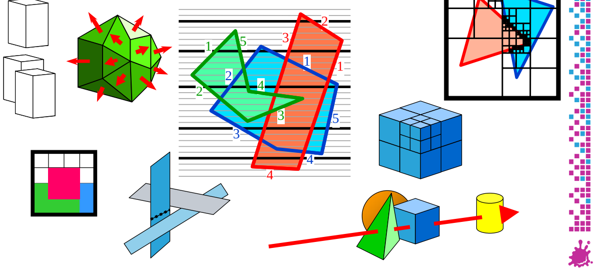
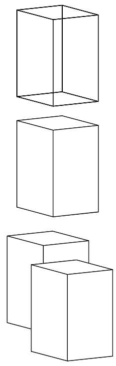
Visible Surface Detection Methods



Visible-Surface Detection

- identifying visible parts of a scene (also *hidden-surface elimination*)
- type of algorithm depends on:
  - complexity of scene
  - type of objects
  - available equipment
  - static or animated displays
- *object-space* methods
  - objects compared to each other
- *image space* methods
  - point by point at each pixel location
- often sorting and coherence used



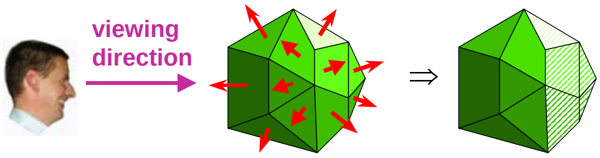
Visible-Surface Detection Methods

- the following algorithms are examples for different classes of methods
  - back-face detection
  - depth buffer method
  - scan-line method
  - depth-sorting method
  - area-subdivision method
  - octree methods
  - ray-casting method

Back-Face Detection (1)

- surfaces (polygons) with a surface normal pointing away from the eye cannot be visible (back faces)

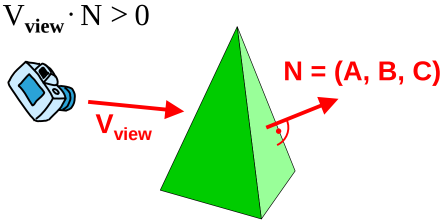
⇒ **eliminate them before visibility algorithm !**



can be eliminated:

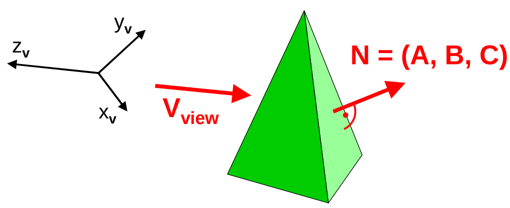
Back-Face Detection (2)

- eliminating back faces of closed polyhedra
- view point (x,y,z) "inside" a polygon surface if
  $$Ax + By + Cz + D < 0$$
- or polygon with normal N=(A, B, C) is a back face if $V_{view} \cdot N > 0$



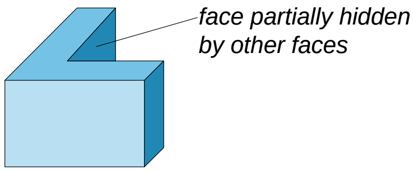
Back-Face Detection (3)

- object description in viewing coordinates ⇒
  $V_{view}=(0,0,V_z)$ $V_{view} \cdot N = V_z C$
- sufficient condition: if $C \leq 0$ then back face



Back-Face Detection (4)

- complete visibility test for non-overlapping convex polyhedra



- preprocessing step for other objects: about 50% of surfaces eliminated

Depth-Buffer Method (1)

- z-buffer method
- image-space method
- hardware implementation
- no sorting!

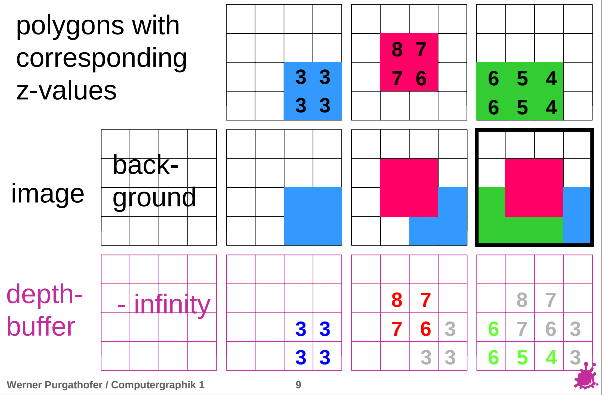
Depth-Buffer Method (2)

- two buffers
  - depth buffer (distance information)
  - refresh buffer (intensity information)

size corresponds to screen resolution (for every pixel: r, g, b, z)

draw something =
- **compare z with z in buffer**
- **if z closer to viewer**
- **then draw and update z in buffer**
- **else nothing!**

Depth-Buffer Algorithm Example



Depth-Buffer Algorithm

```
for all (x,y)
  depthBuff(x,y) = – ∞
  frameBuff(x,y) = backgndColor
for each polygon P
  for each position (x,y) on polygon P
    calculate depth z
    if z > depthBuff(x,y) then
      depthBuff(x,y) = z
      frameBuff(x,y) = surfColor(x,y)
```

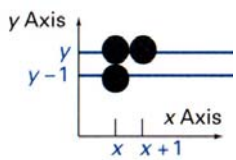

Depth-Buffer: Incremental z-Values

depth at (x,y): $z = \frac{-Ax-By-D}{C}$

depth at (x+1,y): $z' = \frac{-A(x+1)-By-D}{C} = z - \frac{A}{C}$

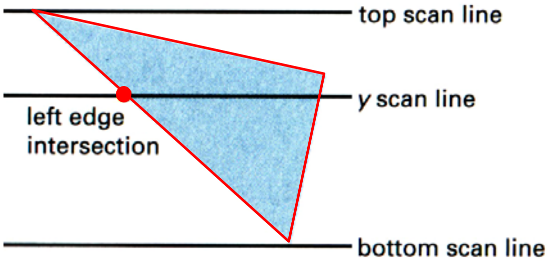
depth at (x,y-1): $z'' = \frac{-Ax-B(y-1)-D}{C} = z + \frac{B}{C}$

constants !



Depth-Buffer: y-Coordinate Intervals

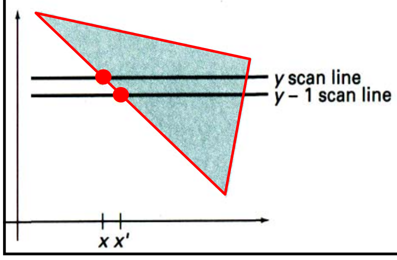
- determine y-coordinate extents of polygon P



Depth-Buffer: Values down an Edge

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

$$x' = x - 1/m \qquad y' = y - 1 \quad \Rightarrow \quad z' = \frac{-A(x-1/m)-B(y-1)-D}{C}$$

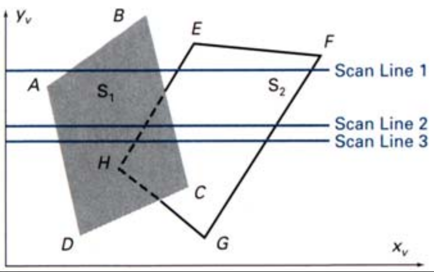


$$= z + \frac{A/m + B}{C}$$

constant !

Scan-Line Method

- image-space method
- extension of scan-line algorithm for polygon filling

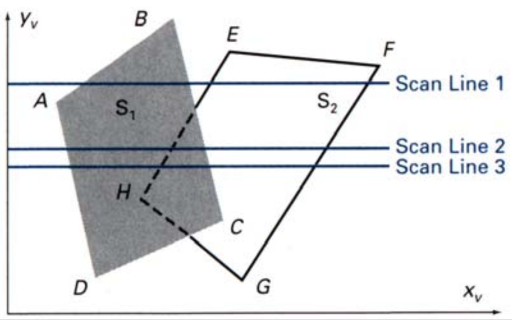


Scan-Line M.: Edge & Polygon Tables

- ***edge table*** (all edges, y-sorted)
  - coordinate endpoints
  - inverse slope
  - pointers into polygon table
- ***polygon table*** (all polygons)
  - coefficients of plane equation
  - intensity information
  - (pointers into edge table)

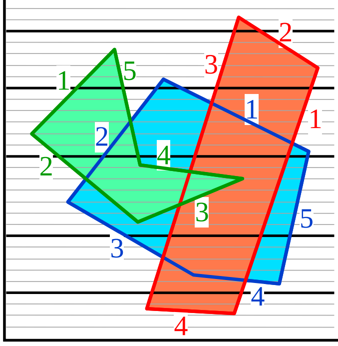
Scan-Line Method: Active Edge List

- active edge list (all edges crossing current scanline, x-sorted, flag)



Scan-Line Method Example

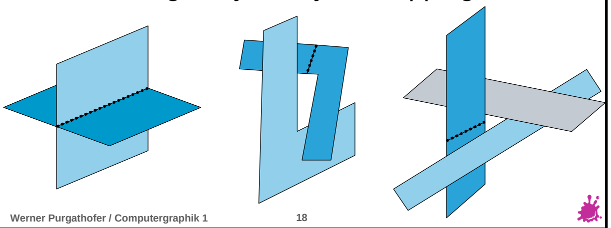
Edge T. 2,3,1,5,1,1,2,2,5,4,3,3,4,4 **Poly.T.** ■, ■, ■



act.edges	act.poly.
3,2	■
1,5,2,1,3,1	■, ■, ■
2,2,5,3,1,5	■, ■, ■
3,3,1,5	■, ■
3,1	■

Scan-Line Method Details

- coherence between adjacent scan lines
  - incremental calculations
  - active edge list very similar
    (easy sorting, avoid depth calculations)
- intersecting or cyclically overlapping surfaces!



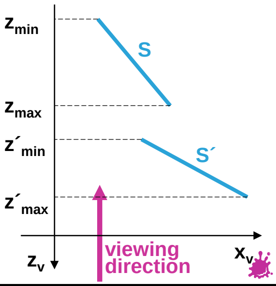
Depth-Sorting Method: Overview

- surfaces sorted in order of decreasing depth (viewing in –z-direction)
  - “approximate”-sorting using smallest z-value (greatest depth)
  - fine-tuning to get correct depth order
- surfaces scan converted in order
- sorting both in image and object space
- scan conversion in image space
- also called “painter’s algorithm”

Depth-Sorting Method: Sorting (1)

- surface S with greatest depth is compared to all other surfaces S’
  - no depth overlap → ordering correct
  - depth overlap → do further tests in increasing order of complexity

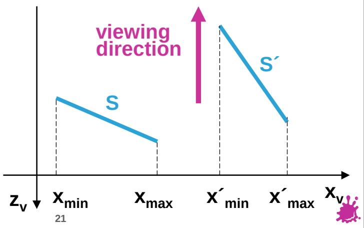
2 surfaces with no depth overlap



Depth-Sorting Method: Sorting (2)

- ordering correct if
  - bounding rectangles in xy-plane do not overlap
  - check x-,y-direction separately

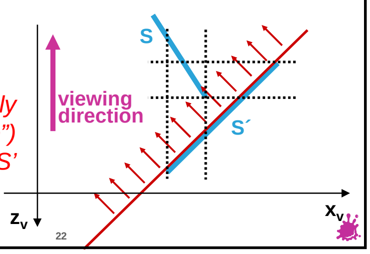
2 surfaces with depth overlap but no overlap in the x-direction



Depth-Sorting Method: Sorting (3)

- ordering correct if
  - S completely behind S’
  - substitute vertices of S into equation of S’

S is completely behind (“inside”) the overlapping S’

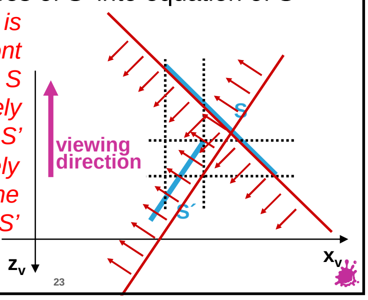


Depth-Sorting Method: Sorting (4)

- ordering correct if
  - S’ completely in front of S
  - substitute vertices of S’ into equation of S

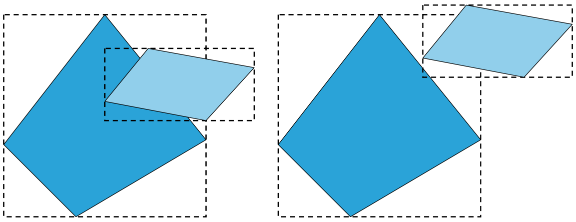
overlapping S’ is completely in front (“outside”) of S, but S is not completely behind S’

*S is **not** completely behind (“inside”) the overlapping S’*



Depth-Sorting Method: Sorting (5)

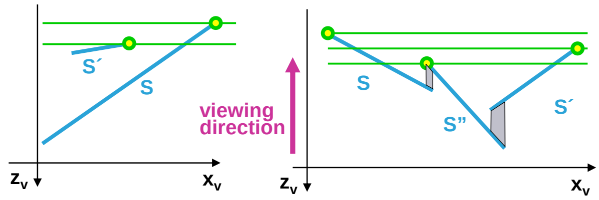
- ordering correct if
  - projections of S,S’ in xy-plane don’t overlap



surfaces with overlapping bounding rectangles

Depth-Sorting Method: Sorting (6)

- all five tests fail ⇒
  - ordering probably wrong
  - interchange surfaces S, S’
  - repeat process for reordered surfaces

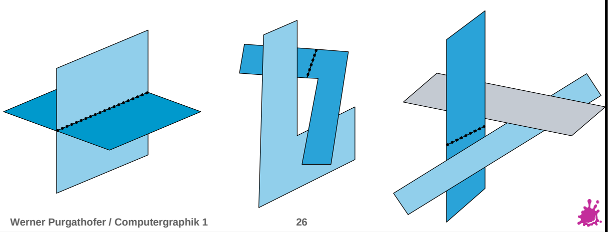


surface S has greater depth but obscures S’

sorted surface list: S, S’, S” should be reordered: S’, S”, S

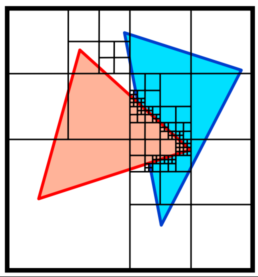
Depth-Sorting: Special Cases

- avoiding infinite loops due to cyclic overlap
  - reordered surfaces S’ are flagged
  - if S’ would have to be reordered again ⇒ divide S’ into two parts



Area-Subdivision Method (1)

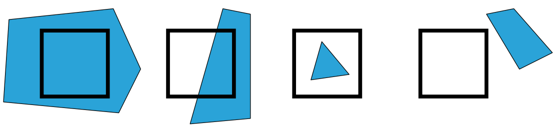
- image-space method
- area coherence exploited
- viewing area subdivided until visibility decision very easy



Area-Subdivision Method (2)

- relationship polygon ⇔ rectangular view area

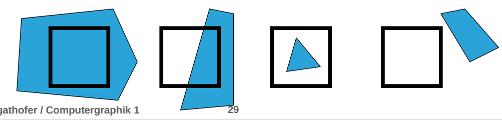
surrounding surface	overlapping surface	inside surface	outside surface



- only these four possibilities

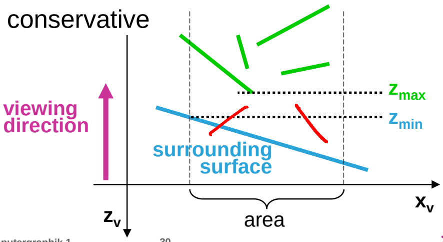
Area-Subdivision Method (3)

- **three easy visibility decisions**
  - *all* surfaces are *outside* of viewing area
    - checking bounding rectangles
  - only *one inside*, overlapping, or surrounding surface is in the area
    - bounding rectangles for initial check
  - one *surrounding* surface obscures all other surfaces within the viewing area
    - minimum depth ordering



Area-Subdivision Method (4)

- a surrounding obscuring surface
  - surfaces ordered according to minimum depth
  - maximum depth of surrounding surface closest to view plane?
  - test is conservative

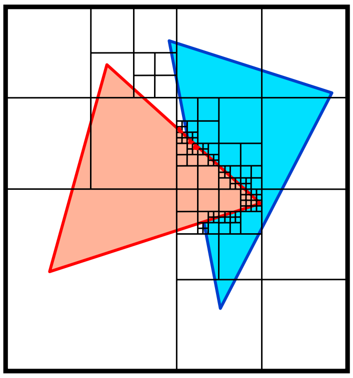


Area-Subdivision Method (5)

- if all three tests fail ⇒ do *subdivision*
  - subdivide area into four equal subareas
  - outside and surrounding surfaces will remain in this status for all subareas
  - some inside and overlapping surfaces will be eliminated
- no further subdivision possible (pixel resolution reached)
  - sort surfaces and take intensity of nearest surface

Area-Subdivision Method Example

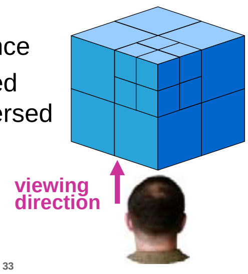
1 2
3 4



finished

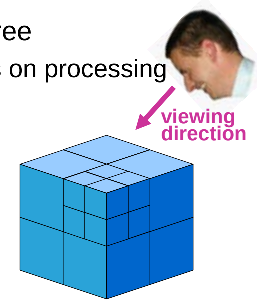
Octree Methods

- recursive traversal of octree
  - traversal order depends on processing direction
- front-to-back:
  - pixel(x,y) written once
  - completely obscured nodes are not traversed
- back-to-front:
  - painter's algorithm



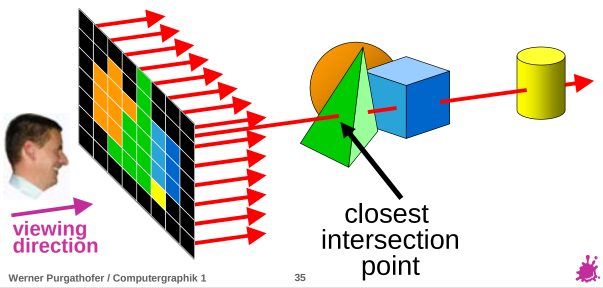
Octree Methods

- recursive traversal of octree
  - traversal order depends on processing direction
- front-to-back:
  - pixel(x,y) written once
  - completely obscured nodes are not traversed
- back-to-front:
  - painter's algorithm



Ray-Casting Method (1)

- line-of-sight of each pixel is intersected with all surfaces
- take closest intersected surface



Ray-Casting Method (2)

- based on geometric optics, tracing paths of light rays
- backward tracing of light rays
- suitable for complex, curved surfaces
- special case of ray-tracing algorithms
- efficient ray-surface intersection techniques necessary
  - intersection point
  - normal vector