Last updated: Yeh-Liang Hsu (2010-12-12).

Note: This is the course material for "ME550 Geometric modeling and computer graphics," Yuan Ze University. Part of this material is adapted from *CAD/CAM Theory and Practice*, by Ibrahim Zeid, McGraw-Hill, 1991. This material is be used strictly for teaching and learning of this course.

Hidden Line and Surface

1. Vector graphics and raster graphics

Vector graphic devices were introduced in the mid 1960s. The display processor reads the display list, the list of graphic commands sent from the application program, stored in the display unit and cast the electron beams onto the display device. Though vector devices can achieve high resolution (e.g., 4096 by 4096), the cost of such devices is high.

Autodesk launched AutoCAD in 1982, which is the first commercially successful 2D vector-based drafting program. **Vector graphics employs geometric elements like points, lines, curves, and polygons to represent images**. Since these elements are defined mathematically, they can be stored in a database and later manipulated, for example, copied, moved, rotated, scaled, arrayed.

Raster graphic device, which were introduced in the mid 1970s, is now the main stream of graphic display devices. The display processor receives the graphics commands from the application programs, converts them into a pixel-by-pixel representation, and stores those raster images in the frame buffer. The display processor also reads the content of the buffer and casts the electron beams onto phosphor coating inside the surface of the display device. Light emission of the phosphor lasts only a brief time, and thus refresh is required, usually every 1/30 seconds for normal TV sets or 1/60 second for high-end raster devices. The electron beam is repeatedly scanned to the right, with scan lines looping downwards. Since the total number of pixels

is fixed, the time required for the refresh will be constant regardless of the complexity of the image to be drawn.

The raster image stored in the frame buffer may carry gray-level of color information if more than one bit is allocated for each pixel of the raster image. Consider using 3 bits for each pixel, the first bit represents the on/off state for the color red, the second bit for green, and the third bit for blue. Eight colors can be defined and displayed simultaneously on the display device. Because of the declining price of memory chips, raster graphic devices with 24-bit per pixel (8 each for red, green, and blue) tend to be prevalent. In those devices, 256 (2^8) different levels can be defined for each color and a total of 16,777,216 (2^{24}) colors can be defined and display simultaneously on the display device.

2. Visibility techniques

The problem of removing hidden edges and surfaces can also be viewed as a visibility problem. Therefore, a clear understanding of it and its solution is useful and can be extended to solve relevant engineering problems. Consider, for example, the vision and path planning problems in robotics applications. In the path planning problem, the knowledge of when a given surface changes from visible to hidden can be utilized to find the minimum path of the robot end effector. Another example is the display of finite element meshes where the hidden elements are removed.

Hidden line and hidden surface algorithms have been classified as object-space methods, image-space methods, or a combination of both (hybrid methods). An object-space algorithm utilizes the spatial and geometrical relationships among the objects in the scene to determine hidden and visible parts of these objects. An image-space algorithm, on the other hand, concentrates on the final image to determine what is visible, say, within each raster pixel in the case of raster displays. Most hidden surface algorithms use raster image-space methods while most hidden line algorithms use object-space methods.

The visibility of parts of objects of a scene depends on the location of the viewing eye, the viewing direction, the type of projection (orthogonal or perspective), and the depth or the distance from various faces of various objects in the scene to the viewing eye.

The depth comparison is the central criterion utilized by hidden line algorithms to determine visibility. The depth comparison determines if a projected point $\mathbf{P_{1v}}(x_{1v}, y_{2v})$ in a given view obscures another point $\mathbf{P_{2v}}(x_{2v}, y_{2v})$. This is equivalent to determining of the

two original corresponding points $\mathbf{P}_1$ and $\mathbf{P}_2$ lie on the same projector. For orthographic projections, projectors are parallel. Therefore, two points $\mathbf{P}_1$ and $\mathbf{P}_2$ are on the same projector if $x_{1v} = x_{2v}$ and $y_{1v} = y_{2v}$. If they are, a comparison of z_{1v} and z_{2v} decides which point is closer to the viewing eye.

Visibility techniques attempt to establish relationships among polygons and edges in the viewing plane. The techniques normally check for overlapping of pairs of polygons (sometimes referred to as lateral comparisons) in the viewing plane (the screen). If overlapping occurs, depth comparisons are used to determine if part or all of one polygon is hidden by another.

◇Assignment 1

Generate two overlapping objects using your CAD software. Display the objects with hidden lines removed. Display the objects using different location of the viewing eye, different viewing direction, and different type of projection (orthogonal or perspective). Discuss your findings. Is there any special functions provided by your CAD software? ◇

2.1 Minimax test

Minimax test (also called the overlap or bounding box test) checks if two polygons overlap. The test provides a quick method to determine if two polygons do not overlap. It surrounds each polygon with a box by finding its extents (minimum and maximum *x* and *y* coordinates) and then checks for the intersection for any two boxes in both the *X* and *Y* directions. If two boxed do not intersect, their corresponding polygons do not overlap (see Figure 1). In such a case, no further testing of the edges of the polygons is required.

If the minimax test fails (two boxes intersect), the two polygons may or may not overlap, as shown in Figure 1. Each edge of one polygon is compared against all the edges of the other polygon to detect intersections. The minimax test can be applied first to any two edges to speed up this process.

◇Assignment 2

Generate an example similar to those in Figure 1 to show that the two polygons may not overlap when the two boxes intersect. Complete the minimax test comparing the edges of the polygons. ◇

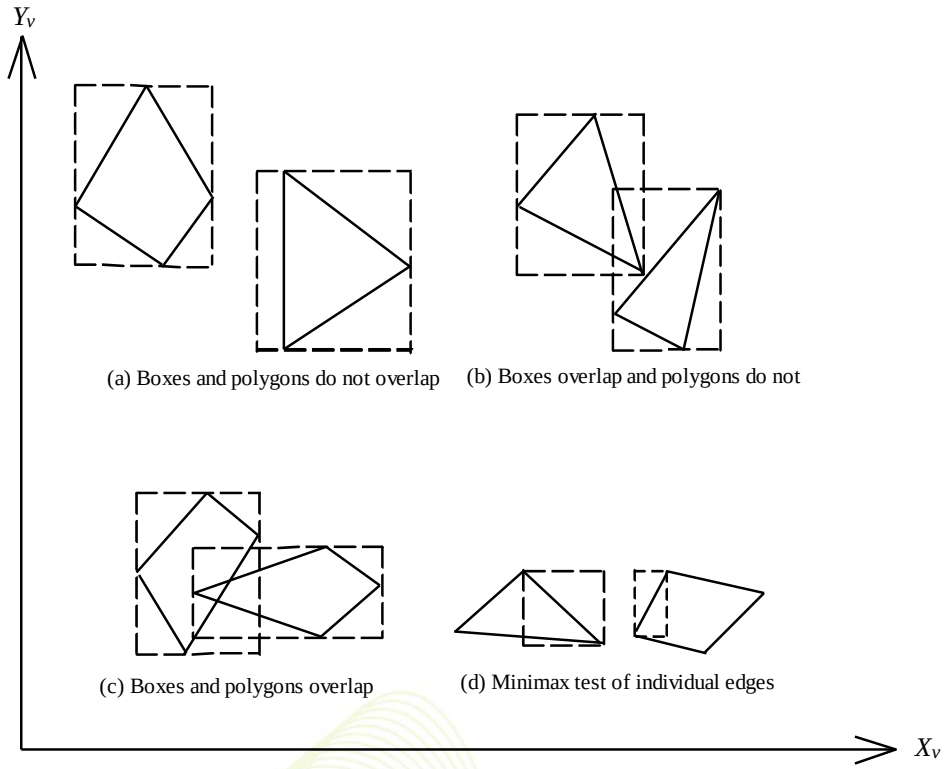


Figure 1. Minimax tests for typical polygons and edges

2.2 Containment test

The containment test checks whether a given point lies inside a given polygon or polyhedron. There are three methods to compute containment or surroundedness. For a convex polygon, one can substitute the x_v and y_v coordinates of the point into the line equation of each edge. If all substitutions result in the same sign, the point is on the same side of each edge and is therefore surrounded. This test requires that the signs of the coefficients of the line equations be chosen correctly.

For nonconvex polygons, two other methods can be used. In the first method, we draw a line from the point under testing to infinity as shown in Figure 2a. The semi-infinite line is intersected with the polygon edges. If the intersection count is even, the point is outside the polygon ($\mathbf{P}_2$ in Figure 2a). If it is odd, the point is inside ($\mathbf{P}_1$ in the Figure). If one of the polygon edges lies on the semi-infinite line, a singular case arises which needs special treatment to guarantee the consistency of the results. The second method for nonconvex polygons (Figure 2b) computes the sum of the angles subtended by each of the oriented edges as seen from the test point. If the sum is zero, the point is outside the polygon. If the sum is 2π or -2π, the point is inside. The minus sign reflects whether the vertices of the polygon are ordered in a clockwise direction instead of counterclockwise.

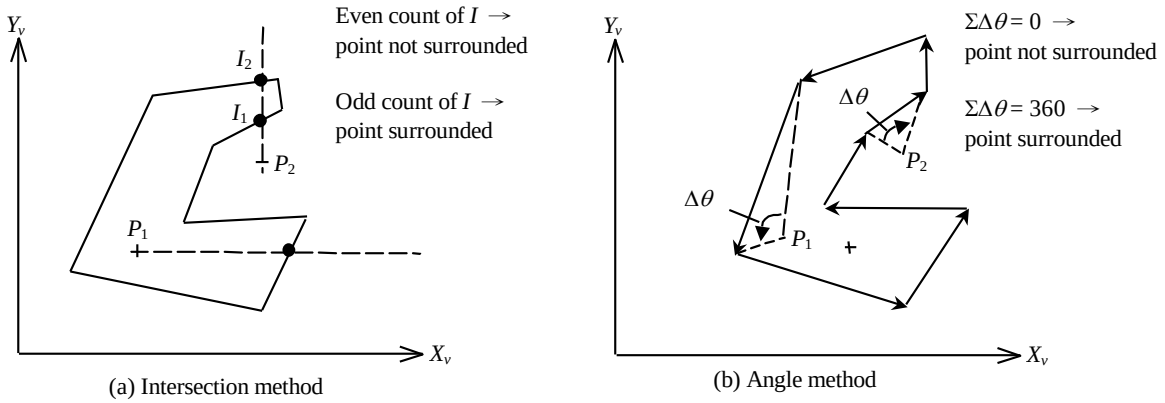


Figure 2. Containment test for nonconvex polygons

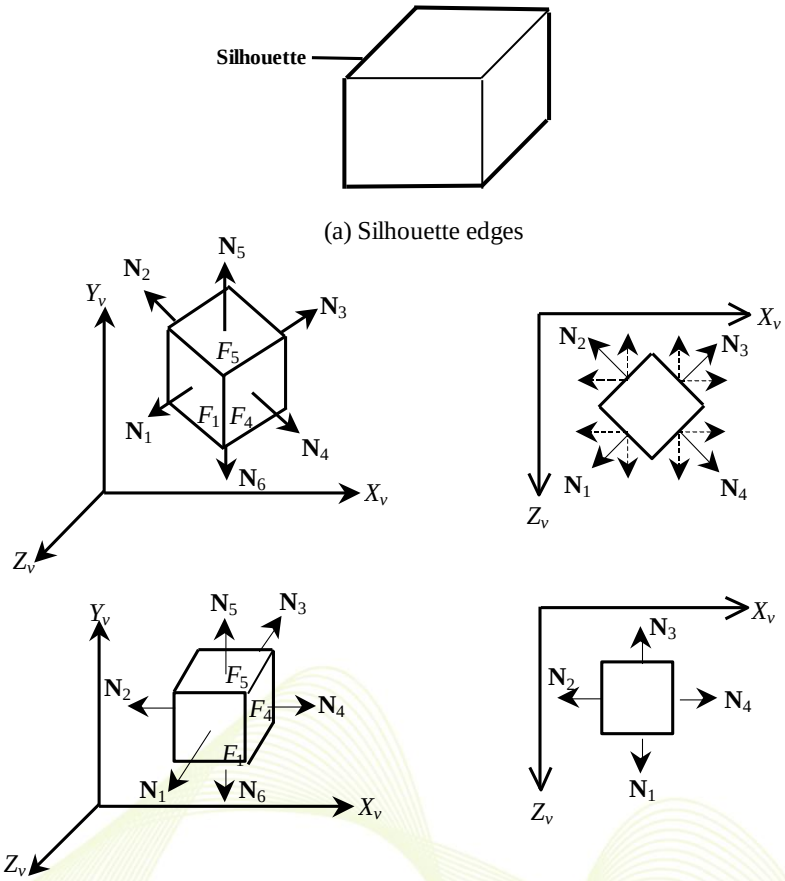
3. Hidden line algorithms

3.1 Computing silhouettes

A set of edges that separates visible faces from invisible faces of an object with respect to a given viewing direction is called silhouette edges (or silhouettes). The signs of the Z_v components of normal vectors of the object faces can be utilized to determine the silhouette. An edge that is part of the silhouette is characterized as the intersection of one visible face and one invisible face. An edge that is the intersection of two visible faces is visible, but does not contribute to the silhouette. The intersection of two invisible faces produces an invisible edge. Figure 3a shows the silhouette of a cube. Figure 4 shows the silhouette curve of a curved surface.

◇Assignment 3

Generate a curved surface using your CAD software. Try to see if you can display its silhouette curve. ◇



(a) Silhouette edges

(b) Determining silhouette edges

Figure 3. Silhouette edges of a polyhedral object.

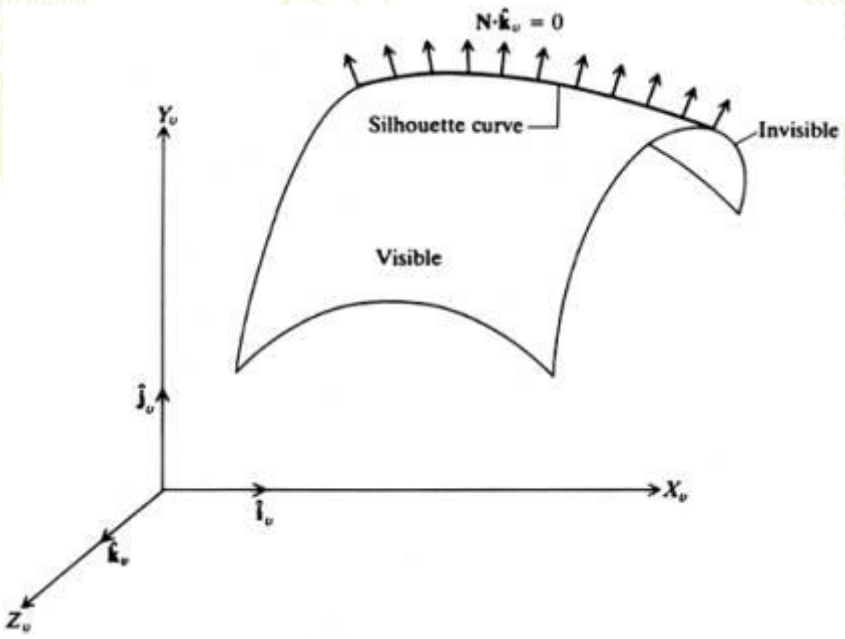


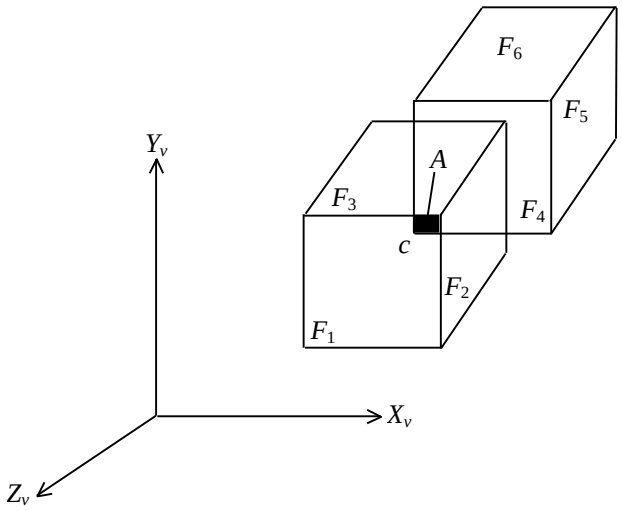
Figure 4. Silhouette curve of a curved surface

3.2 The priority algorithm

This algorithm is also known as the depth or *z* algorithm. The algorithm is based on sorting all the faces (polygons) in the scene according to the largest *z* coordinate value of each. This step is sometimes known as assignment of priorities. If a face intersects more than one face, other visibility tests besides the *z* depth are needed to resolve any ambiguities.

The priority algorithm is based on assigning priorities to the faces in the face list. The priority assignment is determined by comparing two faces at any one time. The priority list is continuously changed and the final list is obtained after few iterations. Here is how priorities can be assigned. Figure 5b shows the changes of the priority list of the objects in Figure 5a. The first face in the face list (F_1 in Figure 5b) is assigned the highest priority 1. F_1 is intersected with the other faces in the list, that is, F_2 to F_6. The intersection between F_1 and another face may be an area as in the case of F_1 and F_4, an edge as for faces F_1 and F_2, or an empty set (no intersection) as for faces F_1 and F_6. In the case of an area of intersection (*A* in Figure 5a), the (x_v, y_v) coordinates of a point *C* inside *A* can be computed (notice the corner points of *A* are known). For both faces F_1 and F_4, the two corresponding z_v values of point *C* can be calculated and compared. The face with the highest z_v values is assigned the highest priority. In the case of an edge of intersection, both faces are assigned the same priority. They obviously do not obscure each other, especially after the removal of the back faces. In the case of no face intersection, no priority is assigned.

Let us apply the above stategy to the scene of Figure 5. F_1 intersects F_2 and F_3 in edges. Therefore both faces are assigned priority 1. F_1 and F_4 intersect in an area. Using the depth test, and assuming the depth of F_4 is less than that of F_1, F_4 is assigned priority 2. When we intersect faces F_1 and F_5, we obtain an empty set, that is, no priority assignment is possible. In this case, the face F_1 is moved to the end of the face list and the sorting process to determine priorities starts all over again. In each iteration, the first face in the face list is assigned priority 1. The end of each iteration is detected by no intersection. Figure 5b shows four iterations before obtaining the final priority list. In iteration 4, faces F_4 to F_6 are assigned the priority 1 first. When F_4 is intersected with F_1, the depth test shows that F_1 has higher priority. Thus, F_1 is assigned priority 1 and priority of F_4 to F_6 is dropped to 2.



(a) Scene of two boxes

Face list	Priority list
F_1	1
F_2	1
F_3	1
F_4	2
F_5	
F_6	

Iteration 1

Face list	Priority list
F_2	1
F_3	1
F_4	2
F_5	
F_6	
F_1	

Iteration 2

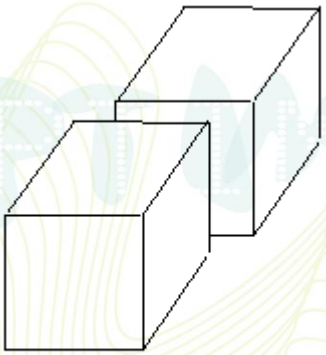
Face list	Priority list
F_3	1
F_4	2
F_5	
F_6	
F_1	
F_2	

Iteration 3

Face list	Priority list
F_4	2
F_5	2
F_6	2
F_1	1
F_2	1
F_3	1

Iteration 4

(b) Assignment of priorities



(c) Image with hidden lines removed

Figure 5. The priority algorithm.

Reorder the face and priority lists so that the highest priority is on top of the list. In this case, the face and priority lists are $[F_1, F_2, F_3, F_4, F_5, F_6]$ and $[1,1,1,2,2,2]$ respectively.

In the case of a raster display, hidden line removal is done by the hardware (frame buffer of the display). We simply display the faces in the reverse order of their priority. Any faces that would have to be hidden by others would thus be displayed first, but would be covered later either partially or entirely by faces of higher priority.

In the case of a vector display, the hidden line removal must be done by software by determining coverings. For this purpose, the edges of a face are compared with all other edges of higher priority. An edge list can be created that maintains a list of all line segments that will have to be drawn as visible. Visibility techniques such as the containment test and edge intersection are useful in this case. Figure 5c shows the scene with hidden lines removed.

◇Assignment 4

Generate two overlapping wedges in positions similar to the blocks in Figure 5 using your CAD software. Display the objects in isometric view then run the priority algorithm to determine the priority list of the faces. ◇

4. The *z*-buffer algorithm

This is also known as the depth-buffer algorithm. In addition to the frame (refresh) buffer, this algorithm requires a *z* buffer in which *z* values can be sorted for each pixel. The *z* buffer is initialized to the smallest *z* value, while the frame buffer is initialized to the background pixel value. Both the frame and *z* buffers are indexed by pixel coordinates (x, y). These coordinates are actually screen coordinates. The *z*-buffer algorithm works as follows. For each polygon in the scene, find all the pixels (x, y) that lie inside or on the boundaries of the polygon when projected onto the screen. For each of these pixels, calculate the depth z of the polygon at (x, y). If z>depth (x, y), the polygon is closer to the viewing eye than others already stored in the pixel. In this case, the *z* buffer is updated by setting the depth (x, y) to z. Similarly, the intensity of the frame buffer location corresponding to the pixel is updated to the intensity of the polygon at (x, y). After all the polygons have been processed, the frame buffer contains the solution.

◇Assignment 5

Render the two objects generated in Assignment 4 with different colors then display them again. Draw a flowchart to describe how the image of each pixel is determined using the concept of the *z*-buffer algorithm. ◇

5. Sectioned Views

A sectioned view of a solid is essentially a view of the part of a solid that is beyond some plane $z=V$. This solid is projected as usual on the x-y plane to give a sectional view of the solid in respect to the section plane. Practically, this means that

(1) All lines that are above this plane are not plotted.
(2) Polygons that are entirely above this plane do not hide lines that are below this plane.
(3) Lines that intersect with the section plane are plotted up to this plane.
(4) For polygons that are intersecting with the section plane, their parts above the section plane has to be replaced with the line of intersection.

◇Assignment 6

Generate a block with a through hole using your CAD software. Display its sectioned view across the center of the hole. ◇