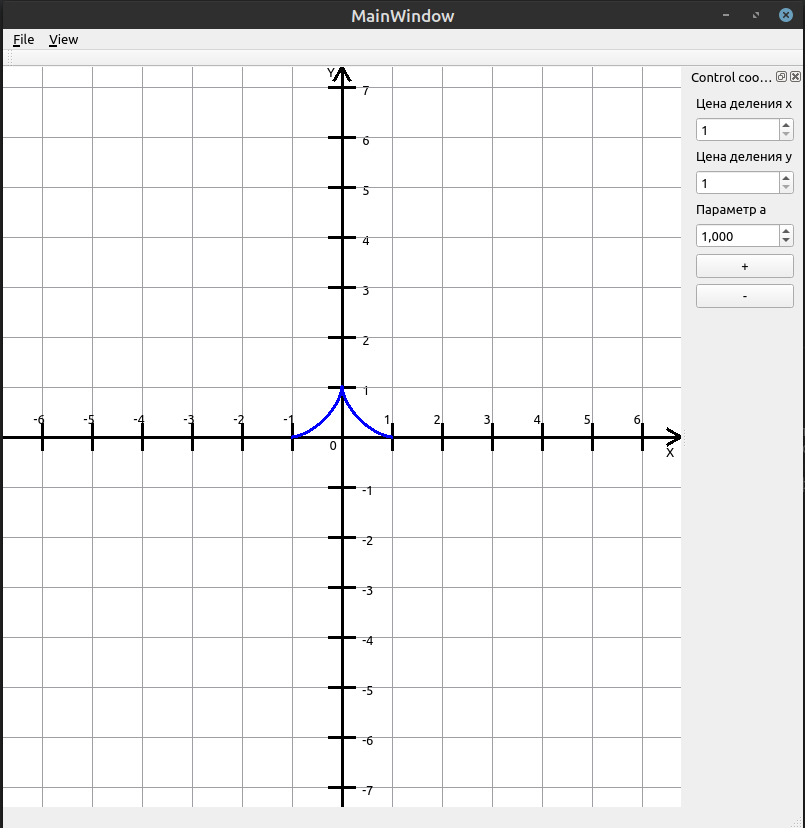
**2. Решение задачи.**

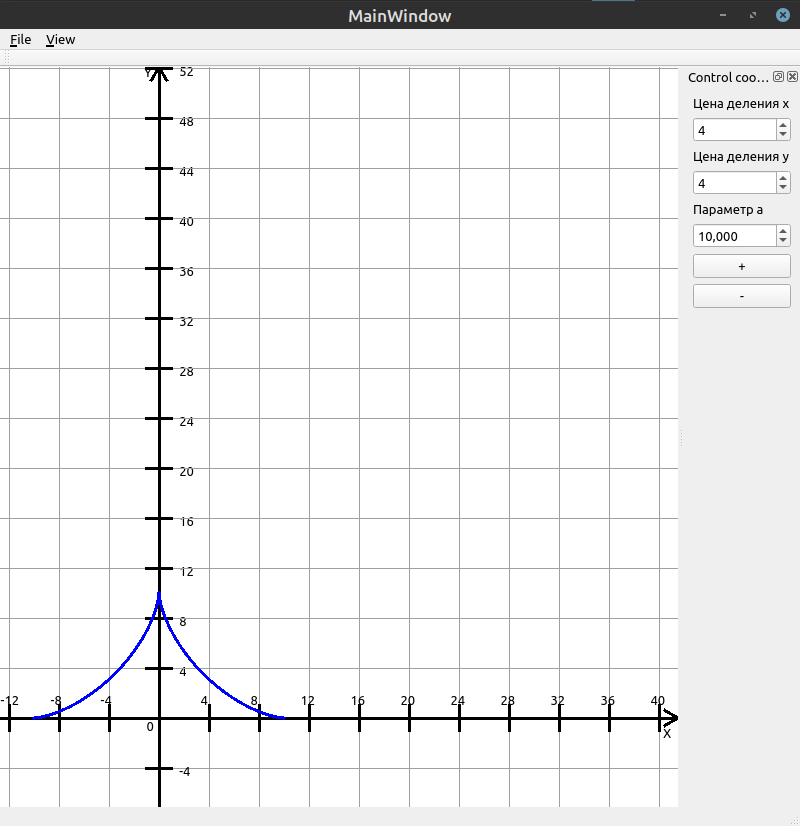
Для решения задачи я решил использовать C++ и фреймворк Qt, в котором использовал библиотеку QPainter для отрисовки точек и линий.

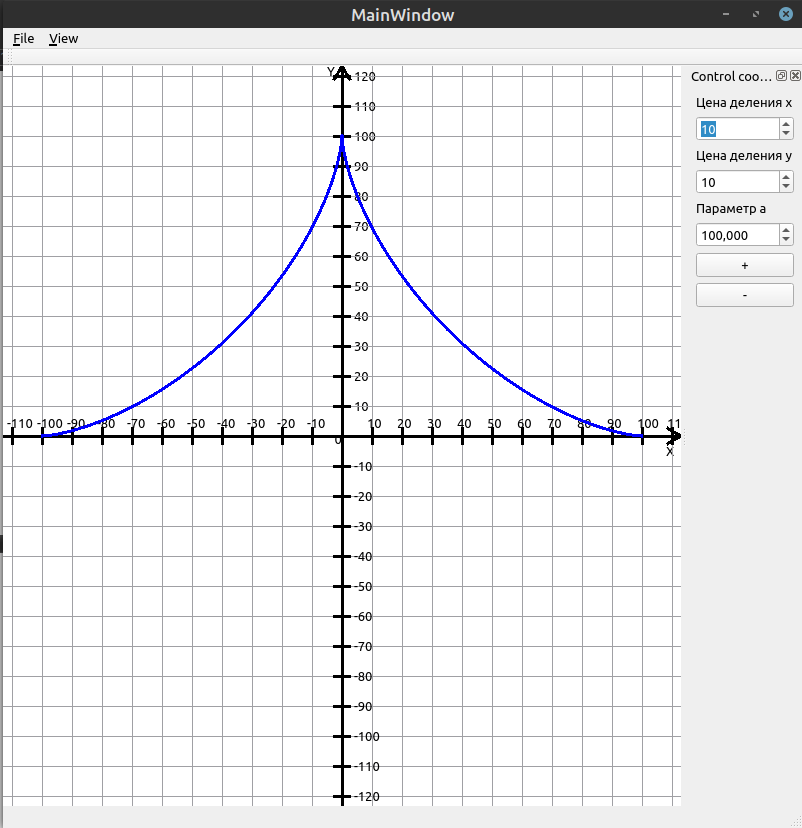
Я создал координатную плоскость, строю график, предлагаемый по умолчанию, со значением константы 1.0 и добавляю панель для ввода значений параметра, цены деления по x и цены деления по y. Эта панель изначально находится справа, но ее можно перемещать и влево, и вверх, и вниз.

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.**



****

****

**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.