



CS 362: Computer Graphics

Hidden Surface Removal



Dr. Samit Bhattacharya
Dept. of Comp. Sc. & Engg.
IIT Guwahati, Assam, India



Why?

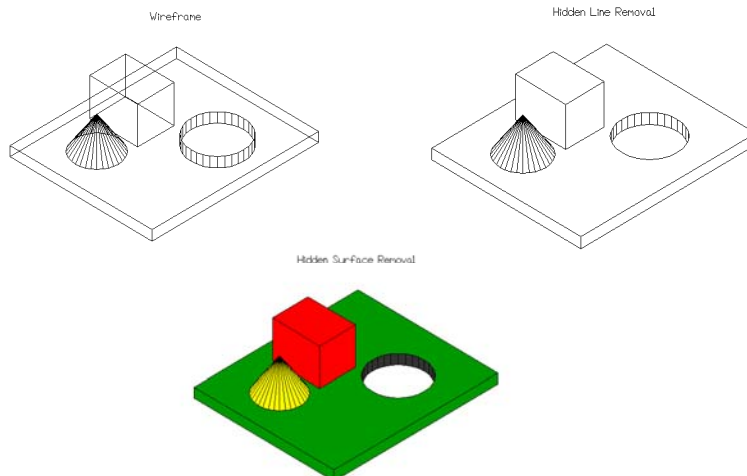
- We must determine what is visible within a scene from a chosen viewing position
- For 3D worlds this is known as **visible surface detection** or **hidden surface removal**




Visibility

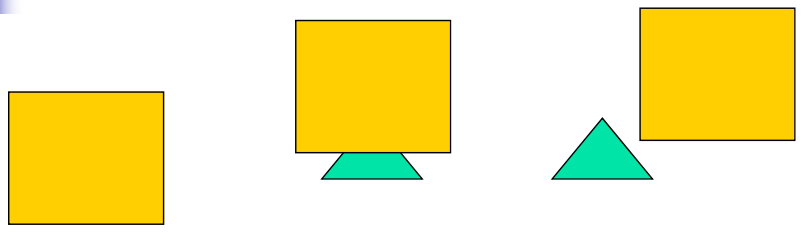
- Assumption: All polygons are **opaque**
- What polygons are visible with respect to view volume?
 - Outside: **Clipping**
 - Remove polygons outside of the view volume
 - Inside: **Hidden Surface Removal**
 - Backface culling: Polygons facing away from the viewer
 - Occlusion: Polygons farther away are obscured by closer polygons (full or partially occluded portions)
- Why should we remove these polygons?
 - Avoid unnecessary expensive operations on these polygons later

HSR Example






Occlusion: Full, Partial, None



Full **Partial** **None**

- The rectangle is closer than the triangle
- Should appear in front of the triangle

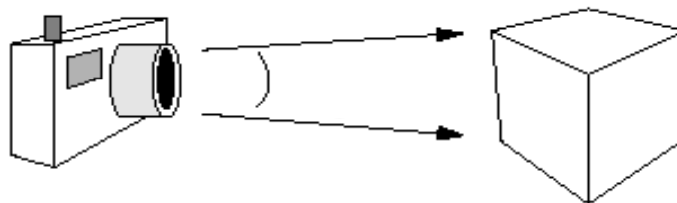


Two Main Approaches

- **Object Space Methods:** Compares objects and parts of objects to each other within the scene definition to determine which surfaces are visible
- **Image Space Methods:** Visibility is decided point-by-point at each pixel position on the projection plane
- Image space methods are more common

Back-Face Culling

- The simplest thing
 - Find the faces on the backs of polyhedra and discard them



Back-Face Culling (cont...)

- We know a point (x, y, z) is behind a polygon surface if:

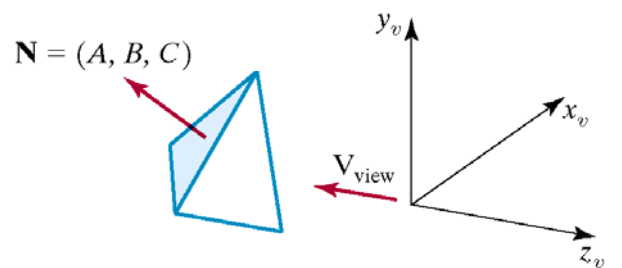
$$Ax + By + Cz + D < 0$$

where A , B , C & D are the plane parameters for the surface

- This can actually be made even easier if we organize things to suit ourselves

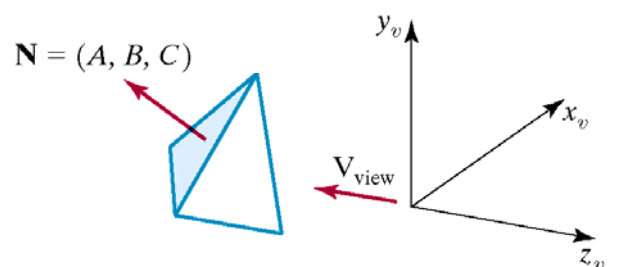
Back-Face Culling (cont...)

- Assume a right handed system with the viewing direction along the negative z -axis
 - If the z component of the polygon's normal is less than zero (i.e. $C < 0$), the surface cannot be seen



Back-Face Culling (cont...)

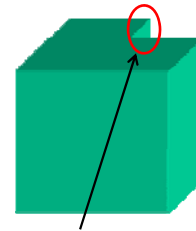
- Assume a right handed system with the viewing direction along the negative z -axis
 - If $C=0$, viewing vector grazing the surface, hence surface is not visible





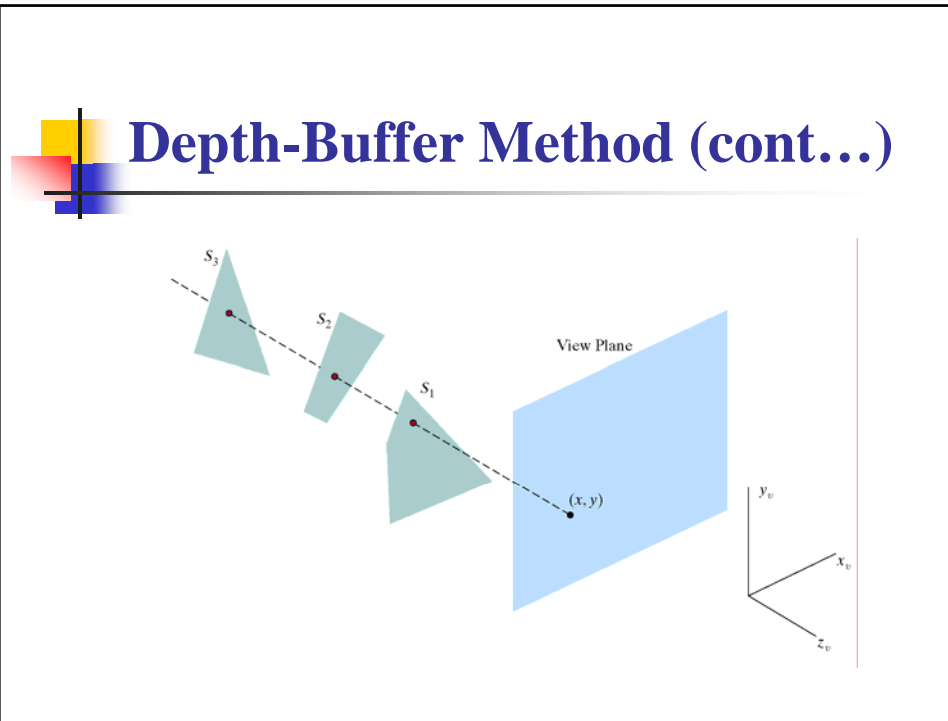
Back-Face Culling (cont...)

- In general, back-face detection can be expected to eliminate about half of the polygon surfaces in a scene from further visibility tests
 - Can be used for pre-processing
- More complicated surfaces though cannot be handled
- We need better techniques to handle **this** kind of situation



Depth-Buffer Method

- Compares surface depth values throughout a scene for each pixel position on the projection plane
 - An image space method
- Usually applied to scenes only containing polygons
- As depth values can be computed easily, this tends to be very fast
- Also called the z-buffer method



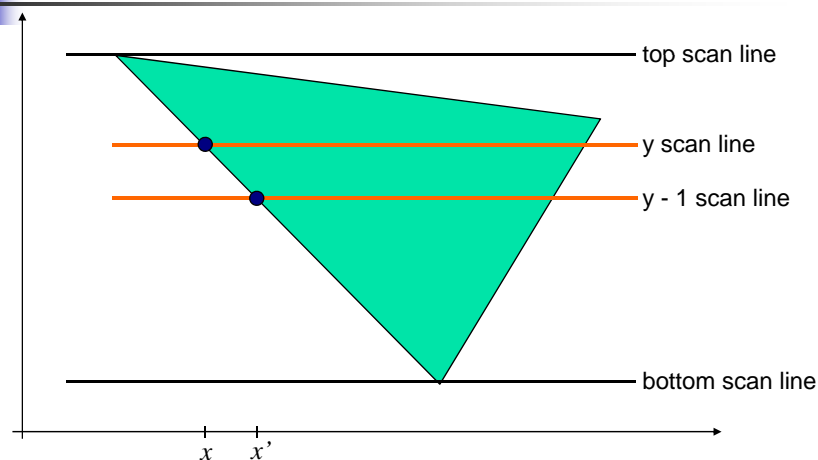
Depth-Buffer Algorithm

- Assumes the presence of extra storage: depth (z) buffer
- Easier to manage if we consider normalized view volume ($-1 \leq z \leq 1$ or $0 \leq z \leq 1$)
 - Otherwise, the depth buffer would have to be of large size to account for any z-value
- The algorithm (next slide) assumes $0 \leq z \leq 1$

Depth-Buffer Algorithm

- Initialize the buffers $\forall i,j$:
 - $\text{DepthBuf}(i,j)=1.0$
 - $\text{FrameBuf}(i,j)=\text{background color}$
- Process each surface in a scene, one at a time
 - For each projected pixel position (i, j) of the surface, calculate the depth z (if not already known)
 - If $z < \text{DepthBuf}(i,j)$
 - $\text{DepthBuf}(i,j) = z$; $\text{FrameBuf}(i,j) = \text{surf color}$
- At the end, the buffers will store the correct values

Iterative Calculation





Iterative Depth Calculation

- At any surface position, depth is calculated from plane equation as:

$$z = \frac{-Ax - By - D}{C}$$

- For any scan line, adjacent x positions differ by ± 1
- Depth at next pixel position (z') along a scan line can be calculated from the current depth (z) as,

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



Iterative Depth Calculation

- The x value for the beginning position on consecutive scan lines can be calculated from the previous one $x' = x - \frac{1}{m}$ where m is the slope

- Depth values along the edge being considered are calculated using

$$z' = z - \frac{A/m + B}{C}$$

- The algorithm starts at the top vertex of the surface and proceeds by iterative calculations



Depth-Sorting Method

- A HSR method that uses both image-space and object-space operations
- Basically, the following two operations are performed
 - Surfaces are sorted on the basis of depth
 - Surfaces are scan-converted in order, starting with the surface of greatest depth
- Also known as the **painter's method**
 - Since it tries to simulate the way an artist draws a scene



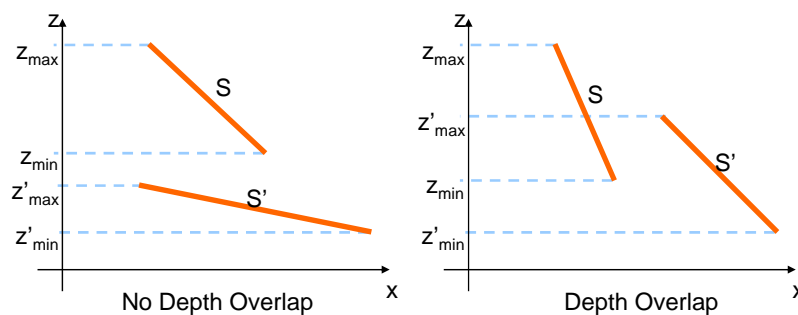
Depth-Sorting Algorithm

- Assume viewing along z direction
- A sorted list L of surfaces is created
 - On the basis of ascending z-value
- The surface S at the end of L (maximum depth) is then compared against all other surfaces
 - To see if there are any depth overlaps

Depth-Sorting Algorithm

- If no overlaps occur, then the surface is scan converted and the process repeats with the surface of next highest depth
- Till all surfaces are checked

Depth Overlapping

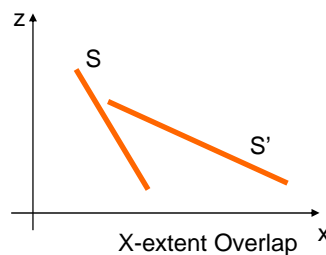


Depth-Sorting Algorithm (cont...)

- When there is depth overlap, we make the following tests (to check for reordering)
 - The bounding rectangles for the two surfaces do not overlap
 - Surface S is completely *behind* the overlapping surface relative to the viewing position
 - The overlapping surface is completely *in front of* S relative to the viewing position
 - The boundary edge projections of the two surfaces onto the view plane do not overlap

Reordering Check

- The bounding rectangles for the two surfaces do not overlap – done in two steps
 - Overlap in the x-extents
 - Overlap in the y-extents

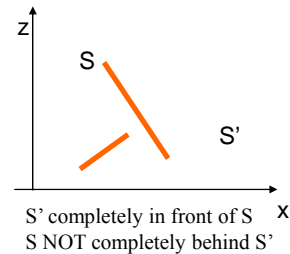
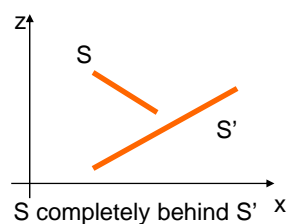


Reordering Check

- Surface S is completely *behind* the overlapping surface S' relative to the viewing position - can be checked using plane equation (PE)
 - Find PE of S', such that its front side is towards the viewing position (normal pointing to the viewing position)
 - Put the vertex coordinates of S on the PE of S'
 - If all the vertices of S is "behind" the plane containing S', S is behind S' ($Ax+By+Cz+D<0$)

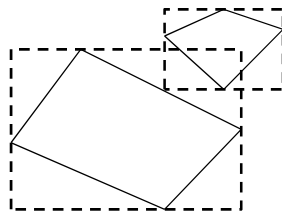
Reordering Check

- The overlapping surface S' is completely *in front of* the surface S relative to the viewing position
 - Can be checked using the plane equation of S and the vertices of S', as before ($Ax+By+Cz+D>0$)

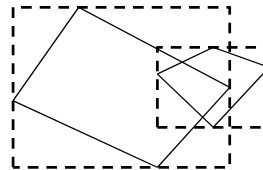


Reordering Check

- The boundary edge projections of the two surfaces onto the view plane do not overlap
 - Although their bounding box may overlap



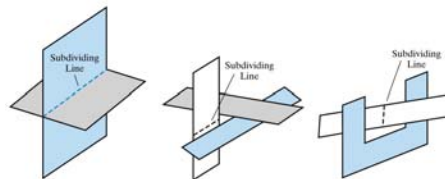
Bounding box overlap but surface don't



Both bounding box and surface overlap

Depth-Sorting Algorithm (cont...)

- The tests are performed in the order listed
- As soon as one is true we move on to the next surface
- If all tests fail, swap the orders of the surfaces
 - Checks should be done for the reordered surfaces again
- Problem: reordering may struck into infinite loop
 - If two/more surfaces alternatively obscure each other





Avoiding Infinite Looping

- Use a flag to indicate if a surface is already reordered once
- If the surface needs to be reordered next time
 - Divide the surface in two to remove cyclic overlap
 - Replace the original surface in the sorted list with the two new surfaces and continue



Scan-Line Method

- An image space method for identifying visible surfaces
- Computes and compares depth values along the various scan-lines for a scene
- Two tables are maintained to get surface information
 - The edge table
 - The surface facet table



Scan-Line Method (cont...)

- The edge table contains
 - Coordinate end points of each line in the scene
 - The inverse slope of each line
 - Pointers into the surface facet table to connect edges to surfaces



Scan-Line Method (cont...)

- The surface facet tables contains
 - The plane coefficients
 - Surface material properties
 - Other surface data
 - Maybe pointers into the edge table



Scan-Line Method (cont...)

- Active edge list is maintained for each scan line
 - Stores edges that cross the scan-line
 - Sorted in order of increasing x (scan line – edge intersection point)



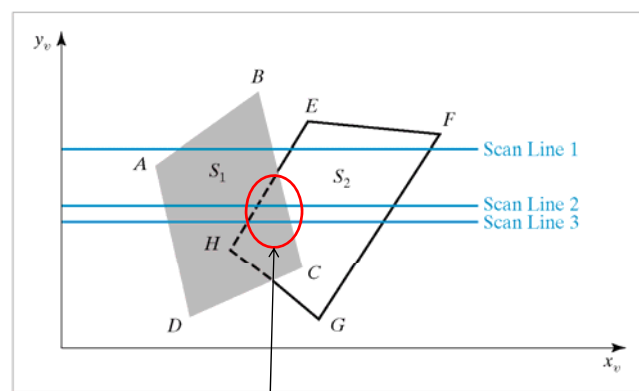
Scan-Line Method (cont...)

- A flag is kept for each surface, to indicate whether a position along a scan-line is either inside or outside the surface
 - Flag = ON at the position, the position is inside
 - Set ON at the leftmost surface edge – scan line intersection point; reset at the rightmost surface edge – scan line intersection point
 - Will it work for concave polygons??

Scan-Line Method (cont...)

- Pixel positions across each scan-line are processed from left to right
- We only need to perform depth calculations when more than one surfaces has its flag turned on at a certain scan-line position
 - In that case, color of the surface with the least depth is set to the pixel

Scan Line Method Example



Need depth checking for pixels in this region