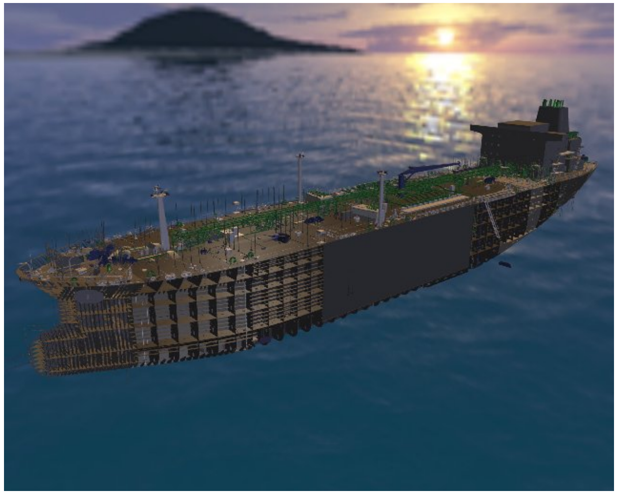
CS475m - Computer Graphics

Lecture 8 : Visibility

Visibility

- What is visible?
  - Which objects are visible?
  - Which pixels(fragments) to render?
- Why check for visibilty?
  - Efficiency
  - Correctness?
  - Disambiguation

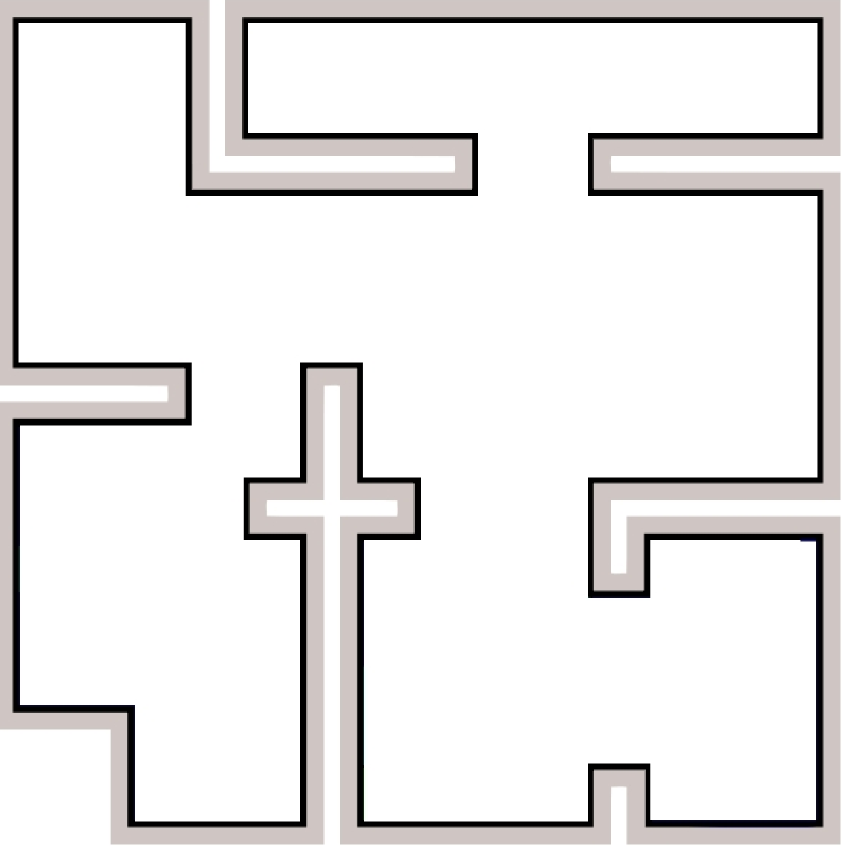


The Double Eagle Tanker:
4GB of data, 82 M Triangles
From: http://www.cs.unc.edu/~geom/hardware/#Vis

Simple question

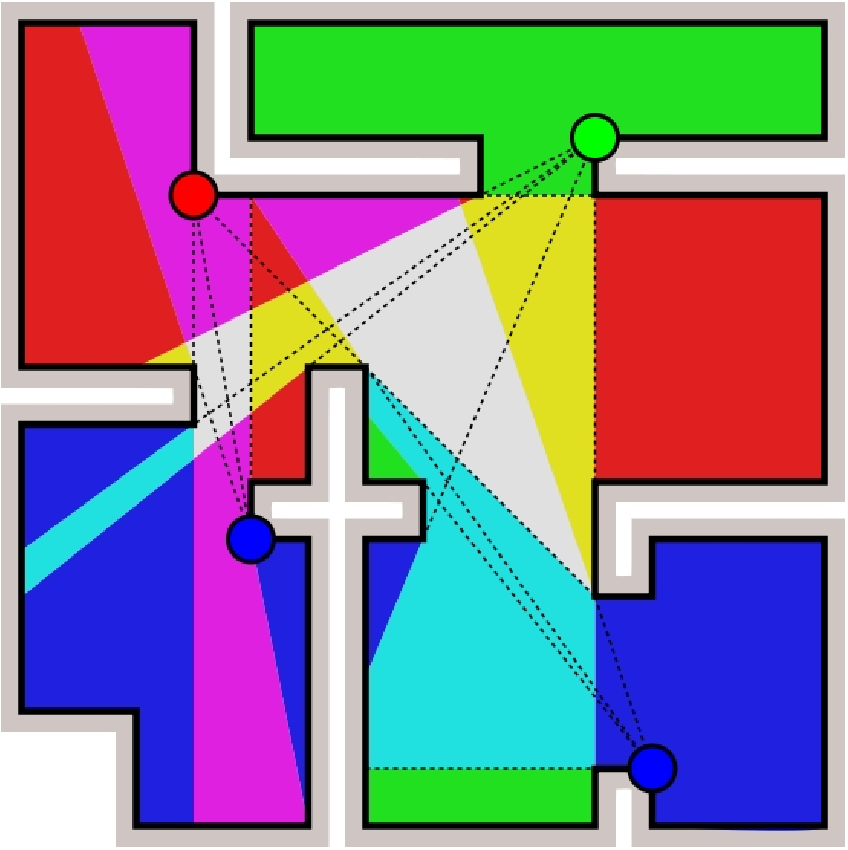
- The *art gallery* problem:

  Given a planar art gallery, what is the minimum number of guards that need to be placed at the corners (but inside the gallery) so that every part of the gallery is visible to at least one guard.



Simple question

- NP – Complete!
- Upper bound: floor(N/3) for a simple polygon with N vertices.
- Determining visibility is not always easy.



Visibility

- The **Image Space** problem formulation

  for (each ***pixel*** in the rendered image)

  {

  - determine the object closest to the viewer that is intercepted by the projector (ray) through the pixel;
  - draw the pixel in the appropriate color;

  }

- Worst case complexity: np

  n : number of objects, p : number of pixels

Visibility

- The **Object Space** problem formulation

  for (every ***object*** in the world)

  {

  - determine those parts of the object whose view is unobstructed by other parts of itself or any other object;
  - draw those parts in the appropriate color;

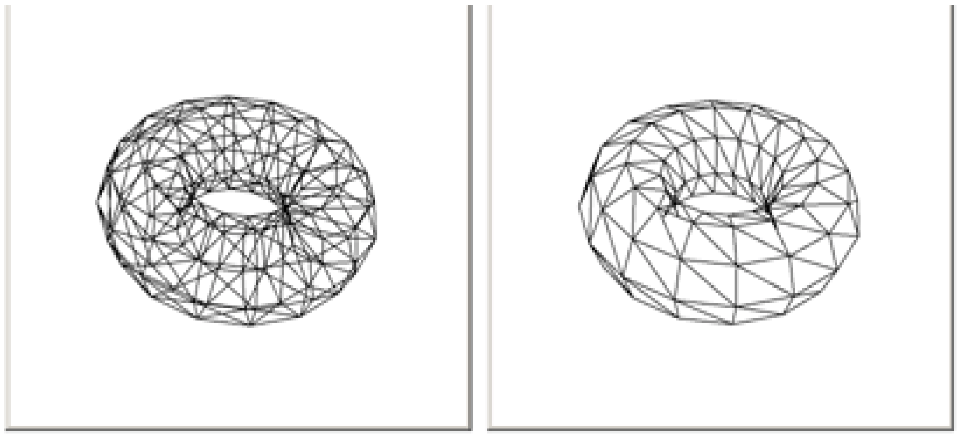
  }

- Worst case complexity: n^2

  n : number of objects

Visibility

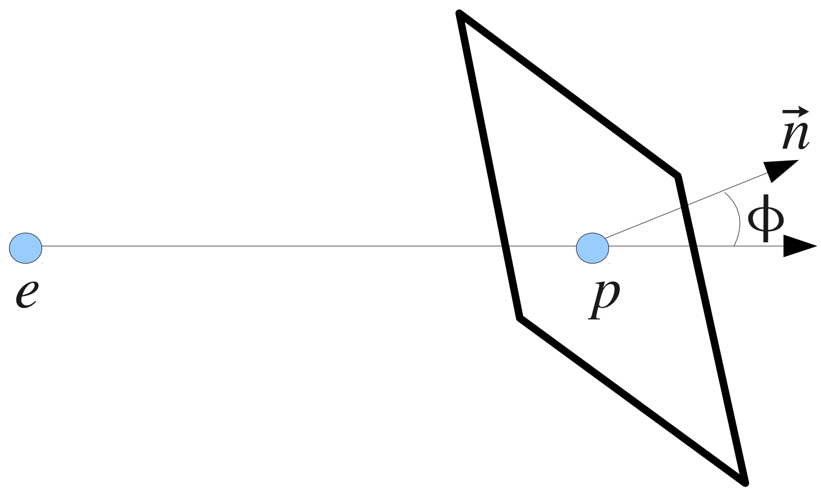
- Types of visibilty computation we have seen:
  - Clipping – 2D and 3D
  - View-frustum clipping/culling
  - Backface culling



http://geometricalgebra.org

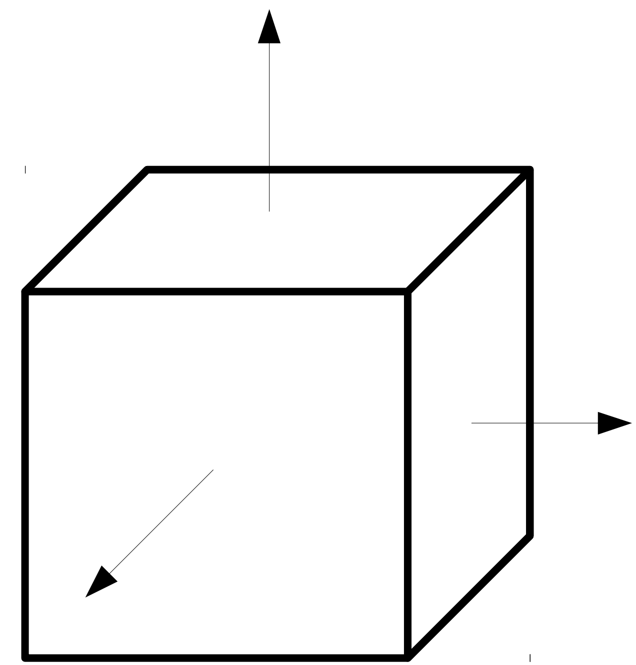
Visibility

- Backface Culling



$(p-e).\vec{n}>0$ Cull

$(p-e).\vec{n}\leq 0$ Do Not Cull (may be visible)

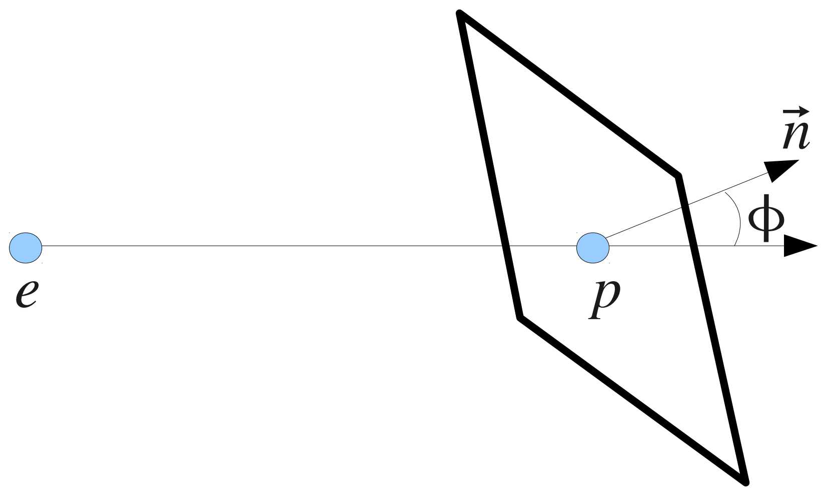


Simple Idea:

Discard surface patches that face away from the camera.

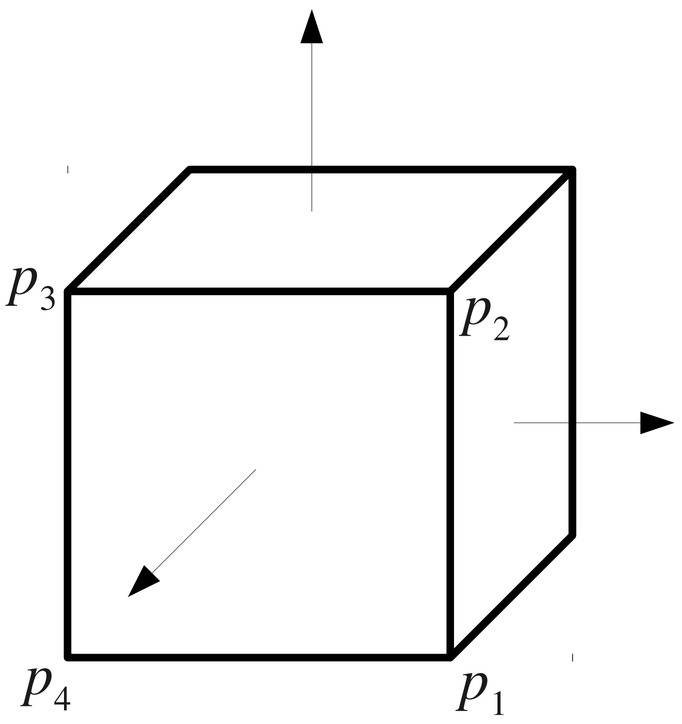
Visibility

- Backface Culling



If $p_{1,} p_{2,} p_{3,} p_4$ are the patch vertices in CCW order seen from outside then the outward facing normal is given by:

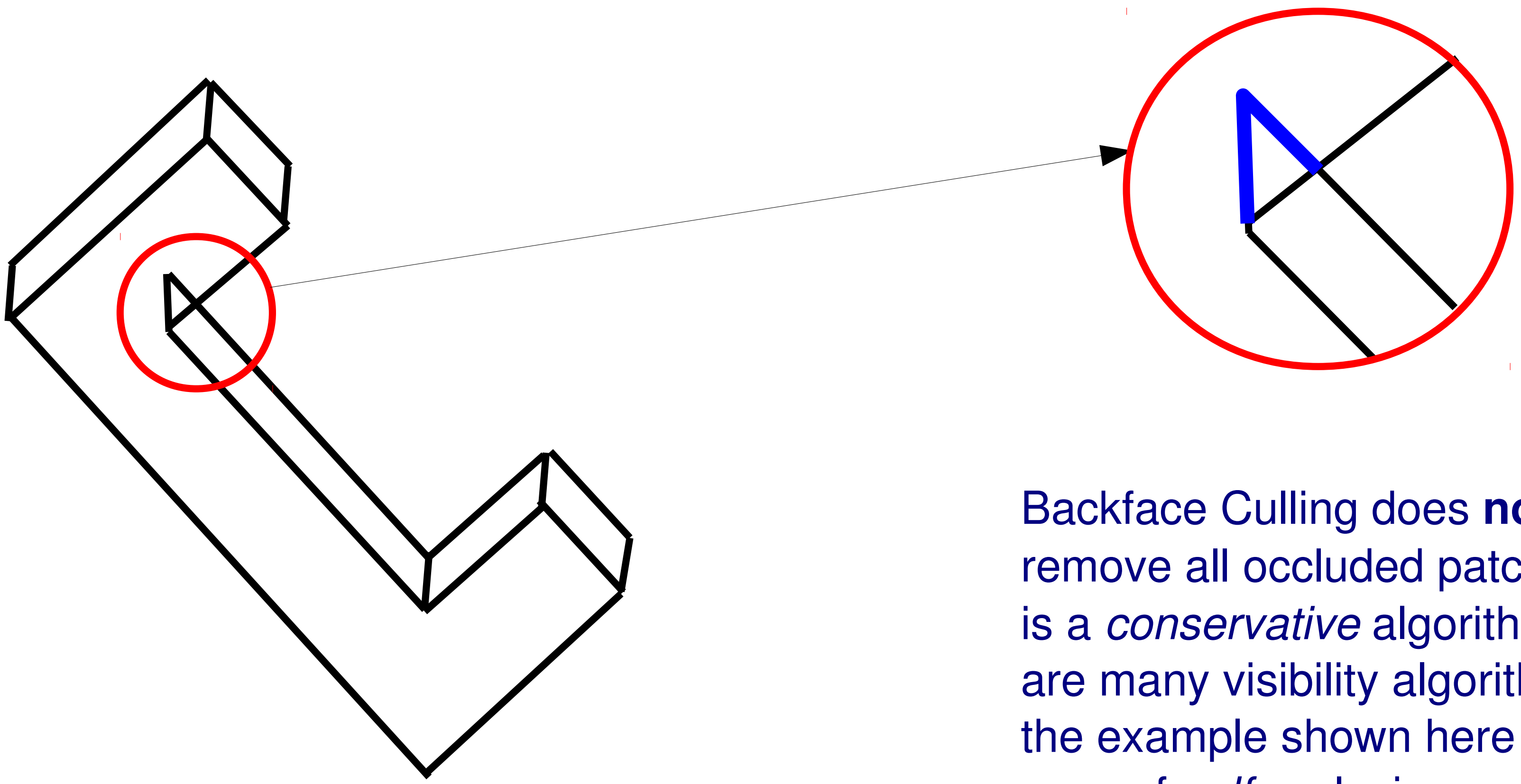
$$\vec{n} = (p_2 - p_1) \times (p_3 - p_1)$$



Compute the outward normals and do Backface culling in the WCS.

Visibility

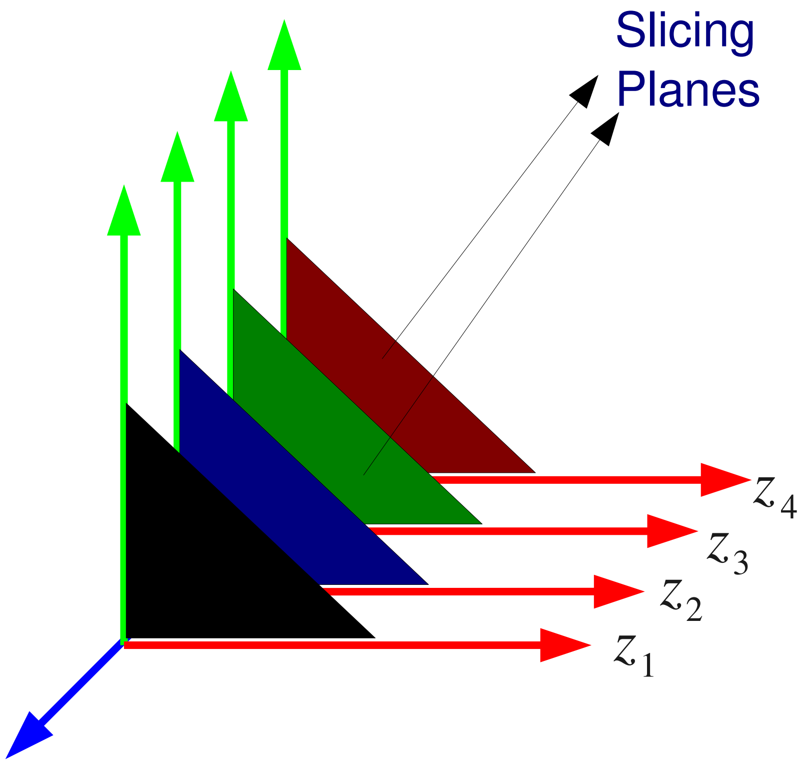
- Backface Culling is not enough

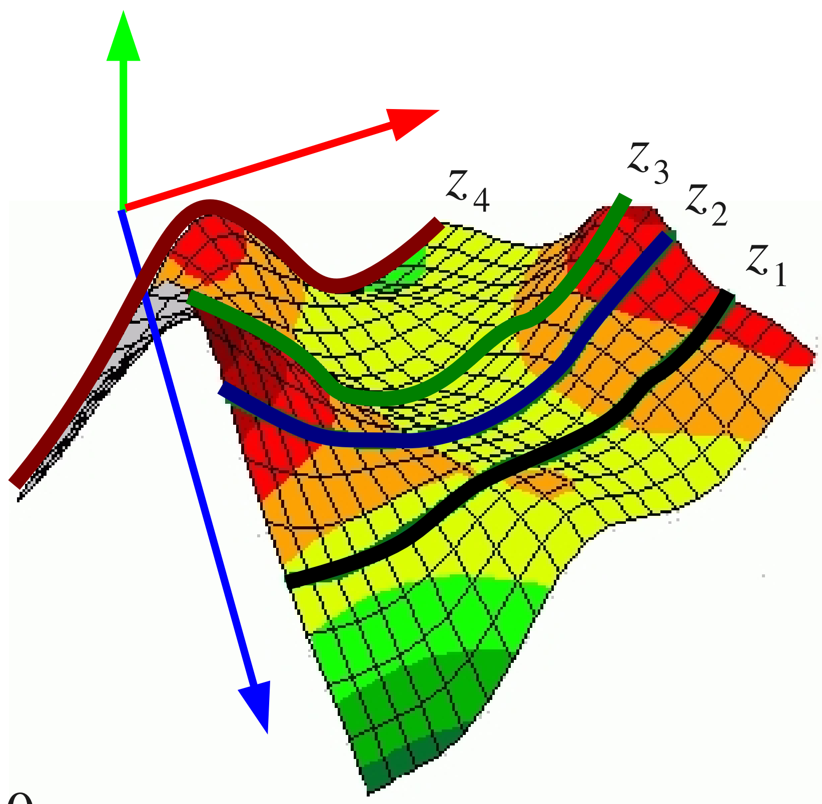


Backface Culling does **not** remove all occluded patches (it is a *conservative* algorithm as are many visibility algorithms) – the example shown here is a case of *self* occlusion.

Visibility

- Floating Horizon Algorithm

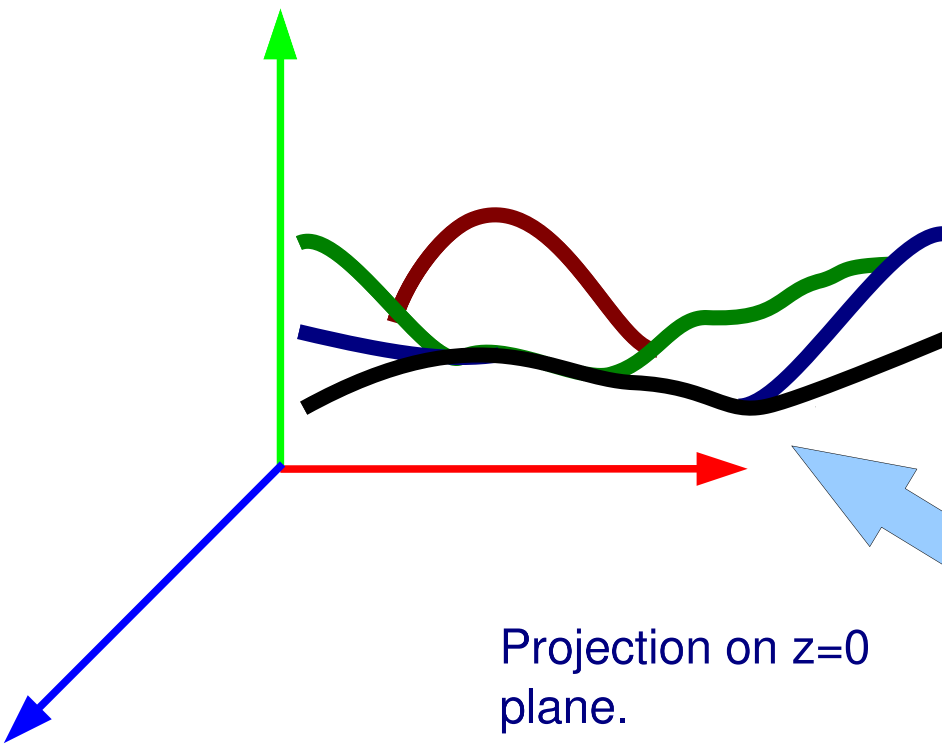


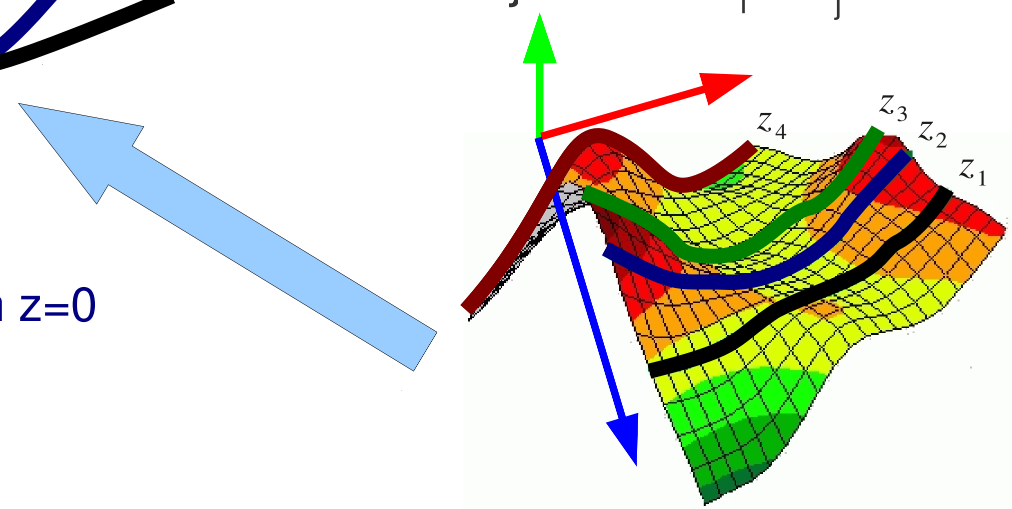


Given a surface defined by $f(x, y, z) = 0$
We can sample it at many 2D cutting planes,
yielding a set of curves of the form $y = f(x, z_i)$

Visibility

- Floating Horizon Algorithm
  - For each slicing plane i, with $z=z_i$
    - Compute y_i for any x_i on the curve.
    - The point (x_i,y_i) is visible if $y_i > y_j$ for all $j < i$ and $x_i = x_j$





Visibility

- Z-Buffer and Scan Conversion

- Initialize the z-buffer

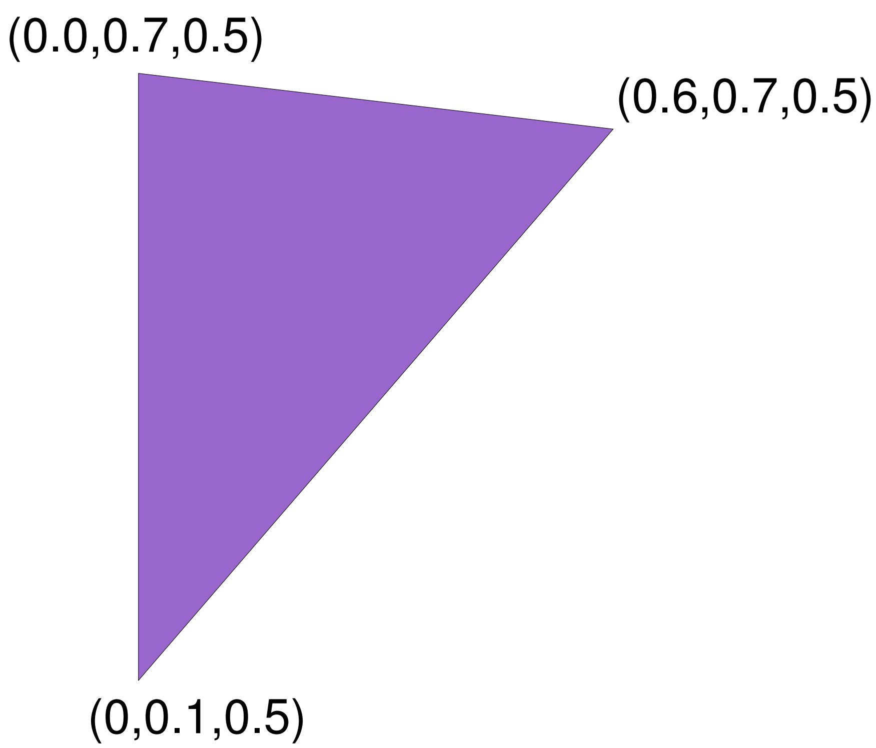
  to the max Z value.

- glClear, glDepthRange

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Visibility

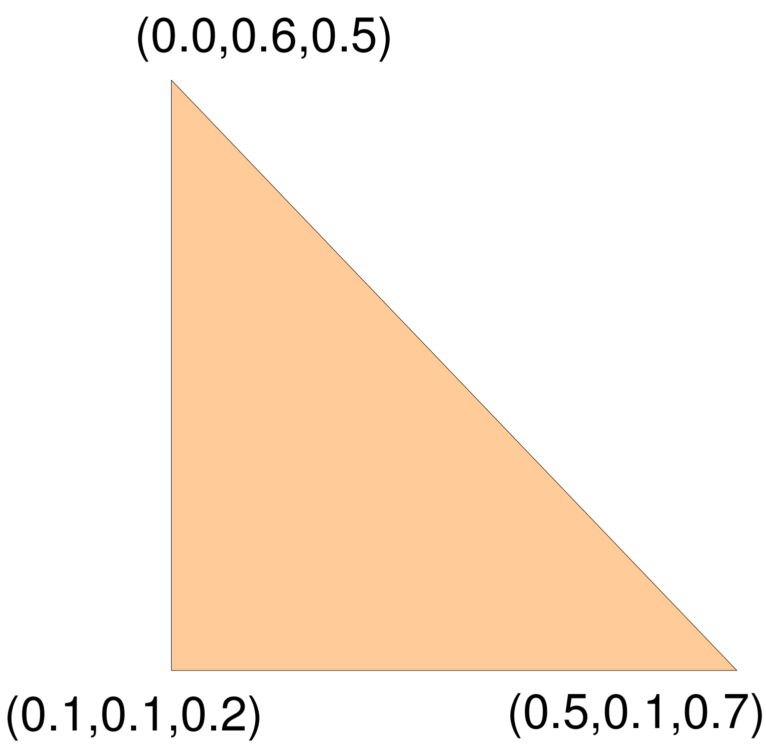
- Z-Buffer and Scan Conversion



1	1	1	1	1	1	1	1
1	0.5	0.5	1	1	1	1	1
1	0.5	0.5	0.5	0.5	0.5	1	1
1	0.5	0.5	0.5	0.5	1	1	1
1	0.5	0.5	0.5	1	1	1	1
1	0.5	0.5	1	1	1	1	1
1	0.5	0.5	1	1	1	1	1
1	0.5	1	1	1	1	1	1

Visibility

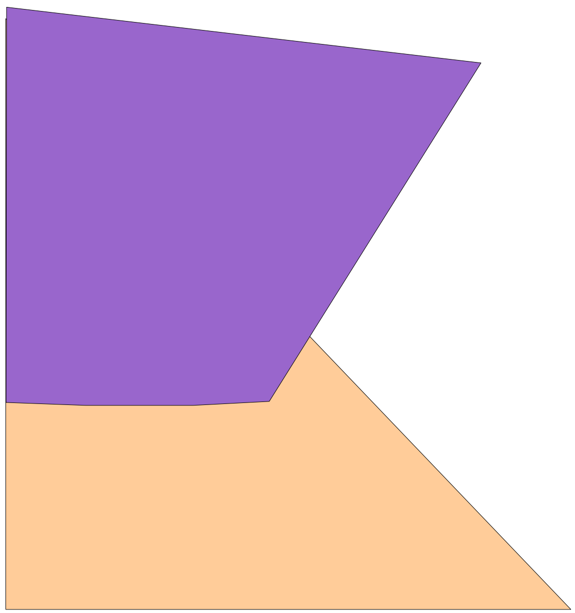
- Z-Buffer and Scan Conversion



1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	0.5	0.5	1	1	1	1	1
1	0.5	0.5	1	1	1	1	1
1	0.5	0.5	0.5	1	1	1	1
1	0.4	0.5	0.6	0.7	1	1	1
1	0.3	0.4	0.5	0.6	0.7	1	1
1	0.2	0.3	0.4	0.5	0.6	0.7	1

Visibility

- Z-Buffer and Scan Conversion



1	1	1	1	1	1	1	1
1	0.5	0.5	1	1	1	1	1
1	0.5	0.5	0.5	0.5	0.5	1	1
1	0.5	0.5	0.5	0.5	1	1	1
1	0.5	0.5	0.5	1	1	1	1
1	0.4	0.5	0.6	0.7	1	1	1
1	0.3	0.4	0.5	0.6	0.7	1	1
1	0.2	0.3	0.4	0.5	0.6	0.7	1

Note that almost everywhere the result is independent of the order of drawing these polygons.
Except at the pixels where the depth may be the same (this is very unlikely).

Visibility

- Z-Buffer Algorithm

```
Initialize
    zbuf[i, j]=MAX_DEPTH
    cbuf[i, j]=BACKGROUND_COLOR
for (each scan converted polygon)
{
    Find pseudodepth, z, of polygon at pixel (x, y) with color c
    If (z < zbuf[i, j]) { zbuf[i, j] = z; cbuf[i, j] = c;}
}
```

Visibility

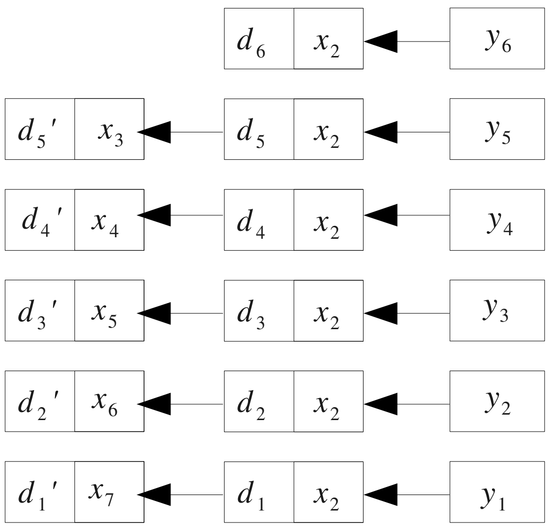
- Z-Buffer Algorithm
- Advantages
  - Simple, Accurate (modulo non-linear z-mapping).
  - Independent of order of drawing polygons.
- Disadvantages
  - Memory (not an issue these days).
  - Wasted computation when over-writing distant points
- Complexity
  - Time: O(nm . k) – nxm pixels, k polygons
  - Space: O(nm . b) – nxm pixels, b bytes precision per pixel

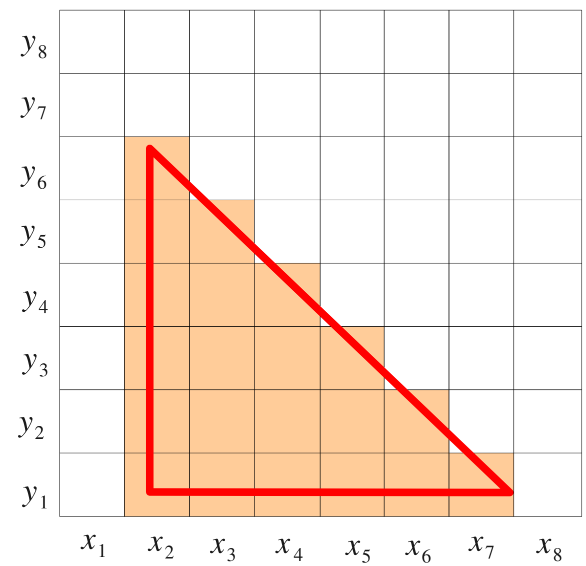
Visibility

- Z-Buffer Algorithm and Scan Conversion

- Construct the active edge list AEL for every scanline.
- Interpolate the pseudodepth for each active edge.

for each edge $[(x_i, y_i, d_i)$ and $(x_j, y_j, d_j)]$ with $y_i < y_j$
{
  $x = x_i, d = d_i, \Delta x = \frac{(x_j - x_i)}{(y_j - y_i)}$ and $\Delta d = \frac{(d_j - d_i)}{(y_j - y_i)}$
  for $(y = y_i, y < y_j; y{+}{+})$
  {
    insert (x, d) into the AEL of scanline y
    such that it is sorted on the x values
    $x = x + \Delta x, d = d + \Delta d$
  }
}





Visibility

- Z-Buffer Algorithm and Scan Conversion

- Compute the active edge list.
- Interpolate the pseudodepth for each active edge.
- Now, for rendering a ΔABC:

$\text{cbuf}[i, j] = \text{BACKGROUND\_COLOR}$

$\text{zbuf}[i, j] = \text{MAX\_DEPTH}, \forall\, 0 \le i < N, 0 \le j < M$

$y_{min} = min(y_A, y_B, y_C) \quad y_{max} = max(y_A, y_B, y_C)$

for $(\, y = y_{min}; y \le y_{max}; y{+}{+})$

{

get (x_p, d_p) and (x_q, d_q) from the AEL with $x_p < x_q$

$\Delta d = \dfrac{(d_q - d_p)}{(x_q - x_p)};$

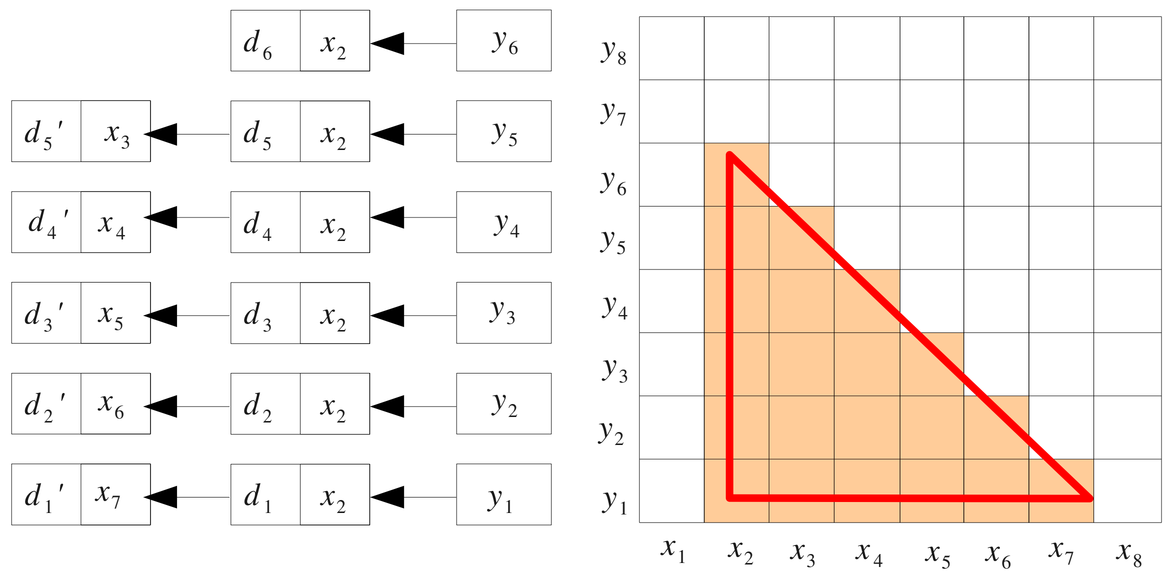
for $(\, x = x_p, d = d_p; x \le x_q; x{+}{+}, d = d + \Delta d\,)$

{

if $(\, d < \text{zbuf}[x, y])$ { $\text{zbuf}[x, y] = d, \text{cbuf}[x, y] = c$ }

}

}

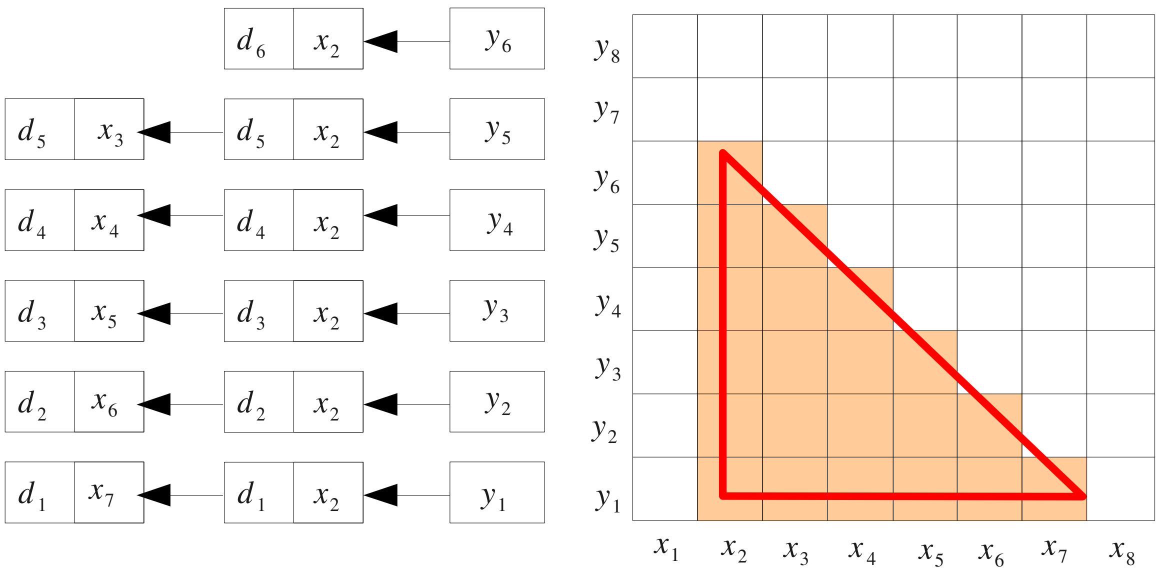


- Note: The color c at a pixel is also interpolated along the scanline like the pseudodepth is

Visibility

- Z-Buffer Algorithm and Scan Conversion

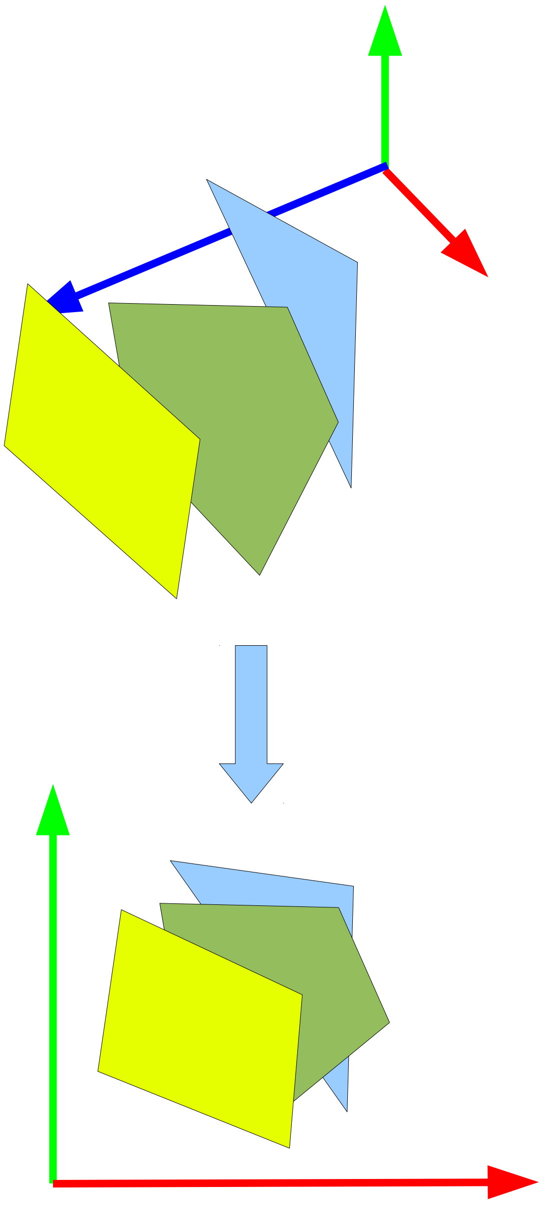
- Compute the active edge list.
- Interpolate the pseudodepth for each active edge.



Visibility

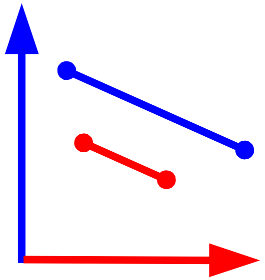
- Painter's Algorithm

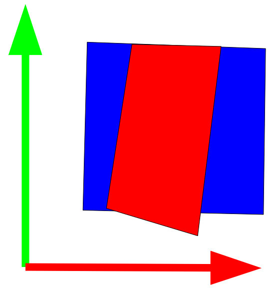
- Sort polygon's in increasing order of depth.
- Draw the sorted list of polygons from back to front, i.e., from greatest depth to lesser depths.
- What happens when a polygon has vertices at different depths?



Visibility

- Painter's Algorithm



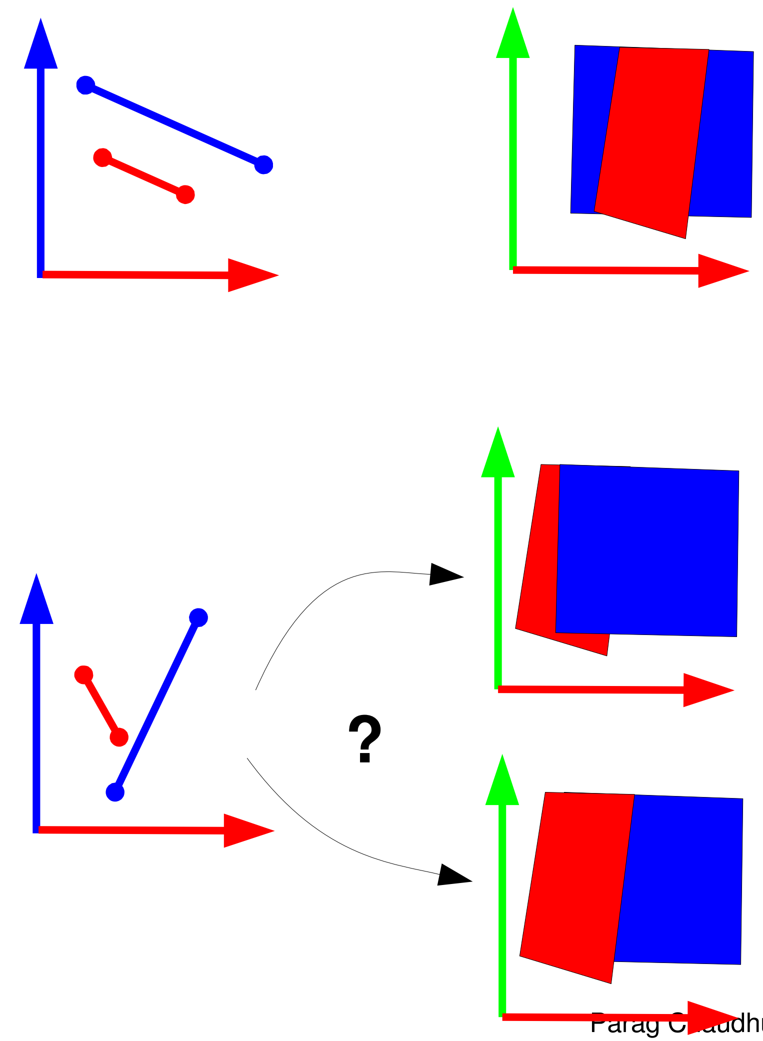


- Sort polygon's in increasing order of depth.
- Draw the sorted list of polygons from back to front, i.e., from greatest depth to lesser depths.
- What happens when a polygon has vertices at different depths?
- Sort according to depth of farthest vertex.

Visibility

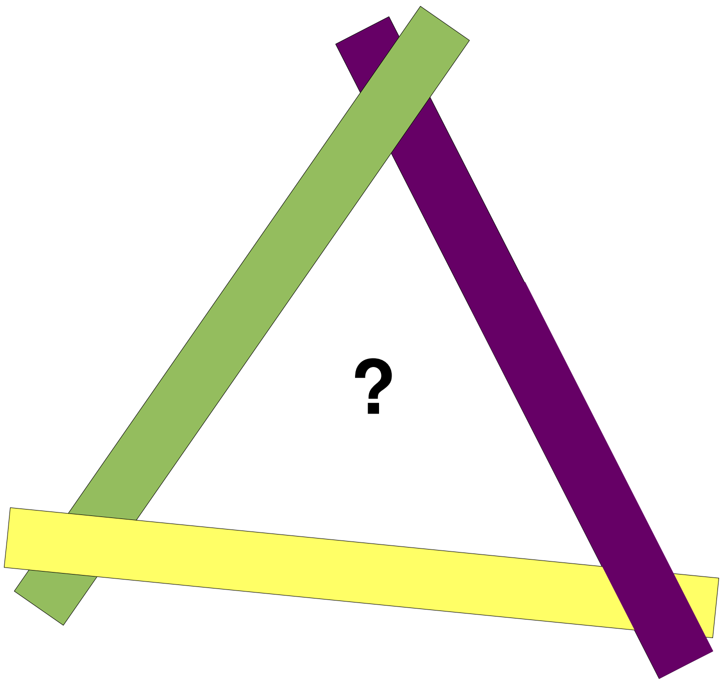
- Painter's Algorithm

  - Sort polygon's in increasing order of depth.
  - Draw the sorted list of polygons from back to front, i.e., from greatest depth to lesser depths.
  - What happens when a polygon has vertices at different depths?
  - Sort according to depth of farthest vertex.
  - Does it always work?
  - How often do we sort?



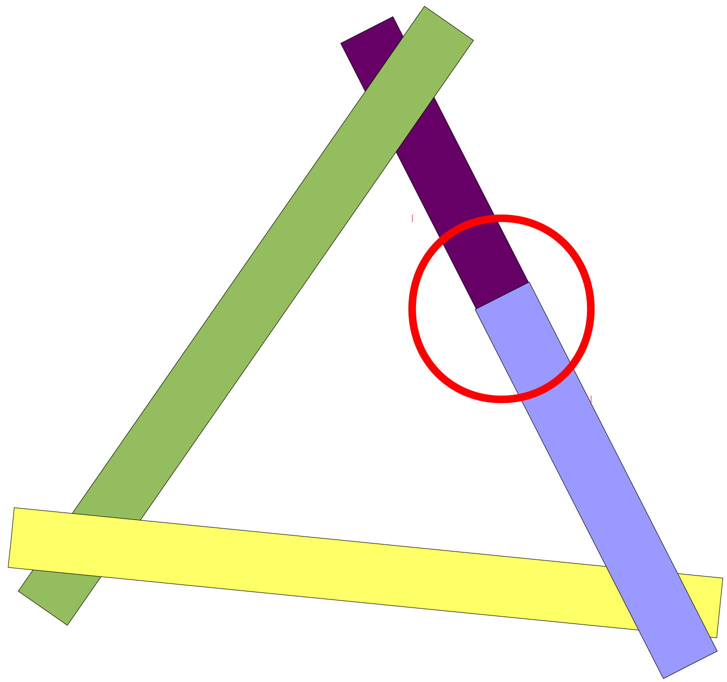
Visibility

- Painter's Algorithm



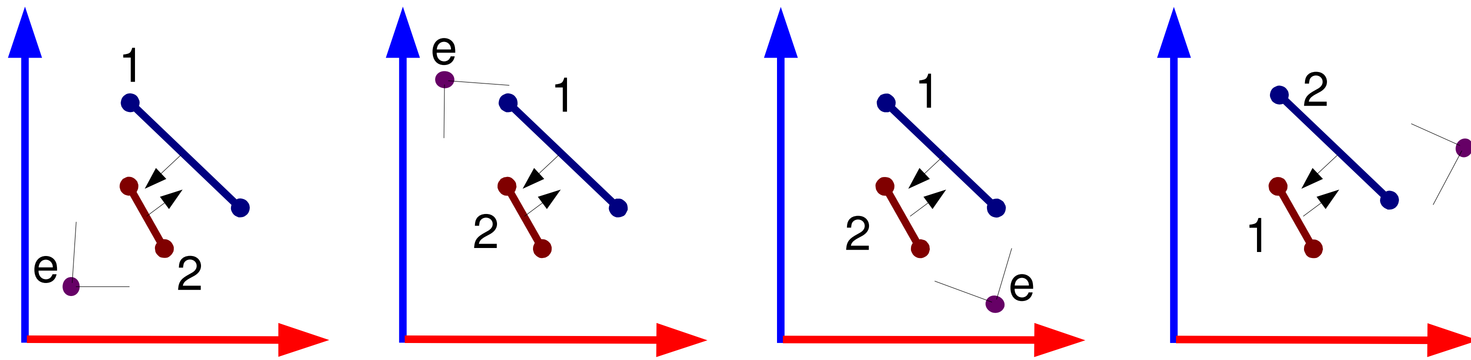
Visibility

- Painter's Algorithm



Visibility

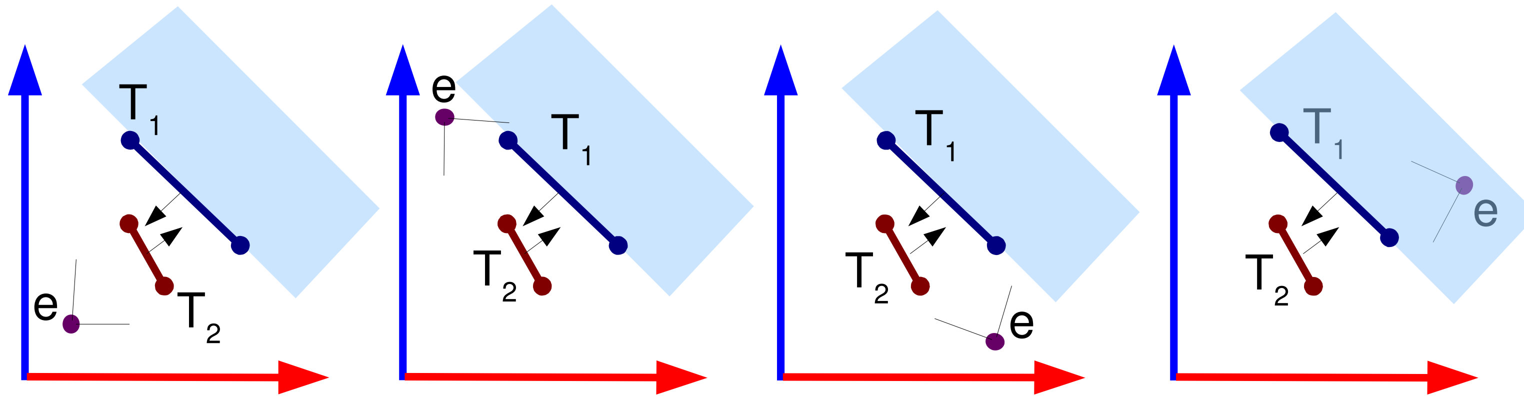
- Binary Space Partitioning (BSP) Trees



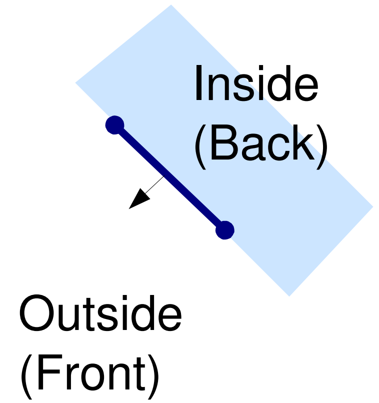
- Observe the correct order of drawing polygons as the eye moves

Visibility

- Binary Space Partitioning (BSP) Trees

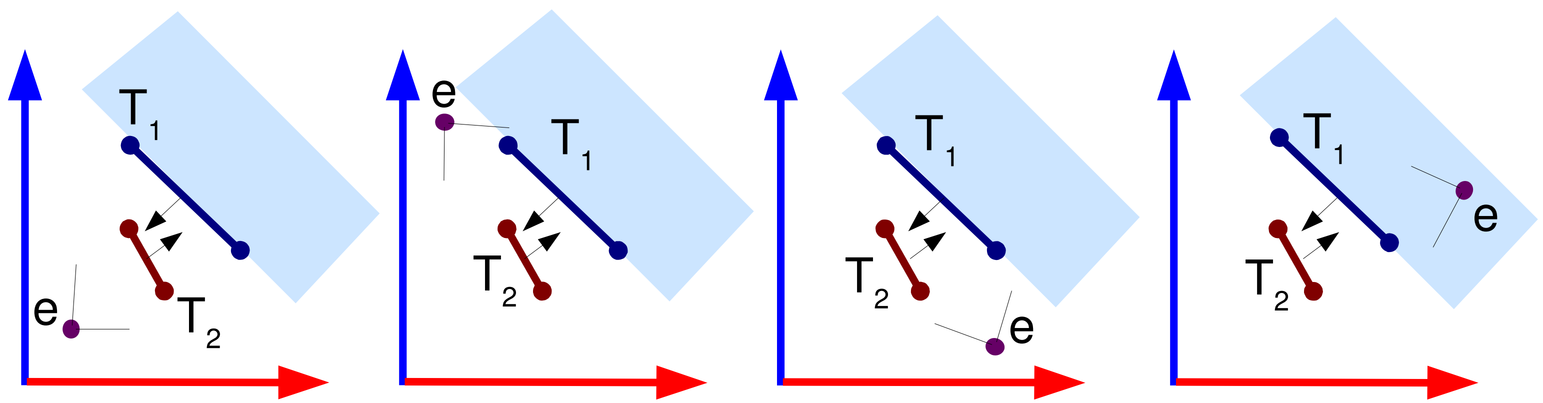


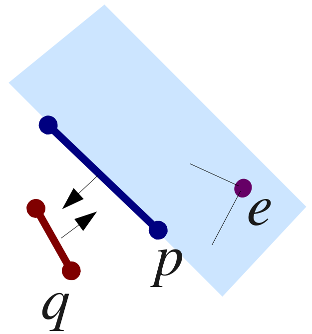
- If e and T_2 are on the same side of T_1
  - Draw T_1 and then draw T_2
- If e and T_2 are on different sides of T_1
  - Draw T_2 and then draw T_1



Visibility

- Binary Space Partitioning (BSP) Trees

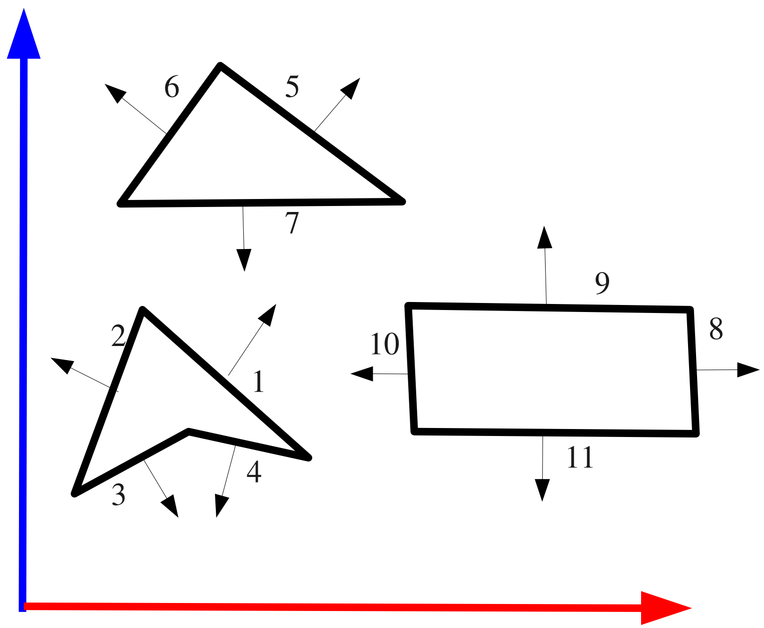




- If the implicit equation of the plane containing T_1 is given by: $f(r)=(r-p).n$
- If $f(q).f(e)>0$ then draw T_1 and then draw T_2
- If $f(q).f(e)<0$ then draw T_2 and then draw T_1

Visibility

- BSP Tree is an efficient data structure for quickly determining the inside/outside relation between polygons and the camera position.



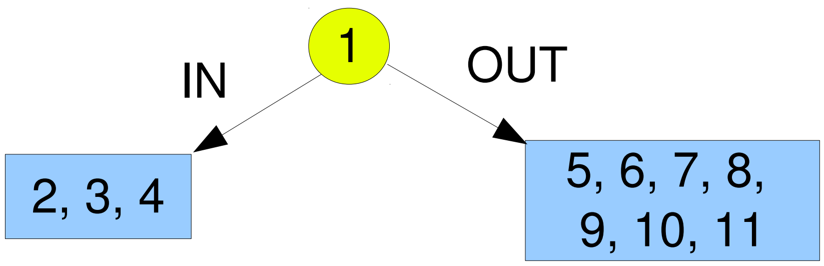
- Two Phases
  - Preprocessing: BSP Tree construction

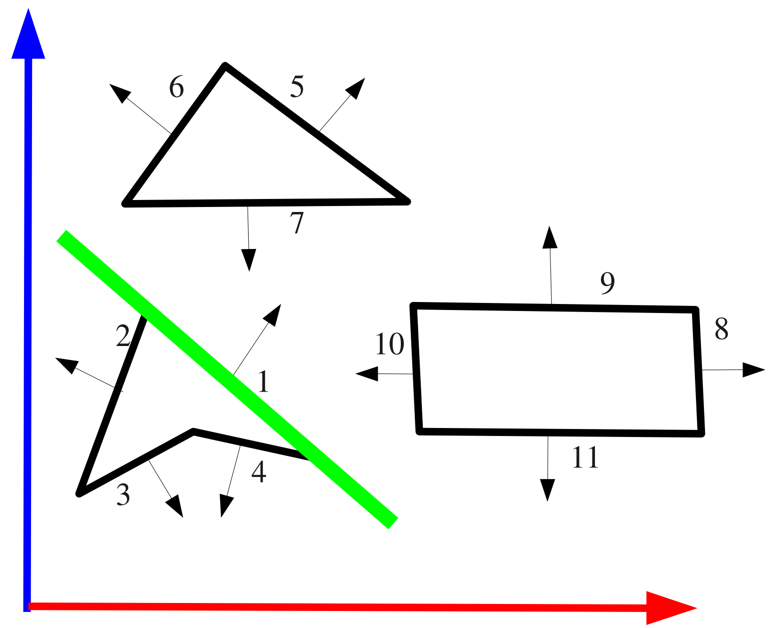
    (done once for a given scene)
  - Rendering: BSP Tree traversal

    (done whenever the eye position changes)

Visibility

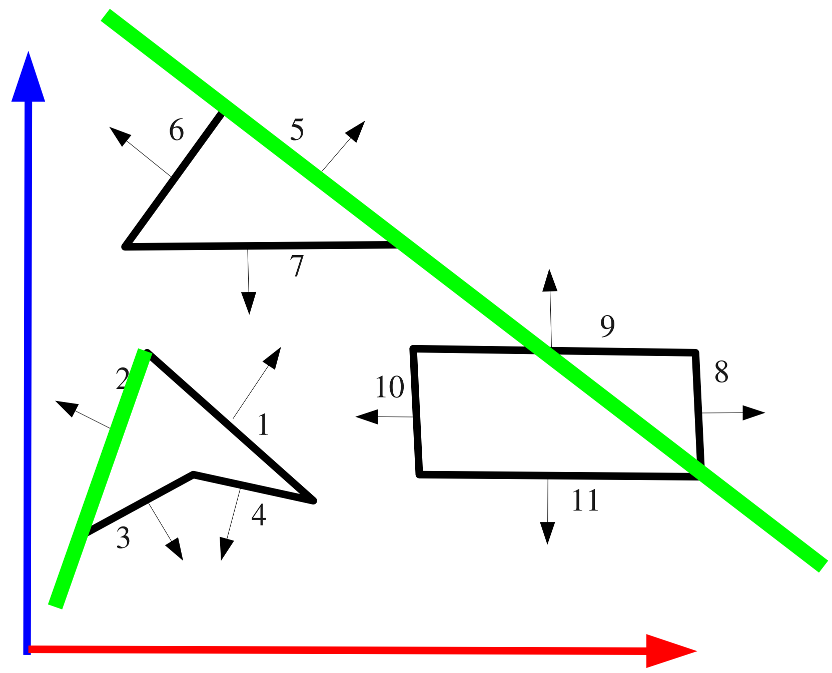
- BSP Tree construction

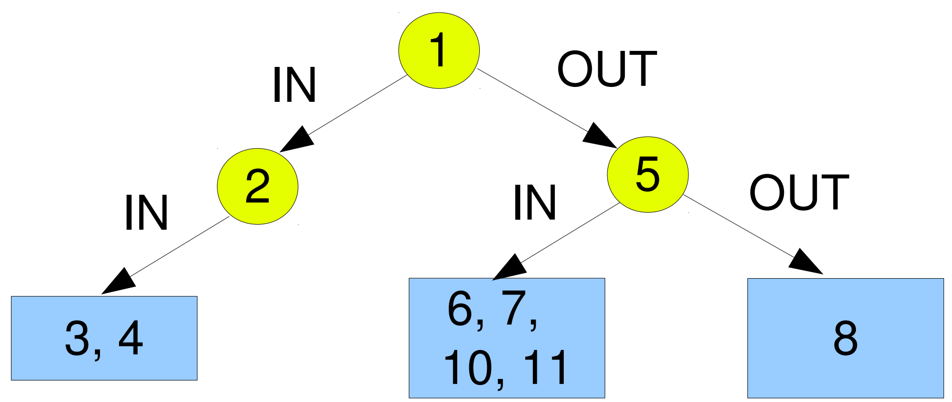




Visibility

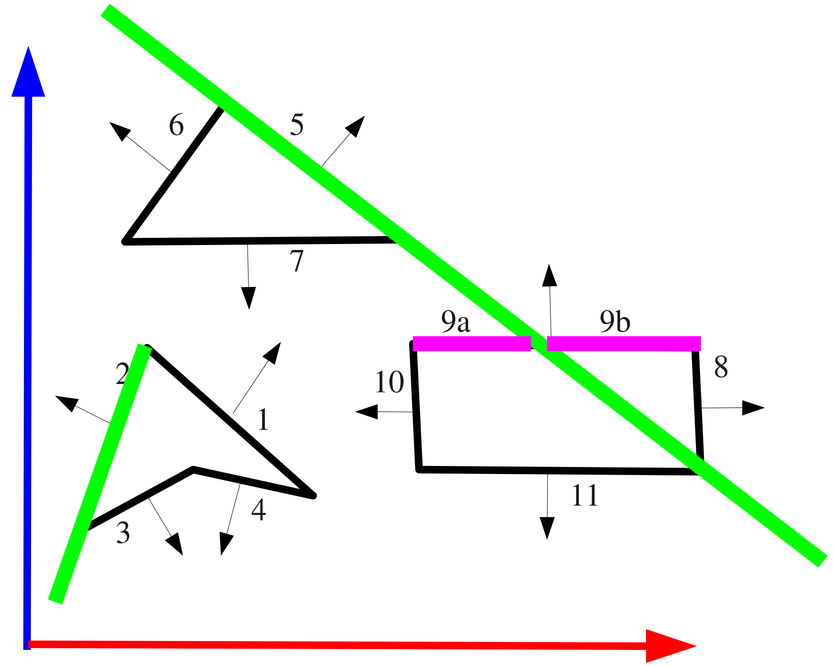
- BSP Tree construction

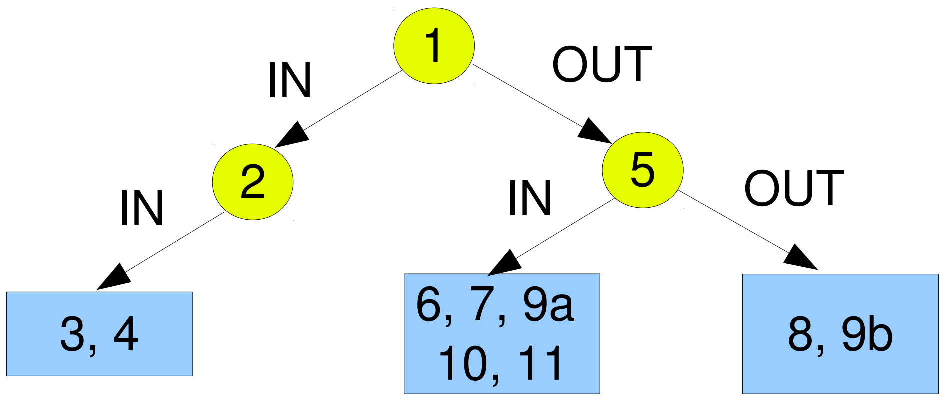




Visibility

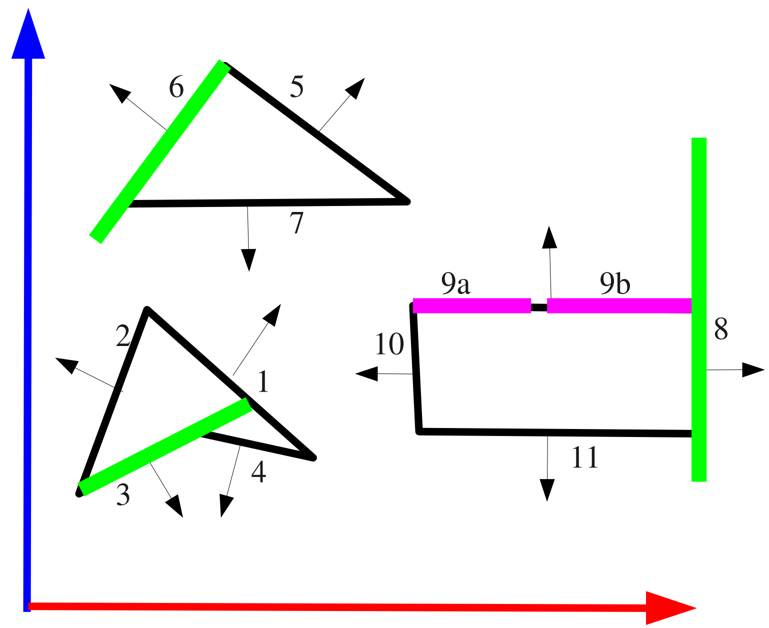
- BSP Tree construction

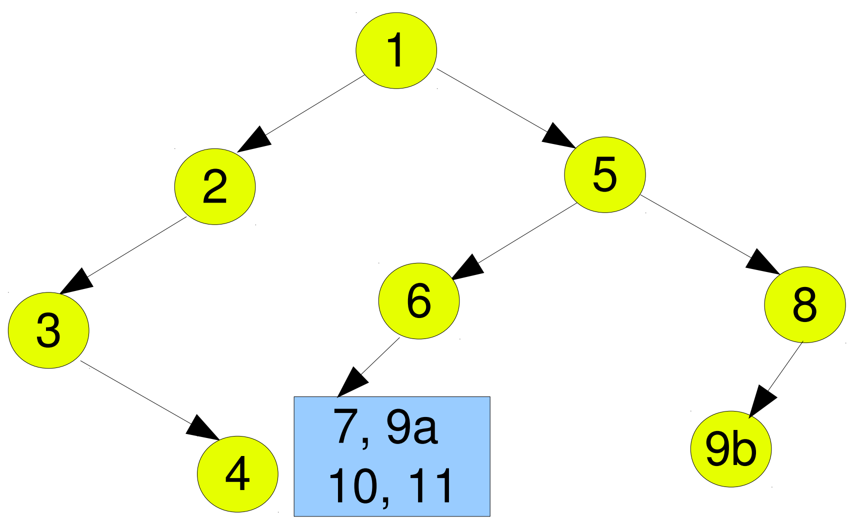




Visibility

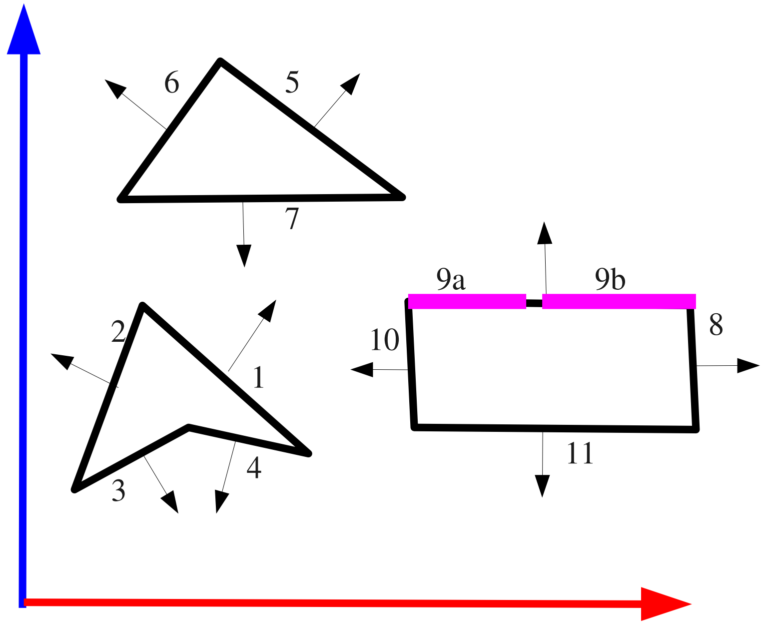
- BSP Tree construction

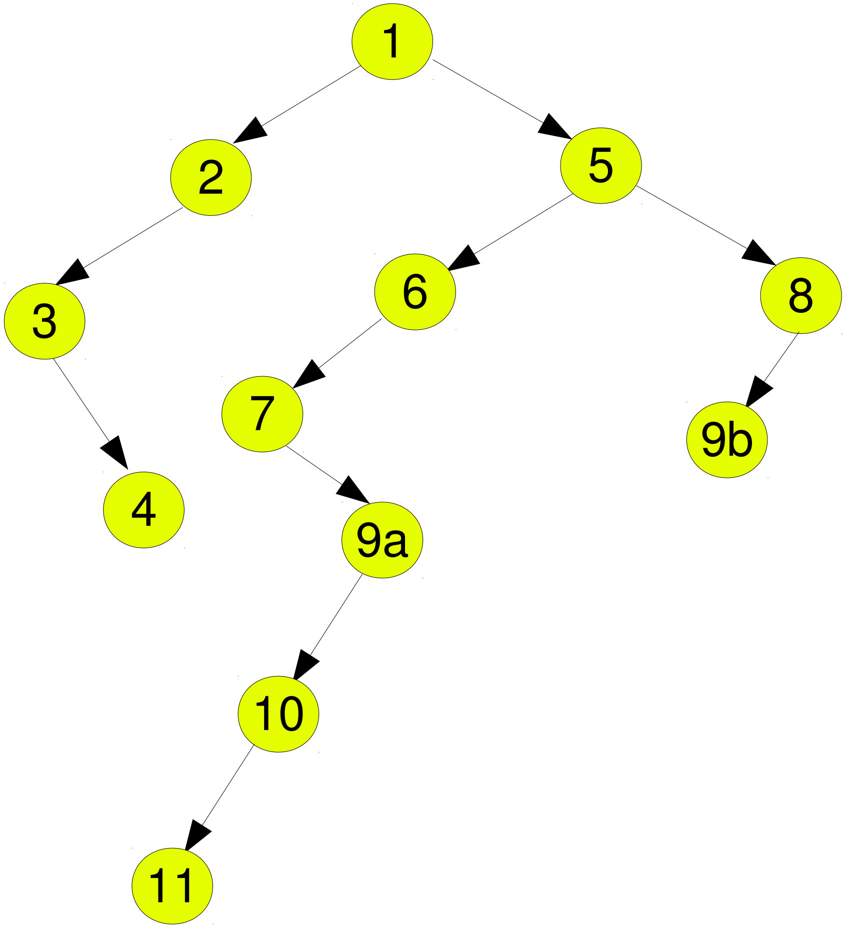




Visibility

- BSP Tree construction

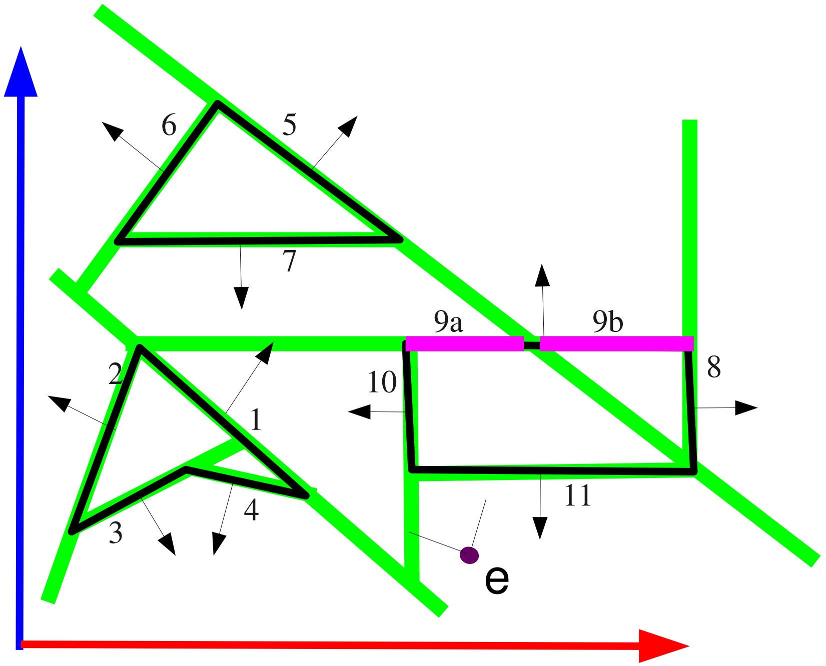


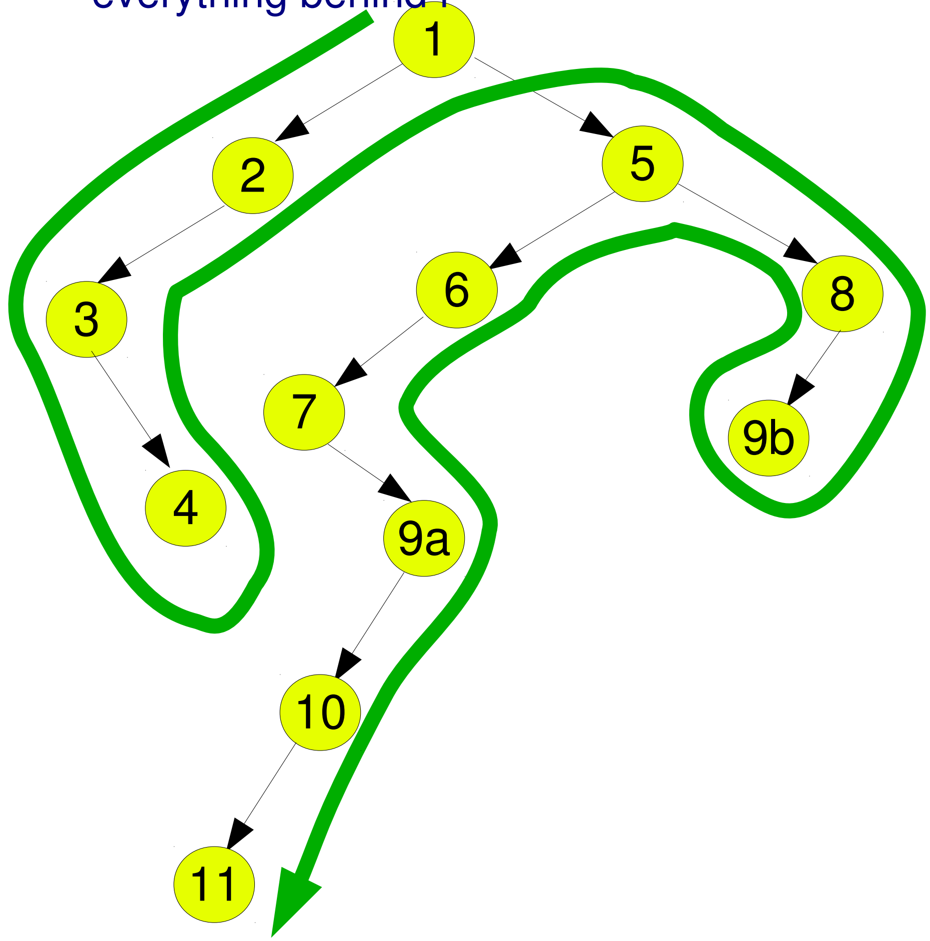


Visibility

- BSP Tree traversal

- If e is outside (or in front of) a face i
  - Draw everything behind i,Draw i, Draw everything in front of i
- If e is inside a (or behind) face i
  - Draw everything in front of i,Draw i, Draw everything behind i



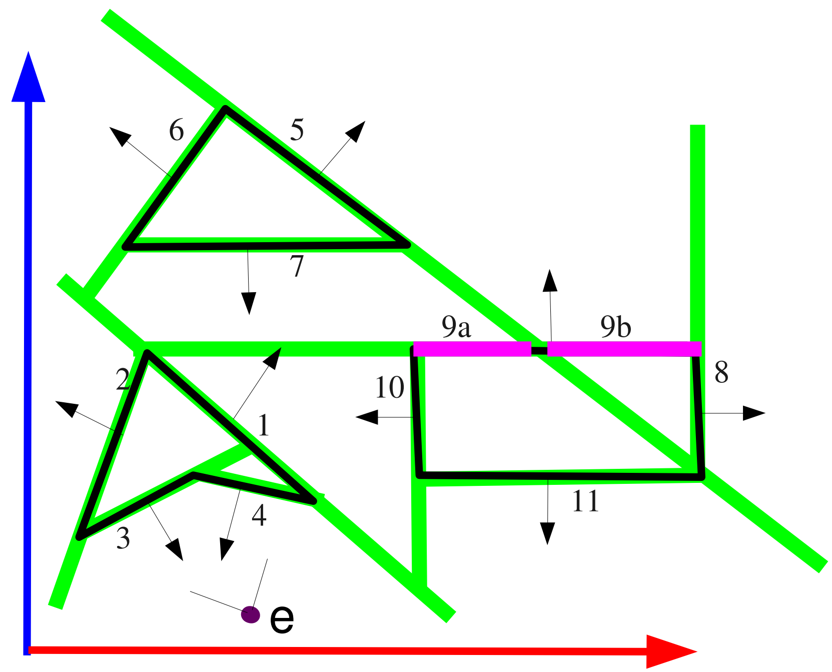


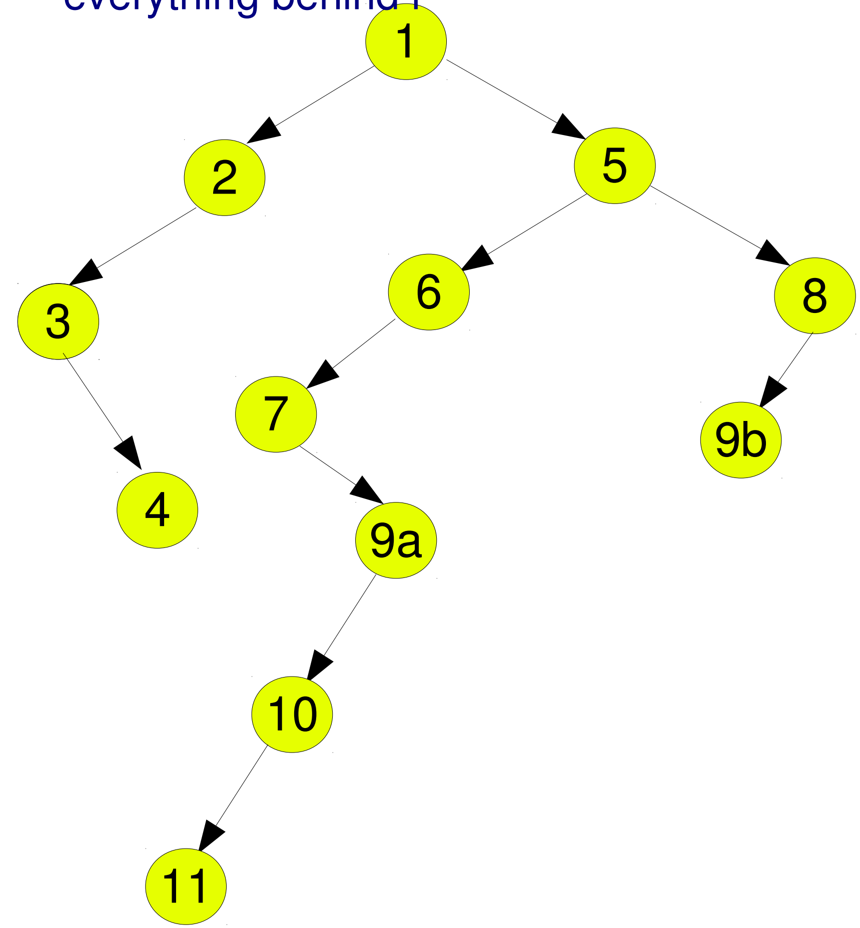
2, 3, 4, 1, 8, 9b, 5, 6, 7, 9a, 10, 11

Visibility

- BSP Tree traversal

- If e is outside (or in front of) a face i
  - Draw everything behind i,Draw i, Draw everything in front of i
- If e is inside a (or behind) face i
  - Draw everything in front of i,Draw i, Draw everything behind i

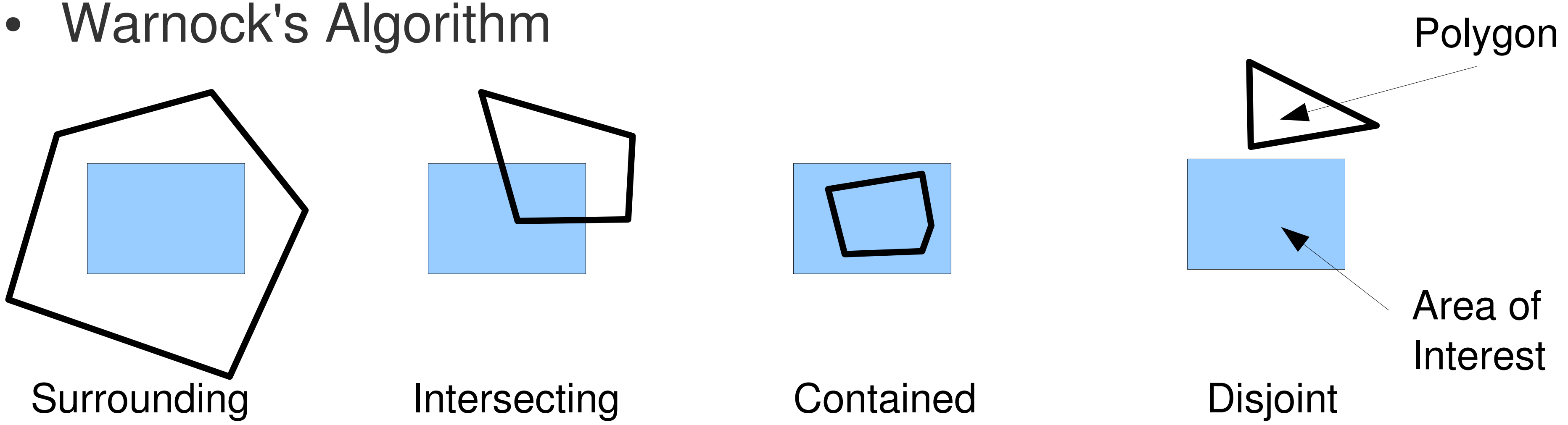




What is the traversal order now?

Visibility

- Warnock's Algorithm



Algorithm:

Consider the projected image area

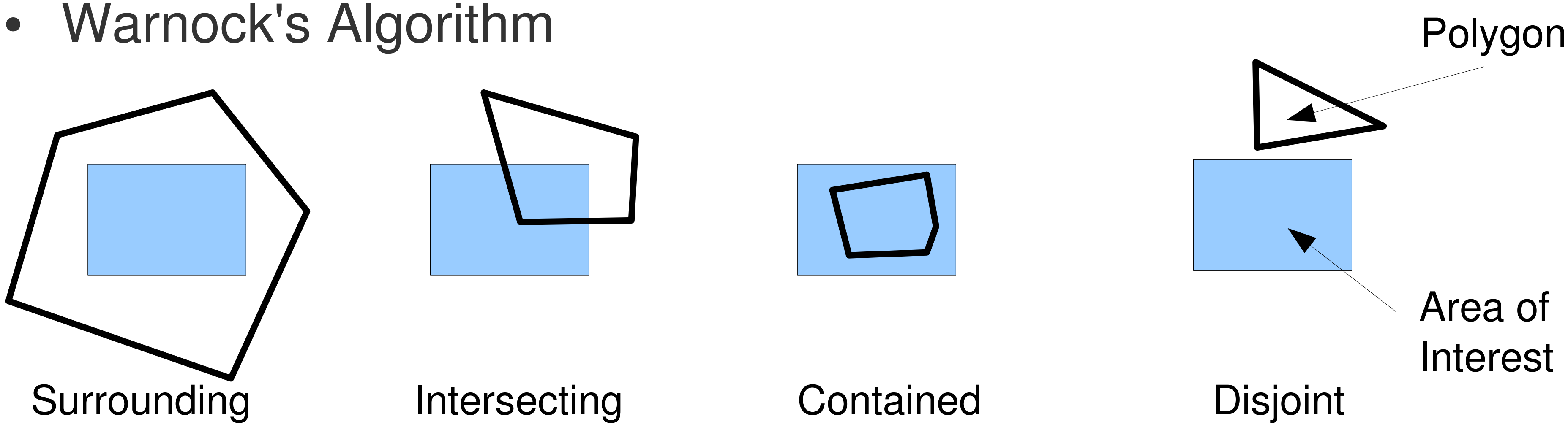
If it is easy to decide which polygons are visible in the area

then display

else subdivide the area and recurse with each subdivided area

Visibility

- Warnock's Algorithm

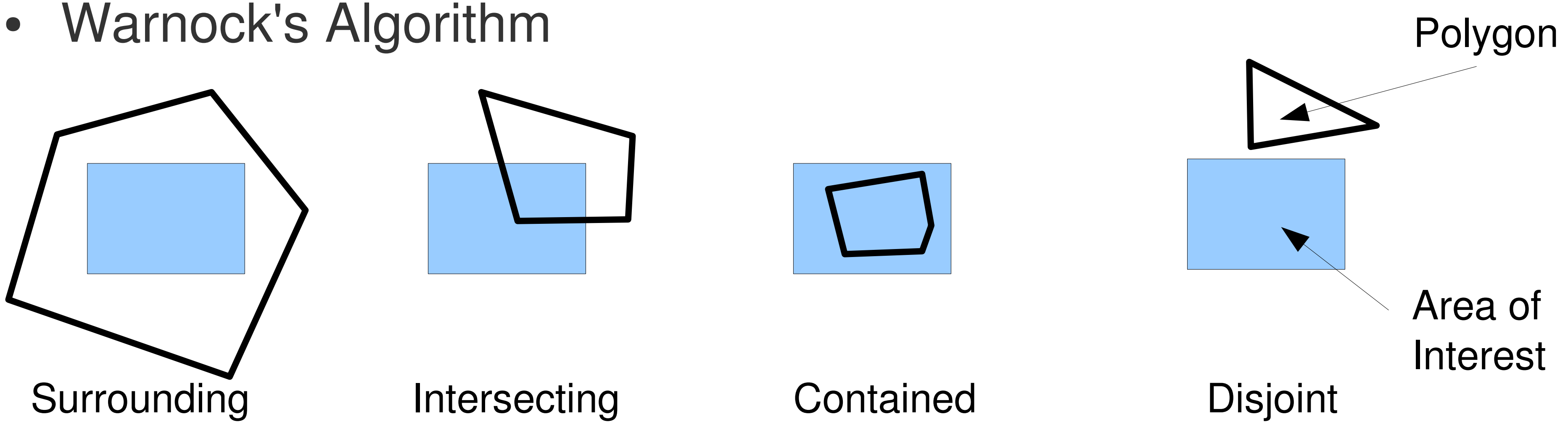


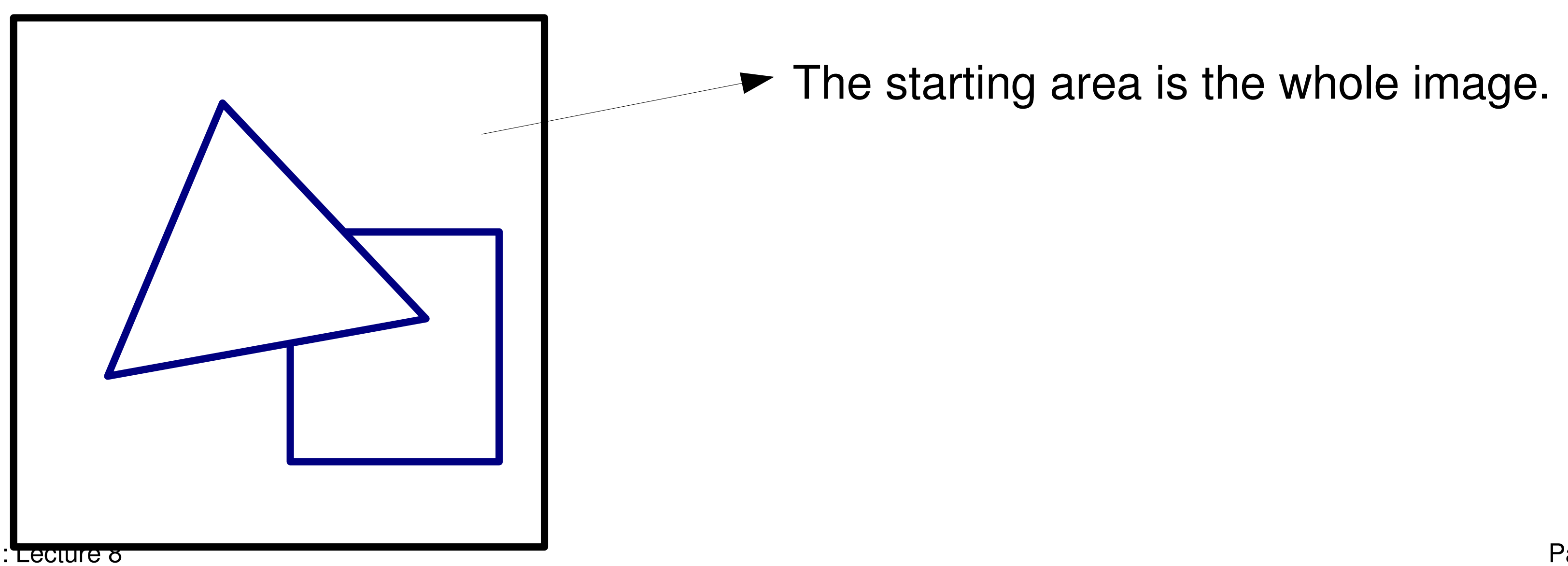
No subdivision for an area is required if

- All the polygons are disjoint with it : fill background color in the area.
- Only one intersecting or only one contained polygon: The area is filled first by background color, then the polygon part contained in the area.
- Only one surrounding polygon (no contained and intersecting polygons): The area is filled with the color of the surrounding polygon.
- More than one polygon is intersecting, contained in, or surrounding the area, with surrounding polygon in front: Fill the area with the color of the surrounding polygon.

Visibility

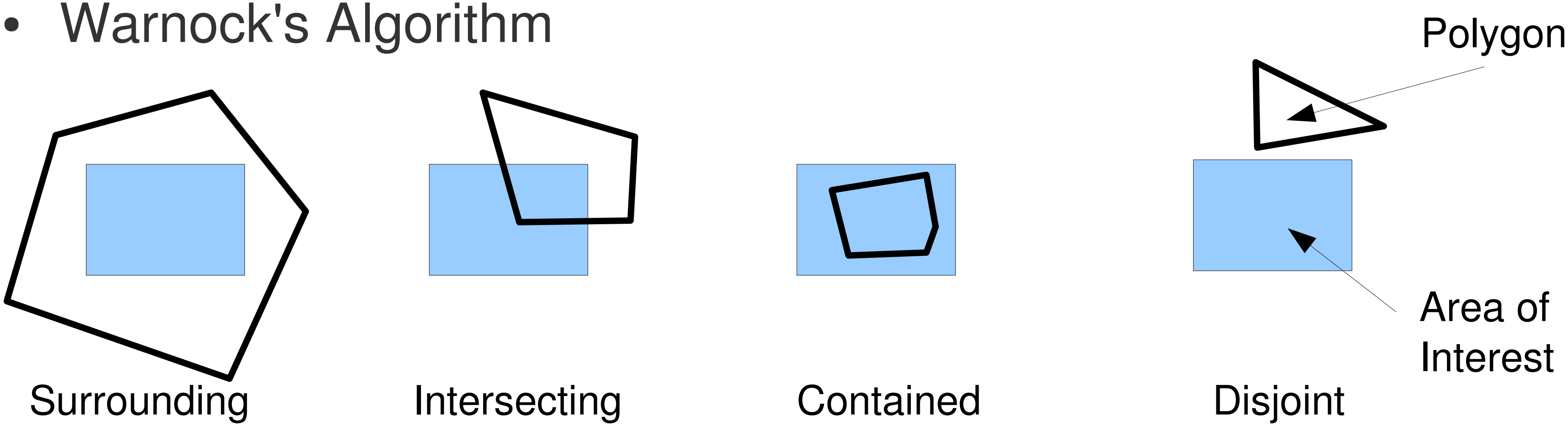
- Warnock's Algorithm

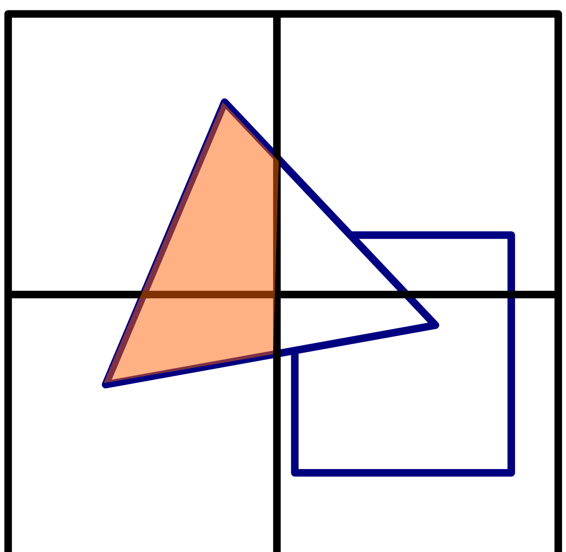




Visibility

- Warnock's Algorithm

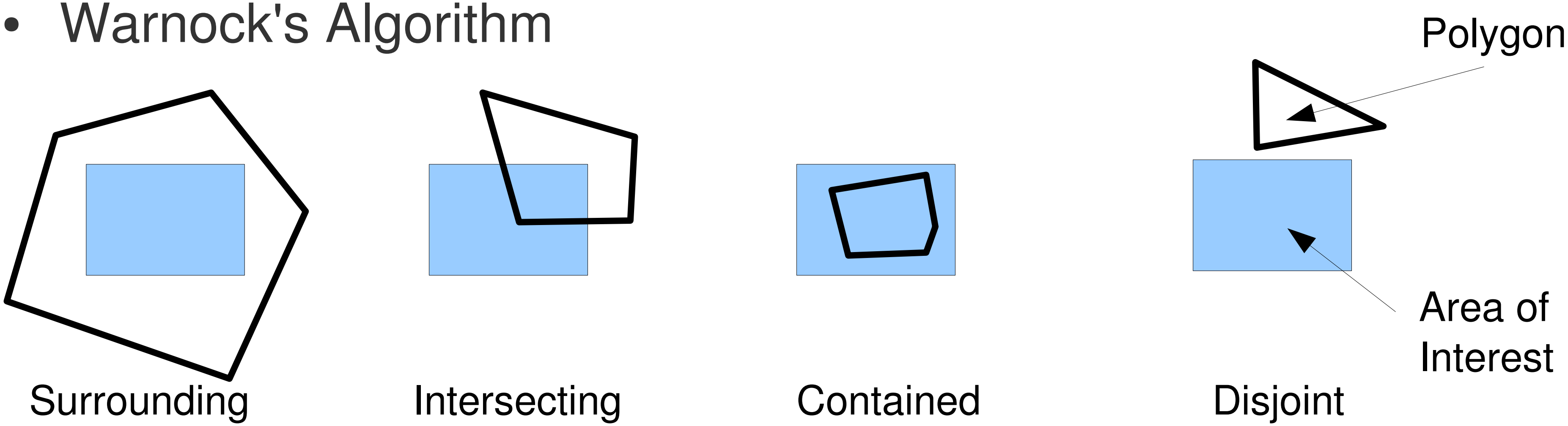


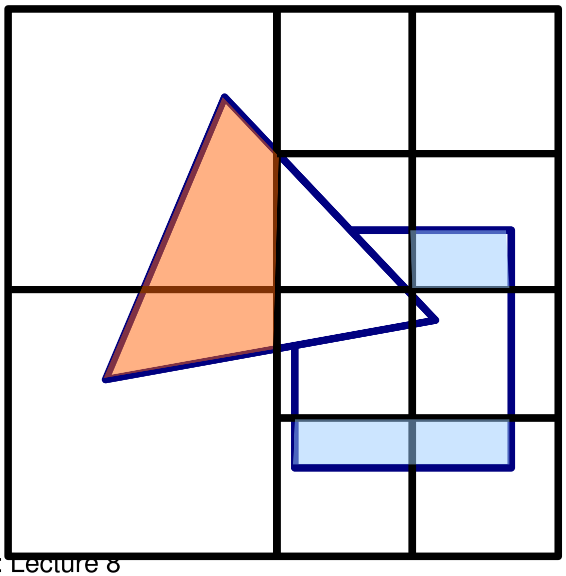


Test in the area and subdivide.

Visibility

- Warnock's Algorithm

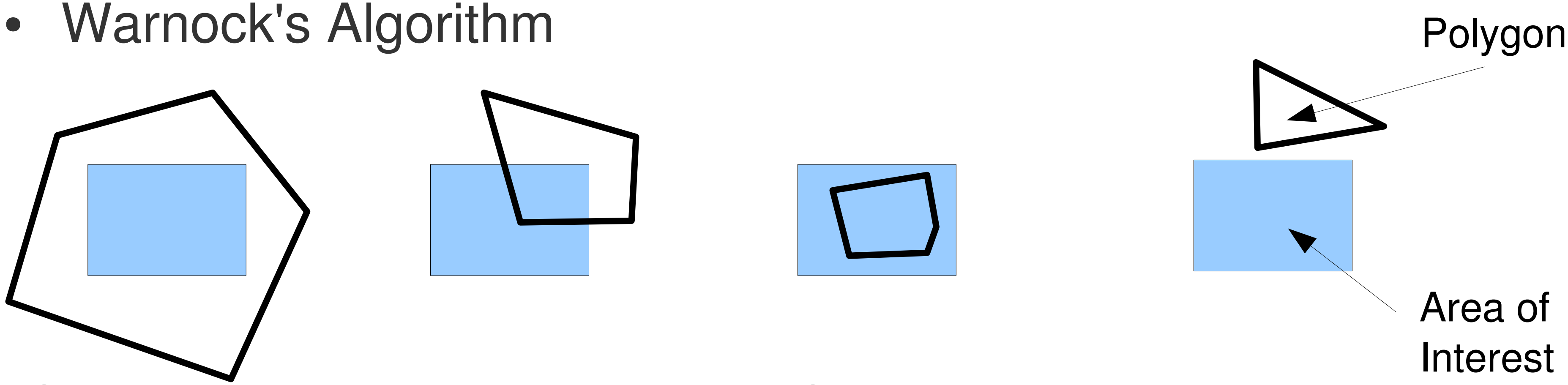




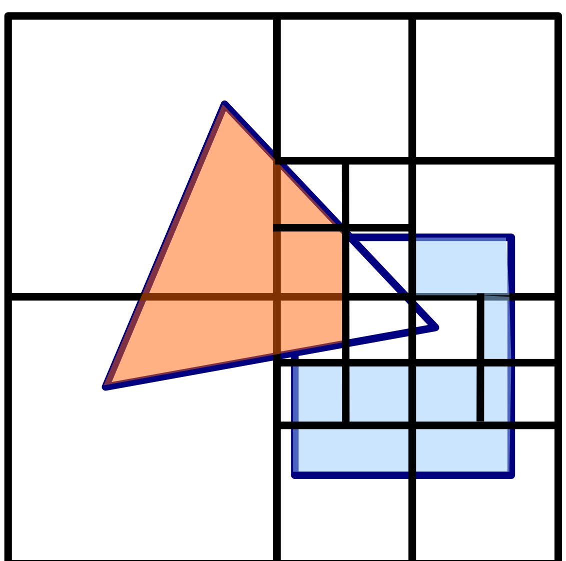
Test in the area and subdivide.

Visibility

- Warnock's Algorithm



Surrounding Intersecting Contained Disjoint



In the worst case you can end up subdividing upto pixel level.