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»
Тема: «Построение изображений 2D-кривых»

Студент: Мариничев И. А.
Группа: М8О-308Б-19
Преподаватель: Филиппов Г. С.
Оценка:

Москва
2021

1. Постановка задачи.

Написать и отладить программу, строящую изображение заданной замечательной кривой.

Вариант №8: $y = a * x^{\frac{3}{2}};\ 0 \leq x \leq B$, где x, y — декартовы координаты, a (a>0), B — константы, значение которых выбирается пользователем (вводится в окне программы).

Обеспечить автоматическое масштабирование и центрирование кривой при изменении размеров окна.

2. Описание программы.

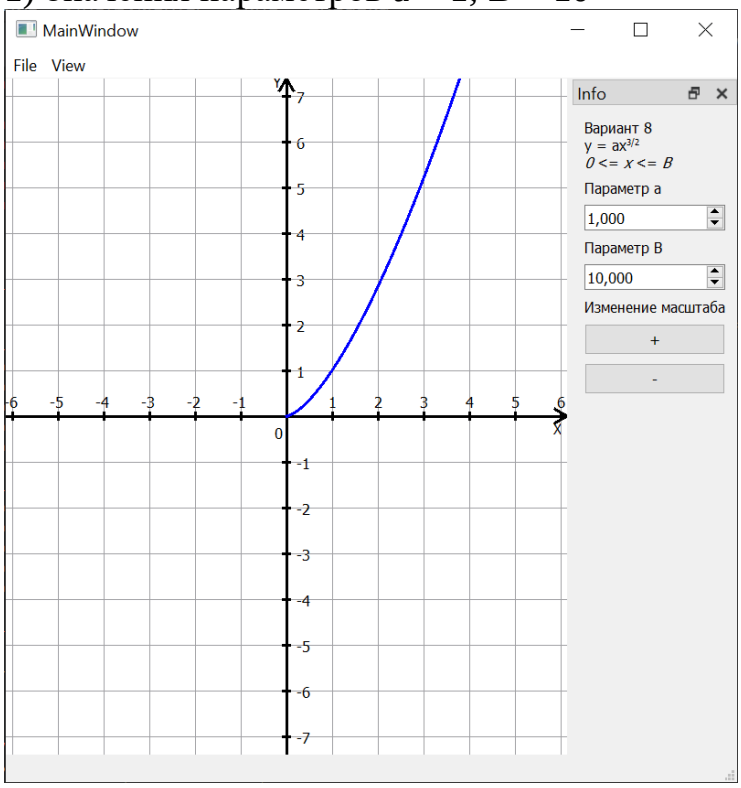
Для решения задачи я решил использовать C++ и фреймворк Qt, в котором использовал библиотеку QPainter для отрисовки точек и линий.

Создаётся координатная плоскость, на которой отрисовывается график, предлагаемый по умолчанию, со значением констант a = 1,000, B = 10,000 и добавляется панель для ввода значений параметра. Эта панель изначально находится справа, но ее можно перемещать влево, вверх, вниз или вообще вынести в отдельное окно.

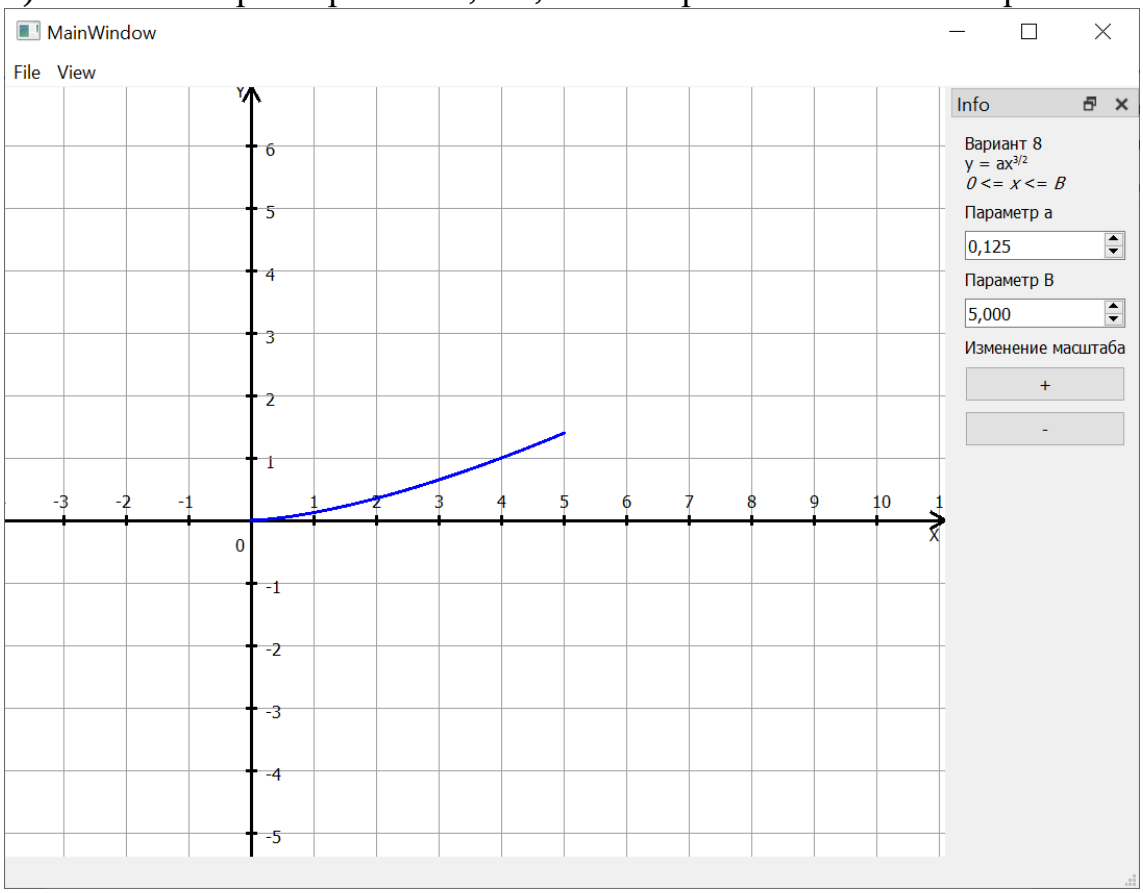
При изменении размера окна вызывается функция resizeEvent(), которая вычисляет коэффициенты изменения ширины и высоты окна, а затем при учете этих коэффициентов график перерисовывается. Таким образом реализуется автоматическое масштабирование. Также реализовано перемещение по координатной плоскости при зажатии ЛКМ, также возможно изменение масштаба при помощи колеса мыши или соответствующих кнопок на панели.

3. Демонстрация работы программы.

1) Значения параметров a = 1; B = 10



2) Значения параметров a = 0,125; B = 5. Применено масштабирование



4. Основной код программы.

1) Отрисовка графика функции

```
void draw_function(widget *w) {
    QPainter ptr{w};
    ptr.setPen(QPen(Qt::blue, 3));
    const double param_a = w->uPanel_pointer->parameter_a();
    const double param_B = w->uPanel_pointer->parameter_B();
    const double step = 0.01;
    const double x_unit_segment = 1.; // value of x coordinate division
    const double y_unit_segment = 1.; // value of y coordinate division
    QPointF p1{w->center_x, w->center_y};
    QPointF p2{};
    for (double x = 0.; x <= param_B; x += step) {
        p2 = {x * (w->step_x / x_unit_segment) + w->center_x, w->center_y -
(param_a * pow(x, 3. / 2.)) * (w->step_y / y_unit_segment)};
        ptr.drawLine(p1, p2);
        p1 = p2;
    }

}
```

2) Отрисовка системы координат, сетки и шкалы со значениями

```
void draw_coord_system(widget *w) {
    const int x_unit_segment = 1.; // value of x coordinate division
    const int y_unit_segment = 1.; // value of y coordinate division
    const double pi = atan(1) * 4;

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

```
    // drawing X axis
    QPoint p1{0, static_cast<int>(w->center_y)};
    QPoint p2{w->width(), static_cast<int>(w->center_y)};
    const int branch_len = 15;

    ptr.drawLine(p1, p2);

    // drawing '>' at the end of X axis
    QPointF  p_branch1{static_cast<double>(w->width()),  static_cast<double>(w-
>center_y)};
    QPointF p_branch2{branch_len * cos(-5 * pi / 6) + w->width(),
                branch_len * sin(-5 * pi / 6) + w->center_y};
    ptr.drawLine(p_branch1, p_branch2);
    p_branch2 = {branch_len * cos(5 * pi / 6) + w->width(),
                 branch_len * sin(5 * pi / 6) + w->center_y};
    ptr.drawLine(p_branch1, p_branch2);

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

    // drawing '>' at the end of y axis
    p_branch1 = {static_cast<double>(w->center_x), 0};
    p_branch2 = {branch_len * cos(pi / 3) + w->center_x,
                branch_len * sin(pi / 3)};
    ptr.drawLine(p_branch1, p_branch2);
    p_branch2 = {branch_len * cos(2 * pi / 3) + w->center_x,
                 branch_len * sin(2 * pi / 3)};
    ptr.drawLine(p_branch1, p_branch2);

    // drawing '0' and 'X' with 'Y'
    ptr.setPen(QPen(Qt::black, 3));
    ptr.drawText(QPointF(w->center_x - (w->step_x / 4), w->center_y + (w->step_y
/ 2)),
                 QString::number(0));
    ptr.drawText(w->width() - 15, static_cast<int>(w->center_y + 20), "X");
    ptr.drawText(static_cast<int>(w->center_x - 15), 10, "Y");

    // drawing grid
    p_branch1.setY(w->center_y + w->step_y / 16);
    p_branch2.setY(w->center_y - w->step_y / 16);
    p1.setY(0);
    p2.setY(w->height());
    for (int x = static_cast<int>(w->step_x), num = 0; x + w->center_x < w-
>width() || w->center_x - x > 0;
         x += w->step_x, num += x_unit_segment) {
        // drawing vertical lines of the grid (right half)
        ptr.setPen(Qt::gray);
        p1.setX(static_cast<int>(x + w->center_x));
        p2.setX(static_cast<int>(x + w->center_x));
        ptr.drawLine(p1, p2);

        // drawing scale on positive X axis
        ptr.setPen(QPen(Qt::black, 3));
        p_branch1.setX(x + w->center_x);
        p_branch2.setX(x + w->center_x);
        ptr.drawLine(p_branch1, p_branch2);

        // drawing scale numbers on positive X axis
        ptr.drawText(QPointF(x    +    w->center_x    -    w->step_x    /    16,
static_cast<int>(p_branch1.y()) - w->step_y / 4),
```

```
                    QString::number(num + x_unit_segment));

        // drawing vertical lines of the grid (left half)
        ptr.setPen(Qt::gray);
        p1.setX(static_cast<int>(w->center_x - x));
        p2.setX(static_cast<int>(w->center_x - x));
        ptr.drawLine(p1, p2);

        // drawing scale on negative X axis
        ptr.setPen(QPen(Qt::black, 3));
        p_branch1.setX(w->center_x - x);
        p_branch2.setX(w->center_x - x);
        ptr.drawLine(p_branch1, p_branch2);

        // drawing scale numbers on negative X axis
        ptr.drawText(QPointF(w->center_x    -    x    -    w->step_x    /    6,
static_cast<int>(p_branch1.y()) - w->step_y / 4),
                    QString::number(-1 * (num + x_unit_segment)));
    }

    p_branch1.setX(w->center_x + w->step_x / 16);
    p_branch2.setX(w->center_x - w->step_x / 16);
    p1.setX(0);
    p2.setX(w->width());
    for (int y = static_cast<int>(w->step_y), num = 0; y + w->center_y < w-
>height() || w->center_y - y > 0;
         y += w->step_y, num +=y_unit_segment) {
        // drawing horizontal lines of the grid (lower half)
        ptr.setPen(Qt::gray);
        p1.setY(static_cast<int>(y + w->center_y));
        p2.setY(static_cast<int>(y + w->center_y));
        ptr.drawLine(p1, p2);

        // drawing scale on negatie Y axis
        ptr.setPen(QPen(Qt::black, 3));
        p_branch1.setY(y + w->center_y);
        p_branch2.setY(y + w->center_y);
        ptr.drawLine(p_branch1, p_branch2);

        // drawing scale numbers on negative Y axis
        ptr.drawText(QPointF(static_cast<int>(p_branch1.x()) + w->step_x / 6, w-
>center_y - y + w->step_y / 6),
                    QString::number(num + y_unit_segment));

        // drawing horizontal lines of the grid (upper half)
        ptr.setPen(Qt::gray);
        p1.setY(static_cast<int>(w->center_y - y));
        p2.setY(static_cast<int>(w->center_y - y));
        ptr.drawLine(p1, p2);

        // drawing scale on positive Y axis
        ptr.setPen(QPen(Qt::black, 3));
        p_branch1.setY(w->center_y - y);
        p_branch2.setY(w->center_y - y);
        ptr.drawLine(p_branch1, p_branch2);

        // drawing scale numbers on positive Y axis
        ptr.drawText(QPointF(static_cast<int>(p_branch1.x()) + w->step_x / 6, w-
>center_y + y + w->step_y / 6),
                    QString::number(-1 * (num + y_unit_segment)));
    }
}
```

3) Автоматическое масштабирование и центрирование кривой при изменении размеров окна

```
void widget::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());

    center_x *= coef_x;
    center_y *= coef_y;
    update();
}
```

4) Вызов функций отрисовки графика и координат, а также задание координат центра при первом вызове

```
void widget::paintEvent(QPaintEvent *) {
    if (uPanel_pointer == nullptr) {
        return;
    }
    if (is_first) {
        center_x = width() / 2;
        center_y = height() / 2;
        is_first = false;
    }
    draw_coord_system(this);
    draw_function(this);
}
```

5. Выводы.

В ходе данной лабораторной работы я познакомился с фреймворком Qt, при помощи которого смог отрисовать график 2-D кривой. Надеюсь, приобретенные навыки помогут мне в дальнейшем, уже при работе с трехмерными фигурами.