Chapter 6

Visible Surface Detection & Animation

Visible surface detection :

- classification of visible surface detection algo
- Back surface detection method
- Depth Buffer mtd,
- Area sub division Mtd.

Animation : Introduction to Ani

- Traditional Ani Tech.
- Principles of Ani
- Key framing
- Character & facial Ani
- Deformation
- Motion Capture.

1. Explain Z Buffer algo for hidden surface removal.
2. Explain the concept of halftoning with example.
3. Short note on i) Sweep representation & octree
   Gouraud & Phong shading
   Dithering.
4. What is shading? Explain Gouraud & Phong shading with their pros & cons.
5. Write short on i) Halftone & Dithering ii) Area Subdivision

Classification of Visible Surface Detection Algorithms

This algos are classified a/c to whether they deal with object definitions directly or with their projected images.

These 2 approaches are called object space methods and image space methods resp.

An object-space method compares objects & parts of object to each other within the scene to determine which surfaces, as a whole are visible.

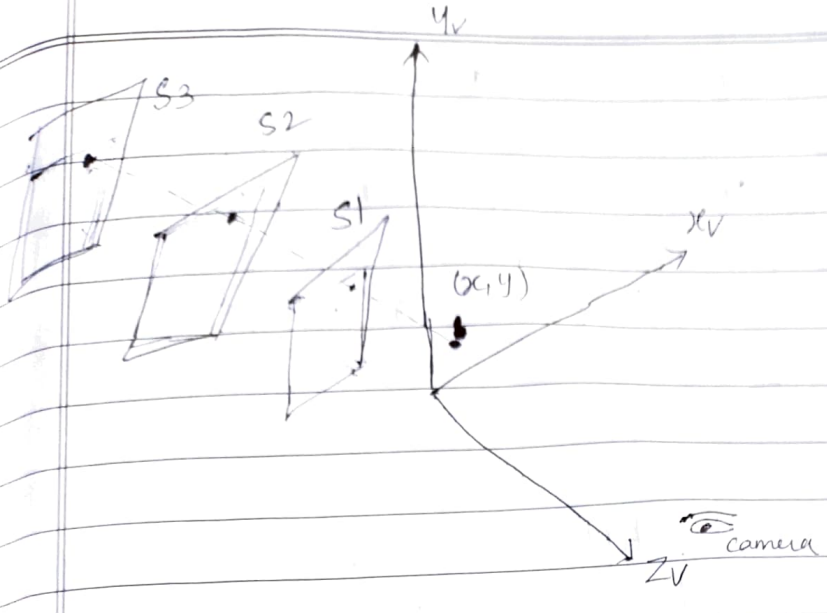
In an image-space algorithm, visibility is decided point by point at each pixel position on projection plane.

Depth-Buffer Method

This uses image space approach to detect visible surfaces, which compares surface depths at each pixel position on the projection plane. This is also referred as z-buffer method. Since object depth is usually measured from view plane along z-axis of the viewing system.

Each surface of a scene is processed separately, one point at a time across the surface.

When object descriptions are converted to projection coordinates, each (x, y) position on a polygon surface corresponds to orthographic project point (x, y) on view plane. Therefore for each pixel (x, y) on view plane, object depths can be compared by comparing z-values.



In the above diagram 3 surfaces are taken at different positions along orthographic projection line from (x, y) position in view plane.

So here from where you are observing the object is important. If u are at Z axis then S1 is closest at the position

But if eye position is along -Z axis then S_3 is the closest & accordingly save the intensity value (x, y).

In this method, 2 buffers are required. A depth buffer is used to store depth values for each (x, y).

Depth is nothing but distance bet^n object & the eye position (or camera).

Refresh buffer stores the intensity values for each position

Initially, all depth of all positions are set to 0, & refresh buffer is initialized to background intensity

Each surface in the list of polygon tables is then processed, one scan line at a time, calculating the depth (z) at each (x, y).

The calculated depth is compared to the

stored in depth buffer at that position.

If the calculated depth is greater than the value stored in depth buffer, the new depth value is stored & intensity of that pixel position i.e (x,y) is placed at xy location in refresh buffer.

Algo

1) Initialize the depth buffer & refresh buffer so that for all buffer positions (x,y)

depth(x,y) = 0 refresh(x,y) = $I_{bkground}$

2) For each position on each polygon surface compare depth value with the stored values in the depth buffer to determine visibility

- Calculate depth z for each (x,y)

if z > depth(x,y) then

depth(x,y) = z

refresh(x,y) = $I_{suface}(x,y)$

where $I_{bkground}$ – background intensity

$I_{suface}(x,y)$ is projected intensity value for pixel position (x,y).

After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces & refresh buffer contains corresponding intensity values for those surfaces.

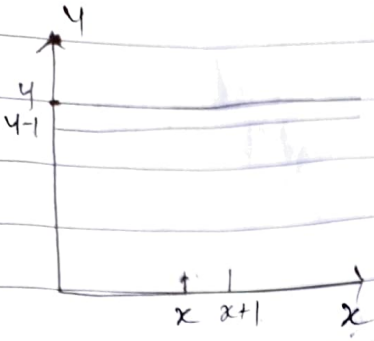
A polygon surface with plane parameters A, B, C & D is

$$Ax + By + Cz + d < 0 \quad (1)$$

Depth values for surface position (x,y) are calculated from plane eqn for each surface

$z =$

$$Z = \frac{-Ax - By - D}{C} \quad \text{---} \cdot (2)$$



For each scan line adjacent horizontal & vertical positions across the line differ by 1

If the depth is Z then z' is the depth of next position ~~(x, y)~~ (x+1, y) along scan line is

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

$$\therefore Z' = Z - \frac{A}{C}$$

Here $-A/C$ is constant for each surface. So next depth values are obtained from ~~previous~~ preceding values with single addition

Advantage

It is very easy to implement & it doesnot requires sorting of the surfaces.

Disadv

It requires 2 buffers.

If system has resolution (1024 x 1024) then it require a million positions in depth buffer. So large amount of memory is required. This algo works only for opaque objects & cannot accumulate intensity values for more than one surface.

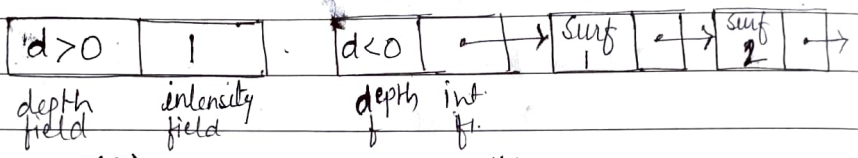
A- Buffer Method.

To overcome the drawbacks of Z- buffer. A- Buffer mtd is used.

In this mtd, it expands the depth buffer so that each position in the buffer can reference a linked list of surfaces. So more than one surface intensity can be taken into consideration at each pixel position.

A - buffer mtd represents an antialiased, area averaged, accumulation buffer mtd developed by Lucasfilm in surface rendering systems called REYES (Render Everything YOU EVER Saw)

Each position in A - buffer has 2 fields
depth field - stores positive or -ve real nos.
intensity field - stores surface intensity information



(a) (b)

Single surface Overlap of corresp. pixel area — Multiple surface overlap.

If the depth field is +ve, the no. stored at that position is the depth of single surface overlapping the corresponding pixel area. The intensity field stores RGB components of the surface color at that pt. & percent of pixel coverage.

If the depth field -ve, this means multiple surface contributions to pixel intensity. The intensity field stores a pointer to a linked list of surface data. Data for each surface in linked list includes.

a) RGB intensity components b) opacity parameter (% of transparency). c) depth d) percent of area coverage e) surface identifier f) other surface rendering parameters g) pt to next surface.

Scan lines are processed to determine surface overlaps of pixels across individual scanlines.

Surfaces are subdivided into polygon mesh & clipped against pixel boundaries.

Using opacity & % of surface overlap, we can calculate intensity of each pixel as an avg. of the contributions from overlapping surfaces.

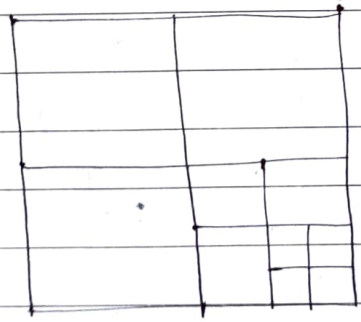
Area-Subdivision Method

This is a hidden surface removal technique which uses image space method but object space operations can be used to find depth ordering of surfaces.

The area-subdivision method takes advantage of area coherence in a scene by locating those view areas that represent part of a single surface.

We apply this method by successively dividing the total viewing area into smaller & smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all.

To implement this method, first step is to quickly identify the area as a part of a single surface or find out whether the area is too complex to analyze. If the tests indicate that the view is sufficiently complex then subdivide it. Next, apply the test to each of the smaller areas, subdivide these if the tests indicate that the visibility of single surface is uncertain. Continue this until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of single pixel. Easiest way to do this is to successively divide the area into 4 equal parts at each step.



Tests to determine the visibility of a single surface within a specified area are made by comparing surfaces to the boundary of a area. There are 4 possible rel^{n}ships that a surface can have with a specified area boundary.

1) Surrounding surface - One that completely encloses the area
2) Overlapping surface - One that is partly inside & partly outside the area.

3) Inside surface - one that is completely inside the area
4) Outside surface - One that is " " outside "

No further subdivisions of a specified area are needed if one of the following cond^n is true

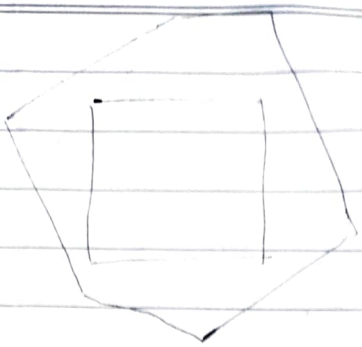
1) All surfaces are outside surfaces with respect to the area
2) Only one inside, overlapping or surrounding surface is in the area.
3) A surrounding surface obscures all other surfaces within the area boundaries.

Test1 can be carried out by checking the bounding rectangles of all surfaces against the area boundaries.

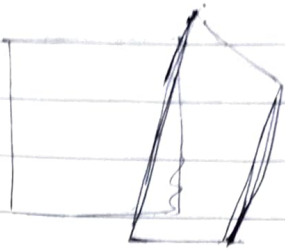
Test2 can also use bounding rectangles in xy plane to identify an inside ~~check~~ surface.

For other types of surfaces, bounding rectangles can be used as an initial check. If a single bounding rectangle intersects the area then additional tests are required to determine whether the surface is surrounding, overlapping or outside. Once a single inside, overlapping or surrounding surface has been identified, its pixel intensities are transferred to the appropriate area within frame buffer.

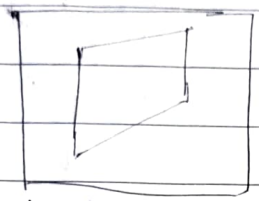
One method for implementing test 3 is to order surfaces acc to their minimum depth from view plan. For each surround surface, we then compute maximum depth within the area under consideration. If the maximum depth of one of these surrounding surfaces is closer to view plane then the minimum depth of all other surfaces within the area, test 3 is satified.



Surrounding surface



Overlapping surface



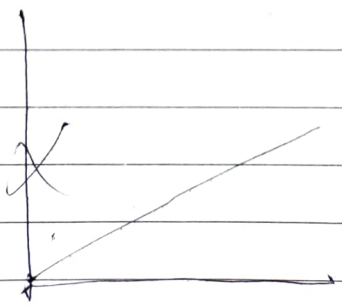
inside surface

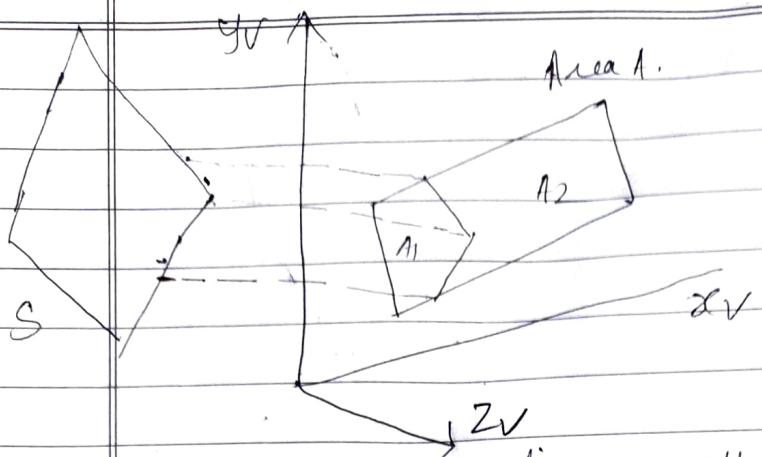
outside surface

Rela^n ship bet^n polygon surfaces & a rectangular area.

Another ~~test~~ method for test 3 ~~is to sort~~ is to use plane equations to calculate depth values at 4 vertices of the area for all surrounding, overlapping & inside surfaces. If the calculated depth for one of the surrounding surfaces is less than the calculated depth for all other surfaces, test 3 is true. Then the area can be filled with intensity values of the surrounding surface.

As a variation on basic subdivision process, we could subdivide areas along surface boundaries instead of dividing them in half. If the surfaces are sorted according to minimum depth, we can use surface with smallest depth value to subdivide a given area.





In the above diagram, the projections of boundary surface S is used to partition ~~the~~ the original area into the subdivisions A1 & A2. Surface S is then surrounding surface for area A1 & visibility tests 2 & 3 can be applied to determine whether further subdivision is necessary. So in this method, fewer subdivisions are required but more processing is needed to subdivide areas and to analyze the relation of surfaces to subdivision boundaries.

Back-Face-Detection

A fast & simple object space method for identifying the back faces of polyhedron is based on inside outside tests.

A point (x, y, z) is inside polygon surface with plane parameters A, B, C & D if

$$Ax + By + Cz + D < 0 \quad - (1)$$

when an inside pt is along the line of sight to the surface, the polygon must be a back-face.

To test this, consider a normal vector N to a polygon surface, which has cartesian components (A, B, C).

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

$$V \cdot N > 0 \quad - (2)$$

If the object descriptions are converted to projection coordinates & our viewing direction is ||el to Z_v then $V = (0, 0, \pm V_z)$

& $V \cdot N = V_z C$

Here confirm the sign of C.

In a right handed viewing system with viewing direction along $-ve\ Z_v$ axis., the polygon is a back face if $C < 0$

Thus in general, we can label any polygon as a back face if its normal vector has z-component value:

$$C \leq 0 \quad - (3)$$

In left handed viewing system,

In Hidden surface, we assume specific viewing direction.
bcoz surface hidden from a particular position may not be so if we change the viewing posi.

Right handed coord. system with viewer looking at the scene along -Z in scene.
Objects which represent with polygonal surfaces.

HSR has 2 types
1) Object space
↓
Compare objects or parts of objects with each other.
For
Advantage is
device independent & work for any resolution
Disadv.
Computation intensive
Sometimes if scene is complex then - is infeasible
Suitable for simple scenes & having small no. of obj.

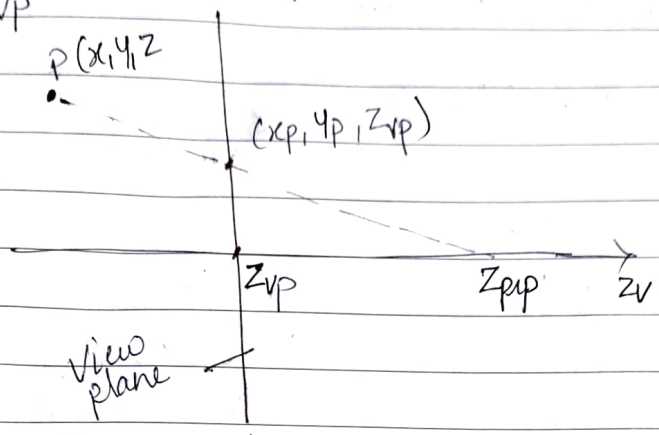
2) Image space
↓
Visibility decided pt-by-pt at each pixel
In this, determine objects closest to viewer & then draw the pixel with object color.
Computations are less
change in resolution requires re-computation of pixels.

Coherence - one way to reduce computation by exploit local similarities - making use of calculated result for nearby objects.

Perspective Projections.

To obtain this of a 3D object, we transform point along projection lines that meet at the projection reference point.

Suppose the projection reference point at position z_{prp} along z_v axis & view plane is z_{vp}



perspective projection of a pt $P(x,y,z)$ to position (x_p, y_p, z_{vp}) on view plane.

Eqns describing coordinate positions along this prospective projection line in parametric form as

$$\left.\begin{aligned} x' &= x - xu \\ y' &= y - yu \\ z' &= z - (z - z_{prp})u \end{aligned}\right\} \quad (1)$$

u takes values from 0 to 1

coordinate position (x', y', z') represents any pt along projection line.

when $u = 0$, we are at position $P = (x,y,z)$

At the end of line, $u = 1$

projection reference pt coordinates are $(0, 0, z_{prp})$

On view plane

$$z' = z_{vp}$$

$$u = \frac{z_{vp} - z}{z_{prp} - z} \quad - (2)$$

Sustitute z into eqns for x' & y'
we obtain perspective transformation eqns

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = x\left(\frac{d_p}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = y\left(\frac{d_p}{z_{prp} - z}\right) \quad -(3)$$

where $d_p = z_{prp} - z_{vp}$ is the distance of view plane from projection reference point.

3D homogeneous coordinate representation is

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{vp}/d_p & z_{vp}(z_{prp}/d_p) \\ 0 & 0 & -1/d_p & z_{prp}/d_p \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad -(4)$$

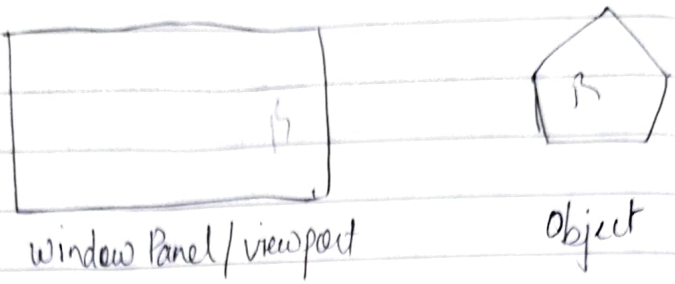
homogenous factor is

$$h = \frac{z_{prp} - z}{d_p} \quad -(5)$$

projection coordinates on view plane are

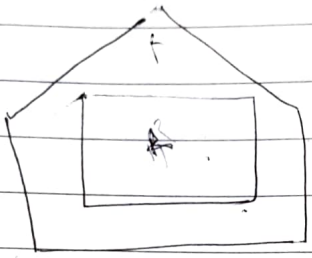
$$x_p = x_h/h \quad \& \quad y_p = y_h/h \quad -(6)$$

When we subdivide the panel aganist the polygon we may come through the following cases which are as follows :

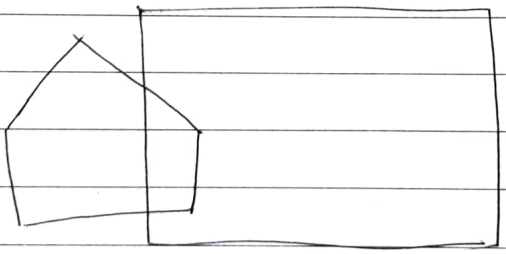


Window Panel / viewport Object

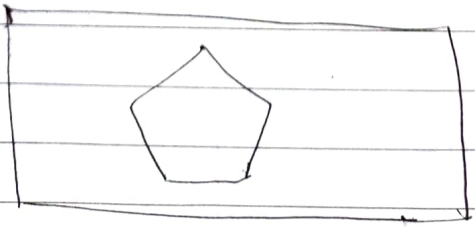
1) Surrounding surface
It's the case in which the viewing polygon surface completely surrounds the whole window panel



2) Overlapping surface -
It's the case in which the window panel & viewing polygon surface both intersect each other

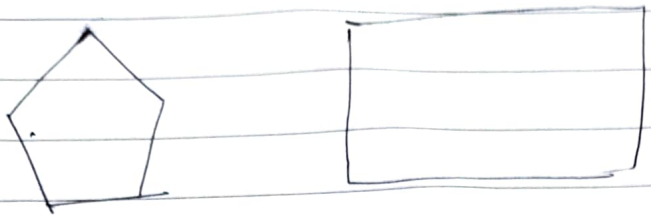


3) Inside surface -
In this case, in which polygon surface is inscribed inside the window panel.



4) Outside surface -

Here the entire surface is completely outside the window panel



Algorithm

1. Initialize the viewing area or window panel
2. Enlist all polygon & sort them a/c to Zmin (depth value) w.r. to window panel
3. Categorize all polygons a/c to their corresponding cases in which they are falling.
4) Now perform visible surface detection test :

i) if a polygon is surrounded the window panel then set the viewing area color to corresponding polygon color that is stored in frame buffer.

ii) if the polygon is outside the viewpot then background color of window plane will be done & polygon will be ignored.

iii) if the polygon is inside the window panel then the color the polygon from its corresponding color & color the rest of surface with bkcolor.

iv) If poly & viewpot intersect each other then following cases are there

a) Fill the overlapped region with poly. color & is set in the frame buffer

b) If we are given more than one polygon, with overlapped surfaces w.r. to viewpot then first find Zmin (depth) in order to find surface which is closer to viewpot & will fill the overlapped region with the color of that polygon that has

5 Repeat all steps for all given polygons then exit.