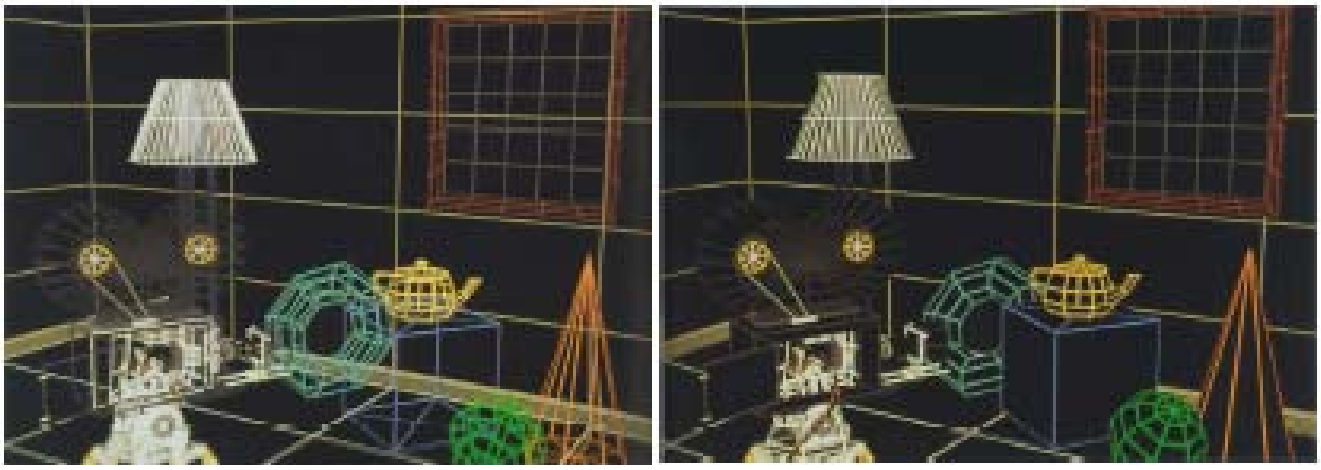
Visible Surface Determination

(Penentuan Permukaan Tampak)

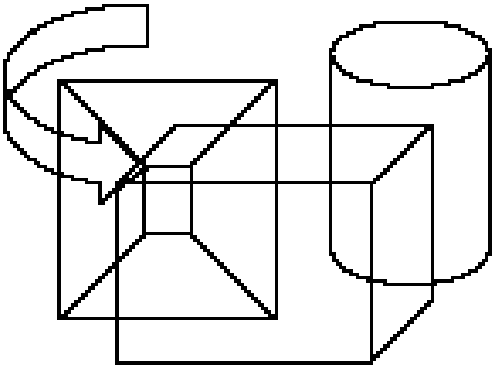


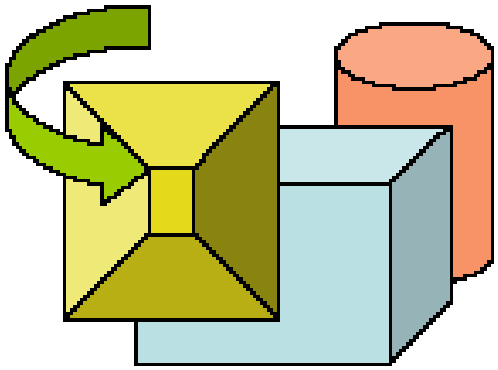
Outline

- Definisi VSD
- Tiga Kelas Algoritma VSD
- Kelas : Conservative
  - Spatial Subdivison
  - Bounding Volume
  - Back Face Culling
- Kelas : Image-Precision
  - Z-Buffer
  - Algoritma Painter
- Kelas : Object-Precision
  - Algoritma 3-D Sort
  - Binary Space Partitioning (BSP)

Visible Surface Determination (1/5)

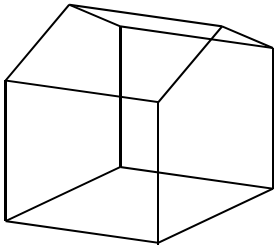
- Definition
  - Given a set of 3-D objects and a view specification (camera), determine which lines or surfaces of the object are visible
  - A surface might be occluded by other objects or by the same object (self occlusion)





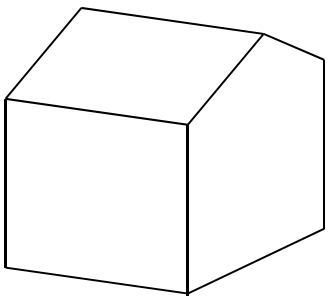
Visible Surface Determination (2/5)

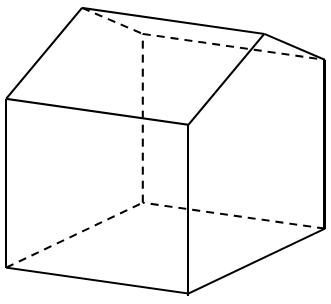
- Historical note
  - Problem first posed for wireframe rendering



canonical house

  - Solution called “hidden-line removal”
    - note: lines themselves don’t hide lines. Lines must be edges of opaque surfaces that hide other lines
    - some techniques show hidden line segments as dotted or dashed lines





Visible Surface Determination (3/5)

- Three classes of algorithms
  - "Conservative" visibility testing: only trivial reject – does not give final answer!
    - e.g., back-face culling, canonical view volume clipping, spatial subdivision
    - have to feed results to algorithms mentioned below
  - Image precision – resolve visibility at discrete points in image
    - sample model, then resolve visibility – i.e., figure out which objects it makes sense to compare with
    - e.g., raytracing, or Z-buffer and scan-line depth buffers (both in hardware!)
  - Object precision – resolve for all possible view directions from a given eye point
    - irrespective of view direction or sampling density
    - resolve visibility exactly, then sample the results
    - e.g., poly's clipping poly's, 3-D depth sort, BSP trees

Visible Surface Determination (4/5)

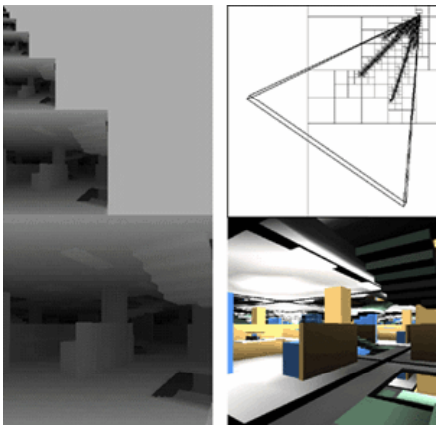
- Criteria to watch for
  - Various differences among algorithms:
    - what sort of geometry? triangles vs. implicit surfaces
    - does it support transparent objects? anti-aliasing?
    - how much of scene has to be considered?
      - e.g., ray-tracing usually able to consider less than Z-buffering
    - must preprocess the model?
    - does it easily handle moving objects?
    - performance: space-time complexity. With large models, even O(*n*) too slow
      - what is *n*? # total objects, # visible objects, # pixels…?
      - really some combination, i.e., # total objects x # pixels
  - Pipelines first use some conservative algorithms, then one of the image-precision or object-precision algorithms

Visible Surface Determination (5/5)

- Exploit coherence
  - The degree to which parts of an environment exhibit logical similarities. Make good guesses! Reuse previous calculations!
  - Image: except at object boundaries, adjacent pixels tend to be from the same object
    - scan-line conversion takes advantage of this
  - Object: objects aren't point clouds – they tend to be continuous
    - scan-line conversion takes advantage of this
  - Visible set: set of visible objects doesn't change much as viewpoint moves incrementally (largely unexploited)
    - most objects don't move that much frame-to-frame
    - background objects tend to stay constant while relatively fewer foreground objects move (games take advantage of this!)
  - Objects tend to be clustered together in space
    - spatial subdivision will take advantage of this

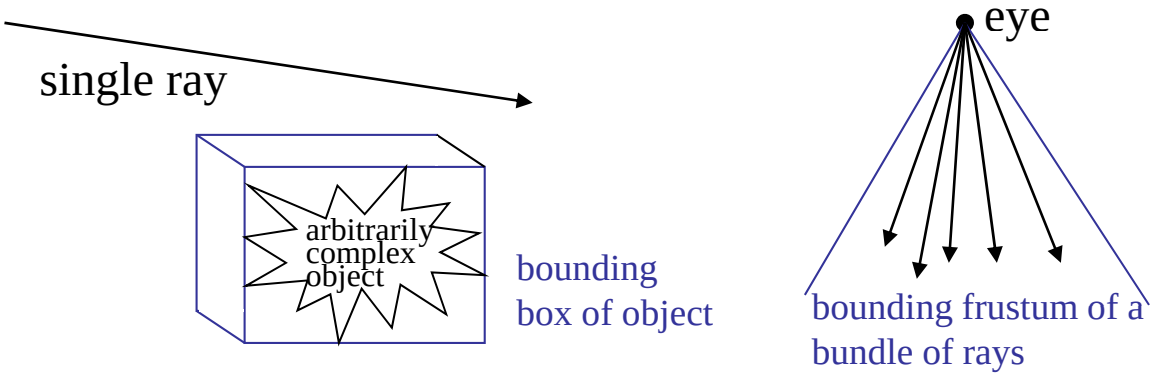
Conservative : Object Database Culling Techniques

- "Conservative" visibility tests
  - Hugely important to minimize the load on the pipeline by culling detail that can't be seen
    - view volume visibility is the most literal application of this principle
    - we only see a small portion of the model from any particular viewpoint, particularly for big models
  - Spatially organize model: spatial subdivision, bounding volume hierarchies
  - Objects occlude parts of themselves: back-face culling
  - Additional techniques (not covered here):
    - take advantage of walls in games and architectural walk-throughs: "cell-portal visibility"
    - use big "blocker" polygons: occlusion culling



Bounding Volumes

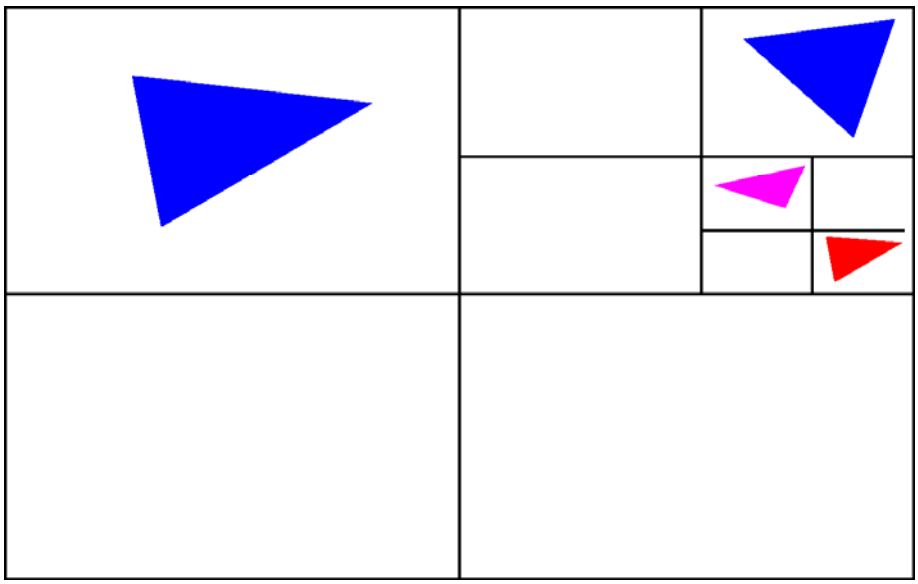
- Extents and Bounding Volumes:
  - bound each complex object with a simpler one
    - examples of simpler volumes: sphere, cuboid
    - Kay & Kajiya SIGGRAPH '86 uses polyhedra
  - if bounding volume isn't visible, neither is object inside it!
  - can put multiple objects into one volume: more efficient
- Great for ray-tracing pipeline!
  - quick reject: check ray against bounding volume first
  - quicker reject: check group of rays (frustum) against bounding volume of object



- Great for traditional hardware pipeline!
  - quick reject large numbers of triangles at a time during clipping
  - e.g., if bounding box of tessellated sphere is not visible, don't have to draw all its triangles!

Spatial Subdivision (1/2)

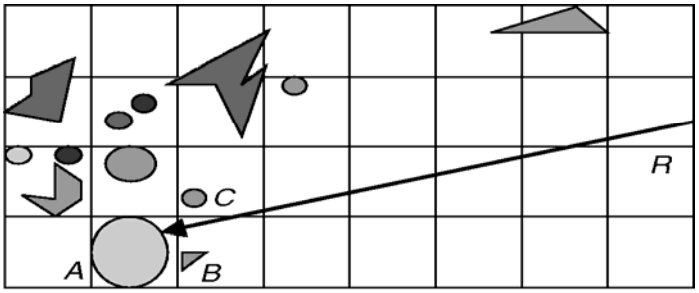
- Another way to organize whole model
  - Top-down construction: divide and conquer!
  - Divide space into cells, place objects in them
    - scan-converter places polygons in the cells
    - raytracer places bounding volumes in the cells



2-D example of spatial subdivision

Spatial Subdivision (2/2)

- Applications for both pipelines
  - Raytracing: rays “walk” through the cells, only checking objects in those cells



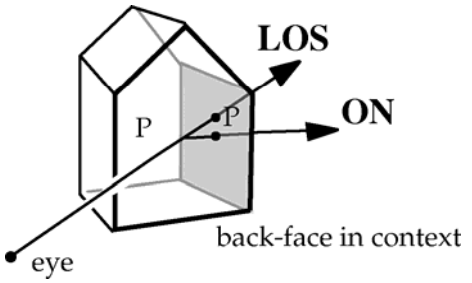
spatial subdivision

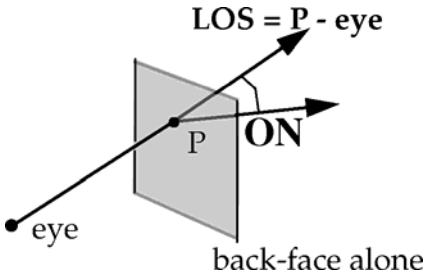
ray R only needs to be intersected with objects A, B, and C

    - start in cell of eye point; skip empty cells; stop walking when we find an intersection
    - speedup: walk bundle of rays through the cells
  - Scan-conversion: clip groups of polygons at a time. Only cells impinging on the view volume have their contents processed

Back-Face Culling

- Line of Sight Interpretation
  - Approach assumes objs defined as closed polyhedra, w/eye pt always outside of them
  - Use *outward normal* (ON) of polygon to test for rejection
  - LOS = *Line of Sight*, the projector from the center of projection (COP) to any point P on the polygon. (For parallel projections LOS = DOP = *direction of projection*)
  - If normal is facing in same direction as LOS, it's a back face:
    - if LOS • ON $\geq$ 0, then polygon is invisible – discard
    - if LOS • ON < 0, then polygon may be visible





  - To render one lone polyhedron, you need back-face culling as VSD!

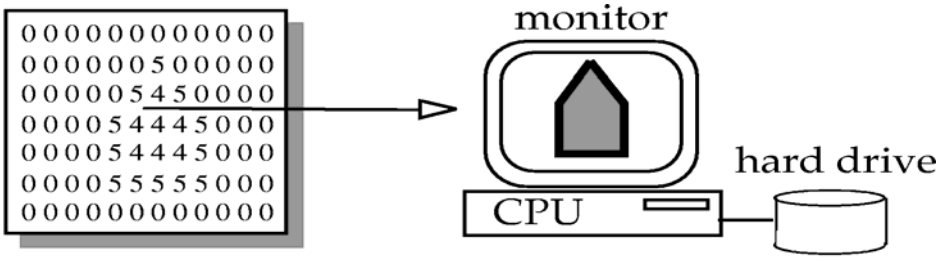
Image Precision : Review of Buffers

- Light in the phosphor display of the CRT monitor decays very quickly. Screen must be frequently refreshed (at least 60 times/second) to maintain a constant image without flickering
- Screen is refreshed one scan line at a time from pixel information held in a *refresh* or *frame buffer*
- Additional buffers can be used to store other pixel information. For example, we will use a *z-buffer* in which *z*-values (depth of points on a polygon) are stored to do VSD

1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	0.2	1.0
1.0	1.0	0.2	0.2	0.2
1.0	0.2	0.2	0.2	0.2
1.0	0.2	0.2	0.2	0.2
1.0	0.2	0.2	0.2	0.2
1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0

z-buffer with depth values for gray subset of frame buffer

0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	5	0	0	0	0	0	0
0	0	0	0	0	5	4	5	0	0	0	0	0
0	0	0	0	5	4	4	4	5	0	0	0	0
0	0	0	0	5	4	4	4	5	0	0	0	0
0	0	0	0	5	5	5	5	5	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

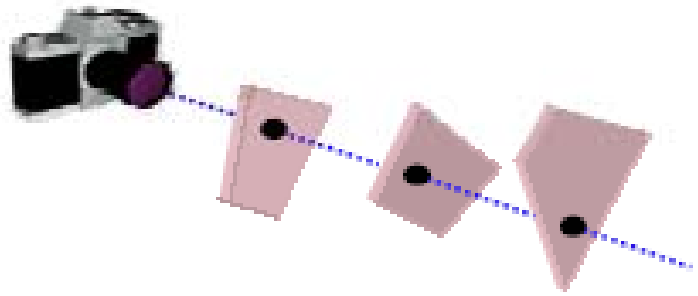


frame buffer with pixmap

Depth-Buffer Method (Z-Buffer) (1/4)

- In addition to the frame buffer (that keeps the pixel values), keep a Z-buffer containing the depth value of each pixel

- Surfaces are scan-converted in an arbitrary order. For each pixel (x, y), the Z-value is computed as well. The pixel (x, y) is overwritten only if it is closer to the viewing plane than the pixel already written at the same location



Depth-Buffer Method (Z-Buffer) (2/4)

Algorithm:

- Initialize the *z*-buffer depth and the frame-buffer I:

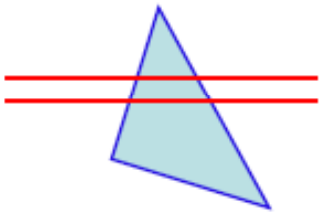
  depth(x,y) = MAX_Z ; I(x, y) = Ibackground

- Calculate the depth z for each (x, y) position on any surface:
  - If z < depth(x, y), then depth(x, y) = z and I(x, y) = Isurf(x,y)

- Very simple implementation in the case of polygon surfaces. Uses polygon scan line conversion, and exploits face coherence and scan-line coherence :
  - z = −(Ax+By+D)/C
  - Along scan lines

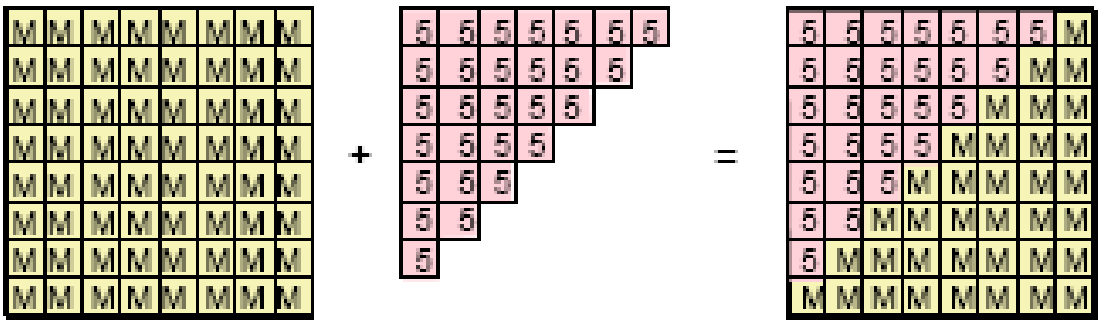
    z'= −(A(x+1)+By+D)/C = z−A/C
  - Between successive scan lines:

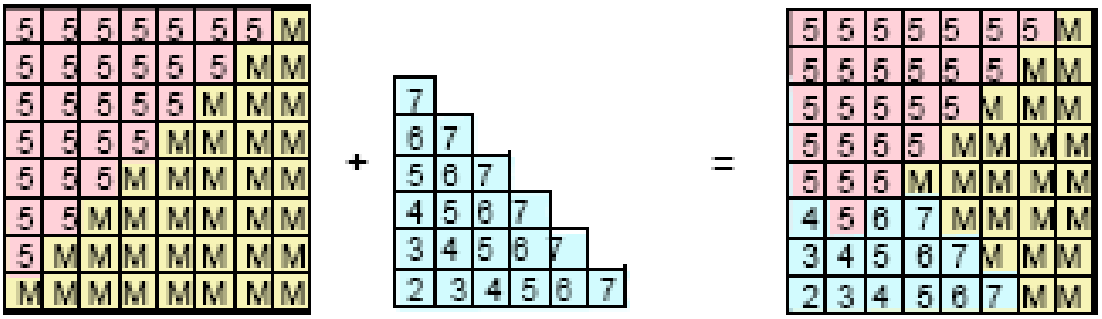
    z'= −(Ax+B(y+1)+D)/C = z−B/C



Depth-Buffer Method (Z-Buffer) (3/4)

- Example :





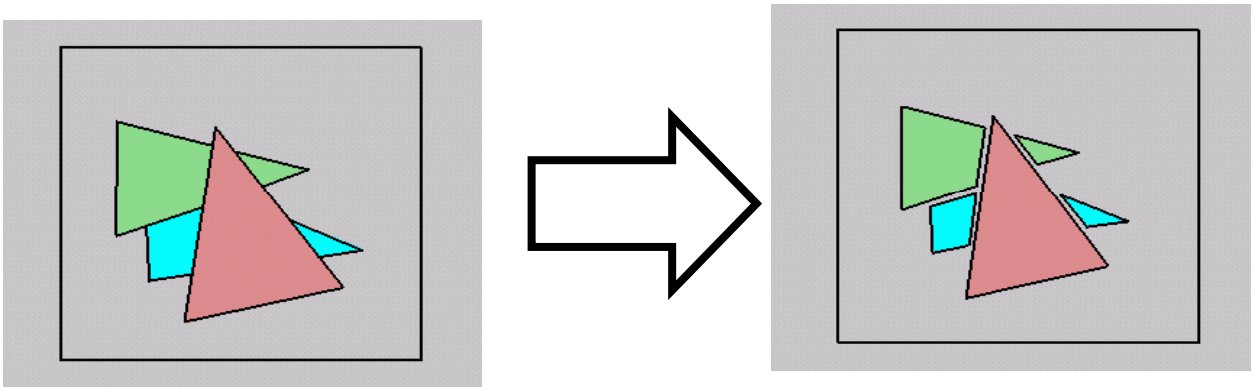
Depth-Buffer Method (Z-Buffer) (4/4)

- Implemented in the image space
- Very common in hardware due its simplicity (SGI)
- 32 bits per pixel for *Z* is common

- **Advantages**:
  - Simple and easy to implement
  - Buffer may be saved with image for re-processing
- **Disadvantages**:
  - Requires a lot of memory
  - Finite depth precision can cause problems
  - Spends time while rendering polygons that are not visible
  - Requires re-calculations when changing the scale

Painter's Algorithm

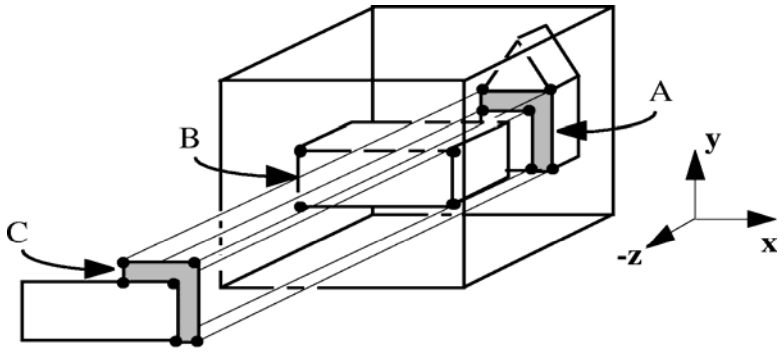
- Simple approach: render the polygons from back to front, "painting over" previous polygons; find a way to sort polygons by depth (z), then draw them in that order
  - do a rough sort of the polygons by the smallest (farthest) *z*-coordinate in each polygon
  - scan-convert the most distant polygon first, then work forward towards the viewpoint ("painters' algorithm")



- *Intersecting polygons* present a problem

Object-Precision

- Historically first approaches
  - Roberts ’63 - hidden line removal
    - compare each edge with every object - eliminate invisible edges or parts of edges.
  - A similar approach for hidden surfaces:
    - each polygon is clipped by the projections of all other polygons in front of it invisible surfaces are eliminated and visible sub-polygons are created SLOW, ugly special cases, polygons only

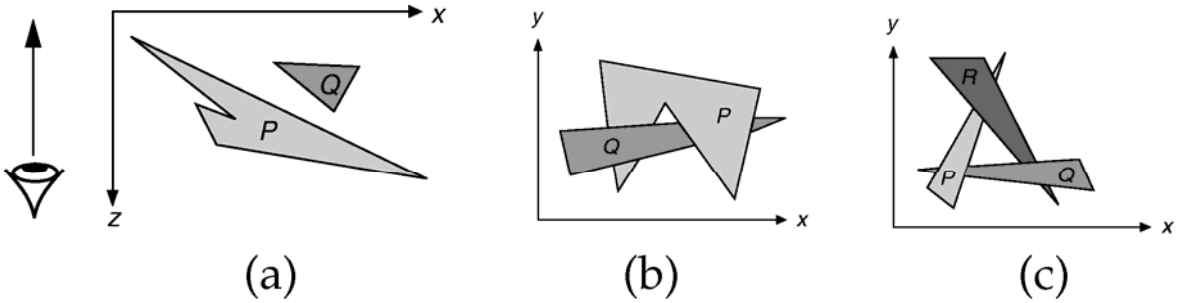


Polygon *A* is clipped by *B* which is in front of it. A new sub-polygon, *C*, is created.

3-D Depth-Sort Algorithm (1/2)

(Newell, Newell, and Sancha, based on work by Schumacher)

- Handles errors/ambiguities of Z-sort:



- Summary of algorithm
  1. Initially, sort by smallest Z
  2. **Resolve ambiguities:**
     - (a) Compare X extents
     - (b) Compare Y extents
     - (c) Is P entirely on one side of Q?
     - (d) Is Q entirely on one side of P?
     - (e) Compare X-Y projections (Polygon Intersection)
     - (f) **Swap or split polygons**
  3. Scan convert back to front

3-D Depth-Sort Algorithm (1/2)

Advantages

- Fast enough for simple scenes
- Fairly intuitive

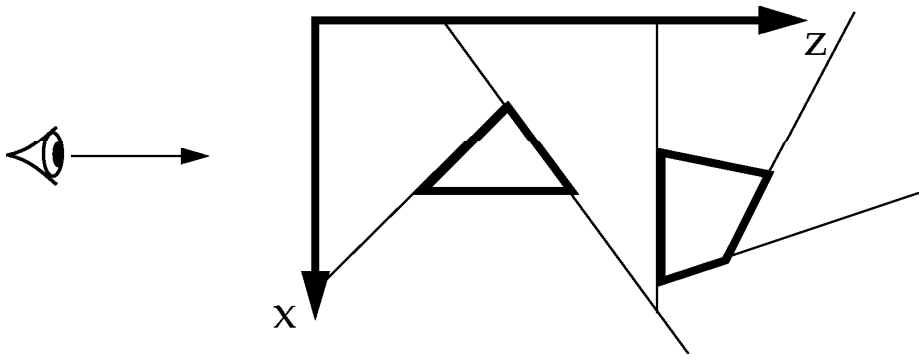
Disadvantages

- Slow for even moderately complex scenes
- Hard to implement and debug
- Lots of special cases

Binary Space Partitioning (1/4)

•(Fuchs, Kedem, and Naylor, based on work by Schumacher)

– Provides spatial subdivision and draw order

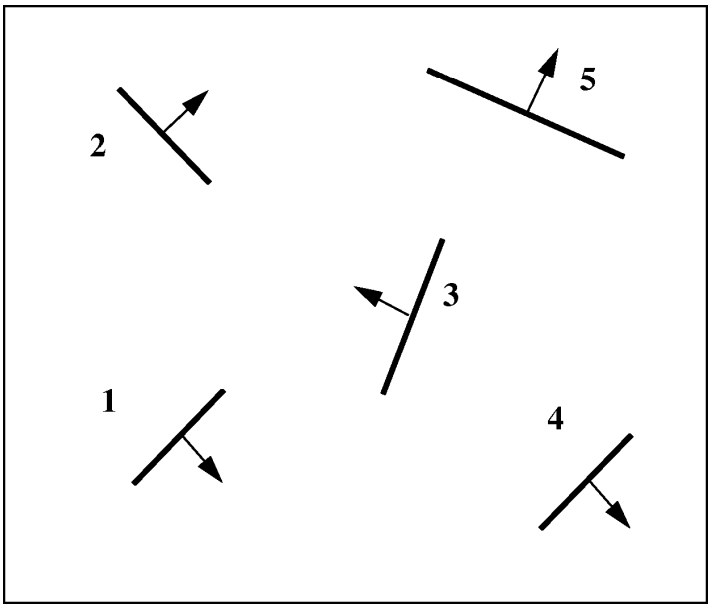


– Divide and conquer:
  - to display any polygon correctly, display all polygons on “far” (relative to viewpoint) side of polygon, then that polygon, then all polygons on polygon’s “near” side.
  - but how to display polygons on one side correctly? Choose one polygon and process it recursively!

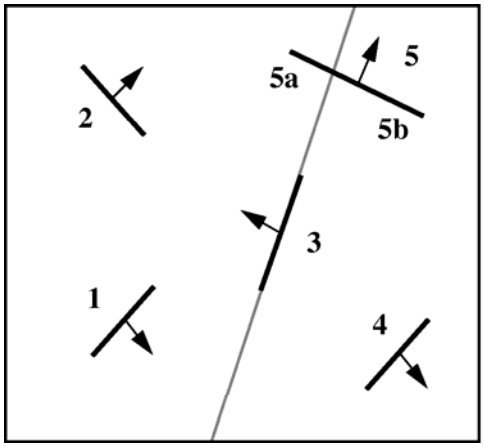
– Trades off view-independent preprocessing step (extra time and space) for low run-time overhead each time view changes

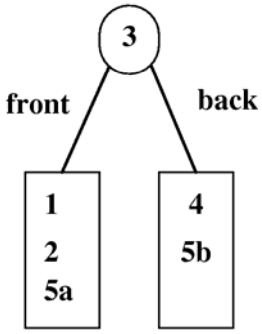
Binary Space Partitioning (2/4)

- Perform view-independent step once each time scene changes:
  - recursively subdivide environment into a hierarchy of half-spaces by dividing polygons in a half-space by the plane of a selected polygon
  - build a BSP tree representing this hierarchy
  - each selected polygon is the root of a sub-tree
- An example:

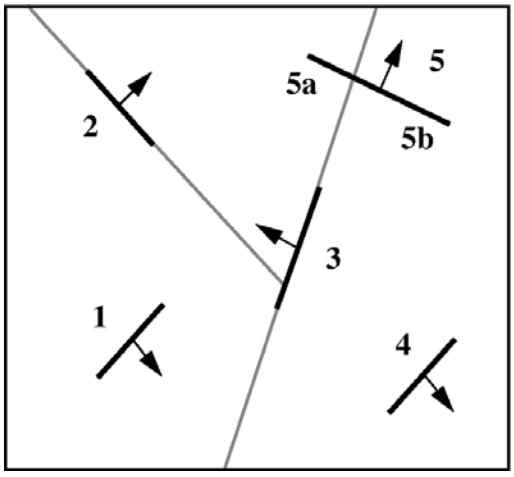


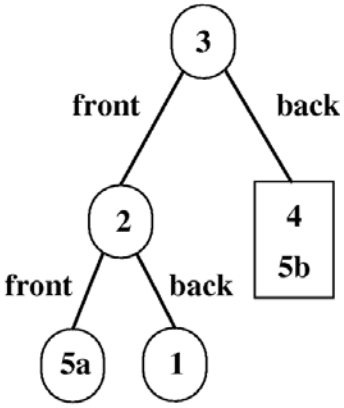
Binary Space Partitioning (3/4)





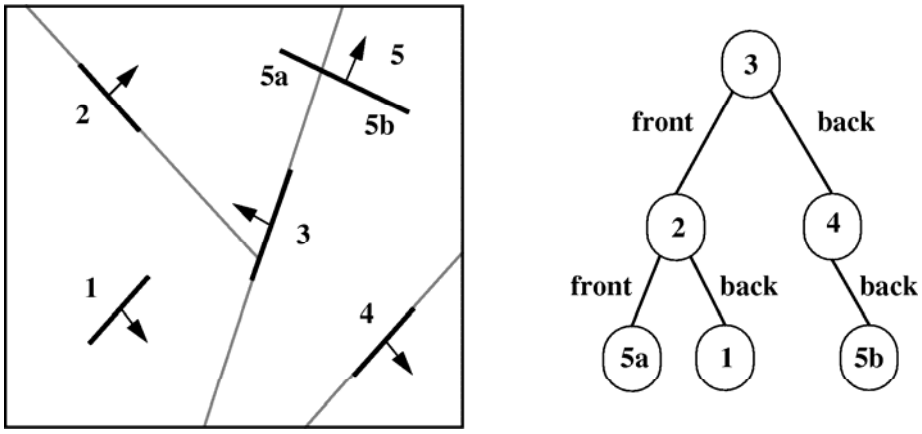
BSP-1: Choose any polygon (e.g., polygon 3) and subdivide Others by its plane, splitting polygons when necessary



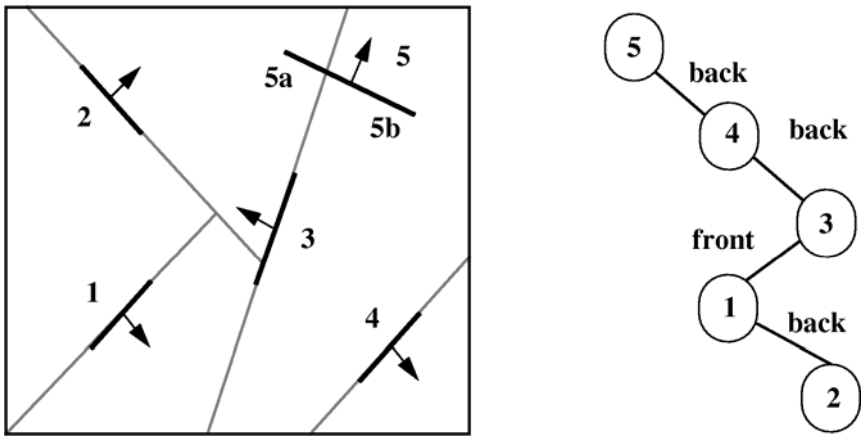


BSP-2: Process front sub-tree recursively

Binary Space Partitioning (4/4)



BSP-3: Process back sub-tree recursively



BSP-4: An alternative BSP tree with polygon 5 at the root

Referensi

- F.S.Hill, Jr., *COMPUTER GRAPHICS – Using Open GL,* Second Edition, Prentice Hall, 2001
- Andries van Dam, *Introduction to Computer Graphics,* Slide-Presentation, Brown University, 2003, (folder : brownUni)
- _____, *Interactive Computer Graphic,* Slide-Presentation, (folder : Lect_IC_AC_UK)
- _____, *CS 445/645 : Introduction to Computer Graphics,* Slide-Presentation, Virginia University (folder :COMP_GRAFIK)