Visible Surface Determination and Antialiasing

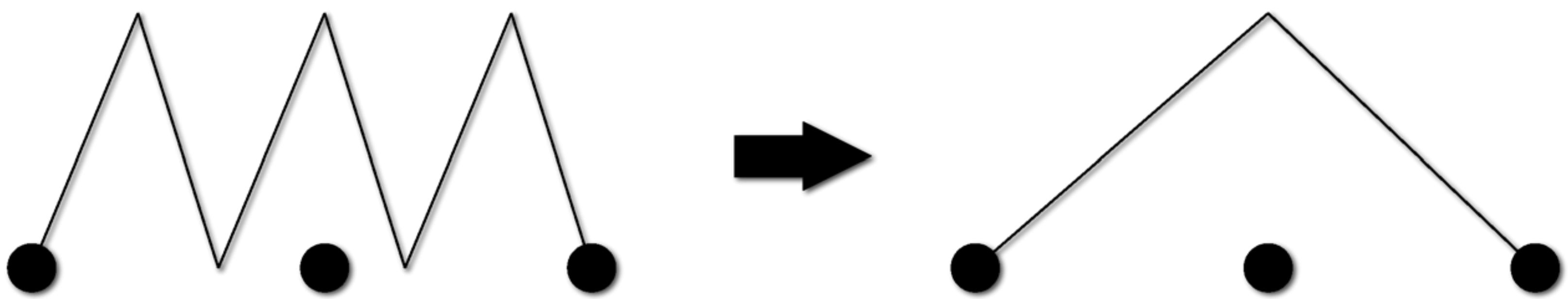
LESSON 9

Computer Graphics 1

Alias and Antialiasing

Alias

- Rasterization algorithms produce stairstep, jagged appearance
- Distortion due to low-frequency sampling
- To avoid alias specific frequency of sampling has to be achieved



example of insufficient sampling

Nyquist Frequency

- Minimal sampling frequency to avoid losing information
- Nyquist sampling frequency:

$$f_s = 2 f_{\max}$$

- $f_{\max}$ highest frequency occurring in the object
- Nyquist sampling interval:

$$\Delta x_s = 2 \frac{\Delta x_{cycle}}{2}$$

$$\Delta x_{cycle} = \frac{1}{f_{\max}}$$

Antialiasing

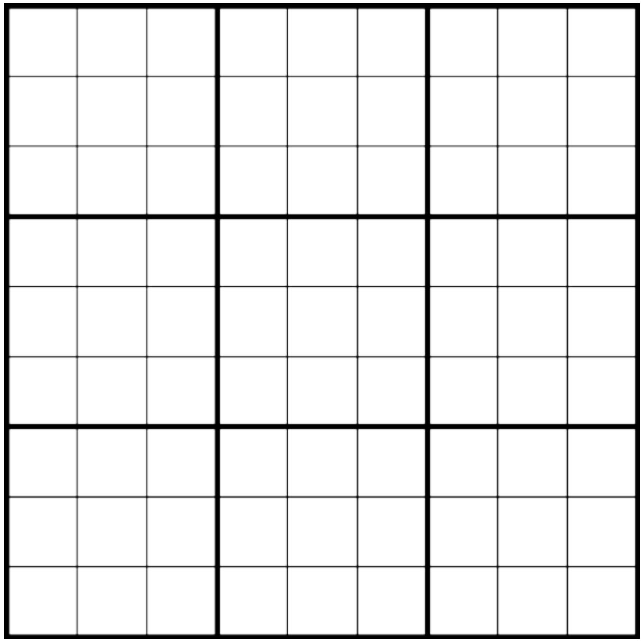
- Represent continuous object accurately needs arbitrary small sampling intervals
- We have limited resolution
- Solution: antialiasing
  - Modify pixel intensity along the boundary
  - More than two intensities needed

Antialiasing Techniques

- Postfiltering
  - Sampling at higher rate
- Prefiltering
  - Treats pixels as having area

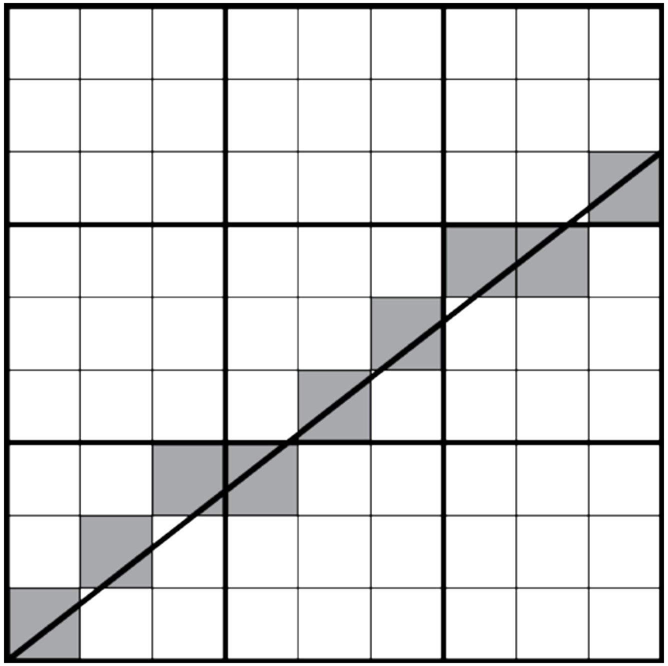
Postfiltering

- Sample at higher frequency
- Oversample the same amount in each direction
- Each pixel is divided into several subpixels
- Filtering: large resolution to small resolution



Postfiltering – Straight Line Segments

- Count of subpixels along the line
- 9 subpixels gives 3 intensity levels above zero
- Increasing resolution increases intensity number of levels



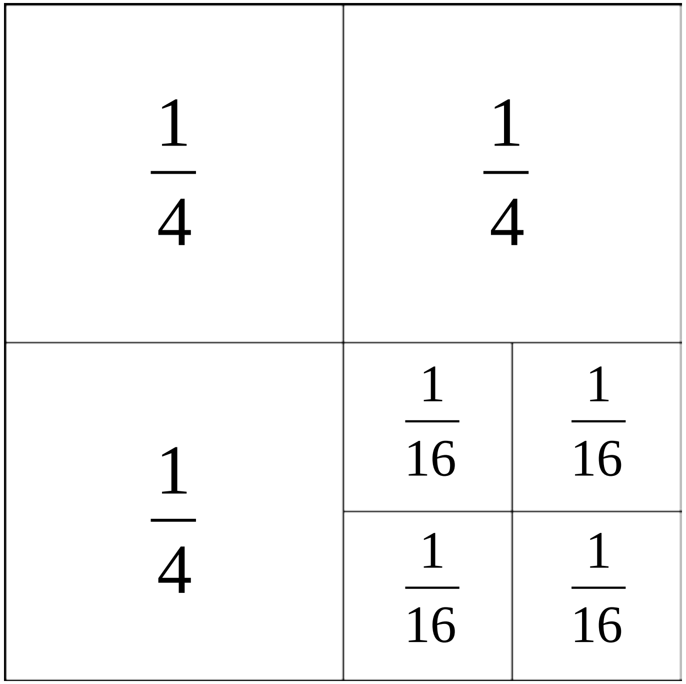
Postfiltering – Pixel Weighting Mask

- Subpixels have the same weight
  - Supersampling
- Subpixels have different weights

$$1/8\begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad 1/16\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

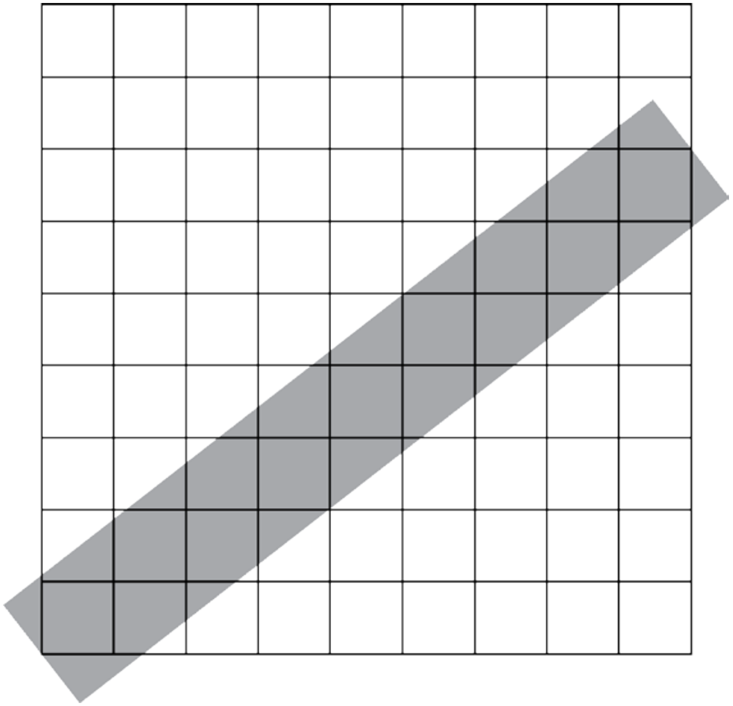
Nonuniform Postifiltering

- Increase sampling only in specific areas
  - Only where it is necessary
  - Where alias may occur
- Intensity
  - Weighted average
- Smaller subpixel has a smaller weight



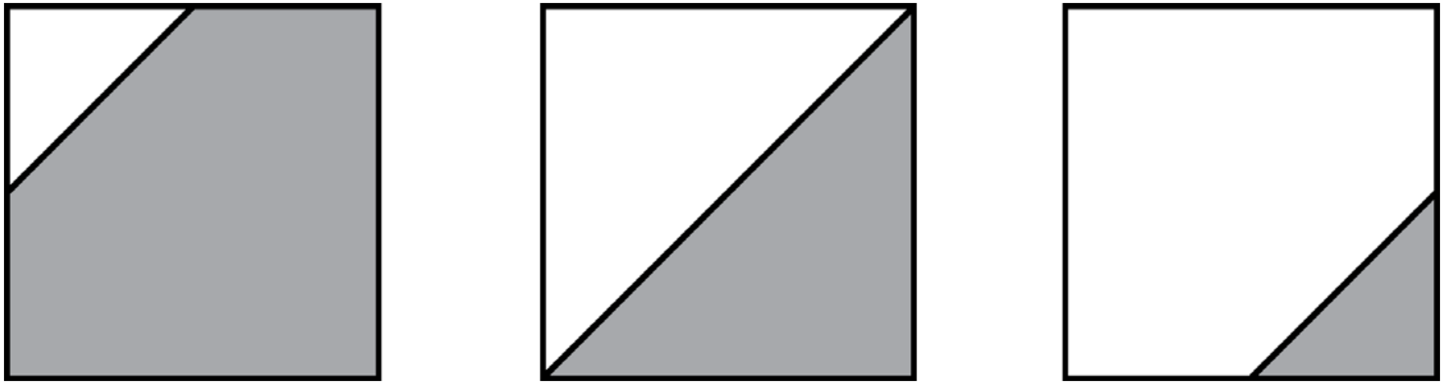
Prefiltering

- Treat pixels as having area
- Lines have finite width
- Compute intensity based on the covered area
- Multiple objects
  - Solve visibility
- Line representation
  - quadrilateral



Prefiltering Simplification

- Computing area is expensive
- Precompute only specific positions of object and pixels
  - Find the most suitable case from the precomputed cases

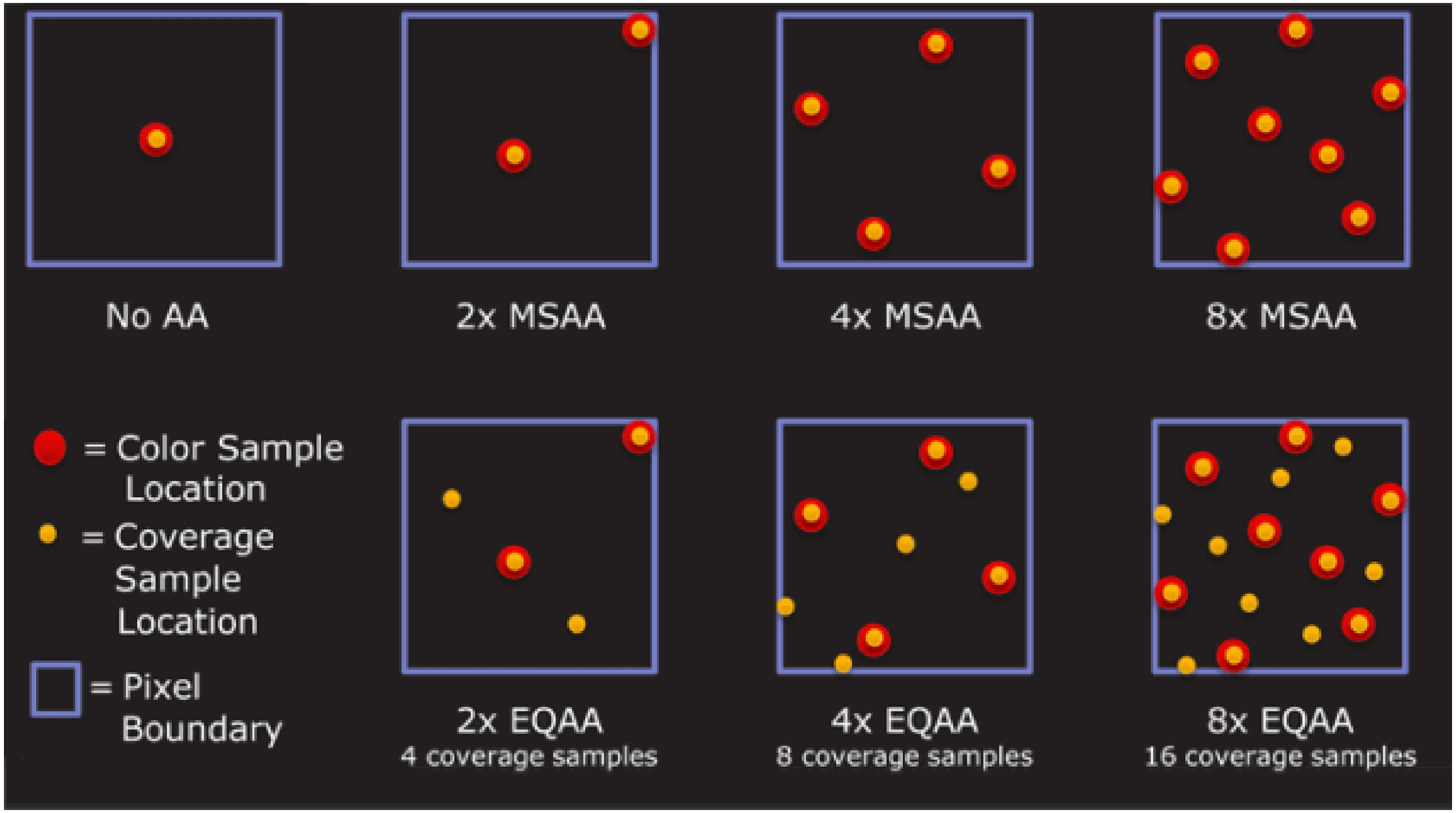


Multisampling (MSAA)

- Used in today graphic cards
- Each pixel is sampled multiple times (2-16)
- Each pass is slightly moved
  - Less than 1 pixel
- Final intensity is acquired by averaging all samples
- Compute only on edges

Coverage Sample Antialiasing (CSAA)

- Sample only coverage and do not sample color



Visible Surface Determination

Visible Surface Determination

- Determine surface patches that will be visible from a given viewpoint
- Hidden surface removal
  - Used in the past
- Three types of algorithms:
  - Object precision (space)
  - Image precision (space)
  - List priority

Object Precision Algorithms


```
for each object O do
   begin
        find the part A of O that is visible;
        display A;
   end
```

Image Precision Algorithms

for each pixel P on the screen **do**

begin

determine the visible object O pierced

by ray R;

(R- ray from the viewer through the pixel P)

if there is such O

then display the pixels in the color of O

else display the pixel in the background color

end

Comparison

- Object precision
  - Computes **all** visible parts
  - Problems with alias
  - Complexity is based on the **number of objects**
  - In early days of CG
- Image precision
  - Determines visibility in **sampled** number of directions
  - Complexity is based on the **resolution**
- List priority
  - Between image space and object space
  - Most of the algorithms

Painter’s Algorithm

- Faces of the scene are listed back to front
- face A is in front of face B = B will not obscure A in any way
- Draw the faces from back to front

- Modification
  - Draw from front to back
  - Store mask of drawn points
  - More efficient (less writing into frame-buffer)

Newel-Newel-Sancha

- Method for sorting faces
- Uses painter's algorithm to draw faces
- Sometimes referred as painters algorithm
- Computes ordering on the fly
- One of earliest list priority algorithms

Newel-Newel-Sancha

- Initial ordering based on the farthest z-coordinate
- Start with the last polygon P
- Find set of lines $Q=\{Q_1,..,Q_n\}$
  - (Minimum z-value of P) > (maximum z-value of Q_i)
  - No overlap in z direction
- If a line is not in Q it is correctly sorted with respect to P

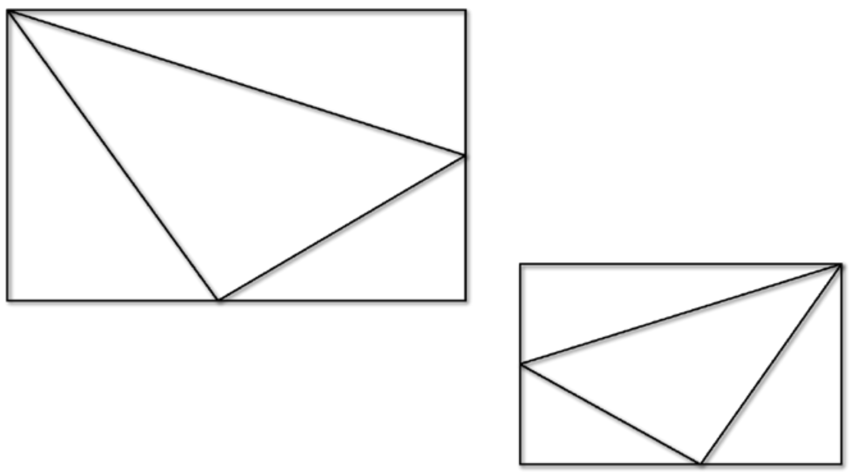
Newel-Newel-Sancha - Test

- Perform tests to sort P and lines in Q

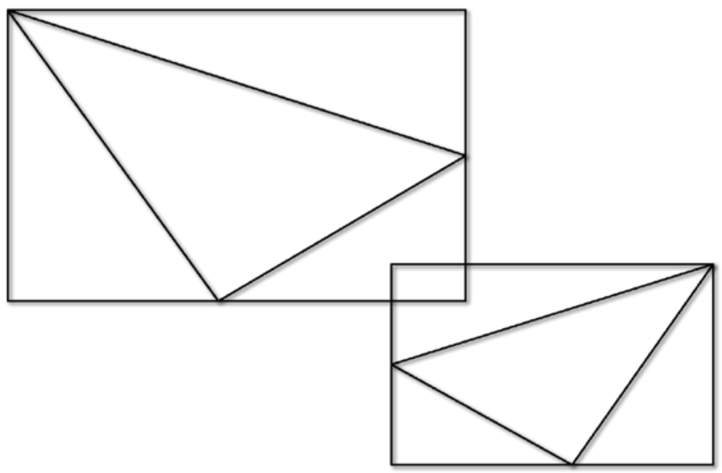
- 1. Can one separate P and Qs in x?
- 2. Can one separate P and Qs in y?
- 3. Is P on the farther side of Qs?
- 4. Are Qs on the near side of P?
- 5. Do P and Qs project to disjoint set?

Test 1,2

- Overlap of xy envelopes



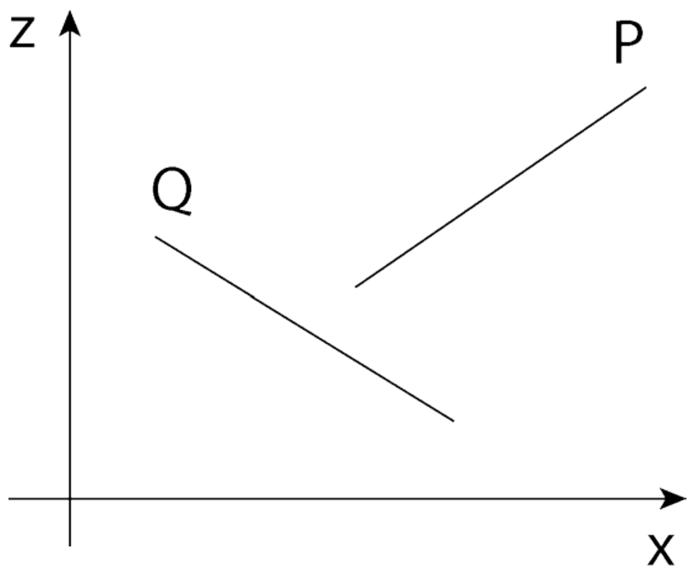
separable in y direction



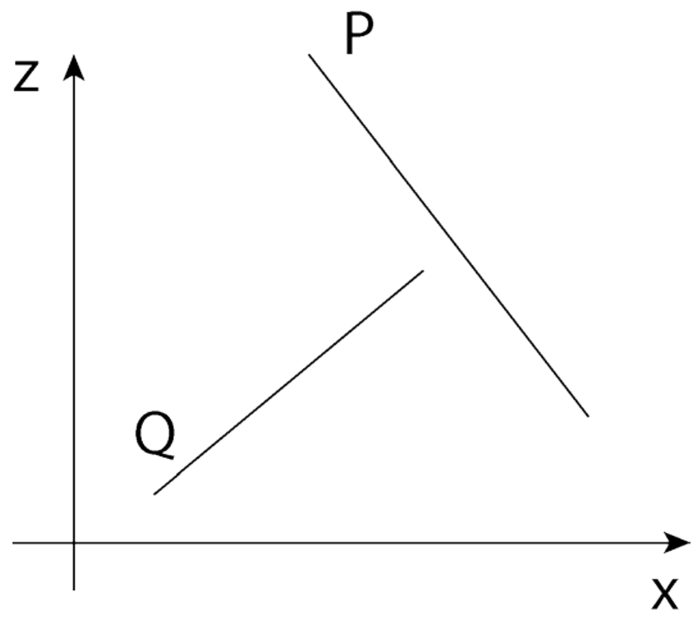
non-separable in x nor in y direction

Test 3 and 4

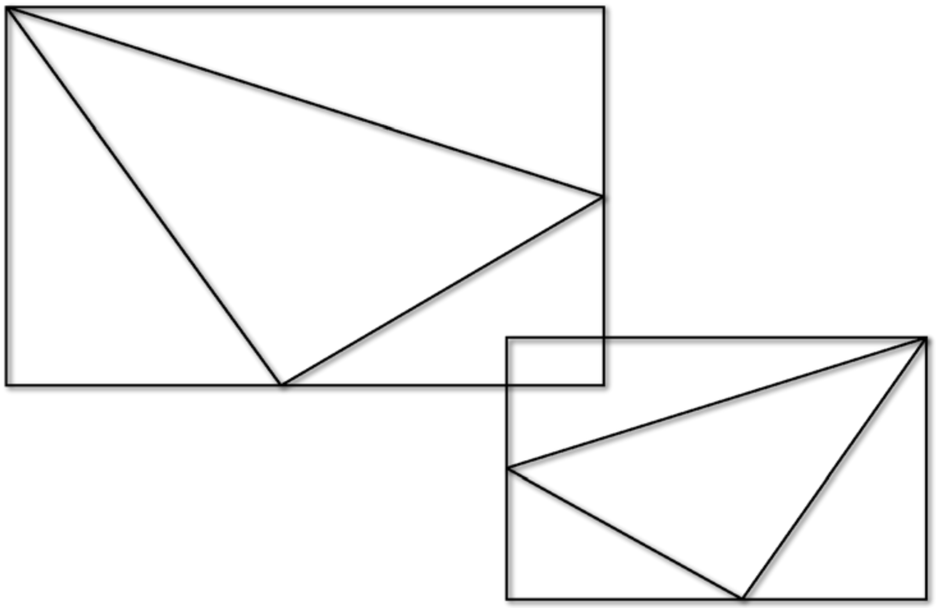
Test 3



Test 4



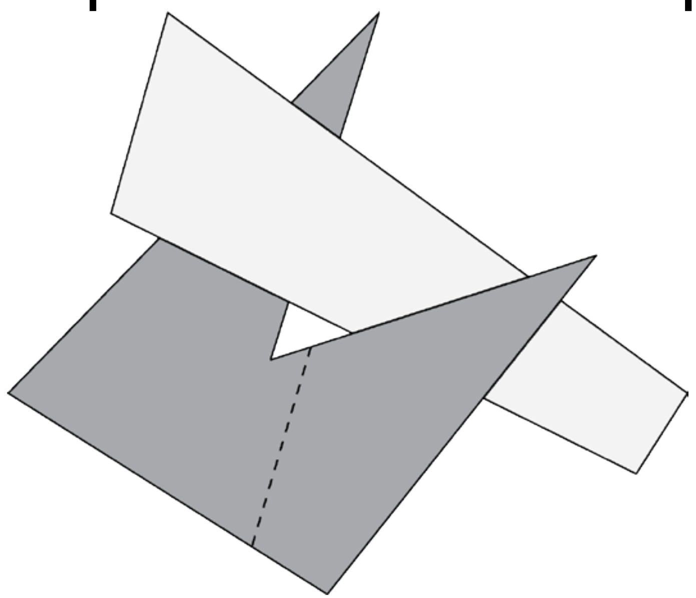
Test 5



overlapping xy envelopes but disjoint polygons

Newel-Newel-Sancha - Test

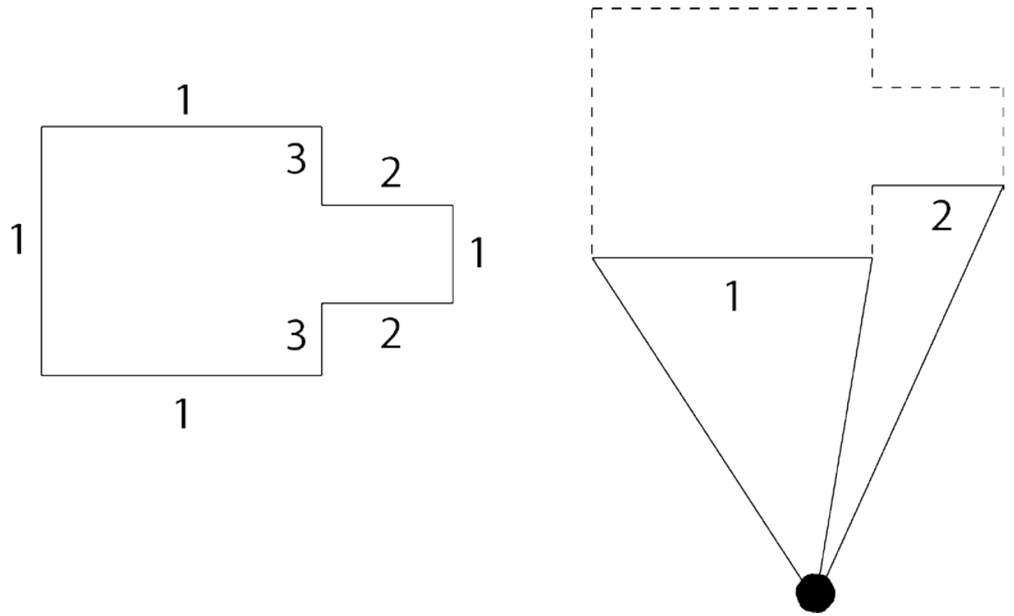
- If P and Qs do not pass all tests swap P with one of Q
- If cyclical overlap occurs cut one of polygons



- Tests are ordered from simple to complicated
- Simple are performed more often

Schumacker List Priority Algorithm

- Make the back to front sort view independent
- Polygon with lower priority will never obscure polygon with higher priority
- Dividing scene into prioritizable clusters



BSP Algorithm

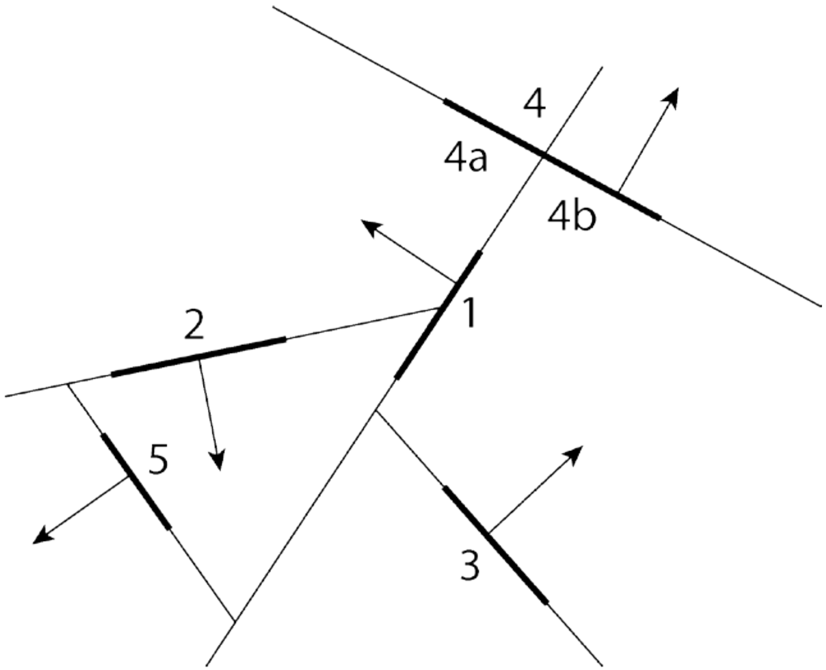
- Exploit idea of separating plane
- No polygon on the viewpoint side of the plane can be obstructed by a polygon from the other side

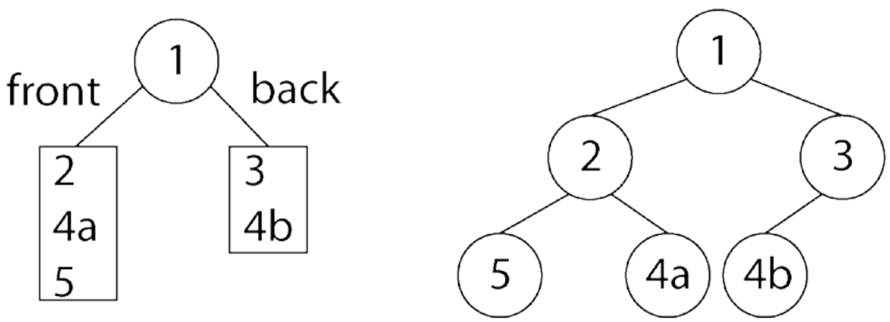
- Two parts
  - Converting polygon list into BSP tree
  - Traversal algorithm for back to front ordering of polygons

Building BSP Tree

- Select any polygon and place it at the root
- Test each remaining polygon
  - Lies on the same side as viewpoint – insert in the left (front) subtree
  - Lies on the opposite side as viewpoint – insert in the right (back) subtree
  - Lies on both sides – divide the polygon along the plane and put each part in he appropriate tree
- Repeat the procedure recursively for the two subtrees

BSP Tree – Example





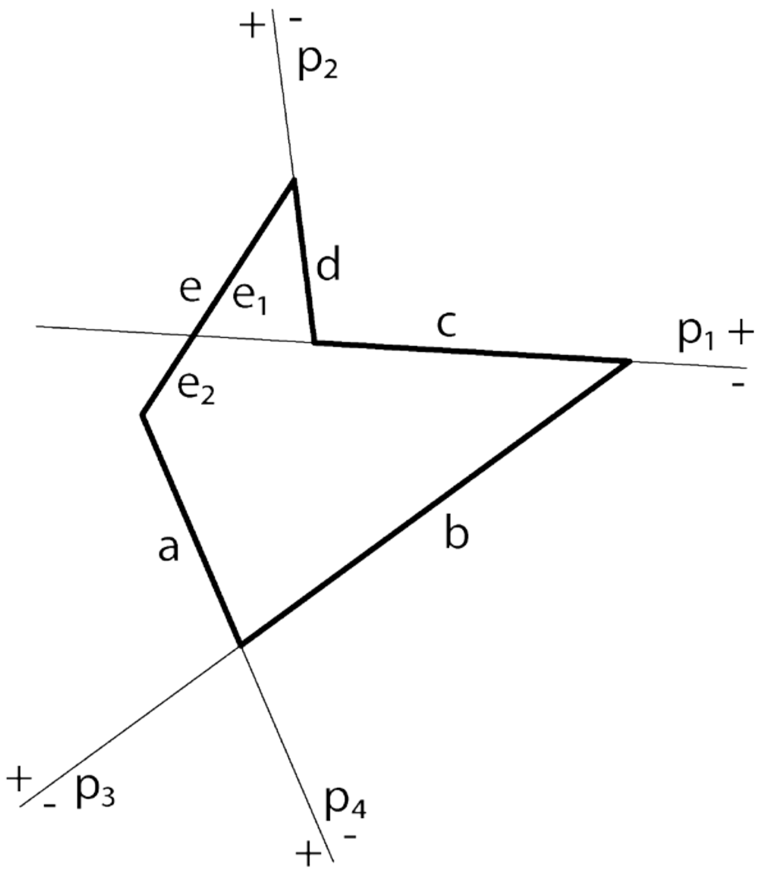
Avoiding Large Trees

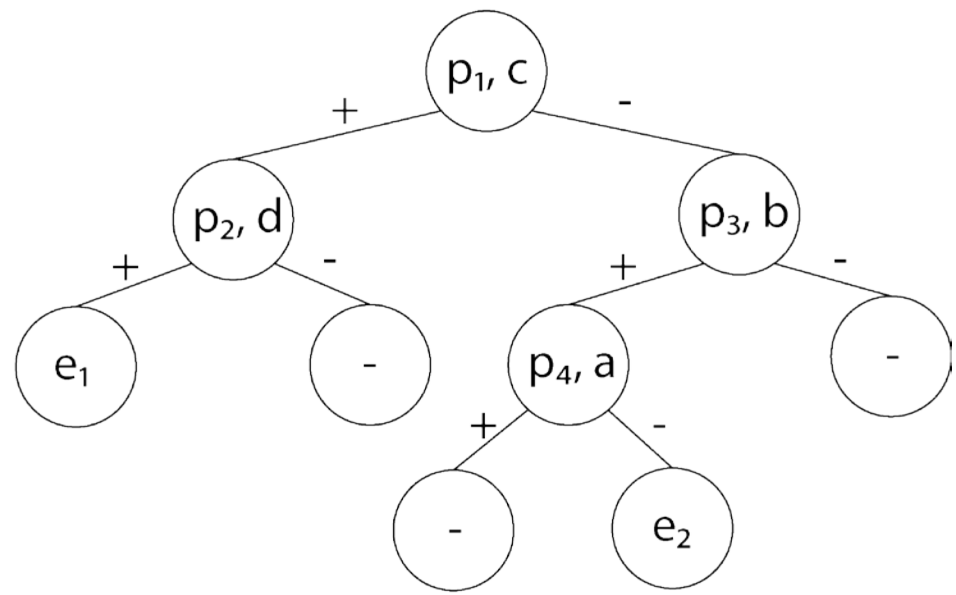
- Use heuristics
- Select the polygon in the root of the subtree
  - Cuts the fewest polygons
- Choose the best from a few chosen at random

BSP Tree Traversal

- If the viewpoint is in the front subtree
  - 1. draw back subtree
  - 2. draw front subtree
- If the viewpoint is in the back subtree
  - 1. draw front subtree
  - 2. draw back subtree

Representing Polygons



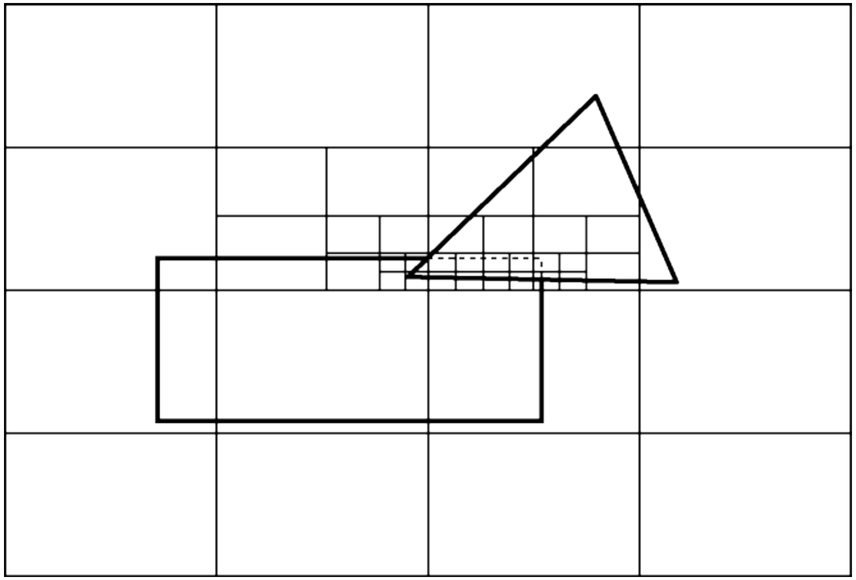


BSP Tree – Conclusion

- Slow building and fast traversal
- Ordering is independent from the viewpoint
  - Possible precomputation
- Efficient for static scenes
- Use
  - Flight simulators
  - Computer games (e.g. Doom)

Warnock's Algorithm

- Image space algorithm
- Find rectangular regions (windows) of the same intensity
- Recursively subdivide window until it has the same intensity



Warnock's Algorithm

- Initialize list of windows L by adding the entire screen
- For each window W in L look for the following trivial cases:
  - 1. all polygons are disjoint from W – draw W in the background color
  - 2. one surrounding polygon in front of all other polygons intersecting the window is found – draw W in the color of the polygon
  - 3. only one polygon Intersects W

Warnock's Algorithm

- 3. only one polygon intersects W
  - Draw intersection in the color of P and the rest in color of W
  - 3 subcases: P is contained in W, P surrounds W, P and W have nontrivial intersection
- None of the 3 cases occurred
  - Divide the window into 4 equal windows and add them to the list L
  - Repeat until windows get to the size of a pixel
  - At that point check which polygon is in front of the others

Warnock's Algorithm – Tests

- Is P disjoint from the window?
- Does P surround a window?
- Does P partially meet a window?
- Does P fall inside a window?
- Is P in front of other polygons?

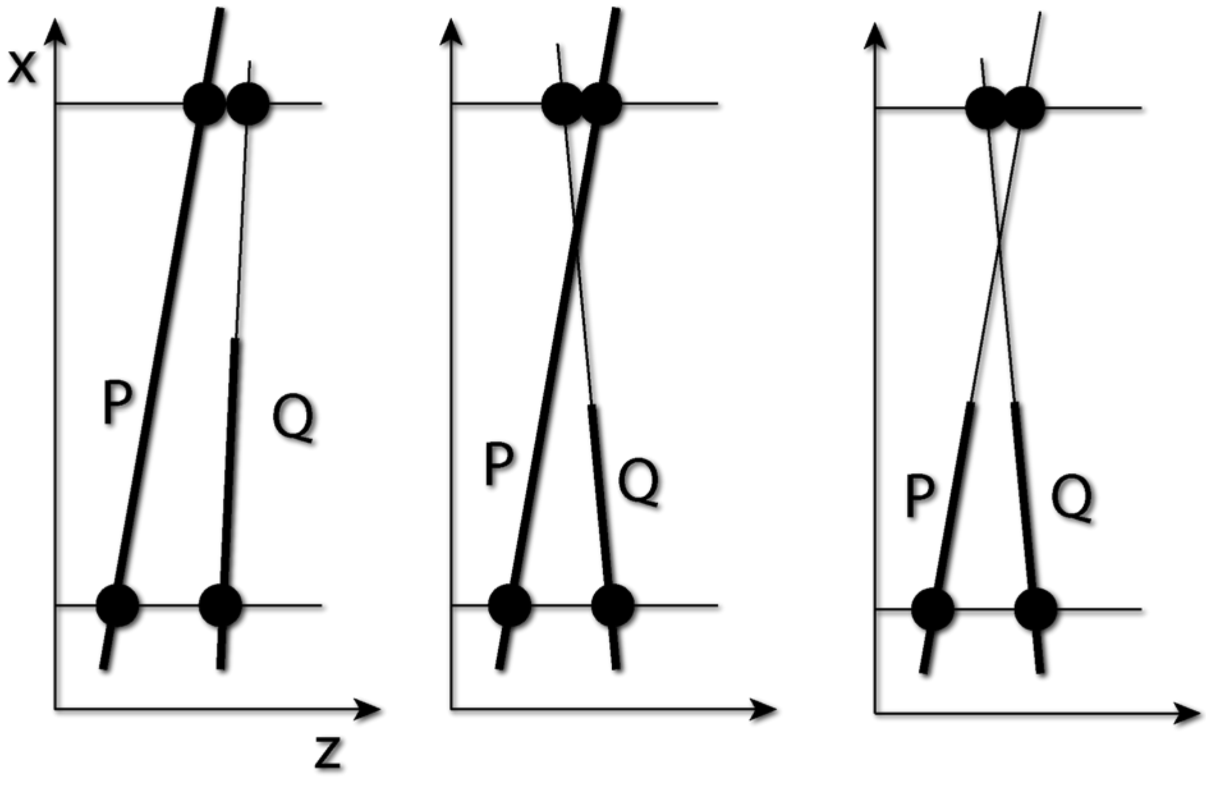
Warnock's Algorithm – Tests

- Is P disjoint from the window?
  - Bounding box
- Does P surround a window?
  - Check if window vertices are inside P
  - If not check if P is surrounded by W (partialy meet aor fall inside)
- Is P in front of other polygons?
  - Involves depth caculations

Is P in Front of Other Polygons?

- P and Q partially meet window W
- Test whether P is in front of Q
  - Only if P is surrounding polygon
- The depth of the plane of P is less than the depth of the plane of Q in all corners of the window
  - Sufficient but not necessary condition
  - Subdivide if the test fails

Face in Front of other Faces



Z-buffer Algorithm

- Image based algorithm
- Record depth information for each pixel
- Z-buffer
  - Two dimensional array of the same size as frame buffer
  - Store depth as real values
- Scan convert in frame-buffer and in Z-buffer

Z-buffer Algorithm

initialize FRAMEBUFFER to the background color

Initialize DEPTH to ∞

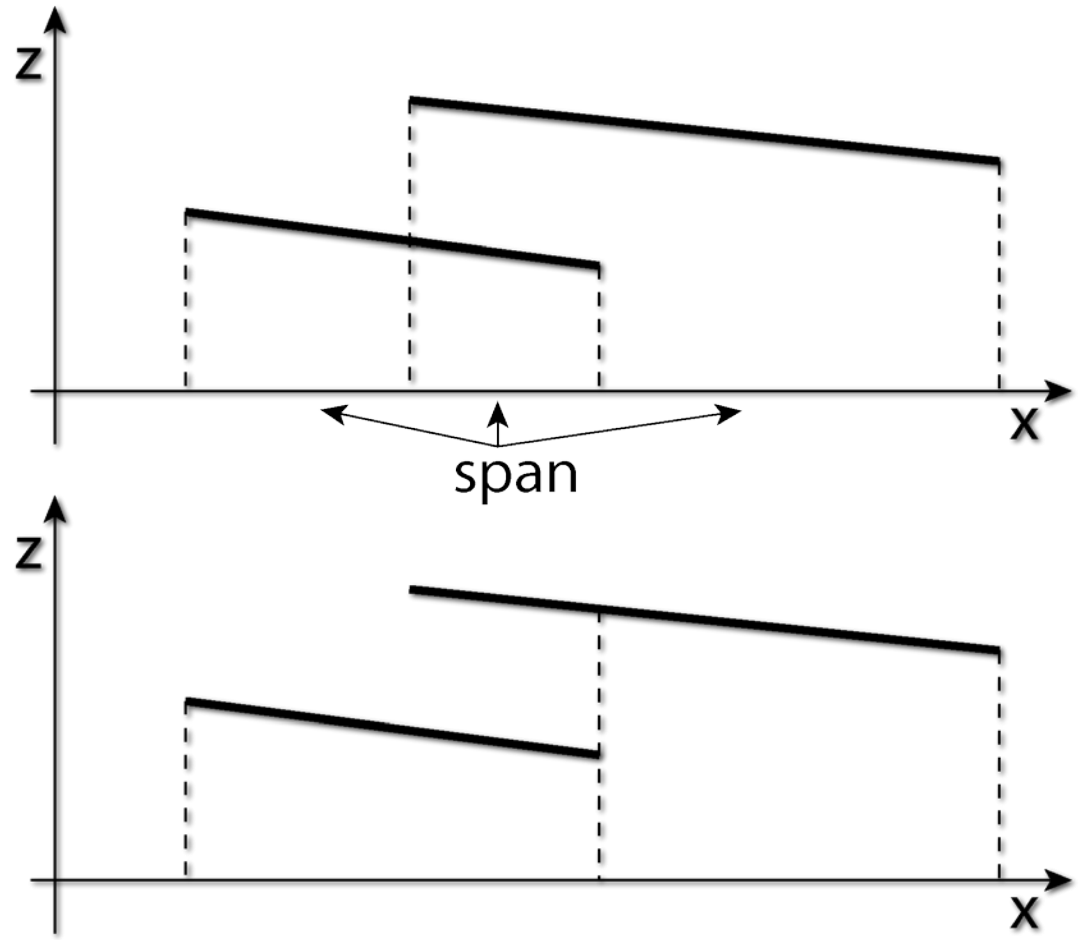
```
for each face F do
    for each point p of F do
        if p project to FRAMEBUFFER[i,j] then
            if Depth(p) < DEPTH[i,j] then
            begin
                FRAMEBUFFER[i,j] = color of F at p
                DEPTH[i,j] = Depth(p)
            end
```

Scan Line Approach to Z-buffer

- Z-buffer takes a lot of memory
- Compute if line separately
- Use array as long as scan line
- 1. y sort to limit attention to the edges or faces intersecting the scan line
  - Use AEL and coherence of edges as by scan conversion
- 2. x sort
  - Divide scan line into spans
- 3. Z depth search
  - Process each span of the scan line

Divide Scan Line into Spans

- Segments can be unambiguously ordered within a span
- Different approaches



Z-buffer Conclusion

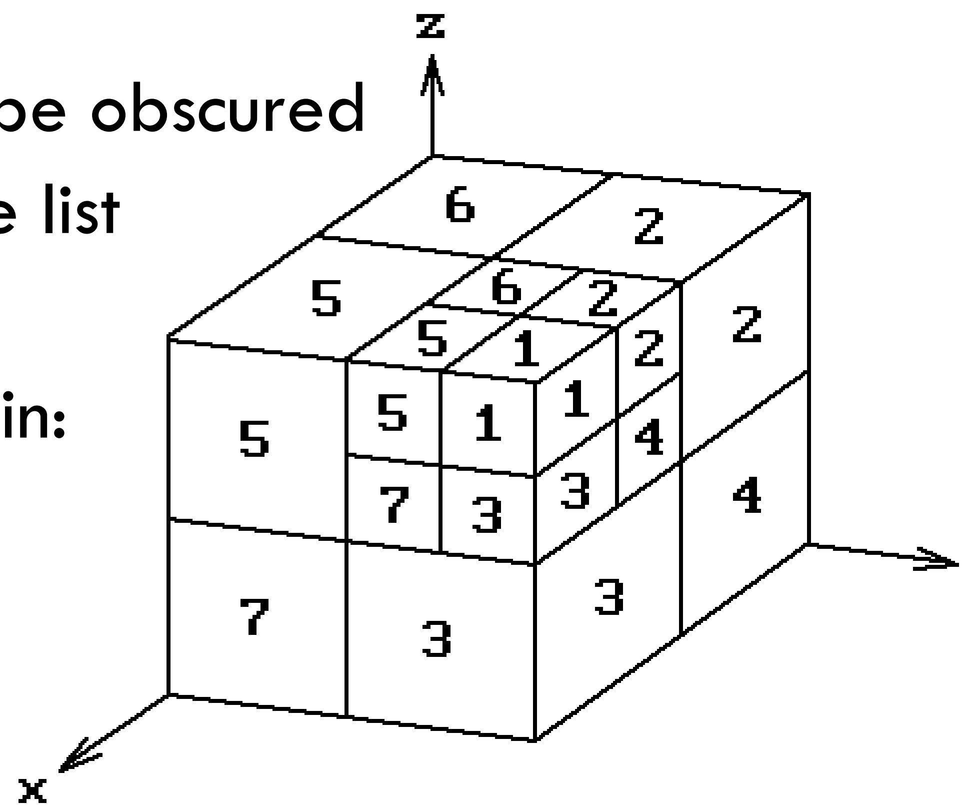
- Advantages
  - Simple algorithm
  - Easy to implement
- Disadvantages
  - Memory consuming
- Suitable for scenes with many polygons
  - Used in today graphic cards

Z-buffer Comparison

- Z-buffer
  - For each polygon
    - For each y
      - For each x
- Z-buffer with scan line
  - For each y
    - For each polygon
      - For each x
- Ray casting
  - For each y
    - For each x
      - For each polygon

Octree Algorithm

- Visualization of volume data
- Draw octants in sequence dependent on the viewing direction
- No voxel in the list will be obscured by a voxel earlier in the list
- Viewer in the 1st octant looking toward the origin: 8, 7, 4, 6, 5, 2, 3, 1 (multiple possibilities)



50

Questions ???