CSE-170 Computer Graphics

Lecture 12

Visible-Surface Determination

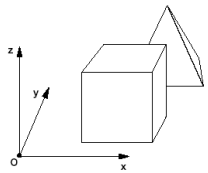
Dr. Renato Farias
rfarias2@ucmerced.edu

Visible-Surface Determination

Remembering the Rendering Pipeline

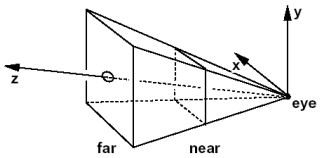
Modeling Transformations

∀ Objects

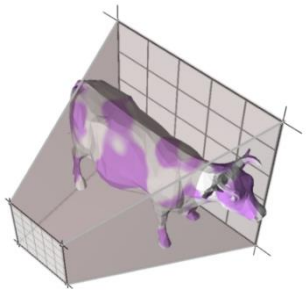


Lighting

Viewing/Camera Transformations



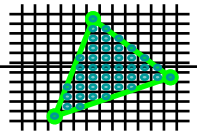
Projection Transformation



Visible surfaces will be determined somewhere here

Clipping

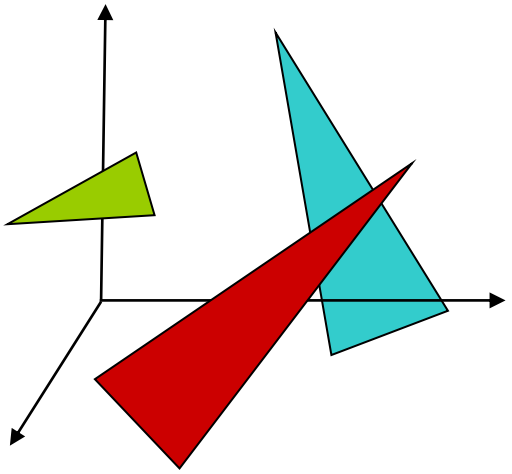
Scan Conversion



Image

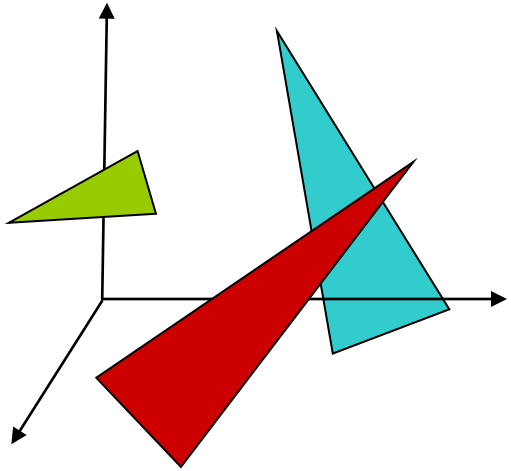
Visible-Surface Determination

- Different algorithms exist
- Algorithms may try to exploit “coherences”
  - face coherence
  - frame coherence
  - etc.
- Spatial organization/subdivision may also be used
  - only compare objects projected on a same cell, etc.
- Generic solution should work for “triangle soups”



Back-face culling

- Back-face culling
  - Allows to only draw a polygon if its normal is facing the camera
    - Significant optimization



Name

glCullFace — specify whether front- or back-facing facets can be culled

C Specification

```
void glCullFace( GLenum mode);
```

Parameters

mode

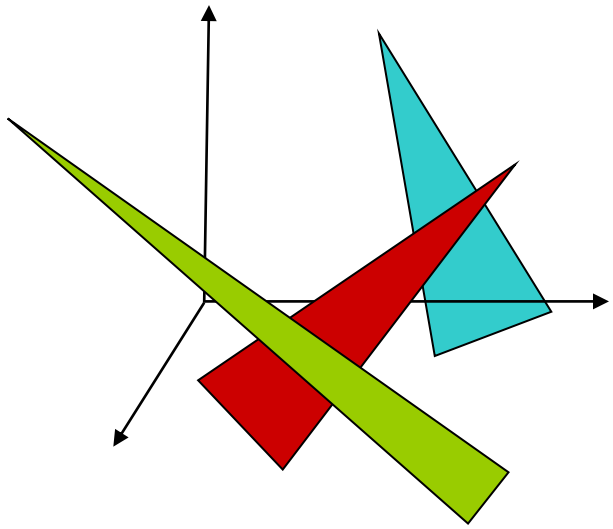
Specifies whether front- or back-facing facets are candidates for culling. Symbolic constants `GL_FRONT`, `GL_BACK`, and `GL_FRONT_AND_BACK` are accepted. The initial value is `GL_BACK`.

Version Support

	OpenGL Version											
Function / Feature Name	2.0	2.1	3.0	3.1	3.2	3.3	4.0	4.1	4.2	4.3	4.4	4.5
glCullFace	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Visible-Surface Determination

- Which polygons go in front and which go behind during rasterization?
  - Depth-Sorting or Painter's Algorithm
  - BSP trees
  - Z-Buffer
  - Scan Line
  - there are others
    - area subdivision, etc.



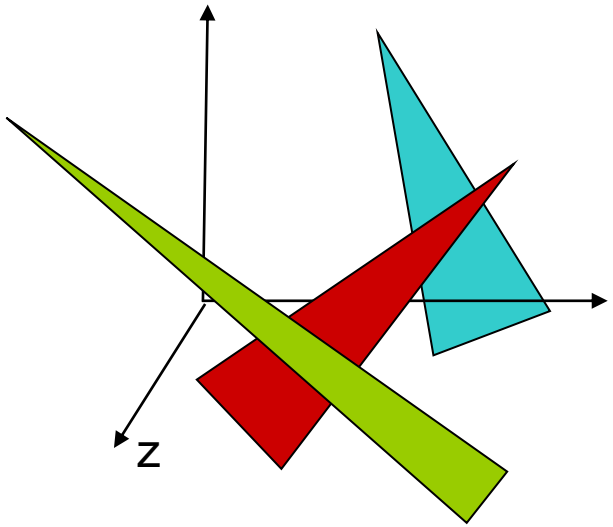
Depth Sorting or Painter's Algorithm

Depth Sorting

- Also called the painter's algorithm:

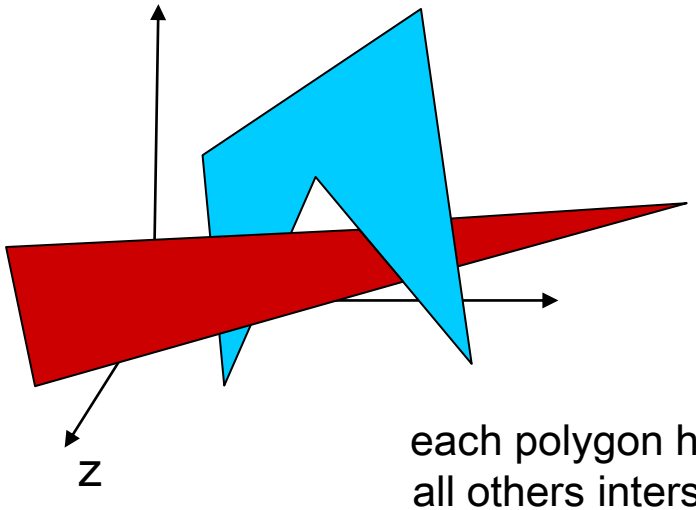
1. Sort polygons according to z coordinate
2. “Paint” polygons in order, starting from the polygons farther away from the camera

Is that all?

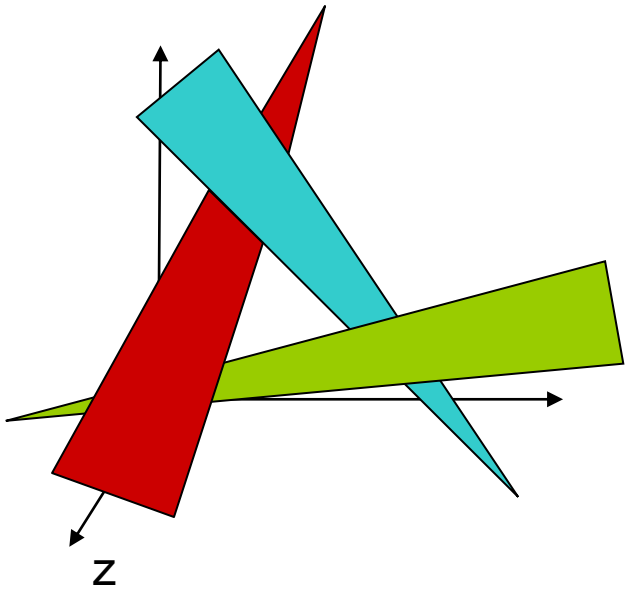


Depth Sorting

1. Sort polygons according to z coordinate
2. Resolve ambiguities when polygon's z extents overlap, split polygons if needed
3. “Paint” polygons in order, starting from the polygons farther away from the camera



each polygon has to test all others intersecting its “z extent”



Binary Space Partition (BSP) Trees

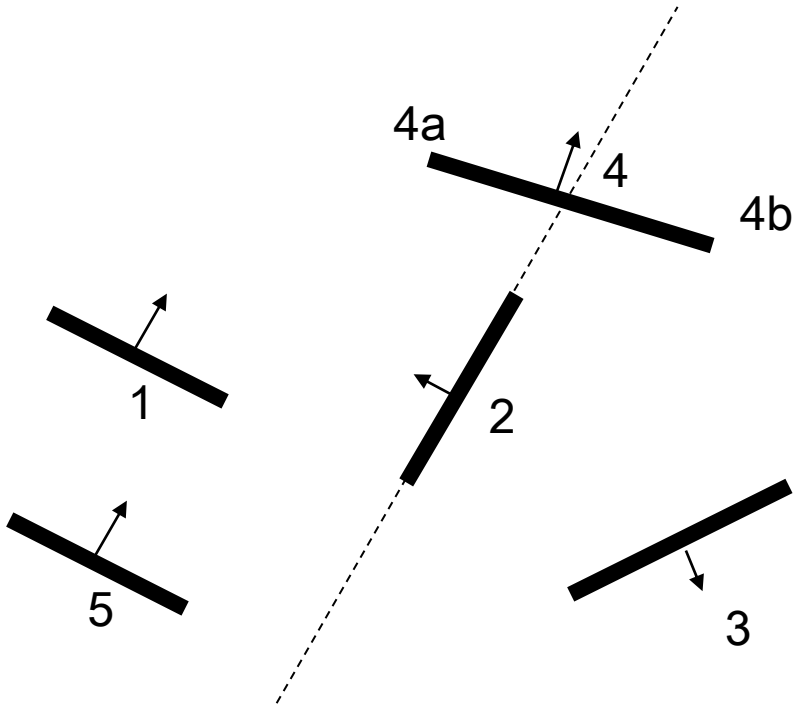
BSP trees

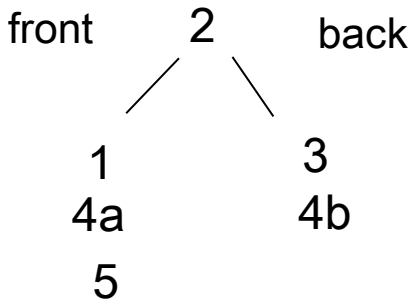
1. Build a tree “ordering” the polygons according to “half spaces separability”
2. Traverse the tree according to view point

(For clarity, the next slides depict examples in 2D to explain how the method works in 3D)

BSP tree construction 1/3

1) Choose a separation plane passing by a polygon and classify other polygons in front or back spaces; subdivide if needed

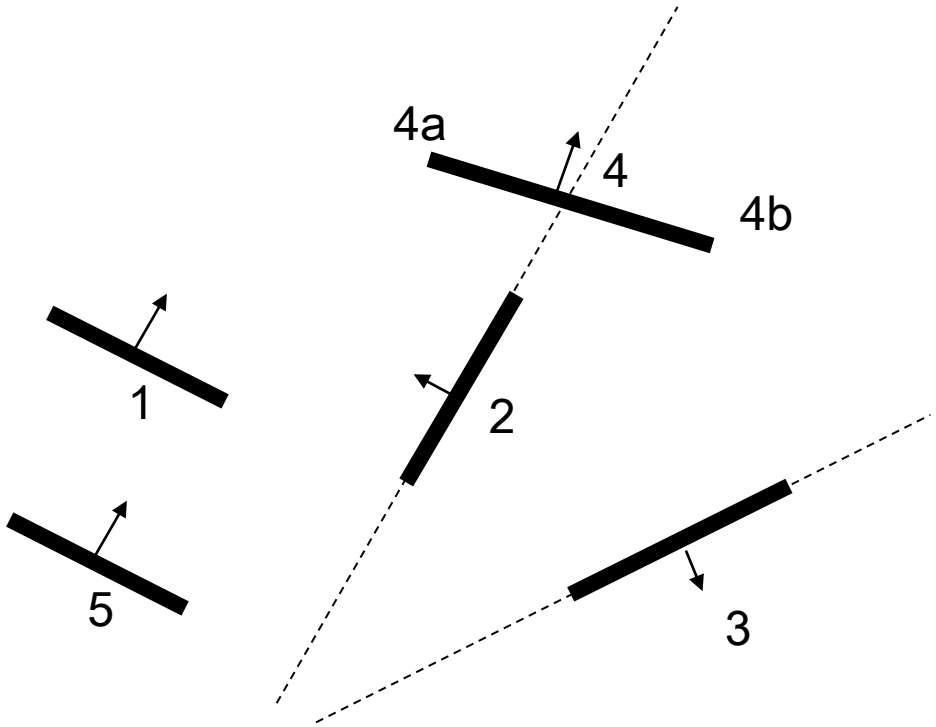


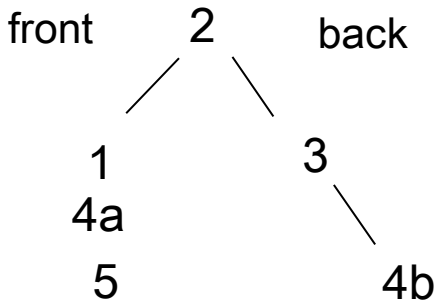


(Important: the normal vector of a polygon is used to define its “front”)

BSP tree construction 2/3

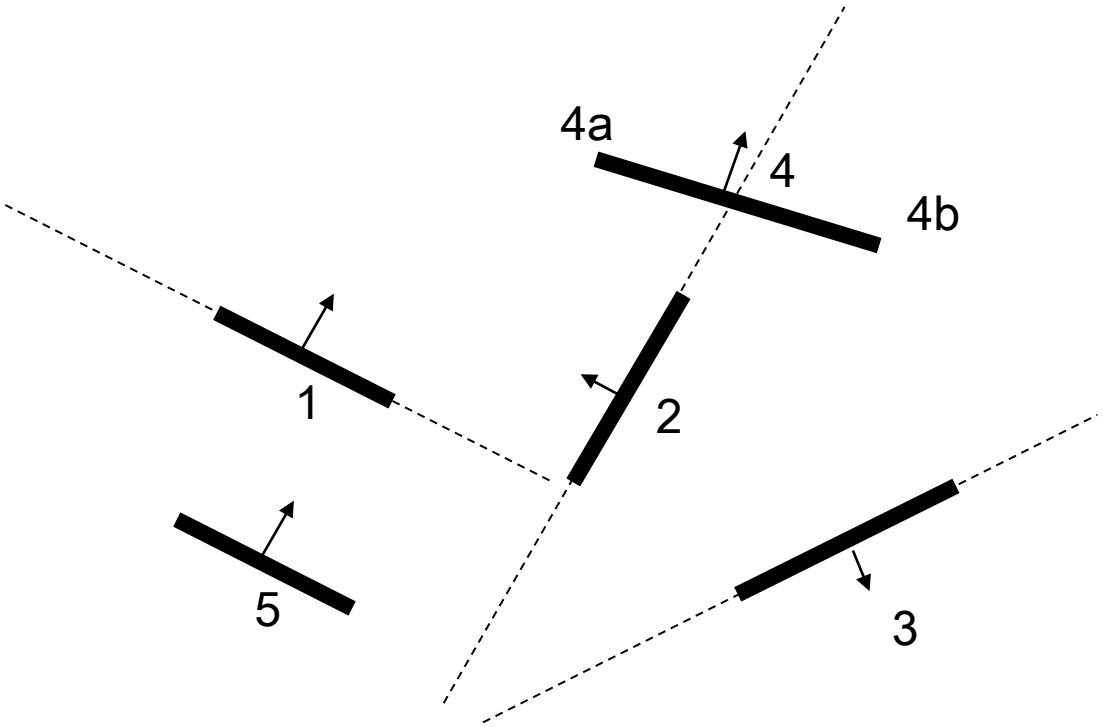
2) Repeat for each unprocessed “front and back space”

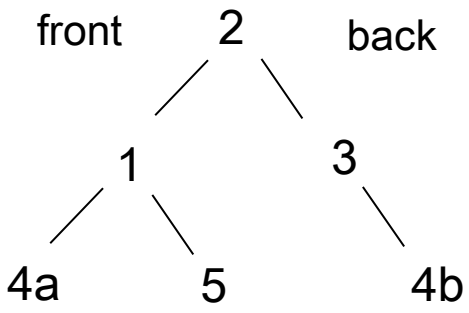




BSP tree construction 3/3

3) Stop when each leaf has a single polygon





BSP tree traversal

Start with node as the root node:

if viewer in node's front half-space

1. display polygons in rear half-space
2. display node’s polygon
3. display polygons in front half-space

else

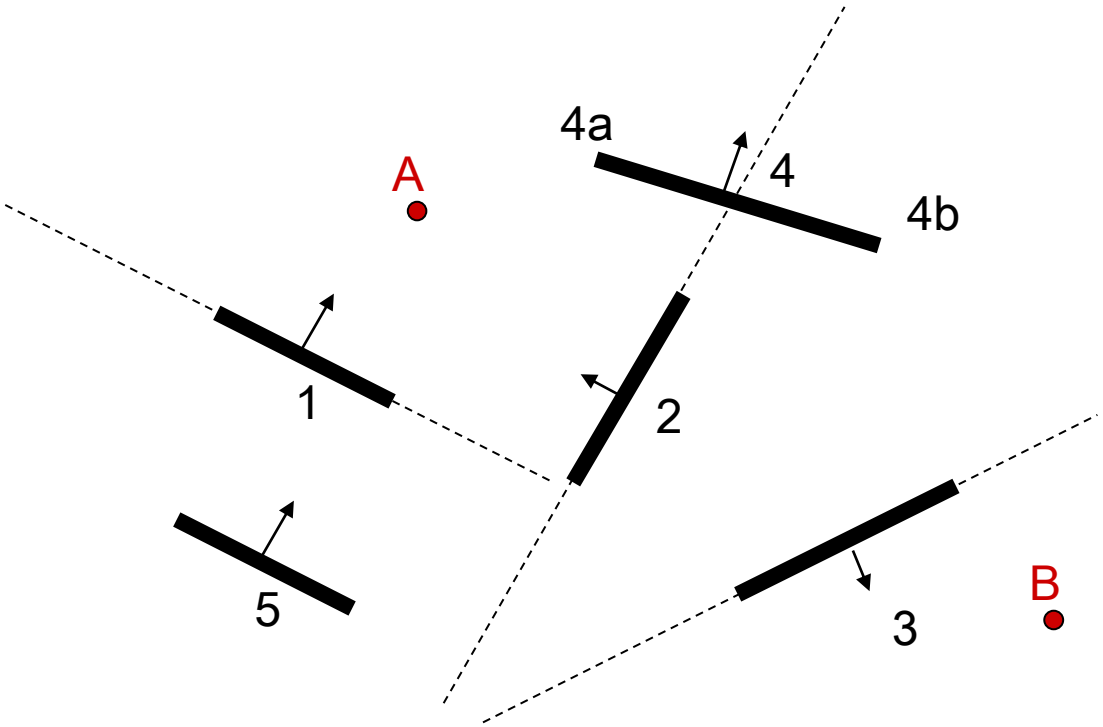
1. display polygons in front half-space
2. display node’s polygon
3. display polygons in rear half-space

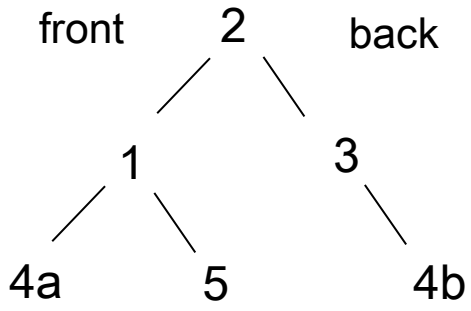
recursively process children of the node

BSP tree traversal

```
traverse ( node )
  if ( node is null ) return;
  if viewer in node's front half-space
    1. traverse ( node->back );
    2. display node;
    3. traverse ( node->front );
  else
    1. traverse ( node->front );
    2. display node;
    3. traverse ( node->back );
```

BSP tree traversal





A: 3, 4b, 2, 5, 1, 4a

B: 5, 1, 4a, 2, 4b, 3

BSP tree traversal

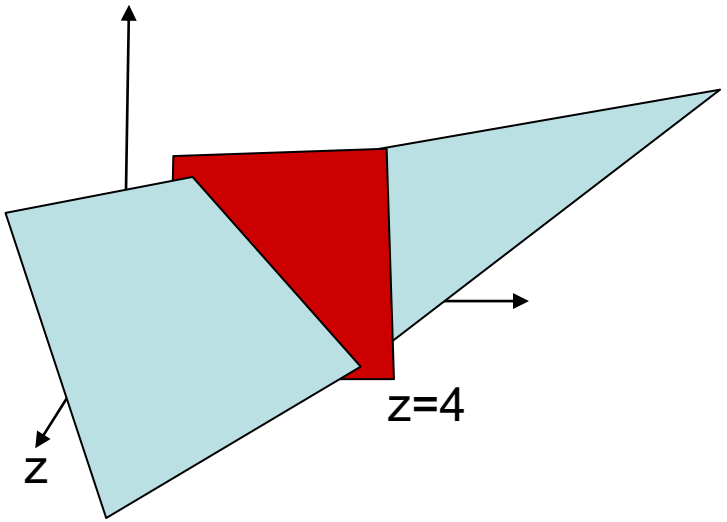
- Tree is constructed at pre-processing
- Viewpoint can move but not polygons
  - Tree has to be reconstructed/updated every time a polygon moves
- Need to know “front” and “back” of polygons
  - easy determination based on normals

- Question
  - What is the main drawback of BSP trees?
    - Have to be recomputed when scene changes

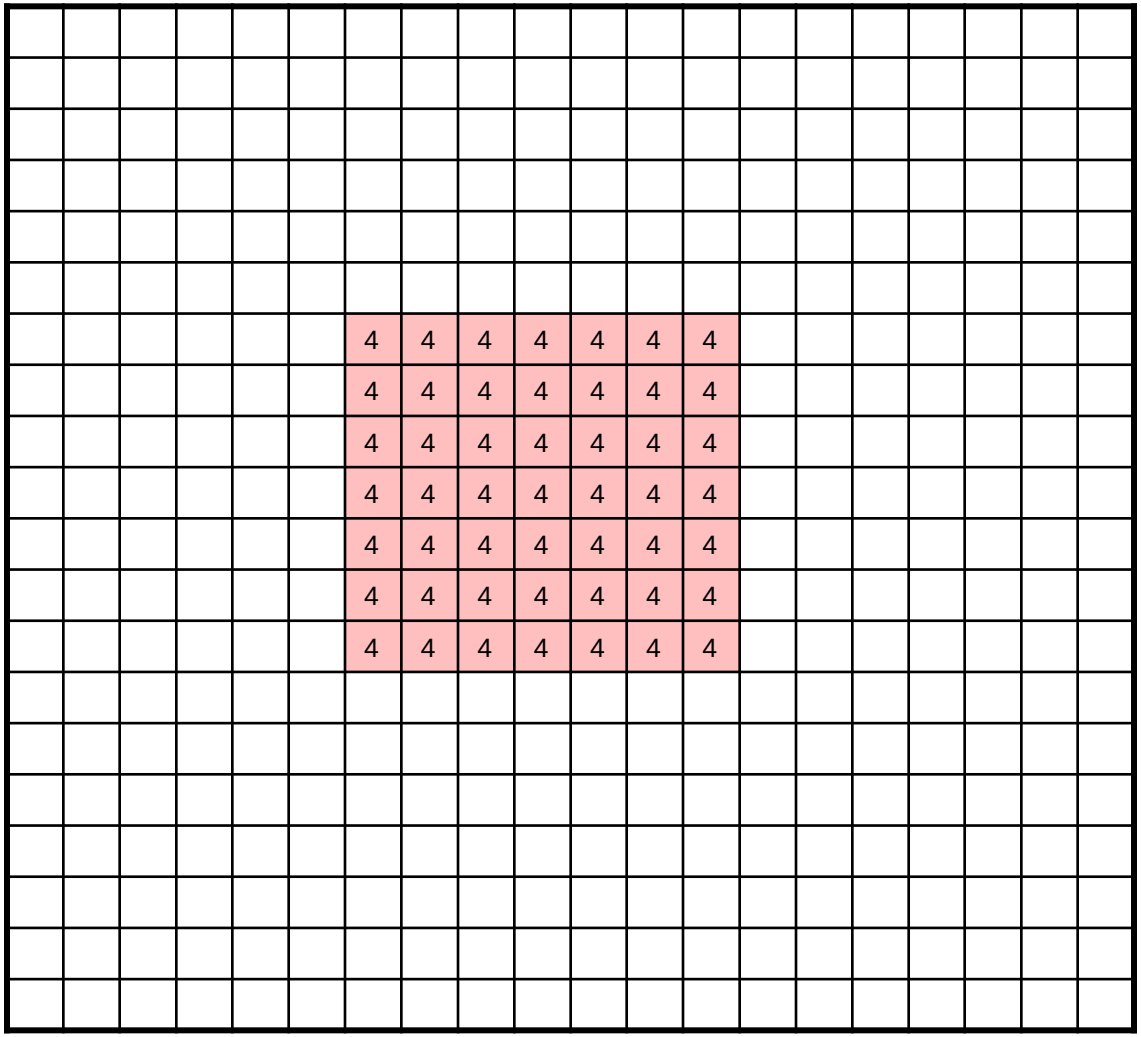
Z-Buffer

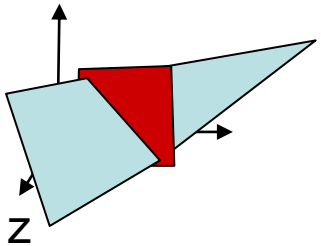
Z-Buffer

- No sorting!
  - Rasterize polygons in any order
- But maintain a buffer:
  - Each pixel in the buffer stores the z-value of the respective pixel in the image
  - New pixels are only displayed if their z-values are greater (closer to viewer) than previous values

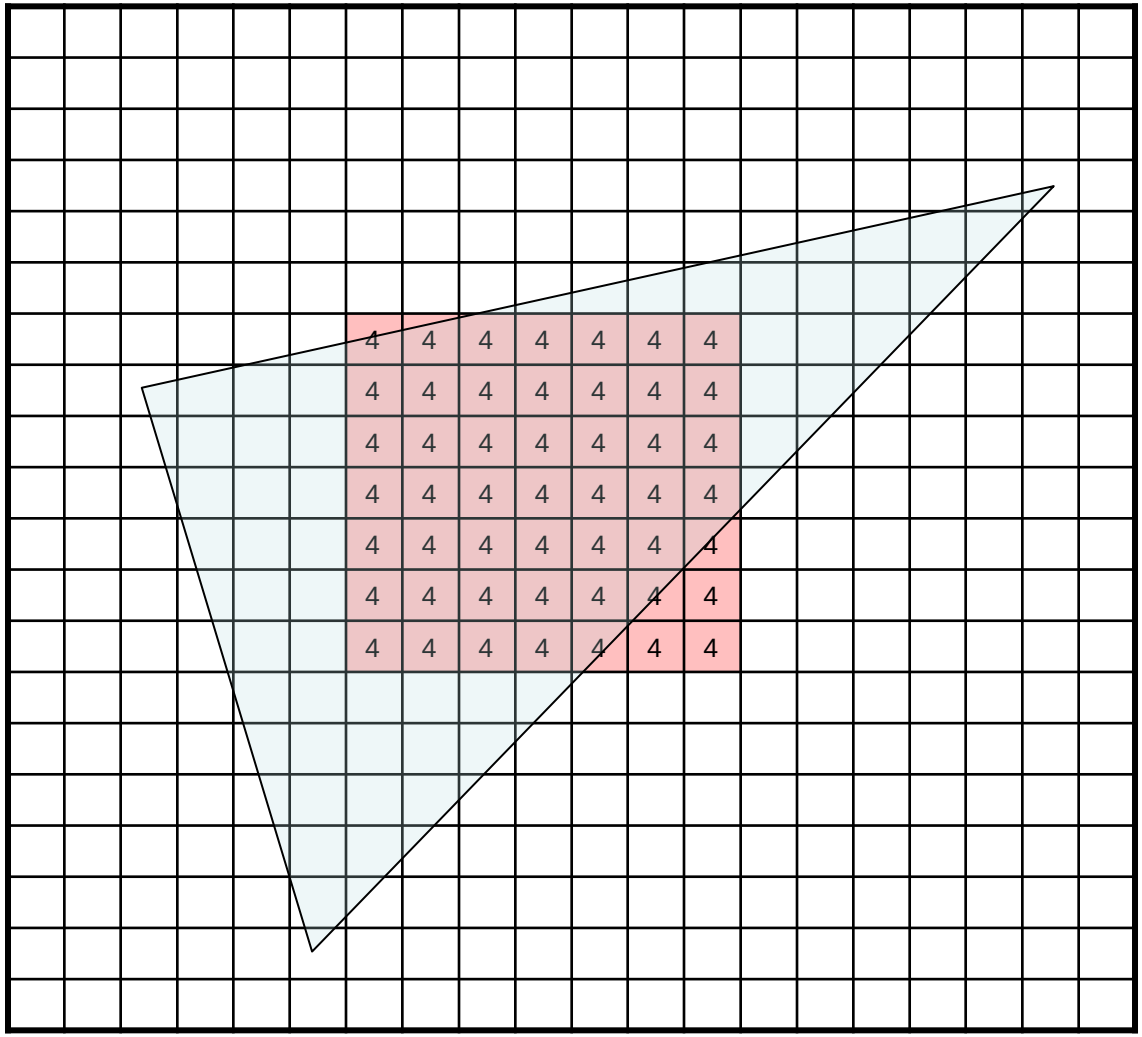


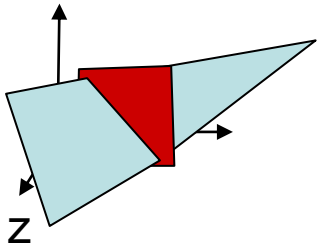
Z-Buffer


4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4

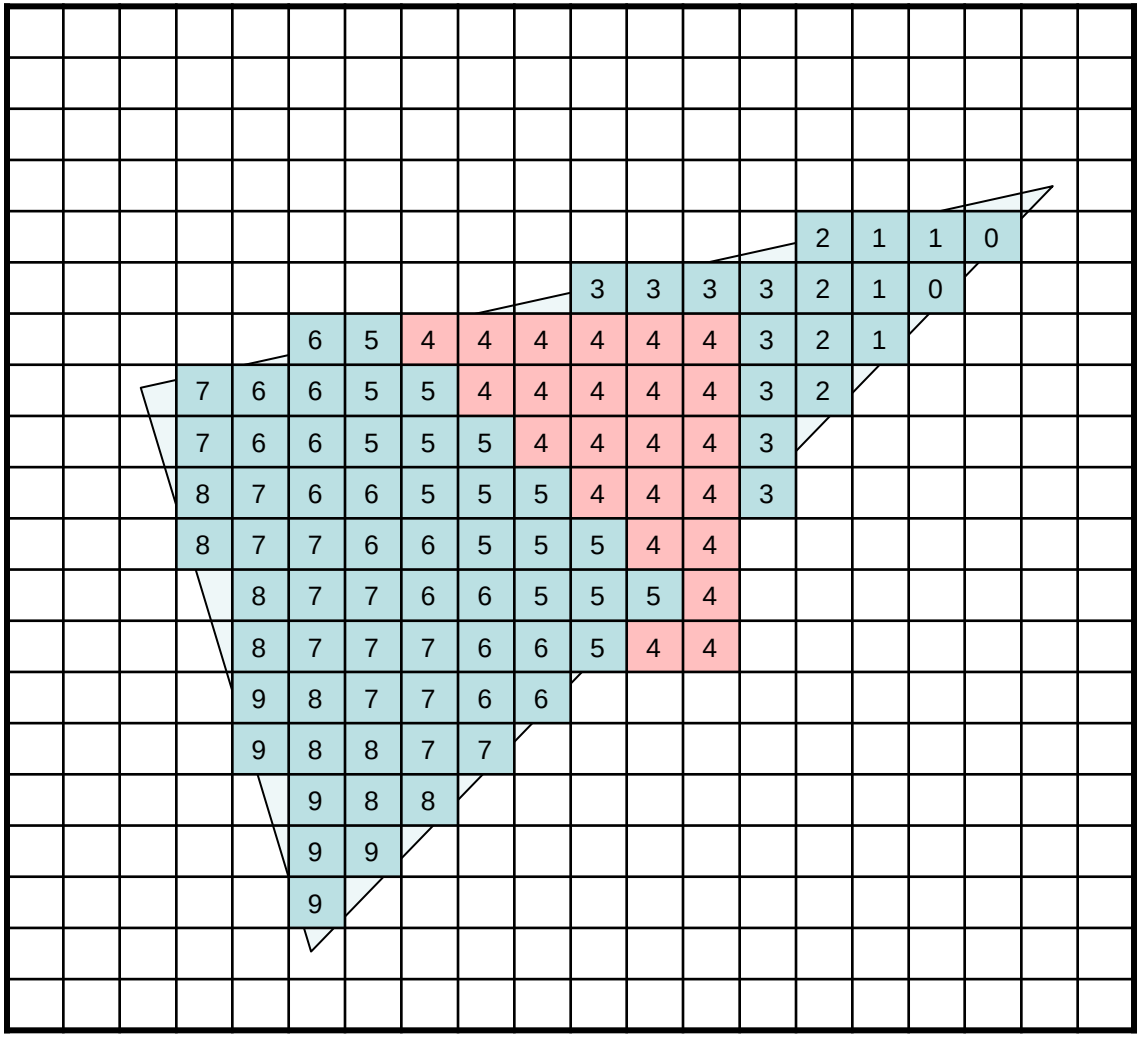

Z

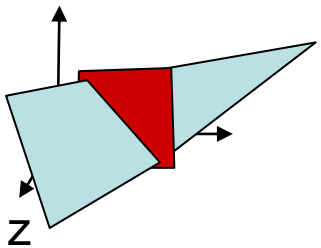
Z-Buffer


4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4
4 4 4 4 4 4 4


Z

Z-Buffer


2 1 1 0
3 3 3 3 2 1 0
6 5 4 4 4 4 4 4 3 2 1
7 6 6 5 5 4 4 4 4 4 3 2
7 6 6 5 5 5 4 4 4 4 3
8 7 6 6 5 5 5 4 4 4 3
8 7 7 6 6 5 5 5 4 4
8 7 7 6 6 5 5 5 4
8 7 7 7 6 6 5 4 4
9 8 7 7 6 6
9 8 8 7 7
9 8 8
9 9
9


Z

Z-Buffer

- Very fast!

- But it uses some memory
  - for ex: 1920x1080x1 ≅ 2MB
  - need to choose depth buffer resolution
  - graphics cards have lots of memory
    - needed for z-buffer, texture buffers, etc.
  - memory is now cheap

- OpenGL uses z-Buffer

Z-Buffer

- OpenGL Buffers
  - Several effects possible with OpenGL buffers
    - There are more buffers than the z-buffer:
      - Color buffer
      - Depth buffer
      - Stencil buffer
      - Accumulation buffer
    - Buffer operations are also possible:

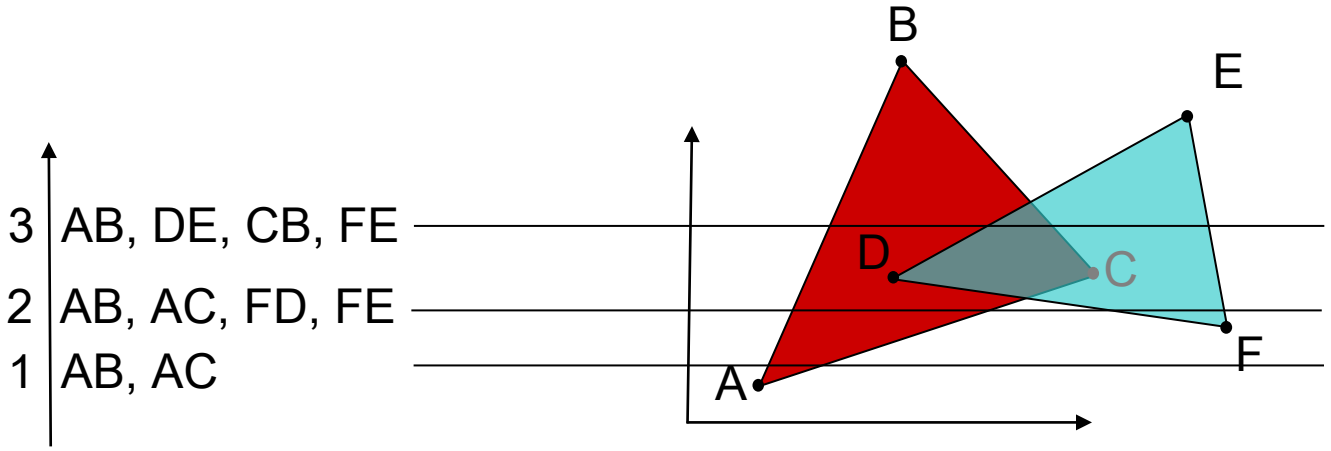
```
glEnable( GL_BLEND ); // for transparency:
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
```

	OpenGL Version											
Function / Feature Name	2.0	2.1	3.0	3.1	3.2	3.3	4.0	4.1	4.2	4.3	4.4	4.5
glBlendFunc	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
glBlendFunci	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓

Scan-Line

Scan-line

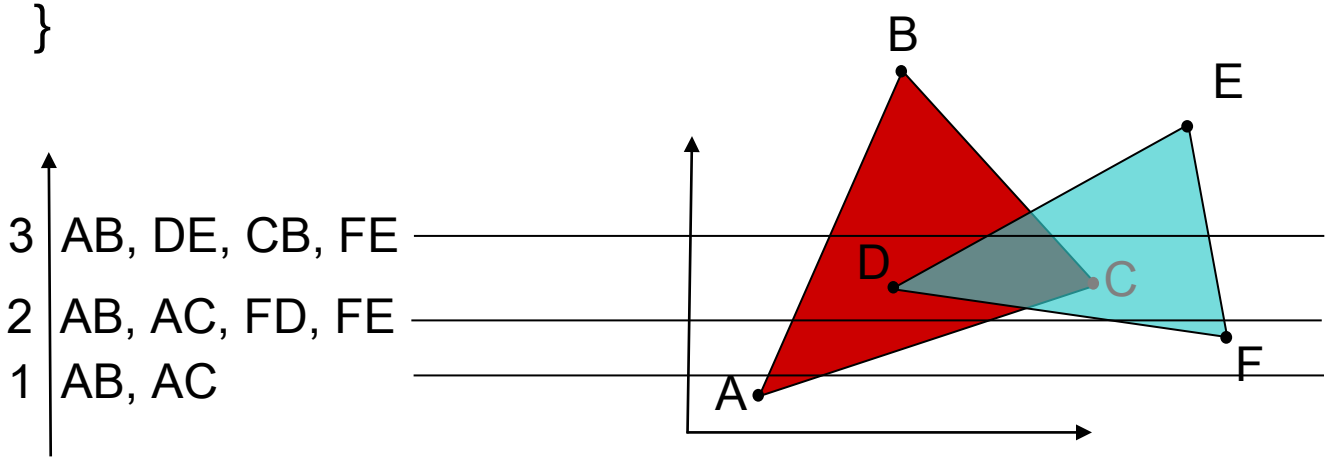
- “Scan line” algorithms represent an efficient and generic approach to process polygons
- The scan line will change “its properties” when the next event happens
  - Events are new vertices encountered on the Y direction
  - All events are sorted vertically as pre-computation
- Tables are used to describe events and the current scan line



Scan-line

- Ex:

```
for (each scan line)
  {   update active surface table;
      for (each pixel on scan line)
      {   determine surfaces in active surface table
              projecting to current pixel;
          find closest surface among them;
          paint pixel;
      }
  }
```



Summary

Summary

- These are the algorithms you have to know:
  - Depth-sorting
  - BSPs
  - Depth buffer (or z-buffer)

- Similar spatial processing algorithms are used for several related problems:
  - Hidden surface elimination for 3D rendering
  - Desktop GUI management
  - 2D drawing tools
  - Collision detection
  - Proximity queries
  - etc.