Unit 4
Visible Surface Detection and Surface-Rendering

Visible Surface Detection Methods (Hidden surface elimination)

Visible surface detection or Hidden surface removal is major concern for realistic graphics for identifying those parts of a scene that are visible from a chosen viewing position. Numerous algorithms have been devised for efficient identification of visible objects for different types of applications. Some methods require more memory, some involve more processing time, and some apply only to special types of objects. Deciding upon a method for a particular application can depend on such factors as the complexity of the scene, type of objects to be displayed, available equipment, and whether static or animated displays are to be generated.

Visible surface detection methods are broadly classified according to whether they deal with objects or with their projected images.

These two approaches are:

- **Object-Space methods:** An object-space method compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.

- **Image-Space methods**: Visibility is decided point by point at each pixel position on the projection plane.

Most visible surface detection algorithm use image-space-method but in some cases object space methods can also be effectively used.

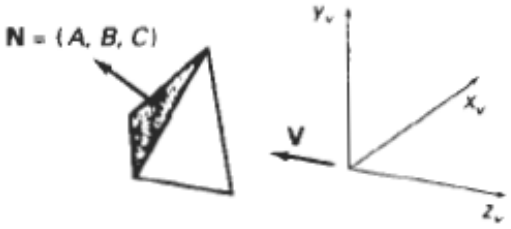
Back Face Detection (Plane Equation method)

A fast and simple object space method used to remove hidden surface from a 3D object drawing is known as "Plane equation method" and applied to each side after any rotation of the object takes place. It is commonly known as back-face detection of a polyhedron is based on the "inside-outside" tests.

A point (x, y, z) is inside a polygon surface if

$$Ax + By + Cz + D < O$$

We can simplify this test by considering the normal vector N to a polygon surface which has Cartesian components (A, B, C).



If V is the vector in viewing direction from the eye position then this polygon is a back face if,

V.N > 0

If object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing z_v axis, then V = (0, 0, V_z) and

$$V.N = V_zC$$

so that we only need to consider the sign of C, the z component of the normal vector N.

In a right-handed viewing system with viewing direction along the negative z_v axis and in general, we can label any polygon as a back face if it's normal vector has a z component value:

$C \leq 0$

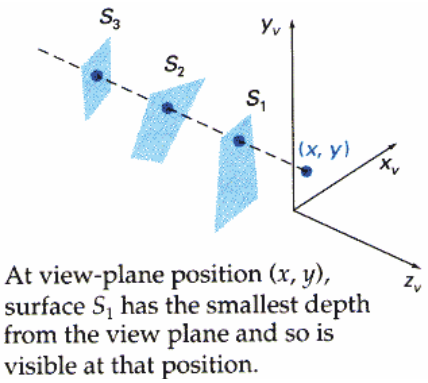
For other cases where there are concave polyhedral or overlapping objects, we still need to apply other methods to further determine where the obscured faces are partially or completely hidden by other objects (eg. Using Depth-Buffer Method or Depth-sort Method).

Depth-Buffer Method

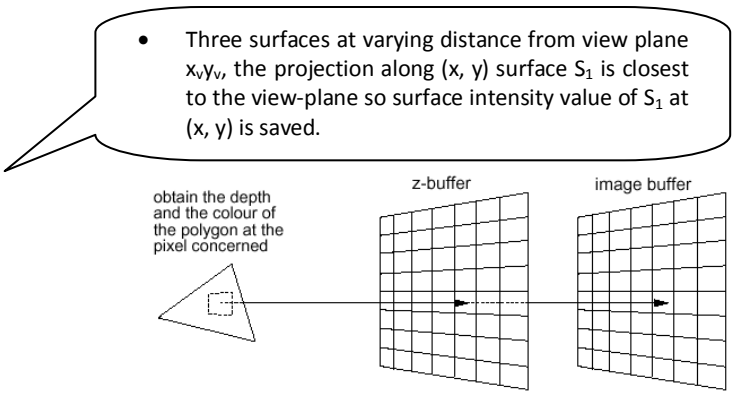
Depth Buffer Method is the commonly used image-space method for detecting visible surface. It is also know as **z-buffer method**. It compares surface depths at each pixel position on the projection plane. It is called z-buffer method since object depth is usually measured from the view plane along the z-axis of a viewing system.

Each surface of scene is processed separately, one point at a time across the surface. The method is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly and method is easy to implement. This method can be applied also to non planer surfaces.

With object description converted to projection co-ordinates, each (x, y, z) position on polygon surface corresponds to the orthographic projection point (x, y) on the view plane. Therefore for each pixel position (x, y) on the view plane, object depth is compared by z-values.



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



In Z-buffer method, two buffers area are required.

- **Depth buffer (z-buffer)**: stores the depth value for each (x, y) position as surfaces are processed.
- **Refresh buffer (Image buffer)**: stores the intensity value for each position.

Initially all the positions in depth buffer are set to 0, and refresh buffer is initialize to background color. Each surfaces listed in polygon table are processed one scan line at a time, calculating the depth (z-val) for each position (x, y). The calculated depth is compared to the value previously stored in depth buffer at that position. If calculated depth is greater than stored depth value in depth buffer, new depth value is stored and the surface intensity at that position is determined and placed in refresh buffer.

Algorithm: Z-buffer

1. Initialize depth buffer and refresh buffer so that for all buffer position (x, y)

   depth (x,y) =0, refresh (x, y) = $I_{background.}$
2. For each position on each polygon surface, compare depth values to previously stored value in depth buffer to determine visibility.
   - Calculate the depth ***z*** for each (x, y) position on polygon
   - If *z* > depth (x, y) then

     depth (x, y) = z, refresh (x, y) = $I_{surface}$(x, y)

Where $I_{background}$ = Intensity value for background

$I_{surface}$ (x, y) = Intensity value for surface at pixel position (x, y) on projected plane.

After all surfaces are processed, the depth buffer contains the depth value of the visible surface and refresh buffer contains the corresponding intensity values for those surfaces.

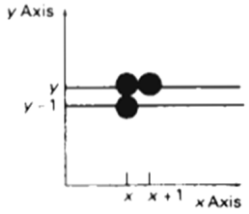
The depth values of the surface position (x, y) are calculated by plane equation of surface.

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

Let Depth *Z'* at position (x+1, y)

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

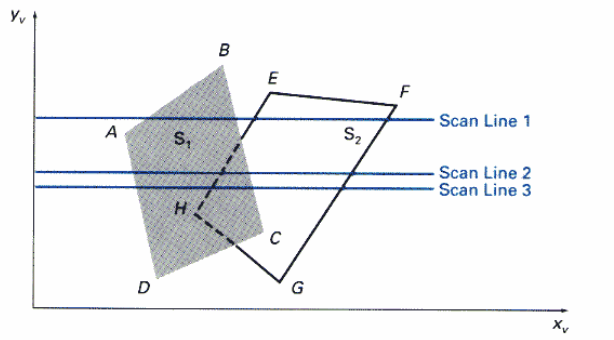
$$\Rightarrow Z' = Z - A/C \qquad (1)$$



$-A/C$ is constant for each surface so succeeding depth value across a scan line are obtained form preceding values by simple calculation.

Scan Line Method

- This is image-space method for removing hidden surface which is extension of the scan line polygon filling for polygon interiors. Instead of filling one surface we deal with multiple surfaces here.
- 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.

Scan Line Method

```
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
```

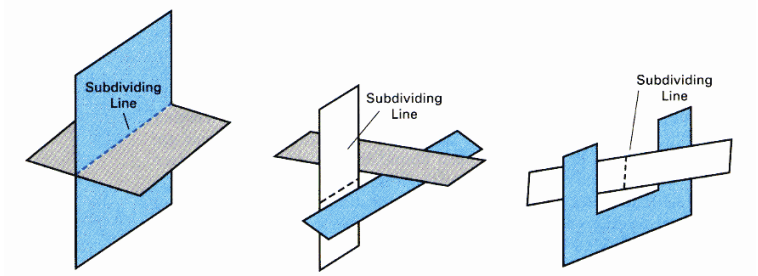
- Step 2 is not efficient because not all polygons necessarily intersect with the scan line.
- Depth calculation in 2a is not needed if only 1 polygon in the scene is mapped onto a segment of the scan line.

Figure above illustrates the scan-line method for locating visible portions of surfaces for pixel positions along the line. The active list for scan line 1 contains information from the edge table for edges AB, BC, EH, and FG. For positions along this scan line between edges AB and BC, only the flag for surface S_1 is on. Therefore, no depth calculations are necessary, and intensity information for surface S_1 is entered from the polygon table into the refresh buffer. Similarly, between edges EH and FG, only the flag for surface S_2 is on. No other positions along scan line 1 intersect surfaces, so the intensity values in the other areas are set to the background intensity. The background intensity can be loaded throughout the buffer in an initialization routine.

For scan lines 2 and 3, the active edge list contains edges *AD, EH, BC,* and *FG.* Along scan line 2 from edge *AD* to edge *EH,* only the flag for surface S_1 is on. **But between edges EH and *BC,* the flags for both surfaces are on**. In this interval, depth calculations must be made using the plane coefficients for the two surfaces. For this example, the depth of surface S_1 is assumed to be less than that of S_2, so intensities for surface S_1 are loaded into the refresh buffer until boundary BC is encountered. Then the flag for surface S_1 goes off, and intensities for surface S_2 are stored until edge FG is passed.

We can take advantage of coherence along the scan lines as we pass from one scan line to the next. In Fig., scan line 3 has the same active list of edges as scan line 2. Since no changes have occurred in line intersections, it is unnecessary again to make depth calculations between edges *EH* and *BC.* The two surfaces must be in the same orientation as determined on scan line 2, so the intensities for surface S_1 can be entered without further calculations.

Any number of overlapping polygon surfaces can be processed with this scan-line method. Flags for the surfaces are set to indicate whether a position is inside or outside, and depth calculations are performed when surfaces overlap. When these coherence methods are used, we need to be careful to keep track of which surface section is visible on each scan line. This works only if surfaces do not cut through or otherwise cyclically overlap each other.



If any kind of cyclic overlap is present in a scene, we can divide the surfaces to eliminate the overlaps. The dashed lines in this figure indicate where planes could be subdivided to form two distinct surfaces, so that the cyclic overlaps are eliminated.

Depth sorting Method

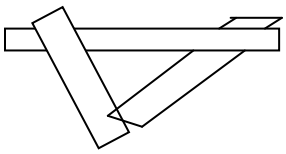
This method uses both object space and image space methods. In this method the surface representation of 3D object are sorted in of decreasing depth from viewer. Then sorted surface are scan converted in order starting with surface of greatest depth for the viewer.

The conceptual steps that performed in depth-sort algorithm are

1. Surfaces are sorted in order of decreasing depth (z-coordinate).
2. Resolve any ambiguity this may cause when the polygons z-extents overlap, splitting polygons if necessary.
3. Surfaces are scan converted in order, starting with the surface of greatest depth.

This algorithm is also called "**Painter's Algorithm**" as it simulates how a painter typically produces his painting by starting with the background and then progressively adding new (nearer) objects to the canvas.

Problem: One of the major problems in this algorithm is intersecting polygon surfaces. As shown in fig. below.



- Different polygons may have same depth.
- The nearest polygon could also be farthest.
- We cannot use simple depth-sorting to remove the hidden-surfaces in the images.

Solution: For intersecting polygons, we can split one polygon into two or more polygons which can then be painted from back to front. This needs more time to compute intersection between polygons. So it becomes complex algorithm for such surface existence.

Example

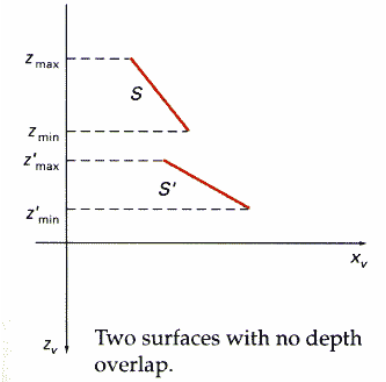
Assuming we are viewing along the z axis. Surface S with the greatest depth is then compared to other surfaces in the list to determine whether there are any overlaps in depth. If no depth overlaps occur, S can be scan converted. This process is repeated for the next surface in the list. However, if depth

overlap is detected, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.

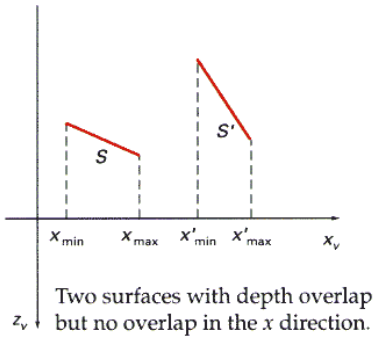
We make the following tests for each surface that overlaps with S. If any one of these tests is true, no reordering is necessary for that surface. The tests are listed in order of increasing difficulty.

1. The bounding rectangles in the xy plane for the two surfaces do not overlap
2. Surface S is completely behind the overlapping surface relative to the viewing position.
3. The overlapping surface is completely in front of S relative to the viewing position.
4. The projections of the two surfaces onto the view plane do not overlap.

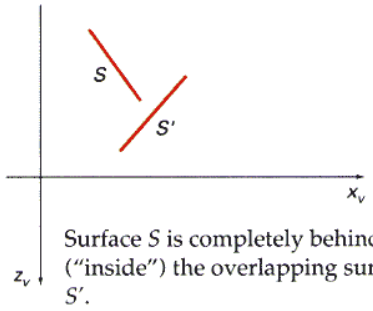
We perform these tests in the order listed and proceed to the next overlapping surface as soon as we find one of the tests is true. If all the overlapping surfaces pass at least one of these tests, none of them is behind S. No reordering is then necessary and S is scan converted.



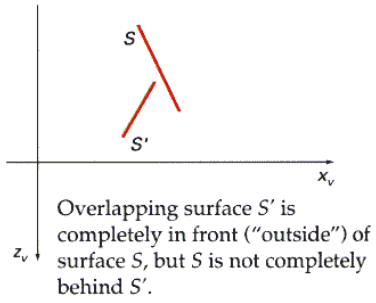
Two surfaces with no depth overlap.



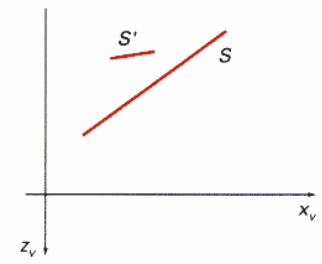
Two surfaces with depth overlap but no overlap in the x direction.



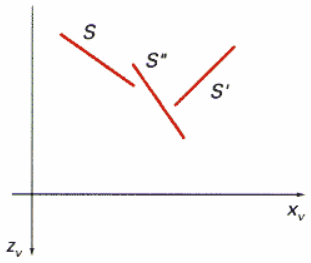
Surface S is completely behind ("inside") the overlapping surface S'.



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.

BSP Tree Method

A binary space partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces onto the screen from back to front as in the painter's algorithm. The BSP tree is particularly useful when the view reference point changes, but object in a scene are at fixed position.

Applying a BSP tree to visibility testing involves identifying surfaces that are "inside" or "outside" the partitioning plane at each step of space subdivision relative to viewing direction. It is useful and efficient for calculating visibility among a static group of 3D polygons as seen from an arbitrary viewpoint.

In the following figure,

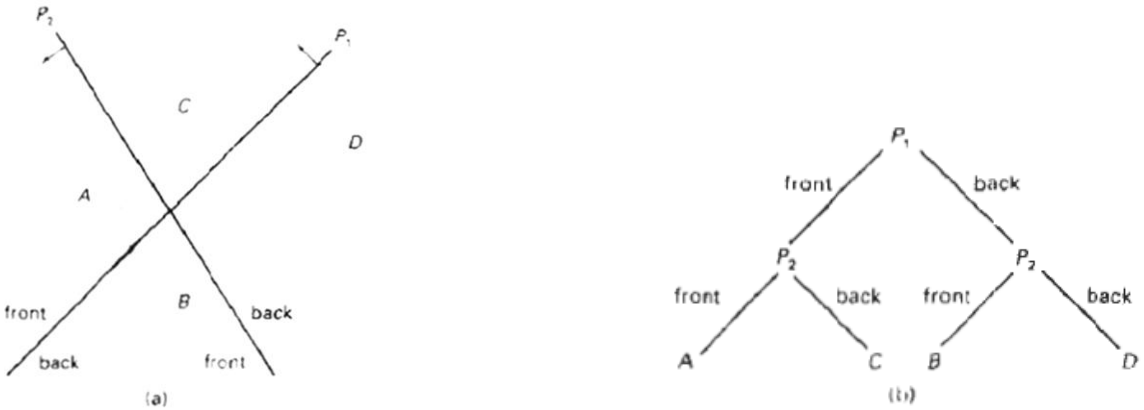


Fig: A region of space (a) partitioned with two planes P_1 and P_2 to form the BSP tree representation in (b).

- Here plane P_1 partitions the space into two sets of objects, one set of object is back and another set is in front of partitioning plane relative to viewing direction. Since one object is intersected by plane P_1, we divide that object into two separate objects labeled A and B. Now object A & C are in front of P_1, B and D are back of P_1.
- We next partition the space with plane P_2 and construct the binary free as fig (b). In this tree, the objects are represented as terminal nodes, with front object as left branches and behind object as right branches.
- When BSP tree is complete, we process the tree by selecting surface for displaying in order back to front. So foreground object are painted over back ground objects.

Octree Method

When an octree representation is used for viewing volume, hidden surface elimination is accomplished by projecting octree nodes into viewing surface in a front to back order. Following figure is the front face of a region space is formed with octants 0, 1, 2, 3. Surface in the front of these octants are visible to the viewer. The back octants 4, 5, 6, 7 are not visible. After octant sub-division and construction of octree, entire region is traversed by depth first traversal.

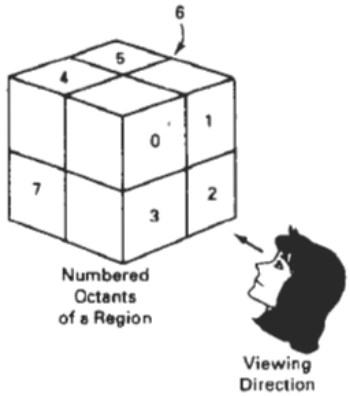
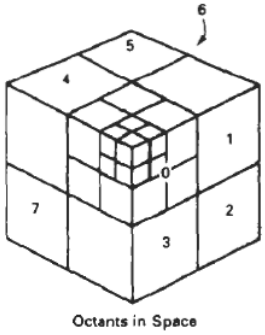


Fig1: Objects in octants 0, 1, 2**,** and 3 obscure objects in the back octants (4, 5, 6, 7) when the viewing direction is as shown.



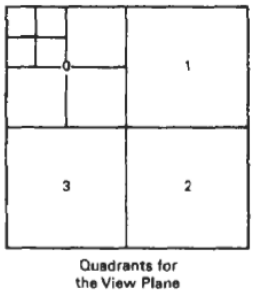


Fig2: Octant divisions for a region of space and the corresponding quadrant plane.

Different views of objects represented as octrees can be obtained by applying transformations to the octree representation that reorient the object according to the view selected.
Fig2 depicts the octants in a region of space and the corresponding quadrants on the view plane. Contributions to quadrant 0 come from octants 0 and 4. Color values in quadrant 1 are obtained from surfaces in octants 1 and 5, and values in each of the other two quadrants are generated from the pair of octants aligned with each of these quadrants.

Ray Casting (Ray Tracing)
Ray tracing also known as ray casting is efficient method for visibility detection in the objects. It can be used effectively with the object with curved surface. But it is also used for polygon surfaces.

- Trace the path of an imaginary ray from the viewing position (eye) through viewing plane t object in the scene.
- Identify the visible surface by determining which surface is intersected first by the ray.
- Can be easily combined with lightning algorithms to generate shadow and reflection.
- It is good for curved surface but too slow for real time application.

Ray casting, as a visibility detection tool, is based on geometric optics methods, which trace the paths of light rays. Since there are an infinite number of light rays in a scene and we are interested only in those rays that pass through pixel positions, we can trace the light-ray paths backward from the pixels through the scene. The ray-casting approach is an effective visibility-detection method for scenes with curved surfaces, particularly spheres.

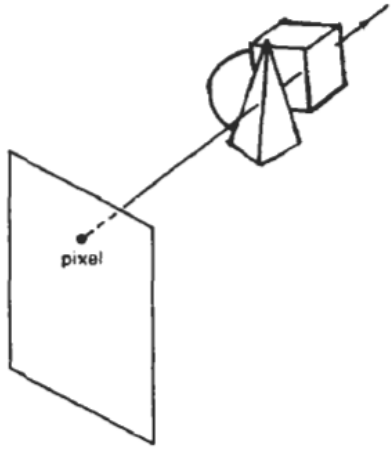


Fig: A ray along the line of sight from a pixel position through a scene.

- In ray casting, we process pixels one at a time and calculate depths for all surfaces along the projection path to that pixel.
- In fact, Ray casting is a special case of *ray-tracing algorithms* that trace multiple ray paths to pick up global reflection and refraction contributions from multiple objects in a scene. With ray casting, we only follow a ray out from each pixel to the nearest object.

Illumination and Surface Rendering

- Realistic displays of a scene are obtained by perspective projections and applying natural lighting effects to the visible surfaces of object.
- An illumination model is also called lighting model and sometimes called as a shading model which is used to calculate the intensity of light that we should see at a given point on the surface of a object.
- A surface-rendering algorithm uses the intensity calculations from an illumination model.

Light Sources

Sometimes light sources are referred as light emitting object and light reflectors. Generally light source is used to mean an object that is emitting radiant energy e.g. Sun.

Point Source: Point source is the simplest light emitter e.g. light bulb.
Distributed light source: Fluorescent light

Fig: Diverging ray paths from the Point light source

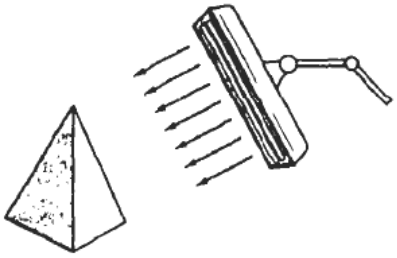
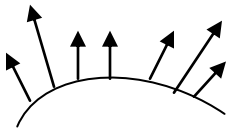


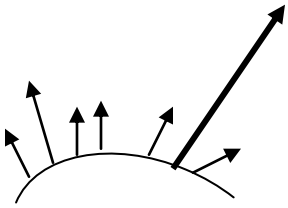
Fig: An object illuminated with a distributed light source

- When light is incident on an opaque surface part of it is reflected and part of it is absorbed.
- Surface that are rough or grainy, tend to scatter the reflected light in all direction which is called diffuse reflection.



Diffuse reflection

- When light sources create highlights, or bright spots, called specular reflection



Specuular reflection

Illumination models

Illumination models are used to calculate light intensities that we should see at a given point on the surface of an object. Lighting calculations are based on the optical properties of surfaces, the background lighting conditions and the light source specifications. All light sources are considered to be

point sources, specified with a co-ordinate position and an intensity value (color). Some illumination models are:

1. Ambient light

- This is a simplest illumination model. We can think of this model, which has no external light source-self-luminous objects. A surface that is not exposed directly to light source still will be visible if nearby objects are illuminated.
- The combinations of light reflections form various surfaces to produce a uniform illumination called ambient light or background light.
- Ambient light has no spatial or directional characteristics and amount on each object is a constant for all surfaces and all directions. In this model, illumination can be expressed by an illumination equation in variables associated with the point on the object being shaded. The equation expressing this simple model is

$$I = K_a$$ K_a ranges from 0 to 1.

Where I is the resulting intensity and K_a is the object's intrinsic intensity.

If we assume that ambient light impinges equally on all surfaces from all direction, then

$$I = I_a K_a$$

Where I_a is intensity of ambient light. The amount of light reflected from an object's surface is determined by K_a, the ambient-reflection coefficient.

2. Diffuse reflection

Objects illuminated by ambient light are uniformly illuminated across their surfaces even though light are more or less bright in direct proportion of ambient intensity. Illuminating object by a point light source, whose rays enumerate uniformly in all directions from a single point. The object's brightness varies from one part to another, depending on the direction of and distance to the light source.

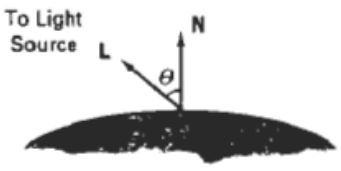
- The fractional amount of the incident light that is diffusely reflected can be set for each surface with parameter K_d, the coefficient of diffuse-reflection.
- Value of K_d is in interval 0 to 1. If surface is highly reflected, K_d is set to near 1. The surface that absorbs almost incident light, K_d is set to nearly 0.
- Diffuse reflection intensity at any point on the surface if exposed only to ambient light is

$$I_{ambdiff} = I_a K_d$$

- Assuming diffuse reflections from the surface are scattered with equal intensity in all directions, independent of the viewing direction (surface called. "Ideal diffuse reflectors") also called Lambertian reflectors and governed by Lambert's cosine law.

$$I_{diff} = K_d I_l \cos\theta$$

Where I_l is the intensity of the point light source.



It N is unit vector normal to the surface & L is unit vector in the direction to the point slight source then

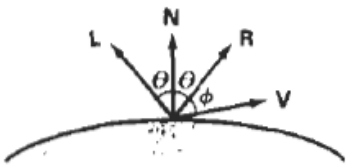
$$I_{l,diff} = K_d I_l (N..L)$$

In addition, many graphics packages introduce an ambient reflection coefficient K_a to modify the ambient-light intensity I_a

$$I_{diff} = K_a I_a + K_d I_l (N..L)$$

3. Specular reflection and pong model

When we look at an illuminated shiny surface, such as polished metal, a person's forehead, we see a highlight or bright spot, at certain viewing direction. Such phenomenon is called specular reflection. It is the result of total or near total reflection of the incident light in a concentrated region around the specular reflection angle.



N - Unit vector normal to surface at incidence point
R - Unit vector in the direction of ideal specular reflection.
L - Unit vector directed to words point light source.
V - Unit vector pointing to the viewer from surface.
$\phi -$ Viewing angle relative to the specular reflection direction.

Fig: Specular-reflection equals the angle of incidence θ

- For ideal reflector (perfect mirror), incident light is reflected only in the specular reflection direction i.e. V and R coincides $(\phi = 0)$.
- Shiny surfaces have a narrow specular-reflection range (narrow ϕ), and dull surfaces have a wider reflection (wider ϕ).
- An empirical model for calculating specular-reflection range developed by Phong Bui Tuong called **Phong specular reflection model** (or simply **Phong model**), sets the intensity of specular reflection proportional to $\cos^{n_s} \phi$ [$\cos \phi$ varies from 0 to 1] where $\mathbf{n_s}$ is a specular reflection parameter.
- Specular reflection parameter n_s is determined by type of surface that we want to display:
    - Very shiny surface: large n_s (say 100 or more) and
    - dull surface , smaller n_s (down to 1)
    - For perfect reflector n_s is infinite.

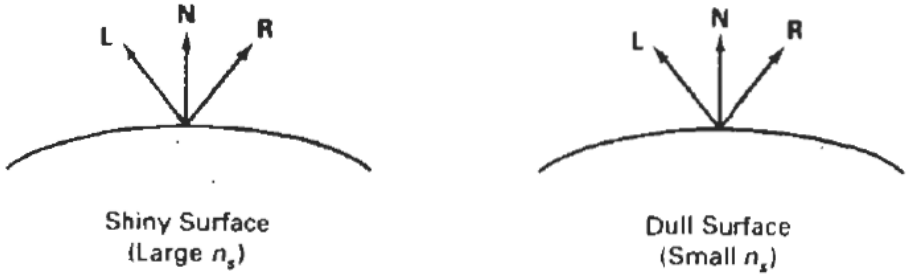


Fig: Modeling specular reflections (shaded area) with parameter $\mathbf{n_s}$

The intensity of specular reflection depends on the material properties of the surface and the angle of incidence (θ), as well as other factors such as the polarization and color of the incident light.

- We can approximately model monochromatic specular intensity variations using a specular-reflection coefficient, W(θ) for each surface over a range θ = *0*° to θ = 90°. In general, W(θ) tends

to increase as the angle of incidence increases. At θ = *90°,* W(θ) = 1 and all of the incident light is reflected.

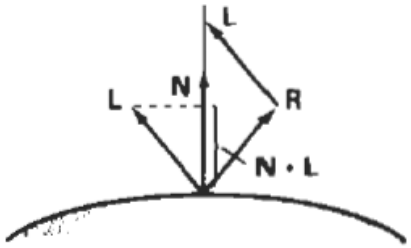
- The variation of specular intensity with angle of incidence is described by ***Fresnel's*** *laws* of *Reflection.* Using the spectral-reflection function *W(θ),* we can write the Phong specular-reflection model as:

$$I_{spec} = w(\theta) I_l \cos^{n_s} \phi$$

Where I_l is intensity of light source. ϕ is viewing angle relative to SR direction R.

- Transparent materials, such as glass, only exhibit appreciable specular reflections as θ approaches 90°**.** At θ **=** 0°**,** about 4 percent of the incident light on a glass surface is reflected.
- For many opaque materials, specular reflection is nearly constant for all incidence angles. In this case, we can reasonably model the reflected light effects by replacing W(θ) with a constant specular-reflection coefficient K_s**.**

So , $I_{spec} = K_s I_l \cos^{n_s} \phi = K_s I_l (V.R)^{n_s}$ Since $\cos\phi$ = V.R



- Vector R in this expression can be calculated in terms of vectors L and N. As seen in Fig. above, the projection of L onto the direction of the normal vector is obtained with the dot product **N.L**. Therefore, from the diagram, we have

R + L = (2N.L)N

and the specular-reflection vector is obtained as

R = (2N.L)N - L

Polygon (surface) Rendering Method

- Application of an illumination model to the rendering of standard graphics objects those formed with polygon surfaces are key technique for polygon rendering algorithm.
- Calculating the surface normal at each visible point and applying the desired illumination model at that point is expensive. We can describe more efficient shading models for surfaces defined by polygons and polygon meshes.
- Scan line algorithms typically apply a lighting model to obtain polygon surface rendering in one of two ways. Each polygon can be rendered with a single intensity, or the intensity can be obtained at each point of the surface using an interpolating scheme.

1. **Constant Intensity Shading** (Flat Shading)

The simplest model for shading for a polygon is constant intensity shading also called as Faceted Shading or flat shading. This approach implies an illumination model once to determine a single intensity value that is then used to render an entire polygon. Constant shading is useful for quickly displaying the general appearance of a curved surface.

This approach is valid if several assumptions are true:

a) The light source is sufficiently far so that **N.L** is constant across the polygon face.

b) The viewing position is sufficiently far from the surface so that **V.R** is constant over the surface
c) The object is a polyhedron and is not an approximation of an object with a curved surface.

> Even if all of these conditions are not true, we can still reasonably approximate surface-lighting effects using small polygon facets with flat shading and calculate the intensity for each facet, say, at the center of the polygon.

2. **Interpolated Shading**:

An alternative to evaluating the illumination equation at each point on the polygon, we can use the interpolated shading, in which shading information is linearly interpolated across a triangle from the values determined for its vertices. Gouraud generalized this technique for arbitrary polygons. This is particularly easy for a scan line algorithm that already interpolates the z-value across a span from interpolated z-values computed for the span's endpoints.

Gouraud Shading

Gouraud shading, also called intensity interpolating shading or color interpolating shading eliminates intensity discontinuities that occur in flat shading. Each polygon surface is rendered with Gouraud shading by performing following calculations.

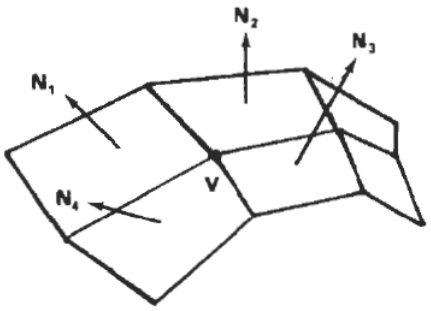
1. Determine the average unit normal vector at each polygon vertex.
2. Apply an illumination model to each vertex to calculate the vertex intensity.
3. Linearly interpolate the vertex intensities over the surface of the polygon

Step 1: At each polygon vertex, we obtain a normal vertex by averaging the surface normals of all polygons sharing the vertex as:

$$N_v = \frac{\sum_{k=1}^{n} N_k}{|\sum_{k=1}^{n} N_k|}$$

Here in example: $N_v = \frac{N_1 + N_2 + N_3 + N_4}{|N_1 + N_2 + N_3 + N_4|}$

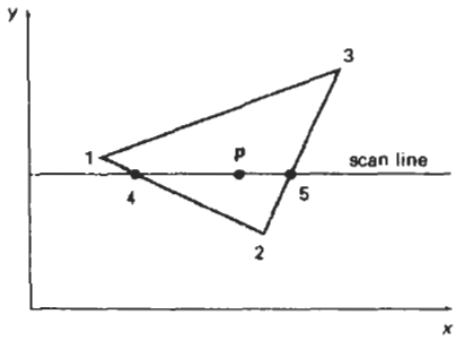
Where N_v is normal vector at a vertex sharing 4 surfaces as in figure.



Step 2: Once we have the vertex normals (N_v), we can determine the intensity at the vertices from a lighting model.

Step 3: Now to interpolate intensities along the polygon edges, we consider following figure (next page…☺):

In figure, the intensity of vertices 1, 2, 3 are I_1, I_2, I_3, which are obtained by averaging normals of each surface sharing the vertices and applying a illumination model. For each scan line, intensity at intersection of line with Polygon edge is linearly interpolated from the intensities at the edge end point.



So intensity at point 4 is to interpolate between intensities I_1 and I_2 using only the vertical displacement of the scan line:

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$

Similarly, the intensity at point 5 is obtained by linearly interpolating intensities at I_2 and I_3 as

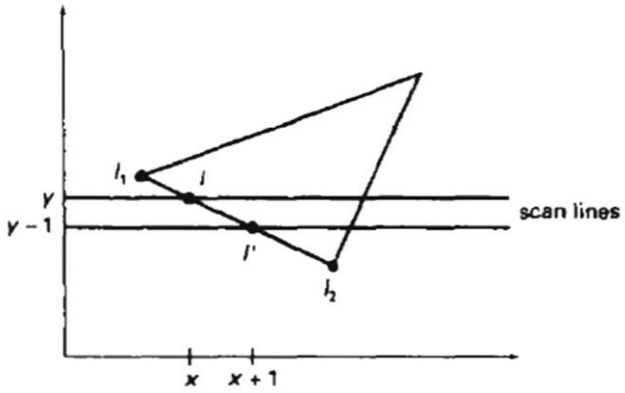
$$I_5 = \frac{y_5 - y_2}{y_3 - y_2} I_3 + \frac{y_3 - y_5}{y_3 - y_2} I_2$$

The intensity of a point P in the polygon surface along scan-line is obtained by linearly interpolating intensities at I_4 and I_5 as,

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

Then incremental calculations are used to obtain Successive edge intensity values between scan-lines as and to obtain successive intensities along a scan line. As shown in Fig. below, if the intensity at edge position (x, y) is interpolated as:

$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2$$



Then, we can obtain the intensity along this edge for next scan line at y-1 position as

$$I' = \frac{y - 1 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - (y-1)}{y_1 - y_2} I_2 \quad = I + \frac{I_2 - I_1}{y_1 - y_2}$$

Similar calculations are made to obtain intensity successive horizontal pixel.

Advantages: Removes intensity discontinuities at the edge as compared to constant shading.
Disadvantages: Highlights on the surface are sometimes displayed with anomalous shape and linear intensity interpolation can cause bright or dark intensity streak called mach-bands.

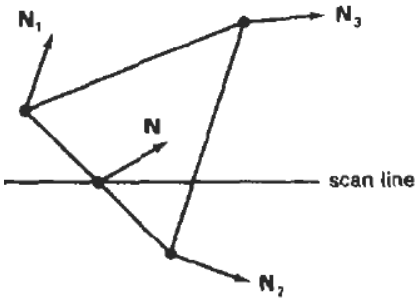
Phong Shading

A more accurate method for rendering a polygon surface is to interpolate normal vector and then apply illumination model to each surface point. This method is called **Phong shading** or **normal vector interpolation shading**. It displays more realistic highlights and greatly reduces the mach-band effect.

A polygon surface is rendered with Phong shading by carrying out following calculations.

- Determine the average normal unit vectors at each polygon vertex.
- Linearly interpolate vertex normals over the surface of polygon.
- Apply illumination model along each scan line to calculate projected pixel intensities for the surface points.

In figure, N_1, N_2, N_3 are the normal unit vectors at each vertex of polygon surface. For scan-line that intersect an edge, the normal vector N can be obtained by vertically interpolating normal vectors of the vertex on that edge as.



$$N = \frac{y - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$

Incremental calculations are used to evaluate normals between scan lines and along each individual scan line as in Gouraud shading. Phong shading produces accurate results than the direct interpolation but it requires considerably more calculations.

Fast Phong Shading

Surface rendering with Phong shading can be speeded up by using approximations in the illumination-model calculations of normal vectors. Fast Phong shading approximates the intensity calculations using a Taylor series expansion and Triangular surface patches. Since Phong shading interpolates normal vectors from vertex normals, we can express the surface normal N at any point (x, y) over a triangle as:

$$N = Ax + By + C$$

Where A, B, C are determined from the three vertex equations.

$$N_k = Ax_k + By_k + C, \quad k = 1, 2, 3 \text{ for } (x_k, y_k) \text{ vertex.}$$

Omitting the reflectivity and attenuation parameters, we can write the calculation for light-source diffuse reflection from a surface point (x, y) as

$$I_{diff}(x, y) = \frac{L.N}{|L|.|N|} = \frac{L.(Ax + By + C)}{|L|.|Ax + By + C|} = \frac{(L.A)x + (L.B)y + (L.C)}{|L|.|Ax + By + C|}$$

Re writing this,

$$I_{diff}(x, y) = \frac{ax + by + c}{(dx^2 + exy + fy^2 + gx + hy + i)^{\frac{1}{2}}} \quad \text{---------- (1)}$$

Where parameters a, b, c, d… are used to represent the various dot products as $a = \frac{L.N}{|L|}$... and so on

Finally, denominator of equation (1) can be expressed as Taylor series expansions and retains terms up to second degree in x and y. This yields

$$I_{diff}(x, y) = T_5 x^2 + T_4 xy + T_3 y^2 + T_2 x + T_1 y + T_0$$

Where each T_k is a function of parameters a, b, c, d, and so forth.

This method still takes twice as long as in Gouraud shading. Normal Phong shading takes six to seven times that of Gouraud shading.