Hidden Surface Removal

1. One of the most challenging problems in computer graphics is the removal of hidden parts from images of solid objects.
2. In real life, the opaque material of these objects obstructs the light rays from hidden parts and prevents us from seeing them.
3. In the computer generation, no such automatic elimination takes place when objects are projected onto the screen coordinate system.
4. Instead, all parts of every object, including many parts that should be invisible are displayed.
5. To remove these parts to create a more realistic image, we must apply a hidden line or hidden surface algorithm to set of objects.
6. The algorithm operates on different kinds of scene models, generate various forms of output or cater to images of different complexities.
7. All use some form of geometric sorting to distinguish visible parts of objects from those that are hidden.
8. Just as alphabetical sorting is used to differentiate words near the beginning of the alphabet from those near the ends.
9. Geometric sorting locates objects that lie near the observer and are therefore visible.
10. Hidden line and Hidden surface algorithms capitalize on various forms of coherence to reduce the computing required to generate an image.
11. Different types of coherence are related to different forms of order or regularity in the image.
12. Scan line coherence arises because the display of a scan line in a raster image is usually very similar to the display of the preceding scan line.
13. **Frame coherence** in a sequence of images designed to show motion recognizes that successive frames are very similar.
14. **Object coherence** results from relationships between different objects or between separate parts of the same objects.
15. A hidden surface algorithm is generally designed to exploit one or more of these coherence properties to increase efficiency.
16. Hidden surface algorithm bears a strong resemblance to two-dimensional scan conversions.

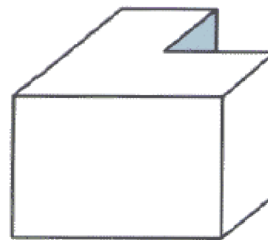
Two main types of Classification:

Object space-
- Determine which part of the object are visible
- Resize dosen’t require recalculation
- May be difficult to determine

Image space
- Determine which object is visible at each pixel
- Resize requires recalculation

Back Face Removal

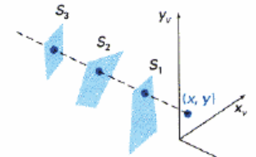
In a solid object, there are surfaces which are facing the viewer (front faces) and there are surfaces
which are opposite to the viewer (back faces). These back faces contribute to approximately half of the total number of surfaces. Since we cannot see these surfaces anyway, to save processing time, we can remove them before the clipping process with a simple test. Each surface has a normal vector. If this vector is pointing in the direction of the center of projection, it is a front face and can be seen by the viewer. If it is pointing away from the center of projection, it is a back face and cannot be seen by the viewer. The test is very simple, if the z component of the normal vector is positive, then, it is a back face. If the z component of the vector is negative, it is a front face. Note that this technique only caters well for non overlapping convex polyhedral. For other cases where there are concave polyhedra or
overlapping objects, we still need to apply other methods to further determine where the obscured faces are partially or completely



hidden by other objects (eg.Using Depth-Buffer Method or Depth-sort Method).

Depth-Buffer Method (Z-Buffer Method)

This approach compare surface depths at each pixel

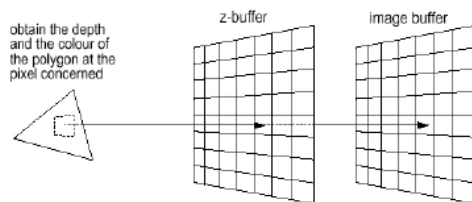


At view-plane position (x, y), surface S_1 has the smallest depth from the view plane and so is visible at that position.

position on the projection plane.

Object depth is usually measured from the view plane

along the z axis of a viewing system. This method requires 2 buffers: one is the image buffer and the other is called the z-buffer (or the depth buffer). Each of these buffers has the same resolution as the image to be



captured. As surfaces are processed, the image buffer is used to store the color values of each pixel position and the z-buffer is used to store the depth values for each (x,y) position.

Algorithm:

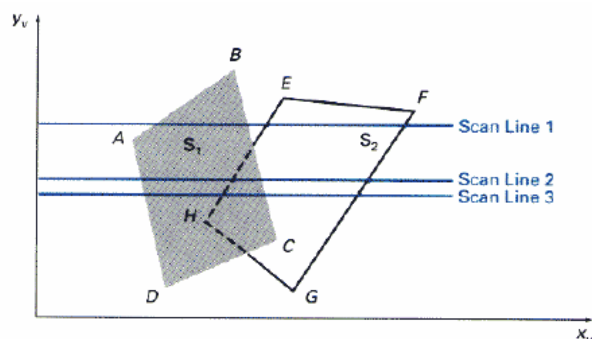
1. Initially each pixel of the z-buffer is set to the maximum depth value (the depth of the back clipping plane).
2. The image buffer is set to the background color.
3. Surfaces are rendered one at a time.
4. For the first surface, the depth value of each pixel is calculated.
5. If this depth value is smaller than the corresponding depth value in the z-buffer (ie. it is closer to the view point), both the depth value in the z-buffer and the color value in the image buffer are replaced by the depth value and the color value of this surface calculated at the pixel position.

6. Repeat step 4 and 5 for the remaining surfaces.
7. After all the surfaces have been processed, each pixel of the image buffer represents the color of a visible surface at that pixel. This method requires an additional buffer (if compared with the Depth-Sort Method) and the overheads involved in updating the buffer. So this method is less attractive in the cases where only a few objects in the scene are to be rendered.
- Simple and does not require additional data structures.
- The z-value of a polygon can be calculated incrementally.
- No pre-sorting of polygons is needed.

- No object-object comparison is required.
- Can be applied to non-polygonal objects.
- Hardware implementations of the algorithm are available in some graphics workstation.
- For large images, the algorithm could be applied to, eg., the 4 quadrants of the image separately, so as to reduce the requirement of a large additional buffer

Scan-Line Method

In this method, as each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible. Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view plane. When the visible surface has been determined, the intensity value for that position is entered into the image buffer.

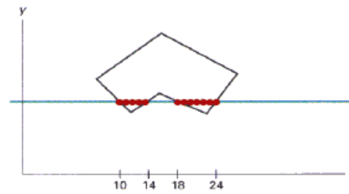


Scan lines crossing the projection of two surfaces, S_1 and S_2, in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

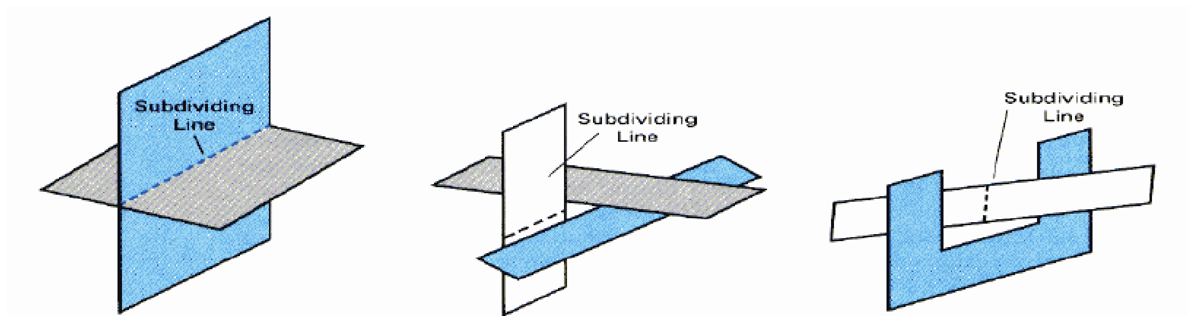
```
For each scan line do
Begin
   For each pixel (x,y) along the scan line do ------------ Step 1
Begin
  z_buffer(x,y) = 0
   Image_buffer(x,y) = background_color
End
  For each polygon in the scene do ----------- Step 2
Begin
   For each pixel (x,y) along the scan line that is covered by the polygon do
Begin
   2a. Compute the depth or z of the polygon at pixel location (x,y).
   2b. If z < z_buffer(x,y) then
   Set z_buffer(x,y) = z
    Set Image_buffer(x,y) = polygon's colour

End
End
End
```

- Step 2 is not efficient because not all polygons necessarily intersect with the scan line.
- Depth calculation in 2a is not needed if only 1 polygon in the scene is mapped onto a segment of
the scan line.
- To speed up the process:

Recall the basic idea of polygon filling: For each scan line crossing a polygon, this algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are sorted from left to right. Then, we fill the pixels between each intersection pair.



With similar idea, we fill every scan line span by span. When polygon overlaps on a scan line, we perform depth calculations at their edges to determine which polygon should be visible at which span. Any number of overlapping polygon surfaces can be processed with this method. Depth calculations are performed only when there are polygons overlapping. We can take advantage of coherence along the scan lines as we pass from one scan line to the next. If no changes in the pattern of the intersection of polygon edges with the successive scan lines, it is not necessary to do depth calculations. This works only if surfaces do not cut through or otherwise cyclically overlap each other. If cyclic overlap happens, we can divide the surfaces to eliminate the overlaps.

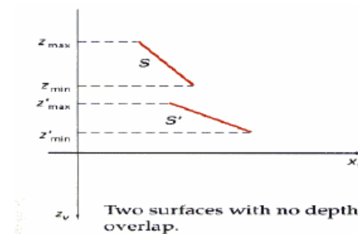


- The algorithm is applicable to non-polygonal surfaces (use of surface and active surface table, zvalue
is computed from surface representation).
- Memory requirement is less than that for depth-buffer method.
- Lot of sortings are done on x-y coordinates and on depths.

1. Sort all surfaces according to their distances from the view point.
2. Render the surfaces to the image buffer one at a time starting from the farthest surface.
3. Surfaces close to the view point will replace those which are far away.
4. After all surfaces have been processed, the image buffer stores the final image.

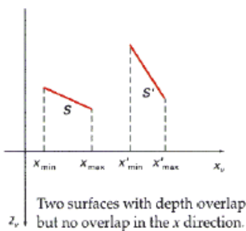
The basic idea of this method is simple. When there are only a few objects in the scene, this method can be very fast. However, as the number of objects increases, the sorting process can become very complex and time consuming.

Example: Assuming we are viewing along the z axis. Surface S with the greatest depth is then compared to other surfaces in the list to determine whether there are any overlaps in depth. If no depth

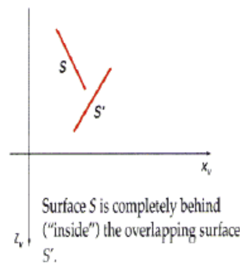


Two surfaces with no depth overlap.

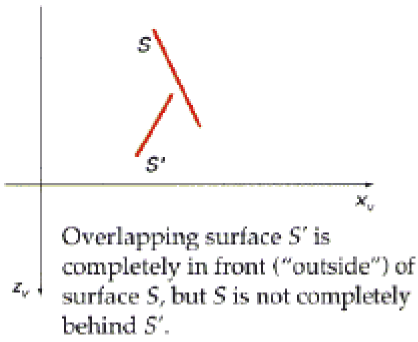
overlaps occur, S can be scan converted. This process is repeated for the next surface in the list. However, if depth overlap is detected, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.



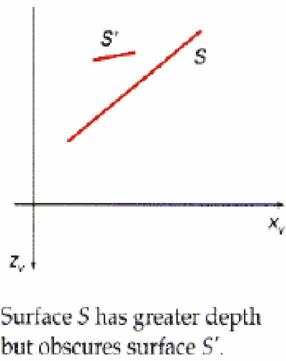
Two surfaces with depth overlap but no overlap in the x direction.



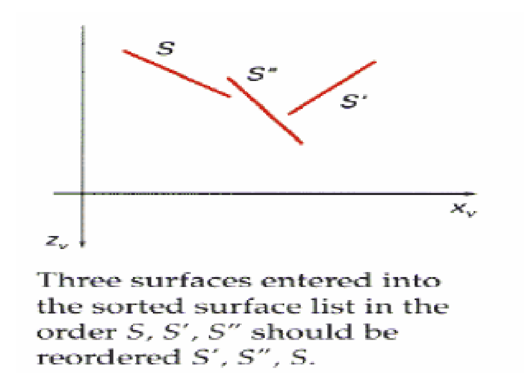
Surface S is completely behind ("inside") the overlapping surface S'.



Overlapping surface S′ is completely in front ("outside") of surface S, but S is not completely behind S′.



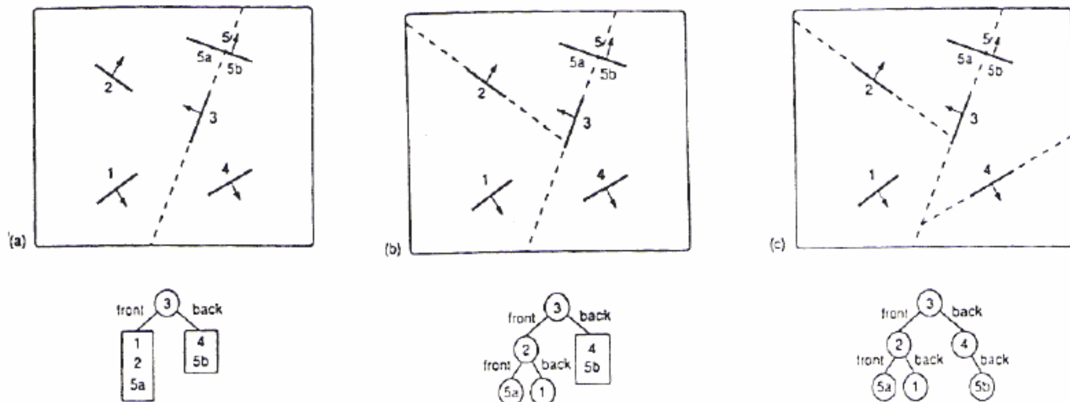
Surface S has greater depth but obscures surface S′.



Three surfaces entered into the sorted surface list in the order S, S′, S″ should be reordered S′, S″, S.

Binary Space Partitioning (BSP)

- suitable for a static group of 3D polygon to be viewed from a number of view points
- based on the observation that hidden surface elimination of a polygon is guaranteed if all polygons on the other side of it as the viewer is painted first, then itself, then all polygons on the same side of it as the viewer



1.The algorithm first build the BSP tree:

- a root polygon is chosen (arbitrarily) which divides the region into 2 half-spaces (2 nodes => front and back)
- a polygon in the front half-space is chosen which divides the half-space into another 2 halfspaces

- the subdivision is repeated until the half-space contains a single polygon (leaf node of the tree)
- the same is done for the back space of the polygon.

2.To display a BSP tree:

- see whether the viewer is in the front or the back half-space of the root polygon.
- if front half-space then first display back child (subtree) then itself, followed by its front child / subtree
- the algorithm is applied recursively to the BSP tree.

BSP Algorithm

Procedure DisplayBSP(tree: BSP_tree)

Begin

If tree is not empty then

If viewer is in front of the root then

Begin

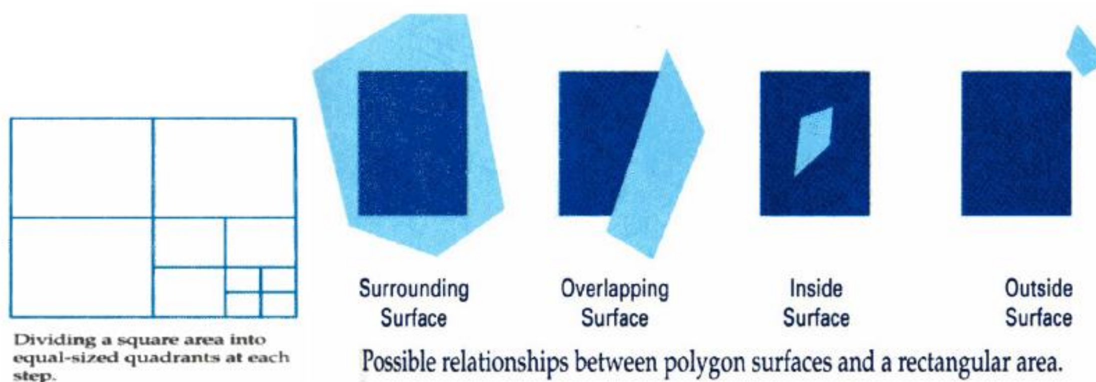
DisplayBSP(tree.back_child)

displayPolygon(tree.root)

```
DisplayBSP(tree.front_child)
End
Else
Begin
DisplayBSP(tree.front_child)
displayPolygon(tree.root)
DisplayBSP(tree.back_child)
End
End
```

Area Subdivision Algorithms

The area-subdivision method takes advantage of area coherence in a scene by locating those view areas that represent part of a single surface. The total viewing area is successively divided into smaller and smaller rectangles until each small area is simple, ie. it is a single pixel, or is covered wholly by a part of a single visible surface or no surface at all.



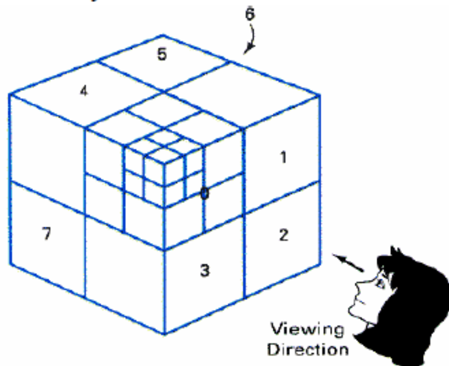
Dividing a square area into equal-sized quadrants at each step.

Possible relationships between polygon surfaces and a rectangular area.

The procedure to determine whether we should subdivide an area into smaller rectangle is:
1. We first classify each of the surfaces, according to their relations with the area:
Surrounding surface - a single surface completely encloses the area Overlapping surface - a single surface that is partly inside and partly outside the area Inside surface - a single surface that is completely inside the area Outside surface - a single surface that is completely outside the area. To improve the speed of classification, we can make use of the bounding rectangles of surfaces for early confirmation or rejection that the surfaces should be belong to that type.
2. Check the result from 1., that, if any of the following condition is true, then, no subdivision of this area is needed.
a. All surfaces are outside the area.
b. Only one surface is inside, overlapping or surrounding surface is in the area.
c. A surrounding surface obscures all other surfaces within the area boundaries.
For cases b and c, the color of the area can be determined from that single surface.

Octree Methods

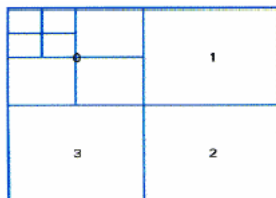
In these methods, octree nodes are projected onto the viewing surface in a front-to-back order. Any surfaces toward the rear of the front octants (0,1,2,3) or in the back octants (4,5,6,7) may be hidden by the front surfaces.



Octants in Space

With the numbering method (0,1,2,3,4,5,6,7), nodes representing octants 0,1,2,3 for the entire region are visited before the nodes representing octants 4,5,6,7. Similarly the nodes for the front four suboctants of octant 0 are visited before the nodes

for the four back suboctants.When a colour is encountered in an octree node, the corresponding



Quadrants for the View Plane

pixel in the frame buffer is painted only if no previous color has been loaded into the same pixel position. In most cases, both a front and a back octant must be considered in determining the correct color values for a quadrant. But

- If the front octant is homogeneously filled with some color, we do not process the back octant.
- If the front is empty, it is necessary only to process the rear octant.
- If the front octant has heterogeneous regions, it has to be subdivided and the sub-octants are handled recursively.