《计算机图形学》实验报告

陈盛恺 181860007

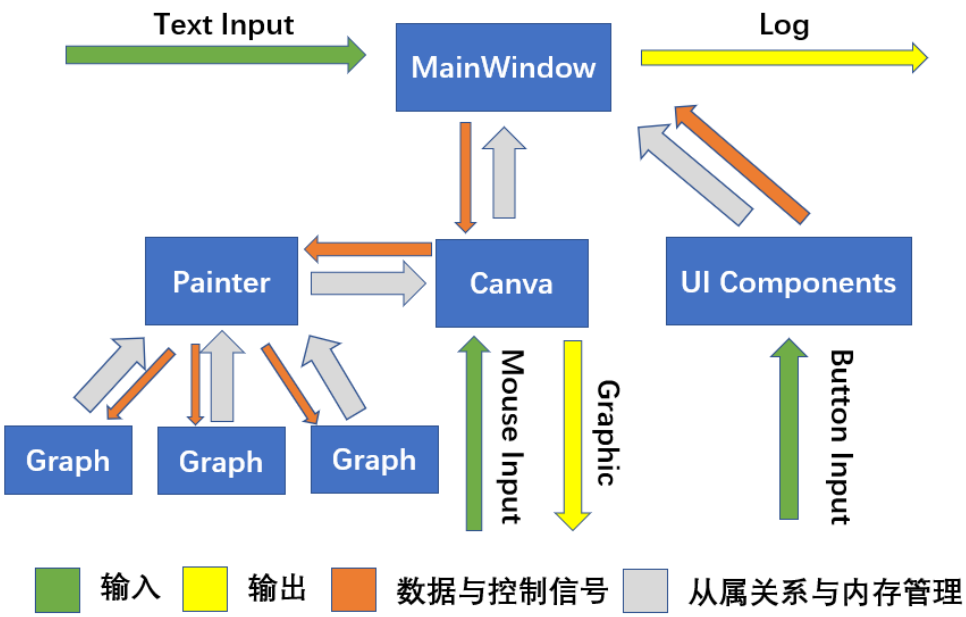
(南京大学 计算机科学与技术系，南京 210093)

摘 要: 本次实验采用的开发语言为 C++(msvc 2014)，开发框架为 Qt(5.12)、Eigen(3.3.7)，开发平台为 Windows10。再此基础上实现了文本与 GUI 两种交互模式下的各种图形学基础功能，包括：DDA 绘制直线，Bresenham 绘制直线、DDA 绘制多边形，Bresenham 绘制多边形、中点圆生成算法绘制椭圆、贝塞尔曲线，3 阶 B 样条曲线、图元的移动、图元的旋转、图元的缩放、Cohen-Sutherland 裁剪直线、Liang-Barsky 裁剪直线。

关键词: 图形学算法、C++、Qt

1 系统框架设计

1.1 框架概览



系统框架

1.2 框架细节

1.2.1 图元基类：Graph

Graph 是一个纯虚类，作为图元的基类。

Graph 的数据如下图所示：

```
protected:
    //变换矩阵
    Matrix3f m_transform;
    //颜色信息
    Byte m_RGB[3];
    //编号
    int m_id;
```

Graph 的函数如下图所示：

```
//在w*h的img数组中绘制当前图元
virtual void Draw(Byte *img,int w,int h) = 0;
//复制当前图元
virtual Graph* Copy() = 0;
//平移变换
Graph* Translate(int x, int y);
//旋转变换
Graph* Rotate(int x, int y,int r);
//缩放变换
Graph* Scale(int x, int y, float sx,float sy);
```

1.2.2 图元子类：Line Ellipse Polygon Curve

Line、Ellipse、Polygon、Curve 都继承自 Graph，各自存储了所需要的数据和对应的绘制算法。（具体见【2 核心算法实现】）

1.2.3 绘制：Painter

Painter 类用于绘制图像。在数据方面，管理 Graph 的内存，将 Graph 指针存在一个以 id 为 key 的 HashMap 中。同时管理 img 图像数组内存和图像的长宽信息。在函数方面，实现了 Graph 子类的生成接口，类似于工厂模式，给定参数之后生成图元并对其内存进行管理，或者根据 id 对某个图元进行操作。

```
Graph* DrawLine(int id, int x1, int y1, int x2, int y2, bool dda) { Delete(id); m_line.insert(id); return m_hash[id] = new Line(id, x1, y1, x2, y2, dda, m_rgb); }
Graph* DrawEllipse(int id, int x, int y, int rx, int ry) { Delete(id); return m_hash[id] = new Ellipse(id, x, y, rx, ry , m_rgb); }
Graph* DrawPolygon(int id, vector<int> const&p, bool dda) { Delete(id); return m_hash[id] = new Polygon(id, p, dda, m_rgb); }
Graph* DrawCurve(int id, vector<int> const&p, bool bezier) { Delete(id); return m_hash[id] = new Curve(id, p, bezier, m_rgb); }
void Translate(int id, int x, int y) { if (m_hash.count(id)) m_hash[id]->Translate(x, y); }
void Rotate(int id, int x, int y, int r) { if (m_hash.count(id)) m_hash[id]->Rotate(x, y, r); }
void Scale(int id, int x, int y, float s) { if (m_hash.count(id)) m_hash[id]->Scale(x, y, s, s); }
void SetClip(int id, int x1, int y1, int x2, int y2, bool C) { if (m_line.count(id)) static_cast<Line*>(m_hash[id])->SetClip(x1, x2, y1, y2, C); };
```

1.2.4 显示与鼠标事件：Canva

Canva 用于捕捉鼠标输入以及将 Painter 中的图像信息输出在图形界面或保存在本地。在数据方面，持有 Painter 的指针，鼠标信息和状态机信息。函数方面，实现了 Qt 提供的鼠标相关的回调函数接口，并在其中维护鼠标的移动点击等信息，并且根据鼠标信息维护一个状态机，实现 GUI 界面的交互。（具体见【系统操作说明书】）

1.2.5 消息处理与控制：MainWindow

MainWindow 作为主控类，数据方面，持有 Canva 和 UI 界面各种交互按键的指针。函数方面，实现了从 UI 界面到数据（按键的信号槽机制）和从文本到数据（字符串分析）的两种交互模式，实现各模块之间数据的传递，并且基于 Canva 状态机和字符串分析结果维护了 Log 信息。

2 核心算法实现

2.1.1 DDA && Bresenham

DDA 是最朴素的直线绘制算法，直接通过不断的对斜率的累计和取整进行像素点的采样，但是由于浮点数的运算性质会带来精度和速度的问题。

Bresenham 基于 DDA 的思想，将斜率改进为△x 和△y 的累计，考虑当△x>0, △y>0, △x>△y 时，每次在 x 轴上增加一个单位像素点时不断累加△y，当累加值>△x 之后就将 y 上升一个像素点，并且将累加值减去△x 再次计算，由此避免了浮点数运算带来的问题。

由于要讨论各种斜率的情况，容易出错，所以在具体实现的时候采用数组对代码进行了优化。例如 p[0]= △x;p[1]= △y; s = abs(△x)<abs(△y); 这样就可以用 p[s]代表较长的增量，p[s^1]作为较短的增量，避免了过多的讨论，利用位运算代替 if 也提升了代码效率和利用率。具体细节详见代码。

```
static void inline DrawLine(Byte *img, int w, int h, bool DDA, Vector2i st, Vector2i ed,Byte m_RGB[]) {
    Vector2i tmp = ed - st;
    int step = abs(tmp(0)) < abs(tmp(1));
    if (tmp(step) < 0) st.swap(ed), tmp *= -1;
    if (DDA) {
        float j = st(!step), delta = 1.0f*tmp(!step) / tmp(step);
        for (tmp = st; tmp(step) <= ed(step); tmp(step)++, j += delta) {
            tmp(!step) = j;
            SetColor(tmp);
        }
    }
    else {
        int dy = abs(tmp(!step) << 1), dydx = dy - abs(tmp(step) << 1), p = dy - abs(tmp(step));
        int x = tmp(!step) < 0 ? -1 : 1;
        for (tmp = st; tmp(step) <= ed(step); tmp(step)++) {
            SetColor(tmp);
            if (p > 0) tmp(!step) += x, p += dydx; else p += dy;
        }
    }
}
```

2.1.2 中点圆算法

中点圆算法的核心思想在于判断像素点到圆心的距离来确定下一个采样点的像素位置。由于椭圆的对称性可以做 1/4 个圆弧并镜像处理，由于椭圆的斜率变化，需要将 1/4 个椭圆分成上下两个部分来处理。具体实现与直线算法类似。但是由于这个椭圆的绘制算法基于坐标轴，所以在变换的时候只能先画完在变换，而不能像直线一样变换关键点，所以可能导致放大后采样率不足。

```
void Ellipse::Draw(Byte *img, int w, int h)
{
    static const int sx[] = { 1,-1,1,-1 };
    static const int sy[] = { 1,1,-1,-1 };
    Vector2i res;
    Vector2i tmp;

    float ry2 = ry * ry, rx2 = rx * rx, rx2ry2 = rx2 * ry2;
    int ry2i = ry * ry, rx2i = rx * rx;
    res << 0, ry;
    for (; res(0)*ry2i < res(1)*rx2i && res(0) <= rx; res(0)++) {
        for (int p = 0; p < 4; p++) {
            tmp(0) = res(0)*sx[p], tmp(1) = res(1)*sy[p];
            tmp = Transform(m_transform, tmp);
            SetColor(tmp);
        }
        res(1) -= (ry2 * (res(0) + 1)*(res(0) + 1) + rx2 * (res(1) - 0.5f)*(res(1) - 0.5f) - rx2ry2 >= 0);
    }
    for (; res(1)>=0 ; res(1)--) {
        for (int p = 0; p < 4; p++) {
            tmp(0) = res(0)*sx[p], tmp(1) = res(1)*sy[p];
            tmp = Transform(m_transform, tmp);
            SetColor(tmp);
        }
        res(0) += (ry2 * (res(0) + 0.5f)*(res(0) + 0.5f) + rx2 * (res(1) - 1)*(res(1) - 1) - rx2ry2 < 0);
    }
}
```

2.1.3 贝塞尔曲线

贝塞尔曲线通过对于控制点的加权和运算得到曲线的采样方程，考虑到由于递推公式可以得出同阶的贝塞尔曲线的系数是相同的，所以可以预处理系数，加快运算效率。同样曲线也有采样不足的问题，同时贝塞尔曲线由于曲线的任意一点都受所有的控制点影响所以局部可控性较差。

```
for (int s = 0; s < Step; s++) {
    double u = 1.0 * s / (Step - 1);
    double nu = 1.0 - u;
    Fac[s][1][0] = u;
    Fac[s][1][1] = nu;
    for (int i = 2; i < N; i++) {
        Fac[s][i][0] = Fac[s][i - 1][0] * u;
        Fac[s][i][i] = Fac[s][i - 1][i - 1] * nu;
        for (int j = 1; j < i; j++) {
            Fac[s][i][j] = Fac[s][i-1][j-1]*nu + Fac[s][i-1][j]*u;
        }
    }
}
```

```
auto p = points;
for (auto &i : p) i = Transform(m_transform, i);
if (bezier) {
    int N = p.size()-1;
    for (int s = 0; s < Step; s++) {
        double x = 0, y = 0;
        for (int i = 0; i <= N; i++) {
            x += Fac[s][N][i] * p[i](0);
            y += Fac[s][N][i] * p[i](1);
        }
        SetColorXY(int(x), int(y));
    }
}
```

2.1.4 B 样条曲线

B 样条曲线是多段贝塞尔曲线的连接，由于递推公式与分段有关所以没有预处理，直接利用 deBoor-Cox 公式计算。效率较低，同样也存在放大之后采样率不足的问题。

```
int M = p.size() + K + 1;
double *U = new double[M];
double st = 1.0 / Step;
for (int i = 0; i < M; i++) U[i] = 1.0*i / (M - 1);
for (double t = U[K-1]; t < U[p.size()]; t+= st) {
    double x = 0, y = 0;
    for (int i = 0; i < p.size(); i++) {
        double B[K] = {};
        for (int j = i; j < i+K ; j++ ) {
            B[j - i] = (t <= U[j + 1] && U[j] <= t);
        }
        for (int p = 1; p < K; p++) {
            for (int j = 0; j < K-p ; j++) {
                double k1 = (t - U[j + i]) < 1e-8 ? 0 : (t - U[j + i]) / (U[j + p + i] - U[j + i]);
                double k2 = (U[i + p + j + 1] - t) < 1e-8 ? 0 : (U[i + p + j + 1] - t) / (U[i + p + j +1] - U[i + j + 1]);
                B[j] = B[j] * k1 + B[j + 1] * k2;
            }
        }
        x += B[0] * p[i](0);
        y += B[0] * p[i](1);
    }
    SetColorXY(int(x), int(y));
}
delete U;
```

2.1.5 图元的平移 旋转 缩放

图元的变换操作只需要在自带的变换矩阵中左乘变换矩阵就可以，但是当旋转和缩放不在原点的时候可以先将图元变换到原点然后再做平移。

```
Graph* Graph::Translate(int x, int y)
{
    Matrix3f tmp;
    tmp <<
        1, 0, x,
        0, 1, y,
        0, 0, 1;
    m_transform = tmp * m_transform;
    return this;
}
```

```
Graph* Graph::Rotate(int x, int y, int r)
{
    Translate(-x, -y);
    float th = r * 3.1415926 / 180;
    Matrix3f tmp;
    tmp <<
        cosf(th), -sinf(th), 0,
        sinf(th), cosf(th), 0,
        0, 0, 1;
    m_transform = tmp * m_transform;
    Translate(x, y);
    return this;
}
```

```
Graph* Graph::Scale(int x, int y, float sx,float sy
{
    Translate(-x, -y);
    Matrix3f tmp;
    tmp <<
        sx, 0, 0,
        0, sy, 0,
        0, 0, 1;
    m_transform = tmp * m_transform;
    Translate(x, y);
    return this;
}
```

2.1.6 Cohen-Sutherland && Liang-Barsky

直线裁剪算法的基本思想就是找到直线与边界之间的交点，Cohen 利用二进制分组的方法把平面分成 9 个区域，来确定直线和裁剪框的相对位置关系，然后从不同方向对直线进行裁剪。而 Barsky 算法则是通过对直线的斜率情况和起点的位置情况判断相对位置关系，然后直接确定裁剪比例进行裁剪。

```
int x1 = st(0), y1 = st(1), x2 = ed(0), y2 = ed(1);
Vector2i delta = ed - st;
double u1 = 0, u2 = 1;
int p[]{ -delta(0),delta(0),-delta(1),delta(1) };
int q[]{ x1 - xmin,xmax - x1,y1 - ymin,ymax - y1 };
for (int i = 0; i < 4; i++) {
    if (p[i] == 0) {
        if (q[i] < 0) return false;
        if (i == 0 || i == 1)
            st(1) = max(ymin, min(y1, y2)), ed(1) = min(ymax, max(y1, y2));
        else
            st(0) = max(xmin, min(x1, x2)), ed(0) = min(xmax, max(x1, x2));
        return true;
    }
    if (p[i] < 0) {
        u1 = max(u1, 1.0* q[i] / p[i]);
    }else{
        u2 = min(u2, 1.0* q[i] / p[i]);
    }
}
if (u1 > u2) return false;
```

```
#define Code(x,y) (((x<xmin)<<0) | ((x>xmax)<<1) | ((y<ymin)<<2) | ((y>ymax)<<3))
bool Line::ClipCohen(Vector2i& st,Vector2i &ed)
{
    int x1 = st(0), y1 = st(1), x2 = ed(0), y2 = ed(1);
    int c1 = Code(x1, y1), c2 = Code(x2, y2);
    while (c1 || c2) {
        if (c1&c2) {
            return false;
        }
        int code = c1 ? c1 : c2, x, y;
        if (code & 1)
            x = xmin, y = y1 + (y2 - y1)*(xmin - x1) / (x2 - x1);
        else if (code & 2)
            x = xmax, y = y1 + (y2 - y1)*(xmax - x1) / (x2 - x1);
        else if (code & 4)
            y = ymin, x = x1 + (x2 - x1)*(ymin - y1) / (y2 - y1);
        else if (code & 8)
            y = ymax, x = x1 + (x2 - x1)*(ymax - y1) / (y2 - y1);
        if (code == c1)
            x1 = x, y1 = y, c1 = Code(x, y);
        else
            x2 = x, y2 = y, c2 = Code(x, y);
    }
    st(0) = x1, st(1) = y1, ed(0) = x2, ed(1) = y2;
    return true;
}
#undef Code
```

3 系统使用说明书

3.1 V0.82更新说明

3.1.1 优化 GUI 操作反馈

在操作时优化去了不必要的鼠标点击，可以通过鼠标滚轮选择图元，具体细节详见提示栏。

3.1.2 增加了控制台接口

在 windows 命令行 输入 ./2019Graphic.exe 【输入文件名】 【输出路径】

可以直接根据文本信息进行图元绘制

3.1.3 增加图元复制功能

选中图元后，点击 copy，然后拖动鼠标左键改变新图元位置 右键完成复制操作。

3.1.4 增加 GUI 界面快捷键

Ctrl+T 测试当前帧绘制时间

Ctrl+R 清除画布

Ctrl+O 打开文件

Ctrl+S 保存图片

Ctrl+C 复制当前图元

3.2 环境信息

开发语言：C++

编译器：msvc 2014

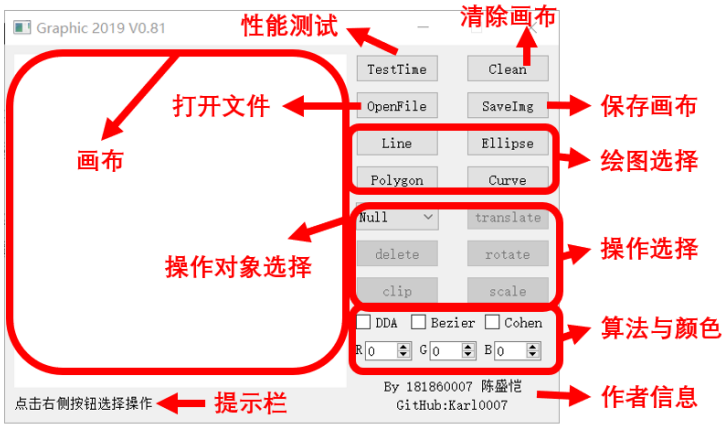
GUI：Qt 5.12

矩阵运算：Eigen 3.3.7

操作系统：Windows10

IDE：Visual Studio 2017

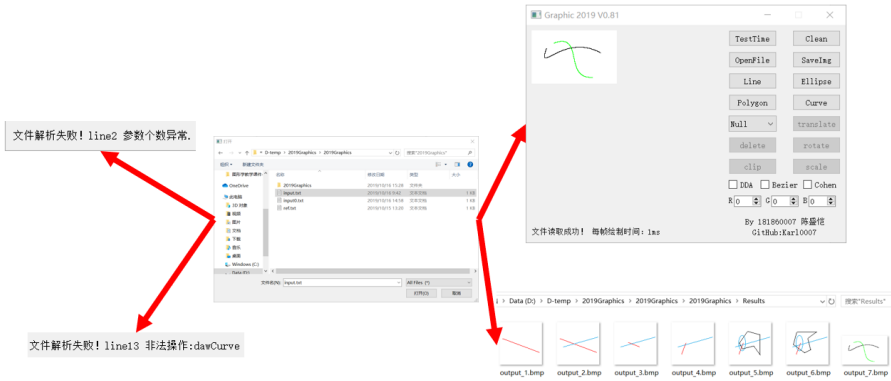
3.3 界面概览



3.4 功能简介

3.4.1 文件读取

点击 OpenFile 然后选取路径可以读取文件内的内容进行作图，结果将保存在 Result 文件夹，文本分析结果和界面刷新时间会在提示栏显示。



3.4.2 文件保存

点击 SaveImg 然后选取路径可以保存当前图片。

3.4.3 性能测试

点击 TestTime 可以在提示栏看到当前画面的刷新所需时间。

3.4.4 重置画布

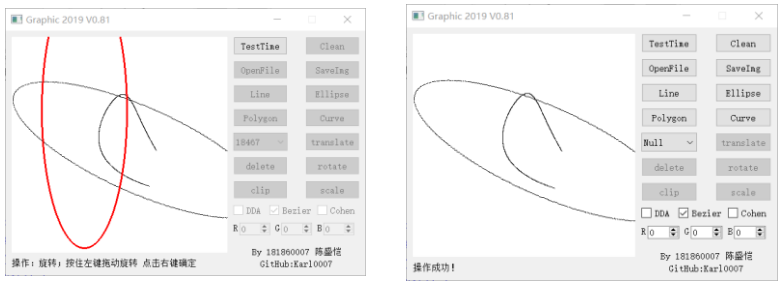
点击 Clean 可以将画布重置到初始状态。

注：由于 UI 界面限制，重置画布不支持更改大小，但是可以通过文件修改和保存，超出画布部分不在 UI 界面显示。

3.4.5 绘图

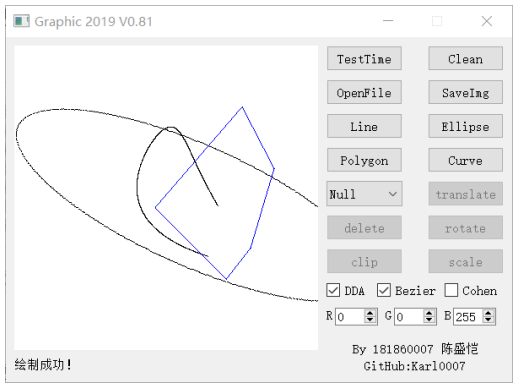
点击绘图选择内的按键，可以用鼠标进行绘图，根据提示栏的提示完成绘制。

3.4.6 操作



选择操作对象（高亮），点击操作种类，然后根据提示栏完成操作。

3.4.7 算法与颜色选择



点击可以勾选算法，改变数值调整颜色。

3.5 下载地址

https://github.com/Karl0007/2019Graphics/archive/master.zip

https://github.com/Karl0007/2019Graphics

References:

[1] https://www.vikingsoftware.com/qwidget-pixel-drawing-2/

[2] http://eigen.tuxfamily.org/dox/AsciiQuickReference.txt

[3] http://blog.csdn.net/augusdi/article/details/12907341

[4] https://www.cnblogs.com/cnblog-wuran/p/9813841.html

[5] https://blog.csdn.net/ZY_cat/article/details/78290047

[6] https://zhuanlan.zhihu.com/p/50626506

[7] https://zhuanlan.zhihu.com/p/50450278

[8] https://blog.csdn.net/u012319493/article/details/54586559

[9] 《Qt5 编程入门》

[10] 《Qt 学习之路 2》

[11] 《C++ Primer 第五版》