《计算机图形学》系统技术报告

作者姓名 银琦　联系电话：18260066573　常用邮箱：141220132@smail.nju.edu.cn

(南京大学 计算机科学与技术系, 南京　210093)

摘　要: 描述了系统中用到的算法及参考资料。

1　绘制直线的算法

1.1 原理：设(x_1, y_1)和(x_2, y_2)分别为所求直线的起点和终点坐标，由直线的微分方程得

$(y_2-y_1)/(x_2-x_1) = m$ =直线的斜率　　(1－1)

可通过计算由 x 方向的增量△x 引起 y 的改变来生成直线：

$x_{i+1}=x_i+\triangle x$　　(1－2)

$y_{i+1}=y_i+\triangle y=y_i+\triangle x \cdot m$　　(1－3)

也可通过计算由 y 方向的增量△y 引起 x 的改变来生成直线：

$y_{i+1}=y_i+\triangle y$　　(1－4)

$x_{i+1}=x_i+\triangle x=x_i+\triangle y/m$　　(1－5)

式(1－2)至(1－5)是递推的。

选定 x_2-x_1 和 y_2-y_1 中较大者作为步进方向(假设 x_2-x_1 较大)，取该方向上的增量为一个象素单位(△x=1)，然后利用式(1－1)计算另一个方向的增量(△y=△x・m=m)。通过递推公式(1－2)至(1－5)，把每次计算出的(x_{i+1}, y_{i+1})经取整后送到显示器输出，则得到扫描转换后的直线。

之所以取 x_2-x_1 和 y_2-y_1 中较大者作为步进方向，是考虑沿着线段分布的象素应均匀.

另外，算法实现中还应注意直线的生成方向，以决定 Δx 及 Δy 是取正值还是负值。

1.2 步骤：

输入直线的起点、终点。

计算 x 方向的间距△X 和 y 方向的间距△Y。

确定单位步进，取 MaxSteps = max（△X，△Y）；若△X>=△Y，则 X 方向的步进为单位步进，X 方向步进一个单位，Y 方向步进△Y/MaxSteps，否则相反。

设置第一个点的像素值。

令循环初始值为 1，循环次数为 MaxSteps，定义变量 x,y，执行以下计算：

a. x 增加一个单位步进，y 增加一个单位步进

b. 设置位置为（x,y）的像素值

2　绘制圆和椭圆的算法

2.1 中点圆算法

2.1.1　原理：构造 F(X,Y)=X^2+Y^2-R^2=0

中点 M=(Xp+1,Yp-0.5)

$$d = F(M) = F(x_p + 1, y_p - 0.5)$$

$$= (x_p + 1)^2 + (y_p - 0.5)^2 - R^2$$

当 F(M)＜0 时，M 在圆内，P1 距离圆弧近，取 P1

当 F(M)＞0 时，M 在圆外，P2 距离圆弧近，取 P2

若 d<0，取 P2 为下一象素，再下一象素的判别式为

$$d' = F(x_p + 2, y_p - 0.5) = (x_p + 2)^2 + (y_p - 0.5)^2 - R^2 = d + 2x_p + 3$$

若 d>=0，取 P1 为下一象素，再下一象素的判别式为

$$d' = F(x_p + 2, y_p - 1.5) = (x_p + 2)^2 + (y_p - 1.5)^2 - R^2 = d + 2(x_p - y_p) + 5$$

初始象素是（0,R），判别式 d 的初值为

$$d_0 = F(1, R - 0.5) = 1.25 - R$$

使用 e=d-0.25 代替 d ,e0=1-R

2.1.2 步骤：获取圆心 P 和半径 r。

计算决策参数的初始值:p0 = 5/4 – r。

在 x_k 每个位置，从 k = 0 开始，完成下列测试：假如 $p_k < 0$，圆心在(0, 0)的圆的下一个点为(x_{k+1}, y_k)，并且 $p_{k+1} = p_k + 2*x_{k+1} +1$，否则，圆的下一个点是$(x_{k+1}, y_k - 1)$，并且 $p_{k+1} = p_k + 2*x_{k+1} +1 – 2*y_{k+1}$ 其中 $2*x_{k+1} = 2*x_k + 2$ 且 $2y_{k+1} = 2y_k – 2$。

确定在其他七个八分圆中的对称点。

将每个计算出的像素位置移动到圆心在 P 的圆路径上。

重复步骤三到步骤五，直至 x >= y。

2.2 中点椭圆算法

2.2.1 原理：中点在坐标原点，焦点在坐标轴上（轴对齐）的椭圆的平面集合方程是：

$x^2 / a^2 + y^2 / b^2 = 1$，也可以转化为如下非参数化方程形式：

$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$ （方程 1）

无论是中点画线算法、中点画圆算法还是中点画椭圆算法，对选择 x 方向像素 Δ 增量还是 y 方向像素 Δ 增量都是很敏感的。举个例子，如果某段圆弧上，x 方向上增量+1 个像素时，y 方向上的增量如果 <1，则比较适合用中点算法，如果 y 方向上的增量 >1，就会产生一些跳跃的点，最后生成的光栅位图圆弧会有一些突变的点，看起来好像不在圆弧上。因此，对于中点画圆弧算法，要区分出椭圆弧上哪段 Δx 增量变化显著，哪段 Δy 增量变化显著，然后区别对待。由于椭圆的对称性，我们只考虑第一象限的椭圆圆弧。

定义椭圆弧上某点的切线法向量 N 如下：

$$N(x,y) = \frac{\partial F(x,y)}{\partial x} i + \frac{\partial F(x,y)}{\partial y} j$$

$$= 2b^2 xi + 2a^2 yj$$

对方程 1 分别求 x 偏导和 y 偏导，最后得到椭圆弧上(x,y)点处的法向量是（2b2x，2a2y）。dy/dx = -1 的点是椭圆弧上的分界点。此点之上的部分椭圆弧法向量的 y 分量比较大，即：2b2(x + 1) < 2a2(y - 0.5)；此点之下的部分椭圆弧法向量的 x 分量比较大，即：2b2(x + 1) > 2a2(y - 0.5)。

对于上部区域，y 方向每变化 1 个单位，x 方向变化大于一个单位，因此中点算法需要沿着 x 方向步进画点，x 每次增量加 1，求 y 的值。同理，对于下部区域，中点算法沿着 y 方向反向步进，y 每次减 1，求 x 的值。

先来讨论上部区域椭圆弧的生成：

假设当前位置是 $P(x_i, y_i)$，则下一个可能的点就是 P 点右边的 $P1(x_i+1, y_i)$点或右下方的 $P2(x_i+1, y_i-1)$点，取舍的方法取决于判别式 d_i，d_i的定义如下：

$d_i = F(x_i+1, y_i-0.5) = b_2(x_i+1)_2 + a_2(y_i-0.5)_2 - a_2b_2$

若 $d_i < 0$，表示像素点 P1 和 P2 的中点在椭圆内，这时可取 P1 为下一个像素点。此时 $x_{i+1} = x_i + 1$，$y_{i+1} = y_i$，代入判别式 d_i得到 d_{i+1}：

$d_{i+1} = F(x_{i+1}+1, y_{i+1}-0.5) = b_2(x_i+2)_2 + a_2(y_i-0.5)_2 - a_2b_2 = d_i + b_2(2x_i + 3)$

计算出 d_i的增量是 $b_2(2x_i + 3)$。同理，若 $d_i >= 0$，表示像素点 P1 和 P2 的中点在椭圆外，这时应当取 P2 为下一个像素点。此时 $x_{i+1} = x_i + 1$，$y_{i+1} = y_i - 1$，代入判别式 d_i得到 d_{i+1}：

$d_{i+1} = F(x_{i+1}+1, y_{i+1}-0.5) = b_2(x_i+2)_2 + a_2(y_i-1.5)_2 - a_2b_2 = d_1 + b_2(2x_i+3) + a_2(-2y_i+2)$

计算出 d_i的增量是 $b_2(2x_i+3)+a_2(-2y_i+2)$。计算 d_i的增量的目的是减少计算量，提高算法效率，每次判断一个点时，不必完整的计算判别式 d_i，只需在上一次计算出的判别式上增加一个增量即可。

接下来继续讨论下部区域椭圆弧的生成：

假设当前位置是 $P(x_i, y_i)$，则下一个可能的点就是 P 点左下方的 $P1(x_i -1, y_i-1)$点或下方的 $P2(x_i, y_i-1)$点，取舍的方法同样取决于判别式 d_i，d_i的定义如下：

$d_i = F(x_i+0.5, y_i-1) = b_2(x_i+0.5)_2 + a_2(y_i-1)_2 - a_2b_2$

若 $d_i < 0$，表示像素点 P1 和 P2 的中点在椭圆内，这时可取 P2 为下一个像素点。此时 $x_{i+1} = x_i + 1$，$y_{i+1} = y_i - 1$，代入判别式 d_i得到 d_{i+1}：

$d_{i+1} = F(x_{i+1}+0.5, y_{i+1}-1) = b_2(x_i+1.5)_2 + a_2(y_i-2)_2 - a_2b_2 = d_i + b_2(2x_i+2)+a_2(-2y_i+3)$

计算出 d_i的增量是 $b_2(2x_i+2)+a_2(-2y_i+3)$。同理，若 $d_i >= 0$，表示像素点 P1 和 P2 的中点在椭圆外，这时应当取 P1 为下一个像素点。此时 $x_{i+1} = x_i$，$y_{i+1} = y_i - 1$，代入判别式 d_i得到 d_{i+1}：

$d_{i+1} = F(x_{i+1}+0.5, y_{i+1}-1) = b_2(x_i+0.5)_2 + a_2(y_i-2)_2 - a_2b_2 = d_1 + a_2(-2y_i+3)$

计算出 d_i的增量是 $a_2(-2y_i+3)$。

2.2.2 步骤：

中点画椭圆算法从(0, b)点开始，第一个中点是(1, b - 0.5)，判别式 d 的初始值是：

$d_0 = F(1, b - 0.5) = b_2 + a_2(-b+0.25)$

上部区域生成算法的循环终止条件是：$2b_2(x + 1) >= 2a_2(y - 0.5)$，下部区域的循环终止条件是 $y = 0$，在每个循环条件中应用上面原理所说的公式。

3 绘制 Beizer 曲线的算法

3.1 Bezier曲线介绍

贝塞尔曲线就是这样的一条曲线，它是依据四个位置任意的点坐标绘制出的一条光滑曲线。在历史上，研究贝塞尔曲线的人最初是按照已知曲线参数方程来确定四个点的思路设计出这种矢量曲线绘制法。1962 年，法国数学家 Pierre Bézier 第一个研究了这种矢量绘制曲线的方法，并给出了详细的计算公式，因此按照这样的公式绘制出来的曲线就用他的姓氏来命名是为贝塞尔曲线。

3.2 生成公式

（1）线性公式（只有两个点情况）

给定点 P0、P1，线性贝兹曲线只是一条两点之间的直线。这条线由下式给出：

$$P(t) = P_0*(1-t) + P_1*t, t \in [0,1]$$ 公式 1

且其等同于线性插值。

（2）二次方公式（三个点组成）

二次方贝兹曲线的路径由给定点 P0、P1、P2 的函数 P（t）追踪：

$$P(t) = P_0 * (1-t)^2 + P_1 * 2 * t * (1-t) + P_2 * t^2, t \in [0,1]$$

TrueType 字型就运用了以贝兹样条组成的二次贝兹曲线。

（3）三次方公式（四个点）

P0、P1、P2、P3 四个点在平面或在三维空间中定义了三次方贝兹曲线。曲线起始于 P0 走向 P1，并从 P2 的方向来到 P3。一般不会经过 P1 或 P2；这两个点只是在那里提供方向资讯。P0 和 P1 之间的间距，决定了曲线在转而趋进 P3 之前，走向 P2 方向的“长度有多长”。

曲线的参数形式为：

$$P(t) = P_0 * (1-t)^3 + P_1 * 3 * t * (1-t)^2 + P_2 * 3 * t^2 * (1-t) + P_3 * t^3, t \in [0,1]$$

现代的成象系统，如 PostScript、Asymptote 和 Metafont，运用了以贝兹样条组成的三次贝兹曲线，用来描绘曲线轮廓。

（4）一般参数公式（n+1 个点）阶贝兹曲线可如下推断。给定点 P0、P1、…、Pn，其贝兹曲线即：

$$P(t) = \sum_{i=0}^{n} P_i * BEZ_{i,n}$$

$$BEZ_{i,n}(t) = C_n^i * t^i * (1-t)^{n-i}$$

$$C_n^i = \frac{\mathrm{n!}}{i! * (n-i)!}$$

N 阶的贝兹曲线，即 N-1 阶贝兹曲线之间的插值。

4 绘制 B 样条曲线的算法

我是直接根据最原始的公式画出的 B 样条曲线。

给定 m+n+1 个空间向量 $\overrightarrow{B_k}$，(k=0,1,…,m+n)，称 n 次参数曲线

$$\overrightarrow{P_{i,\ n}}(t) = \sum_{l=0}^{n} \overrightarrow{B_{i+l}} F_{l,n}(t) (0 \le t \le 1)$$

为 n 次 B 样条曲线的第 i 段曲线（i=0,1,…,m），它的全体称为 n 次 B 样条曲线，它具有 $C^{\mathrm{n-1}}$ 连续性。

其中

$$F_{l,n}(t) = \frac{1}{n!} \sum_{j=0}^{n-l} (-1)^j C_{n+1}^j (t+n-l-j)^n (0 \le t \le 1)$$

然后按每一段 n 次 B 样条来画，一共 m+1 段，将 i 的取值从 0 增加到 m。

5 绘制多边形的算法

在 DDA 算法的基础上，将前一条直线的尾当做新直线的头，继续画线，选点结束后，将最后一个点和第一个点相连，即成为了封闭的多边形。

6 绘制六面体的算法

三维六面体逐面画出，使用摄像机。

摄像机视景体： 一个平行六面体，上下左右四个侧面由窗口边界决定，前后两个面是近平面和远平面，沿着 z 轴投影到窗口，外面的则被裁剪。

7 多边形填充算法

水平扫描线填充算法的基本思想是：用水平扫描线从上到下（或从下到上）扫描由多条首尾相连的线段构成的多边形，每根扫描线与多边形的某些边产生一系列交点。将这些交点按照 x 坐标排序，将排序后的点两两成对，作为线段的两个端点，以所填的颜色画水平直线。多边形被扫描完毕后，颜色填充也就完成了。

扫描线填充算法也可以归纳为以下 4 个步骤：

A. 求交，计算扫描线与多边形的交点

B. 交点排序，对第 2 步得到的交点按照 x 值从小到大进行排序；

C. 颜色填充，对排序后的交点两两组成一个水平线段，以画线段的方式进行颜色填充；

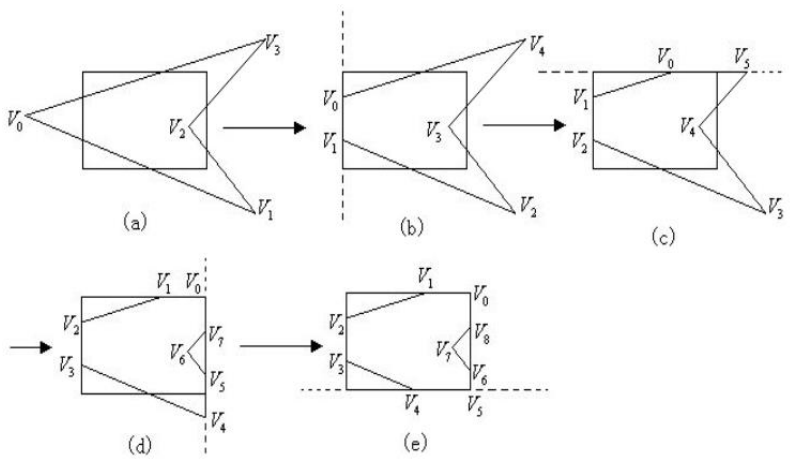
D. 是否完成多边形扫描？如果是就结束算法，如果不是就改变扫描线，然后转 A 继续处理；

8 多边形裁剪算法

分割处理策略：

将多边形关于矩形窗口的裁剪分解为多边形关于窗口四边所在直线的裁剪。

流水线过程(左上右下)：左边的结果是右边的开始。

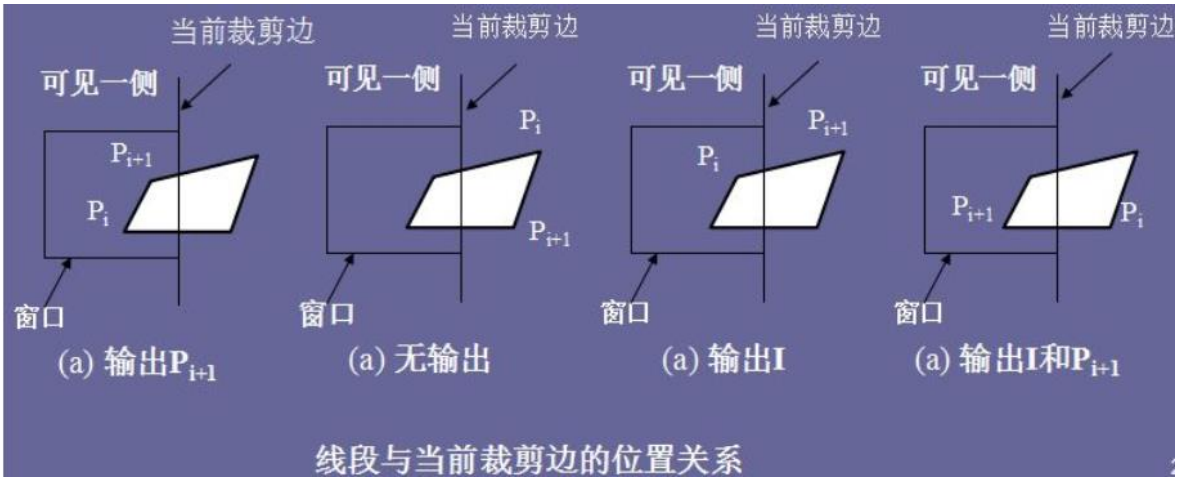


多边形裁剪算法流程图示例

通过两个顶点分别在矩形窗口的可见一侧与不可见一侧来划分多边形对应顶点的输出，如下图所示（I 为多边形的边与裁减边的交点）

总体上来说：

在剪取过程中，实际是多边形的每一边与窗口的一边界进行比较，从而确定它们的位置关系。多边形是



用顶点表示的，相邻的一对顶点构成一条边。具体实现时首先把待裁剪多边形各顶点按照一定方向有次序地组成顶点序列。然后用窗口的一条边界裁剪多边形，产生新的顶点序列。

当多边形顶点序列中一条边的起点和终点被一窗口边界裁剪时，会遇到边与窗口的四种情况之一，做如下处理：

① 如果起点在窗口边界外侧而终点在窗口边界内侧，则将多边形的该边与窗口边界的交点和终点都加到输出顶点表中；

② 如果两顶点都在窗口边界内侧，则只有终点加入输出顶点表中；

③ 如果起点在窗口边界内侧而终点在外侧，则只将与窗口边界的交点加到输出顶点表中；

④ 如果两个点都在窗口边界外侧，输出表中不增加任何点

但是在裁减凹多边形时有时候会画出一些多余的线段。

9 平移算法

平移变换是指将图形从一个坐标位置移到另一个坐标位置的重定位变换。已知一点的原始坐标是P(x,y)，加上一个沿 X，Y 方向的平移量 t_x 和 t_y ，平移此点到新坐标 $(x + t_x, y + t_y)$ 。

如果对一图形的每个点都进行上述变换，即可得到该图形的平移变换。实际上，线段是通过对其两端点进行平移变换，多边形的平移是平移每个顶点的坐标位置，曲线可以通过平移定义曲线的坐标点位置，用平移过的坐标点重构曲线路径来实现。

平移变换只改变图形的位置，不改变图形的大小和形状。

10 旋转算法

以任意点 Pc(xc,yc）为中心做旋转变换。其变换公式为：

$$x' = (x - x_c)\cos\theta - (y - yc)\sin\theta + x_c$$

$$y' = (x - x_c)\sin\theta - (y - y_c)\cos\theta + y_c$$

此公式的推导过程可以这样考虑，先平移坐标原点（0,0）到（ x_c, y_c ），然后进行旋转变换，变换后再将坐标原点移回到（0,0）。三个过程的结果就是以点（ x_c, y_c ）为中心的旋转变换。

旋转变换只能改变图形的方位，而图形的大小和形状不变。

11 缩放算法

一个图形中的坐标点 P(x,y）若在 X 轴方向变化一个比例系数 sx，在 Y 轴方向变化一个比例系数 sy，则新坐标点 P(x,y）的表达式为：

$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

这一变换称为相对于坐标原点的比例变换, sx 和 sy 分别表示点 P(x,y）沿 X 轴方向和 Y 轴方向相对坐标原点的比例变换系数。比例变换改变图形的大小。

比例变换系数 s_x 和 s_y 可赋予任何正数值。当值小于 1 时缩小图形，值大于 1 则放大图形。当 sx 和 sy 被赋予相同值时，就产生保持图形相对比例一致的变换, s_x 和 s_y 值不等时产生 X 轴方向和 Y 轴方向大小不等的比例变换。 s_x 和 s_y 都指定为 1 时，图形大小不改变。

实际上，相对于坐标原点图形的比例变换，相当于每一点相对于坐标原点的变换，因此，它不但改变图形的大小，而且改变图形的位置。

可以通过选择一个在变换后不改变位置的固定点 Pc(x_c, y_c),来控制图形变换的位置。 例对于多边形图形，固定点的坐标(x_c, y_c)可以选择图形的某个顶点、图形几何中心点或任何其它位置，这样变换后固定点坐标不改变，多边形每个顶点相对于固定点缩放。对于坐标为 P(x, y)的顶点，相对于固定点 Pc(x_c, y_c) 变换后的坐标 P(x, y)可计算为：

$$x' = (x - x_c) \cdot s_x + x_c = x \cdot s_x + x_c(1 - s_x)$$

$$y' = (y - y_c) \cdot s_y + y_c = y \cdot s_y + y_c(1 - s_y)$$

12 性能测试

12.1 绘制直线、圆、椭圆的普遍方法是按下鼠标并拖动进行绘制，而我采用了选点的方法，椭圆还需要输入，交互方面欠缺。

12.2 绘制椭圆时只支持绘制长短轴平行于坐标轴的椭圆，导致椭圆无法旋转。

12.3 绘制圆、曲线以及填充时速度较慢，产生了肉眼可见的延时，在平移、旋转等需要重绘时会出现闪烁。

12.4 填充时边界情况处理的不太好，导致填充后多边形的边界可能被覆盖，不太清晰。

12.5 绘制三维六面体的算法理解的不透彻，导致绘制有缺陷。

12.6 在编辑、填充、裁剪等对图形操作时，只能操作最后一次绘制的图形，没有实现在屏幕上选择图形，交互方面欠缺。

12.7 在窗口大小发生变化后，已经绘制的图形不会产生变化，但是之后再绘制图形会出现比例不均的问题。

12.8 代码中有一些处理函数功能重复，但实现了多次，有些冗余。

13 遇到的问题

13.1 最开始存储点坐标的数据结构是int类型的，在写旋转的时候会产生很大误差，所以将点的类型改为了double，解决了这一问题，但是之后在写填充算法的时候，填充后旋转或者旋转后填充都会出现数组越界的问题，经过多次排查，找到了问题，最后在填充时将数据强制转化为了int型，虽然有误差，但是影响很小。

13.2 对回调函数的机制理解的不够透彻，在没有注意到的空闲函数会访问的函数里很容易产生数组越界，所以访问数组时要进行严格的条件控制。

References:

[1] 孙正兴.计算机图形学教程.机械工业出版社

[2] http://www.cppblog.com/doing5552/archive/2009/01/08/71532.html OpenGL 入门学习

[3] http://blog.csdn.net/zhouzhouzf/article/details/39638619 OpenGL 学习之简单的图形用户界面和交互输入

[4] http://cuiqingcai.com/1776.html OpenGL 绘图实例六之鼠标监听事件

[5] http://blog.csdn.net/clever101/article/details/6076841 DDA 算法和 Bresenham 算法

[6] http://blog.csdn.net/u012866328/article/details/52607439【OpenGL】中点圆、椭圆生成算法

[7] http://blog.csdn.net/syx1065001748/article/details/46594079 扫描线填充算法

[8] http://blog.csdn.net/orbit/article/details/7368996 扫描线填充算法

[9] http://blog.csdn.net/yangtrees/article/details/9026411 绘制 B 样条曲线

[10] http://wenku.baidu.com/view/692adf09763231126edb11e0.html?re=view 图形裁剪

[11] http://blog.csdn.net/haimian520/article/details/15334415使用 openGL 绘制三维场景

[12] http://blog.sina.com.cn/s/blog_78c952670100q3p0.html OpenGL 鼠标控制旋转及缩放 (依赖 glut)

[13] http://blog.csdn.net/wangqinghao/article/details/14002077 gluLookAt 函数详解