

Review of Computer Graphics

1 概述

定义

CG中国定义：研究用计算机表示、生成、处理和显示图形的原理、算法、方法和技术的一门学科。

IEEE: the art or science of producing graphical images with the aid of computer

ISO: 一门研究通过计算机将数据转化成图形，并在专门显示设备显示的原理方法和技术的学科。它是建立在传统图学理论、应用数学及计算机科学基础上的一门边缘学科。

主要研究内容

60年代

Sutherland 交互图形学

贝塞尔 贝塞尔曲线、曲面理论

Coons 超限插值，4个曲线插值成一个曲面 Coons Award

70年代

光栅显示器诞生：第一个兴盛期

光栅图形学算法迅速发展

图形软件标准化 ISO发布

1. CGI 计算机图形接口
2. CGM ~~源文件标准
3. GKS 计算机图形核心系统
4. PHIGS 面向程序员的层次交互图形标准

真实感图形学、实体造型技术：简单光照模型

80年代

光线跟踪算法

热辐射度模拟漫反射

超大规模集成电路：硬件基础

70年代汉字之难 80年代解决汉字字形的计算机表示

90年代

GPU

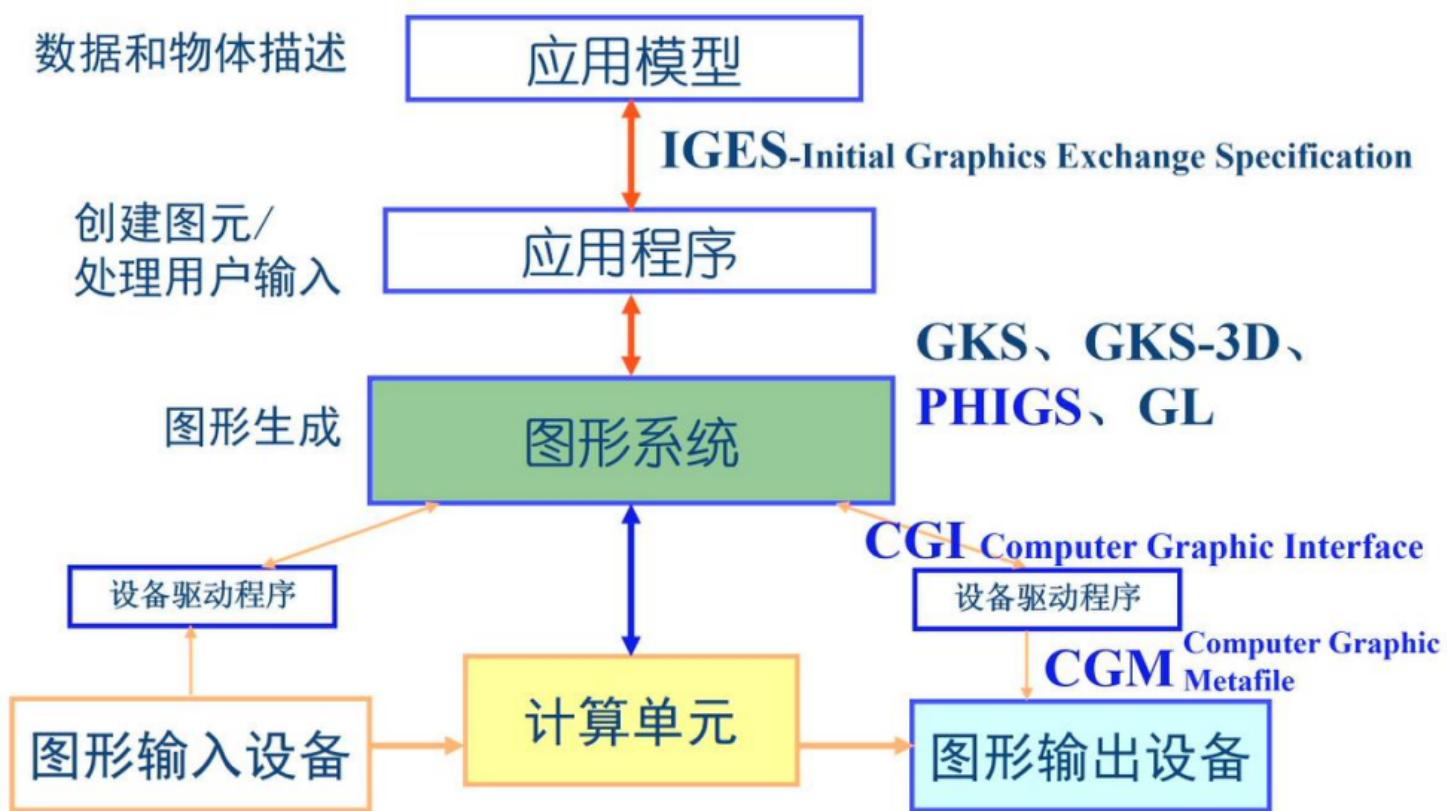
全局光照模型

三维造型技术

交互式图形系统的概念框架

与设备无关、与应用无关

CGI规定硬件接口（图形信息的描述与通信），CGM（图形文件格式），PHIGS（图形系统标准，应用程序和图形输入输出的中介，功能接口）



- 图形数据按层次结构组织；
- 动态修改和绘制图形数据；
- 在三维世界坐标系中操作。

OpenGL

DirectX

Postscript

典型三维模型

犹他壶

西洋跳板棋

山魈

骨骼模型

炸面圈

大众

斯坦福兔

2 三维物体的形体几何模型与表示

内涵：三维物体计算机表示的数据结构和存储结构；三维物体几何形状数据的获取（怎么得到，怎么存储）

多边形网络模型与表示

足够多的多边形平面可无限逼近三维物体表面几何形体。

多边形网格=(点集, 边集, 属性)

属性指定材质

点, 边, 面表示

多边形网格表示三维物体表面几何形体 (三维表面模型)

$$\mathbf{V} = (V_1, V_2, \dots, V_n)$$

$$= ((x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n))$$

$$\mathbf{E} = (E_1, E_2, \dots, E_m)$$

$$\mathbf{F} = (F_1, F_2, \dots, F_l)$$

$$E_1 = (V_1, V_2, P_1, \lambda)$$

$$E_2 = (V_2, V_3, P_2, \lambda)$$

$$E_3 = (V_3, V_4, P_2, \lambda)$$

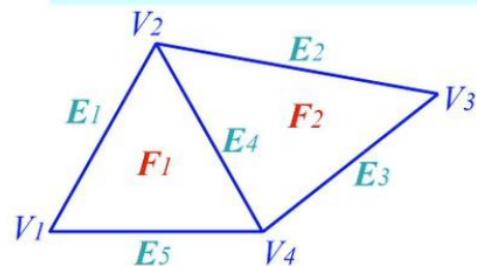
$$E_4 = (V_4, V_2, P_1, P_2)$$

$$E_5 = (V_4, V_1, P_1, \lambda)$$

$$F_1 = (E_1, E_4, E_5)$$

$$F_2 = (E_2, E_3, E_4)$$

三维物体多边形网格
表示的概念层次分解



示例：由两个多边形平面按照
SPHIGS定义的
一个多边形网格
的存储数据结构：**vertex list**、
edge list

点：三维坐标

边：顶点和相邻的面，最外边的面记作 λ

面：由哪些边围成

一致性约束：

1. 所有多边形闭合
2. 一个顶点至少有两条边共享
3. 一条边至少是一个多边形的一部分
4. 每个多边形至少有一条公共边
5. 多边形网格是全连通图
6. 相邻顶点的二元关系可以用一个平面图表示（拓扑平面）

属性

	属性实例
多边形 属性	三角形或不是；平面法向；平面面积
	光洁度；平面方程系数；是否凸多边形；是否有洞
边属性	长度；边的位置（多边形或表面之间）
	该边每一侧的多边形
顶点属性	顶点法向（多边形明暗处理）；顶点关联多边形
	纹理坐标（表面纹理映射）

多边形网格表示的特点：

1. 构造简单，可表示任意三维物体表面几何形体
2. 已形成完善有效的明暗处理方式、硬件可实现
3. 表示精度与多边形网格数量成正比
4. 表示精度可伸缩是多边形网格模型的追求（增加和减少网格）
5. 可编辑是多边形网格表示的挑战

曲面片模型与表示

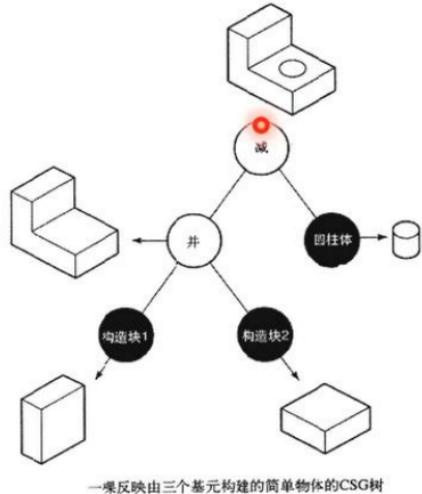
基元是曲边四边形的曲面片

1. 边是三次曲线
2. 精确参数表示，曲面由数学方程定义，面上的每个点都有定义
3. 三维形状编辑潜力，CAD交互式设计基础
4. 可能是一种更经济的物体几何形状表示方法

构造实体几何模型 (CSG: Constructive Solid Geometry)

三维基本构造块组合构建的三维物体形体层次表示（搭积木）

CSG是一种分解表示的有序二叉树，叶子结点是体素或形体变换参数，分支结点是正则集合运算（交并差等）或几何变换操作



CSG是一种分解表示的有序二叉树：
叶子节点是体素或形体变换参数；
分支节点是正则集合运算或几何变换操作；

- CSG树无二义性：定义域由所用体素、允许的几何变换和正则集合运算算子决定。
- 体素不唯一：立方体、圆柱、基本构件等组成一个体素正则集合。

特点

1. 正则集合运算和几何变换描述三维物体组成过程
2. 隐含表示形体几何边界元素
3. 需特殊绘制或多边形网格转换
4. 支持实现交互式实体建模

个人理解：比较偏高层，靠简单实体的搭建来表示复杂几何体。几何边界等元素隐含在搭建过程中，需要实现底层的绘制和网格等才能真正显示几何图形。对用户友好，适合交互式建模。

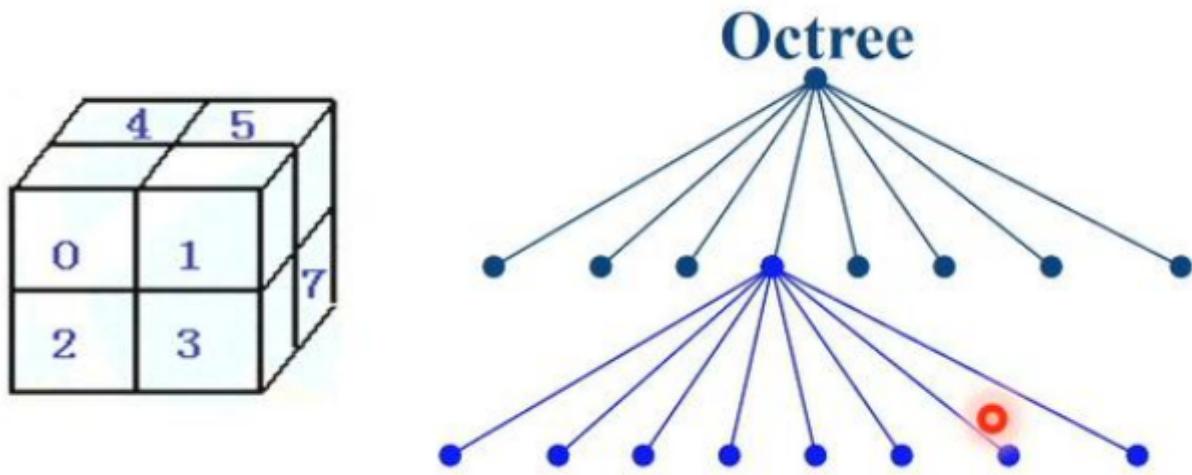
空间细分表示

把三维物体所在的整个世界空间细分为更小的立方体基元，按是否在物体中标记每一个体素（空间微元）

数据存储结构是八叉树，可表示三维实体内部分层树形结构。八叉树可转成二叉空间分区树（BSP）
光流跟踪具有明显优势

医学图像是主要用途之一

八叉树：



按照上图将一个大体素不断划分为8个小体素

1. 几何形体表示，数据结构简单（非解析表达）
2. 简化了形体集合运算
3. 简化了隐藏线、面消除算法（保持体素空间信息）
4. 占用存储多、形体边界计算不易、形体近似表示

小结



本章小结

多边形网格

多边形小平面，表示物体的精确度；
明暗处理很好解决了绘制问题；
复杂物体和动画仍然很棘手；

双三次参数曲面片

曲面片，参数化定义；
很强的交互能力；昂贵的可视化；

构造实体几何

基元构件“构造”刚性实体；
是一种体表示方法；

空间细分技术

被物体占据的三维空间表示；
体素既定义三维物体表面也定义内部组织；

隐函数表示

由解析表达式定义物体表面形状；
可用于形态变化动画；
表示现实物体用途有限；

考虑技术成熟度和计算资源，多边形网格是交互图形引擎最普及支持的三维物体表面几何形体表示模型。

3 三角网格的几何计算

三角网格及其存储表示

三角网格模型：三角形顶点集+顶点间拓扑关系集（边、面等）+属性
两种典型存储结构

obj文件格式

```
# This is a cube with polygon mesh
v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
# 8 vertices
f 1 2 3 4
f 8 7 6 5
f 4 3 7 8
f 5 1 4 8
f 5 6 2 1
f 2 6 7 3
# 6 faces
```

```
# This is a cube with triangular mesh
v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
# 8 vertices
f 1 2 3
f 3 4 1
f 8 7 6
f 6 5 8
f 4 3 7
f 7 8 4
f 5 1 4
f 4 8 5
f 5 6 2
f 2 1 5
f 2 6 7
f 7 3 2
# 12 faces
```

3D坐标表示点

用点下标表示面

(点+点间拓扑关系)

双向链接边表 (half-edge半边表示)

将三角网格的无向边拆成两条有向边，取其中一个（一般是按逆时针顺序）

精确地表示三角网格：只需存储点和拓扑关系

但如果要遍历某个顶点构成的面，只存储点和拓扑关系就需要遍历整个模型，这显然比较浪费时间。

半边：增加少量存储，提供更方便的访问。

半边e和其对边Opposite(e)对应同一条边

存储信息包括：

1. 顶点id和几何信息，即空间坐标
2. 半边e
 - (1) 该半边的源顶点Origin(e)
 - (2) 该半边在同一三角形的下一条边Next(e)

(3) 对边Opposite(e)

(4) 该半边所属面IncFace(e)

由此便于各种网格上的遍历操作

如获得上一条边

```
Prev(e) = Next(Next(e))
```

半边e指向的顶点

```
Target(e) = Origin(Next(e))
```

PPT上的C++代码定义

```
class vertex{
public:
    int id;
    double x;
    double y;
    double z;
    halfEdge* inc_Edge;
    vertex(int id1, double x1, double z1): id(id1), x(x1), y(z1){inc_Edge = NULL};
};

class halfEdge{
public:
    int id;
    halfEdge* opposite;
    halfEdge* prev;
    halfEdge* next;
    vertex* origin;
    face* inc_Face;
    halfEdge(int id1): id(id1){opposite=NULL; prev=NULL; next=NULL; origin=NULL; inc_Face=NULL};
};

class face{
public:
    int id;
    halfEdge* inc_Edge;
    face(int id1): id(id1){inc_Edge=NULL;};
};
```

三角网格的生成

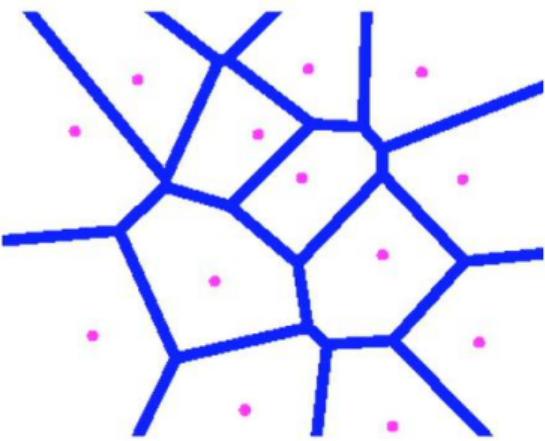
三角网格生成的本质是将三维物体表面划分为三角形面片

现有传感器很容易获取物体表面离散点的空间坐标。

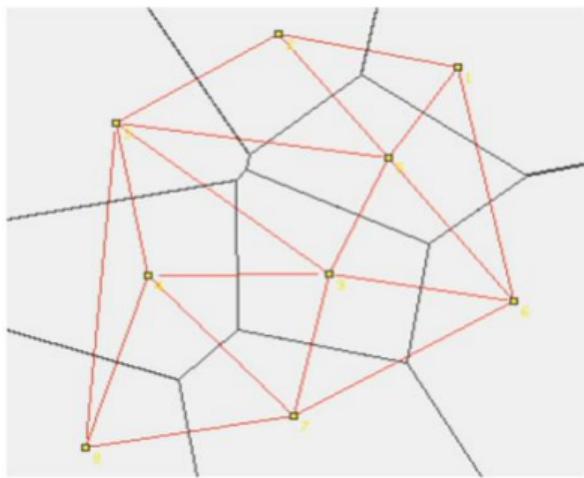
对区域内一组给定点，Voronoi图是对区域内给定一组基点的单元分区；Delaunay三角剖分完成区域内对给定基点的拓扑连接，形成三角形闭合平面。

二维空间的Delaunay三角剖分图和它的对偶Voronoi图

Voronoi图—邻接点的垂直平分线
组成的凸多边形（泰森多边形法）

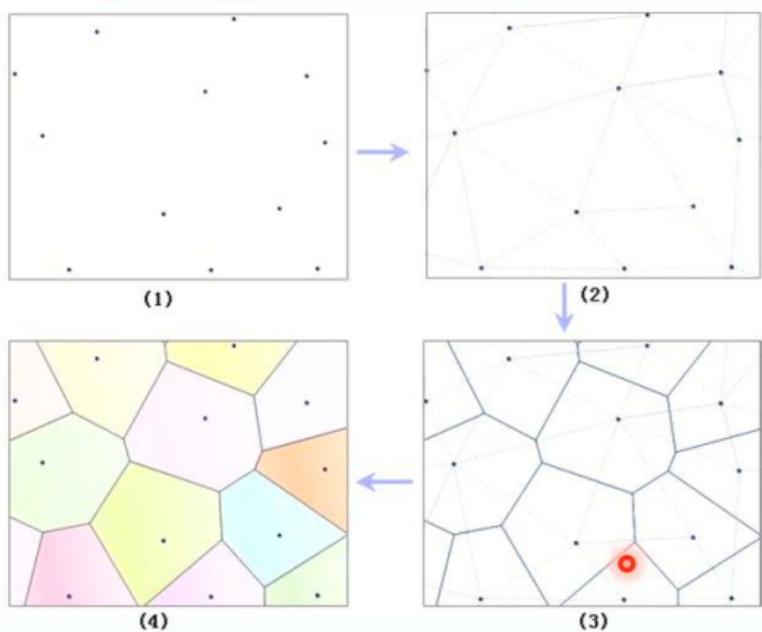


Delaunay三角剖分—三角剖分任意边
的两个端点在**Voronoi图**中是相邻的



二者是对偶关系

Voronoi图生成的泰森法



Voronoi图生成的泰森法：

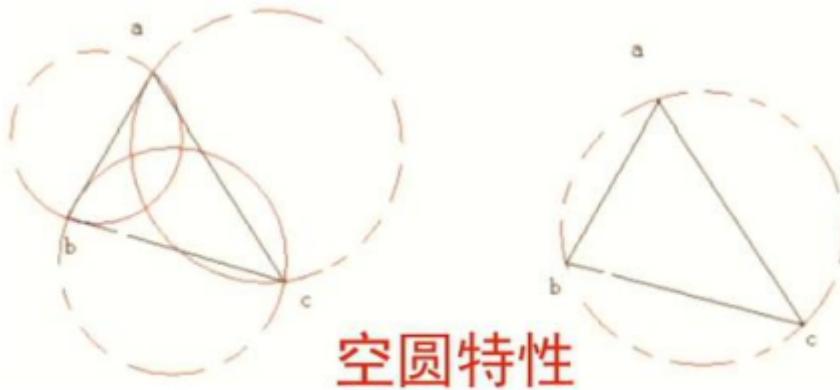
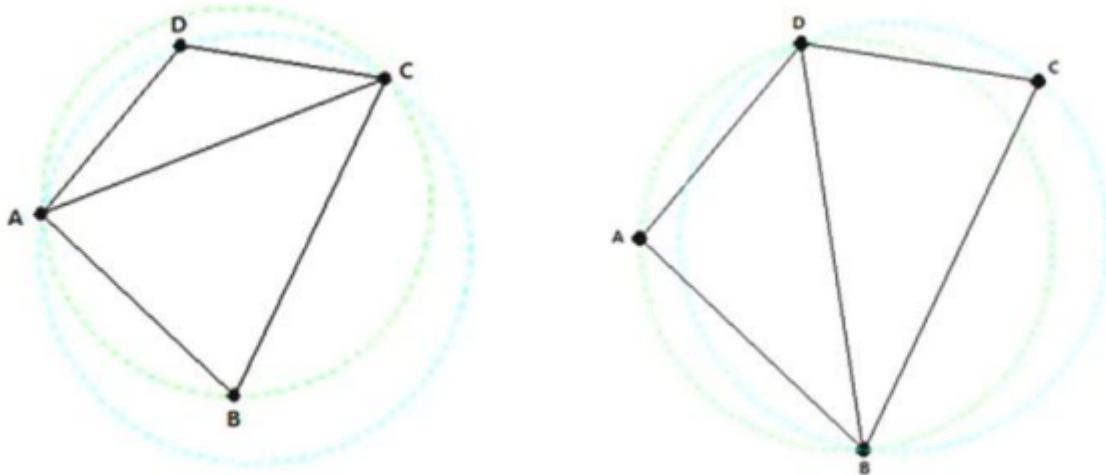
1. 构建Delaunay三角形；
2. 选择一个点的所有相关三角形；
3. 计算每个三角形的外接圆心；
4. 连接圆心。

xier1_d312

Delaunay三角剖分

Delaunay三角剖分是一组相连但不重叠的三角形的集合

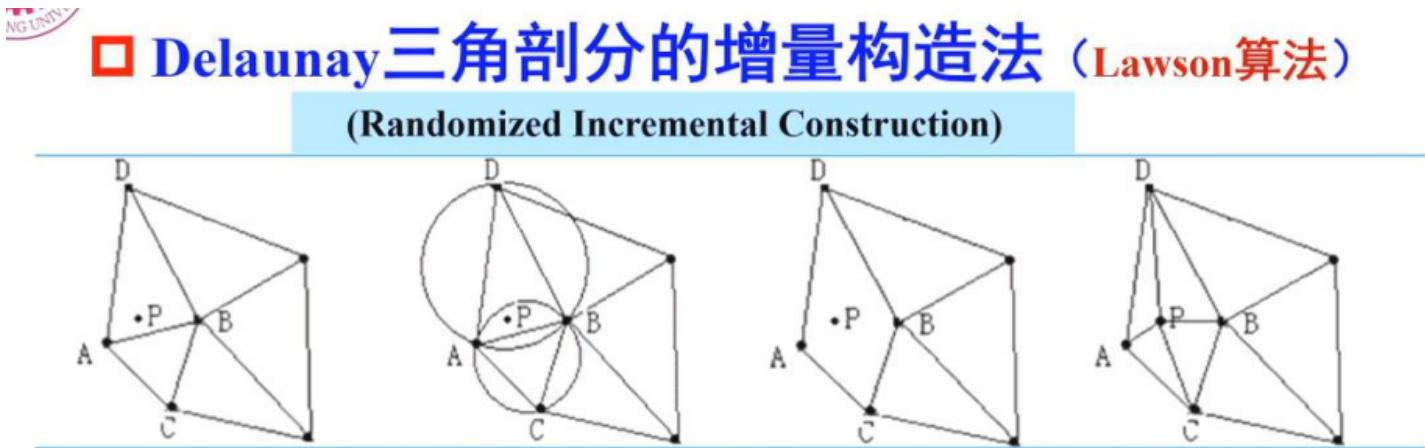
空圆特性：两个共边三角形，任意一个三角形的外接圆中都不包含另一个三角形的第三个顶点。



Delaunay三角剖分的意义：

1. 保证剖分产生的三角形最小角最大（最大化最小角）
2. 可避免狭长三角形产生（Delaunay三角剖分应用）

Delaunay三角剖分的增量构造方法（Lawson算法）



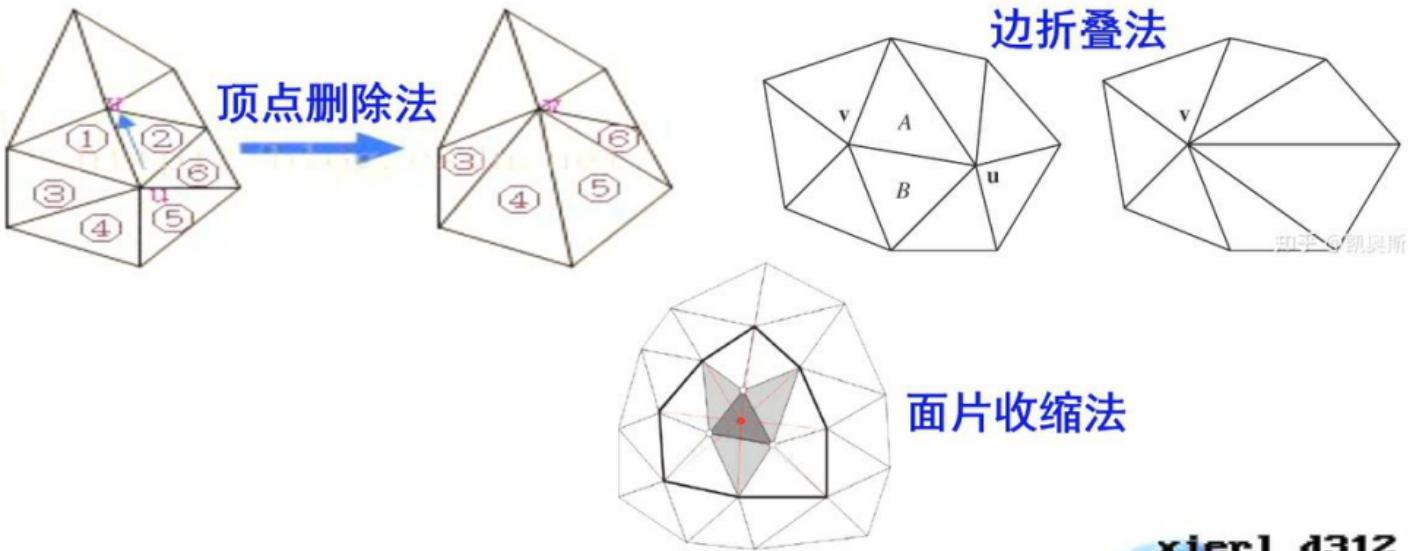
增加散点P的计算过程：决定P连接的顶点；删除失效边；形成新三角形

- 1、构造一个超级三角形，插入三角形链表；
- 2、对点云的一个散点，从三角形链表中找出其外接圆包含该点的三角形。
 删除相关三角形的公共边，将散点同相关三角形所有顶点连接，完成散点所在三角形的链表插入；
- 3、局部三角形优化，记入Delaunay三角形链表；
- 4、循环执行上述第2、3直到所有散点处理完毕。

网格简化

目标：保持三角网格对三维物体表面几何形态逼近的条件下，减少网格的顶点、边和三角形面片的数量。

三条技术路线：顶点删除法、边折叠法、面片收缩法



顶点删除法

1. 删除顶点及相邻的面
2. 空洞填补

顶点可删除性判定条件

顶点平均平面距离小于误差阈值

平均平面距离：

可删除性判别

采用顶点到平均平面的距离定义顶点重要度，对任一顶点 v_i ：
搜索共享 v_i 的三角形组，计算平均平面的法向量、中心点

$$N = \frac{\sum n_k S_k}{\sum S_k} \quad n = \frac{N}{|N|} \quad c = \frac{\sum c_k S_k}{\sum S_k}$$

计算 v_i 到平均平面的距离 $d = \frac{|n_x d_x + n_y d_y + n_z d_z|}{\sqrt{n_x^2 + n_y^2 + n_z^2}}$

当 d (重要度) 小于设定阈值时，即可删除顶点 v_i

n 是平均平面法向量，由各相邻面的法向量对面积加权再归一得到。然后用点到面的距离计算出点到平均平面的距离。

空洞填补

删除顶点后就会得到一个大的多边形，对这个多边形再使用Delaunay三角剖分就可以得到新的三角网格。（凹凸顶点判定的Delaunay三角剖分法）

边折叠法

选择相邻的顶点，删除边(u, v)及其三角形，将两个顶点合并成一个新顶点 w

关键问题：折叠边的选择；新顶点的计算

对任一边 $E_i = (v_{i1}, v_{i2})$ ，搜索共享边 E_i 的相关三角形组：

- 1) 设定边误差矩阵，计算边 E_i 折叠操作带来的误差；
- 2) 通过误差矩阵计算新顶点 v_{i0} 的位置；
- 3) 计算新顶点的折叠误差；
- 4) 按折叠误差从小到大排序，从边序列中取折叠误差最小的边执行折叠操作，更新相关信息；
- 5) 当边序列为空或误差满足阈值时，结束。

面片收缩法

S1、给定多边形表面，将模型所在空间分成小格；

S2、计算每个小格的一个代表顶点，将原始模型落到小格内的顶点合并到代表顶点；

S3、若三角形有一个以上顶点位于同一小格则被删除；

关键问题：小格划分、代表顶点选择

步骤简述：划分小格，选择小格的代表顶点，将小格中的顶点合并到代表顶点（三角形多个顶点位于同一个小格，自然就因为收缩而消失了）

三种方法的总结

顶点删除法：删除不重要的顶点

边折叠法，折叠不重要的边，边顶点合二为一

面收缩法：以小格为单位进行收缩

网格细分

与网格简化相反

目标：用更小的三角平面替代原有三角平面，以更加逼近物体表面几何形状。

需要确定几何规则和拓扑规则：怎么打点？怎么连线？

几何规则：计算新顶点的坐标和顶点的新坐标

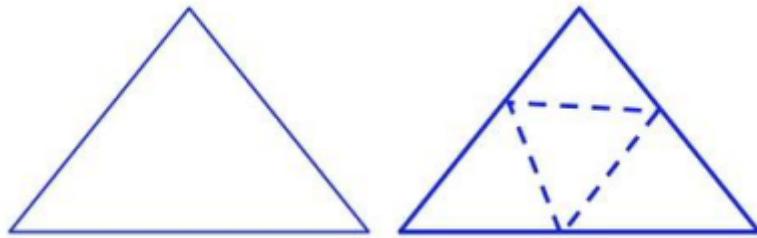
拓扑规则：确定新顶点的连接关系

Loop细分

三角网格逼近型细分方法（分裂）

1-4三角形分裂法

o



为了保证参数连续性（个人理解是由于是空间三维坐标系，本来是一个三角形刚性结构，增加顶点后就变软了，可以向四周弯折保证平滑性）：

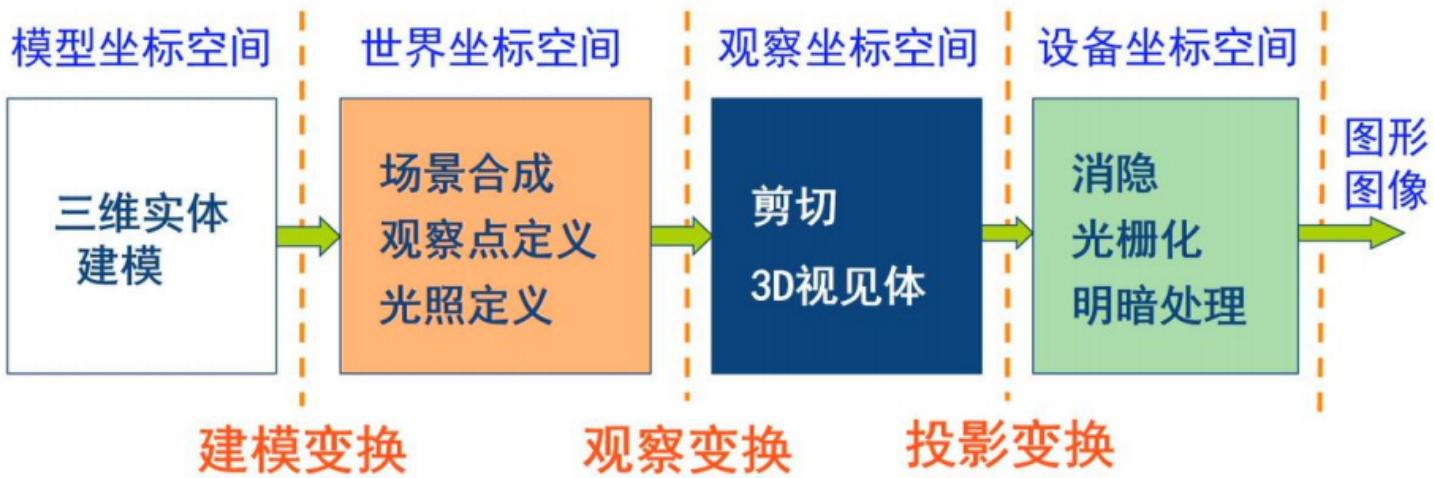
1. 对原始顶点根据邻接顶点进行坐标更新
2. 新增顶点坐标根据共享边的两个三角形顶点进行坐标更新

4 多边形网格的三维物体绘制

图形绘制流水线

图形绘制流水线是将多边形网格表示的三维物体表面几何形体转换为能在显示设备上显示的、经过明暗处理的像素亮度计算过程。

三维物体几何--计算-->能显示的像素



模型→世界坐标表示→观察→显示到设备

可见面判定与隐藏面消除

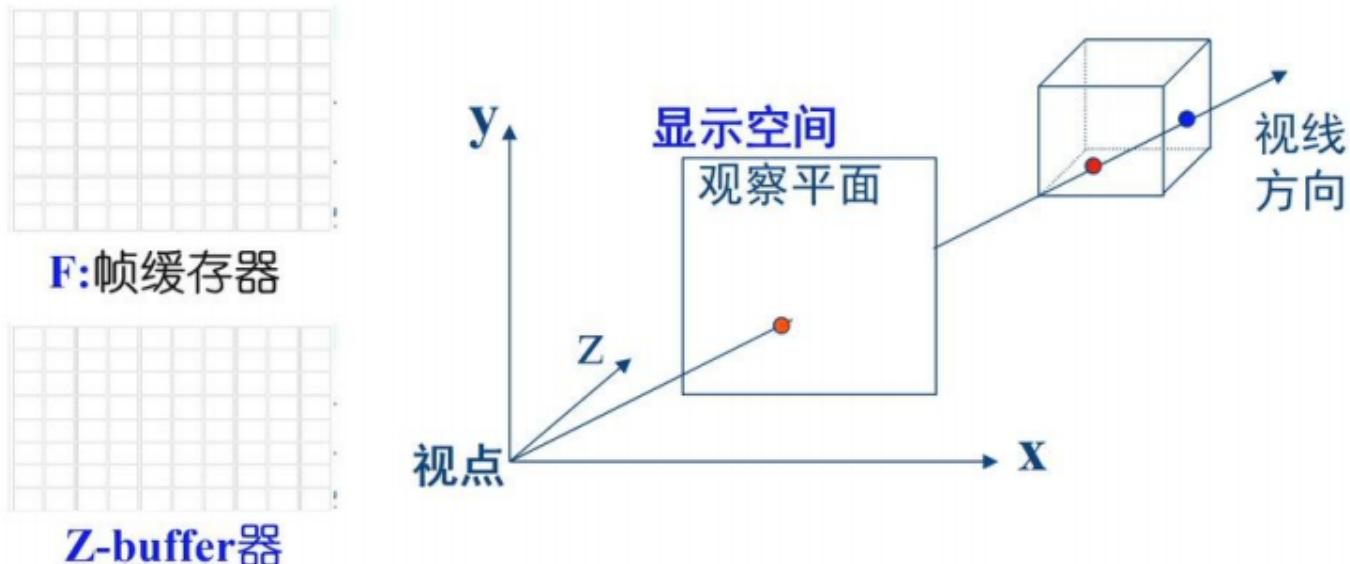
真实感绘制的要求，消除被遮挡的不可见线面，在二维显示设备把三维物体表面的当前可见表面绘制出来。

Z-Buffer

最简单的可见面判定算法

帧缓存F：存储每个像素的颜色值

深度值缓存Z-Buffer：存储像素对应点的最小深度值



```

void zBuffer(){
    int x, y; // 投影平面坐标
    for(y = 0; y < YMAX; y++)
        for(x = 0; x < XMAX; x++){
            WritePixel(x, y, BACKGROUND_VALUE);
            WriteZ(x, y, ZMAX);
        }
    // 绘制背景, 将背景置于最底层
    for(每个多边形)
        for(投影中的每个像素){
            double pz = (x, y)处的深度z值
            if(pz < ReadZ(x, y)){
                WriteZ(x, y, pz);
                WritePixel(x, y, (x, y)处的颜色值)
            }
        }
    // 记录最小深度
}

```

优点：易于硬件实现

缺点：

1. 需要较大容量的Z-Buffer
2. 每个像素对应多边形区域的深度计算
3. 实现反走样、透明和半透明效果困难（因为是在投影面上操作的，并且只记录了浅层的值）

不要求图形一定是多边形组成的，只要确定对象投影每个点的深度和像素值就可以绘制。

光线投影算法

最自然的消隐算法

基本思想：考察由视点出发穿过观察平面的一像素而射入场景的一道射线，则可确定场景中与该射线相交的物体（模拟一道光照）

```

for(屏幕的每一像素){
    形成通过屏幕像素(u, v)的射线;
    for(场景中的每个物体){
        将射线与该物体求交;
        if(存在交点)
            以最近交点所属颜色显示像素(u, v);
        else 以背景色显示像素
    }
}

```

深度排序算法（画家算法）

最偷懒的消隐算法

原理：把表示三维实体的每一个多边形按距离视点由远及近排序，构成一张深度优先表，按照顺序逐个绘制各个多边形。

总体评价：

1. 多边形面片数量很大时，画家消隐算法最快（因为GPU结构？）
2. 不能处理多边形循环遮挡、相互穿透
3. 解决办法：分割多边形面片

多边形网格裁剪

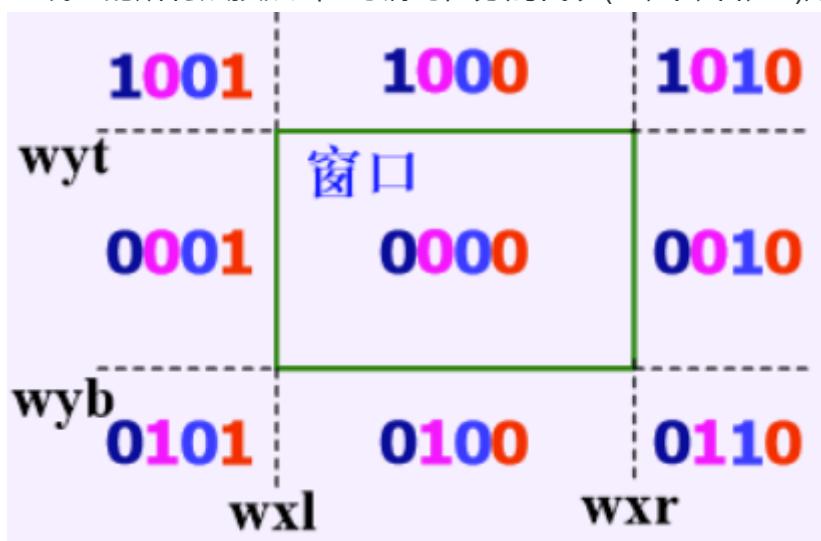
显示设备视窗大小固定，显示实体范围有限，可避免不必要的无效计算，节省计算资源。

直线段裁剪

Cohen-Sutherland算法

端点区位码判定

直线上的所有点按照4位码编码，分别代表(上, 下, 右, 左)是否出界，例如右上角的点编码就是1010



算法流程

1. 找出直线段端点的编码code1, code2
2. $code1 | code2 == 0$ (二者都是0000)，线段完全位于窗口内； $code1 & code2 != 0$ ，线段完全位于窗口外
3. 不属于2的情况，线段部分在窗口内，取线段的一个端点p
4. 从p编码的低位向高位找的第一个1，确定一个相交的边界，求出线段与该边界的交点，用该交点代替p作为端点，外围的剪裁掉

5. 重复4直到没有窗口外的点

梁友栋-Barsky算法

参数推理判定的线段剪裁算法

利用直线的参数方程

直线的参数方程：

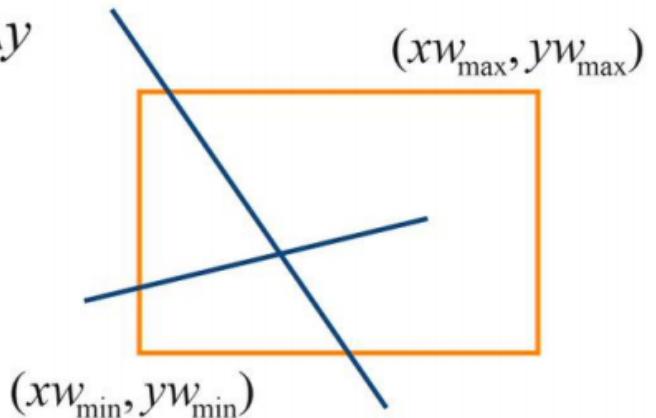
$$x = x_1 + u(x_2 - x_1) = x_1 + u\Delta x$$

$$y = y_1 + u(y_2 - y_1) = y_1 + u\Delta y$$

裁剪窗口内的线段点满足：

$$xw_{\min} \leq x_1 + u\Delta x \leq xw_{\max}$$

$$yw_{\min} \leq y_1 + u\Delta y \leq yw_{\max}$$



选择一个端点作为起始点P0，另一个端点作为终止点P1，二者相减得到 $\Delta x, \Delta y$

把上图中的式子转化一下可以得到如下形式的式子

$$up_k \leq q_k$$

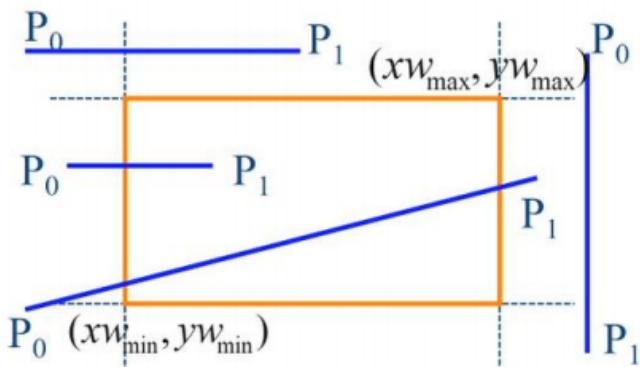
然后就可以利用p和q进行判断

$$u(-\Delta x) \leq x_1 - xw_{\min}$$

$$u(\Delta x) \leq xw_{\max} - x_1$$

$$u(-\Delta y) \leq y_1 - yw_{\min}$$

$$u(\Delta y) \leq yw_{\max} - y_1$$



裁剪判定1：任何平行于裁剪边界的直线

$P_k = 0 \Rightarrow q_k < 0$ 直线在窗口外部，舍弃该线段；
 $q_k \geq 0$ 直线在窗口内部，保留该线段；

裁剪判定2：任何其它直线，有两种情况：

线段从边界延长线外部延伸到内部

线段从边界延长线内部延伸到外部

外—》内 $P_k < 0 \Rightarrow u_1 = \max(0, \frac{q_k}{p_k}, \frac{q_{k'}}{p_{k'}})$

内—》外 $P_k > 0 \Rightarrow u_2 = \min(1, \frac{q_k}{p_k}, \frac{q_{k'}}{p_{k'}})$

注：内外方向的判别根据边界外侧法向量与线段的夹角，大于90°为负（外—内），小于90°为正（内—外）

裁剪判定3：

如果 $u_1 > u_2$, 线段在窗口之外;
否则利用 u_1 和 u_2 计算裁剪线段的两个新端点。

```
if  $u_2 < 1.0$ 
{  $x_2 = x_1 + u_2 \times dx;$ 
   $y_2 = y_1 + u_2 \times dy;$  }
if  $u_1 > 0$ 
{  $x_1 = x_1 + u_1 \times dx;$ 
   $y_1 = y_1 + u_1 \times dy;$  }
```

u 其实就是线段到边界的裁断比例

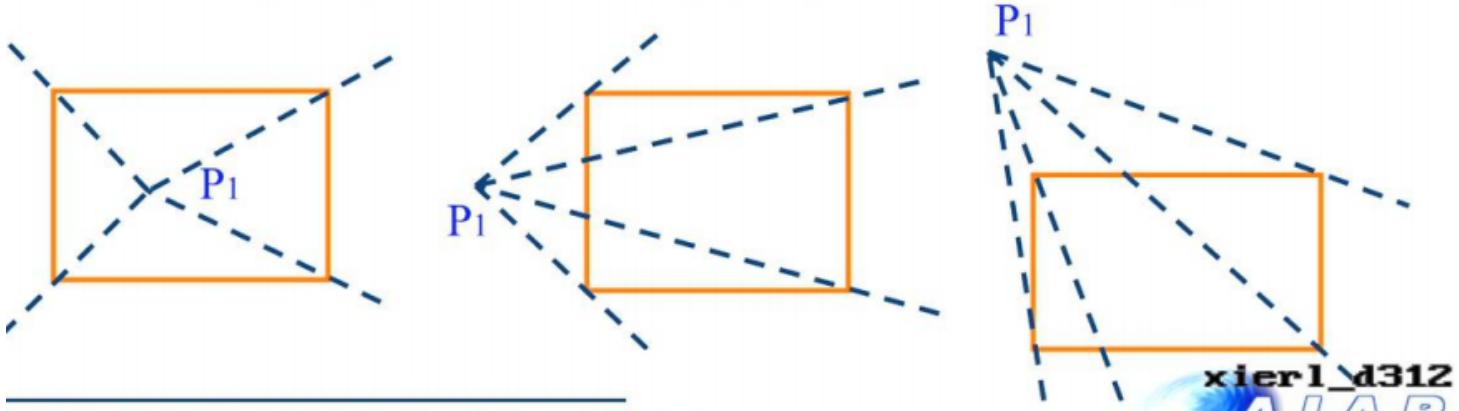
Nicholl-Lee-Nicholl 算法

专门针对二维剪裁

分情况大势判定的算法，比较运算和除法运算执行次数远少于前两种

基本思想：

- 1、确定直线段的端点 P_1 所在区域；
- 2、根据 P_1 的位置组合判定端点 P_2 所在的区域；
- 3、根据端点 P_1 、 P_2 的位置确定对直线的裁剪操作；

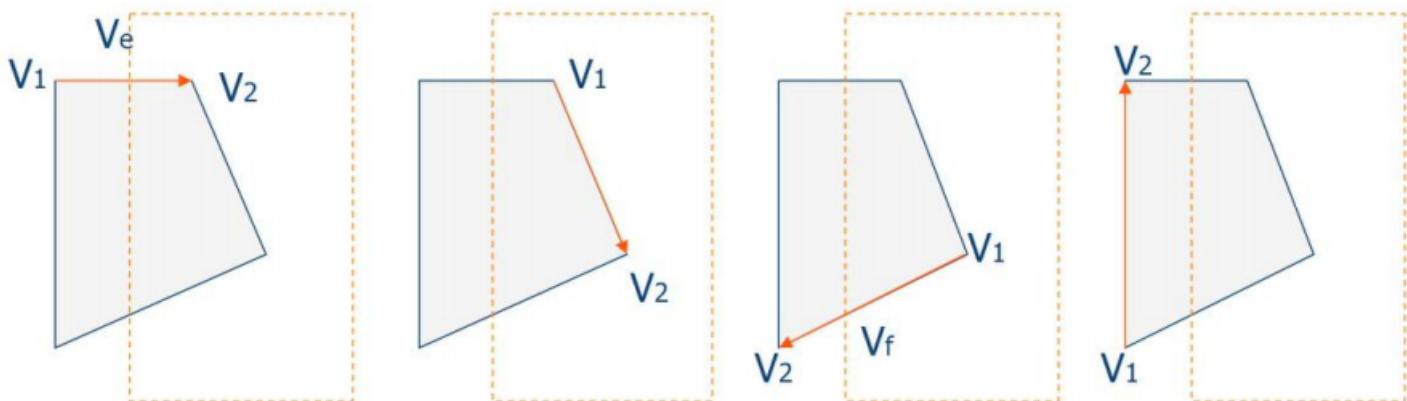
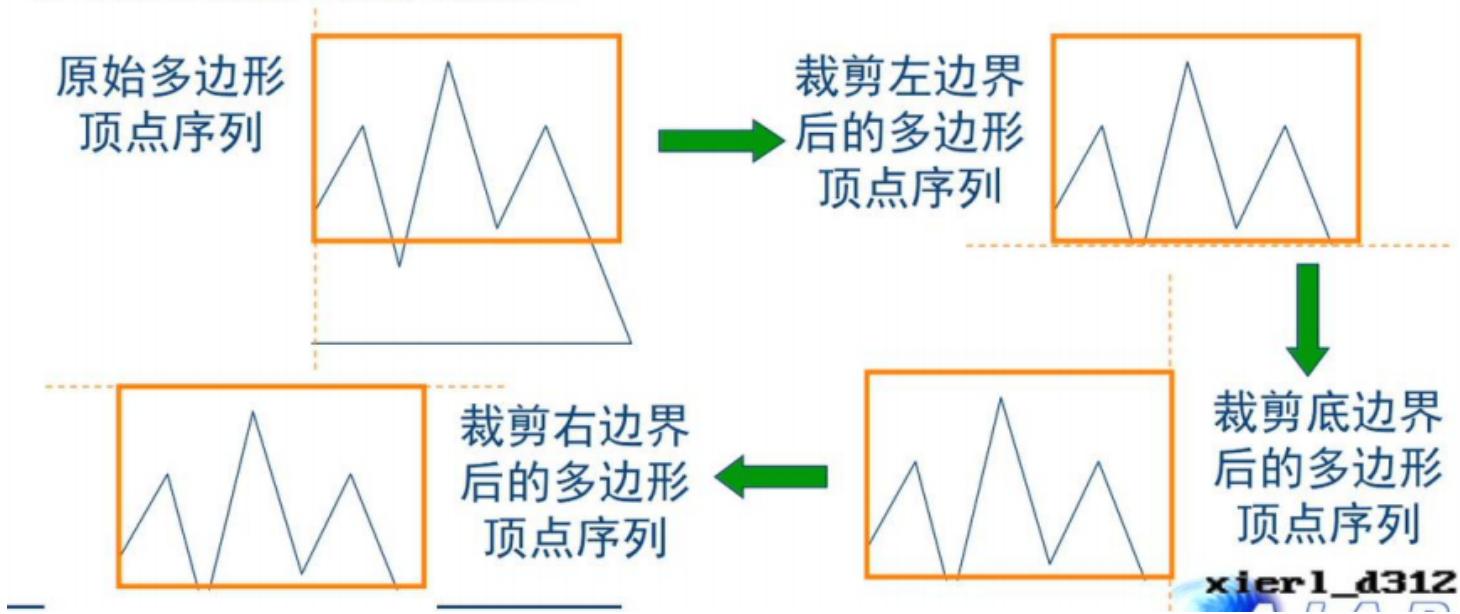


二维多边形裁剪

Sutherland-Hodgeman剪裁

将多边形看作整体，对所有多边形顶点进行处理（把裁剪过后的所有相邻顶点都连起来）

多边形顶点序列裁剪流程：



外→内:

Input: V_1, V_2

Output: V_e, V_2

内→内:

Input: V_1, V_2

Output: V_1, V_2

内→外:

Input: V_1, V_2

Output: V_f

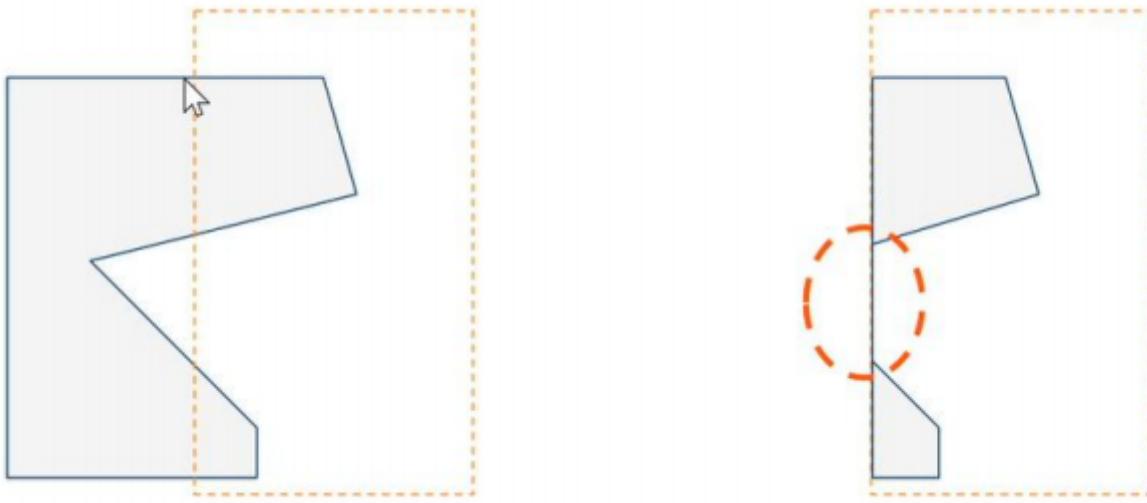
外→外:

Input: V_1, V_2

Output:

问题：

剪裁凹多边形时可能出现多余的边



解决策略：将凹多边形转成凸多边形

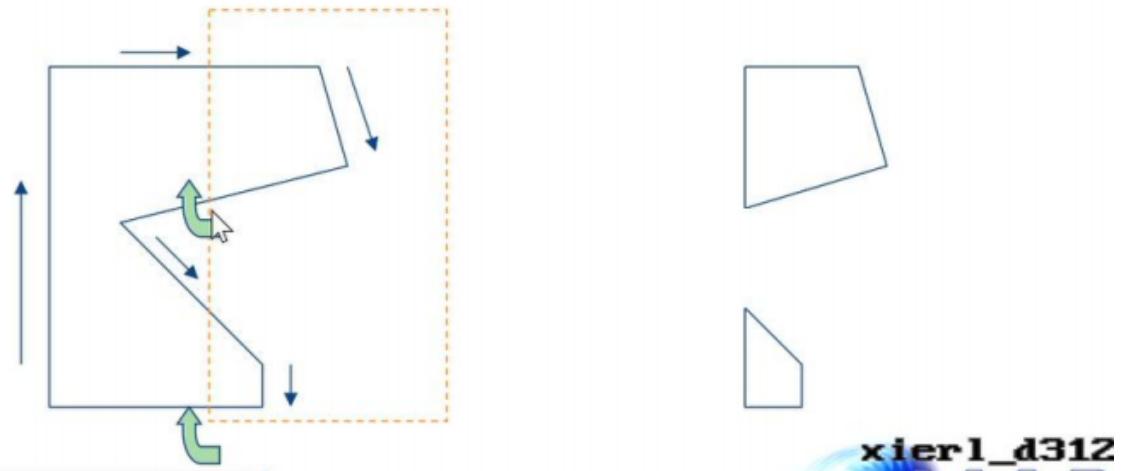
Weiler-Atherton算法不会有这个问题

Weiler-Atherton算法

按照一定的方向处理顶点

根据多边形边的方向，决定沿多边形某一边的方向处理顶点、或者沿窗口的边界方向处理顶点。对顺时针处理顶点，采用如下规则：

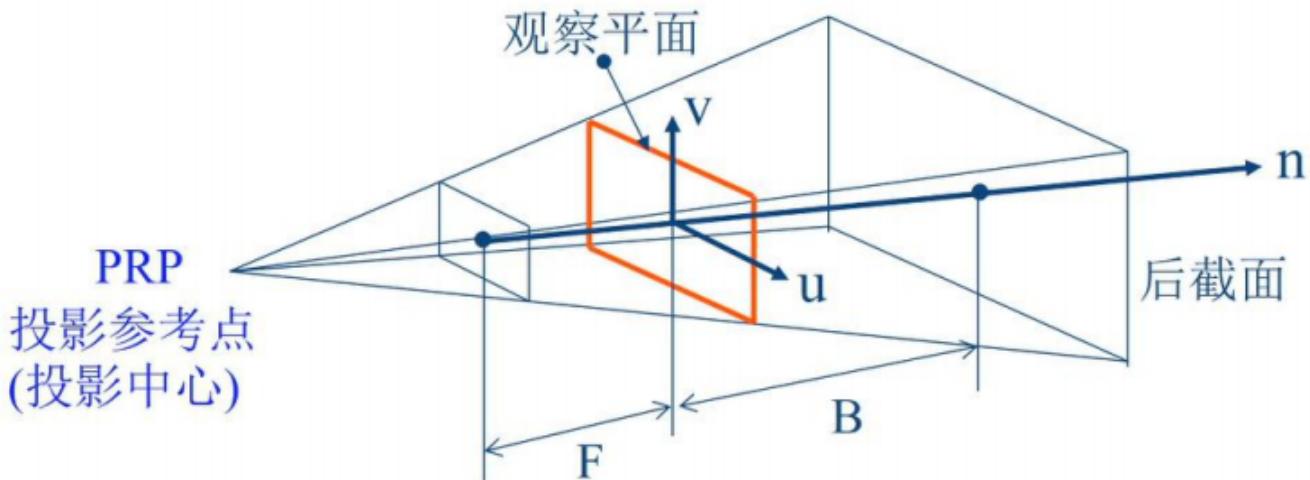
1. 由外到内的顶点对，沿多边形边界的方向；
2. 由内到外的顶点对，按顺时针沿着窗口边界的方向；



绿色箭头所示的两条边是从内向外的，所以沿着边界顺时针的方向去连，把顶点连到各自的区域去，而不会出现多余的边。

三维视见体与线段裁剪

- 三维视见体—由投影中心和观察窗口的四个角点的连线及其延长线构成的无底四棱锥(透视投影)或无限四棱柱(平行投影)。



二维扩展到三维

将Cohen-Sutherland的端点区位码方法扩展一下变成6位，即
(后, 前, 上, 下, 右, 左)

5 光栅化计算

光栅显示器看作一个pixels矩阵，每个pixel对应一个显示色彩亮度的物理小区域，每个小区域显示一致的色彩亮度，显示屏的可用pixel有限。

光栅化：确定显示屏最佳逼近图形的pixels集合，并指定pixel属性的过程

像素扫描变换：图元到光栅显示器的pixel中心采样过程

数学图元→有限的像素

直线扫描转换算法

无限→有限

数学直线无宽度、无限个点

直线光栅化：在有限可用像素中，确定最佳逼近直线的一组像素

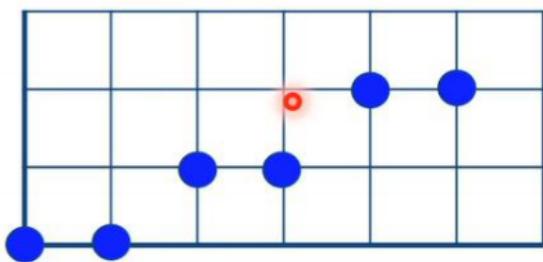
基本增量算法 (数值微分DDA法)

设直线段的端点是： $P_0(x_0, y_0), P_1(x_1, y_1)$

斜率是： $k = \frac{y_1 - y_0}{x_1 - x_0}$

直线方程： $y_{i+1} = y_i + k\Delta x$

当 $\Delta x=1$ 直线上象素 (x_i, y_i) 的下一象素坐标是



$(x_i + 1, Round(y_i + k))$

直接采样pixel，绘制线宽为pixel的直线（涉及四舍五入，最近邻插值？）

```
void DDALine(int x0, int y0, int x1, int y1, int value){  
    int x;  
    double dx = x1 - x0;  
    double dy = y1 - y0;  
    double k = dy / dx;  
    double y = y0;  
    for(x = x0; x < x1; x++){  
        WritePixel(x, Round(y), value);  
        y = y + k;  
    }  
}
```

问题：

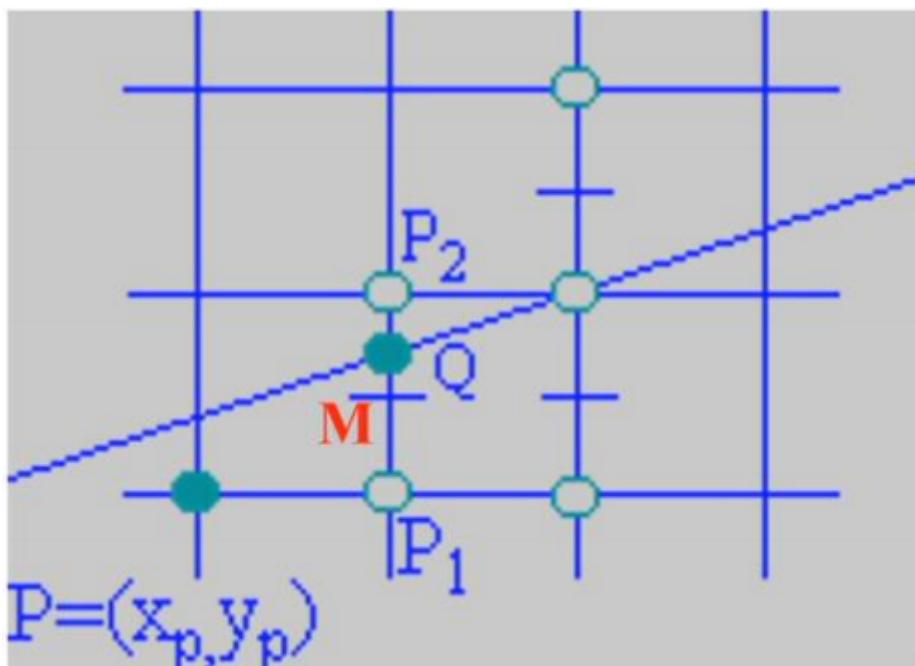
1. 仅适用于 $|k| \leq 1$ 的情况， $|k| > 1$ 时需要颠倒x, y（不然就不是连续采样了）
2. x, y, k用浮点数表示，且需四舍五入取整，不利于硬件实现（浮点数转整型）

直线扫描转换的中点线算法

上一个算法中涉及浮点运算。此算法的目标是只使用整型运算，根据当前点计算下一个像素点。

中点方法：两个光栅像素点P1, P2，中点M，考察M与直线的相对位置，直线在M上方则选择P2，否则选择P1

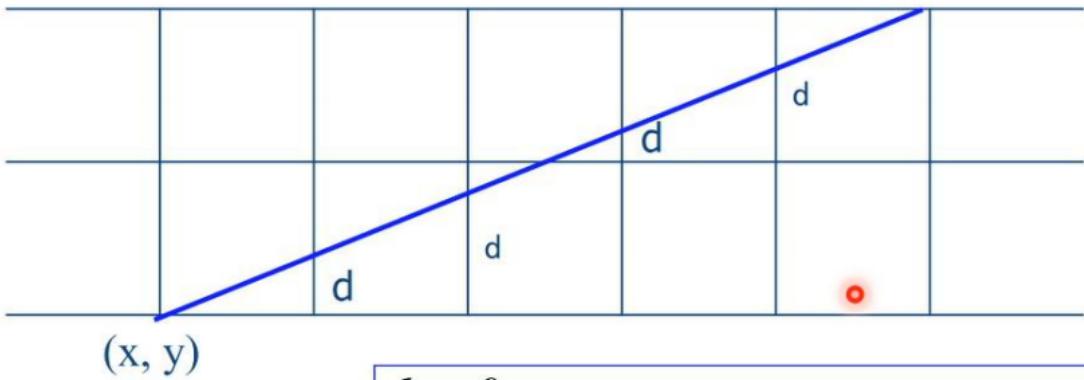
(以中点作为参考点，这样就不用四舍五入了)



```
void MidpointLine(int x0, int y0, int x1, int y1, int value){  
    int x;  
    int dx = x1 - x0;  
    int dy = y1 - y0;  
    int d = 2 * dy - dx; // d>0则高于中点，反之低于  
    int incrP1 = 2 * dy;  
    int incrP2 = 2 * (dy - dx);  
    int x = x0; int y = y0;  
    WritePixel(x, y, value);  
    while(x < x1){  
        if(d > 0){  
            x++;  
            y++;  
            d += incrP2;  
        }  
        else{  
            x++;  
            d += incrP1;  
        }  
        WritePixel(x, y, value);  
    }  
}
```

Bresenham直线转换算法

改变了中点线算法选择pixel的判据（判定条件）



(x, y)在直线上的下一个象素位置($x + 1, y$)或($x + 1, y + 1$)

决定要素: d

$d = 0;$
 $k = dy/dx; \quad d = d + k;$
 $\text{if } d > 1 \text{ then } d = d - 1;$
 $\text{if } d \geq 0.5 \text{ then } (x+1, y+1);$
 $\text{if } d < 0.5 \text{ then } (x+1, y);$
 $e = d - 0.5$

将 d 维持在[0, 1]的范围内不断累加, d 就代表了在栅格内的相对位置, 大于0.5则在中点之上, 小于0.5则在中点之下。

转换一下: 用 $e = d - 0.5$ 的正负来判断。

```
void Bresenham_Line(int x0, int y0, int x1, int y1, int value){
    int x;
    int dx = x1 - x0;
    int dy = y1 - y0;
    double k = dy / dx;
    double e = -0.5;
    int x = x0;
    int y = y0;
    while(x < x1){
        WritePixel(x, y, value);
        x++; e += k;
        if(e > 0){
            y++;
            e--;
        }
    }
}
```

$$e = e_0 + k = -0.5 + dy/dx \quad 2e \bullet dx = -dx + 2dy \quad e' = -dx + 2dy;$$

xier1_d312

$$\text{if } e > 0 \text{ then } e = e - 1 \rightarrow e' = e' - 2dx$$

将 $2e \bullet dx$ 替换为 e' , 用这种方法可以用整型表示 e , 不再需要浮点, 直接代入 dx , dy 进行计算, 也不需要斜率 k 。

```
void Bresenham_Line(int x0, int y0, int x1, int y1, int value){
    int x;
    int dx = x1 - x0;
    int dy = y1 - y0;
    int e = -dx;
    int x = x0;
    int y = y0;
    while(x < x1){
        WritePixel(x, y, value);
        x++; e += 2 * dy;
        if(e > 0){
            y++;
            e -= 2 * dx;
        }
    }
}
```

三角形的像素扫描转换