《计算机图形学》12 月报告

171860588，史文泰，171860588@smail.nju.edu.cn

2019 年 12 月 11 日

1 综述

计算机图形学课程的实习作业目标是完成简易绘图程序，主要内容分为 4 个模块：

- **核心算法模块**（图元生成算法，图元变换算法，以及整体程序的框架关系）
- **辅助功能模块**（为辅助核心算法以及用户交互模块而实现的功能，包括重置画布等）
- **文件输入输出接口**（包括读取指令文件并解析以及输出绘图结束后的图像）
- **用户交互模块**（包括鼠标，键盘与核心算法的交互，实现便捷绘制）

至本报告完成时，**核心算法模块**，**辅助功能模块**，**用户交互模块和文件输入输出接口模块**全部完成。

本报告只涉及**算法/功能原理概述**，代码实现详见**《系统使用说明书》**。

2 算法介绍

2.1 线段绘制算法（已完成）

2.1.1 DDA 算法

2.1.1.1 原理介绍

DDA 算法即**数值微分算法**，是一种基于直线的微分方程来生成直线的方法。它利用计算两个坐标方向的差分来确定线段显示的屏幕像素位置，是一种线段扫描转换算法。具体算法如下：

考虑直线具有正斜率$m=\frac{\Delta y}{\Delta x}$，且$0\leqslant m\leqslant 1$，则取单位$x$间隔（$\Delta x=1$）采样并计算每个对应的$y$值：

$$y_{k+1}=y_k+m$$

其中，下标k取整数值，并从第一个点开始递增，至最后端点。由于$0\leqslant m\leqslant 1$，为实数，计算出的y值必须取整。

若$m>1$，则取单位y间隔（$\Delta y=1$）采样，并计算连续的x值（增量为$\frac{1}{m}$）。

若$-1\leqslant m<0$，则取单位x间隔（$\Delta x=-1$）采样并计算每个对应的y值。

若$m<-1$，则取单位y间隔（$\Delta y=-1$）采样并计算每个对应的x值。

2.1.1.2 算法理解

DDA 算法直接利用了直线的坐标方程，但是并非使用方程直接计算出每个点的坐标，而是采用增量的思想，固定x轴和y轴方向的增量Δx和Δy中较大者，以此为单位间隔离散取样，逐步确定沿路每个像素的位置。

增量的思想消除了直线方程中的乘法运算，加快了算法执行速度。取增量较大者为单位间隔避免了取样点稀疏问题。

2.1.1.3 算法优化

DDA 算法在每次计算像素位置时涉及浮点运算和取整运算，两者均较为耗时。可将增量m和$\frac{1}{m}$分离成整数和小数部分使所有的计算都简化为整数操作。

DDA 算法的取整运算有累积误差，使长线段所计算的像素位置偏离实际线段。可采取增量叠加后取整的方式避免累积误差。以$0\leqslant m\leqslant 1$为例：

$$y_{k+1}=y_1+m_k$$
$$m_k=m_{k-1}+m$$

其中$m_0=0$，m_k（$k=1,2\ldots,n-1$）为实数。每次对y_k取整，可避免累积误差。

2.1.1.4 性能测试

以 DDA 算法运行以下命令 100000 次：

$$drawLine\ 5\ 7\ 39\ 293\ 171\ DDA$$

共耗时 **2.499**s。

2.1.2 BresenHam 算法

2.1.2.1 原理介绍

BRESENHAM 算法是一种精确而有效的光栅线段生成算法。通过引入整形参量定义来衡量两候选像素与线段路径上实际数学点间在某方向上的相对偏移，并利用对整形参量符号的检测来确定最接近实际路径的像素。

令Δx和Δy分别为端点的水平和垂直偏离量，令$m=\frac{\Delta y}{\Delta x}$，定义算法执行到**第$k$步的决策参数**为$p_k$。以$0\leqslant m\leqslant 1$为例，则有：

$$y = mx + b$$
$$p_k = \Delta x(d_1 - d_2) = 2\Delta y x_k - 2\Delta x y_k + c$$
$$c = 2\Delta y + \Delta x(2b - 1)$$

又由递推关系式可得:

$$p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c$$

两式相减可得:

$$p_{k+1} = p_k + 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

像素的选择决定了$y_{k+1} - y_k$的值，故:

$$p_{k+1} = \begin{cases} p_k + 2\Delta y - 2\Delta x & (p_k > 0，选择(x_k + 1，y_k + 1)) \\ p_k + 2\Delta y & (p_k < 0，选择(x_k + 1，y_k)) \end{cases}$$
$$p_0 = 2\Delta y - \Delta x$$

若$m > 1$，则交换x与y之间的规则，沿y方向为单位不长步进并计算即可。易知:

$$p_{k+1} = \begin{cases} p_k + 2\Delta x - 2\Delta y & (p_k > 0，选择(x_k + 1，y_k + 1)) \\ p_k + 2\Delta x & (p_k < 0，选择(x_k，y_k + 1)) \end{cases}$$
$$p_0 = 2\Delta y - \Delta x$$

若$m < 0$，决策参数p_k与$|m|$的决策参数相同，仅是递进的方向改变，故不再赘述。

2.1.2.2 算法理解

BresenHam 算法利用了一个基本的事实，即实际直线方程与像素网格坐标系相交时，至多有两个候选点，而当限定增量较大者作为单位增量方向使得每次的增量较小候选坐标限定在当前坐标或其增量方向上的下一个坐标，避免了取整运算。

同时，算法巧妙地引入了决策参数p_k，由p_k的符号确定p_{k+1}和点的选择，使得整个运算过程只涉及加法运算，加快了算法的执行速度。

2.1.2.3 算法优化

当直线垂直，即$m = \infty$时，前述算法无法处理，需要单独考虑。此时只需要由y_0至y_n依次递增，即可得到对应的点。

2.1.2.4 性能测试

以 BresenHam 算法运行以下命令 100000 次:

$$drawLine\ 5\ 7\ 39\ 293\ 171\ Bresenham$$

共耗时 **2.475**s。

2.2 多边形绘制算法（已完成）

2.2.1 DDA 算法

2.2.1.1 原理介绍

对于多边形绘制的每一条边，使用线段绘制的 DDA 算法。

2.2.2 BresenHam 算法

2.2.2.1 原理介绍

对于多边形绘制的每一条边，使用线段绘制的 BresenHam 算法。

2.3 椭圆绘制算法（已完成）

2.3.1 中点椭圆生成算法

2.3.1.1 原理介绍

椭圆的生成可以简化为中心在原点的椭圆在第一象限内的椭圆弧的生成，其余三个象限利用对称性可以轻松得到，最后将椭圆平移至指定位置即可。由椭圆方程可知斜率m：

$$m=\frac{dy}{dx}=-2\frac{{r_y}^2x}{{r_x}^2y}$$

故在第一象限内由$m=-1$分割为两个区域，分割边界是：

$$2{r_y}^2x\geqslant 2{r_x}^2y$$

在上半部分，以x方向取单位步长；在下半部分以y方向取单位步长。

定义椭圆函数：

$$f_{ellipse}(x,\ y)={r_y}^2x^2+{r_x}^2y^2-{r_x}^2{r_y}^2$$

则在区域 1 中决策参数$p1_k$为：

$$p1_{k+1}=\begin{cases}p1_k+2{r_y}^2x_k+3{r_y}^2 & p_k<0\\ p1_k+2{r_y}^2x_k-2{r_x}^2y_k+2{r_x}^2+3{r_y}^2 & p_k\geqslant 0\end{cases}$$

$$p1_0={r_y}^2-{r_x}^2r_y+\frac{{r_x}^2}{4}$$

在区域 2 中决策参数$p2_k$为：

$$p2_{k+1}=\begin{cases} p2_k-2{r_x}^2y_k+3{r_x}^2 & p_k<0 \\ p2_k+2{r_y}^2x_k-2{r_x}^2y_k+2{r_y}^2+3{r_x}^2 & p_k\geqslant 0 \end{cases}$$

$$p2_0={r_y}^2\left(x+\frac{1}{2}\right)^2+{r_x}^2(y_k-1)^2-{r_x}^2{r_y}^2$$

2.3.1.2 算法理解

椭圆的中点生成算法其实是线段的 BresenHam 算法的另一个应用，均是通过决策参数p_k的符号从候选点中选出最接近实际方程的点。候选点的增多并没有带来递推复杂性的增加。不同的一点是$|m|$会随着生成过程而跨越 1 的分界，故需要在分界处改变单位步长的选取。

2.3.1.3 算法优化

除了前述提及的利用对称性，可将${r_x}^2$，${r_y}^2$等常量提前算出，避免重复计算。

2.3.1.4 性能测试

暂未进行性能测试。

2.4 圆生成算法（已完成）

2.4.1 中点圆生成算法

2.4.1.1 原理介绍

圆的生成可以视作椭圆生成的特例，其中$r_x=r_y$。也可以利用圆本身的对称性只计算$\frac{1}{8}$圆弧，避免了单位步长选取的改变。本作业中我选择了后一种方式。以下只给出决策参数p_k的推导式：

$$p_{k+1}=\begin{cases} p_k+2x_k+3 & p_k<0 \\ p_k+2x_{k+1}+3-2y_k-2 & p_k\geqslant 0 \end{cases}$$

$$p_0=\frac{5}{4}-r$$

2.5 裁剪算法（已完成）

2.5.1 Cohen-Sutherland 裁剪算法

2.5.1.1 原理介绍

Cohen-Sutherland 算法通过区域检查的方法，根据线段端点与四条裁剪窗口的相对位置关系，将线段端点按区域赋以四位二进制编码，能够有效减少求交点的次数。

将区域码各位从右到左编号:

- 位 1：上边界
- 位 2：下边界
- 位 3：右边界
- 位 4：左边界

区域码位为 1，表示端点落在相应的位置上。

区域码位为 0，表示端点不在相应的位置上。

根据区域码能够快速排除线段位于窗口内和线段端点位于窗口同侧外部的情况。对于其他情况，则按“左-右-上-下”顺序用裁剪边界检查线段端点，裁剪掉对应部分，将剩余部分再次判断。每条线段求交次数至多为 4 次。

2.5.1.2 算法理解

Cohen-Sutherland 算法在直接求交的基础上利用了线段端点和裁剪窗口之间的位置关系信息，使得能够直接判定一些位置关系下的裁剪结果，实质上是通过编码测试来减少求交的次数。

2.5.1.3 算法优化

在求解端点的区域码时，可采用如下策略：由于不同位置分别对应不同的位，故可以分别求解左右关系和上下关系，对应结果的高两位和低两位，并使用“或操作”进行整合。详见代码。

2.5.1.4 性能测试

- 以 Cohen-Sutherland 算法运行以下命令 20000 次，其中线段与裁剪边界有**两交点**。

$$drawLine\ 5\ 7\ 39\ 93\ 71\ DDA$$
$$clip\ 5\ 33\ 10\ 70\ 58\ Cohen-Sutherland$$

  共耗时 **9.982**s。

- 以 Cohen-Sutherland 算法运行以下命令 100000 次，其中线段与裁剪边界**同侧无交点**。

$$drawLine\ 5\ 7\ 39\ 93\ 71\ DDA$$
$$clip\ 5\ 112\ 103\ 200\ 230\ Cohen-Sutherland$$

  共耗时 **1.129**s。

2.5.2 Liang-Barsky 裁剪算法

2.5.2.1 原理介绍

Liang-Barsky 认为，线段裁剪的基本问题是二维窗口与一维线段的维数不匹配问题，故他们将待裁剪线段和裁剪矩形窗口均看作点集，而裁剪结果为两点集的交集。

线段的参数表示为：

$$\begin{cases} x = x_1 + u \cdot \triangle x \\ y = y_1 + u \cdot \triangle y \end{cases} \qquad 0 \leqslant u \leqslant 1$$

则裁剪条件的参数化表示为：

$$\begin{cases} xw_{\min} \leqslant x_1 + u \cdot \triangle x \leqslant xw_{\max} \\ yw_{\min} \leqslant y_1 + u \cdot \triangle y \leqslant yw_{\max} \end{cases}$$

上式可统一为$u \cdot p_k \leqslant q_k$，其中，$k = 1$，$2$，$3$，$4$，分别对应裁剪窗口的左，右，上，下边界。参数$p$，$q$定义如下：

$$\begin{cases} p_1 = -\triangle x,\ q_1 = x_1 - xw_{\min} \\ p_2 = \triangle x,\ q_2 = x_{\max} - xw_1 \\ p_3 = -\triangle y,\ q_3 = y_1 - yw_{\min} \\ p_4 = \triangle y,\ q_4 = y_{\max} - yw_1 \end{cases}$$

根据p_k可以确定线段与裁剪窗口的相互位置关系。根据线段的延展方向更新线段与裁剪边界的交点，即可得到裁剪结果的u值。

2.5.2.2 算法理解

算法的核心公式为：$u \cdot p_k \leqslant q_k \qquad k = 1, 2, 3, 4$。也即有：

$$\begin{cases} 0 \leqslant u_k \leqslant \dfrac{q_k}{p_k} & p_k > 0 \\ 1 \geqslant u_k \geqslant \dfrac{q_k}{p_k} & p_k < 0 \end{cases}$$

所以，$p_k > 0$时，取u_k较小部分，线段从裁剪边界延长线内部延伸到外部；$p_k < 0$时，取u_k较大部分，线段从裁剪边界延长线外部延伸到内部。该公式也给出了更新u值的选取策略。$p_k = 0$时，$u \cdot p_k = 0$，在$q_k \geqslant 0$时恒成立，在$q_k < 0$时恒不成立。

将线段延展成为直线，则该直线与裁剪窗口所在直线必有 4 个交点（不平行时），u_1为一侧交点，u_2为另一侧交点，则无论方向如何，u_1均与直线由外到

内与边界的交点有关，u_2均与直线由内到外与边界的交点有关。对于线段同理，只是可能出现负值。

对于$k=1,2,3,4$，相当于分别考察线段所在直线与四条裁剪边界之一所在直线的关系。每一次更新u_1,u_2的值，表示当满足不等式条件时，两直线交点的位置。若出现$u_1>u_2$的矛盾情况，则说明不等式不满足，也即线段在裁剪窗口外，应舍去。

2.5.2.3 算法优化

暂无优化。

2.5.2.4 性能测试

- 以 Liang-Barsky 算法运行以下命令 20000 次，其中线段与裁剪边界有**两交点**。

$$drawLine\ 5\ 7\ 39\ 93\ 71\ DDA$$
$$clip\ 5\ 33\ 10\ 70\ 58\ Liang-Barsky$$

  共耗时 **10.008**s。

- 以 Liang-Barsky 算法运行以下命令 100000 次，其中线段与裁剪边界**同侧无交点**。

$$drawLine\ 5\ 7\ 39\ 93\ 71\ DDA$$
$$clip\ 5\ 112\ 103\ 200\ 230\ Liang-Barsky$$

  共耗时 **1.168**s。

2.6 曲线生成算法

2.6.1 Bezier 曲线

2.6.1.1 原理介绍

n次 Bernstein 基函数的多项式形式为：

$$BEZ_{i,\,n}(u)=C_n^i u^i(1-u)^{n-i}$$

由上，给出$n+1$控制顶点位置$P_i=(x_i,\,y_i,\,z_i)\,(i=0,\,1,\,...,\,n)$，这些点混合产生位置向量$P_u$，用来描述$P_0$和$P_i$之间逼近 Bezier 多项式函数的曲线。

$$P_u=\sum_{i=0}^{n}P_i BEZ_{i,\,n}(u)\,(0\leqslant u\leqslant 1)$$

可见，Bezier 多项式的次数k要比控制顶点个数小 1，即四个控制点生成三

次曲线。同时 Bezier 曲线具有**对称性，仿射不变性，凸包性**等性质。

Bezier 曲线生成的分割递归算法即“德克斯特里奥”算法描述了直接利用控制多边形顶点从参数u计算n次 Bezier 曲线的方法，其公式为：

$$P_i^r = \begin{cases} P_i, & r = 0 \\ (1-u)P_i^{r-1} + uP_{i-1}^{r-1}, & r = 1,\ 2,\ ...,\ n;\ \ i = 0,\ 1,\ ...,\ n-r \end{cases}$$

当$r = 0$时，计算结果为控制顶点本身，而曲线上的型值点为$P(u) = P_0^n$。

2.6.1.2 算法理解

Bezier 曲线同样由基函数和对应点坐标加权生成曲线，且由于 Bernstein 基函数的对称性，线性无关性，使得 Bezier 曲线同样具有对称性和仿射不变性。由于基函数的最大值特性，即$BEZ_{i,\,n}(u)\ (i = 0,\ 1,\ ...,\ n)$在$u = i/n$处取得最大值，Bezier 具有**拟局部性**，使其优于自然插值样条，更易修改形状。

对于 Bezier 曲线的“德卡斯特里奥”递推算法，其实质上只是将人自然计算的过程通过递归表达式表现出来，从而省去了定义式中的计算次数i。因此省去了乘法，乘方和组合数的计算，简化了计算形式。

2.6.1.3 算法优化

在实际代码实现中，我选择使用预处理来加快处理速度，而非使用“德卡斯特里奥”算法。由于程序中固定u的步长为 **0.001**，即从 0~1 共有 **1001** 个待求点坐标，故u的乘方可预处理时求出。在项目中我限定最大控制点数为 50，故只需预先求的“**50 以内的所有组合数结果**”和“**0~1 步长为 0.001 的所有 1001 个数的 0~50 次方结果**”。如此，在实际计算时仅需根据定义式直接取得对应的乘方和组合数结果进行计算即可，大大减少了计算复杂度。

根据实际测试，预处理的时间总和为 0.006s，在窗口初始化时进行预处理即可。

2.6.1.4 性能测试

运行以下命令绘制 Bezier 曲线 **1000** 次：

$$drawCurve\ 10\ 4\ Bezier$$
$$28\ 34\ 9\ 86\ 61\ 4\ 129\ 42$$

共耗时：**9.679**s

2.6.2 B-Spline 曲线

2.6.2.1 原理介绍

Bezier 曲线具有一些问题：特征多边形的定点数量直接决定了 Bezier 曲线的次数，这不够灵活；Bezier 曲线只具有拟局部性，修改某个控制点将影响整个曲线；控制多边形在顶点数多时与曲线的拟合度较差，等等。由此引申出 B 样条曲线。

$U_{n,\,k}$上的k阶 B 样条基函数递推公式为：

$$B_{i,\,k}(u)=\left[\frac{u-u_i}{u_{i+k-1}-u_i}\right]B_{i,\,k-1}(u)+\left[\frac{u_{i+k}-u}{u_{i+k}-u_{i+1}}\right]B_{i+1,\,k-1}(u)\,(i=0,\,1,\,...,\,n)$$

由上，可知 B 样条基函数具有局部性，即：

$$\begin{cases}B_{i,\,k}(u)>0 & u\in[u_i,\,u_{i+k}]\\ B_{i,\,k}(u)=0 & u\notin[u_i,\,u_{i+k}]\end{cases}$$

给出$n+1$控制点及参数节点向量$U_{n,\,k}$，则参数曲线$P(u)$为k阶 B 样条曲线：

$$P(u)=\sum_{i=0}^{n}P_iB_{i,\,k}(u),\qquad u\in[u_{k-1},\,u_{n+1}]$$

由上可知，B 样条曲线的次数k与控制顶点个数n无关，每一个u值的计算仅与$k+1$控制点相关。

B 样条曲线的主要性质有：局部性，凸包性，仿射不变性，平面曲线的保型性。

2.6.2.2 算法理解

B 样条曲线的局部性来自于其**基函数的局部性**。基函数仅在$(u_i,\,u_{i+k})$中取正值，其余部分取 0，这使得$P(u)=\sum_{i=0}^{n}P_iB_{i,\,k}(u)$中每一个$B_{i,\,k}(u)$仅仅影响$P_i,\,P_{i+1},\,...,\,P_{i+k}$这$k+1$个点，所以 B 样条曲线在$(u_i,\,u_{i+1})$段中只被$(P_{i-k},\,P_{i-k+1},\,...,\,P_i)$这些控制点控制。这使得 B 样条次数与控制顶点个数无关，而每个控制顶点仅能影响到曲线的一部分。这是 B 样条曲线局部性的保证。

2.6.2.3 算法优化

在项目中实际实现时，由于固定 B 样条基函数的次数不会超过 3，故无需使用k阶 B 样条的递推公式进行求解，而可以使用三次 B 样条曲线的公式直接求解。公式的矩阵表示为：

$$P(u)=\frac{1}{6}[u^3 \quad u^2 \quad u \quad 1]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}P_i\\ P_{i+1}\\ P_{i+2}\\ P_{i+3}\end{bmatrix}$$

对于二次 B 样条曲线和一次 B 样条曲线，同理使用公式即可。

2.6.2.4 性能测试

运行以下命令绘制 Bezier 曲线 1000 次:

$$drawCurve\ 10\ 4\ B-spline$$
$$28\ 34\ 9\ 86\ 61\ 4\ 129\ 42$$

共耗时：**1.688**s

3 系统介绍

截至报告完成时，我使用了 **Windows & QT & C++**的环境，进一步完善了自由画图软件的功能和操作方式。在 QT 与 C++的交互中，我定义主窗口类 **MainWindow** 了一个画板类 **Paint2DWidget**，继承自 **QWidget**，用以进行绘画。

在 **MainWindow** 中进行软件外观的布局，并设置快捷键，完成读取文件，写出文件，复制图元，粘贴图元等操作。通过按钮、Spin Box 等控件的槽函数触发来向 Paint2DWidget 传递信号，控制绘画模式的改变。

在 **Paint2DWidget** 中接收信号，选择特定的绘画模式。监听鼠标的按下，释放，拖动和滚轮事件，并实时调用对应的算法进行绘制，实现了实时交互的图形绘制，平移，旋转，缩放，裁剪等操作。

图像的绘制采用**继承**的方式来实现。定义图像基类 Graphics，接着直线，椭圆，圆，多边形，曲线等图形继承 Graphics，便于使用动态绑定来管理画板上的所有图形。

在完成 GUI 版本的基础上实现了控制台版本的图形绘制。将命令行内输入的文件参数传入到 **Paint2DWidget** 中，解析命令并执行操作，最终保存为图片即可，所有函数均可调用 GUI 版本。

系统框架的实现详见**《系统使用说明书》**

4 附加功能介绍

- 实现了文件的**新建**与**保存**功能。
- 实现了可自由画线的图形类：**RandomLine**，可任意绘画线段。

- 实现了 B 样条曲线的**次数选择**，可选择 **1/2/3** 次 B 样条曲线。
- 实现了曲线在绘制过程中**改变控制点的位置**，用于调整曲线绘制。
- 实现了图形橡皮擦：**Eraser**，可以一笔擦除涉及到的所有图形。
- 实现了图元的**复制**，**粘贴**功能。
- 实现了**撤回**功能，可以撤回已经绘画完毕的图形。若当前正在绘制多边形，则可撤回多边形的每一条边。若当前正在绘制曲线，则可撤回曲线的每个控制点。
- 实现了**复原**功能。在撤回操作执行后可原样复原。包括多边形和曲线的**部分撤回**。
- 实现了**点击选择**和**块选择**功能，选择后可进行平移等操作。
- 实现了生成**带有图标**的 exe 文件。

5 总结

目前已实现 GUI 版本和控制台双版本，且统一到同一个工程下。故进行了一些初步的性能测试。从测试结果可知在我的实现及简单测试中，**BresenHam** 算法优于 **DDA** 算法，**Cohen-Sutherland** 算法略优于 **Liang-Barsky** 算法。**B-spline** 算法优于 **Bezier** 算法。

作业的难点在于算法的具体实现，交互的整体框架以及撤回等辅助功能的实现上。对于 QT，我刚刚入门，所以也遇到了很多困难，参考了很多技术博客，最终将他们解决。这次作业使我锻炼了编码能力，加深了对于图形绘制的算法的理解。

参考文献

[1] https://blog.csdn.net/qq_37233607/article/details/79303173

[2] https://blog.csdn.net/liang19890820/article/details/49874033

[3] https://blog.csdn.net/toby54king/article/details/78880227

[4] https://blog.csdn.net/fanyun_01/article/details/78379019

[5] https://blog.csdn.net/kabuto_hui/article/details/51288063

[6] https://blog.csdn.net/dong_zhihong/article/details/8208139

[7] http://www.voidcn.com/article/p-sflzniok-hx.html

[8] https://github.com/loveyu/simple-graph

[9] https://github.com/triumphalLiu/DrawingBoard

[10] https://github.com/Moya-Zhang/nju-Computer-Graphics

[11] https://blog.csdn.net/linuxwuj/article/details/78548043

[12] https://blog.csdn.net/king16304/article/details/52172115

[13] http://c.biancheng.net/qt/

[14] https://blog.csdn.net/dong_zhihong/article/details/8208139

[15] https://blog.csdn.net/vincent2610/article/details/47948737

[16] https://blog.csdn.net/yangxi_pekin/article/details/37738219

[17] https://blog.csdn.net/jinbing_peng/article/details/44793393

[18] https://blog.csdn.net/q1007729991/article/details/56012253

[19] http://c.biancheng.net/view/529.html

[20] https://www.cnblogs.com/icmzn/p/5100829.html

[21] http://www.whudj.cn/?p=493

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

[23] https://www.cnblogs.com/nobodyzhou/p/5451528.html

[24] https://blog.csdn.net/so_geili/article/details/51172471