華東理工大學

计算机图形学

2023年11月

信息楼101A

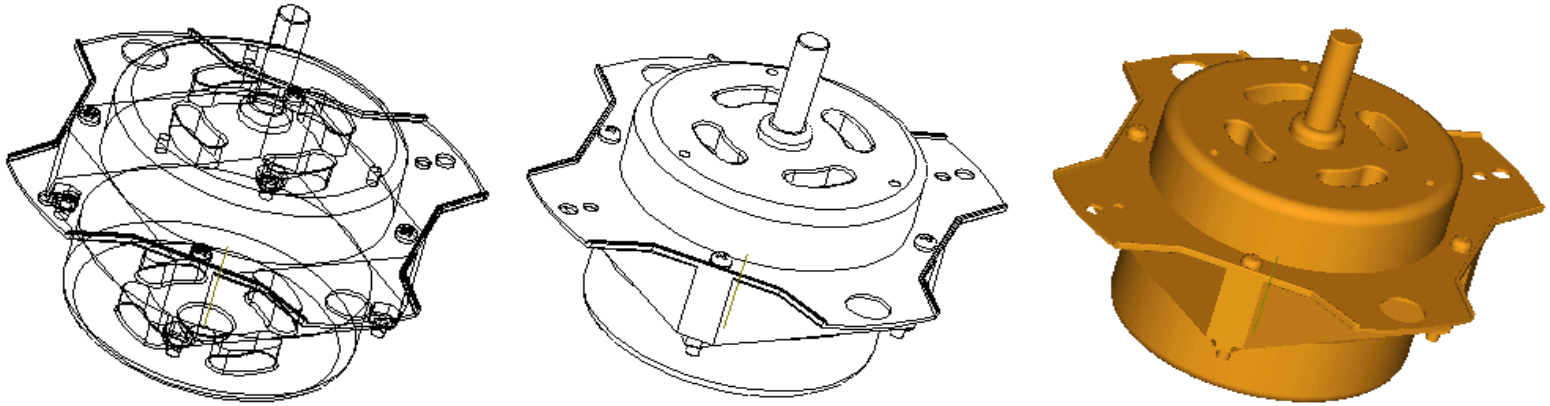
09

消隐

Hidden Surface Removal

本章要点

- 什么是消隐，为什么要消隐
- 如何进行消隐
- 在OpenGL中如何实现消隐

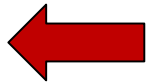


消隐

1 消隐的基本概念

2 消隐的主要算法

3 OpenGL中的消隐



基本概念

- **消隐：**物体的消隐或隐藏线面的消除
- 在给定视点和视线方向后，决定场景中哪些物体的表面是可见的，哪些是被遮挡不可见的。

基本概念

- 消隐算法按实现方式分类
  - 图像空间消隐算法　　（光栅化后）
  - 景物空间消隐算法　　（光栅化前）
  - 介于二者之间的算法

基本概念

- **图像空间**消隐算法
  - 以屏幕像素为采样单位，确定投影于每一像素的可见景物表面区域，并将其颜色作为该像素的显示颜色。
  - 典型算法：**深度缓冲器（Z-Buffer）算法**、A缓冲器算法、区间扫描线算法等。

基本概念

- **景物空间**消隐算法
  - 直接在景物空间（观察坐标系）中确定视点不可见的表面区域，并将它们表达成同原表面一致的数据结构。
  - 典型算法：**BSP算法**、多边形区域排序算法等。

基本概念

- 介于二者之间的算法
  - 典型算法：深度排序算法、区域细分算法、**光线投射算法**等。

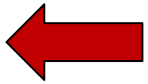
基本概念

- 基本的原则
  - **排序**：各景物表面按照距离视点远近排序的结果，用于确定消隐对象之间的遮挡关系。
  - **连贯性**：连贯性是指所考察的物体或视区内的图像局部保持不变的一种性质，用于提高排序效率。
  - 什么地方还用到连贯性？

消隐

1 消隐的基本概念

2 消隐的主要算法



3 OpenGL中的消隐

深度缓存器算法（ Z-buffer）

- 基本原理
  - 帧缓存：保存各点的颜色。
  - Z缓存：保存屏幕坐标系上各象素点所对应的深度值。

深度缓存器算法（ Z-buffer）

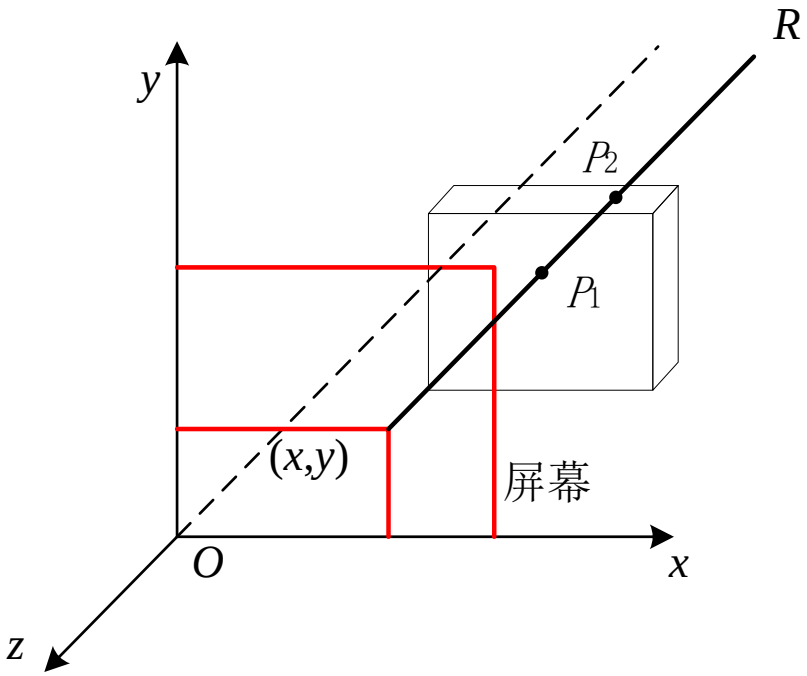
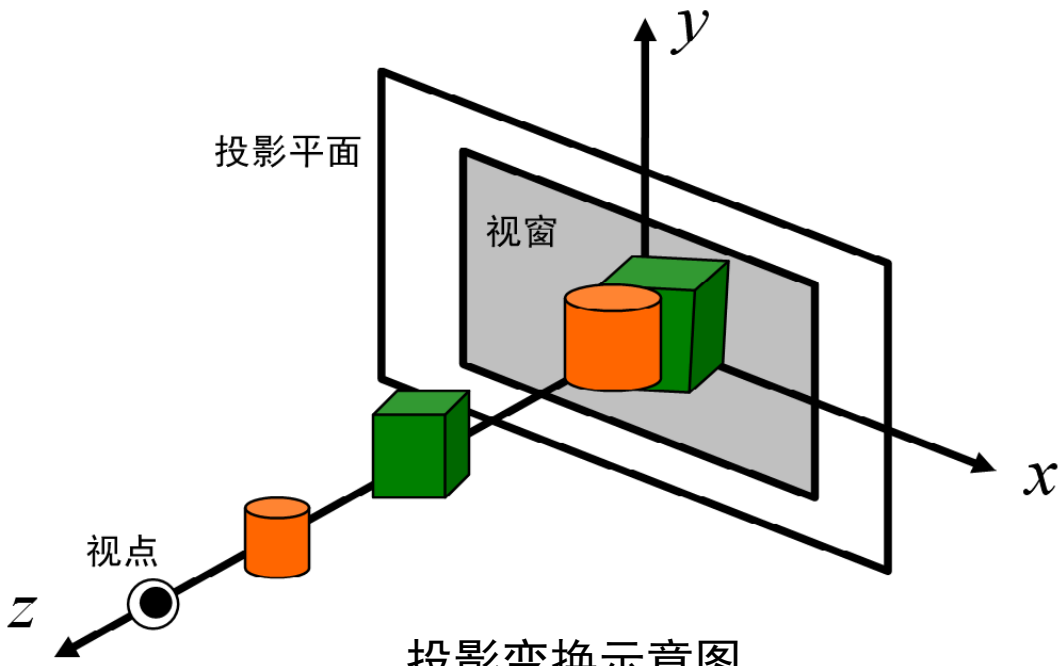


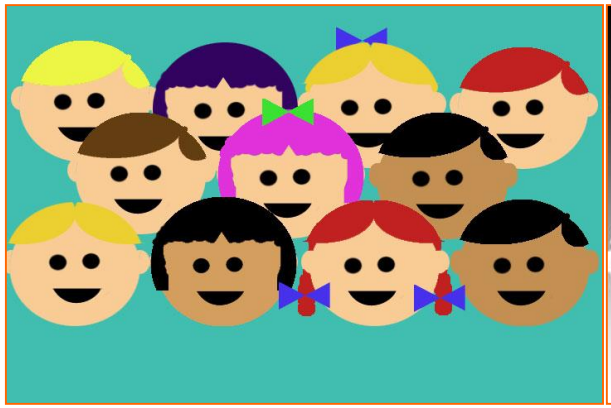
图9.1 深度缓存器算法的原理

深度缓存器算法（Z-buffer）



投影变换示意图

颜色与深度缓冲举例



颜色缓冲

深度缓冲

深度缓存器算法（ Z-buffer）

- 步骤
  - 初始化：把Z缓存中各(x, y)单元置为**z的最小值**，而帧缓存各(x, y)单元置为**背景色**。
  - 在把物体表面相应的多边形扫描转换成帧缓存中的信息时，对于多边形内的每一采样点(x, y)进行处理：

深度缓存器算法（ Z-buffer）

- 计算采样点(x, y)的深度z(x, y)；
- 如z(x, y)大于Z缓存中在(x, y)处的值，则把z(x, y)存入Z缓存中的(x, y)处，再把多边形在z(x, y)处的颜色值存入帧缓存的(x, y)地址中。

深度缓存器算法（ Z-buffer）

- 如何计算采样点(x, y)的深度z(x, y)。
  - 假定多边形的平面方程为：Ax+By+Cz+D=0。

$$z(x,y) = \frac{-Ax - By - D}{C}$$

深度缓存器算法（ Z-buffer）

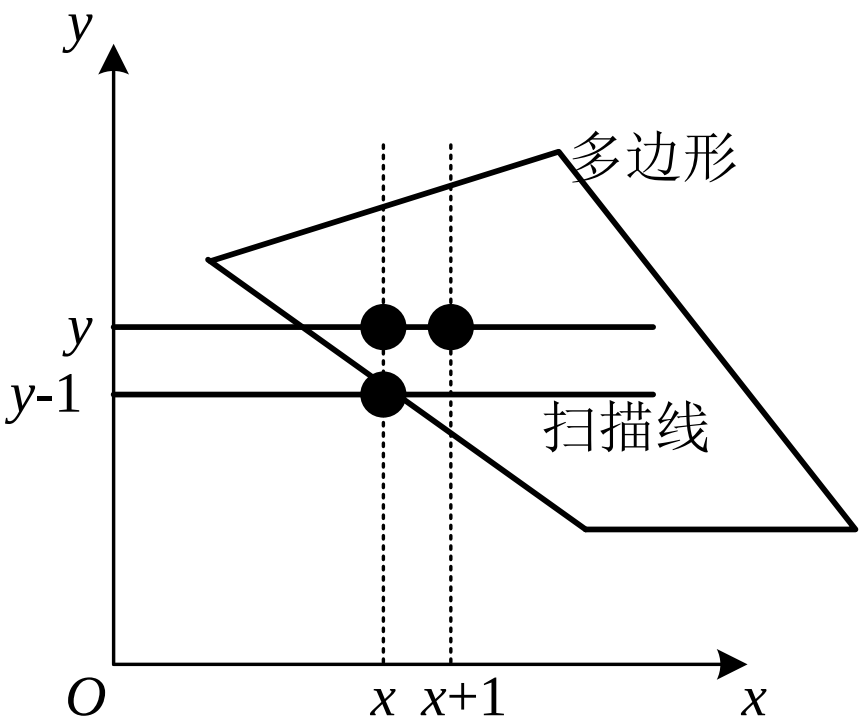


图9.2 利用扫描线的连贯性加速深度的计算

深度缓存器算法（ Z-buffer）

- 扫描线上所有后继点的深度值：

$$z(x+1,y)=\frac{-A(x+1)-By-D}{C}=z(x,y)-\frac{A}{C}$$

- 当处理下一条扫描线y=y-1时，该扫描线上与多边形相交的最左边（x最小）交点的x值可以利用上一条扫描线上的最左边的x值计算：

$$x\big|_{y-1,\min}=x\big|_{y,\min}-\frac{1}{k}$$

深度缓存器算法（ Z-buffer）

$$z(x|_{y-1,\min}, y-1) = \frac{-Ax|_{y-1,\min} - B(y-1) - D}{C}$$

$$= \frac{-A(x|_{y,\min} - \frac{1}{k}) - B(y-1) - D}{C}$$

$$= z(x|_{y,\min}, y) + \frac{\frac{A}{k} + B}{C}$$

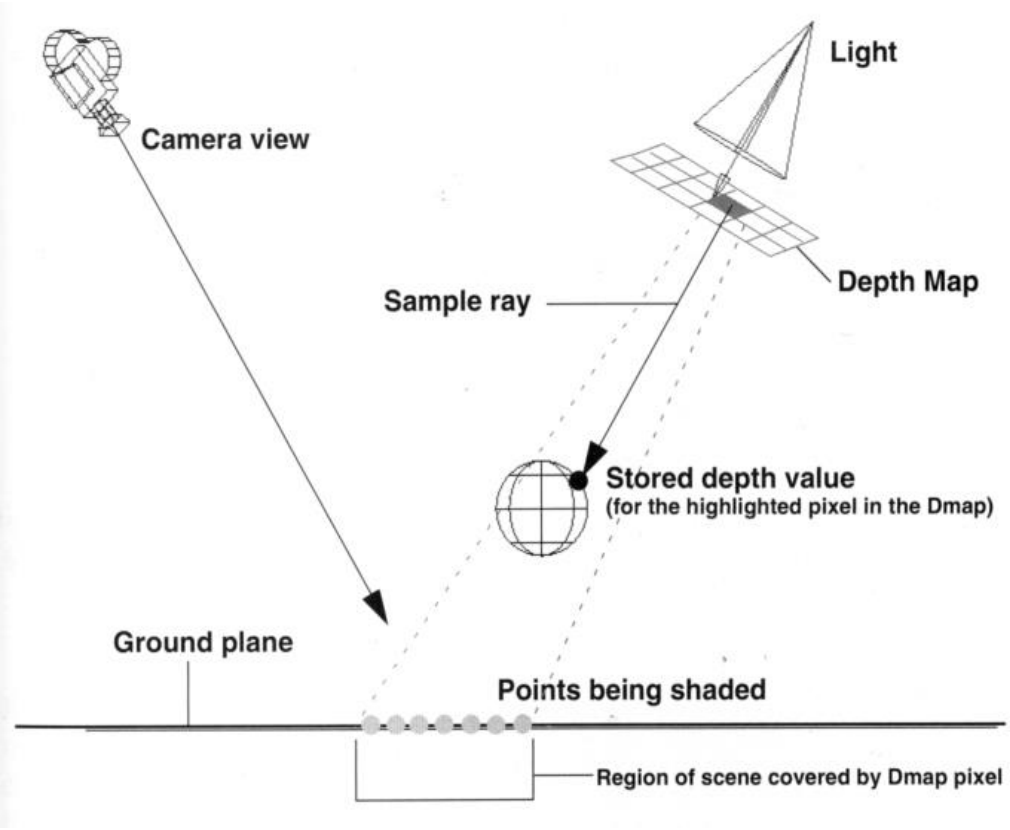
- 扫描线深度缓存器算法

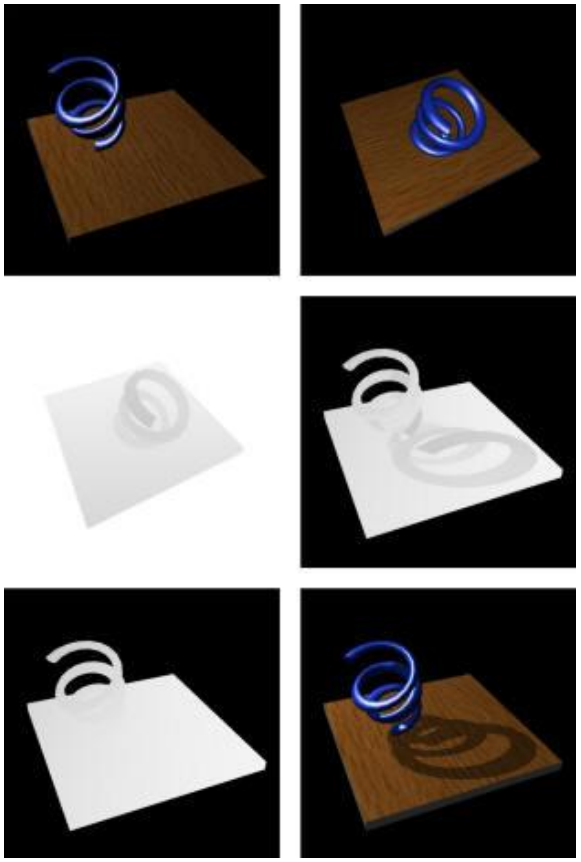
深度缓存器算法（ Z-buffer）

- 优点
  - 简单
  - 便于硬件实现
- 缺点
  - 占用太多的存储单元
  - 在实现反走样、透明和半透明等效果方面有困难

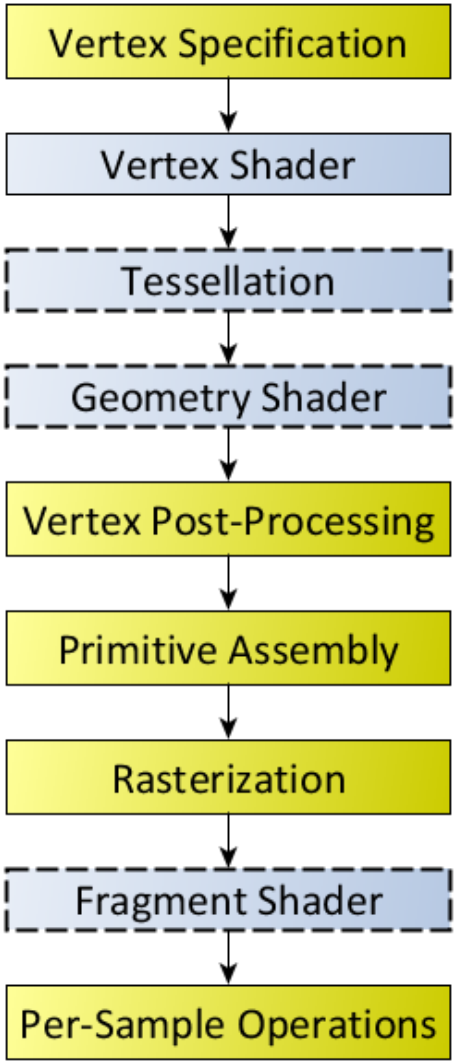
Z-buffer的其它应用

⑩ 阴影算法：以光源为视点的z缓冲器





OpenGL绘制流水线



1. Vertex Processing（**Vertex Shader**）：
   ① Each vertex retrieved from the vertex arrays (as defined by the VAO) is acted upon by a Vertex Shader.
   ② Optional primitive tessellation stages.
   ③ Optional **Geometry Shader** primitive processing
2. Vertex Post-Processing, the outputs of the last stage are adjusted or shipped to different locations.
   ① Transform Feedback happens here.
   ② Primitive Assembly
   ③ Primitive Clipping, the perspective divide, and the viewport transform to window space.
3. Scan conversion and primitive parameter interpolation, which generates a number of Fragments.
4. A **Fragment Shader** processes each fragment. Each fragment generates a number of outputs.
5. Per-Sample_Processing, including but not limited to:
   Scissor Test /Stencil Test / Depth Test / Blending
   Logical Operation/ Write Mask

https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

区间扫描线算法

- 在深度缓存器算法基础上简化
- 利用了沿扫描线的连贯性，采用增量算法求解多边形上的采样点，占用的存储量减少
- 仍需对投影在屏幕上的所有多边形进行采样

区间扫描线算法

- 避免对被遮挡区域的采样是进一步提高扫描线算法计算效率的关键。

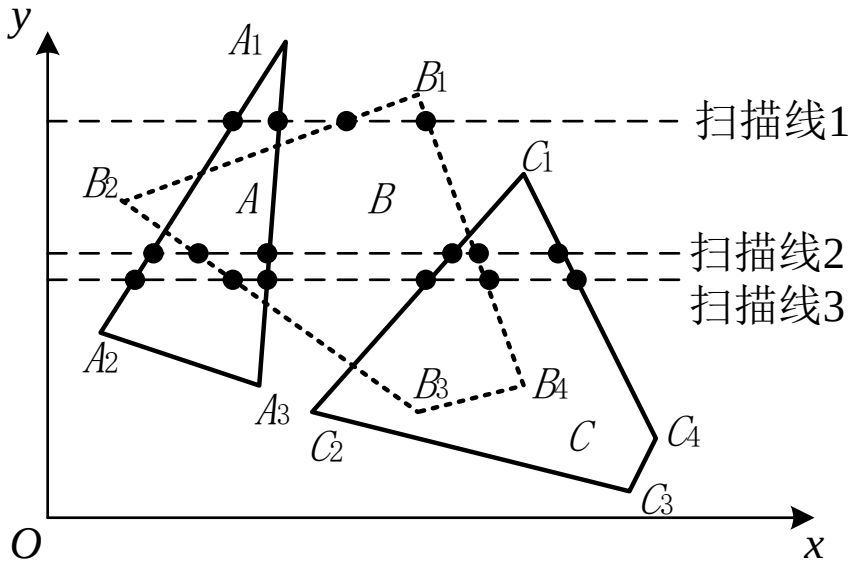


图9.3 区间扫描线算法原理

区间扫描线算法

- 算法
  - 三张表：边表、多边形表、有效边表。
  - 分割子区间，确定子区间上的唯一可见面。

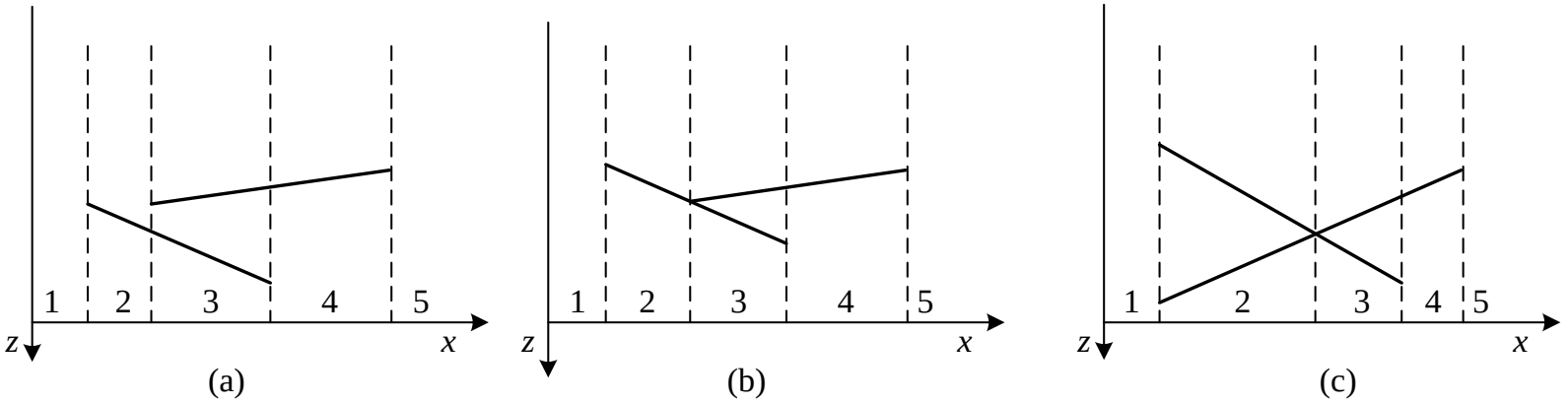


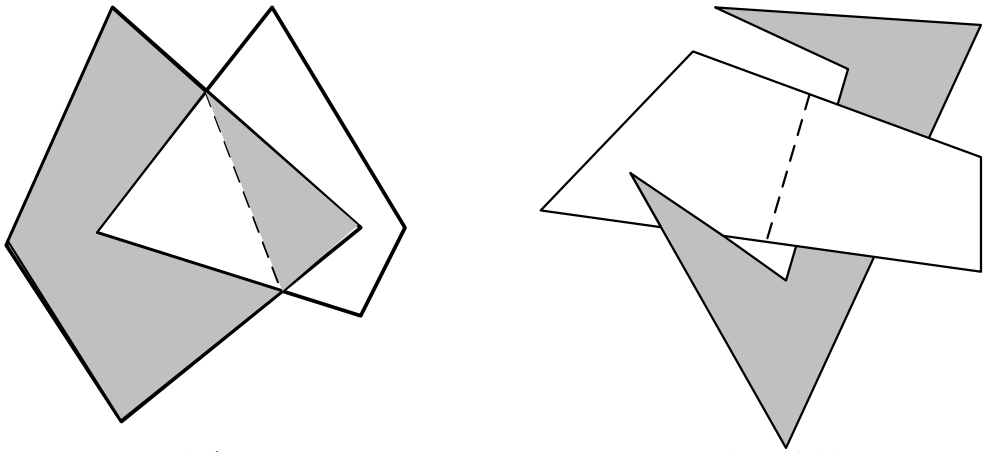
图9.4 扫描线子区间

区间扫描线算法

- 特殊情形
  - 贯穿情形：为了使算法能处理互相贯穿的多边形，扫描线上的分割点不仅应包含各多边形的边与扫描线的交点，而且应包含这些贯穿边界与扫描线的交点。

区间扫描线算法

- 循环遮挡：将多边形进行划分以消除循环遮挡。



(a)贯穿　　(b)循环遮挡

图9.5 多边形贯穿和循环遮挡的情形

深度排序算法（画家算法）

- 介于图像空间消隐算法和景物空间消隐算法之间的一种算法
- 在景物空间中预先计算物体上各多边形可见性的优先级，然后再在图像空间中产生消隐图

深度排序算法（画家算法）

- 算法原理：算法约定距视点近的优先级高，距视点远的优先级低。生成图像时，优先级低的多边形先画，优先级高的多边形后画。这样，后画的多边形就会将先画的多边形遮挡住，从而达到消隐的效果。
- 算法的关键是多边形排序。

深度排序算法（画家算法）

- 算法步骤
  - 将多边形按深度进行排序：距视点近的优先级高，距视点远的优先级低
  - 由优先级低的多边形开始，逐个对多边形进行扫描转换

深度排序算法（画家算法）

- 排序步骤
  - 对场景中的所有多边形按z_{min}(多边形)由小到大的顺序存入一个先进先出队列中，记为M，同时初始化一空的先进先出队列N

深度排序算法（画家算法）

- 若*M*中的多边形个数为1，则将*M*中的多边形直接加入到*M*中，算法结束；
- 否则按先进先出的原则从*M*中取出第一个多边形*A*进行处理，同时将*A*从*M*中删除

深度排序算法（画家算法）

- 从当前*M*中任意选择一多边形*B*,

  对*A*与*B*进行判别

①若对*M*中任意的*B*均

$z_{min}(B) > z_{max}(A)$， *A*按先进先出原

则加入*M*中

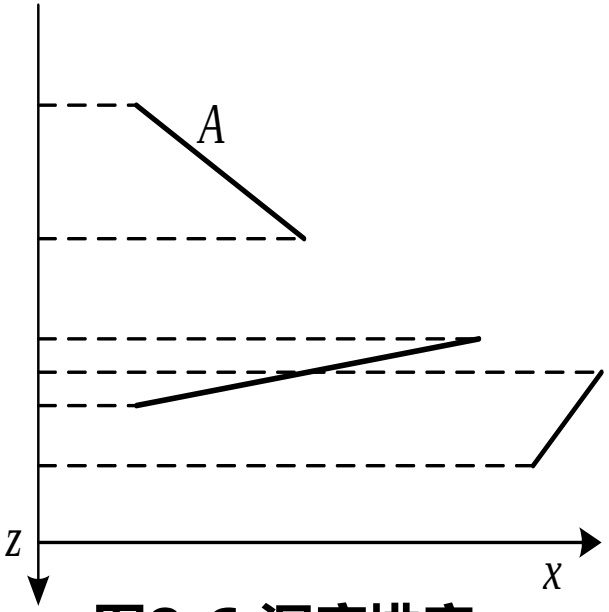


图9.6 深度排序

深度排序算法（画家算法）

②判别多边形*A*和*B*在*xoy*平面上投影的包围盒有无重叠，若无重叠将*A*按先进先出原则加入*M*中

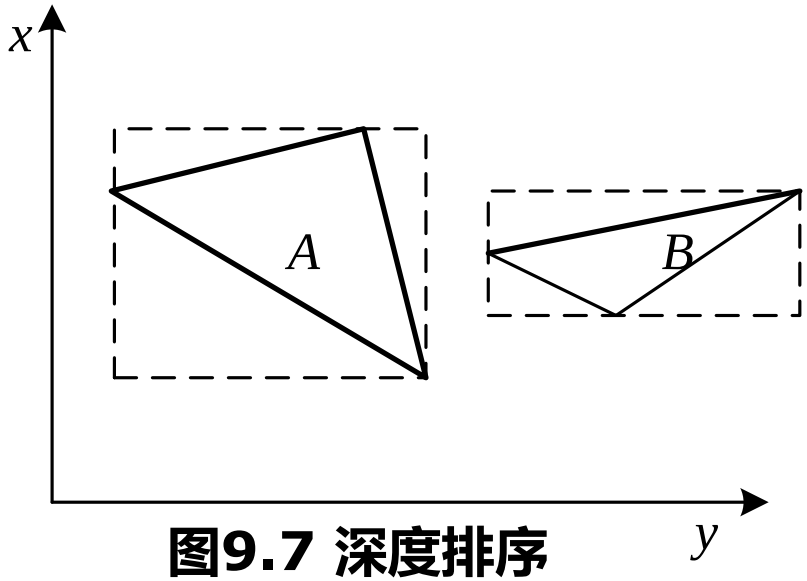


图9.7 深度排序

深度排序算法（画家算法）

③ 判别平面*A*是否完全位于*B*上*A*与*B*的重叠平面之后，

若是，将*A*按先进先出原则加入*M*中

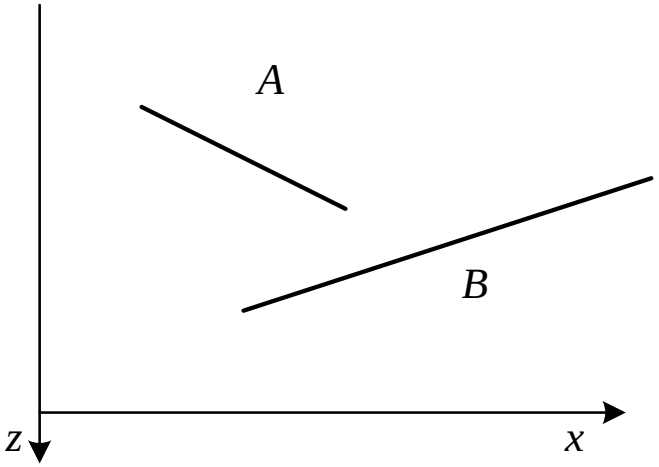


图9.8 深度排序

深度排序算法（画家算法）

④ 判别*B*上平面*A*与*B*的重叠平面是否完全位于*A*之前，

若是，将*A*按先进先出原则加入*M*中

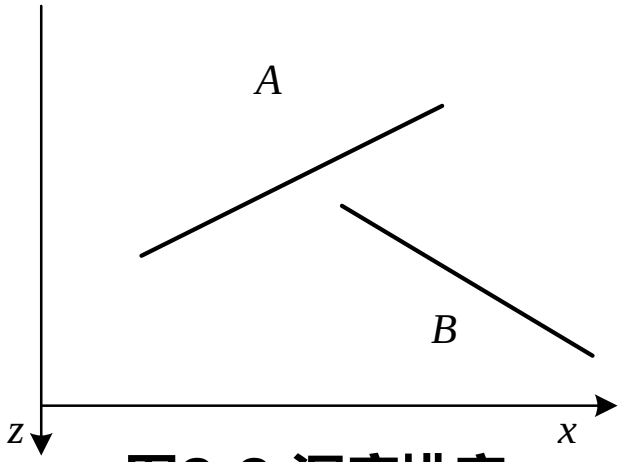


图9.9 深度排序

深度排序算法（画家算法）

⑤ 判别多边形*A*和*B*在*xoy*平面上的投影若无重叠，将*A*按先进先出原则加入*M*中；

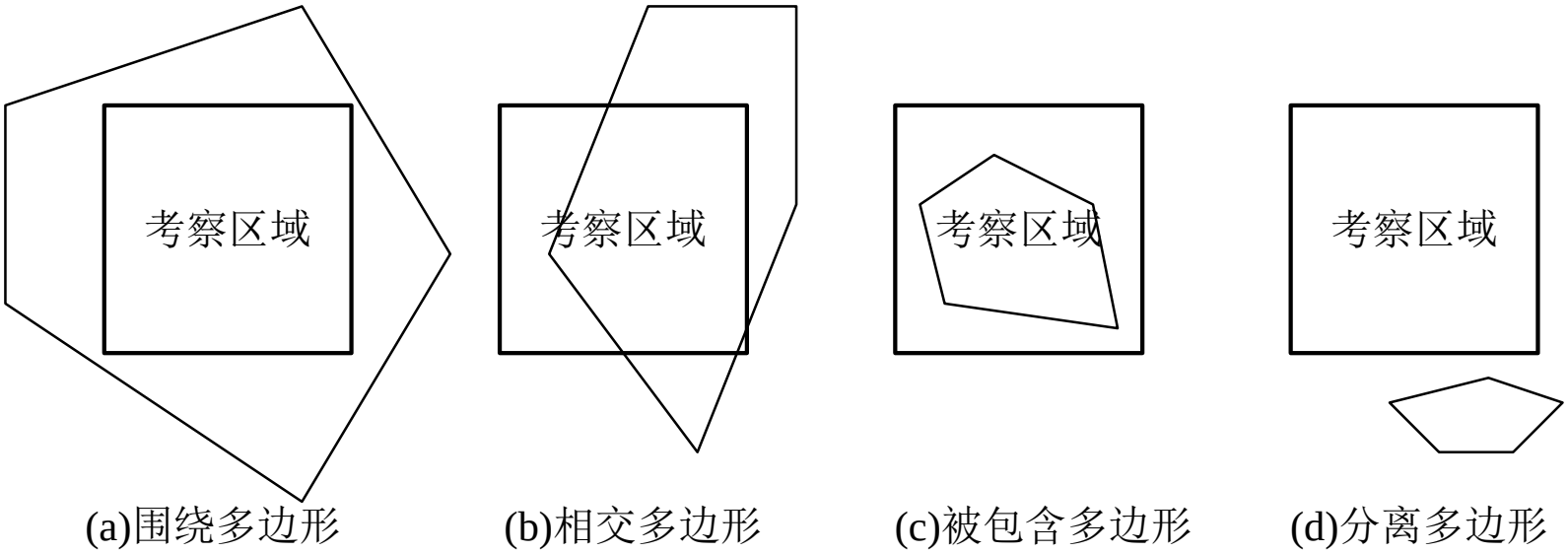
⑥ 在*A*与*B*投影的重叠区域中任取一点，分别计算出*A*、*B*在该点处的*z*值，若*A*的*z*值小，将*A*加入*M*中；若*A*的*z*值大，则交换*A*和*B*的关系，继续进行*M*中其它多边形的判别

区域细分算法

- 算法原理：考察投影平面上的一块区域，如果可以很“容易”地判断覆盖该区域中的哪个或哪些多边形是可见的，则可按这些多边形的光照属性和几何位置计算确定子区域内各像素的显示颜色；否则就将这块区域细分为若干较小的区域，并把上述推断原则递归地应用到每个较小的区域中去。

区域细分算法

- 多边形的分类



(a)围绕多边形　(b)相交多边形　(c)被包含多边形　(d)分离多边形

图9.10 多边形的投影与考察区域之间的关系

区域细分算法

- 可见性测试
  - 所有多边形均是该区域的分离多边形，于是可直接将该区域中的所有像素点置为背景颜色。
  - 针对该区域，仅存在一个相交多边形，或仅存在一个被包含多边形，或仅存在一个围绕多边形。则可先将该区域中的所有像素点置为背景颜色，再将相应多边形的颜色值填入对应像素点的帧缓存中。

区域细分算法

- 针对该区域，有多于一个的相交多边形、被包含多边形或围绕多边形，则计算所有围绕的、相交的、以及被包含的多边形在该区域4个顶点处的*z*坐标，如果存在一个围绕多边性，它的4个*z*坐标比其它任何多边性的*z*坐标都大（最靠近视点），那么，可将该区域中的所有像素点置为该多边形的颜色值。

区域细分算法

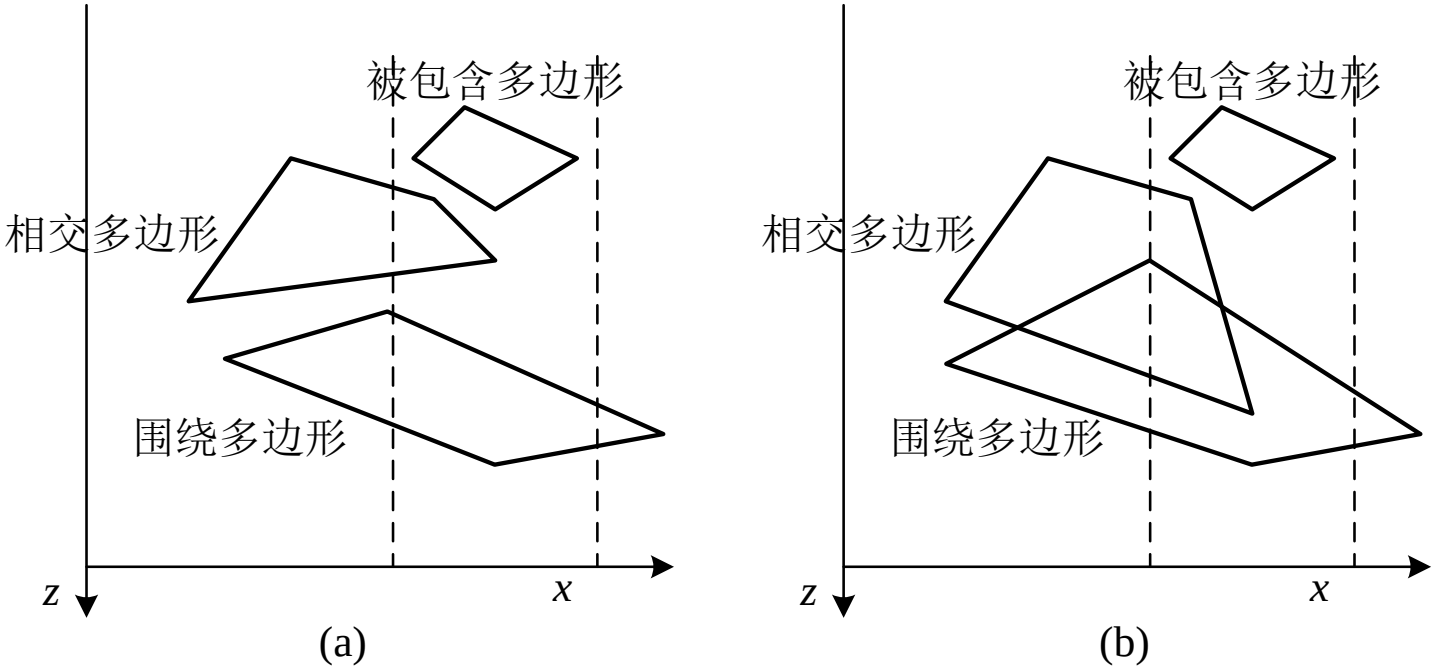


图9.11 满足测试条件3的两个例子

光线投射算法

- 算法原理
  - 其建立在几何光学的基础上，它沿光线的路径追踪可见面，是一种有效的可见性判别技术。其主要从像素出发，逆向追踪射入场景的光线路径。

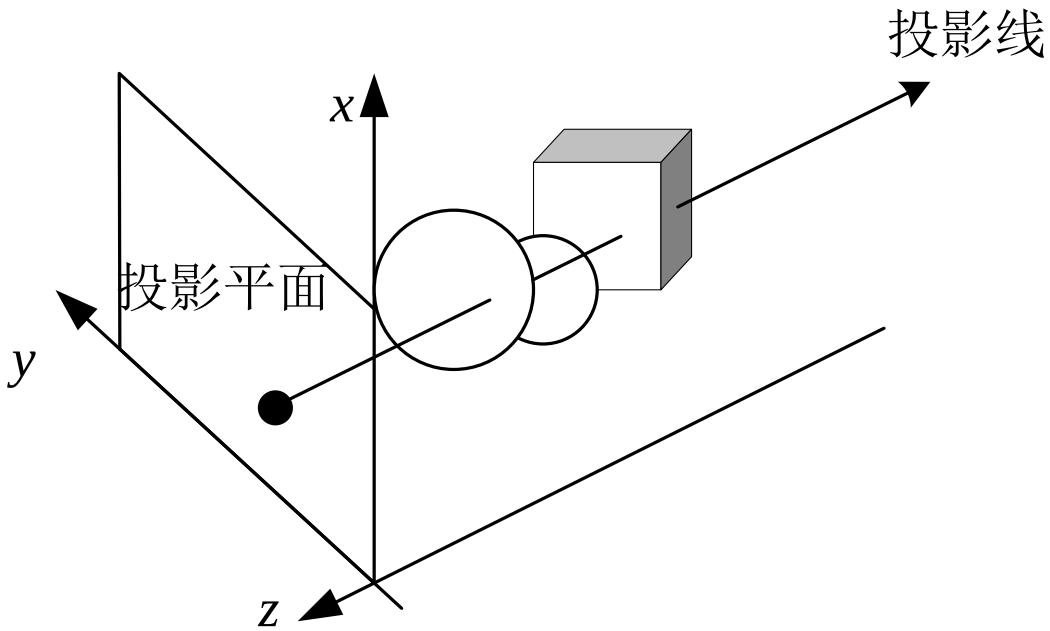


图9.12 光线投射算法

光线投射算法

- 算法步骤
  - 通过视点和投影平面（显示屏幕）上的所有像素点作一入射线，形成投影线。
  - 将任一投影线与场景中的所有多边形求交。
  - 若有交点，则将所有交点按z值的大小进行排序，取出最近交点所属多边形的颜色；若没有交点，则取出背景的颜色。
  - 将该射线穿过的像素点置为取出的颜色。

光线投射算法

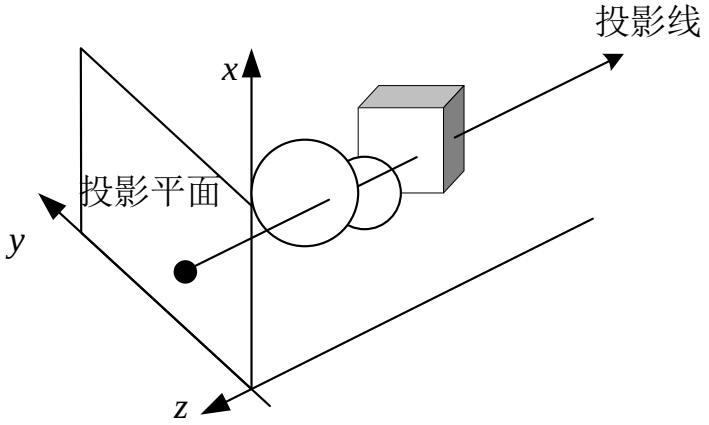
➢光线投射算法可看作是z缓冲器算法的一种变形

✓z缓冲器算法

每次处理一个面片，对面片上的每个投影点计算z值。计算出来的值与以前保存的z值进行比较，从而确定每个像素所对应的可见面片

✓光线投射算法

每次处理一个像素，并沿光线的投射路径计算出该像素所对应的所有面片的z值。



BSP树算法

- 算法原理

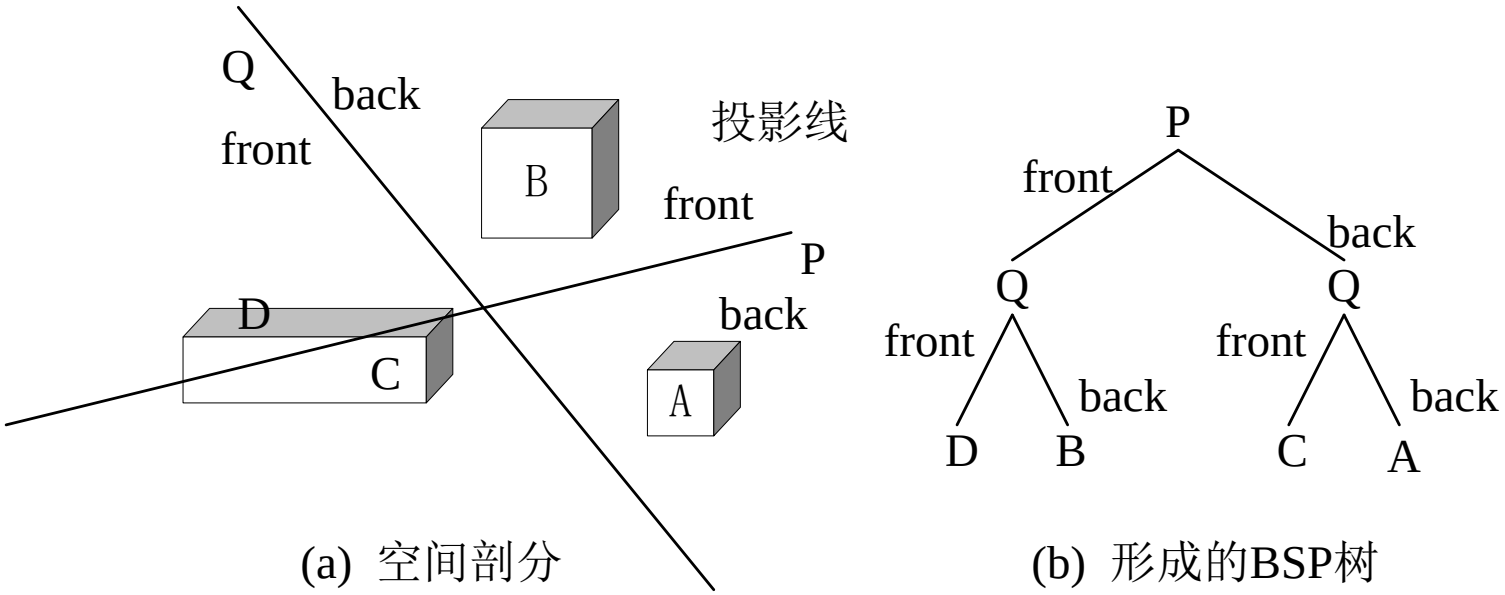


图9.13 BSP树算法原理

多边形区域排序算法

- 算法原理

将多边形按深度值由小到大排序，用前面的可见多边形去切割位于其后的多边形，使得最终每一个多边形要么是完全可见的，要么是完全不可见的。

多边形区域排序算法

- 算法过程

① 初始化：将场景中的多边形全部放入集合*M*中，集合*N*、*L*置为空。

② 若集合*M*中的多边形的个数为1，则从集合*M*中取出该多边形放入*N*中，转（6）。否则对*M*中的所有多边形按$z_{min}(P)$由大到小的顺序进行预排序，放入*L*中

多边形区域排序算法

③ 从L中取出当前深度（z值）最大的多边形作为裁剪多边形P_A，求出该多边形在xoy面上的投影多边形A。且从集合M和L中去掉P_A。

④ 若L为空，则将P_A放入M中，转（2）。　　否则，从L中取出一个多边形P_B，并从集合L中去掉P_B，用A对P_B的投影B进行裁剪，得到B_{in}。

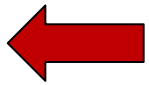
多边形区域排序算法

⑤ 若B_{in}为空，即没有重叠部分，转（4）；否则求出重叠多边形P_{Ain}和P_{Bin}各自顶点的z值，据此比较其深度，以确定P_A是否是离视点较近的多边形。若是，将多边形P_B从M中去掉，将P_{Bout}加入到M中，转（4）；否则，将多边形P_{Aout}放入M中，L置空，转（2）。

⑥ 扫描转换集合M中的多边形。

消隐

1. 消隐的基本概念
2. 消隐的主要算法
3. OpenGL中的消隐



OpenGL中的消隐

- 多边形剔除：主要用于去除多边形物体本身的不可见面，以提高图形系统的性能。

```
glEnable(GL_CULL_FACE);
glCullFace (mode);
```

OpenGL中的消隐

- 深度测试： OpenGL中的深度测试是采用深度缓存器算法，消除场景中的不可见面。
  - 窗口模式中使用深度缓存

    glutInitDisplayMode(GLUT_DEPTH);
  - 启用深度测试

    glEnable(GL_DEPTH_TEST);

OpenGL中的消隐

- 在每次绘制图形时清空深度缓存。

  glClear(GL_DEPTH_BUFFER_BIT);

- 设定深度缓存的范围值

  glDepthRange (nearNormDepth, farNormalDepth);

OpenGL中的消隐

- 设置深度函数

glDepthFunc(func);

- Func表示深度数据更新的方式
- GL_NEVER（不更新）、GL_ALWAYS（始终更新）、GL_LESS（小于，默认值）、GL_LEGUAL（小于等于）、GL_EQUAL（等于）、GL_GEQUAL（大于等于）、GL_GREATER（大于）、GL_NOTEQUAL（不等于）

OpenGL中的消隐

- OpenGL实例

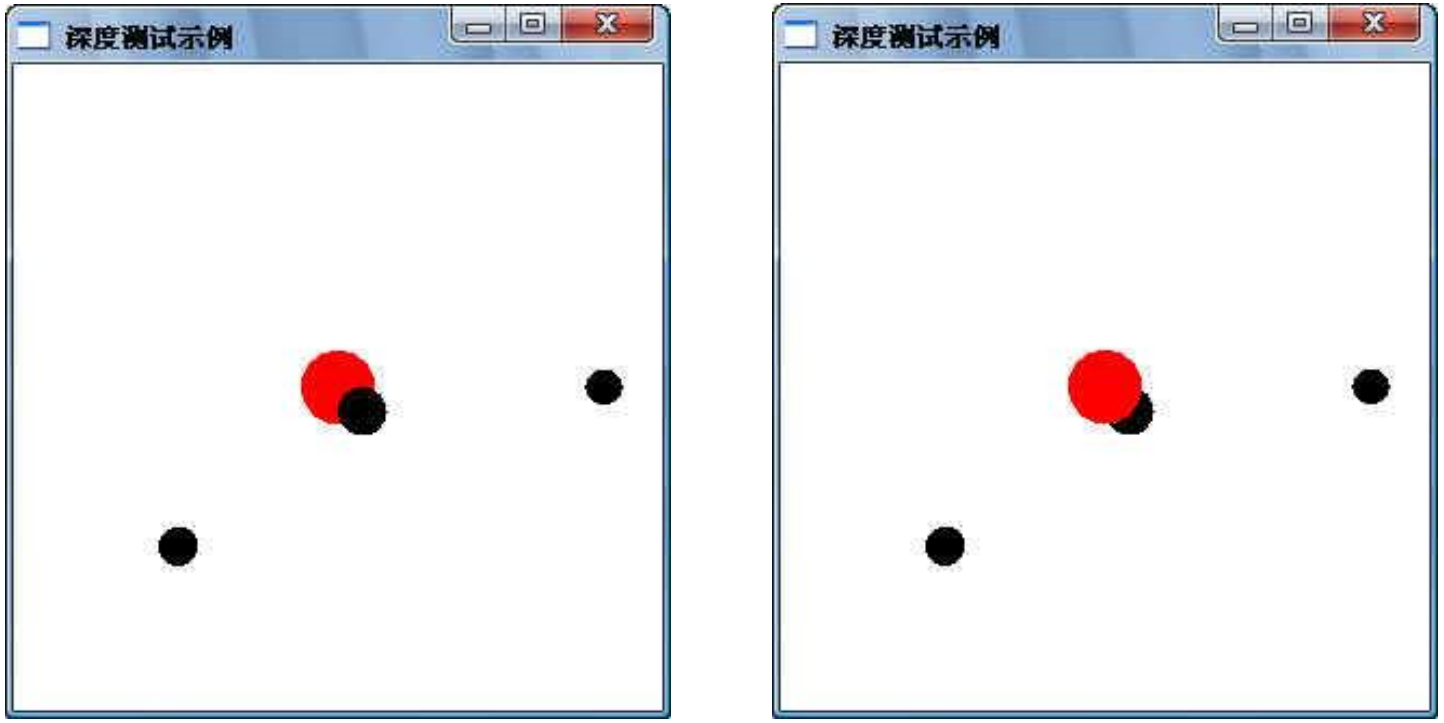


图9.14 实例

谢谢！