Clipping, Rasterization

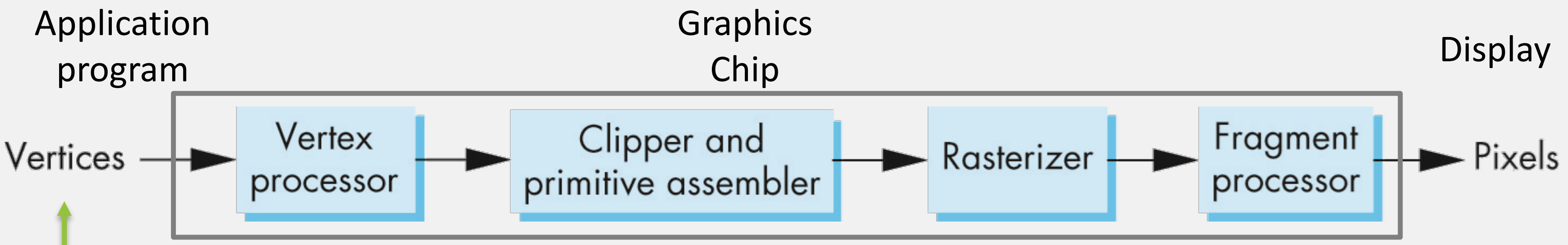
Junho Kim

Kookmin University

Clipping

Where we are in Rendering Pipeline is …

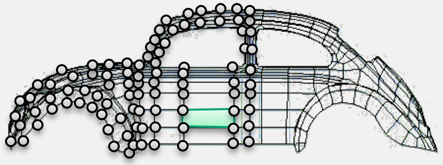
- Pipeline architecture
  - This is everything for interactive computer graphics!
    - First, we focus on the *fixed rendering pipeline*
  - Mechanism: a *state* machine
    - All information for image formations should be specified



Positions: (x, y ,z), …
Normals: (nx, ny, nz), …
Texcoords: (s, t), …
Primitive type: Quads

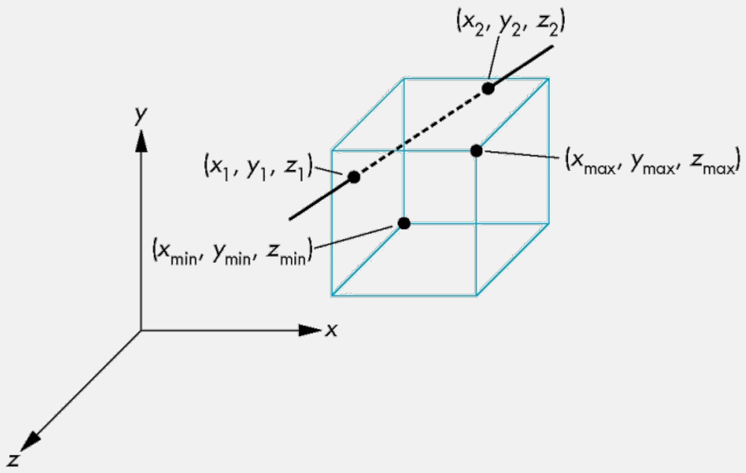
Our input

Our output



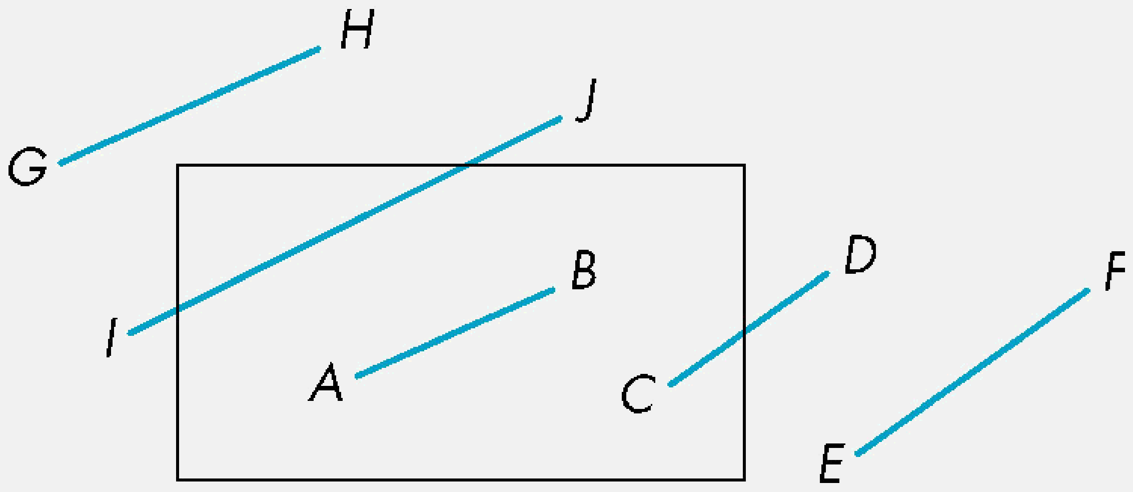
Clipping

- To eliminate objects that lie outside of viewing volume
  - Performed in several places in the pipeline
  - Accept / Reject(or cull)
  - Supported by H/W or S/W



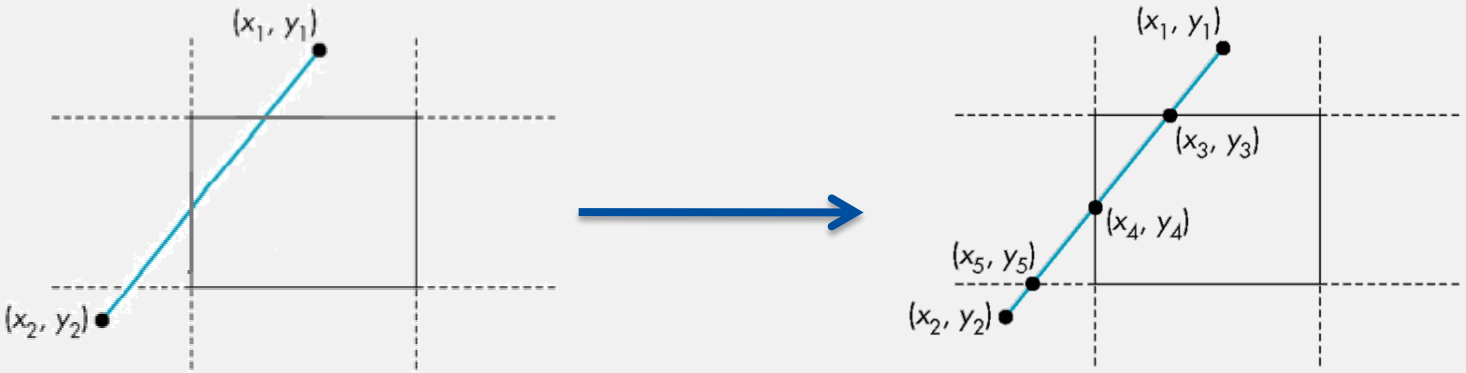
Line-Segment Clipping

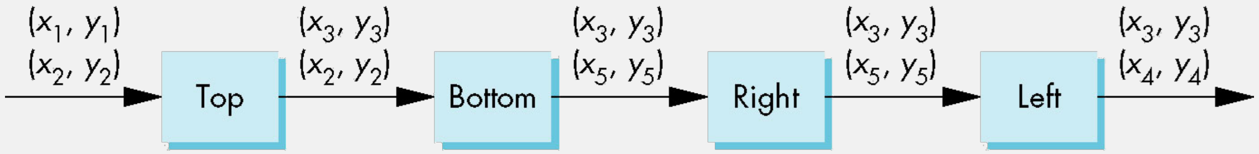
- From an input line, to clip out a portion which passes through the view volume



Line-Segment Clipping

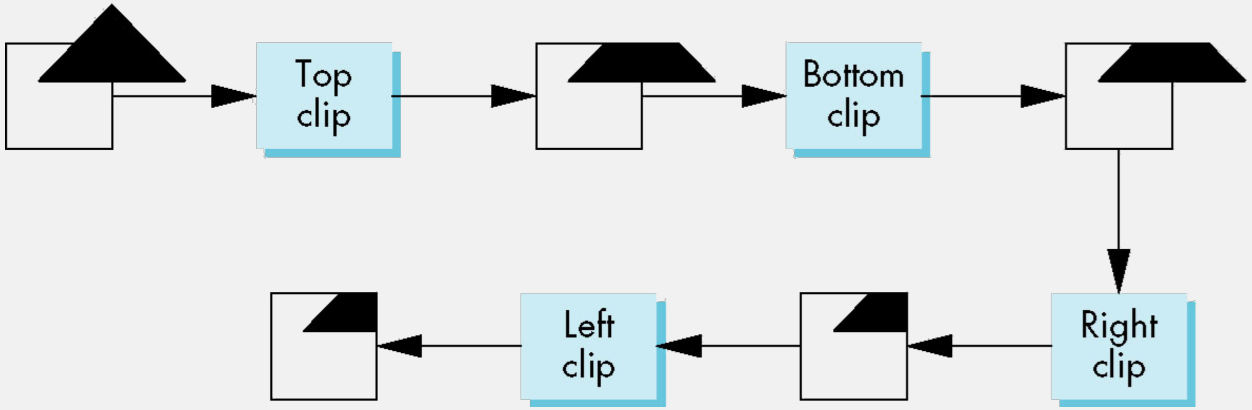
- From an input line, to clip out a portion which passes through the view volume





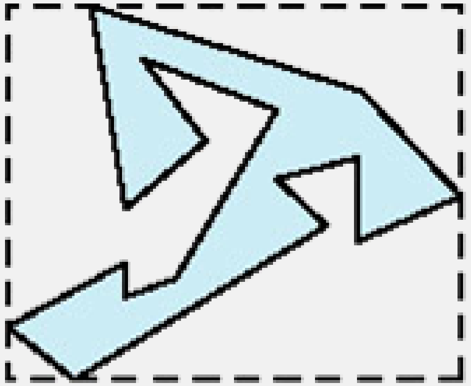
Polygon Clipping

- From a given polygon, to clip out portions which are inside of the view volume



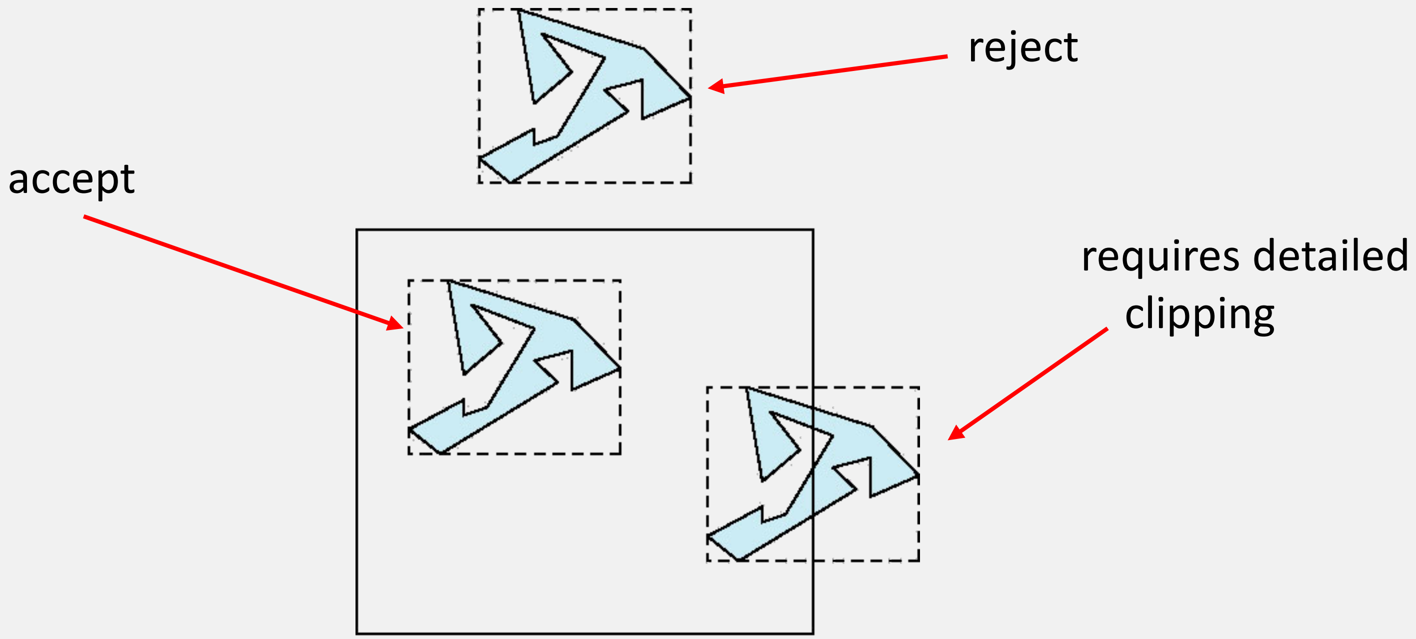
Bounding Boxes and Volumes

- Rather than doing clipping on a complex polygon, we can use an axis-aligned bounding box or extent
  - Usually, used in the game-engine



Bounding Boxes and Volumes

- Determine accept/reject based only on bounding box



Rasterization

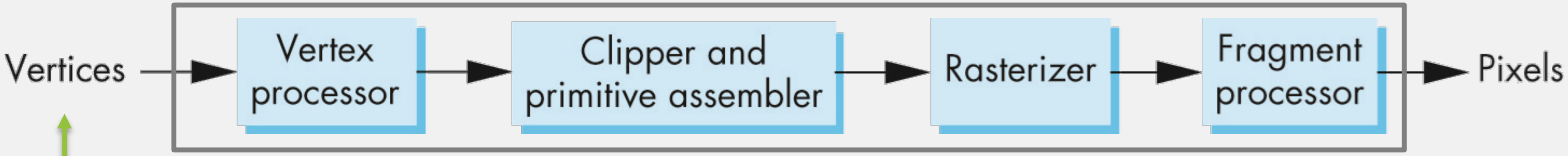
Where we are in Rendering Pipeline is …

- Pipeline architecture
  - This is everything for interactive computer graphics!
    - First, we focus on the *fixed rendering pipeline*
  - Mechanism: a *state* machine
    - All information for image formations should be specified

Application program

Graphics Chip

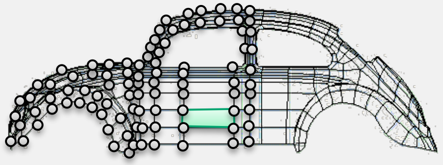
Display



Vertices → Vertex processor → Clipper and primitive assembler → Rasterizer → Fragment processor → Pixels

Positions: (x, y ,z), …
Normals: (nx, ny, nz), …
Texcoords: (s, t), …
Primitive type: Quads

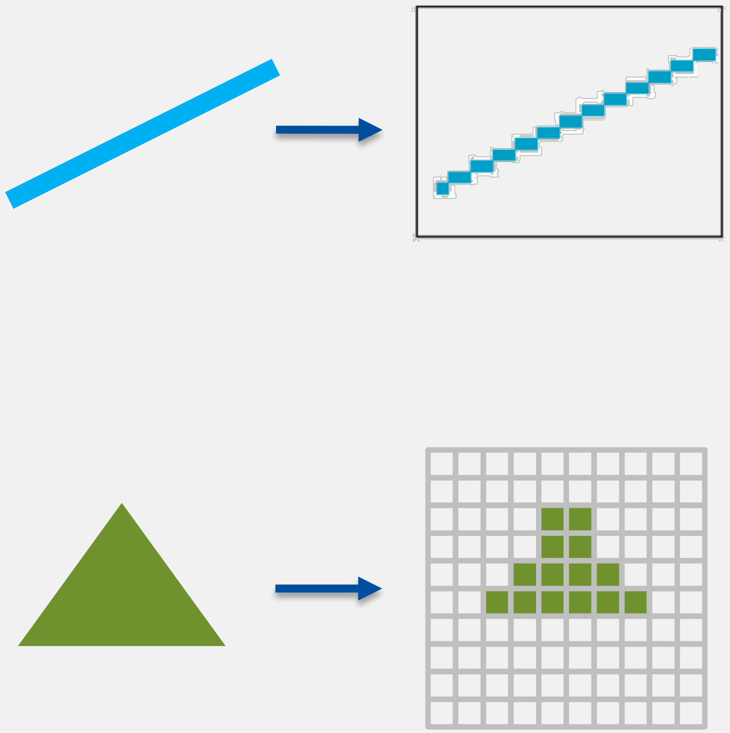
Our input



Our output

Rasterization

- Rasterization (Scan conversion)
  - The process of converting a primitive into a set of pixels
  - It computes
    - Fragment location
      - Which pixels that are inside primitive specified by a set of vertices
    - Per-fragment attributes
      - Attributes, such as color and texture coordinates are determined by interpolating values at vertices

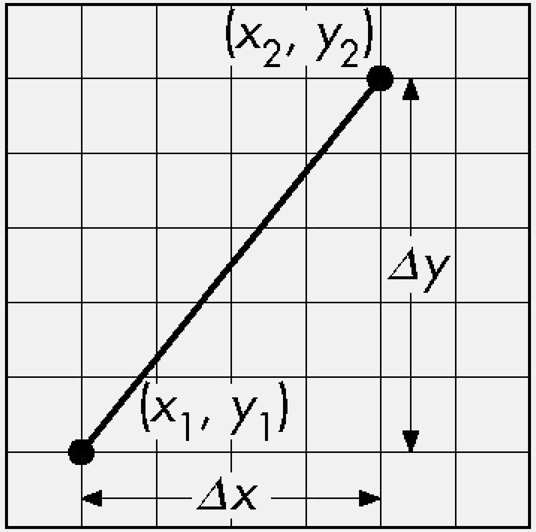


Line Rasterization

- DDA algorithm
  - With a given line equation (i.e., *y* = *mx* + *h*), compute *y* by increasing x by Δx

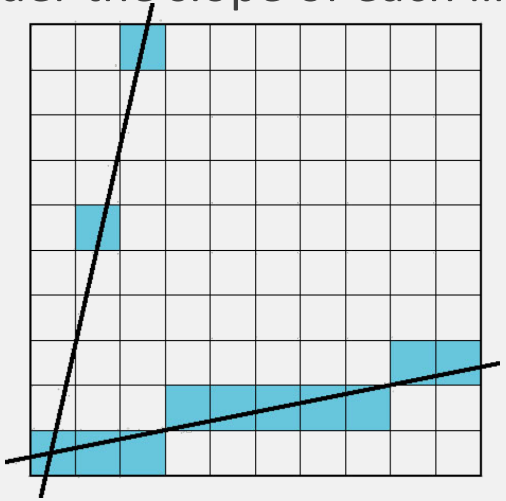
$$y = mx + h$$

$$m = \frac{\Delta y}{\Delta x}$$

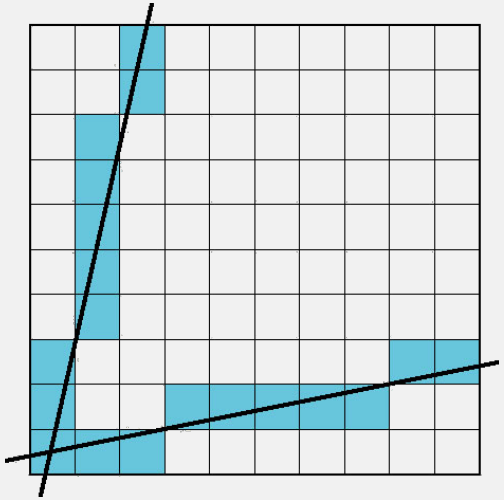


Line Rasterization

- DDA algorithm
  - We have to consider the slope of each line



DDA Algorithm w/o considering slopes

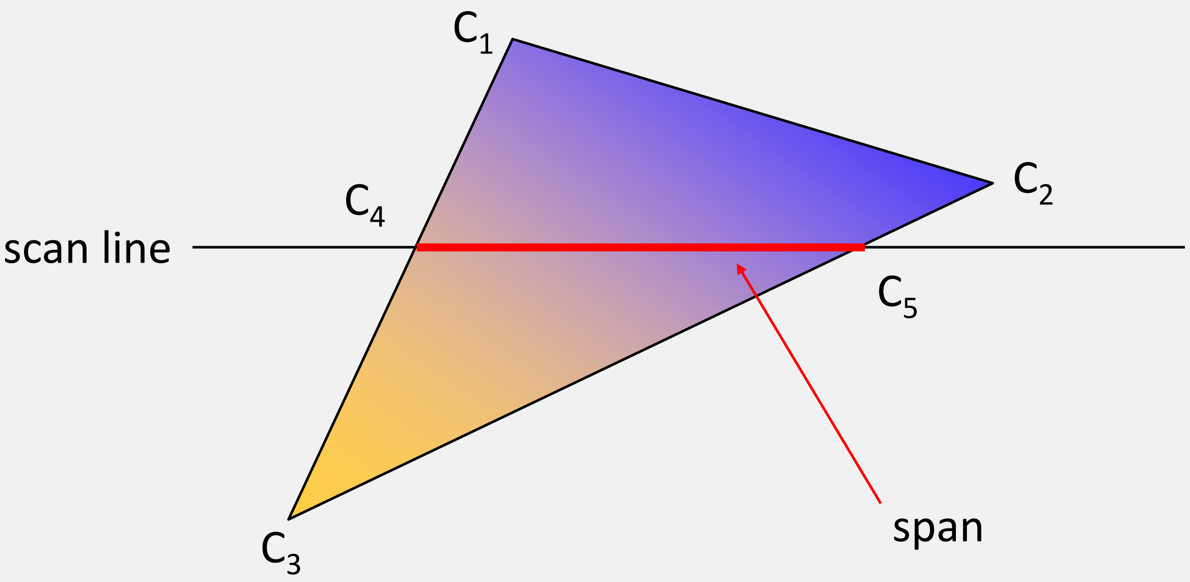


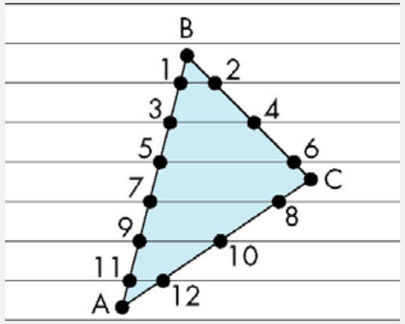
DDA Algorithm w considering slopes

- Bresenham's line algorithm is implemented in graphics HW, in practice

Polygon Rasterization

- Bilinear interpolation
  - First, colors on the line is interpolated
  - Second, colors on each scan line interpolated
- Here, several attributes are interpolated over the fragments in a triangle

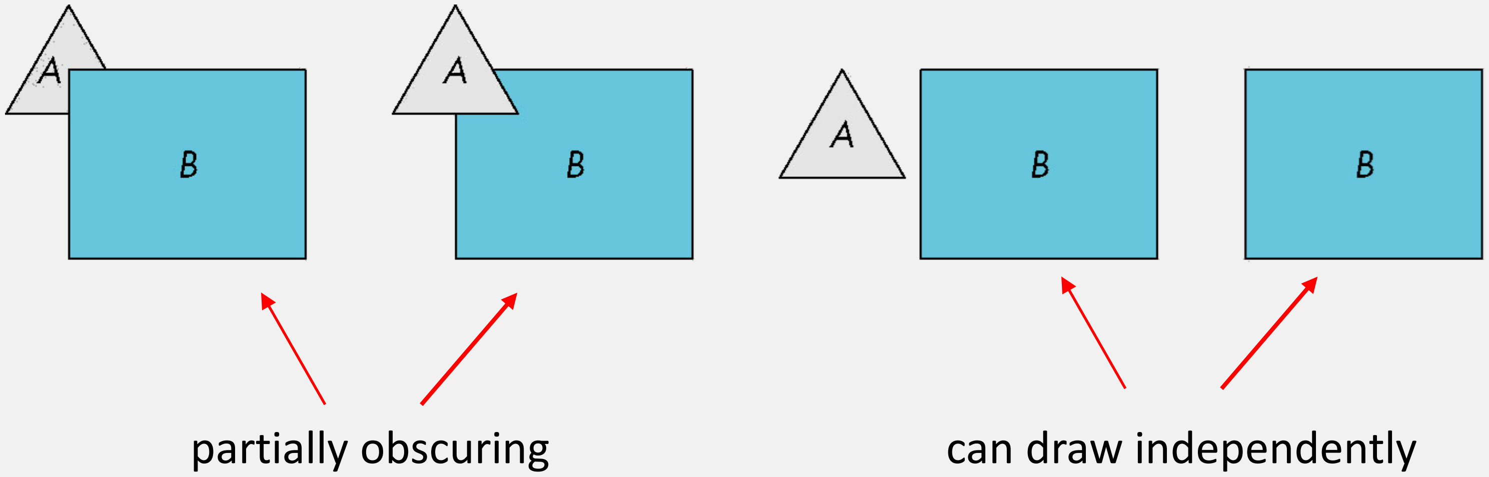




Hidden Surface Removal

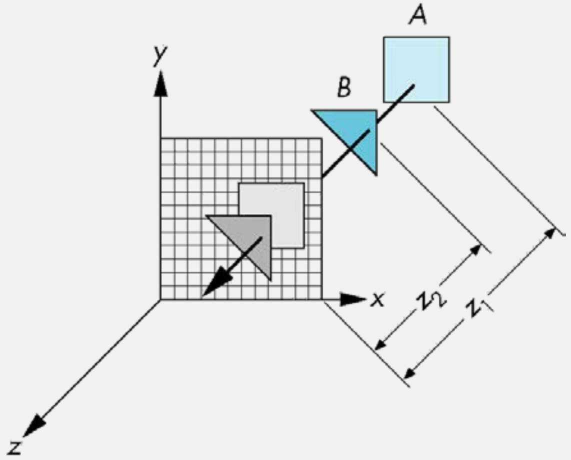
Hidden Surface Removal

- General concept



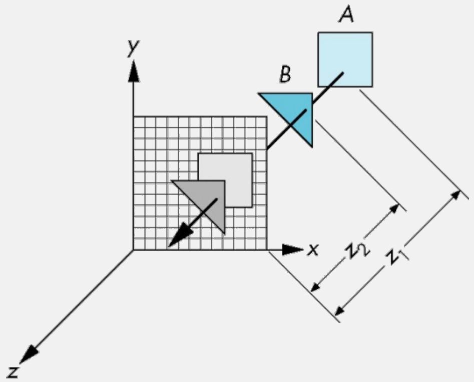
Hidden Surface Removal – z-buffer Algorithm

- z-buffer (or depth-buffer) algorithm
  - It uses a buffer called z- or depth-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



Hidden Surface Removal – z-buffer Algorithm

- z-buffer (or depth-buffer) algorithm
  - We have an additional buffer whose size is identical to the color buffer
  - Each pixel in the depth buffer keeps a depth-value, which is the sitance to the point on the nearest object from the synthetic camera



			🟩	🟩			
			🟩	🟩			
		🟩	🟩	🟩	🟩		
	🟩	🟩	🟩	🟩	🟩	🟩	

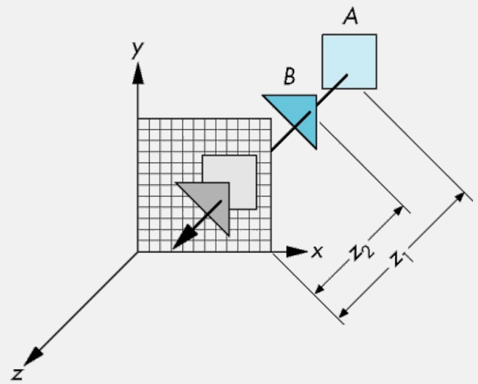
Color buffer

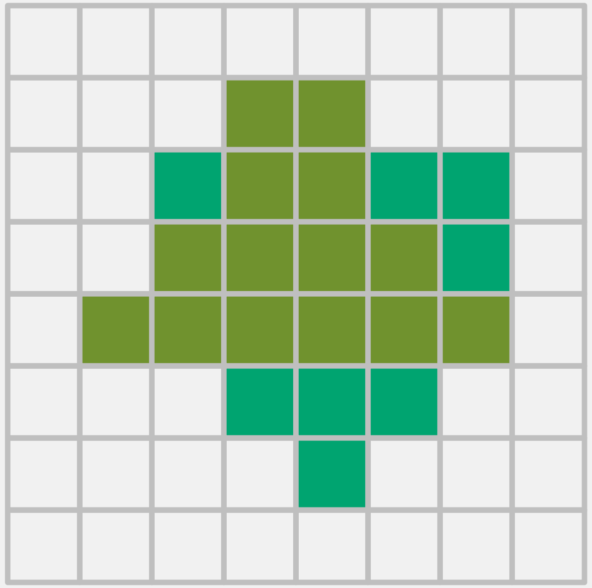
			0.7	0.6			
			0.8	0.7			
		1.0	0.9	0.8	0.7		
	1.2	1.1	1.0	0.9	0.8	0.7	

Depth buffer

Hidden Surface Removal – z-buffer Algorithm

- z-buffer (or depth-buffer) algorithm
  - We have an additional buffer whose size is identical to the color buffer
  - Each pixel in the depth buffer keeps a depth-value, which is the sitance to the point on the nearest object from the synthetic camera





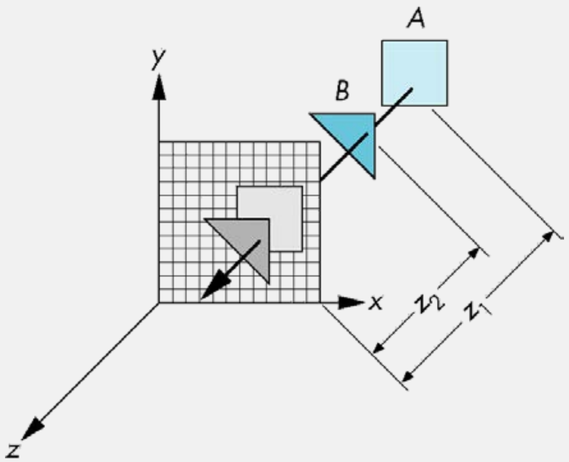
Color buffer

			0.7	0.6			
		2.5	0.8	0.7	2.8	2.9	
		1.0	0.9	0.8	0.7	2.9	
	1.2	1.1	1.0	0.9	0.8	0.7	
			2.5	2.6	2.7		
				2.5			

Depth buffer

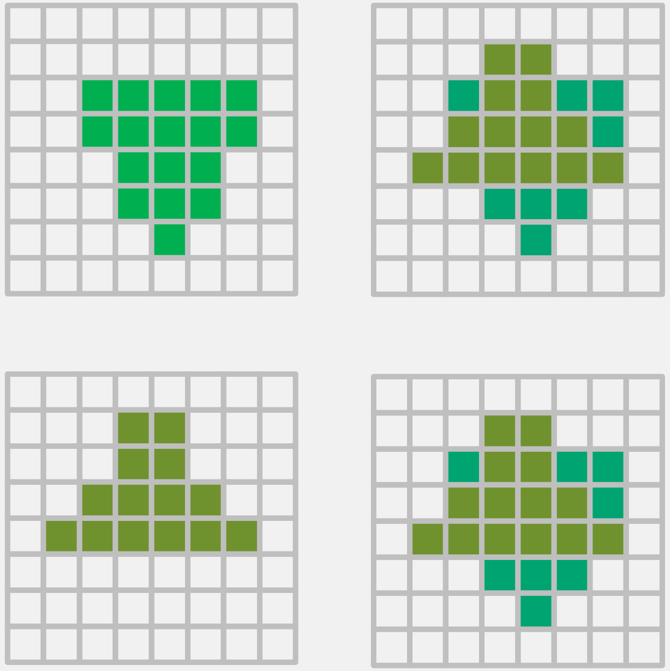
Hidden Surface Removal – z-buffer Algorithm

- Advantages
  - The programmer do not need to care about the rendering order of objects in a scene
    - A → B
    - B → A



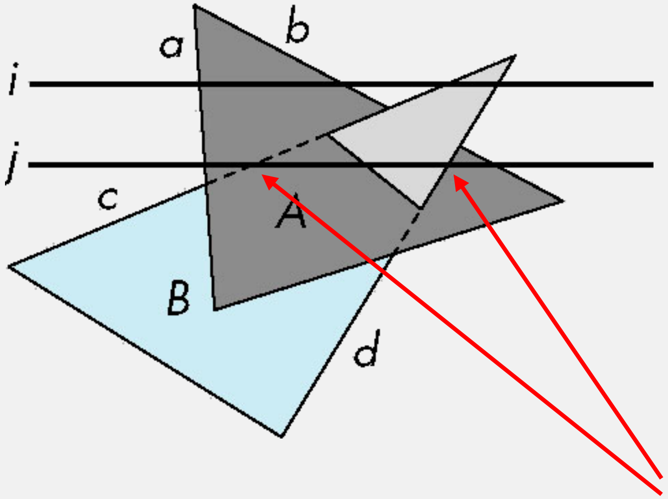
A → B

B → A



Hidden Surface Removal – z-buffer Algorithm

- Advantage
  - Z-buffer algorithm can combine shading and hiddn surface removal through scan line algorithm

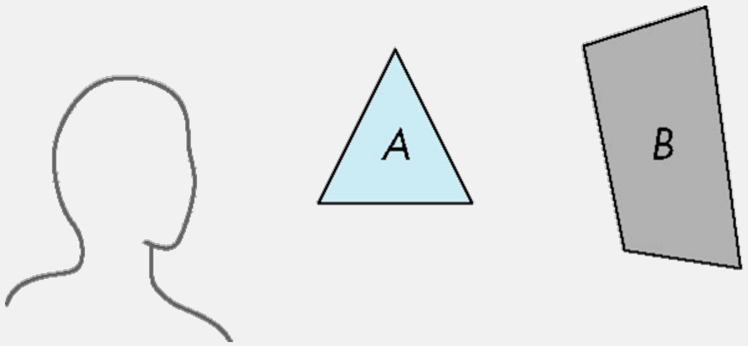


scan line *i*: no need for depth information, can only be in no or one polygon

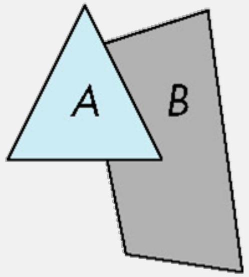
scan line *j*: need depth information only when in more than one polygon

Hidden Surface Removal – Painter's Algorithm

- A kind of S/W approach
- Render polygons a back to front order so that polygons behind others are simply painted over
  - We need to perform a sorting algorithm



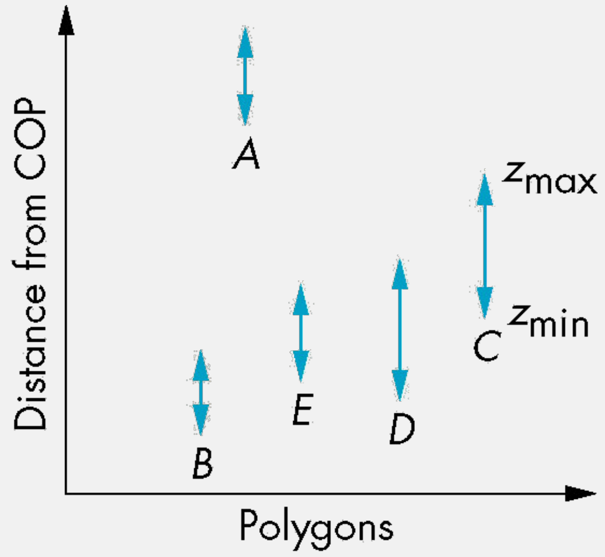
B behind A as seen by viewer



1) Fill B
2) Fill A

Hidden Surface Removal – Painter's Algorithm

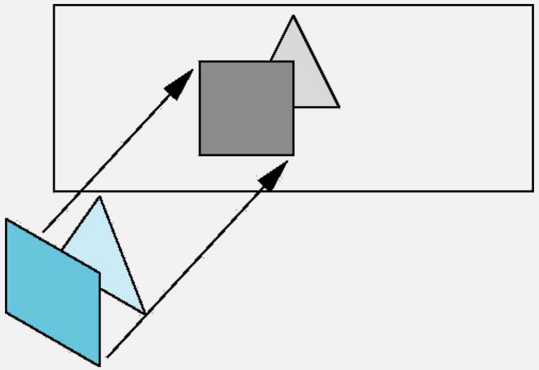
- Requires ordering of polygons first
  - O(*n*log*n*) caculation for ordering
  - Not every polygon is eithter in front or behind all other polygons



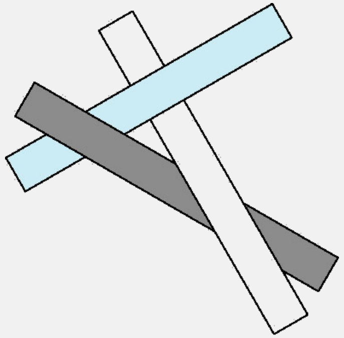
Polygons sorted by distance from COP

Hidden Surface Removal – Painter's Algorithm

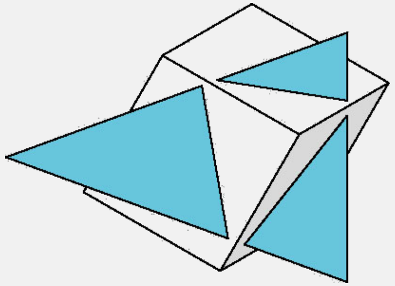
- There are non-trivial cases exist for depth sorts
- Why?



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



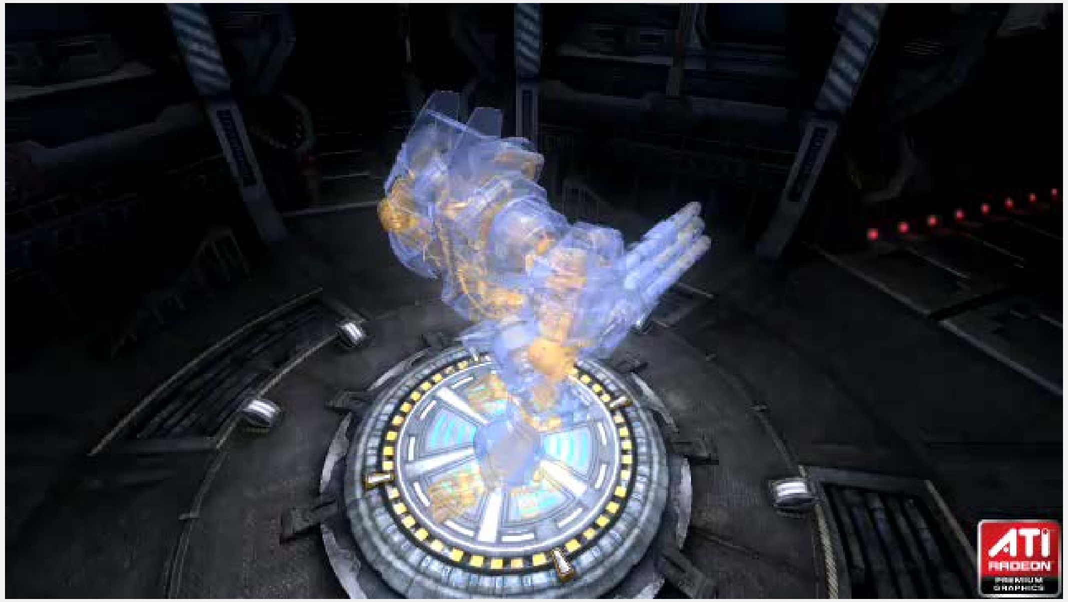
cyclic overlap



penetration

Hidden Surface Removal – Painter’s Algorithm

- Why we should learn about painter’s algorithm, even though graphics HW supports the z-buffer algorithm
  - For using alpha blending, we have to render polygons a back to front order



[AMD DirectX 11 Demo for H/W accelerated alpha blending]
(video, youtube)

Hidden-Surface Removal – Back-face Removal

- Back-face removal (culling) algorithm
  - Face is visible iff $90 \geq \theta \geq -90$
    - equivalently $\cos\theta \geq 0$ or $\mathbf{v} \bullet \mathbf{n} \geq 0$
  - Recall that you always send the vertices in a polygon in the order of CCW.
    - Simply, almost 50% polygons are culled from the back-face removal