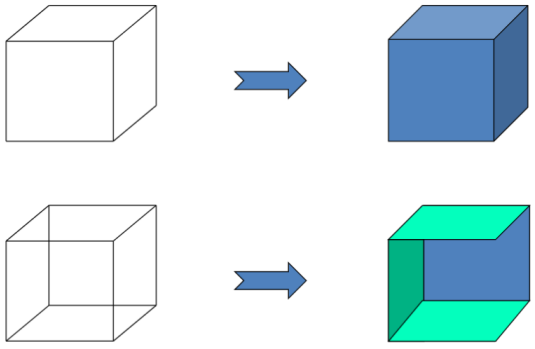
CPT205W12_隐藏面剔除 Hidden-Surface Removal

> 隐藏面剔除与裁剪有很多相同之处，都是尝试删除相机不可见的物体部分；在通过整个管线（光栅化）前，一般会使用可见性测试（visibility test）或遮挡测试（occlusion test）以尽可能多地消除不必要的多边形。

概念

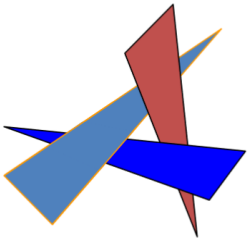


显示所有可见表面，不显示任何被遮挡的表面。（=Visible-surface detection=Hidden-surface elimination）

确定那些表面可见或不可见，例如Z-buffer

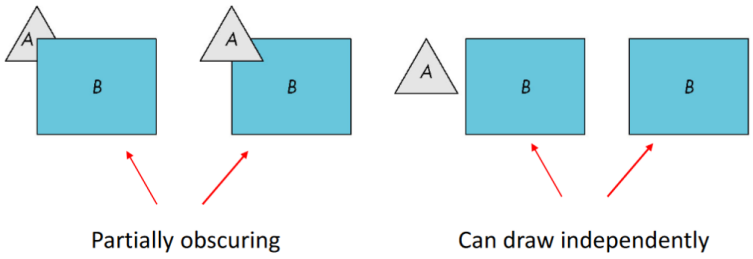
分为两种方法：

- 对象空间法（Object-space methods）：确定那些对象在其他对象的前面。调整大小（resize）不需要重新计算，适用于静态场景，但对象的前后关系往往难以确定（如下图）
- 图像空间法（Image-space methods）：确定每个像素处所可见的物体。但调整大小需要重新计算，适合动态场景。



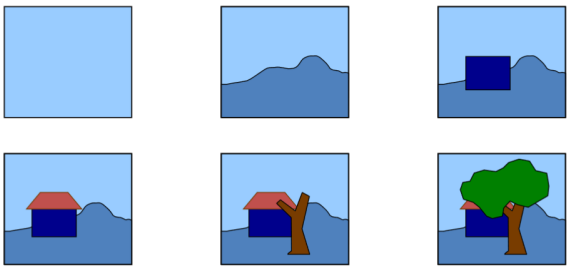
对象空间方法

在多边形（对象）之间使用成对测试（pairwise testing）两两判断遮挡关系，n个多边形的最坏复杂度是$O(n^2)$

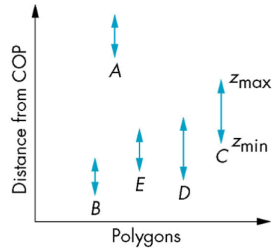


画家算法（Painter's algorithm）

以从后到前的顺序渲染多边形，这样前方的多边形可以简单地覆盖后面的多边形，按深度（z值）对表面（多边形）排序。

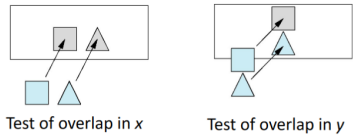


需要先对多边形排序，复杂度$O(nlog(n))$但并非每个多边形的深度都不重叠。先处理简单的情况再处理困难的情况。

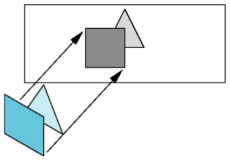


按距投影中心（Center of Projection COP）排序的多边形

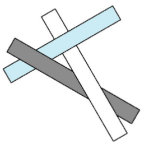
简单的情况：多边形A位于所有其他多边形的后面，深度没有重叠，可以先绘制；此外，在z上重叠但在xy没有重叠（overlap）的多边形，可以独立绘制。



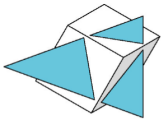
困难的情况：循环重叠（Cyclic overlap），穿透（penetration），在所有方向上重叠（重叠投影）



Overlap in all directions
(polygons with
overlapping projections)



Cyclic overlap



Penetration

背面剔除 Back-face culling

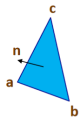
将多边形的位置和方向与观察方向 v 进行比较的过程，其中去除背对摄像机的多边形。最大限度地减少了去除隐藏表面所涉及的计算开销。基本上是确定多边形可见性的测试，基于此测试，如果多边形不可见，则可以将其删除。（=back-surface removal）

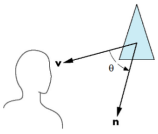
假设对象是实心多面体（solid polyhedron）

计算多边形法线n：逆时针顶点顺序$n = ab \times bc$

若夹角$-90^\circ \leq \theta \leq 90^\circ$即$v \cdot n \geq 0$则此多边形可见

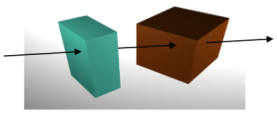
还可将对象转为投影坐标，投影观察方向与z轴平行，那么只需判断法向量在z上分量的正负。

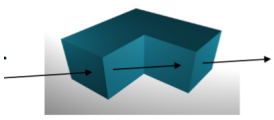




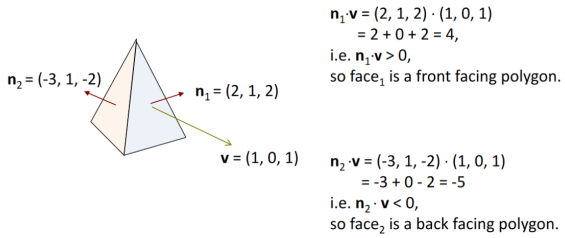
对每个多边形P_i：查找多边形法线n，找视角方向v，若$n \cdot n < 0$，剔除P_i

不适用于：以下原因导致的正面重叠：多个对象，凹对象；非多边形模型；非闭合对象。





计算示例：



图像空间方法

查看每个投影像素（对于$n * m$帧缓冲有$n * m$投影像素）在总共k个多边形中找到离其最近的。

复杂度$O(nmk)$

光线追踪（ray tracing），深度缓冲（Z-buffer）

深度缓冲（Z-buffer）

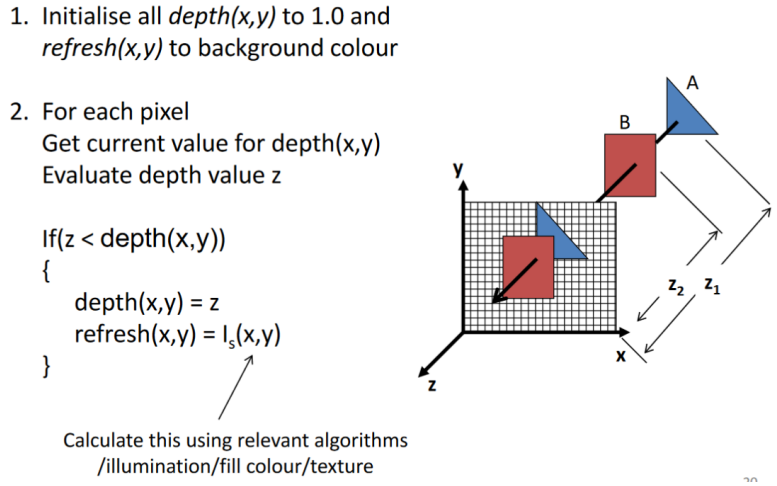
由于某些像素可能包含多个对象，故需要计算这些对象那些是可见的，那些是隐藏的。在图形硬件中，隐藏表面的去除使用Z缓冲算法。

算法留出一个与屏幕大小相同的二维内存数组（Z 缓冲区）（行数 * 列数）。是对颜色缓冲区（帧缓冲区，存储要显示的像素的颜色值）的补充。Z 缓冲区将保存深度值（通常是 z 值）。

初始化Z缓冲，每个元素=远裁剪平面值。初始化颜色缓冲区=背景颜色。

由于每个多边形在屏幕上都有其对应的一组像素，对每个像素将其深度（z值）和Z-Buffer中的值比较，如果小于Buffer中的值则替换。对画面中所有多边形重复操作。

这一实现在归一化坐标中完成，深度值从近剪切平面（near Clipping plane）的0到远剪切平面的1.0。



优点：

- 最广泛使用的隐藏面去除算法；
- 作为图像空间算法，遍历场景并按多边形而不是每个像素进行操作；
- 对多边形逐个光栅化并确定哪些多边形的部分可以绘制在屏幕上；
- 在软件和硬件中相对容易实现。

内存需求：

- 依赖第二个缓冲区（secondary buffer）Z-buffer或叫做depth buffer；
- Z-buffer的长宽等于frame buffer
- 每个单元格包含该像素位置物体的深度信息。

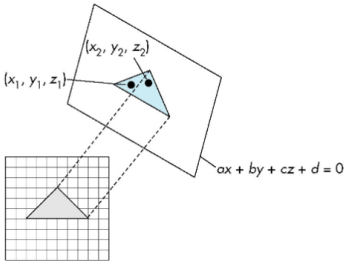
扫描线 Scan-line

如果逐行扫描，我们在扫描线上移动时，深度变化满足$a\Delta x + b\Delta y + c\Delta z = 0$，若多边形平面用$ax + by + cz + d = 0$表示。

通过这种方法可以一次求出一个多边形在屏幕上的深度：

ax + by + cz +d = 0

Along scan line
Δy = 0
Δz = -(a/c)*Δx
In screen space Δx = 1
Only computed once per polygon.

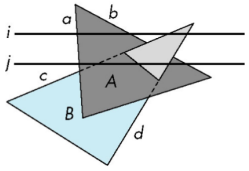


图像空间算法的最坏情况与基元数量成正比；z缓冲区的性能与栅格化生成的片段数量成正比，即取决于栅格化多边形的面积。

通过扫描线算法可以同时处理shading和隐藏面剔除：

扫描线i：无需深度信息，只能用于0个或1个多边形中

扫描线j：需要深度信息，仅当1个或多个多边形中



构造一个数据结构储存：

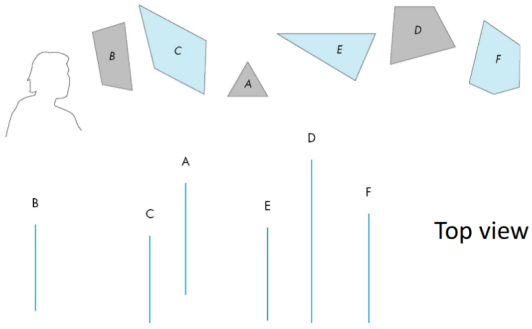
- 每个多边形的flag（inside/outside）
- Incremental structure给扫描线储存遇到的边缘

- 平面的参数

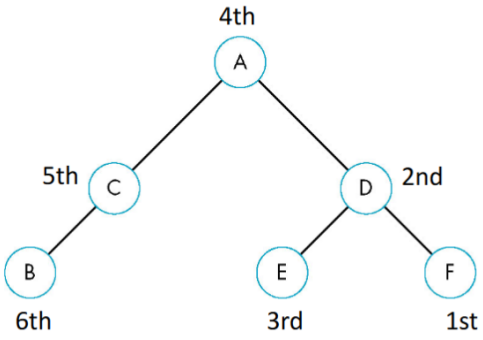
BSP Tree

在许多如游戏的实时应用上，我们希望在程序中消除尽可能多的对象，以便减轻管线负担，减少总线流量。

可以使用二进制空间分区树（nary Spatial Partition (BSP) Tree）区分空间



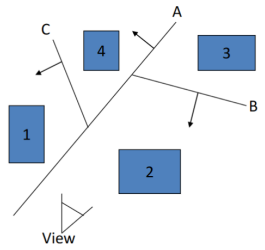
假设有6个平行的平面，平面A隔开了BC和EDF，那我们可以递归地继续：C隔开了B和A，D隔开了E和F。将此信息存于BSP树中，从而进行可见性（visibility，inside）和遮挡（occlusion，outside）测试。

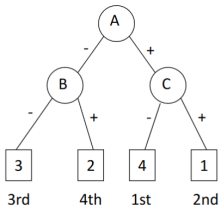


- 流程：选择任意多边形，用其所在平面分割出单元（cell），继续操作，直到单元内仅存1个多边形片段（polygon fragment）。

可以通过其他方式选择分割平面，但目前没有最优算法构建BSP树。故BSP树不唯一，“Optimal”意味着平衡树中多边形片段的最小数量。

- 渲染：渲染过程是BSP树的递归（recursive），你会发现切割面向（包含）视点的一侧的物体是不会被背向视点的物体遮挡的。于是可以进行从后向前的渲染，在每个节点递归到不含视点的一侧来确定多边形，然后绘制多边形，直接覆盖到已绘制的画面上；然后递归到包含视点的一侧继续。





OpenGL函数

去除隐藏面：Opengl使用z-buffer算法，在渲染对象/三角形时保存深度信息，使得只有前方的对象出现在画面中。

Z-buffer

算法使用一个额外的缓冲区，即 z 缓冲区，来存储几何图形沿管道移动时的深度信息。

```
//requested in main()
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)
//enabled in init()
glEnable(GL_DEPTH_TEST)
//cleared in the display callback
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

OpenGL 基本库提供用于背面去除和深度缓冲区可见性测试的功能。还有用于线框显示的隐藏线去除功能，并且可以通过深度提示显示场景。

面剔除 Face-culling

```
glEnable(GL_CULL_FACE);
glCullFace(GLenum)
```

参数：GL_BACK、GL_FRONT 、 GL_FRONT_AND_BACK。故可以删除正面而不是背面，也可以同时删除正面和背面，默认剔除背面。

深度缓冲 Depth-buffer

```
//GLUT initialisation
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
//initialise Depth buffer
glClear(GL_DEPTH_BUFFER_BIT);
//Enable or Disable depth-buffer visibility detection
```

```
glEnable(GL_DEPTH_TEST);
glDisable(GL_DEPTH_TEST);
```

设定

```
glDepthMask(writeStatus);
```

深度缓冲区的状态可以设置为只读或读写，当 writeStatus = GL_FALSE 时，深度缓冲区中的写入模式被禁用，只能检索值进行比较。当使用复杂的背景并显示不同的前景对象时，它很有用。显示透明对象时也很有用。

指定最大深度值

```
glClearDepth(maxDepth);
```

深度值在[0, 1.0]。投影坐标在 [-1.0， 1.0] 范围内进行标准化，近距和远距剪切平面之间的深度值在 [0.0， 1.0] 范围内进一步标准化，其中 0.0 和 1.0 分别对应于近距和远距剪切平面。

调整剪切平面

```
glDepthRange(nearNormDepth, farNormDepth);
```

参数的默认值分别为 0.0 和 1.0，并且它们的值可以在 [0.0， 1.0] 范围内。

```
glDepthFunc(testCondition);
```

参数：L_LESS(default), GL_GREATER, GL_EQUAL, GL_NOTEQUAL, GL_LEQUAL, GL_GEQUAL, GL_NEVER (no points are processed) and GL_ALWAYS (all points are processed)

线框可见性

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
// In this case, both visible and hidden edges are displaye
```

深度提示 Depth-cueing

物体的亮度随其到观察位置的距离而变化

```
glEnable(GL_FOG);
glFog{if}[v](GL_FOG_MODE, GL_LINEAR);
// linear depth function for colour in [0.0, 1.0]
glFog{if}[v](GL_FOG_START, minDepth);
// specifies a different value for dmin
glFog{if}[v](GL_FOG_END, maxDepth);
// specifies a different value for dmax
```

重点理解：glutInitDisplayMode(), glClear(), glEnable(GL_DEPTH_TEST)