Tribhuvan University
Institute of Science and Technology
Bachelor of Science in
Computer Science and Information Technology

3RD Semester

CSC209: Computer Graphics

-RAJESH KUMAR BAJGAIN

U7:Visible Surface Detections 5 Hrs

7.1 Image Space and Object Space Techniques

7.2 Back Face Detection, Depth Buffer (Z-buffer), A-Buffer and Scan-Line Algorithms.

7.3 Depth Sorting Method (Painter's Algorithm)

7.4 BSP tree Method, Octree and Ray Tracing

Visible Surface Detection (Hidden Surface Removal) Method

The process of identifying those parts of a scene that are visible from a chosen viewing position.

Numerous algorithms for efficient identification of visible objects for different types of applications.

These algorithms are referred to as visible-surface detection methods or hidden-surface elimination methods.

Visible Surface Detection (Hidden Surface Removal) Method

To identify those parts of a scene that are visible from a chosen viewing position (**visible-surface detection methods**).

Surfaces which are obscured by other opaque (solid) surfaces along the line of sight are invisible to the viewer so can be eliminated (**hidden-surface elimination methods**).

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

1. Object-Space methods(OSM)
2. Image-Space methods(ISM)

Visible Surface Detection (Hidden Surface Removal) Method

Object-Space methods(OSM):

- Deal with object definition
- Compares objects and parts of objects to each other within the scene definition to determine which surface as a whole we should label as visible.
- E.g. Back-face detection method

Visible Surface Detection (Hidden Surface Removal) Method

Image-Space methods(ISM):

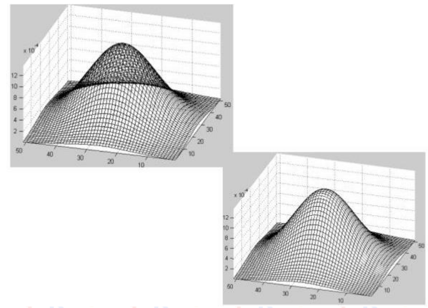
- Deal with projected image
- Visibility is decided point by point at each pixel position on the projection plane.
- E.g. Depth-buffer method, Scan-line method, Area-subdivision method

Most visible surface detection algorithm use image-space-method but in some cases object space methods are also used for it.

Visible Surface Detection (Hidden Surface Removal) Method

List Priority Algorithms

- This is a hybrid model that combines both object and image precision operations. Here, depth comparison & object splitting are done with object precision and scan conversion (which relies on ability of graphics device to overwrite pixels of previously drawn objects) is done with image precision.
- E.g. Depth-Shorting method, BSP-tree method



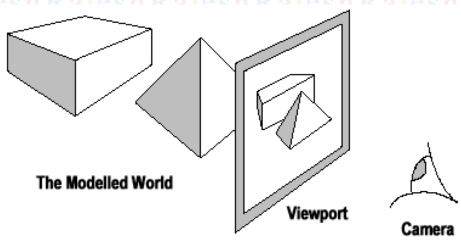


Figure 1. The Basics of 3D Graphical Processing

Object-Space methods

* Algorithms to determine which parts of the shapes are to be rendered in 3D coordinates.

* Methods based on comparison of objects for their 3D positions and dimensions with respect to a viewing position.

* For N objects, may require N*N comparison operations.

* Efficient for small number of objects but difficult to implement.

* Depth sorting, area subdivision methods.

Object-Space methods

* Deals with object definitions directly.

* Compare objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.

* It is a continuous method.

* Compare each object with all other objects to determine the visibility of the object parts.

Image Space Methods

* Deals with the projected images of the objects and not directly with objects.

* Visibility is determined point by point at each pixel position on the projection plane.

* It is a discrete method.

* Accuracy of the calculation is bounded by the display resolution.

* A change of display resolution requires re-calculation

* Based on the pixels to be drawn on 2D. Try to determine which object should contribute to that pixel.

Image Space Methods

* Running time complexity is the number of pixels times number of objects.

* Space complexity is two times the number of pixels:

- One array of pixels for the frame buffer

- One array of pixels for the depth buffer

* Coherence properties of surfaces can be used.

* Depth-buffer and ray casting methods.

Back – Face Detection Method (Plane equation Method)

• A fast and simple object-space method for identifying the back faces of a polyhedron.

• It is based on the performing inside-outside test.

TWO METHODS:

First Method:

• A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C, and D if $Ax+By+Cz+D < 0$ (from plane equation).

• When an inside point is along the line of sight to the surface, the polygon must be a back face.

Back – Face Detection Method (Plane equation Method)

• In eq. $Ax+By+Cz+D=0$

if A,B,C remain constant , then varying value of D result in a whole family of parallel plane

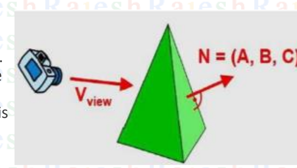
if D>0, plane is behind the origin (Away from observer)

if D<0 , plane is in front of origin (toward the observer)

Back – Face Detection Method (Plane equation Method)

Second Method:

• Let N be normal vector 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 (or "camera") position, then this polygon is a back face if V.N>0



Back – Face Detection Method (Plane equation Method)

In a left-handed viewing system, the viewer's camera is positioned so that the positive x-axis points to the right, the positive y-axis points up, and the positive z-axis points away from the viewer (towards the back of the scene).

For the left handed viewing system if the 'z' component of the normal vector is positive, then it is back face. If the 'z' component of the vector is negative then it is a front face.

Back – Face Detection Method (Plane equation Method)

In a right-handed viewing system, the viewer's camera is positioned so that the positive x-axis points to the right, the positive y-axis points up, and the positive z-axis points towards the viewer (towards the front of the scene).

For the right handed viewing system if the 'z' component of the normal vector is negative, then it is back face. If the 'z' component of the vector is positive then it is a front face.

Back – Face Detection Method (Plane equation Method)

Case-I: (FRONT FACE)

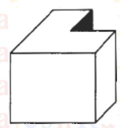
If V.N < 0 the face is visible else face is hidden

Case-II: (BACK FACE)

If V.N > 0 the face is visible else face is hidden

Case-III:

For other objects, such as the concave polyhedron in Fig., more tests need to be carried out to determine whether there are additional faces that are totally or partly obscured by other faces.



Advantage

It is a simple and straight forward method.

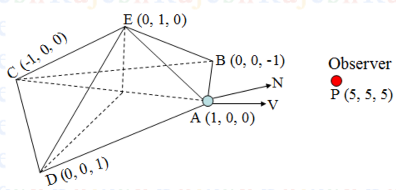
It reduces the size of databases, because no need of store all surfaces in the database, only the visible surface is stored.

Limitations

- This method works fine for convex polyhedral, but not necessarily for concave polyhedral or overlapping objects. So, we need to apply other methods to further determine where the obscured faces are partially or completely hidden by other objects (e.g. Using Depth-Buffer Method or Depth-sort Method).

- This method can only be used on solid objects modeled as a polygon mesh.

Find the visibility for the surface AED in rectangular pyramid where an observer is at P (5, 5, 5).



Find the visibility for the surface AED in rectangular pyramid where an observer is at P (5, 5, 5).

Here,

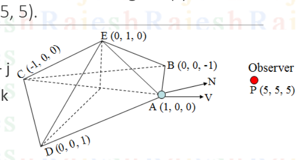
AE = (0-1)i + (1-0)j + (0-0)k = -i + j

AD= (0-1)i + (1-0)j + (1-0)k =-i + k

Step-1:

Normal vector N for AED

Thus, N = AE x AD = $\begin{pmatrix} i & j & k \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$ = i (1-0) –j (-1+0) +k (0+1)
= i + j + k



Find the visibility for the surface AED in rectangular pyramid where an observer is at P (5, 5, 5).

Case-II

Step-2: If observer at P(5, 5, 5) so we can construct the view vector V from surface to view point A(1, 0, 0) as:

V = AP = (5-1)i + (5-0)j + (5-0)k = 4i + 5j + 5k

Step-3: To find the visibility of the object, we use dot product of view vector V and normal vector N as:

V.N = (4i + 5j + 5k).(i + j + k) = 4+5+5 = 14> 0

This shows that the surface is visible for the observer.

Find the visibility for the surface AED in rectangular pyramid where an observer is at P (5, 5, 5).

Case-I

Step-2: If observer at P(5, 5, 5) so we can construct the view vector V from surface to view point A(1, 0, 0) as:

V = PA = (1-5)i + (0-5)j + (0-5)k = -4i - 5j - 5k

Step-3: To find the visibility of the object, we use dot product of view vector V and normal vector N as:

V.N = (-4i - 5j - 5k).(i + j + k) = -4-5-5 = -14< 0

This shows that the surface is visible for the observer.

Find the visibility for the surface AED in rectangular pyramid where an observer is at P (0,0.5, 0).

Depth – Buffer (Z – Buffer Method)

- A commonly used **image-space approach** to detect visible surfaces.
- Also called z-buffer method since depth usually measured along z-axis.
- This approach compares surface depths at each pixel position on the projection plane.
- Each surface of a scene is processed separately, one point at a time across the surface.
- And each (x, y, z) position on a polygon surface corresponds to the projection point (x, y) on the view plane.

Depth – Buffer (Z – Buffer Method)

This method requires two buffers:

- A **z-buffer** or **depth buffer**: Stores depth values for each pixel position (x, y).
- **Frame buffer** (**Refresh buffer**): Stores the surface-intensity values or color values for each pixel position.
- 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.

Depth – Buffer (Z – Buffer Method)

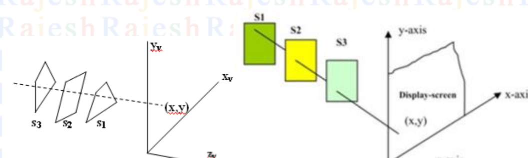
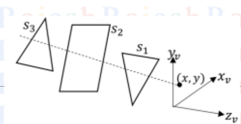


fig.1.

Algorithm

1) Set the buffer values:
   DepthBuffer(x, y)=INFINITE,
   FrameBuffer(x, y)=Background color
2) Process each polygon, one at a time as follows:
   - For each projected (x, y) pixel position of a polygon, calculate depth 'z'.
     - If z<DepthBuffer(x,y), compute surface color and set
       DepthBuffer(x, y)=z,
       FrameBuffer(x, y)=surfacecolor(x, y)

After all surface have been processed, the depth buffer contains depth values for visible surface and frame buffer contains corresponding color values for those surface.



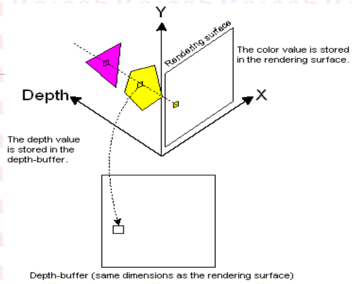
In the figure, at view plane position (x, y), surface $s1$ has the smallest depth from the view plane, so it is visible at that position.

Depth value for a surface position (x, y) is

$z = (-Ax - By - D)/c$

Now, the depth z' of the next position (x+1, y) is obtained as

$z' = z - A/c$



Depth – Buffer (Z – Buffer Method)

Advantages:

- It is easy to implement.
- It reduces the speed problem if implemented in hardware.
- It processes one object at a time

Disadvantages:

- It requires large memory
- It is a time consuming process as it requires comparison for each pixel instead of for the entire polygon.
- Deals only with opaque object but not for transparent object.

A-Buffer Method

• The A-buffer (anti-aliased, area-averaged, accumulation buffer) is an extension of the ideas in the depth-buffer method.

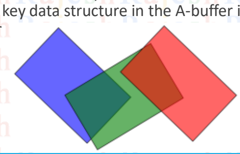
• A drawback of the depth-buffer method is that it deals only with opaque(Solid) surfaces and cannot accumulate intensity values for more than one transparent surfaces.

• The A-buffer is incorporated into the REYES ("Renders Everything You Ever Saw") 3-D rendering system. accumulation buffer

A-Buffer Method

• The A-buffer method calculates the surface intensity for multiple surfaces at each pixel position, and object edges can be anti aliased.

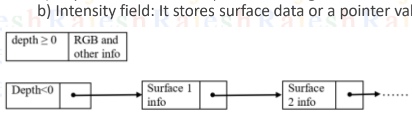
• The A-buffer expands on the depth buffer method to allow transparencies. The key data structure in the A-buffer is the accumulation buffer



A-Buffer Method

It consists of accumulation(A) buffer which as two fields:

a) Depth field: It stores a positive or negative real number.
b) Intensity field: It stores surface data or a pointer value.



A-Buffer Method

If depth field is non negative (>=0), it indicates single surface.

The intensity field then stores the RGB components of the surface color at that point and the percent of pixel coverage.

If depth is negative (<0) it indicates multiple surface.

The intensity field then stores a pointer to a linked list of surface data.

Surface information in A-buffer includes:

- Depth - Surface identifier - Opacity parameter
- Percentage of area coverage - RGB intensity component
- Pointer to the next surface

A-Buffer Method

Advantage:

- It provides anti-aliasing in addition to what Z-buffer does.

Disadvantage:

- It is slightly costly than Z-buffer method because it requires more memory in comparison to the Z-buffer method.

Scan-Line Algorithms

- It is an image-space method for identifying visible surface.
- It computes and compares depth values along the various scan lines for the scene.
- Surfaces are processed using information stored in the polygon table.
- An active list of edges is formed for each scan line which stores only those edges that crosses the scan line in order of increasing 'x'.

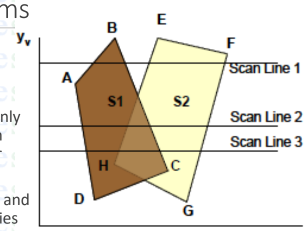
Scan-Line Algorithms

- Also a flag is set for each surface that is set on or off to indicate whether a position along a scan line is either inside or outside the surface.
- Pixel position across each scan-line are processed from left to right.
- At the left intersection with a surface the surface flag is turned on and at the right intersection point the flag is turned off.
- We only need to perform depth calculation when more than one surface has its flag turned on at a certain scan-line position

Scan-Line Algorithms

For scan line 1

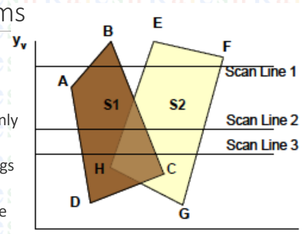
- The active edge list contains edges AB,BC,EH, FG
- Between edges AB and BC, only flags for s1 == on and between edges EH and FG, only flags for s2==on
- no depth calculation needed and corresponding surface intensities are entered in refresh buffer



Scan-Line Algorithms

For scan line 2

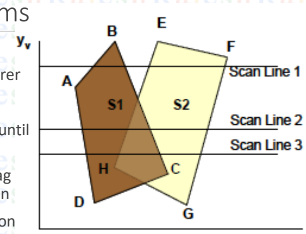
- The active edge list contains edges AD,EH,BC and FG
- Between edges AD and EH, only the flag for surface s1 == on
- Between edges EH and BC flags for both surfaces == on
- Depth calculation (using plane coefficients) is needed.



Scan-Line Algorithms

For scan line 2

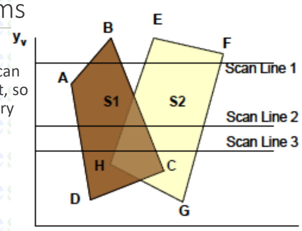
- In this example ,say s2 is nearer to the view plane than s1, so intensities for surface s2 are loaded into the refresh buffer until boundary BC is encountered
- Between edges BC and FG flag for s1==off and flag for s2 == on
- Intensities for s2 are loaded on refresh buffer



Scan-Line Algorithms

For scan line 3

- Same coherent property as scan line 2 as noticed from active list, so no depth calculation is necessary

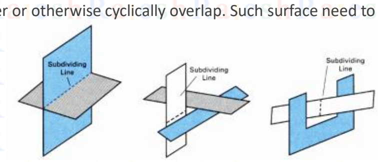


Advantage:

- Any number of overlapping polygon surfaces can be processed with this method.

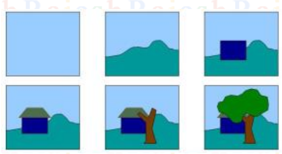
Limitation:

- The scan line method runs into trouble when surface cut through each other or otherwise cyclically overlap. Such surface need to be divided.



Depth Sorting Method (Painter's Algorithm)

- A visible surface detection method that uses both object space and image space 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.



Depth Sorting Method (Painter's Algorithm)

It follows following steps:

i. All surfaces in the scene are ordered according to the smallest 'z' value on each surface.

ii. The surface 'S' at the end of the list is then compared against all other surface to see if there are any depth overlap.

iii. If no overlap occurs then the surface is scan converted and the process is repeated with the next surface.

Depth Sorting Method (Painter's Algorithm)

When there is depth overlap with 'S' we make the following tests:

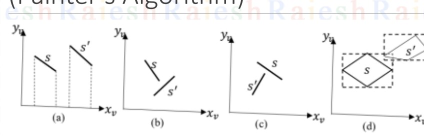
a) The bounding rectangle for the two surfaces do not overlap.

b) Surface 'S' is completely behind the overlapping surface relative to viewing position.

c) The overlapping surface is completely in front of 'S' related to viewing position.

d) The boundary edge projection of the two surfaces do not overlap.

Depth Sorting Method (Painter's Algorithm)



If any of this test is true, no reordering is necessary.

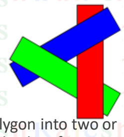
But if all four tests fail then we have to interchange surface S and S' in sorted list.

DEPTH SORT (Painter Algorithm):

Problem: Intersecting polygon surfaces.

o Different polygons may have same depth.

o The nearest polygon could also be farthest.



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.

BSP tree Method

A 3-D scene is split in two using a 2-D plane, then again that scene is divided in two using a 2-D plane, and so on.

The resulting data structure is a binary tree, or a tree where every node has two branches.

The technique is widely used to speed up rendering of 3-D scenes, especially in games.

BSP tree Method

The structure of a BSP tree allows spatial information about the objects in a scene that is useful in rendering,

such as their ordering from front-to-back with respect to a viewer at a given location, to be accessed rapidly.

Other applications include performing geometrical operations with shapes (constructive solid geometry) in CAD,

collision detection in robotics and 3D video games, ray tracing and other computer applications that involve handling of complex spatial scenes.

BSP tree Method

This method has two steps:

• building of the tree independently of viewpoint

• traversing the tree from a given viewpoint to get visibility ordering

BSP tree is a general solution, but with its own problems

• Tree size • Tree accuracy

Other disadvantages are:

- More polygon splitting may occur than in painter's algorithm.

- Appropriate partitioning hyperplane selection is quite complicated and difficult.

Building a BSP Tree (Recursive)

• Start with a set of polygons and an empty tree

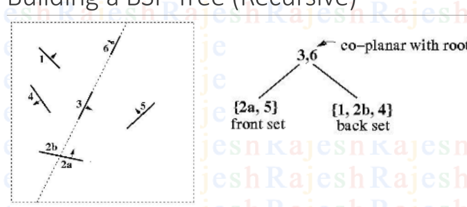
• Select one of them and make it the root of the tree

• Use its plane to divide the rest of the polygons in 3 sets: front, back, coplanar.

• Any polygon crossing the plane is split

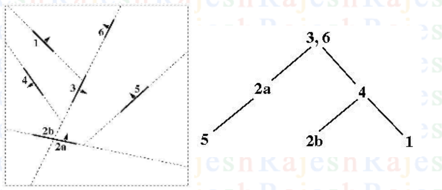
• Repeat the process recursively with the front and back sets, creating the front and back sub-trees respectively

Building a BSP Tree (Recursive)



Select one polygon and partition the space and the polygons

Building a BSP Tree (Recursive)



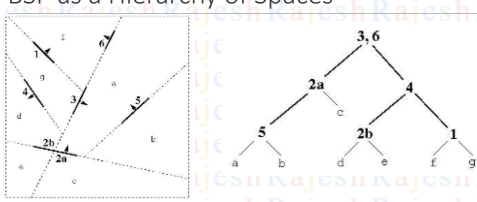
Recursively partition each sub-tree until all polygons are used up

Building a BSP Tree (Incremental)

The tree can also be built incrementally:

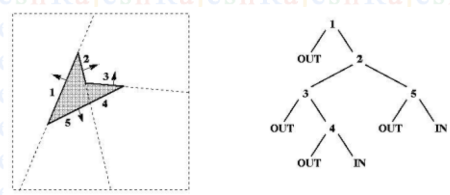
- start with a set of polygons and an empty tree
- insert the polygons into the tree one at a time
- insertion of a polygon is done by comparing it against the plane at each node and propagating it to the right side, splitting if necessary
- when the polygon reaches an empty cell, make a node with its supporting plane

BSP as a Hierarchy of Spaces



Each node corresponds to a region or surface and the leaves are homogeneous regions

Representation of Polygons



Area-Subdivision Method

The total viewing area is successively divided into smaller and smaller rectangles until each small area is simple

i.e. it is a single pixel or is covered wholly by a part of a single visible surface or no surface at all.

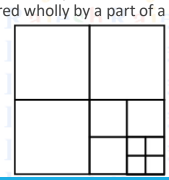


Fig: Dividing a square area into equal-sized quadrants at each step.

Area-Subdivision Method

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- One that completely encloses the area.
- Overlapping surface- One that is partly inside and partly outside the area.
- Inside surface: One that is completely inside the area.
- Outside surface: One that is completely outside the area.

Area-Subdivision Method

2) Check the result from 1, if one of the following condition is true, then no further subdivision of this area is needed.

a) All surface are outside the area.

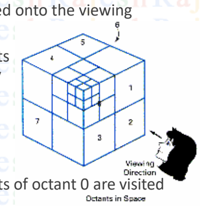
b) Only one surface is inside, overlapping or surrounding surface is in the area.

c) A surrounding surface obscures all other surfaces within the area boundaries.

Octree

In this representation 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.



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 sub-octants of octant 0 are visited before the nodes for the four back suboctants.

Octree

When a color is encountered in an octree node, the corresponding 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 same 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.

Ray-casting Method

Ray casting is a simpler and faster method compared to ray tracing.

In ray casting, a single ray is cast from the viewer's perspective for each pixel on the screen to determine the color and visibility of that pixel.

The rays are typically not traced further after they intersect with an object, which means ray casting only detects the first object hit by each ray.

Ray casting does not handle complex lighting effects like reflections, refractions, and shadows, making it more suitable for simpler and faster rendering tasks.

Ray-casting Method

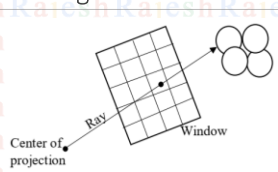


Fig: A ray is fired from the center of projection through each pixel to which the window maps, to determine the closest object intersected.

Ray-Tracing Method

Ray tracing is a more advanced and computationally intensive method.

In ray tracing, rays are traced not only from the viewer's perspective but also from the light sources, creating more realistic lighting and shadow effects.

Each ray is recursively traced through the scene, allowing for complex interactions like reflections (rays bouncing off surfaces) and refractions (rays passing through transparent materials).

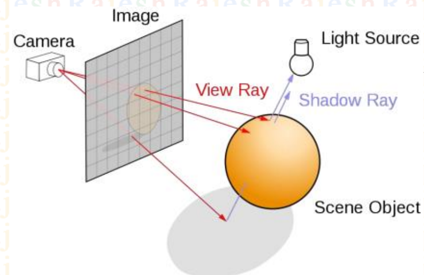
Ray tracing is capable of simulating a wide variety of optical effects, such as reflection and refraction, scattering,and dispersion phenomena

Ray-Tracing Method

Ray tracing can simulate more advanced lighting models such as global illumination, caustics, and soft shadows, leading to highly realistic renderings.

It is computationally demanding, requiring significant processing power, and often relies on optimization techniques like bounding volume hierarchies or acceleration structures to speed up rendering.

Ray tracing is best suited for applications where taking a relatively long time to render a frame can be tolerated, such as in still images and film and television visual effects, and more poorly suited for real-time applications such as video games where speed is critical.



END OF UNIT