系统技术报告

151220129 计科 吴政亿

Bresenham画线算法

- 我们知道，在现实世界和数学中，一条直线是有无限精细的粒度的，而计算机屏幕上却是一个一个的像素，精度有限。所以，我们只有直线的数学表达式并不够，我们还需要决定屏幕上哪些像素应该被选中并组成离散化表示后用像素表示的直线，此时就是Bresenham画线算法发挥作用的时候。

- 首先，算法根据直线斜率来确定每次取样一个单位是在 x 轴上进行，还是 y 轴上进行，取样后，在另一轴上的增量为0还是为1，则是取决于另外的因素——实际直线与最近像素点的距离。设一个直线方程为 $y = mx + b$，以在x轴正方向取样为例，假设在 (x_k, y_k) 处画好了点，则下一个点应该是 (x_{k+1}, y_k) 或者 (x_{k+1}, y_{k+1})，前者在 y 轴上没有增量，后者在 y 轴上有1的增量。

- 在像素 x_{k+1} 处，线段的 y 坐标应为 $y = mx_k + 1 + b = m(x_k + 1) + b$

- 而取样位置处上下两个像素与实际直线的垂直距离为 d_1, d_2

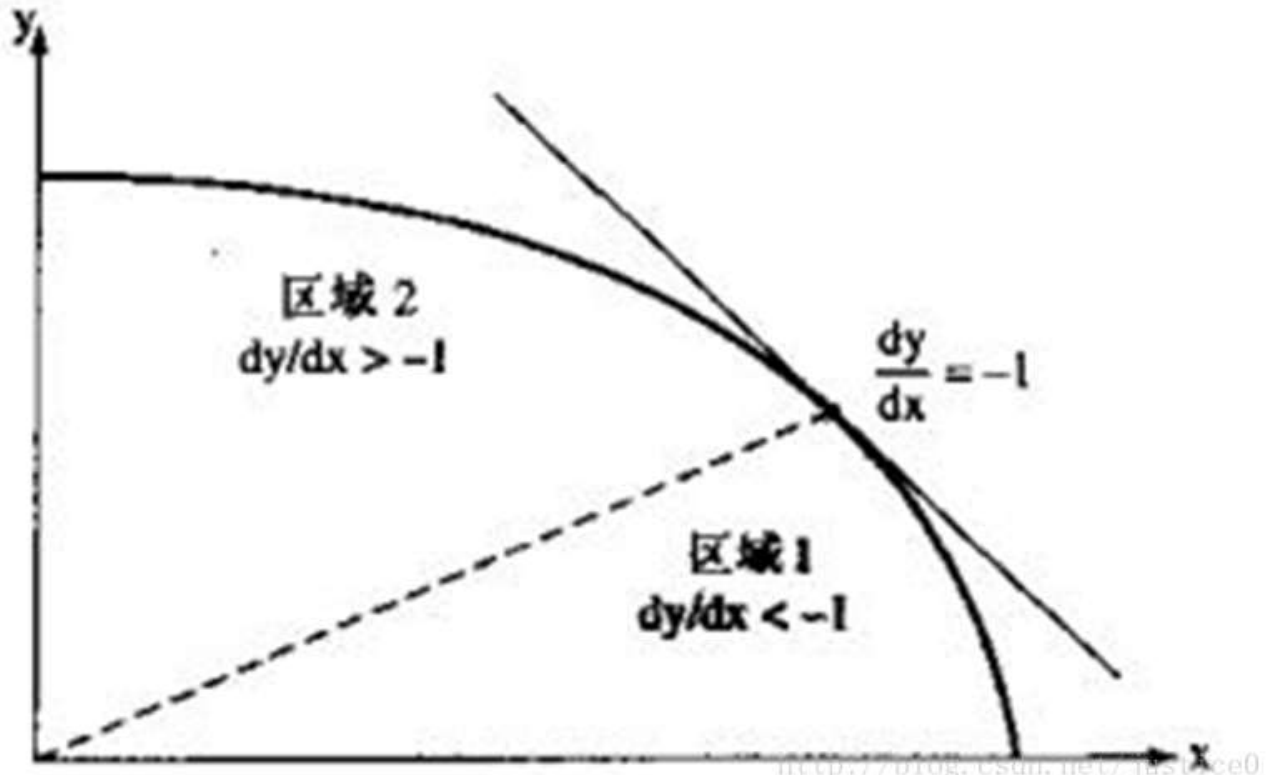
  - $d_1 = y - y_k = m(x_{k+1}) + b - y_k$
  - $d_2 = y_{k+1} - y = y_{k+1} - m(x_{k+1}) - b$

- 两者的差为 $d_1 - d_2 = 2m(x_{k+1}) - 2yk + 2b - 1$

- 而 $m = \frac{\Delta y}{\Delta x}$
- 代入上面有 $\Delta x(d_1 - d_2) = 2\Delta y * x_k - 2\Delta x * y_k + c$（ c是与像素位置无关的常量，可省去)
- 这样，就得到了画线时第k步的决策参数
  - $pk = \Delta x(d_1 - d_2) = 2\Delta y * x_k - 2\Delta x * y_k + c$
- **决策过程如下:**
  - 若 $p_k > 0$，则说明 $d_1 > d_2$，也就是说较高像素(xk+1,yk+1)更接近实际直线
  - 递推可得，下一步时：
    - $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
  - 若 $pk <= 0$，则说明d1 <= d2，也就是说较低像素(xk+1,yk)更接近实际直线
  - 递推可得，下一步时：
    - $p_{k+1} = p_k + 2\Delta y$
- **因此，画线时步骤如下**
  - 画第一个点 (x_0, y_0)
  - 计算常量 $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x$，起始决策参数为 $p_0 = 2\Delta y - \Delta x$
  - 从 $k = 0$ 开始
    - 若 $pk > 0$，则说明 $d_1 > d_2$，也就是说较高像素 (x_{k+1}, y_{k+1}) 更接近实际直线 递推可得，下一步时： $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
    - 若 $pk <= 0$，则说明 $d_1 <= d_2$，也就是说较低像素 (x_{k+1}, y_k) 更接近实际直线 递推可得，下一步时: $p_{k+1} = p_k + 2\Delta y$
  - $k = k + 1$ 重复3,4步骤，共 Δx 次，这样一来，直线就画完了

圆(椭圆)中心点生成算法

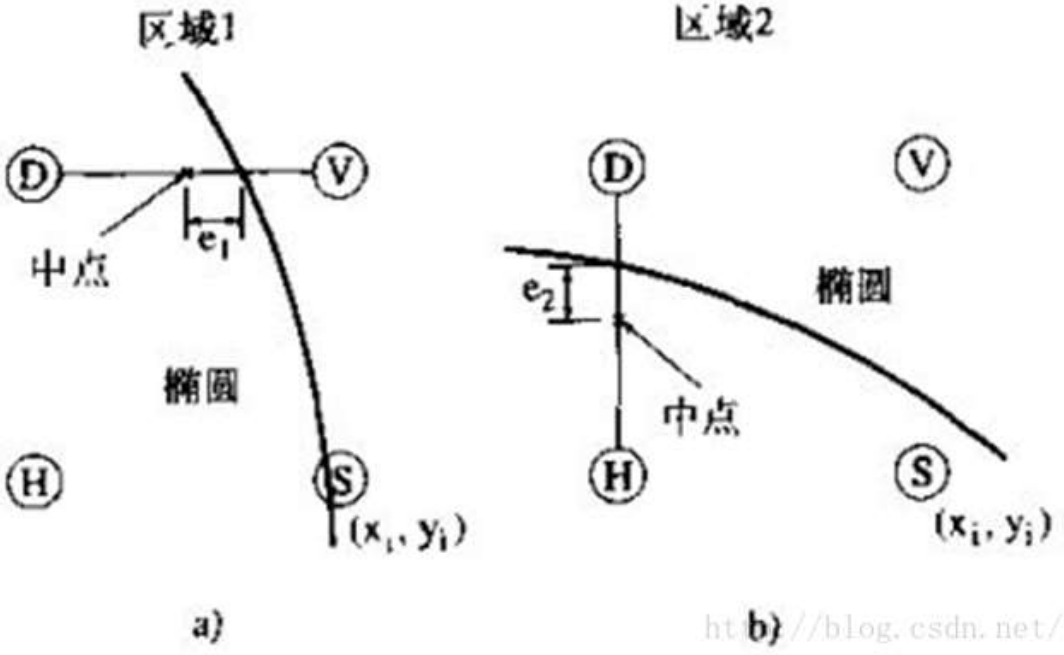
- 此处主要思想与上面的画线算法一致，也是根据每一步计算出的决策参数决定下一个像素该是哪一个像素，不同之处在于，圆（椭圆）在某一个象限内会被分为两个区域，这是因为切线斜率可能会大于1，也可以小于1，而等于1的那一个点就成为了两个区域的分界。
- **圆其实是椭圆的一个特殊情况（焦点重合，离心率为0）**，因此下面以说明椭圆中点生成算法为主。
- 椭圆的非参数化表示为 $\frac{(x - x_0)}{a^2} + \frac{(y - y_0)}{b^2} = 1$
- 传统写法表示为 $f(x, y) = b^2(x - x_0)^2 + a^2(y - y_0)^2 - a^2b^2 = 0$
- 该椭圆中心在 (x_0, y_0)，半长轴为 a，半短轴为 b.

- 

其在第一象限的部分被分为两个区域，如图所示。

区域1：决策参数计算

- 此处参考了Van Aken给出的公式，将决策变量 $d1_i$ 定义为 $f(x, y)$ 在中点处值的两倍，此处在区域1,即 $(x_i - \frac{1}{2}, y_i + 1)$.
- $d1_i = 2f(x_i - \frac{1}{2}, y_i + 1) = 2\left[b^2(x_i - \frac{1}{2})^2 + a^2(y_i + 1)^2 - a^2b^2\right]$
  $= b^2(2x_i^2 - 2x_i + \frac{1}{2}) + a^2(2y_i^2 + 4y_i + 2) - 2a^2b^2$

- 

- 如果椭圆刚好通过中点，则 $d1_i$ 值为0，但很可能，椭圆会经过中点 $(x_i - \frac{1}{2} + e_1, y_i + 1)$ 的右边或左边(e_1 不为0).如上图所示。
- 将中点代入椭圆。则有下面的公式:
  - $$f(x_i - \frac{1}{2} + e_1, y_1 + 1) = b^2(x_i - \frac{1}{2} + e_1) + a^2(y_i + 1) - a^2b^2 = 0$$
  - $$= f(x_i - \frac{1}{2}, y_i + 1) + 2b^2e_1(x_i - \frac{1}{2}) + b^2e_1^2 = 0$$
- 再把 $d1_i = f(x_i - 1/2, y_i + 1)$ 代入，则有
  - $d1_i = -4b^2e_1(x_i - \frac{1}{2}) - 2b^2e_1^2 = -2b^2e_1[(2x_i - 1) + e_1]$
  - $d1_i$ 大于0，则 e_1 小于0，选择左方像素 $(x_i, y_i + 1)$
  - $d1_i$ 小于0，则 e_1 大于0，选择左上方像素 $(x_i - 1, y_i + 1)$
- 区域2同理可得
  - $d2_i = -4a^2e_2(y_i + \frac{1}{2}) - 2a^2e_2^2 = -2a^2e_2[(2y_i + 1) + e_2]$
- 因此，
  - $d2_i$ 大于0，则 e_2 小于0，选择左方像素 $(x_i - 1, y_i)$
  - $d2_i$ 小于0，则 e_2 大于0，选择左上方像素 $(x_i - 1, y_i + 1)$

据此，由对称性可以画出椭圆和圆。

泛滥填充算法

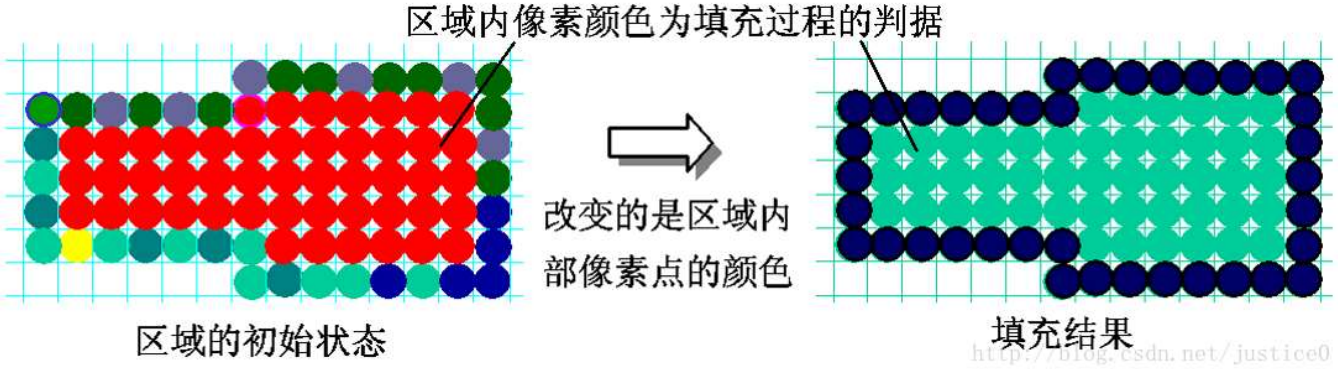
区域内部用单一颜色定义的区域填充。通过替换指定的内部颜色来对这个区域着色(填充)。

递归填充：

从种子点开始，按像素连通定义，递归检测和扩展区域内部像素，并将填充颜色赋给这些像素，直到所有内部点均被着色。

扫描线转换填充：

- 减少对堆栈的存储要求。从每个区间第一个位置开始，按扫描线逐步将区域内扫描线上每个像素颜色替换为填充颜色，直到所有内部点均被着色。



- 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。
- 首先，填充过程中，我们需要两个表，一是当前多边形所有边的表(allEdges)，二是当前多边形与当前扫描线相交的所有边的表(activeEdges)。

填充流程

- 初始化扫描线，扫描线应该从当前多边形y值开始,然后更新活化边表，第一次更新的话的话就是最小y值为扫描线值的两条边.
- 只要活化边表不为空时，执行下面的循环：
  - 以活化边表中相邻两个边的xValue为填充范围的横坐标，以扫描线值为y坐标，填充这一个区域（其实就是画这根线）
  - 填充完后，扫描线值加1
  - 根据扫描线值移除移已经处理完的活化边(扫描线值大于该边最大y值)
  - 然后将活化边表中相邻两个边的xValue（之前用于确定填充横坐标区间的值）更新，也即加上各自斜率的倒数
  - 再次更新一下活化边表
  - 根据活化边表中每条边的xValue来进行排序
- 循环直至活化边表为空（所有边都处理完）,为了防止对多边形编辑后填充不正确，每次编辑完后，都要重新校正allEdges表的值.

旋转

- 物体沿xy平面内圆弧路径重定位。
  - 指定旋转基准点位置 (X_r, Y_r) 和旋转角0 (逆时针旋转为正);
  - 绕通过基准点且垂直于xy平面的旋转轴旋转。
- 基准点为坐标原点:
  - $x_1 = x\cos\theta - y\sin\theta$
  - $y_1 = s\sin\theta + y\cos\theta$
  - $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
- 任意基准位置:
  - $x_1 = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$
  - $y_1 = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$
- 与绕原点旋转的差异:
  - 一个加项(平移项)及坐标值上的多重系数。
- 列向量矩阵: $P_i = R \cdot P$ (旋转矩阵R)
- 行向量矩阵: $P_1^T = (R \cdot P)^T = P^T \cdot R^T$ 。
  - R的转置 R^T :交换行和列;C
  - R的置换:改变sin项符号。
- 不变形的刚体变换: 物体上的所有点旋转相同的角度:

- 直线段:旋转每个线端点;
- 多边形:每个顶点旋转指定旋转角;
- 曲线: 旋转控制/取样点。

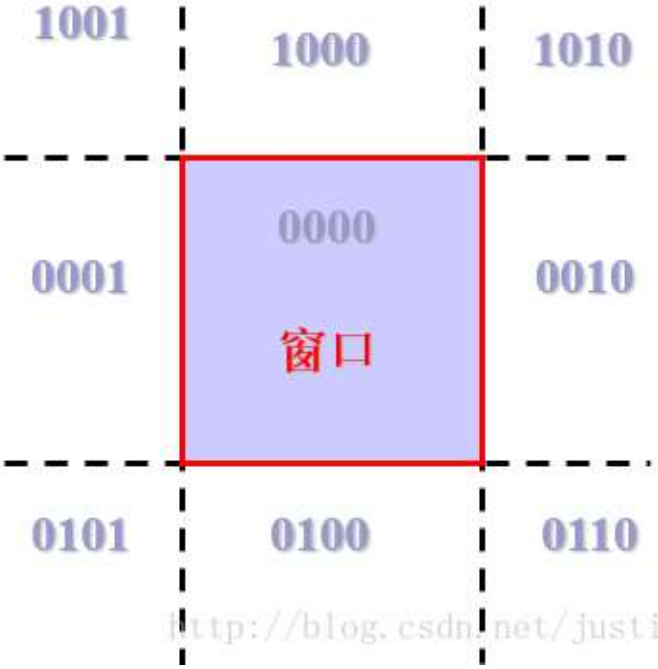
缩放

- 缩放变换改变物体尺寸—— **多边形**。
- 相对于原点的缩放:
  - 将每个顶点坐标(x,y)乘缩放系数 S_x 和 S_y ,产生变换坐标 $(x_1, y_1): x_1 = x \cdot S_x, y_1 = y \cdot S_y.$
  - S_x 和 S_y 分别为x和y方向的缩放。
  - 矩阵形式: $P_1 = S \cdot P$
  - 缩放矩阵S: $S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$
- 缩放系数 s (S_x , S_y) 可赋予任何正数。
  - $s < 1$: 缩小， $s > 1$: 放大;
  - $S_x = S_y$: 一致缩放;
  - $S_x \neq S_y$:差值缩放。
- 多边形: 变换每个顶点的坐标。
- 其它物体: 变换定义物体的参数。
- 缩放变换缩放物体且对其重定位。
  - $s > 1$,坐标位置远离原点。
- 固定点缩放: 选择变换后不改变物体位置的点 (x_f, y_f) 进行缩放。
- 多边形顶点、物体中点。
  - 顶点 (x, y) 缩放后坐标 (x_1, y_1) 为:
    - $y_1 = X * S_x + x_f(1 - S_x)$;
    - $y_1 = Y * S_y + y_f(1 - s_y)$;
- $x_f(1 - s_x)$ 和 $y_f(1 - s_y)$ 对任何点都是常数 $\rightarrow$ 原点缩放+平移:可将此项作为列向量，再将其加到原点缩放上。
- 多边形固定点缩放: 缩放每个顶点到固定点的距离。

裁剪算法

直线裁剪算法Cohen-Sutherland

- Cohen-Sutherland 算法是最早、最流行的线段裁剪算法,
- **核心思想**：通过编码测试来减少要计算交点的次数 — 编码算法。
  - 线段端点按区域赋以四位二进制码(区域码)。



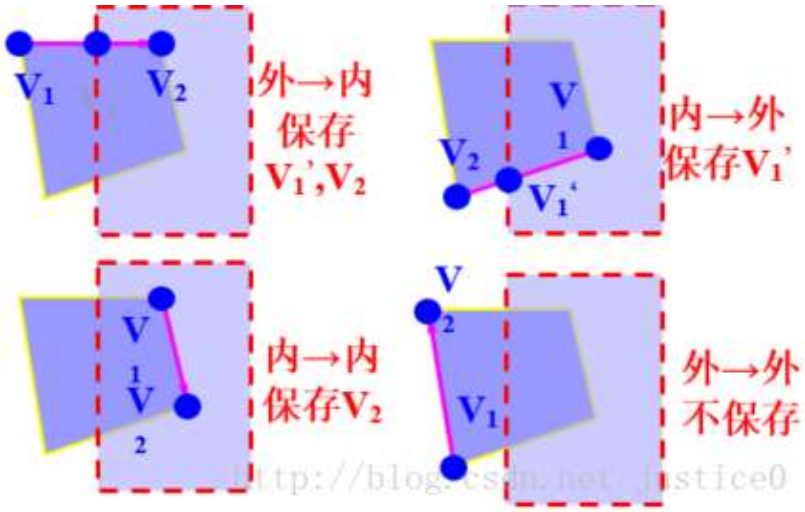
    - 区域码的各位：线段端点与四条裁剪窗口边界的相对位置关系，如图所示.
    - 区域码各位从右到左编号：
        - 位1：上边界
        - 位2：下边界
        - 位3：右边界
        - 位4：左边界
    - 区域码位=1：端点落在相应位置上；区域码位=0：端点不在相应位置上。
    - 区位码标识了端点与裁剪矩形边界的相对位置。
- 算法过程:根据线段端点的区域码快速判断：
    - 完全在窗口边界内的线段：两端点区域码均为0000；
    - 完全在裁剪矩形外的线段：两端点区域码同样位置都为1。
    - 对两个端点区域码进行逻辑与操作，结果不为0000。
    - 不能确定完全在窗口内外的线段:进行求交运算：
    - 按"左-右-上-下"顺序用裁剪边界检查线段端点。
    - 将线段的外端点与裁剪边界进行比较和求交，确定应裁剪掉的线段部分；
    - 反复对线段的剩下部分与其它裁剪边界进行比较和求交，直到该线段完全被舍弃或找到位于窗口内的一段线段为止。

多边形裁剪算法:Sutherland-Hodgeman算法

多边形边界作为一个整体，多边形裁剪是关于裁剪窗口每条边界的裁剪。 裁剪过程：对多边形顶点集初始化(顺或逆时针排序)；依次用裁剪窗口左右上下边界对多边形裁剪；每次裁剪结果依次作为下次裁剪的输入；每次裁剪结果 = 新顶点序列 + 原顶点系列。

基本思想：

将二维空间中的凸多边形区域看成是其各边所在直线形成的多个半空间的交。裁剪窗口边界所在直线将二维空间分成两个半空间：内侧空间 + 外侧空间。依多边形每边与裁剪窗口边界所形成的半空间的相交关系，确定所产生作为下次输入的有效新顶点系列：与内侧空间相交的边界产生有效的新顶点； 依相交方式产生0个、1个或2个有效顶点。



相交方式决定了输出顶点的个数和类型：根据构成多边形边的每对顶点在裁剪窗口半空间中的位置关系确定多边形边界与裁剪窗口的相交方式,见上图.

- 思路
  - 起始点在窗口边界外侧空间，终止点在内侧空间，则该边与窗口边界的交点和终止点都输出。
  - 起始和终止点都在窗口边界内侧空间，则只有终止点输出。
  - 起始点在窗口边界内侧空间，终止点在外侧空间，则只有该边界与窗口边界的交点输出。
  - 起始和终止点都在窗口外侧空间，不增加任何点。
- 裁剪结果多边形顶点由两部分组成：
  - 落在裁剪窗口内侧空间的原多边形顶点
  - 多边形边与裁剪窗口边界的交点。 只要将这两部分顶点按与处理过程相同的顺序(顺时针或逆时针)连接起来，就得到裁剪结果多边形。

3D

1. 实现功能:显示一个正方体, 拖动鼠标时,正方体绕中心点转动 创建正方体时,使用了一个与显示设备(这时指屏幕)无关的空间直角坐标系,我们称之为"对象坐标系".(关于坐标系,图形学中有多种:"世界坐标系"（或称：全局坐标系）、"对象坐标系" (或称:局部坐标系,本地坐标系)、 "观察坐标系"（或称：相机坐标系)..... 本例只有一个物体,无须用世界坐标系来确定它在游戏空间中的绝对位置. 对象坐标系系属"右手系": 用右手握信Z轴,大拇指指向Z轴的正方向,其余四指从X轴正方向到Y轴正方向形成一个弧. 更具体一点: 正对着XOY平面时, Y轴垂直向上增大,X轴水平向右增大,Z轴向观察者方向增大(越往远处Z坐标越小,无限远处为负无穷大).

2. 公式：

- 空间向量(X0,Y0,Z0)旋转角度B的公式:
  - 绕Z轴:

        - $X_1 = X_0 * \cos B - Y_0 * \sin B$
        - $Y_1 = X0 * \sin B + Y_0 * \cos B$
        - $Z_1 = Z_0$
    - 绕X轴:
        - $Y_1 = Y_0 * \cos B - Z_0 * \sin B$
        - $Z_1 = Y_0 * \sin B + Z_0 * \cos B$
        - $X_1 = X_0$
    - 绕Y轴:
        - $X_1 = X_0 * \cos B - Z_0 * \sin B$
        - $Z_1 = X_0 * \sin B + Z_0 * \cos B$
        - $Y_1 = Y_0$
- 空间向量(X_0,Y_0,Z_0)平移的公式为
    - $X_1 = X_0 + T_x$ (T_x 为X轴上平移量,下同)
    - $Y_1 = Y_0 + T_y$
    - $Z_1 = Z_0 + T_z$

  上述公式可以构造成的矩阵乘法,详见"计算机图形学与矩阵".本例未用到矩阵,但对于复杂图形的变换,一定要借助矩阵的乘法运算

3. 图形最终要在设备上输出.我们的图形要在屏幕上显示.这就需要将我们绘制的三维物体投影到二维的屏幕上.按照投影线角度的不同,有两种基本投影:"平行投影"和"透视投影",我们要用的是"平行投影".平行投影又分两种:"正交平行投影"和"斜交平行投影",我们要用的是"正交平行投影": 投影线与投影平面垂直.我们选择的屏幕作为投影面,并设屏幕是与局部坐标系的XOY面平行的一个平面,这样问题就变简单了:只要取每个点的X坐标和Y坐标应可以了,但是:
    - 屏幕坐标系的Y轴方向与我们局部坐标系相反,是向下增大的.所以Y坐标要进行变换(取相反数即可).
    - 还要将投景的成像平移到屏幕适当的地方(本例是将图形的中点平移至屏幕中心). 回忆一下,在二维图形的旋转变换中,一般步骤为: "原点平移"-->旋转--->"复位平移" 在本例正方体旋转变换中,可以将步骤归纳为: "原点平移"-->>旋转--->"投影平移"

Bezier曲线

贝塞尔曲线贝塞尔曲线是计算机图形图像造型的基本工具,是图形造型运用得最多的基本线条之一。它通过控制曲线上的四个点（起始点、终止点以及两个相互分离的中间点）来创造、编辑图形。其中起重要作用的是位于曲线中央的控制线。这条线是虚拟的,中间与贝塞尔曲线交叉,两端是控制端点。移动两端的端点时贝塞尔曲线改变曲线的曲率（弯曲的程度）；移动中间点（也就是移动虚拟的控制线）时,贝塞尔曲线在起始点和终止点锁定的情况下做均匀移动。贝塞尔曲线上的所有控制点、节点均可编辑。

以下公式中：B(t)为t时间下 点的坐标； P0为起点,Pn为终点,Pi为控制点

一阶贝塞尔曲线(线段)：

意义：由P0至P1的连续点,描述的一条线段.

二阶贝塞尔曲线(抛物线)：

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

- 原理：由P0至P1的连续点Q0,描述一条线段。由P1至P2的连续点Q1,描述一条线段。由Q0至Q1的连续点B(t),描述一条二次贝塞尔曲线。
- 经验：P1-P0为曲线在P0处的切线。

三阶贝塞尔曲线：

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

通用公式：$k=0$ 时：$P_i^k = P_i$ $k \neq 0$ 时：$P_i^k = (1-t)P_i^{k-1} + tP_{i+i}^{k-1}, k = 1,2,\ldots,n, i = 0,1,\ldots,n-k$

B样条曲线

给定 $n+m+1$ 个平面或空间顶点 $P_i(i=0,1,\ldots,n+m)$,称为n次参数曲线段：$P_{k,n}(t) = \sum_{i=0}^{n} P_{i+k} G_{i,n}(t), t \in [0,1]$ 为第 k 段 n 次B样条曲线段 $(k=0,1,\ldots,m)$,这些曲线段的全体称为n次B样条曲线,其顶点 $P_i(i=0,1,\ldots,n+m)$ 所组成的多边形称为B样条曲线的特征多边形。其中,基函数 $G_{i,n}(t)$ 定义为：$G_{i,n}(t) = \frac{1}{n!} \sum_{j=0}^{n-i} (-1)^j C_{n+1}^j (t+n-i-j)^n t \in [0,1], i = 0,1,\ldots,n$

二次B样条曲线

取 $n=2$,则有二次B样条曲线的基函数如下：$\begin{cases} G_{0,2}(t) = \frac{1}{2}(t-1)^2 \\ G_{0,2}(t) = \frac{1}{2}(-2t^2+2t+1) \\ G_{0,2}(t) = \frac{1}{2}t^2 \end{cases}$

二次B样条曲线段 $P_{0,2}(t) = \sum_{i=0}^{2} P_i G_{i,2}(t)$ 是一段抛物线。二次B样条曲线的矩阵表示为：

$$P_{0,2}(t) = \frac{1}{2} \begin{bmatrix} l & t & t^2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ -2 & 3 & 0 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \end{bmatrix}, t \in [0,1]$$

端点位置:

$$P_{0,2}(0) = \frac{1}{2}(P_0 + P_1)$$