Hidden surface removal

Object-space approach
Image-space approach

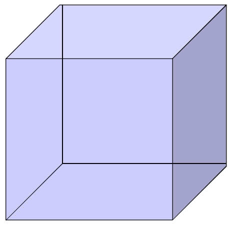
Depth-sorting methods
Scan-line methods
Depth buffer methods

Visible surface determination

- An opaque cube has at most 3 sides visible

Problems to solve:

- Which sides are visible from a given viewpoint? Which are hidden?
- How to make the rendering efficient?

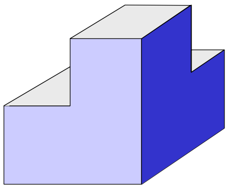


Visible surface determination

- An opaque cube has at most 3 sides visible

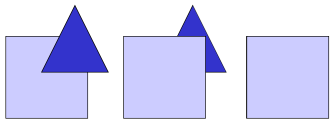
Problems to solve:

- Which sides are visible from a given viewpoint? Which are hidden?
- How to make the rendering efficient?

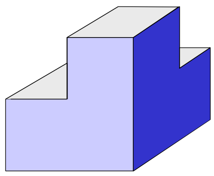


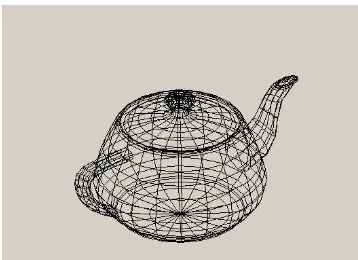
Cases for hidden surface removal

- Occluded surfaces (partially or fully)

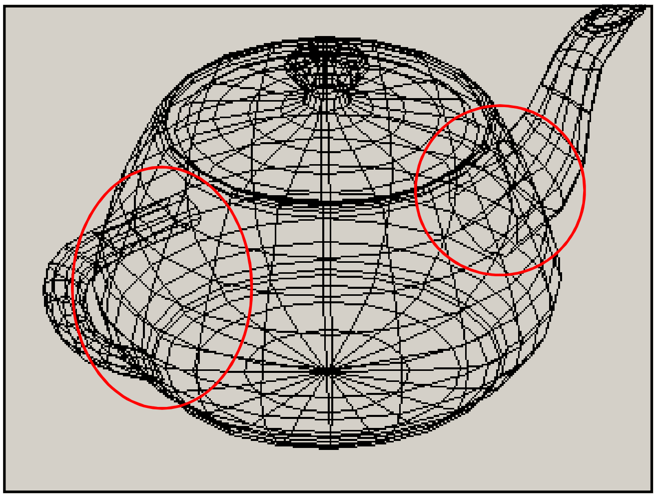


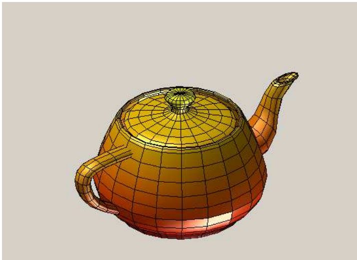
- Back faces



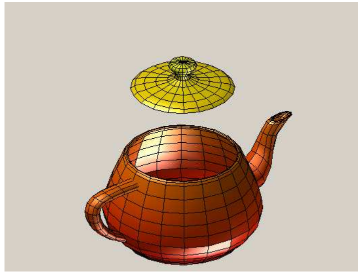


Utah Teapot - wireframe





Utah Teapot – surface rendering - outside



Utah Teapot – surface rendering – inside and outside

Two classes of algorithms

- Object-space (OS)
  - Operates on 3D object entities (vertices, edges, surfaces)
- Image space (IS)
  - Operates on 2D images (pixels)
- Operations normally applied to polygonal representations of objects (e.g. triangulated surfaces)

Back face removal (Polygon culling): OS

Principles

- Remove all surfaces pointing away from the viewer
- Eliminate the surface if it is completely obscured by other surfaces in front of it
- Render only the visible surfaces facing the viewer

How to determine which surfaces face away from the viewer?

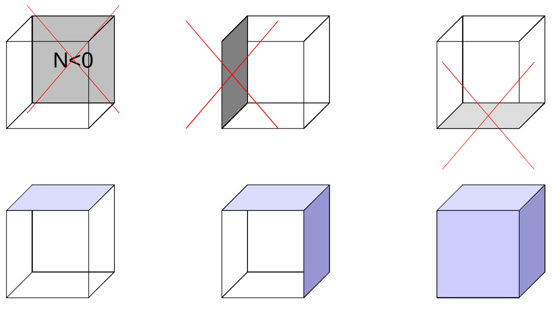
- Compute vector **N** normal to a surface patch (e.g. a triangle)
- In the right-handed coordinate system a surface facing away from the viewer will have <u>negative</u> value of z-coordinate of the normal vector: $N_Z<0$

Details were discussed in lecture “Object rendering”

Back face removal (Polygon culling)

- Algorithm
  - Compute 3D coordinates of the object in the camera coordinate system
  - For each surface patch compute the normal vector N
  - If the z-coordinate of the normal vector $N_z < 0$ the surface patch does not need to be displayed
- Advantages / problems
  - Speeds up rendering
  - Works only for convex objects

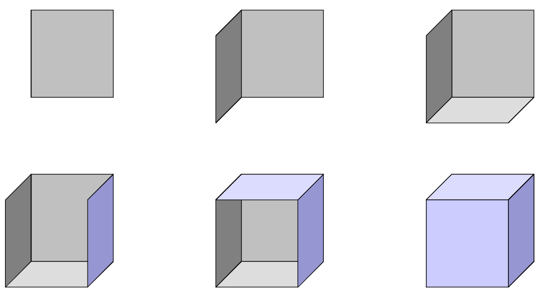
Back face removal (Polygon culling)



Painter's algorithm (depth sorting method): OS/IS

- Surfaces sorted in order of increasing depth in 3D
- Surfaces drawn starting with the surface of greatest depth (largest Z-coordinate)
- Surfaces with smaller depth are "painted over" surfaces that are already scan-converted.

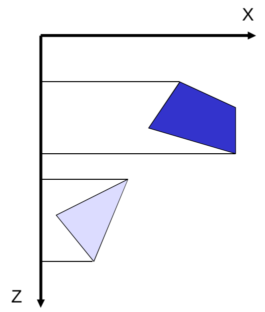
Painter's algorithm Example for a cube



Painter's algorithm

Steps of the algorithm

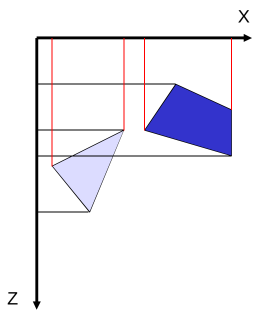
- Order surfaces according to the largest z value
- Select surface with the greatest depth
- Select next greatest-depth surface
- If no depth overlap occurs - paint the surfaces



No depth overlap

Painter's algorithm

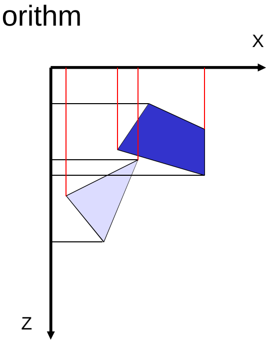
- If depth overlap then carry out a number of tests to establish whether surface projections (X and Y) overlap
  - For detail see Hearn & Baker



Depth overlap
No X projection overlap

Painter's algorithm

- If depth overlap then carry out a number of tests to establish whether surface projections (X and Y) overlap
  - For detail see Hearn & Baker
- If all tests fail, swap the order in which surfaces are painted

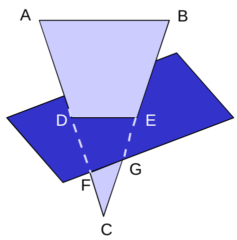


Depth overlap
X projection overlap

Painter's algorithm

- Advantages:
  - Very good if a valid order is easy to establish; not so good for more complex surface topolgies (e.g. presence of holes)
  - For simple cases very easy to implement
  - Fits well with the rendering pipeline
- Problems
  - Not very efficient – all polygons are rendered, even when they become invisible
  - Complex processing (sorting + tests) for complex objects
  - There could be no solution for sorting order
  - Possibility of an infinite loop as surface order is swapped

Painter's algorithm

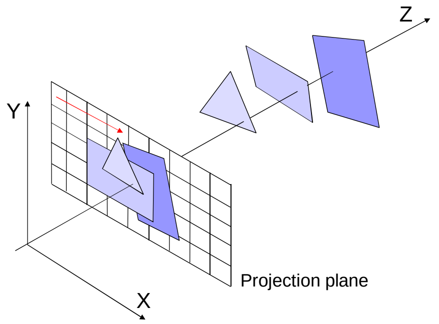


No solution for sorting order

Z-buffer algorithm (IS)

- Test visibility of surfaces **one point at a time**
- The surface with the z-coordinate closest to VRP is visible (largest z in RH coordinate sytem; smallest z in LH coordinate system)
- Two storage areas required:
  - depth buffer - z value for **each pixel** at (x,y)
  - display buffer – pixel value (colour) for each pixel at (x,y)

Z-buffer algorithm



Z-buffer algorithm

```
fill z_buffer with "infinite" distance
for each polygon
   compute 2D projections (rasterize)
   for each pixel (x,y) inside the rasterized polygon
        calculate z-value (see next slide)
        if z(x,y) is closer than z_buffer(x,y)
                display_buffer(x,y)=polygon colour at (x,y)
                z_buffer(x,y)=polygon z(x,y)
        end
   end
end
```

Z-buffer algorithm – computing depth

- Depth values can be evaluated iteratively using the plane equation ($Ax + By + Cz + D = 0$ see next slide)
  - for x increments, along each scan line
    (x+1, y):
    $\mathbf{z' = z - A/C}$
  - for y increments, for each new scan line
    (x, y+1):
    $\mathbf{z'' = z + B/C}$

Compute A, B and C from the plane equation

Construct the equation of the plane passing through three points:

$P1=(x_1, y_1, z_1)$ $P2=(x_2, y_2, z_2)$ $P3=(x_3, y_3, z_3)$

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0$$

Equivalent to: $Ax + By + Cz + D = 0$

e.g.

$C = (x2 - x1)*(y3 - y1) - (x3 - x1)*(y2 - y1)$

Z-buffer algorithm

- Advantages
  - Easy to implement
  - Fits well with the rendering pipeline
  - Can be implemented in hardware
  - Always correct results
- Problems
  - Some inefficiency as pixels in polygons nearer the viewer will be drawn over polygons at greater depth
- It is a standard in many graphics packages (e.g. Open GL)

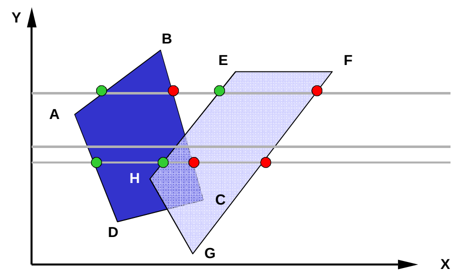
Other methods

Scan-line method (IS)

- Each scan line is processed
- list of edges (of **ALL** polygons) crossing a current line sorted in order of increasing x
- on/off flag for each surface
- depth sorting is necessary if multiple flags are on
- uses coherence along the scan lines to prevent unnecessary sorting
- Necessary to divide surfaces with multiple visibility

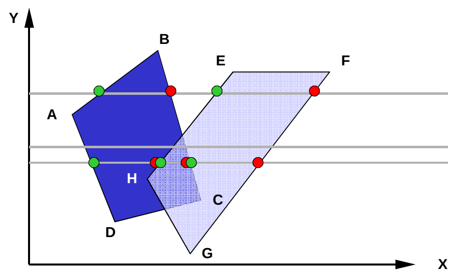
Other methods

Scan-line method (IS)



Other methods

Scan-line method (IS)



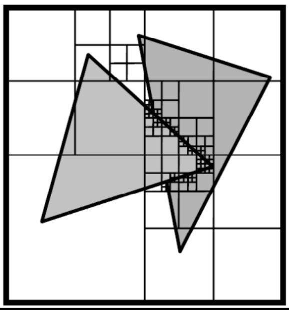
Other methods

Area subdivision method (used also in ray tracing)

- Locate view areas (normally squares or rectangles) which represent a part of a single surface
- This is done by successively dividing the total view area into smaller rectangles
- Stop dividing a given rectangle when it contains a single surface or visibility precedence can be easily determined

Other methods

Area subdivision method



Recommendations for hidden surface methods

Surfaces are distributed in z	Depth sorting
Surfaces are well separated in y	Scan-line or area-subdivision
Only a few surfaces present	Depth sorting or scan-line
Scene with at least a few thousand surfaces	Depth-buffer method or area-subdivision

Next lecture

Texture mapping