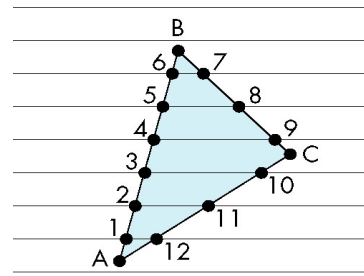
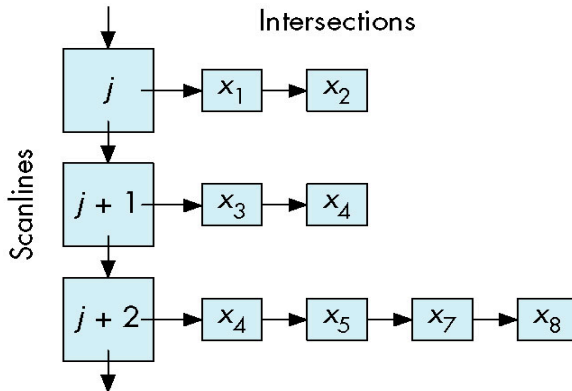
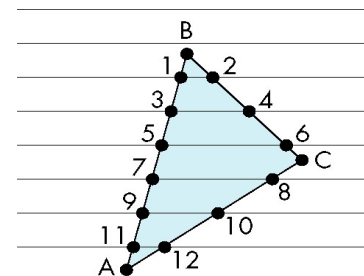


# Scan-Line Fill

- Can also fill by maintaining a data structure of all intersections of polygons with scan lines
  - Sort by scan line
  - Fill each span



vertex order generated by vertex list



desired order

# Scan-Line Algorithm

For each scan line:

1. Find the intersections of the scan line with all edges of the polygon.
2. Sort the intersections by increasing x-coordinate.
3. Fill in all pixels between pairs of intersections.

## Problem:

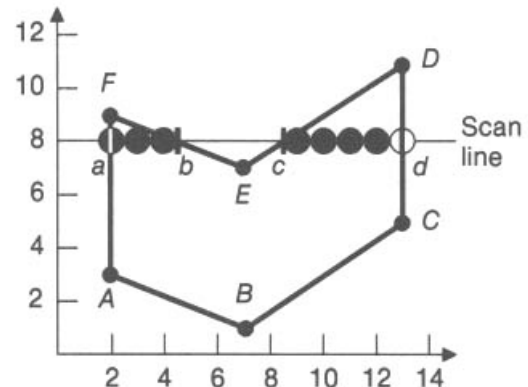
Calculating intersections is slow.

## Solution:

Incremental computation / coherence

For scan line number 8 the sorted list of x-coordinates is (2,4,9,13) (b and c are initially no integers)

Therefore fill pixels with x-coordinates 2-4 and 9-13.



# Edge Coherence

- Observation: Not all edges intersect each scanline.
- Many edges intersected by scanline  $i$  will also be intersected by scanline  $i+1$
- Formula for scanline  $s$  is  $y = s$ , for an edge is  $y = mx + b$
- Their intersection is

$$s = mx_s + b \rightarrow x_s = (s-b)/m$$

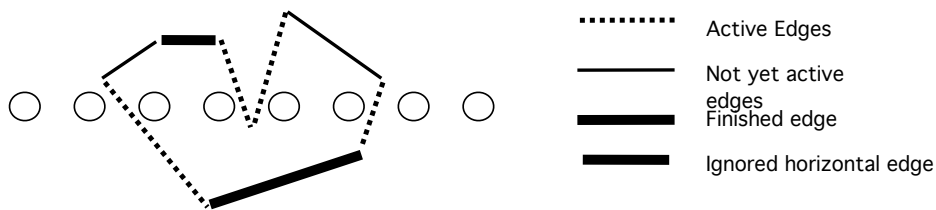
- For scanline  $s + 1$ ,

$$x_{s+1} = (s+1 - b)/m = x_s + 1/m$$

$$\text{Incremental calculation: } x_{s+1} = x_s + 1/m$$

# Processing Polygons

- Polygon edges are sorted according to their minimum / maximum Y.
- Scan lines are processed in increasing (upward) / decreasing (downward) Y order.
- When the current scan line reaches the lower / upper endpoint of an edge it becomes active.
- When the current scan line moves above the upper / below the lower endpoint, the edge becomes inactive.



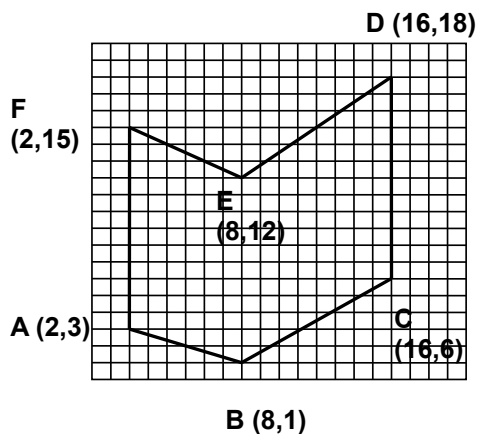
- Active edges are sorted according to increasing X. Filling the scan line starts at the leftmost edge intersection and stops at the second. It restarts at the third intersection and stops at the fourth. . . (spans)

# Polygon fill rules (to ensure consistency)

1. Horizontal edges: Do not include in edge table
2. Horizontal edges: Drawn either on the bottom or on the top.
3. Vertices: If local max or min, then count twice, else count once.
4. Either vertices at local minima or at local maxima are drawn.
5. Only turn on pixels whose centers are *interior* to the polygon:  
round up values on the left edge of a span, round down on the right edge

# Polygon fill example

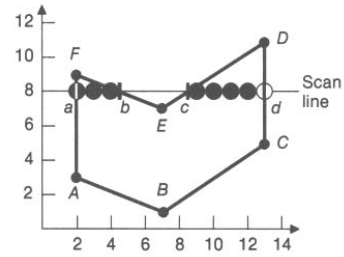
- The edge table (ET) with edges entries sorted in increasing y and x of the lower end.
  - $y_{max}$ : max y-coordinate of edge
  - $x_{min}$ : x-coordinate of lowest edge point
  - $1/m$ : x-increment used for stepping from one scan line to the next



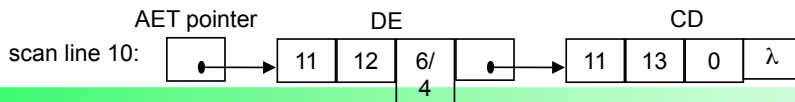
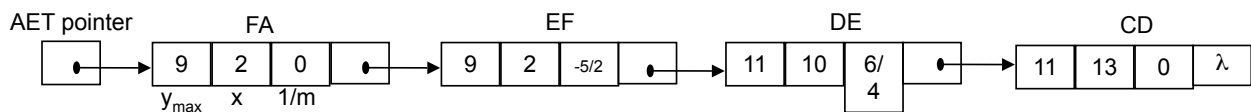
		$y_{max}$	$x_{min}$	$1/m$		$y_{max}$	$x_{min}$	$1/m$	
19	NULL								
18	NULL								
17	NULL								
16	NULL								
15	NULL								
14	NULL								
13	NULL								
12	->	15	8	-2	->	18	8	4/3	NULL
11	NULL								
10	NULL								
9	NULL								
8	NULL								
7	NULL								
6	->	18	16	0	NULL				
5	NULL								
4	NULL								
3	->	15	2	0	NULL				
2	NULL								
1	->	3	8	-3	->	6	8	8/5	NULL

# Processing steps

1. Set  $y$  to smallest  $y$  with entry in ET, i.e.,  $y$  for the first non-empty bucket
2. Init Active Edge Table (AET) to be empty
3. Repeat until AET and ET are empty:
  1. Move from ET bucket  $y$  to the AET those edges whose  $y_{\min} = y$  (entering edges)
  2. Remove from AET those edges for which  $y = y_{\max}$  (not involved in next scan line), then sort AET (remember: ET is presorted)
  3. Fill desired pixel values on scan line  $y$  by using pairs of  $x$ -coords from AET
  4. Increment  $y$  by 1 (next scan line)
  5. For each nonvertical edge remaining in AET, update  $x$  for new  $y$



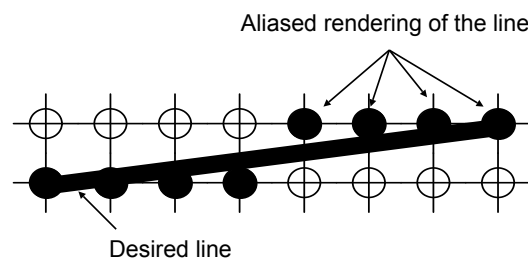
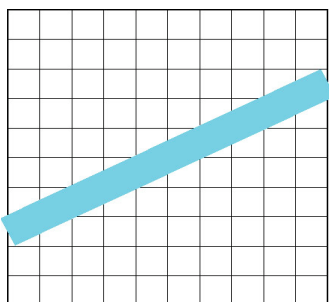
scan line 9:



Realtime 3D Computer Graphics / Virtual Reality – WS 2006/2007 – Marc Erich Latoschik

# Aliasing

- Aliasing is caused by finite addressability of the display.
- Approximation of lines and circles with discrete points often gives a staircase appearance or "Jaggies".
- Ideal rasterized line should be 1 pixel wide

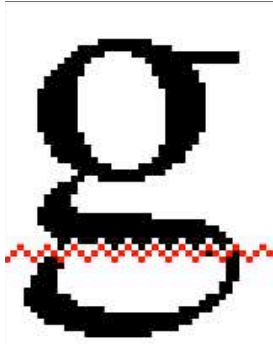


- Choosing best  $y$  for each  $x$  (or visa versa) produces aliased raster lines

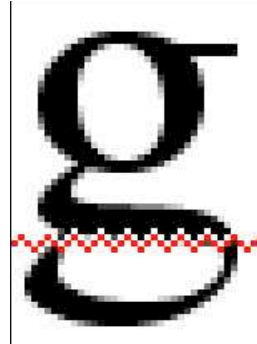
Realtime 3D Computer Graphics / Virtual Reality – WS 2006/2007 – Marc Erich Latoschik

## Aliasing / Antialiasing Examples

"Jaggies"

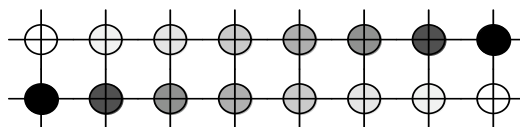


"Jaggies"



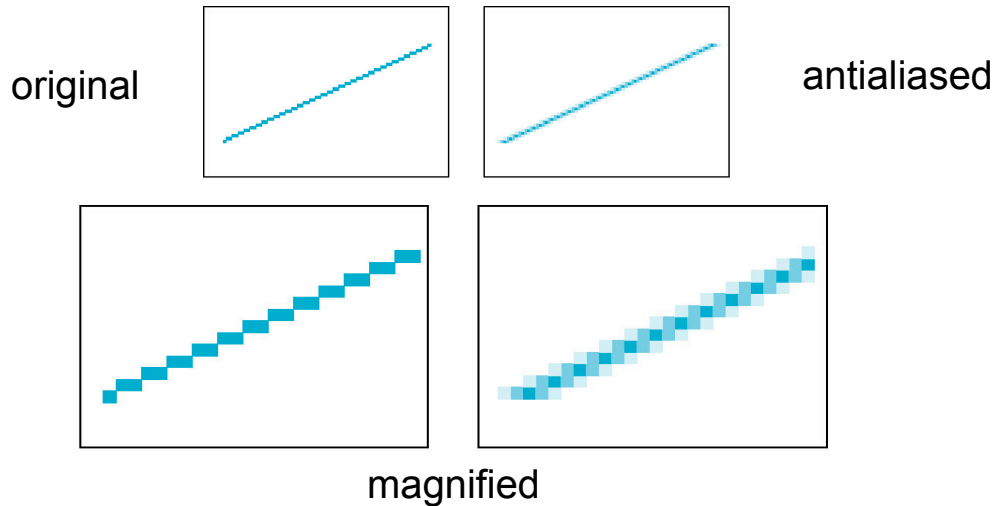
## Antialiasing - solutions

- Aliasing can be smoothed out by using higher addressability.
- If addressability is fixed but intensity is variable, use the intensity to control the address of a "virtual pixel".
  - Two adjacent pixels can be used to give the impression of a point part way between them.
  - The perceived location of the point is dependent upon the ratio of the intensities used at each.
  - The impression of a pixel located halfway between two addressable points can be given by having two adjacent pixels at half intensity.
- An antialiased line has a series of virtual pixels each located at the proper address.



# Antialiasing by Area Averaging

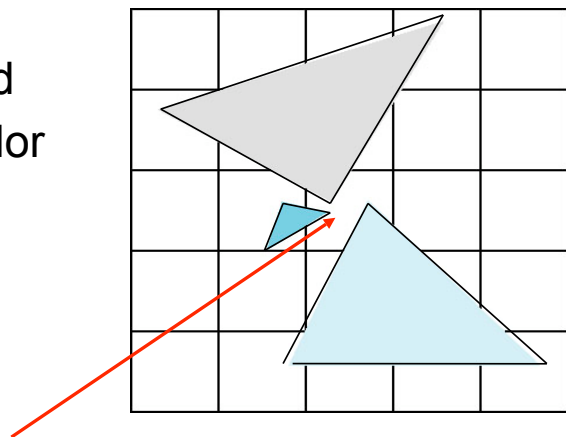
- Color multiple pixels for each x depending on coverage by ideal line



Realtime 3D Computer Graphics / Virtual Reality – WS 2006/2007 – Marc Erich Latoschik

# Polygon Aliasing

- Aliasing problems can be serious for polygons
  - Jaggedness of edges
  - Small polygons neglected
  - Need compositing, so color of one polygon does not totally determine color of pixel

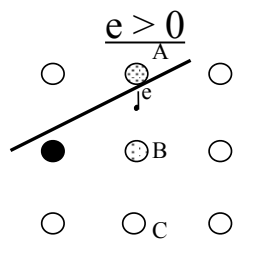


All three polygons should contribute to color

Realtime 3D Computer Graphics / Virtual Reality – WS 2006/2007 – Marc Erich Latoschik

# Antialiased Bresenham Lines

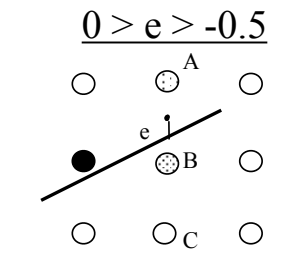
- Line drawing algorithms such as Bresenham's can easily be modified to implement virtual pixels. We use the distance ( $e = d/a$ ) value to determine pixel intensities.
- Three possible cases which occur during the Bresenham algorithm:



$$A = 0.5 + e$$

$$B = 1 - \text{abs}(e+0.5)$$

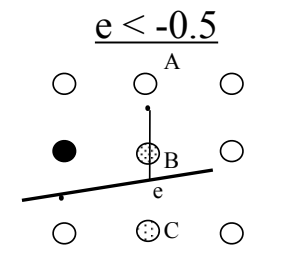
$$C = 0$$



$$A = 0.5 + e$$

$$B = 1 - \text{abs}(e+0.5)$$

$$C = 0$$



$$A = 0$$

$$B = 1 - \text{abs}(e+0.5)$$

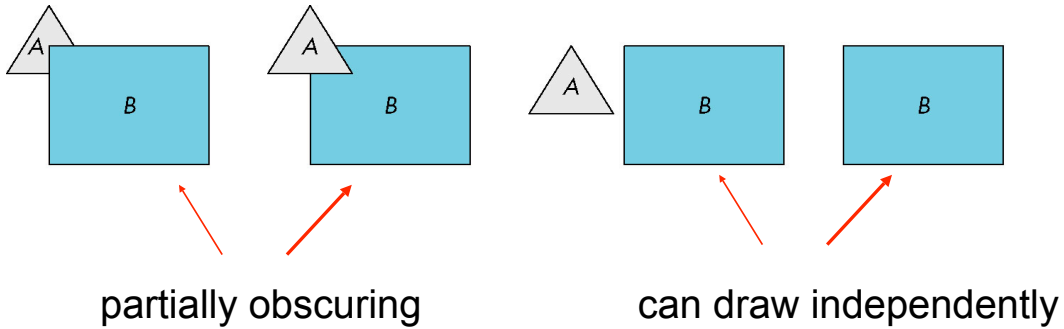
$$C = -0.5 - e$$

# Clipping and Visibility

- Clipping has much in common with hidden-surface removal
- In both cases, we are trying to remove objects that are not visible to the camera
- Often we can use visibility or occlusion testing early in the process to eliminate as many polygons as possible before going through the entire pipeline

## Hidden Surface Removal

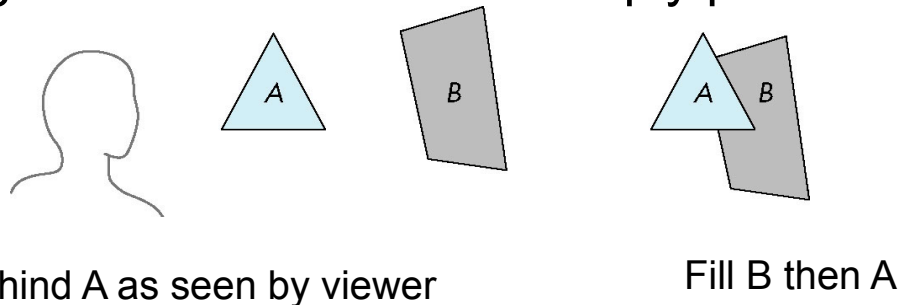
- Object-space approach: use pairwise testing between polygons (objects)



- Worst case complexity  $O(n^2)$  for  $n$  polygons

## Painter's Algorithm

- Render polygons a back to front order so that polygons behind others are simply painted over



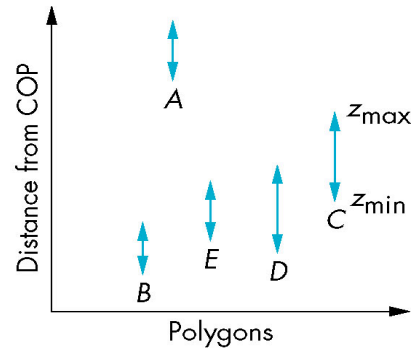


# Depth Sort

- Requires ordering of polygons first
  - $O(n \log n)$  calculation for ordering
  - Not every polygon is either in front or behind all other polygons

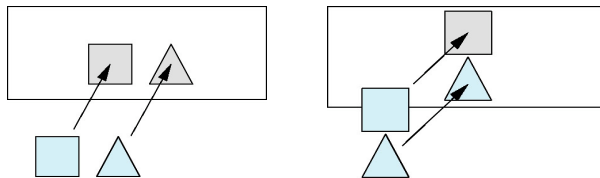
- Order polygons and deal with easy cases first, harder later

Polygons sorted by distance from COP



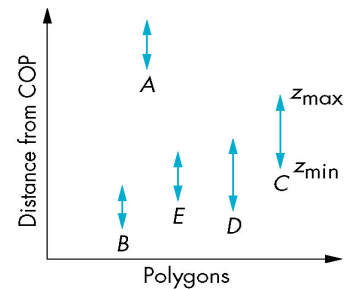
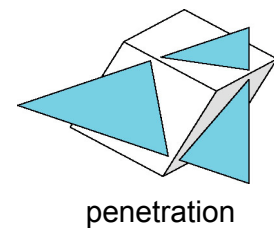
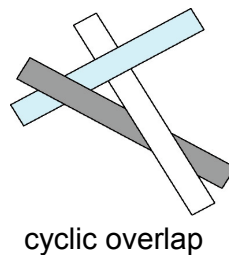
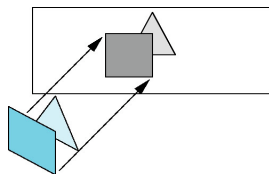
# Depth sort cases

- Easy cases:
  - Lies behind all other polygons (can render):
  - Polygons overlap in z but not in either x or y (can render independently):



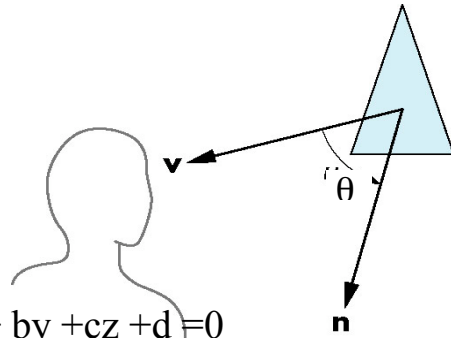
- Hard cases:

Overlap in all directions but one is fully on one side of the other



## Back-Face Removal (Culling)

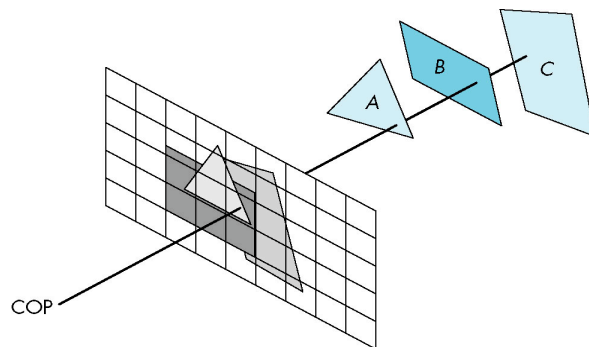
- face is visible iff  $90 \geq \theta \geq -90$   
equivalently  $\cos \theta \geq 0$   
or  $\mathbf{v} \cdot \mathbf{n} \geq 0$



- plane of face has form  $ax + by + cz + d = 0$   
but after normalization  $\mathbf{n} = (0 \ 0 \ 1 \ 0)^T$
- need only test the sign of  $c$
- In OpenGL we can simply enable culling  
but may not work correctly if we have nonconvex objects

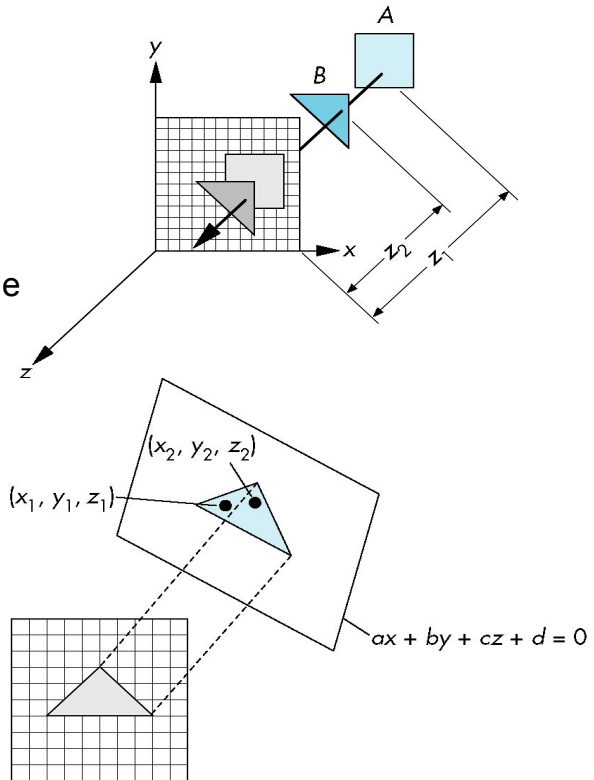
## Image Space Approach

- Look at each projector ( $nm$  for an  $n \times m$  frame buffer) and find closest of  $k$  polygons
- Complexity  $O(nmk)$
- Ray tracing
- z-buffer



# z-Buffer Algorithm

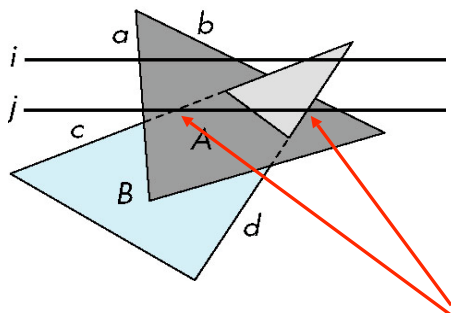
- Use a depth buffer called the z-buffer to store the depth of the closest object at each pixel found so far
- As we render each polygon, compare the depth of each pixel to depth in z buffer
- If less, place shade of pixel in color buffer and update z buffer
- Efficiency:
  - If we work scan line by scan line as we move across a scan line, the depth changes satisfy  $a\Delta x + b\Delta y + c\Delta z = 0$
  - Along scan line  $\Delta y = 0, \Delta z = -\frac{a}{c} \Delta x$
  - In screen space  $\Delta x = 1$



Realtime 3D Computer Graphics / Virtual Reality – WS 2006/2007 – Marc Erich Latoschik

# Scan-Line Algorithm

- Can combine shading and hsr through scan line algorithm



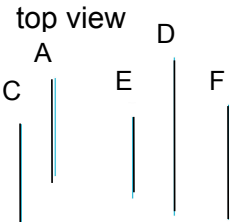
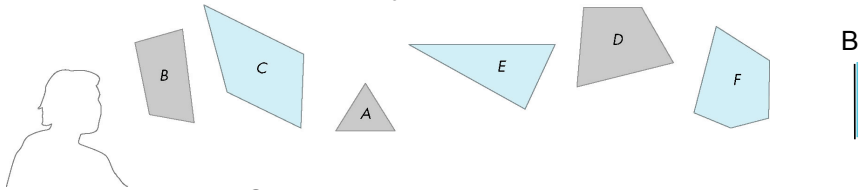
scan line i: no need for depth information, can only be in no or one polygon

scan line j: need depth information only when in more than one polygon

Realtime 3D Computer Graphics / Virtual Reality – WS 2006/2007 – Marc Erich Latoschik

# Visibility Testing

- In realtime applications, eliminate as many objects as possible within the application
  - Reduce burden on pipeline
  - Reduce traffic on bus
- Partition space with Binary Spatial Partition (BSP) Tree



- Easy example: Consider 6 parallel polygons. The plane of A separates B and C from D, E and F
  - Can continue recursively
    - Plane of C separates B from A
    - Plane of D separates E and F
  - Can put this information in a BSP tree
    - Use for visibility and occlusion testing

