消隐算法—图像空间算法

冯结青

浙江大学 CAD&CG国家重点实验室

主要内容

- 消隐的基本概念
- 线消隐算法
- z缓冲器(*z-buffer*)算法
- 扫描线z缓冲器算法

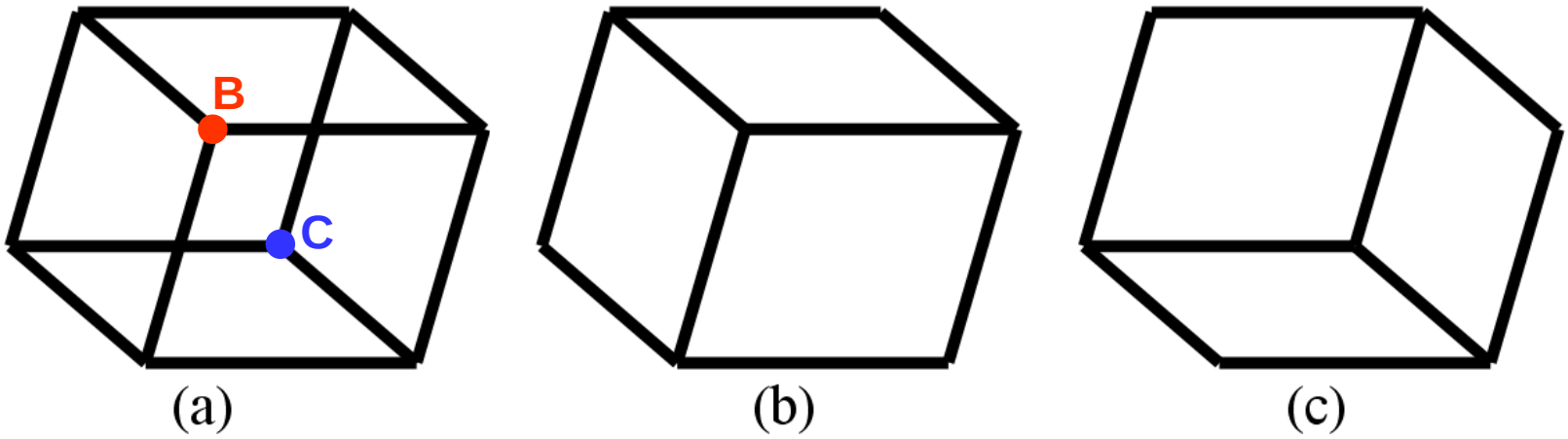
主要内容

- 消隐的基本概念
- 线消隐算法
- z缓冲器(*z-buffer*)算法
- 扫描线z缓冲器算法

消隐的基本概念

- 消隐 (隐藏线或面消除)：相对于观察者，确定场景中物体是**可见的**、**部分可见的**、**不可见的**
- 消隐可以增加图形的真实感
  - 投影：三维空间→二维平面，损失部分信息
  - 消隐：确定物体前后关系获得更多信息
- 消隐是图形学中非常重要的一个**基本问题**

消隐的基本概念



没有消隐的图形具有二义性：(a) 立方体的线框图；(b) 顶点B离视点最近时的消隐；(c) 顶点C离视点最近时的消隐

消隐的基本概念

- 针对消隐问题的复杂性，提出了许多适合不同应用的精巧算法
  - 实时绘制：要求消隐算法速度快，通常生成的图形质量一般
  - 真实感图形：要生成高质量的图形，通常消隐算法速度较慢
- 消隐算法的权衡：消隐效率、图形质量

消隐与排序

- **消隐的本质是排序**：判断场景中的物体全部或者部分与视点之间的远近
  - 离视点近的物体可能遮挡离视点远的物体
  - 离视点远的物体不可能遮挡离视点近的物体
- 结论：消隐算法的效率很大程度上取决于排序的效率

消隐与连贯性 (1)

- 消隐中的连贯性：场景中物体或其投影，在离视点距离所表现出的相似程度
  - 物体的连贯性：分离测试
  - 面的连贯性：曲面的属性发生连续变化
  - 边的连贯性：只有边从后面穿过可见边或面时，边的可见性会发生改变
  - 隐含边的连贯性：两个平面间的交线上两点可见性，可以确定交线的可见性

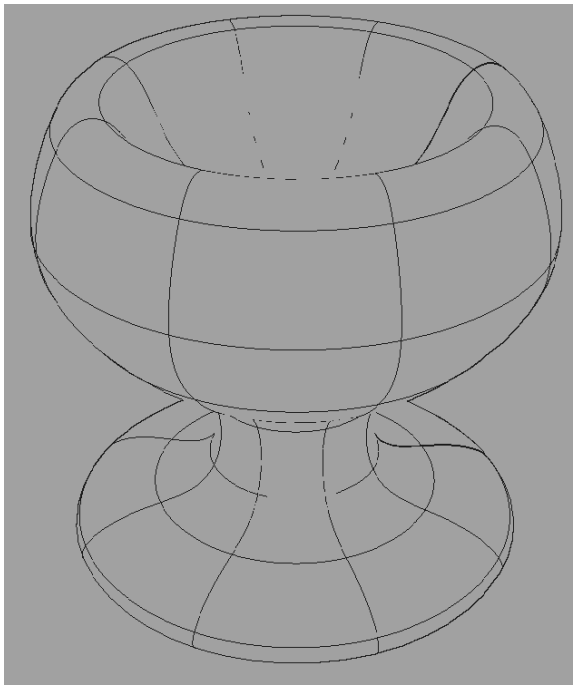
消隐与连贯性 (2)

- 扫描线之间的连贯性：物体在相邻扫描线上的可见跨度(深度值)
- 区域连贯性：一组像素被同一个物体遮挡
- 深度连贯性：深度增量计算
- 帧间的连贯性：动态图形中

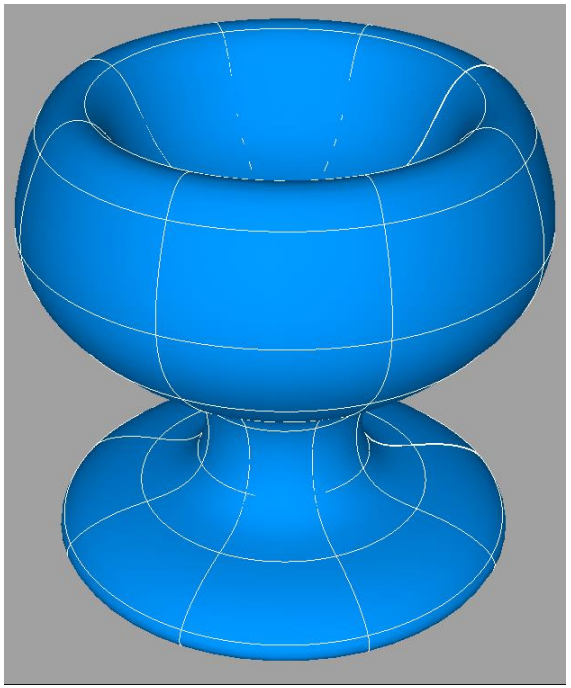
- 结论：充分地利用各种连贯性是提高消隐算法效率的一个重要因素

消隐的分类—对象与输出

- 根据消隐对象和输出结果



线消隐：输出线框图



面消隐：输出着色图

消隐的分类：实现的坐标空间

- 算法实现时所在的坐标系(空间)进行分类：
  - 图像空间消隐
  - 景物空间消隐

图像空间消隐

- 描述

  for(图像中每一个像素) {

  确定由投影点与像素连线穿过的距离观察点最近的物体；

  用适当的颜色绘制该像素；

  }

- 特点：在屏幕坐标系中进行的，生成的图像一般受限于显示器的分辨率
- 算法复杂度为$O(nN)$：场景中每一个物体要和屏幕中每一个像素进行排序比较，n为物体个数，N为像素个数
- 代表方法：z缓冲器算法，扫描线算法等

景物空间消隐

- 描述

  for(世界坐标系中的每一个物体) {

  确定未被遮挡的物体或者部分物体；

  用恰当的颜色绘制出可见部分；

  }

- 特点：算法精度高，与显示器的分辨率无关，适合于精密的CAD工程领域
- 算法复杂度为O(n^2)：场景中每一个物体都要和场景中其它的物体进行排序比较，n为物体个数
- 代表方法：背面剔除、表优先级算法等

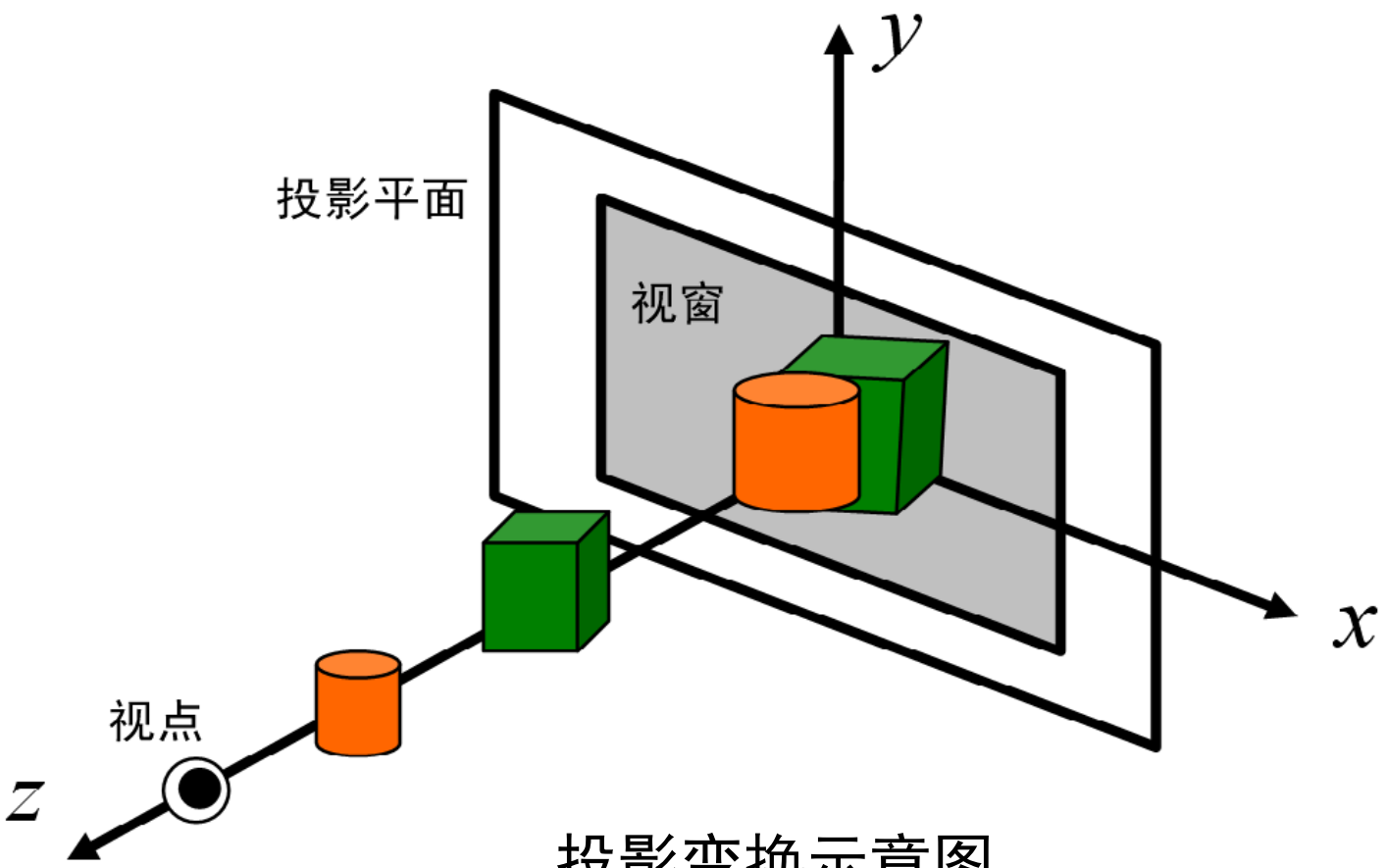
景物和图形空间消隐方法的比较

- 理论上
  - 如果 n(物体数) < N(像素数)，则景物空间算法的计算量$O(n^2)$小于图像空间算法$O(nN)$
- 实际应用中通常会考虑画面的连贯性，所以图像空间算法的效率有可能更高
- 景物空间和图像空间的混合消隐算法

消隐算法的几个假设

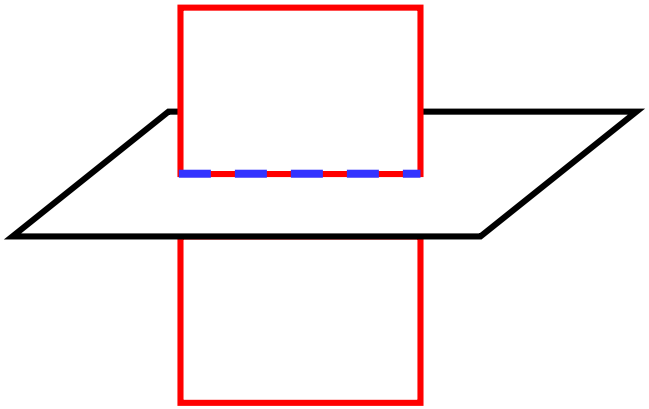
- 图形对象需含有**面信息**
- 投影平面是视点坐标系的oxy平面
- 投影方向为负z轴方向的平行投影
  - z值越大，离视点越近
  - 透视变换可以转化为平行投影
- 有些算法不能处理相互贯穿或循环遮挡的物体，此时应做特殊处理

消隐算法的几个假设

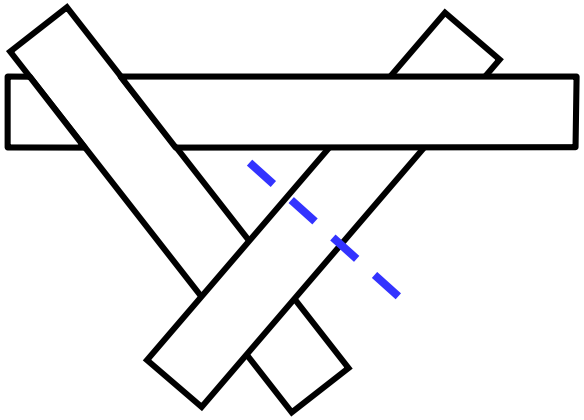


投影变换示意图

消隐算法的几个假设



相互贯穿



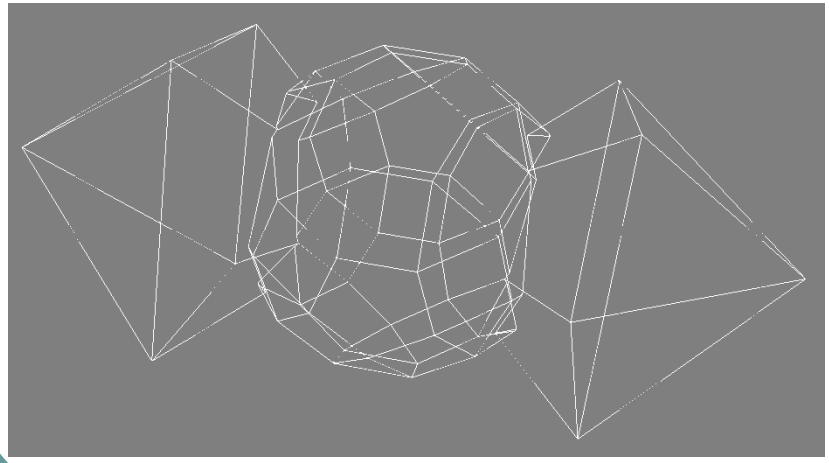
循环遮挡

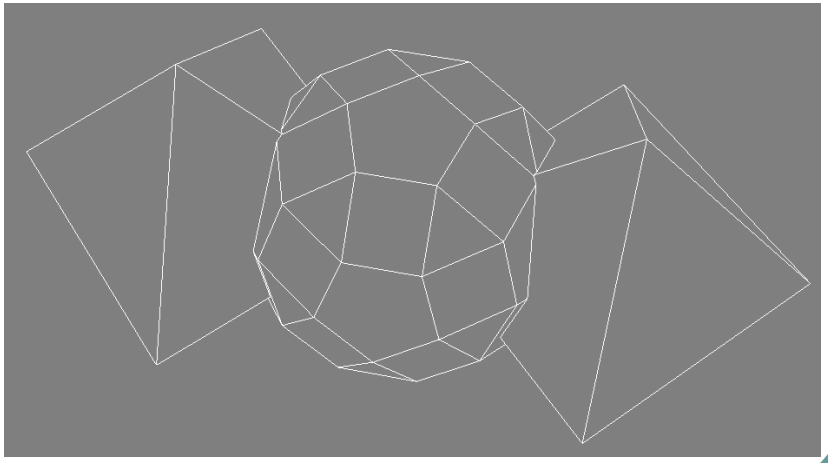
主要内容

- 消隐的基本概念
- 线消隐算法
- z缓冲器(*z-buffer*)算法
- 扫描线z缓冲器算法

线消隐

- 线消隐的简单处理方法：在**景物空间面消隐算法**中
  - 用指定颜色绘制面的边
  - 用背景颜色绘制面

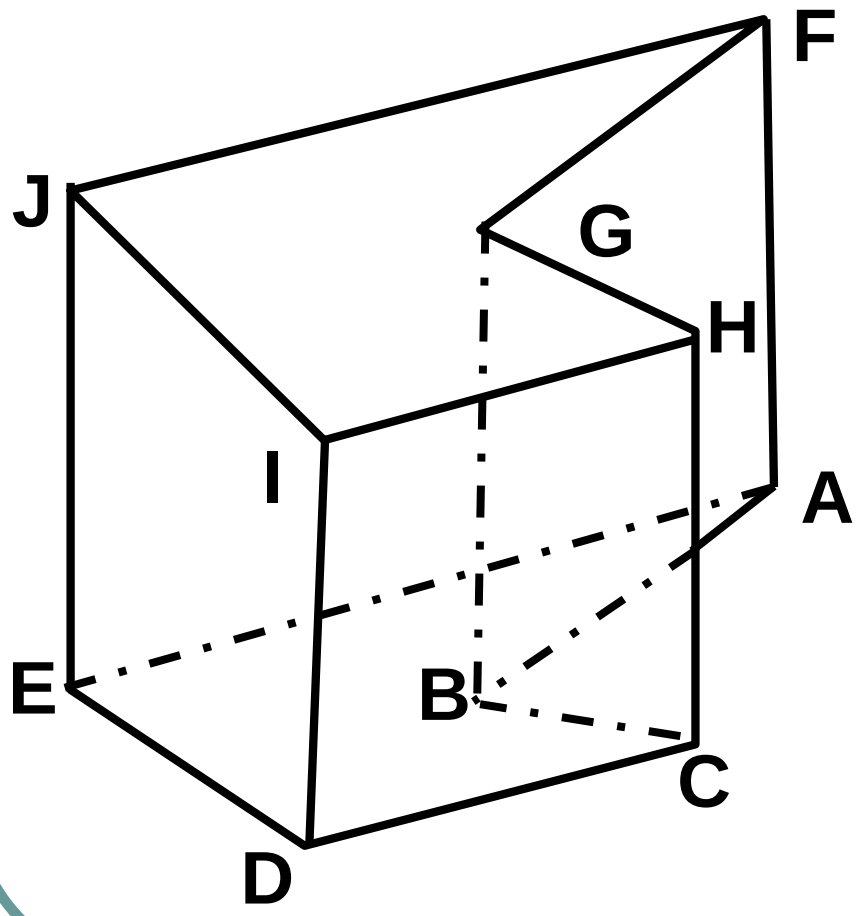




线消隐

- 线消隐的核心：面对线段的遮挡关系
  - 物体对线段的遮挡关系：物体是面的集合，转化为面对线段的遮挡关系
  - 线段与面(三维)、线段与线段(二维)的求交计算

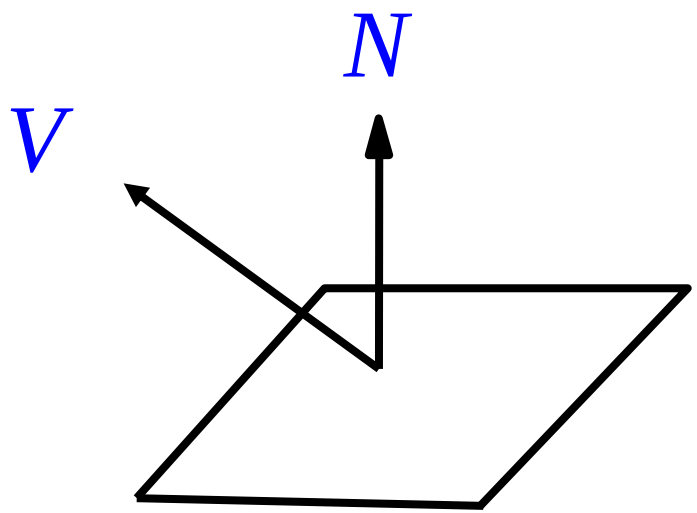
多面体的线消隐



- 算法复杂度
  - 多边形有n条边，N个面
  - 简单的两两求交算法复杂度为

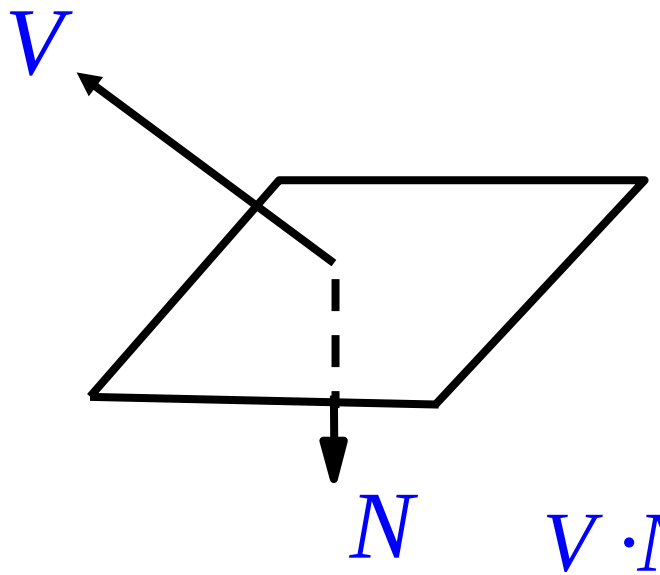
$$O(nN)$$

多面体的线消隐加速



$V \cdot N \geq 0$

前面

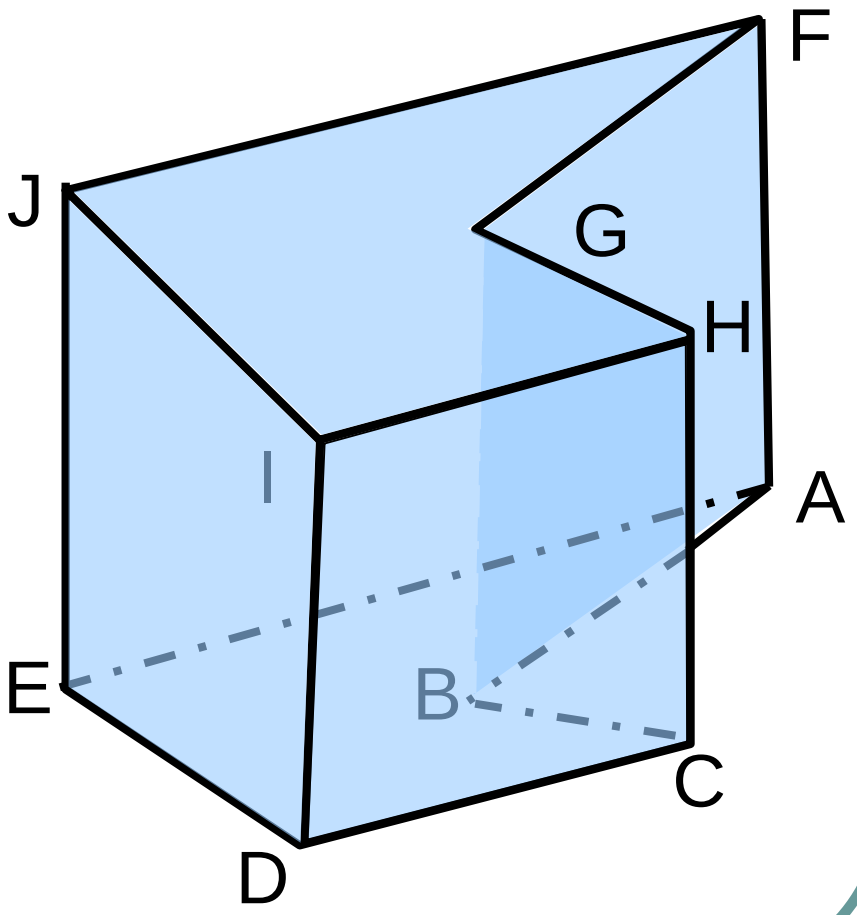


$V \cdot N < 0$

背面

多面体的线消隐加速

- 背面对于视点是不可见的
- 在进行面对线段的遮挡测试时，只考虑前面，而忽略背面!



- 课后阅读：多面体、曲面的线消隐算法

主要内容

- 消隐的基本概念
- 线消隐算法
- z缓冲器(*z-buffer*)算法
- 扫描线z缓冲器算法

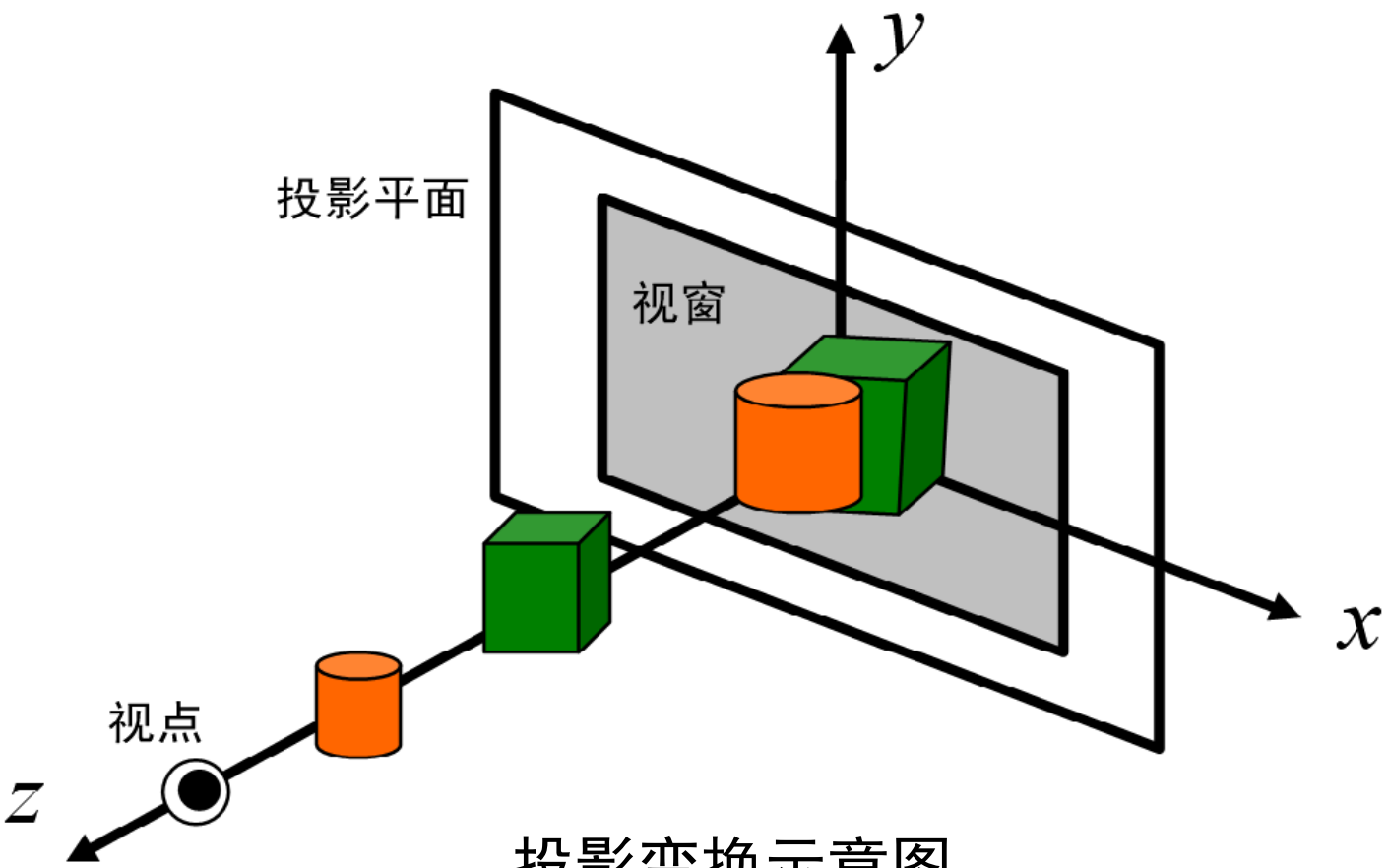
Wolfgang Straßer：TU Berlin, April 26, 1974

Edwin Catmull ：Univ. of Utah， 8 months later

z缓冲器算法

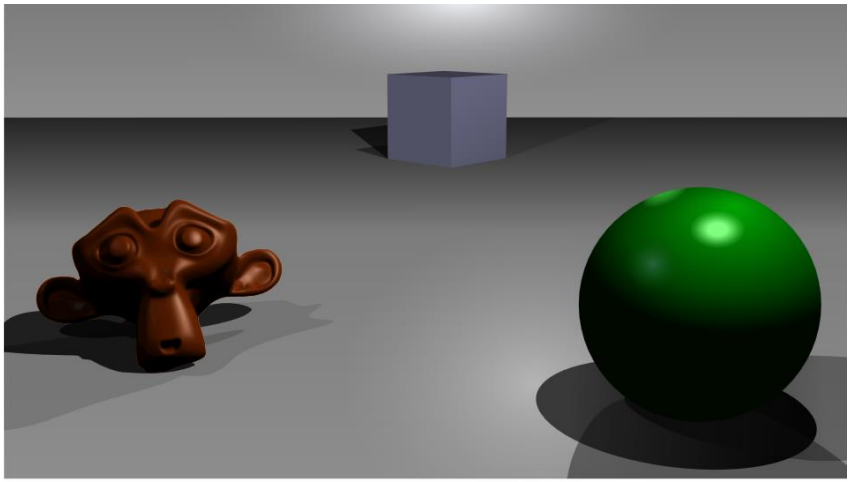
- z (深度)缓冲器算法属于图像空间算法
- z缓冲器是帧缓冲器的推广
  - 帧缓冲器：存储的是像素的颜色属性
  - z缓冲器：存储的是对应像素的z值
    - 假设在视点坐标系($oxyz$)中，投影平面为 $z = 0$，视线方向沿($-z$)轴方向，投影为平行投影
    - 深度值就是物体沿着视线($-z$)方向、与视点的距离
    - 离视点近的物体遮挡离视点远的物体：z值越大，离视点越近

z缓冲器算法



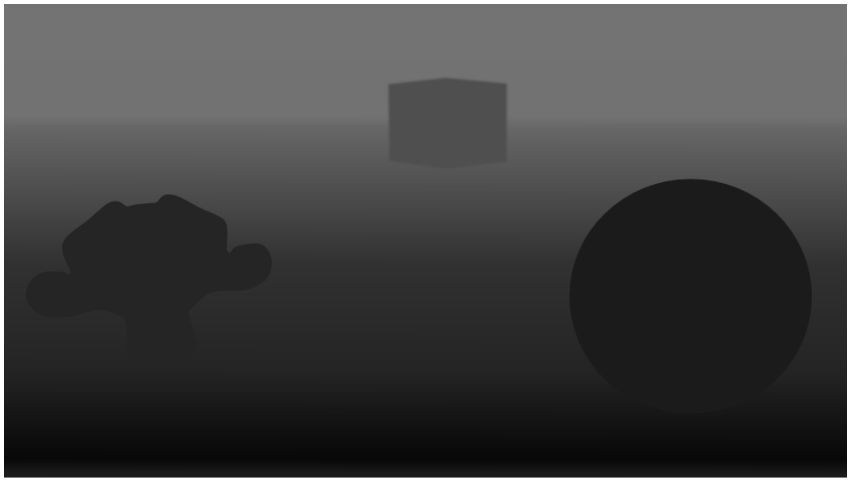
投影变换示意图

颜色与深度缓冲举例



A simple three-dimensional scene

颜色缓冲



Z-buffer representation

深度缓冲

z缓冲器算法描述

(1) 帧缓冲器中的颜色置为背景颜色

(2) z缓冲器中的z值置成最小值(离视点最远)

(3) 以任意顺序扫描各多边形

a) 对于多边形中的每一像素，计算其深度值$z(x,y)$

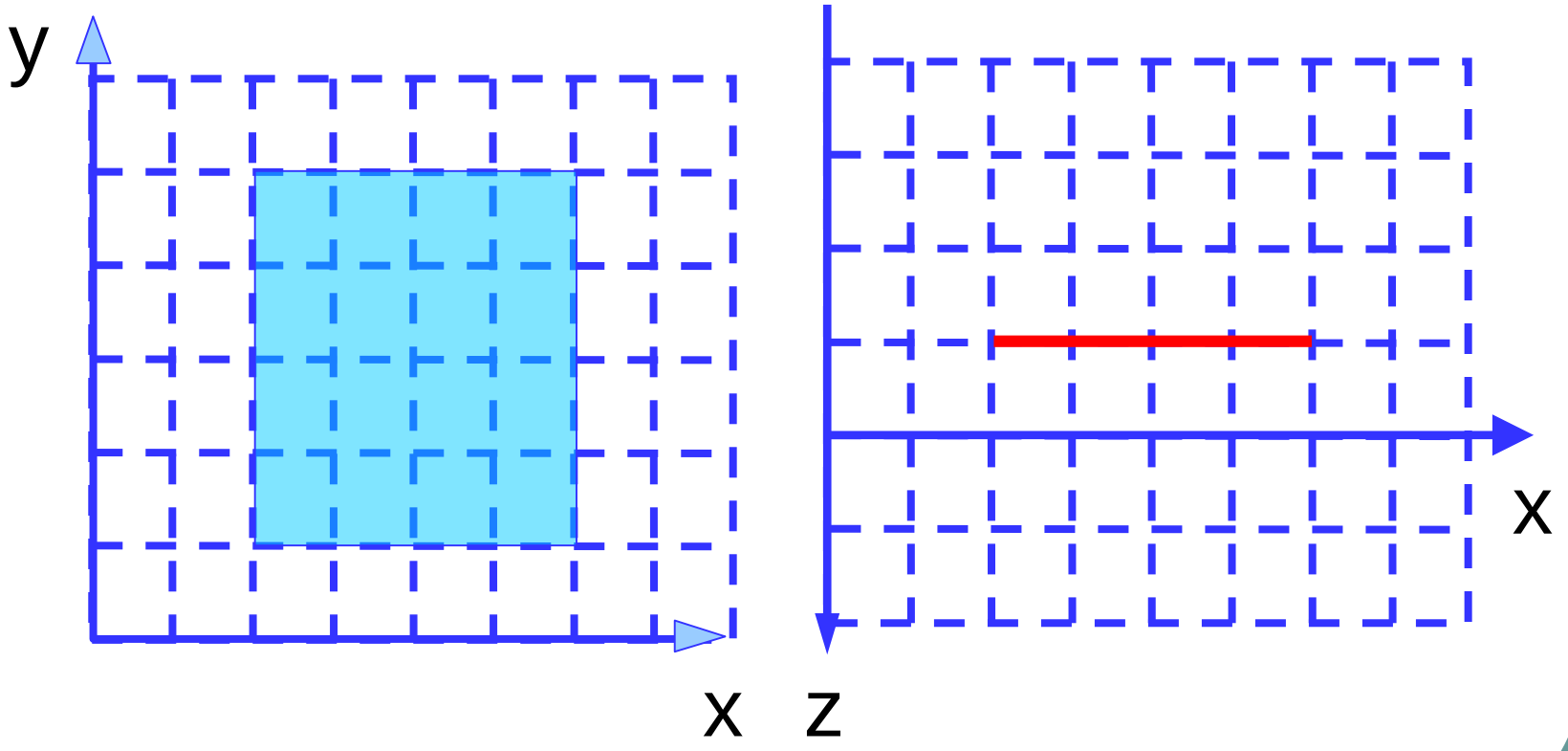
b) 比较$z(x, y)$与z缓冲器中已有的值$zbuffer(x,y)$

如果$z(x, y) > zbuffer(x, y)$，那么

计算该像素(x,y)的光亮值属性并写入帧缓冲器

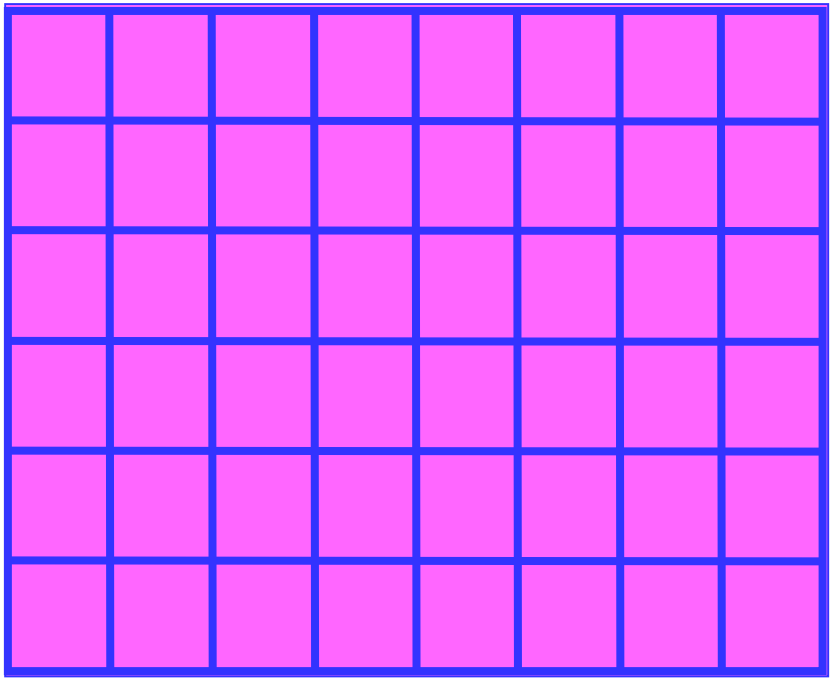
更新z缓冲器 $zbuffer(x, y) = z(x, y)$

z缓冲器算法举例 (1)



z缓冲器算法举例(1)

帧缓冲器

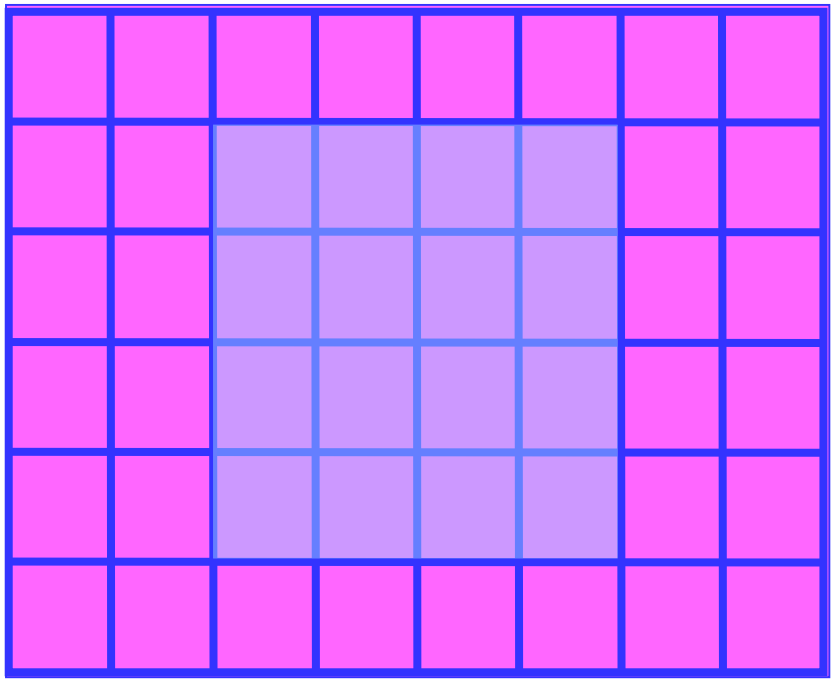


z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

z缓冲器算法举例 (1)

帧缓冲器

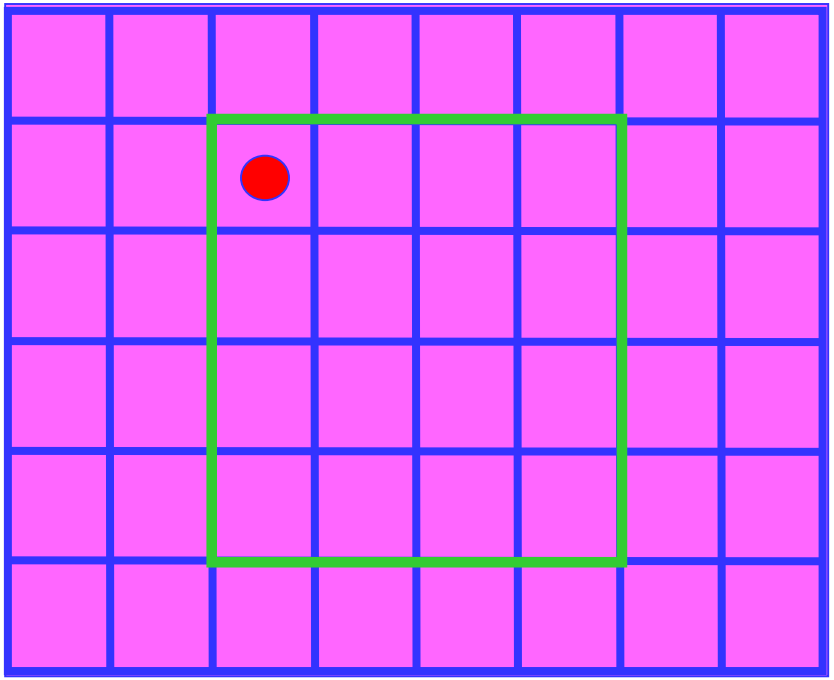


z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

z缓冲器算法举例(1)

帧缓冲器

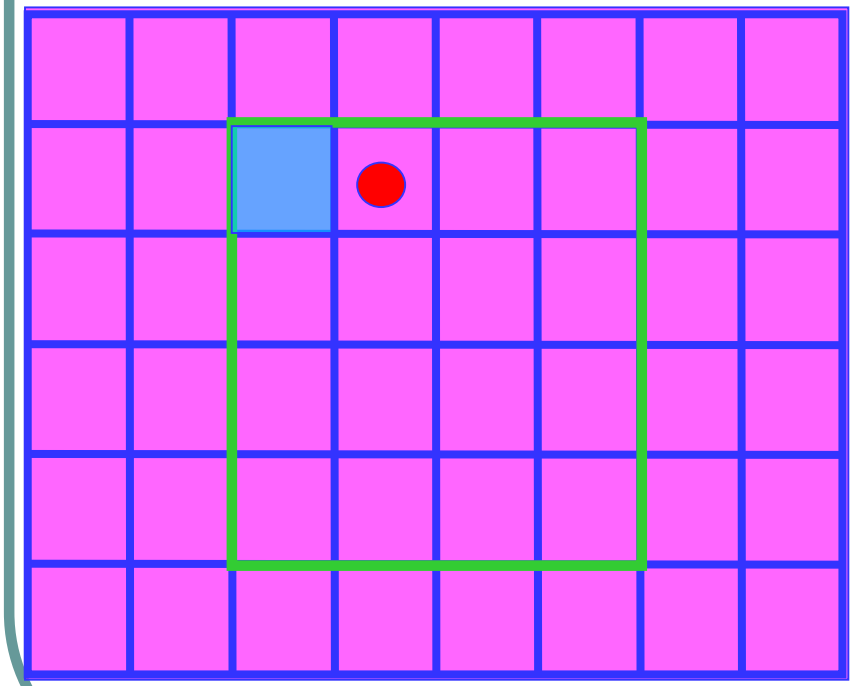


z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

z缓冲器算法举例(1)

帧缓冲器

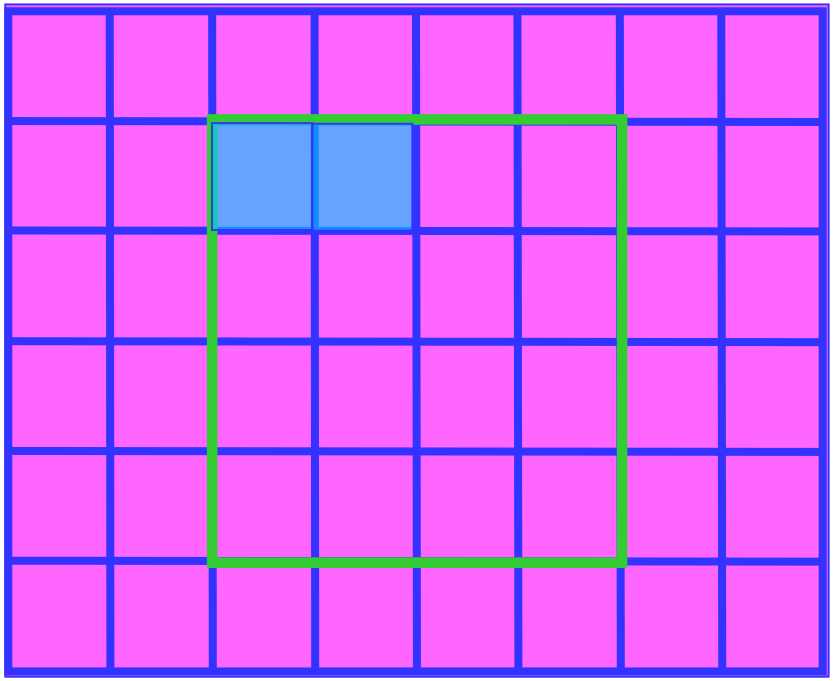


z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

z缓冲器算法举例(1)

帧缓冲器

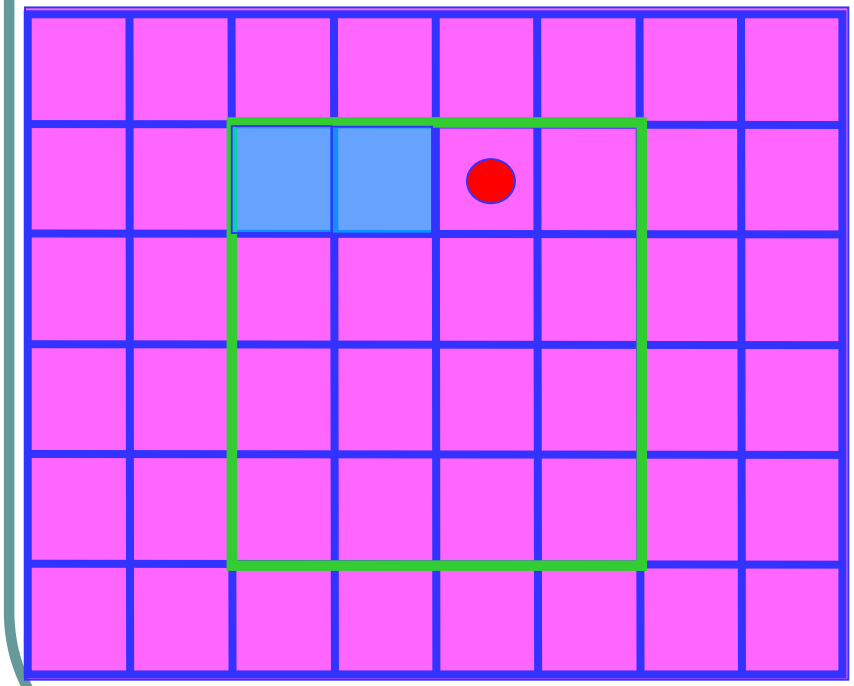


z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

z缓冲器算法举例(1)

帧缓冲器

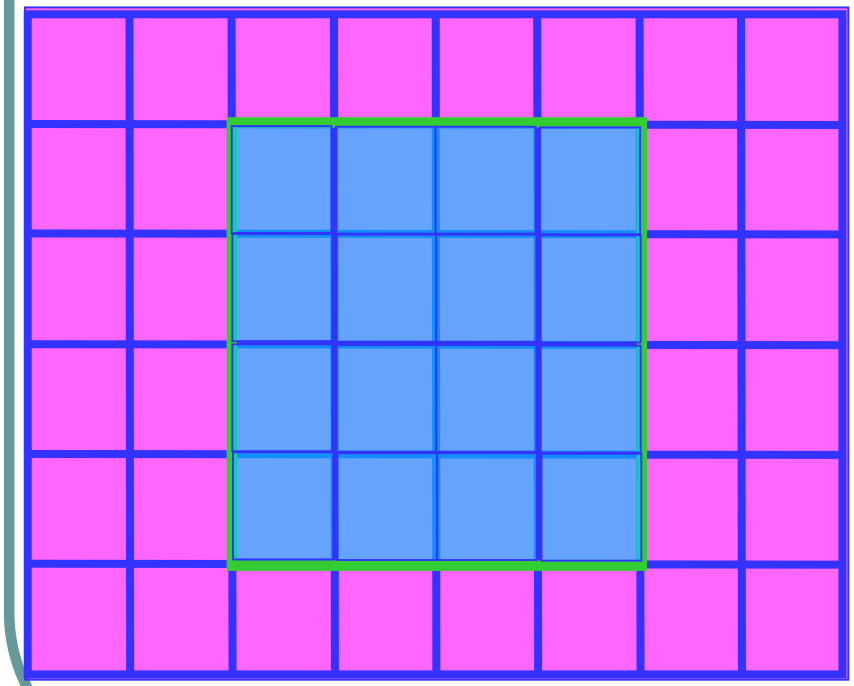


z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

z缓冲器算法举例(1)

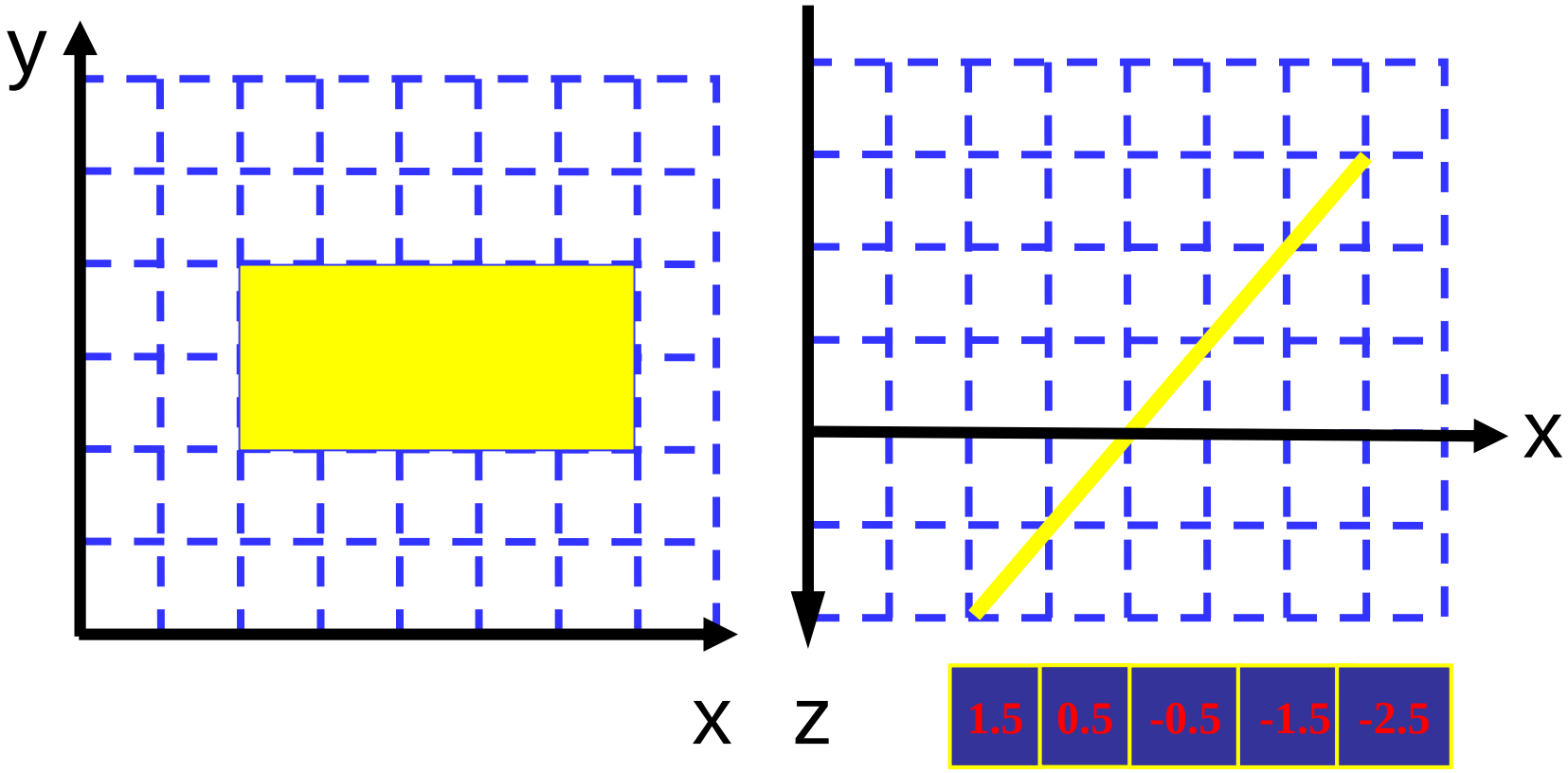
帧缓冲器



z缓冲器

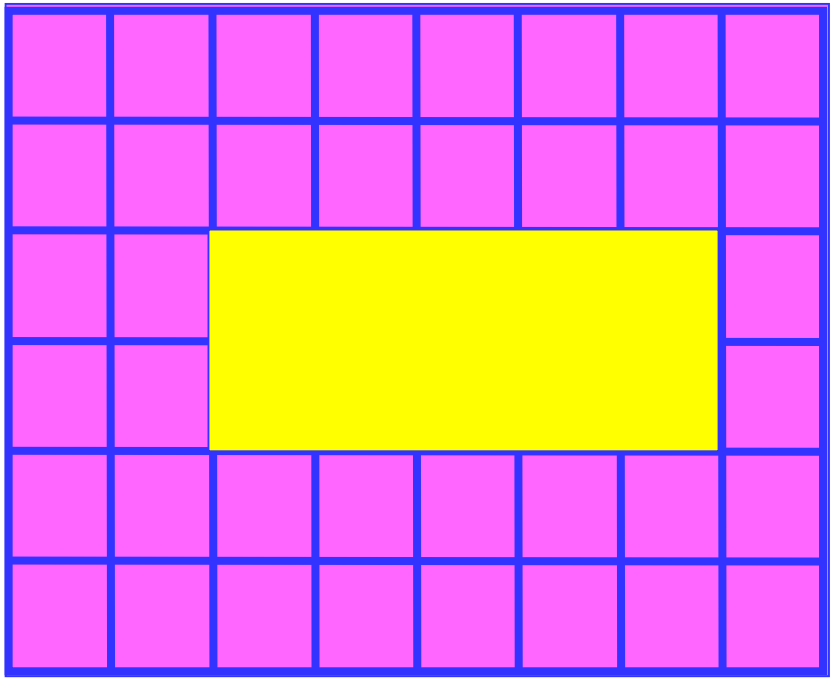
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

z缓冲器算法举例(2)



z缓冲器算法举例(2)

帧缓冲器

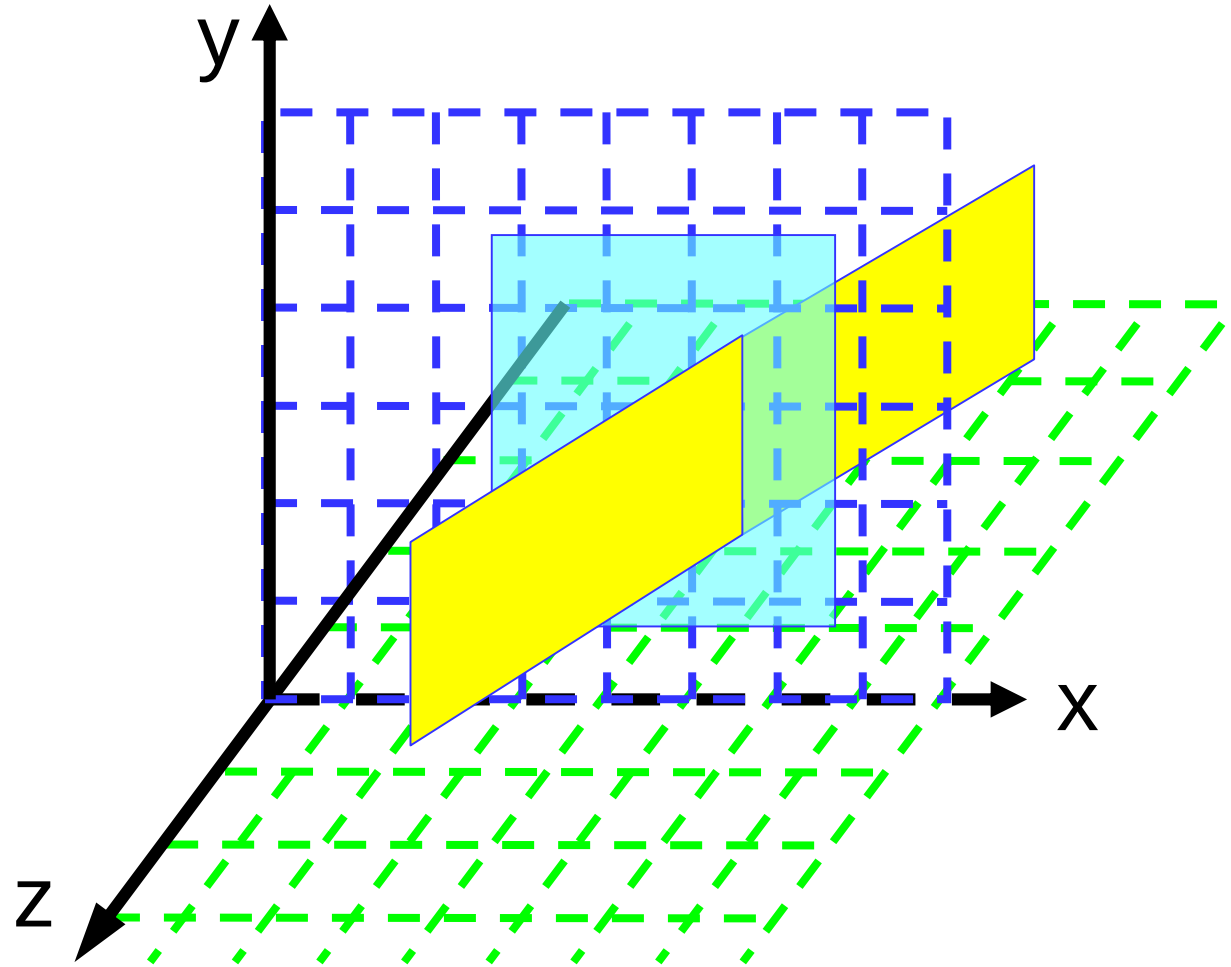


z缓冲器

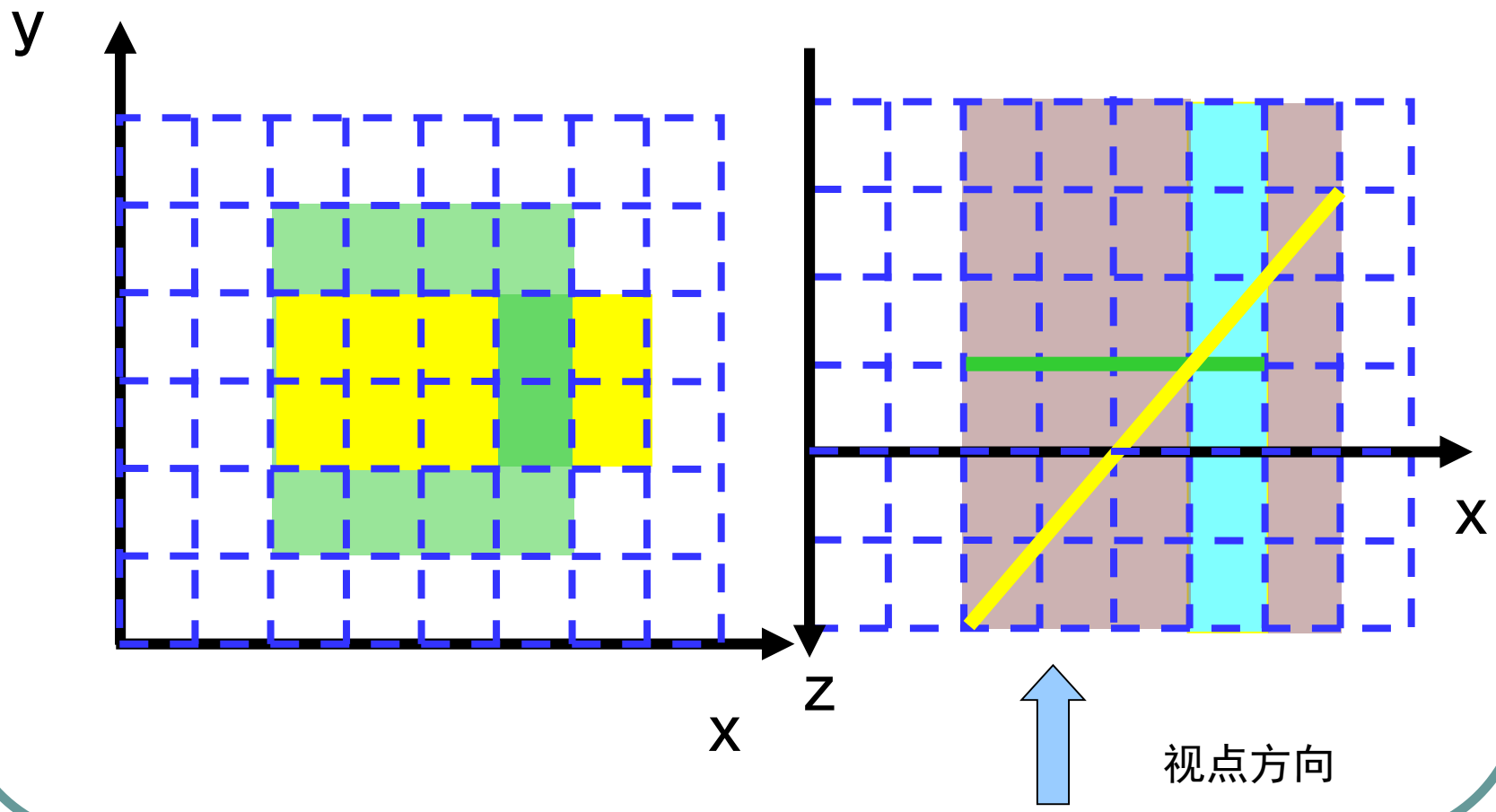
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	2	1	0	-1	-2	-5
-5	-5	2	1	0	-1	-2	-5
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5	0.5	-0.5	-1.5	-2.5

z缓冲器算法举例(2)

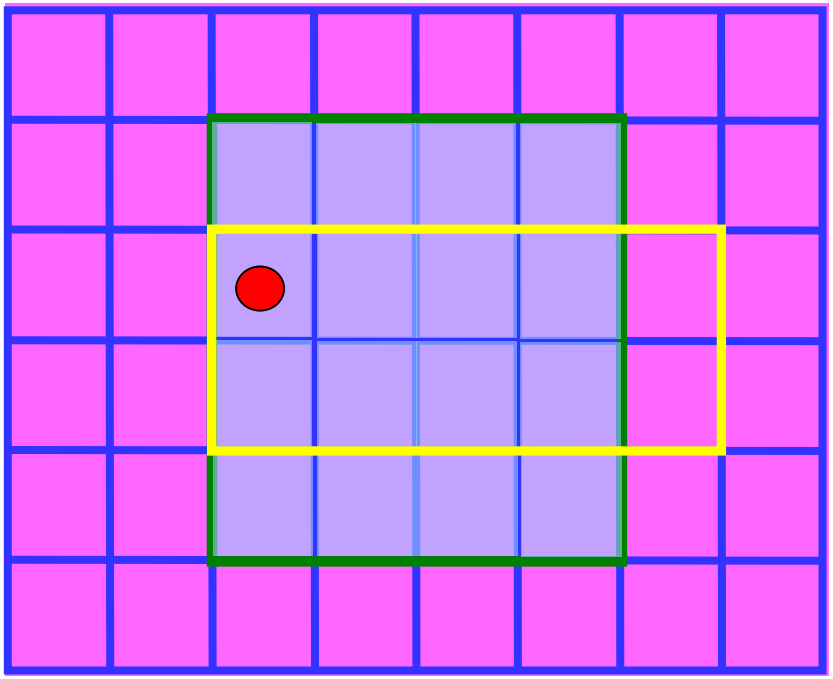


z缓冲器算法举例(2)



z缓冲器算法举例(2)

帧缓冲器



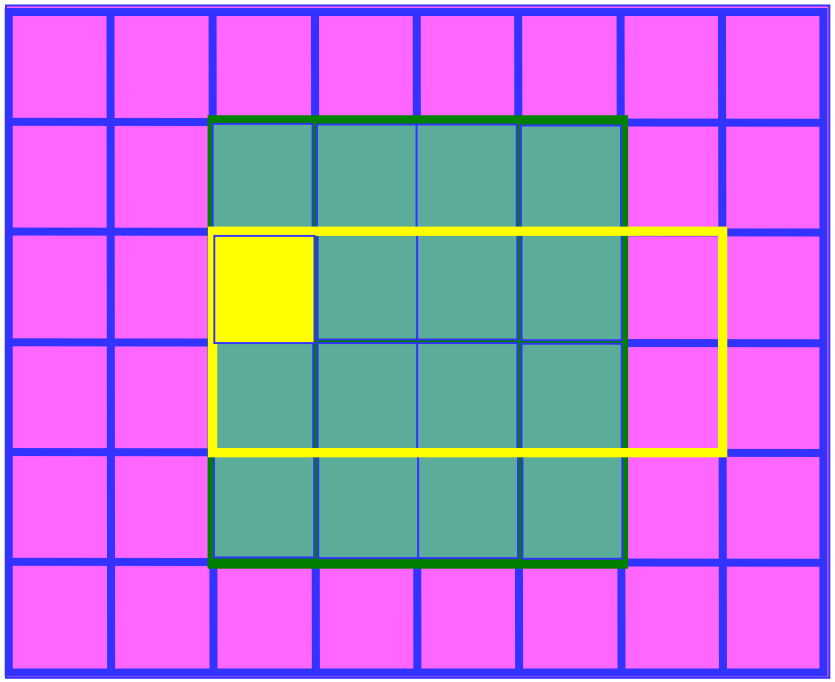
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5

z缓冲器算法举例(2)

帧缓冲器



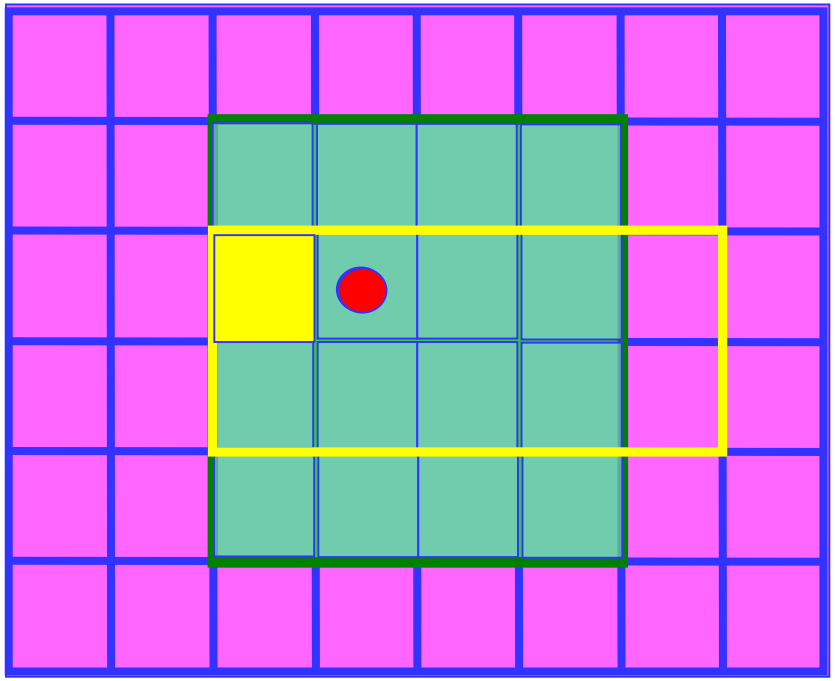
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5

z缓冲器算法举例(2)

帧缓冲器



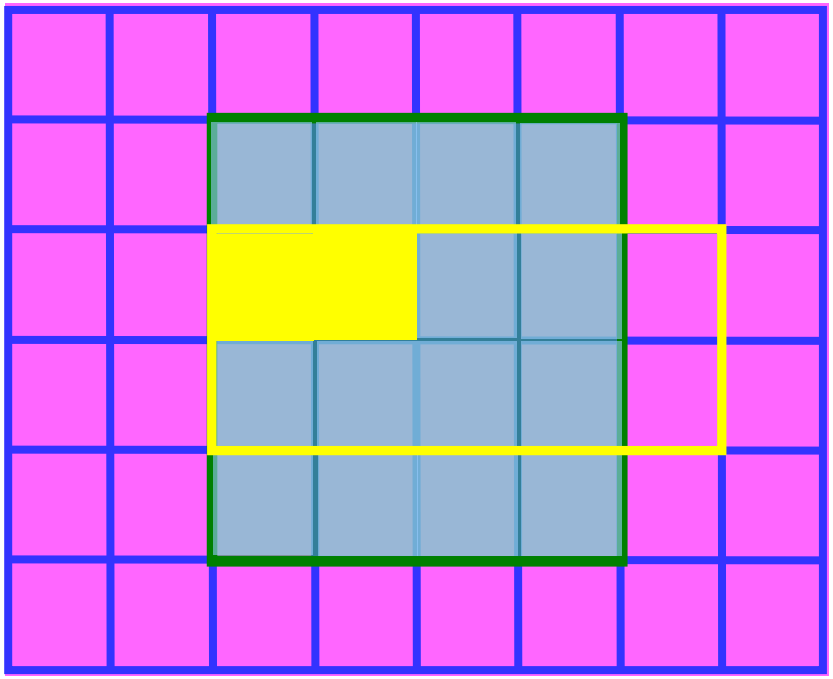
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5	0.5

z缓冲器算法举例(2)

帧缓冲器



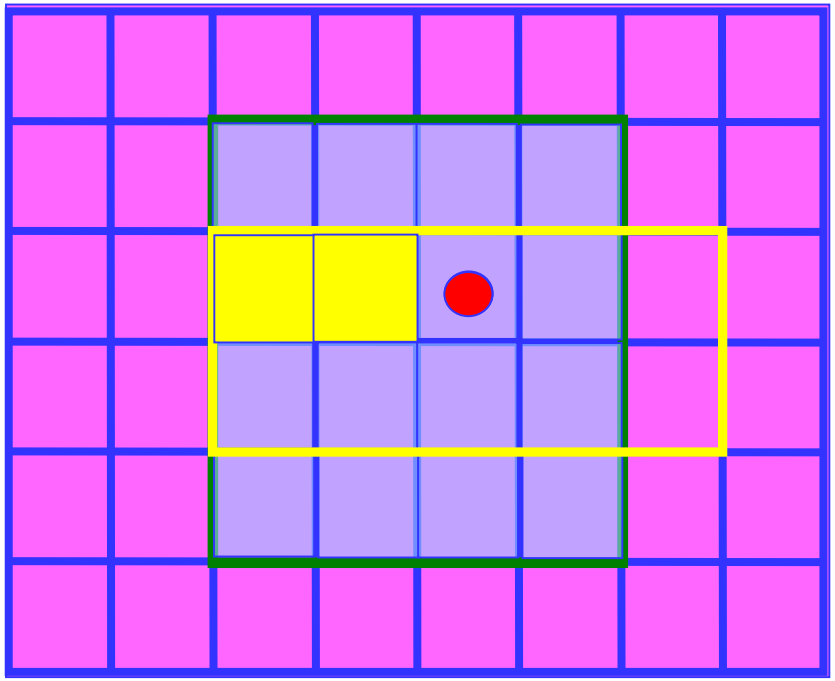
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5	0.5

z缓冲器算法举例(2)

帧缓冲器



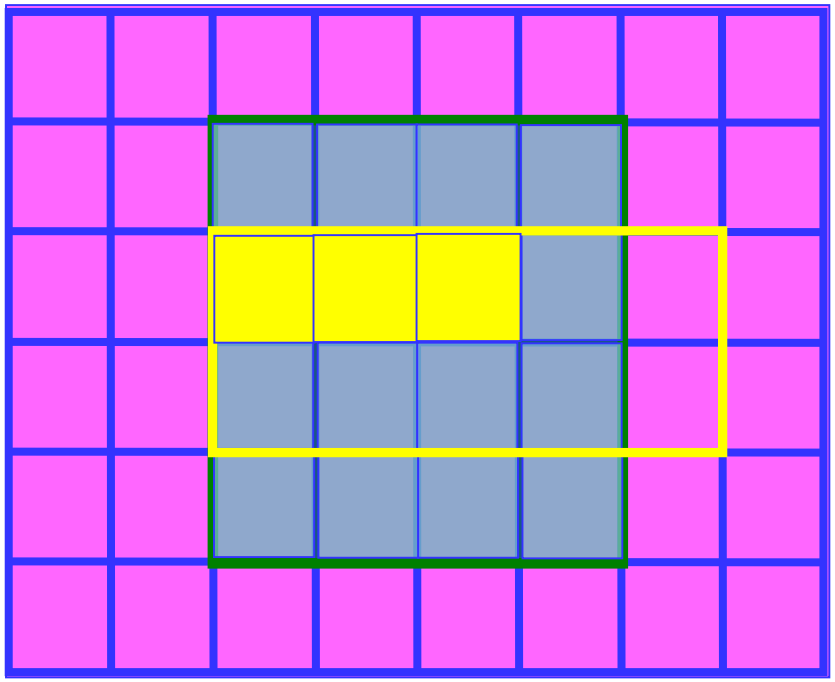
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5	0.5	-0.5

z缓冲器算法举例(2)

帧缓冲器



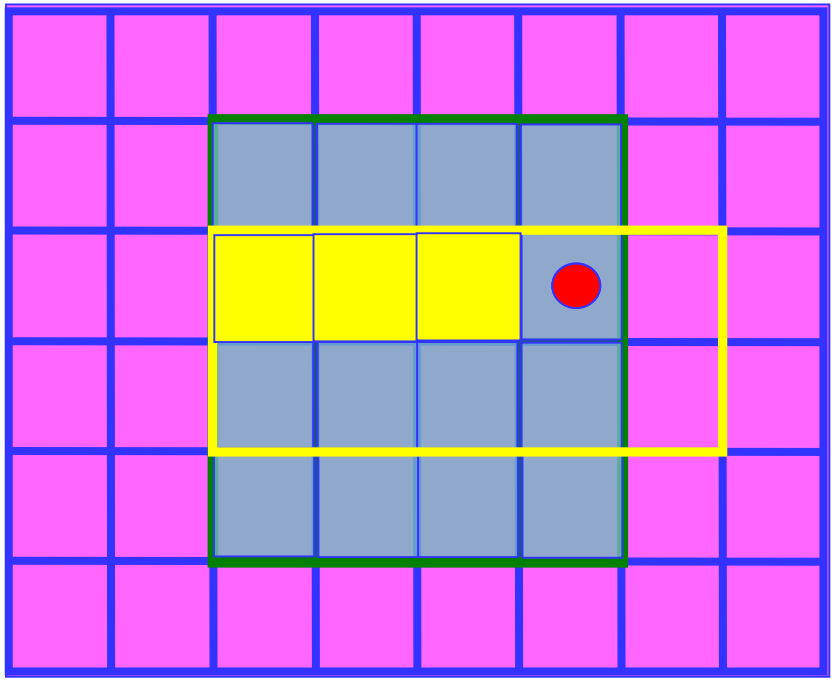
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	1	0	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5	0.5	-0.5

z缓冲器算法举例(2)

帧缓冲器



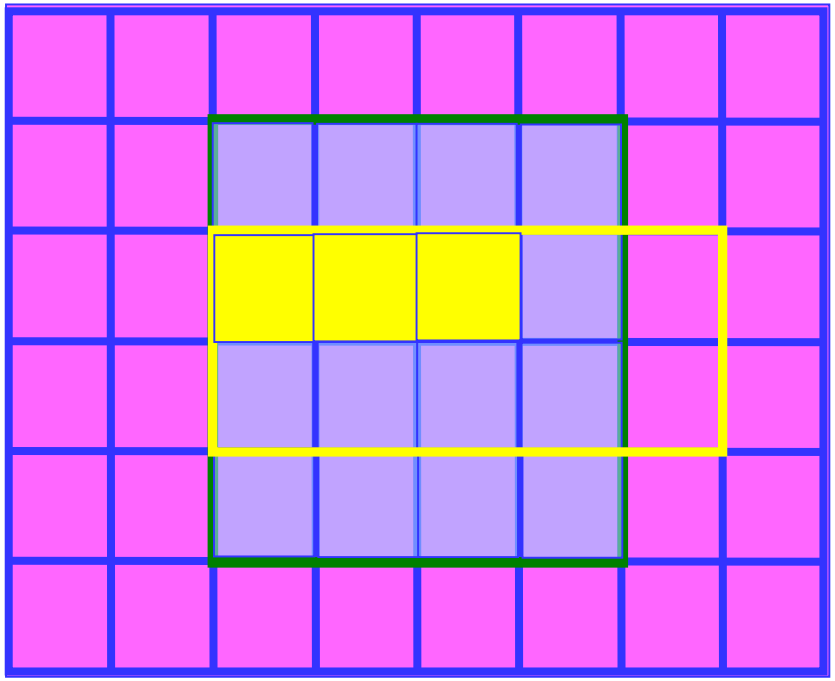
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	1	0	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5	0.5	-0.5	-1.5

z缓冲器算法举例(2)

帧缓冲器



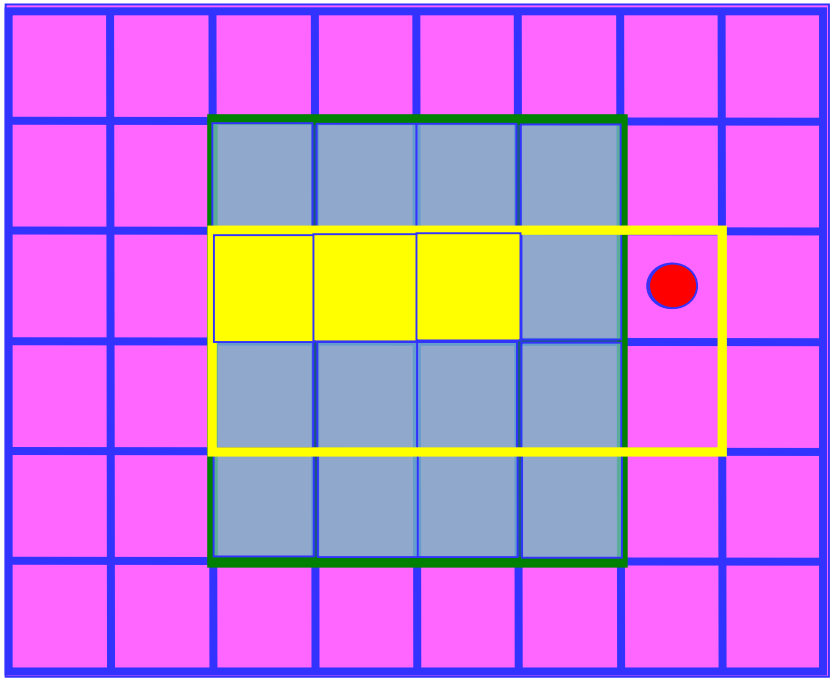
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	1	0	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5	0.5	-0.5	-1.5

z缓冲器算法举例(2)

帧缓冲器



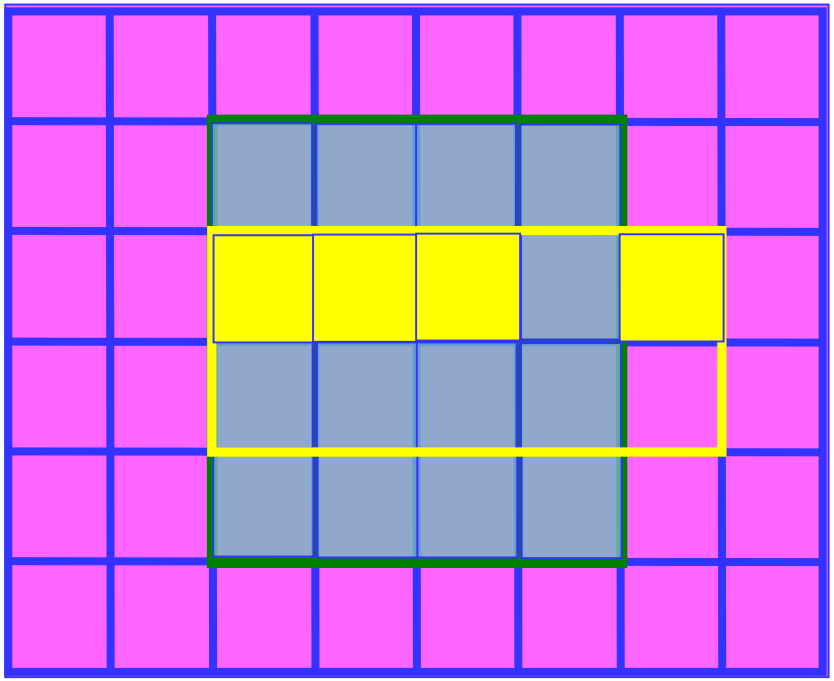
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	1	0	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5	0.5	-0.5	-1.5	-2.5

z缓冲器算法举例(2)

帧缓冲器



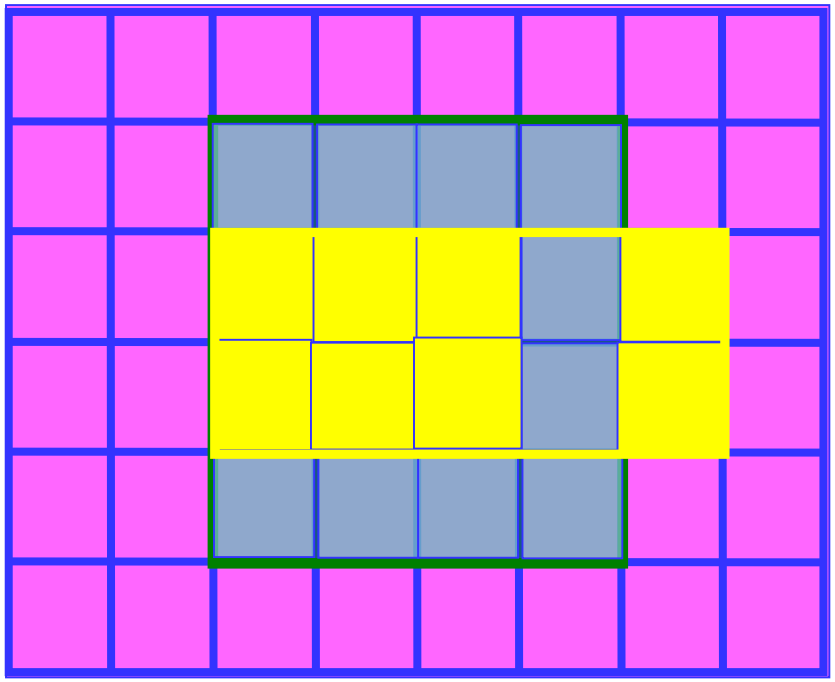
z缓冲器

-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	1	0	-1	-2	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

1.5	0.5	-0.5	-1.5	-2.5

z缓冲器算法举例(2)

帧缓冲器



z缓冲器

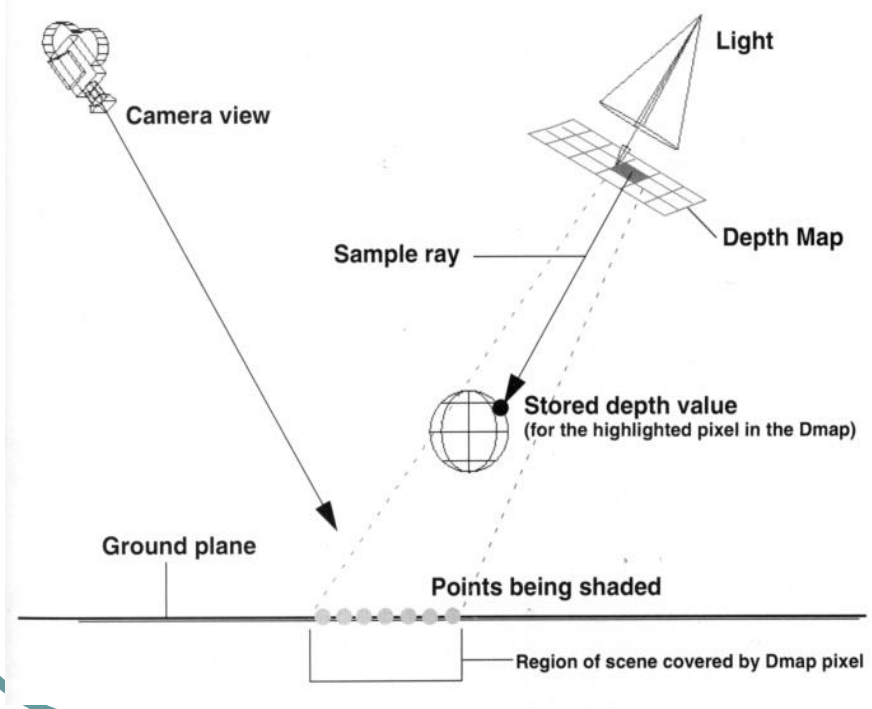
-5	-5	-5	-5	-5	-5	-5	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	2	1	0	-1	-2	-5
-5	-5	2	1	0	-1	-2	-5
-5	-5	-1	-1	-1	-1	-5	-5
-5	-5	-5	-5	-5	-5	-5	-5

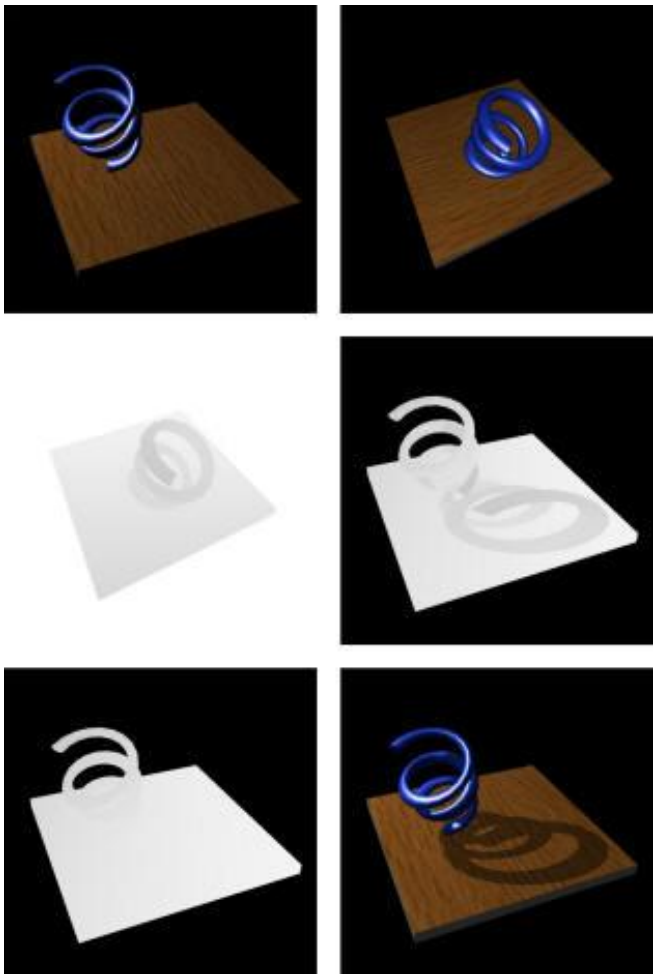
1.5	0.5	-0.5	-1.5	-2.5

z缓冲器的其它应用

- 阴影图 (shadow map)：以光源为视点的z缓冲器

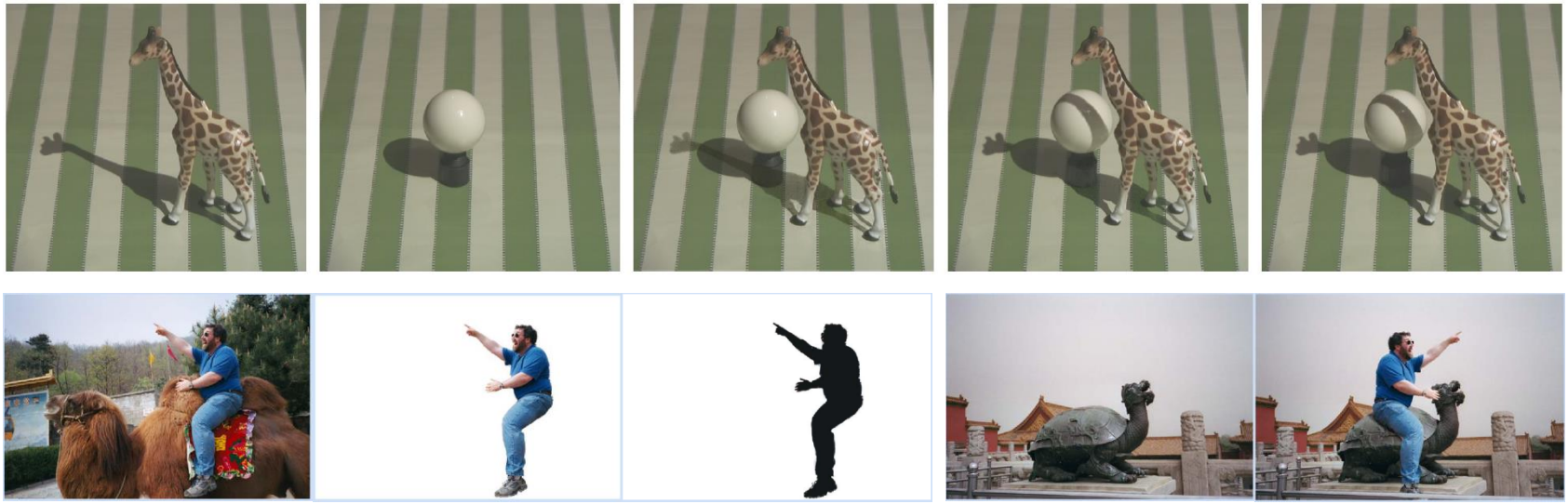
https://en.wikipedia.org/wiki/Shadow_mapping





z缓冲器的其它应用

- rgbα和z缓冲器相结合，实现图像的合成



z缓冲器算法优点

- 给定分辨率条件下，算法线性复杂度 O(nN)
- 算法简单，易于实现：场景中的物体是按任意顺序写入帧缓冲器和z缓冲器，无需排序
- 适合于任何几何物体：能够计算与直线交点
- 可以处理循环遮挡问题
- 可以与扫描线算法相结合
- 适合于并行实现(硬件加速)

z缓冲器算法不足

- z缓冲器需要占用大量的存储单元

  一个大规模复杂场景中：深度范围可能为10^6，一个像素需要24bit来存储其深度信息。如果显示分辨率为1280×1024，那么深度缓冲器需要4MB存储空间

- 深度的采样与量化带来走样现象
- 反走样方法复杂
- 难以引入整体光照明效果
  - 透明物体
  - 阴影效果

主要内容

- 消隐的基本概念
- 线消隐算法
- z缓冲器(*z-buffer*)算法
- 扫描线z缓冲器算法

扫描线z缓冲算法

- 分区域z缓冲器算法
  - 逐个区域地进行z缓冲器消隐，这样z缓冲器的大小就仅为一个子区域的大小
  - 子区域为扫描线时：扫描线z缓冲器算法+连贯性

注：扫描线从上往下扫描

扫描线z缓冲算法的基本思想

- 申请两个一维数组($x_{resolution}$)分别作为当前扫描线的z缓冲器和帧缓冲器
- 在处理当前扫描线时
  - 帧缓冲器的初值取背景颜色
  - z缓冲器的初始值取为最小z值
- 求出该扫描线与场景中各多边形的二维投影之间的交点
  - 一条扫描线和一个多边形有偶数个交点

扫描线z缓冲算法的基本思想

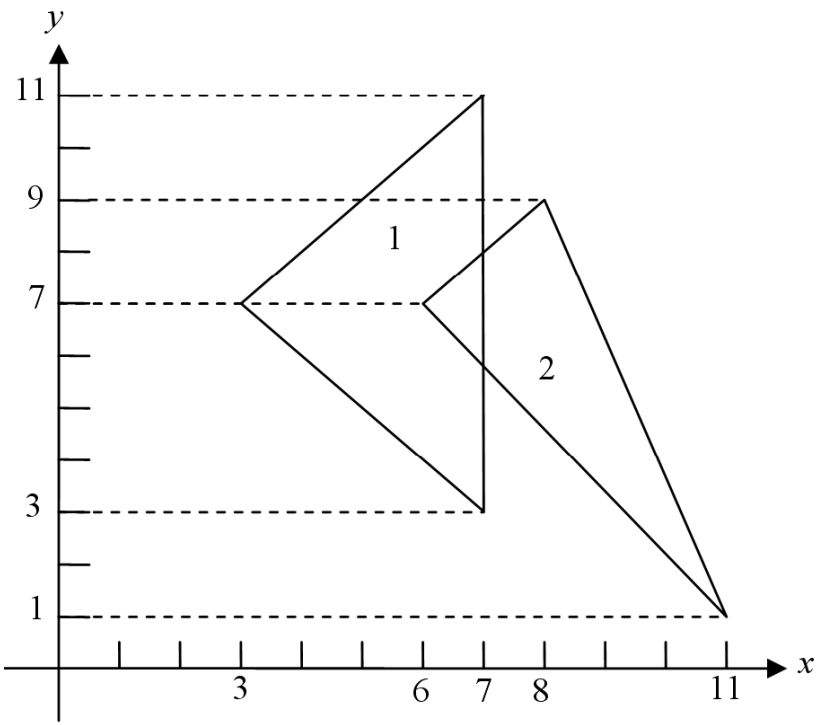
- 对每对交点之间的像素
  - 计算其深度值
  - 与z缓冲中的值比较
    - 若可见，将该多边形的属性写入帧缓冲器，更新z-buffer中的深度值
- 当场景中所有多边形处理完毕时，扫描线帧缓冲器中的内容为画面在此扫描线的消隐结果

扫描线z缓冲器算法的数据结构

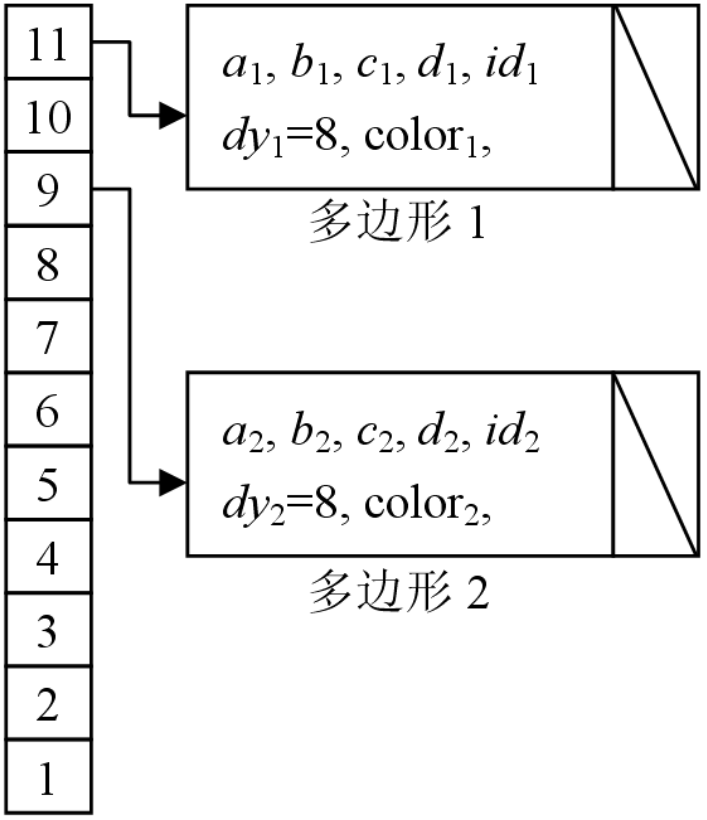
- 分类多边形表： 根据y_{max}(多边形的最大y坐标)将多边形放入相应的类中
  - a,b,c,d：多边形所在平面的方程系数

    $$ax+by+cz+d=0$$
  - id：多边形的编号
  - dy：多边形跨越的扫描线数目
  - $color$：多边形的颜色

分类多边形表举例



两个空间多边形在oxy平面上的投影

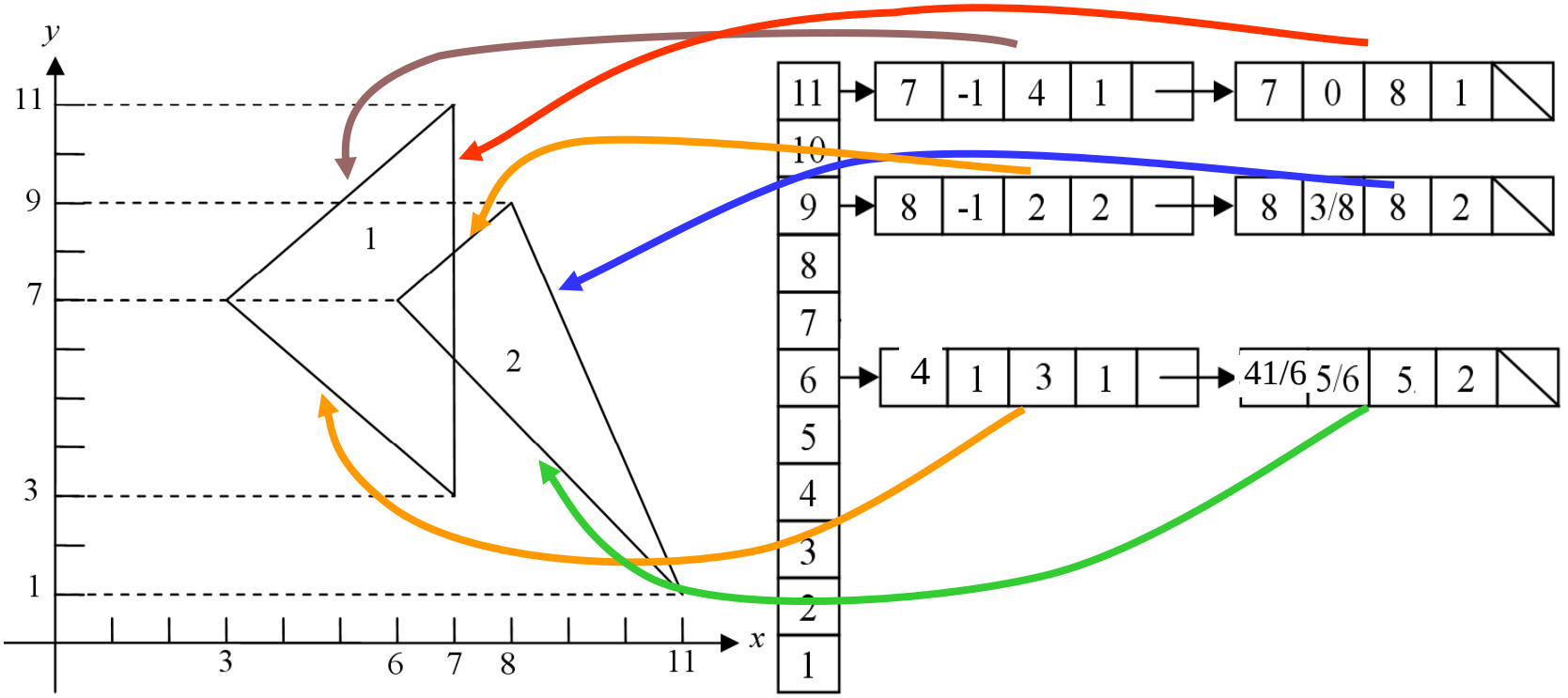


分类的多边形表

扫描线z缓冲器算法的数据结构

- 分类边表： 根据y_{max}(边的上端点y坐标)将边放入相应的类中
  - x：边上端点的x坐标 (非极值点：截断1 Pixel)
  - dx：相邻两条扫描线交点的x坐标差dx ($-1/k$)
  - dy：边跨越的扫描线数目
  - id：边所属多边形的编号

分类边表



两个空间多边形在*oxy*平面上的投影

分类边表

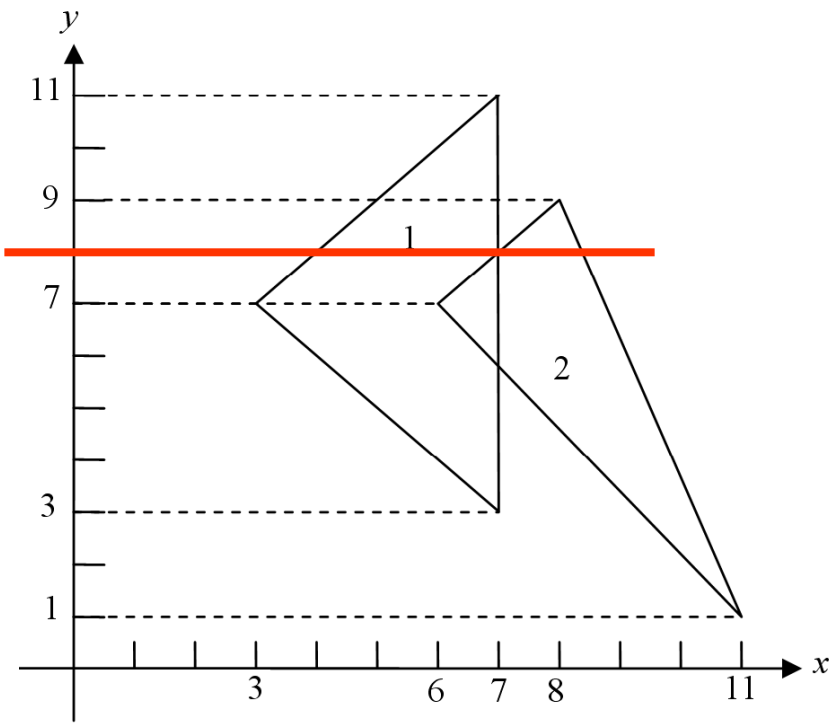
扫描线z缓冲器算法的数据结构

- 算法中间的数据结构
  - 活化多边形表：记录了当前扫描线与多边形在*oxy*投影面上投影相交的多边形
    - *a*,*b*,*c*,*d*：多边形所在平面的方程系数

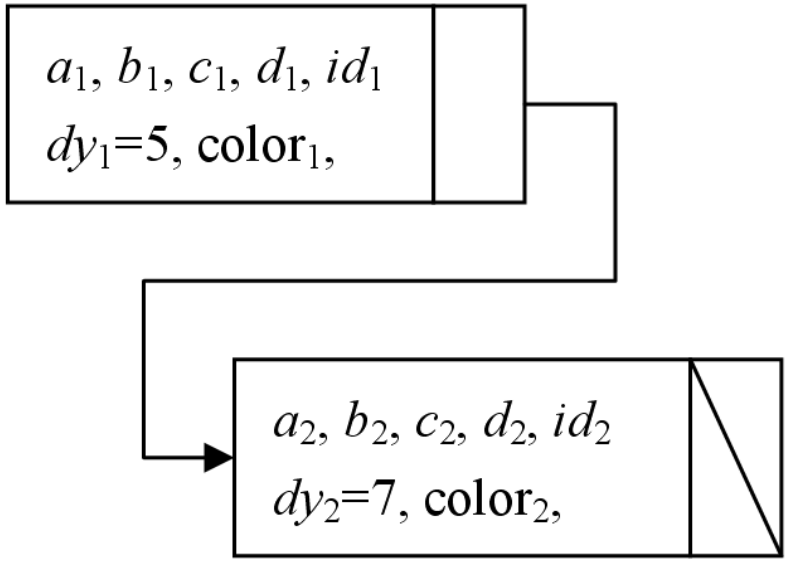
      $$ax+by+cz+d=0$$

    - *id*：多边形的编号
    - *dy*：多边形跨越的剩余扫描线数目
    - *color*：多边形的颜色

活化多边形表



两个空间多边形在oxy平面上的投影



y=8活化多边形表

扫描线z缓冲器算法

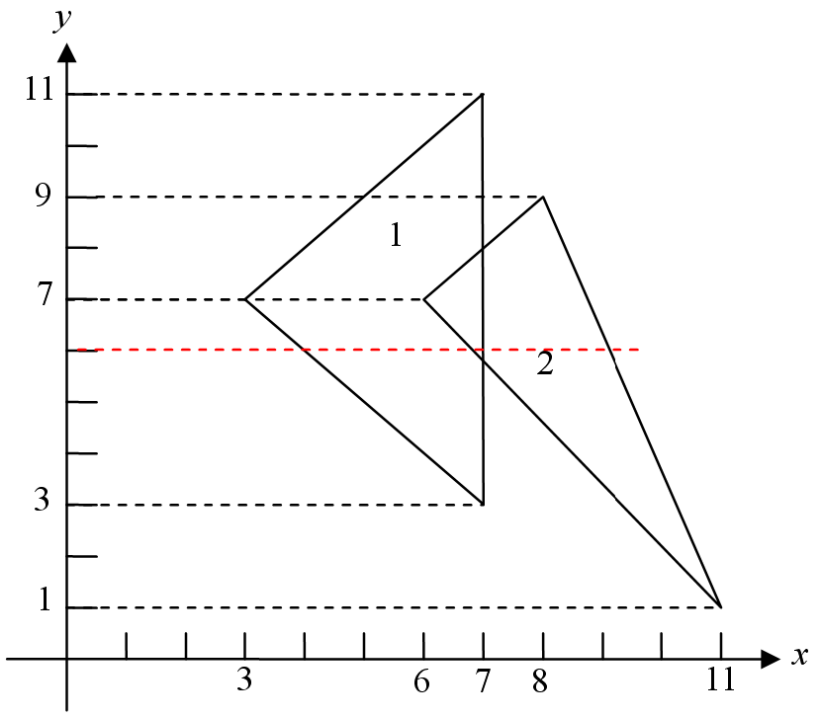
- 算法中间的数据结构
  - 活化边表：存放投影多边形边界与扫描线相交的边对
    - x_l： 左交点的x坐标
    - dx_l：(左交点边上)两相邻扫描线交点的x坐标之差
    - dy_l： 以和左交点所在边相交的扫描线数为初值，以后向下每处理一条扫描线减1
    - x_r、dx_r和dy_r：右边的交点的三个对应分量；其含义与左边交点一样

(接下一页)

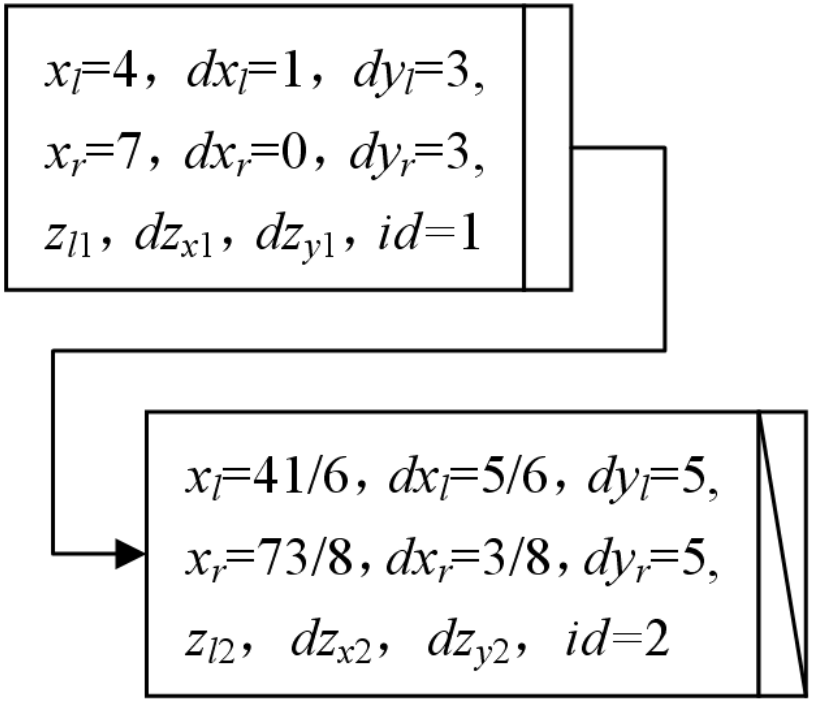
扫描线z缓冲器算法

- 算法中间的数据结构
  - 活化边表：存放多边形边界与扫描线相交的边对
    - z_l：左交点处多边形所在平面的深度值；
    - dz_x：沿扫描线向右走过一个像素时，多边形所在平面的深度增量。对于平面方程，$dz_x = -a/c\ (c \neq 0)$；
    - dz_y：沿y方向向下移过一根扫描线时，多边形所在平面的深度增量。对于平面方程，$dz_y = b/c\ (c \neq 0)$；
    - id：交点对所在的多边形的编号；

活化边表



两个空间多边形在oxy平面上的投影



y=6活化边表

扫描线z缓冲器算法描述

- 首先建立分类的多边形表和边表
- 扫描顺序从上到下：在处理最上面一条扫描线之前，活化的多边形表和边表是空的
- 在处理每条扫描线时，作如下工作
  - 把帧缓冲器的相应行置成底色
  - 把z缓冲器的各个单元置成最小值(表示离视点最远)

扫描线z缓冲器算法描述

- 检查分类的多边形表，如果有新的多边形涉及该扫描线，则把它放入活化的多边形表中
- 如果有新的多边形加入到活化多边形表中，则把该多边形在oxy平面上的投影和扫描线相交的边加入到活化边表中
- 如果有些边在这条扫描线处结束了，而其所在的多边形仍在活化多边形表内，则可以从分类多边形表中找到该多边形在oxy平面上的投影与扫描线相交的新边或边对，修改或加到活化边表中去。边对在活化边表中的次序是不重要的

扫描线z缓冲器算法描述

- 增量式的深度更新：从形成的活化边表中取出一个边对
  - 当前扫描线y，当前位置$x(x_l \le x \le x_r)$，深度值z_x(左交点$z_x=z_l$)
  - 每向右前进一个像素：$z_x = z_x + dz_x$；
  - 比较z_x与当前z缓冲器中的z值：
    - 如果$z_x>z$，那么$z = z_x$，将对应像素颜色置为该多边形的颜色
- 对活化边表中每一个边对按如上方法处理

扫描线z缓冲器算法描述

- 活化边表中的元素修改：修改后的活化边表是下一条扫描线的活化边表
  - 对于每一条边对可计算：

    $dy_l = dy_l - 1 \quad dy_r = dy_r - 1$

    若dy_l或dy_r小于0，相应的边就要从一个边对中去掉，从活化边表中找合适的边来代替
  - 边和下一条扫描线交点的x值：

    $x_l = x_l + dx_l \quad x_r = x_r + dx_r$

扫描线z缓冲器算法描述

- 活化边表中的元素修改：修改后的活化边表是下一条扫描线的活化边表
  - 多边形所在的平面对应下一条扫描线在$x = x_l$处的深度为

$$z_l = z_l + dz_l\, dx_l + dz_y$$

- 活化多边形表中的元素修改
  - 每一个多边形的dy：$dy = dy - 1$
  - 当$dy < 0$时，该多边形要从多边形活化表中删除

扫描线z缓冲器算法分析

- 与z缓冲算法相比，扫描线z算法的优点
  - 将整个窗口的z缓冲问题分解到一条条扫描线上解决，算法所需的z缓冲区大大减少
  - 计算深度时，充分利用了各种连贯性，只用到了加法
- 不足之处：计算量仍然较大！
  - 每一个像素都要进行深度计算以及深度比较
  - 被多个多边形覆盖的像素要进行多次比较

小结

- 消隐的基本概念
- 线消隐算法
- z缓冲器(*z-buffer*)算法
- 扫描线z缓冲器算法