Московский Авиационный Институт
(национальный исследовательский университет)

Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Курсовая работа
по курсу «Компьютерная графика»
на тему: «Алгоритмы рисования сглаженных отрезков и дуг»

V семестр

Студентка Черыгова Е.Е.
Группа 8О-304Б

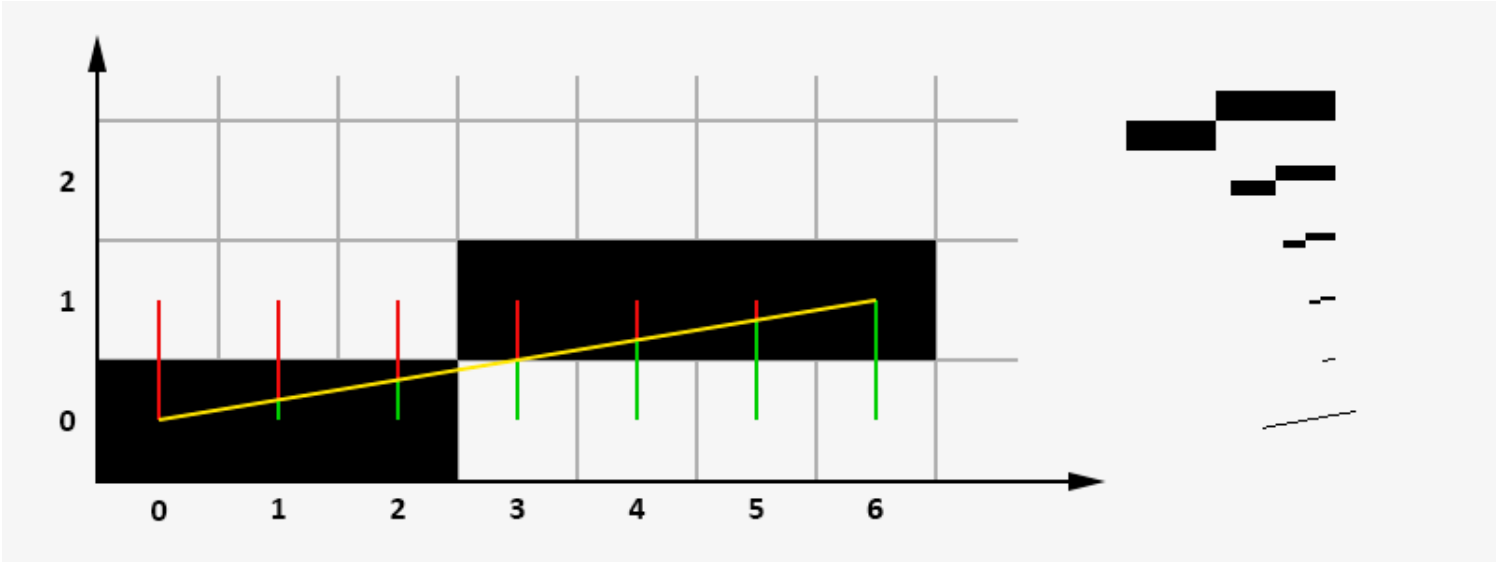
Москва, 2017 г.

Теоретическая часть

Алгоритм Брезенхема

Принцип работы алгоритма Брезенхема очень простой. Берётся отрезок и его начальная координата x. К иксу в цикле прибавляем по единичке в сторону конца отрезка. На каждом шаге вычисляется ошибка — расстояние между реальной координатой y в этом месте и ближайшей ячейкой сетки. Если ошибка не превышает половину высоты ячейки, то она заполняется. Вот и весь алгоритм.

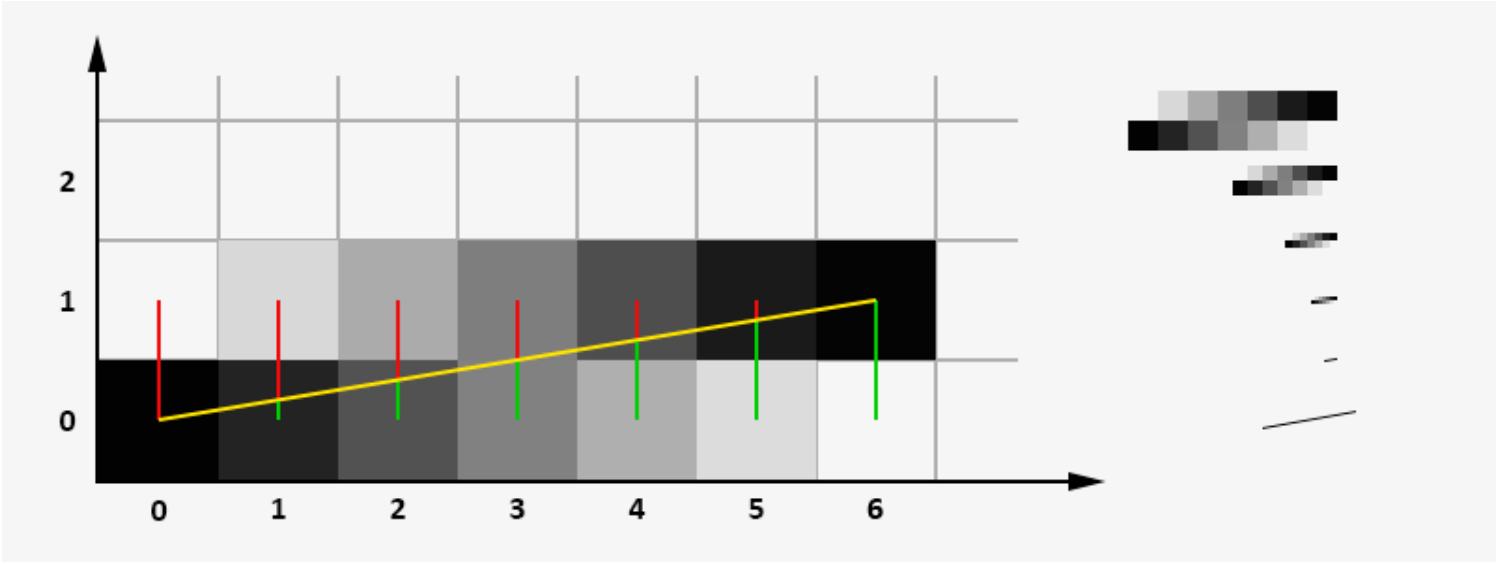
Это была суть алгоритма, на деле всё выглядит следующим образом. Сначала вычисляется угловой коэффициент $(y_1 - y_0)/(x_1 - x_0)$. Значение ошибки в начальной точке отрезка (0,0) принимается равным нулю и первая ячейка заполняется. На следующем шаге к ошибке прибавляется угловой коэффициент и анализируется её значение, если ошибка меньше 0.5, то заполняется ячейка $(x_0 + 1, y_0)$, если больше, то заполняется ячейка $(x_0 + 1, y_0 + 1)$ и из значения ошибки вычитается единица. На картинке ниже жёлтым цветом показана линия до растеризации, зелёным и красным — расстояние до ближайших ячеек. Угловой коэффициент равняется одной шестой, на первом шаге ошибка становится равной угловому коэффициенту, что меньше 0.5, а значит ордината остаётся прежней. К середине линии ошибка пересекает рубеж, из неё вычитается единица, а новый пиксель поднимается выше. И так до конца отрезка.



Алгоритм Ву (У Сяолиня)

Теперь про алгоритм У Сяолиня для рисования сглаженных линий. Он отличается тем, что на каждом шаге ведётся расчёт для двух ближайших к прямой пикселей, и они закрашиваются с разной интенсивностью, в зависимости от удаленности. Точное пересечение середины пикселя даёт 1 интенсивности, если пиксель находится на расстоянии в

0.9 пикселя, то интенсивность будет 0.1. Иными словами, сто процентов интенсивности делится между пикселями, которые ограничивают векторную линию с двух сторон.



На картинке выше красным и зелёным цветом показаны расстояния до двух соседних пикселей.

Алгоритм SSAA

Избыточная выборка сглаживания или суперсемплинг (англ. Supersample anti-aliasing, SSAA) — это наиболее простая и, вместе с тем, ресурсоёмкая техника сглаживания (англ. Anti-aliasing, AA).

В процессе растеризации графического примитива (треугольника, линии или точки) без сглаживания GPU определяет цвет каждого пикселя на основе выборки (англ. sample) из центра этого пикселя. Если центр пикселя лежит внутри примитива, то этот пиксель закрашивается цветом выборки.

Суперсэмплинг увеличивает число дискретных выборок на пиксель (как правило, в 2N раз). В случае с N = 2 выборка производится с удвоенной частотой по обеим осям и сохраняется в экранном буфере (англ. back buffer) При использовании SSAA Nx размер этого буфера увеличивается в N раз. Так, для разрешения 1280x1024 с SSAA 4x необходим экранный буфер такого же размера, как при разрешении 2560x2048 без SSAA. Очевидно, что каждому пикселю на экране в таком случае соответствует 4 пикселя в экранном буфере, поэтому пиксели экранного буфера называют субпикселями (англ. subpixel).

Если очередная выборка оказывается внутри растеризуемого примитива, её результат сохраняется в соответствующий субпиксель. В остальных случаях результат выборки игнорируется. После того, как все нужные выборки сохранены в экранном буфере, итоговый цвет пикселя определяется как усреднённый цвет всех соответствующих ему

субпикселей:

$$result = \frac{sample_0 + sample_1 + ... + sample_{n-1}}{n} = \frac{\sum_{i=0}^{n-1} sample_i}{n}$$

где:

result — итоговый цвет пикселя,

n — количество выборок на пиксель,

$sample_i$ — цвет i-й выборки.

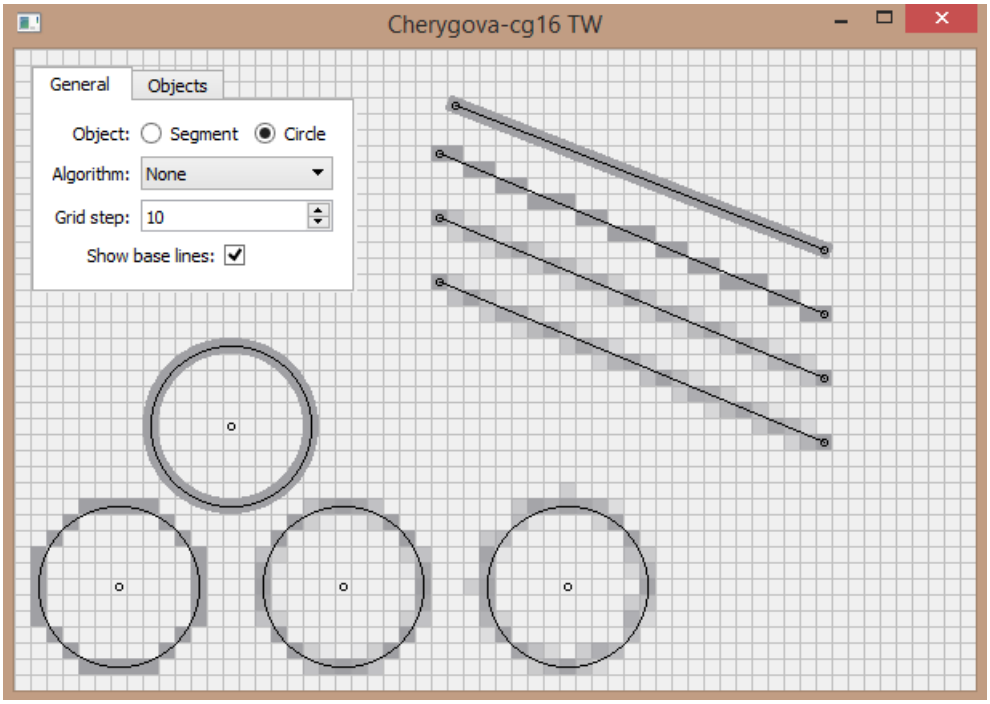
Выборки могут иметь разное физическое расположение внутри пикселя. Можно отметить следующие способы расположения:

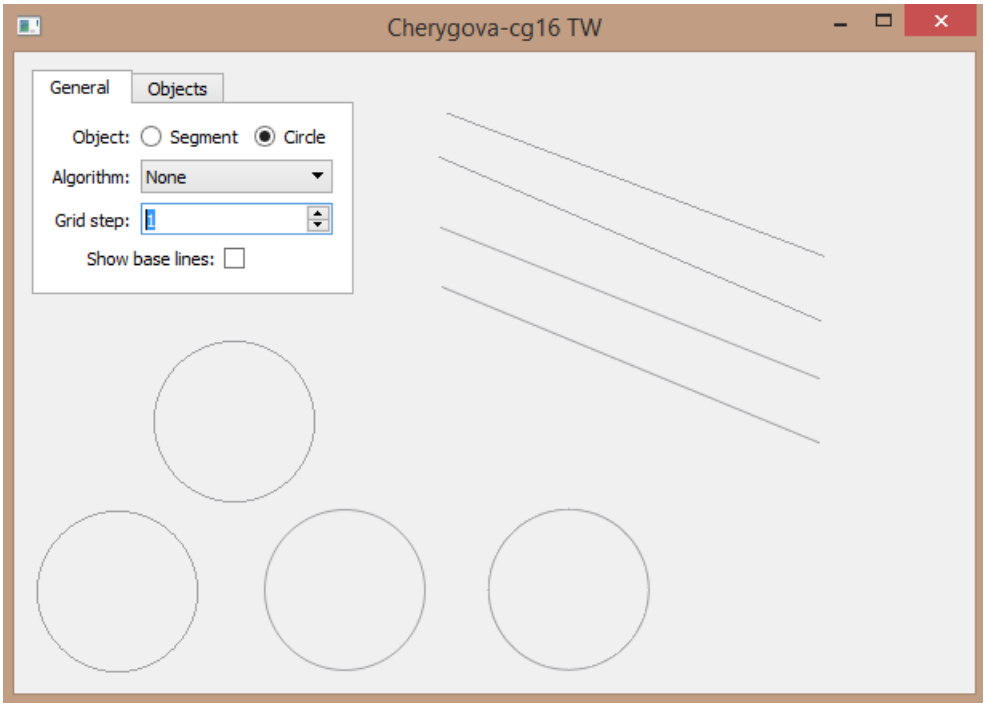
1). Ordered Grid SuperSampling (OGSS) — выборки располагаются на обычной регулярной сетке.

2). Rotated Grid SuperSampling (RGSS) — выборки располагаются на повёрнутой сетке. Этом метод дает особенно хороший результат на близких к горизонтальным и вертикальным линиях.

3). Sparse Grid SuperSampling (SGSS) — выборки располагаются на регулярной сетке, как в OGSS. В отличие от последнего, выборка производится лишь на некоторых узлах сетки. Это первый метод сглаживания, являющийся компромиссом между скоростью работы и качеством изображения.

Скриншоты


Cherygova-cg16 TW
General
Objects
Object: Segment Circle
Algorithm: None
Grid step: 10
Show base lines:


Cherygova-cg16 TW
General
Objects
Object: Segment Circle
Algorithm: None
Grid step: 1
Show base lines:

Практическая часть

segment.hpp

```
#ifndef SEGMENT_HPP
#define SEGMENT_HPP

#include <QPainter>
#include <algorithm>
#include <cmath>
using namespace std;

class Segment
{
private:
    int algorithm, step;
    QPoint sPointScr, ePointScr;
    QPoint sPointGrd, ePointGrd;

public:
    Segment() {}
    Segment(int alg, int step, QPoint start_p, QPoint end_p)
    {
        algorithm = alg;
        this->step = step;
        sPointScr = start_p;
        ePointScr = end_p;
    }
    ~Segment() {}

    void ScrToGrd()
    {
sPointGrd = QPoint(sPointScr.x() / step, sPointScr.y() / step);
ePointGrd = QPoint(ePointScr.x() / step, ePointScr.y() / step);
```

```
}
QPoint GrdToScr(QPoint point)
{
    return QPoint(point.x() * step + step / 2,
                  point.y() * step + step / 2);
}

void None(QPainter& painter)
{
    QPen pen;
    pen.setWidth(step);
    pen.setColor(Qt::gray);
    painter.setPen(pen);
    painter.drawLine(GrdToScr(sPointGrd), GrdToScr(ePointGrd));
}

void drawPixel(QPainter& painter, int x,
                        int y, float alpha, bool wasSwapped)
{
    if (wasSwapped) swap(x, y);
    QColor color = Qt::gray;
    color.setAlphaF(alpha);
    painter.fillRect(x * step, y * step, step, step, color);
}

bool needSwapping(int& x0, int& y0, int& x1, int& y1)
{
    bool wasSwapped = false;
    if (abs(y1 - y0) > abs(x1 - x0))
    {
        swap(x0, y0);
        swap(x1, y1);
        wasSwapped = true;
    }
    if (x0 > x1)
```

```
    {
        swap(x0, x1);
        swap(y0, y1);
    }
    return wasSwapped;
}

void Bresenham(QPainter& painter)
{
    int x0 = sPointGrd.x(), y0 = sPointGrd.y();
    int x1 = ePointGrd.x(), y1 = ePointGrd.y();

    bool wasSwapped = needSwapping(x0, y0, x1, y1);

    int dx = x1 - x0;
    int dy = abs(y1 - y0);
    int error = 0;
    int ystep = (y0 < y1) ? 1 : -1;
    int y = y0;
    for (int x = x0; x <= x1; ++x)
    {
        drawPixel(painter, x, y, 1.0, wasSwapped);
        error += dy;
        if (2 * error >= dx)
        {
            y += ystep;
            error -= dx;
        }
    }
}

void Wu(QPainter& painter)
{
    int x0 = sPointGrd.x(), y0 = sPointGrd.y();
    int x1 = ePointGrd.x(), y1 = ePointGrd.y();
```

```
    bool wasSwapped = needSwapping(x0, y0, x1, y1);

    drawPixel(painter, x0, y0, 1, wasSwapped);
    drawPixel(painter, x1, y1, 1, wasSwapped);

    float gradient = (y1 - y0) / (float)(x1 - x0);
    float y = y0 + gradient;
    for (int x = x0 + 1; x < x1; ++x)
    {
        drawPixel(painter, x, (int)y,
                        1 - (y - (int)y), wasSwapped);
        drawPixel(painter, x, (int)y + 1,
                        y - (int)y, wasSwapped);
        y += gradient;
    }
}

float altitudeFromSubPixCenter(float Ax, float Ay,
                               float Bx, float By,
                               float Cx, float Cy,
                               float AB, bool wasSwapped)
{
    if (wasSwapped)
    {
        swap(Ax, Ay);
        swap(Bx, By);
        swap(Cx, Cy);
    }
    float doubleSquare = (Bx - Ax) * (Cy - Ay) -
                             (Cx - Ax) * (By - Ay);
    if (doubleSquare < 0) doubleSquare *= -1.0F;
    return doubleSquare / AB;
}
```

```
    void SSAA(QPainter& painter, int N)
    {
        int x0 = sPointGrd.x(), y0 = sPointGrd.y();
        int x1 = ePointGrd.x(), y1 = ePointGrd.y();

        bool wasSwapped = needSwapping(x0, y0, x1, y1);

        if (y0 != y1)
        {
            drawPixel(painter, x0, y0, 1, wasSwapped);
            drawPixel(painter, x1, y1, 1, wasSwapped);

            float gradient = (y1 - y0) / (float)(x1 - x0);
            float y = y0 + gradient;
            float AB = sqrt(pow(x1 - x0, 2) + pow(y1 - y0, 2));

            for (int x = x0 + 1; x < x1; ++x)
            {
                float averageColorF = 0.0F;
                float averageColorS = 0.0F;

                for (int i = 0; i < N; ++i)
                {
                    for (int j = 0; j < N; ++j)
                    {
float altitudeF = 1 - altitudeFromSubPixCenter(x0 + 0.5F, y0 + 0.5F,
                                               x1 + 0.5F, y1 + 0.5F,
                                               x + 0.5F / N + i / N,
                                               (int)y + 0.5F / N + j / N,
                                               AB, wasSwapped);
float altitudeS = 1 - altitudeFromSubPixCenter(x0 + 0.5F, y0 + 0.5F,
                                               x1 + 0.5F, y1 + 0.5F,
                                               x + 0.5F / N + i / N,
                                               (int)y + 0.5F / N + j / N + 1,
                                               AB, wasSwapped);
```

```
averageColorF += (altitudeF < 0) ? 0 : altitudeF;
averageColorS += (altitudeS < 0) ? 0 : altitudeS;
                    }
                }

drawPixel(painter, x, (int)y, averageColorF / pow(N, 2), wasSwapped);
drawPixel(painter, x, (int)y + 1, averageColorS / pow(N, 2), wasSwapped);
y += gradient;
            }
        }
        else
            None(painter);
    }

    void Draw(QPainter& painter, int step, bool baseLineVisible)
    {
        this->step = step;
        ScrToGrd();

        switch (algorithm)
        {
            case 0: None(painter); break;
            case 1: Bresenham(painter); break;
            case 2: Wu(painter); break;
            case 3: SSAA(painter, 2); break;
            case 4: SSAA(painter, 4); break;
            case 5: SSAA(painter, 8); break;
            default: break;
        }

        if (baseLineVisible)
        {
            painter.setPen(Qt::black);
            painter.drawEllipse(GrdToScr(sPointGrd), 2, 2);
```

```
            painter.drawEllipse(GrdToScr(ePointGrd), 2, 2);
    painter.drawLine(GrdToScr(sPointGrd), GrdToScr(ePointGrd));
        }
    }
};

#endif // SEGMENT_HPP
```

circle.hpp

```
#ifndef CIRCLE_HPP
#define CIRCLE_HPP

#include <QPainter>
#include <cmath>

class Circle
{
private:
    int algorithm, step;
    QPoint centerPointScr, centerPointGrd;
    int radiusScr, radiusGrd;

public:
    Circle() {}
    Circle(int alg, int step, QPoint center_p, int rad)
    {
        algorithm = alg;
        this->step = step;
        centerPointScr = center_p;
        radiusScr = rad;
    }
    ~Circle() {}

    void ScrToGrd()
    {
        centerPointGrd = QPoint(centerPointScr.x() / step,
                                centerPointScr.y() / step);
        radiusGrd = radiusScr / step;
    }
    QPoint GrdToScr(QPoint point)
    {
```

```
    return QPoint(point.x() * step + step / 2,
                  point.y() * step + step / 2);
}

void None(QPainter& painter)
{
    painter.setPen(QPen(Qt::gray, step));
    painter.drawEllipse(GrdToScr(centerPointGrd),
                    radiusGrd * step, radiusGrd * step);
}

void drawPixel(QPainter& painter, int x, int y, float alpha)
{
    QColor color = Qt::gray;
    color.setAlphaF(alpha);
    painter.fillRect(x * step, y * step, step, step, color);
}

void Bresenham(QPainter& painter)
{
    int x0 = centerPointGrd.x(), y0 = centerPointGrd.y();
    int x = radiusGrd, y = 0;
    int radiusError = 1 - x;

    while (x >= y)
    {
        // I квадрант
        drawPixel(painter, x0 + x, y0 - y, 1); // X
        drawPixel(painter, x0 + y, y0 - x, 1); // Y

        // II квадрант
        drawPixel(painter, x0 - x, y0 - y, 1); // X
        drawPixel(painter, x0 - y, y0 - x, 1); // Y

        // III квадрант
```

```
        drawPixel(painter, x0 - x, y0 + y, 1); // X
        drawPixel(painter, x0 - y, y0 + x, 1); // Y

        // IV квадрант
        drawPixel(painter, x0 + x, y0 + y, 1); // X
        drawPixel(painter, x0 + y, y0 + x, 1); // Y

        y++;
        if (radiusError < 0)
        {
            radiusError += 2 * y + 1;
        }
        else
        {
            x--;
            radiusError += 2 * (y - x + 1);
        }
    }
}

void Wu(QPainter& painter)
{
    int x0 = centerPointGrd.x(), y0 = centerPointGrd.y();

    for (int x = 0; x <= radiusGrd * cos(M_PI / 4); ++x)
    {
        //Вычисление точного значения координаты Y
        float y = sqrt(radiusGrd * radiusGrd - x * x);

        //                      X              Y
        //                            I
        //I квадрант, X
        drawPixel(painter, x0 + (int)y,      y0 - x,
                           1 - (y - (int)y));
        drawPixel(painter, x0 + (int)y + 1,  y0 - x,
```

```
                        y - (int)y);

//I квадрант, Y
drawPixel(painter, x0 + x,          y0 - (int)y - 1,
                        y - (int)y);
drawPixel(painter, x0 + x,          y0 - (int)y,
                        1 - (y - (int)y));

//II квадрант, X
drawPixel(painter, x0 - (int)y - 1,  y0 - x,
                        y - (int)y);
drawPixel(painter, x0 - (int)y ,     y0 - x,
                        1 - (y - (int)y));

//II квадрант, Y
drawPixel(painter, x0 - x,          y0 - (int)y - 1,
                        y - (int)y);
drawPixel(painter, x0 - x,          y0 - (int)y ,
                        1 - (y - (int)y));

//III квадрант, X
drawPixel(painter, x0 - (int)y - 1,  y0 + x,
                        y - (int)y);
drawPixel(painter, x0 - (int)y,      y0 + x,
                        1 - (y - (int)y));

//III квадрант, Y
drawPixel(painter, x0 - x,          y0 + (int)y,
                        1 - (y - (int)y));
drawPixel(painter, x0 - x,          y0 + (int)y + 1,
                        y - (int)y);

//IV квадрант, X
drawPixel(painter, x0 + (int)y,      y0 + x,
                        1 - (y - (int)y));
```

```
        drawPixel(painter, x0 + (int)y + 1 , y0 + x,
                                    y - (int)y);

        //IV квадрант, Y
        drawPixel(painter, x0 + x,           y0 + (int)y,
                                    1 - (y - (int)y));
        drawPixel(painter, x0 + x,           y0 + (int)y + 1,
                                    y - (int)y);
    }
}

float distanceToCircle(float x, float y)
{
    float distance = sqrt(pow(x - centerPointGrd.x(), 2) +
                    pow(y - centerPointGrd.y(), 2)) - radiusGrd;
    return (distance > 0) ? distance : distance * -1.0F;
}

void pixelSSAA(QPainter& painter, int x, int y, int N)
{
    float alpha = 0;
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            float alphaSubpixel = 1 -
distanceToCircle(x + 0.5F / N + i / N, y + 0.5F / N + j / N);
            if (alphaSubpixel > 0) alpha += alphaSubpixel;
        }
    }

    drawPixel(painter, x, y, alpha / pow(N, 2));
}

void SSAA(QPainter& painter, int N)
```

```
{
    int x0 = centerPointGrd.x(), y0 = centerPointGrd.y();

    for (int x = 0; x <= radiusGrd * cos(M_PI / 4); ++x)
    {
        //Вычисление точного значения координаты Y
        float y = sqrt(radiusGrd * radiusGrd - x * x);

        //                        X              Y
        //I квадрант, X
        pixelSSAA(painter, x0 + (int)y,      y0 - x,          N);
        pixelSSAA(painter, x0 + (int)y + 1,  y0 - x,          N);
        //I квадрант, Y
        pixelSSAA(painter, x0 + x,           y0 - (int)y - 1, N);
        pixelSSAA(painter, x0 + x,           y0 - (int)y,     N);

        //II квадрант, X
        pixelSSAA(painter, x0 - (int)y - 1,  y0 - x,          N);
        pixelSSAA(painter, x0 - (int)y ,     y0 - x,          N);
        //II квадрант, Y
        pixelSSAA(painter, x0 - x,           y0 - (int)y - 1, N);
        pixelSSAA(painter, x0 - x,           y0 - (int)y ,    N);

        //III квадрант, X
        pixelSSAA(painter, x0 - (int)y - 1,  y0 + x,          N);
        pixelSSAA(painter, x0 - (int)y,      y0 + x,          N);
        //III квадрант, Y
        pixelSSAA(painter, x0 - x,           y0 + (int)y,     N);
        pixelSSAA(painter, x0 - x,           y0 + (int)y + 1, N);

        //IV квадрант, X
        pixelSSAA(painter, x0 + (int)y,      y0 + x,          N);
        pixelSSAA(painter, x0 + (int)y + 1 , y0 + x,          N);
        //IV квадрант, Y
        pixelSSAA(painter, x0 + x,           y0 + (int)y,     N);
```

```
            pixelSSAA(painter, x0 + x,           y0 + (int)y + 1,  N);
        }
    }

    void Draw(QPainter& painter, int step, bool baseLineVisible)
    {
        this->step = step;
        ScrToGrd();

        switch (algorithm)
        {
            case 0: None(painter); break;
            case 1: Bresenham(painter); break;
            case 2: Wu(painter); break;
            case 3: SSAA(painter, 2); break;
            case 4: SSAA(painter, 4); break;
            case 5: SSAA(painter, 8); break;
            default: break;
        }

        if (baseLineVisible)
        {
            painter.setPen(Qt::black);
            painter.drawEllipse(GrdToScr(centerPointGrd), 2, 2);
            painter.drawEllipse(GrdToScr(centerPointGrd),
                        radiusGrd * step, radiusGrd * step);
        }
    }
};

#endif // CIRCLE_HPP
```

widget.cpp

```
#include "widget.h"
#include "ui_widget.h"

using namespace std;

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);

    step = 10;
    ui->StepSpinBox->setValue(step);
    startPoint = QPoint(-1, -1);
}
Widget::~Widget()
{
    delete ui;
}

void Widget::mousePressEvent(QMouseEvent *e)
{
    if (e->button() == Qt::LeftButton)
    {
        startPoint = e->pos();
    }
    if (e->button() == Qt::RightButton)
    {
if (ui->SegmentButton->isChecked() && Segments.size() > 0) Segments.pop_back();
if (ui->CircleButton->isChecked() && Circles.size() > 0) Circles.pop_back();
repaint();
    }
```

```
}
void Widget::mouseMoveEvent(QMouseEvent *e)
{
    endPoint = e->pos();
    repaint();
}
void Widget::mouseReleaseEvent(QMouseEvent *e)
{
    if (e->button() == Qt::LeftButton && startPoint != e->pos())
    {
        int algorithm = ui->AlgoComboBox->currentIndex();

        if (ui->SegmentButton->isChecked())
            Segments.push_back(Segment(algorithm, step, startPoint, endPoint));

        if (ui->CircleButton->isChecked())
        {
            int radius = abs(endPoint.x() - startPoint.x()) / step * step;
            Circles.push_back(Circle(algorithm, step, startPoint, radius));
        }
    }
    startPoint = QPoint(-1, -1);
}

void Widget::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    painter.setPen(Qt::lightGray);
    int algorithm = ui->AlgoComboBox->currentIndex();
    bool showBaseLines = ui->BaseLinesCheckBox->isChecked();

    // Сетка
    if (step > 1)
    {
for(int i = 0; i <= height(); i += step) painter.drawLine(0, i, width(), i);
```

```
for(int i = 0; i <= width(); i += step) painter.drawLine(i, 0, i, height());
    }

    // Нарисованные линии
    for (uint i = 0; i < Segments.size(); ++i)
            Segments[i].Draw(painter, step, showBaseLines);
    for (uint i = 0; i < Circles.size(); ++i)
            Circles[i].Draw(painter, step, showBaseLines);

    // Рисуемая линия
    if (startPoint.x() >= 0)
    {
        if (ui->SegmentButton->isChecked())
Segment(algorithm, step, startPoint, endPoint).Draw(painter, step, showBaseLines
        if (ui->CircleButton->isChecked())
        {
            int radius = abs(endPoint.x() - startPoint.x()) / step * step;
Circle(algorithm, step, startPoint, radius).Draw(painter, step, showBaseLines);
        }
    }
}

void Widget::on_StepSpinBox_valueChanged(int arg)
{
    step = arg;

    static bool BaseLinesToggled = false;
    if (step == 1)
    {
        BaseLinesToggled = ui->BaseLinesCheckBox->isChecked();
        ui->BaseLinesCheckBox->setChecked(false);
    }
    else if (BaseLinesToggled)
    {
        ui->BaseLinesCheckBox->setChecked(true);
```

```
        BaseLinesToggled = false;
    }

    repaint();
}
void Widget::on_BaseLinesCheckBox_toggled()
{
    repaint();
}
```

Вывод

Получившаяся программа наглядно представляет нам работу некоторых алгоритмов сглаживания. Наглядность заключается в использовании сетки с заданным шагом, каждая клетка которой является условным пикселем экрана. Поначалу было сложно реализовать рисование частей отрезков и окружностей по клеткам (учесть это непосредственно в алгоритмах), но это получилось изящно решить через преобразование экранных координат к координатам сетки и затем обратно перед закрашиванием. Немало времени было уделено написанию удобного и интуитивно понятного интерфейса. Итог курсовой можно назвать несколько разочаровывающим: результат применения суперсэмплинга слабо отличается от результата алгоритма Bv, хотя сложность реализации первого значительно выше. Возможно, это связано с выбранным разбиением (Ordered Grid — упорядоченная сетка), но всё же алгоритм Bv в итоге кажется намного более удачным решением подобного класса задач.