



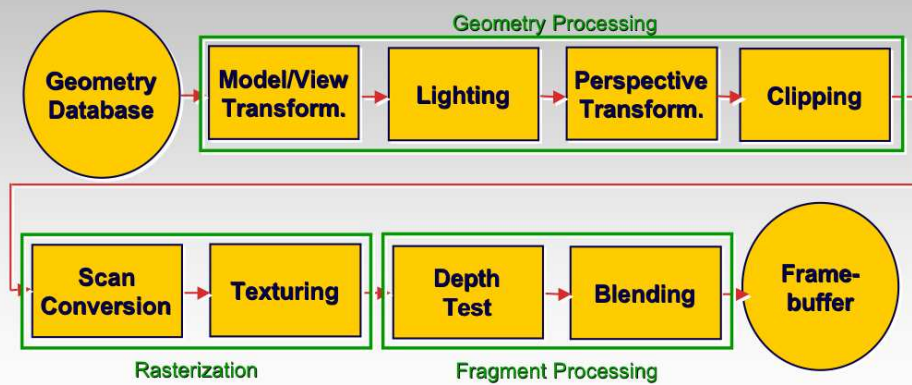
Clipping

CPSC 314

© Wolfgang Heidrich



The Rendering Pipeline



© Wolfgang Heidrich



Line Clipping

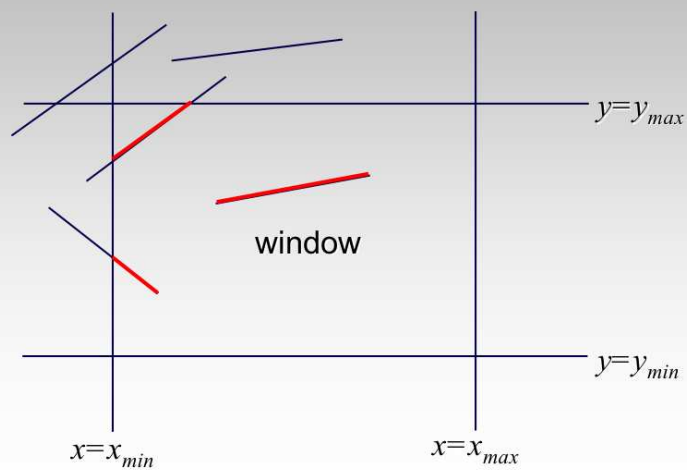
Purpose

- Originally: 2D
 - Determine portion of line inside an axis-aligned rectangle (screen or window)
- 3D
 - Determine portion of line inside axis-aligned parallelepiped (viewing frustum in NDC)
 - Simple extension to the 2D algorithms

© Wolfgang Heidrich



Line Clipping



© Wolfgang Heidrich



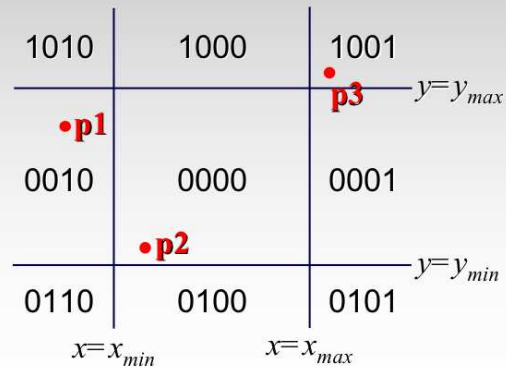
Line Clipping

Outcodes (Cohen, Sutherland '74)

- 4 flags encoding position of a point relative to top, bottom, left, and right boundary

- E.g.:

- OC(p1)=0010
- OC(p2)=0000
- OC(p3)=1001



© Wolfgang Heidrich



Line Clipping

Line segment:

- (p1,p2)

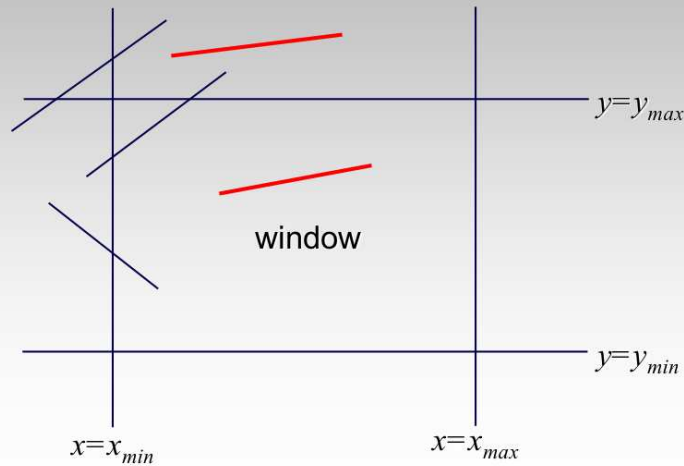
Trivial cases:

- $OC(p1) == 0 \ \&\& \ OC(p2) == 0$
 - Both points inside window, thus line segment completely visible (trivial accept)
- $(OC(p1) \ \& \ OC(p2)) != 0$
 - There is (at least) one boundary for which both points are outside (same flag set in both outcodes)
 - Thus line segment completely outside window (trivial reject)

© Wolfgang Heidrich



Line Clipping



© Wolfgang Heidrich



Line Clipping

α -Clipping

- Handling of all the non-trivial cases
- Improvement of earlier algorithms (Cohen/Sutherland, Cyrus/Beck, Liang/Barsky)
- Define window-edge-coordinates of a point $\mathbf{p}=(x,y)^T$
 - $WEC_L(\mathbf{p})= x-x_{min}$
 - $WEC_R(\mathbf{p})= x_{max}-x$
 - $WEC_B(\mathbf{p})= y-y_{min}$
 - $WEC_T(\mathbf{p})= y_{min}-y$

Negative if outside!

© Wolfgang Heidrich



Line Clipping

α -Clipping

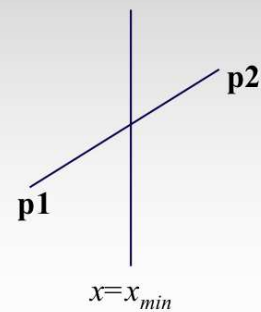
- Line segment defined as: $p1 + \alpha(p2 - p1)$
- Intersection point with one of the borders (say, left):

$$x_1 + \alpha(x_2 - x_1) = x_{min} \Leftrightarrow$$

$$\alpha = \frac{x_{min} - x_1}{x_2 - x_1}$$

$$= \frac{x_{min} - x_1}{(x_2 - x_{min}) - (x_1 - x_{min})}$$

$$= \frac{WEC_L(x_1)}{WEC_L(x_1) - WEC_L(x_2)}$$



© Wolfgang Heidrich



Line Clipping

α -Clipping: algorithm

alphaClip(p1, p2, window) {

 Determine window-edge-coordinates of p1, p2

 Determine outcodes $OC(p1)$, $OC(p2)$

 Handle trivial accept and reject

$\alpha1 = 0$; // line parameter for first point

$\alpha2 = 1$; // line parameter for second point

 ...

© Wolfgang Heidrich



Line Clipping

α -Clipping: algorithm (cont.)

```
...  
// now clip point p1 against all edges  
if( OC(p1) & LEFT_FLAG ) {  
     $\alpha = \text{WEC}_L(p1) / (\text{WEC}_L(p1) - \text{WEC}_L(p2));$   
     $\alpha1 = \max(\alpha1, \alpha);$   
}
```

Similarly clip p1 against other edges

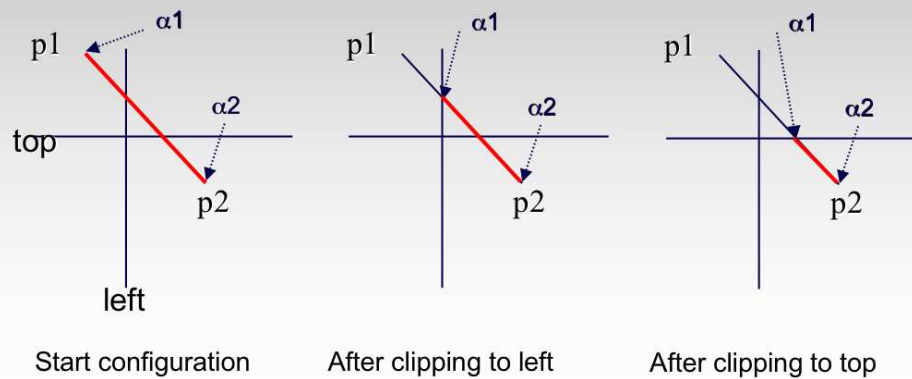
...

© Wolfgang Heidrich



Line Clipping

α -Clipping: example for clipping p1



© Wolfgang Heidrich



Line Clipping

α -Clipping: algorithm (cont.)

```
...  
// now clip point p2 against all edges  
if( OC(p2) & LEFT_FLAG ) {  
     $\alpha = \text{WEC}_L(\mathbf{p2}) / (\text{WEC}_L(p1) - \text{WEC}_L(p2));$   
     $\alpha2 = \min(\alpha2, \alpha);$   
}
```

Similarly clip p1 against other edges

...

© Wolfgang Heidrich



Line Clipping

α -Clipping: algorithm (cont.)

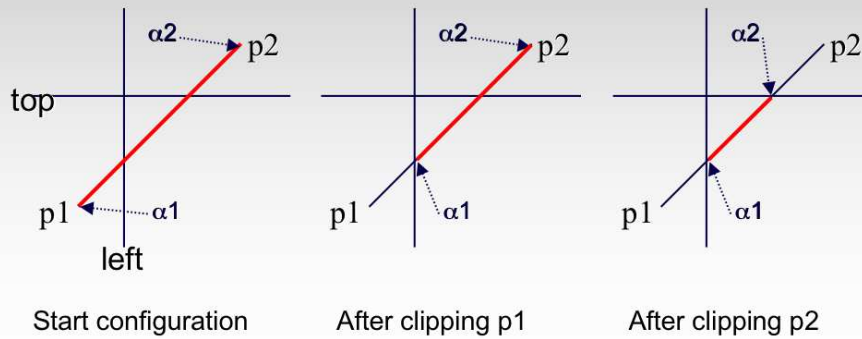
```
...  
// wrap-up  
if( $\alpha1 > \alpha2$ )  
    no output;  
else  
    output line from  $p1 + \alpha1(p2 - p1)$  to  $p1 + \alpha2(p2 - p1)$   
} // end of algorithm
```

© Wolfgang Heidrich



Line Clipping

Example

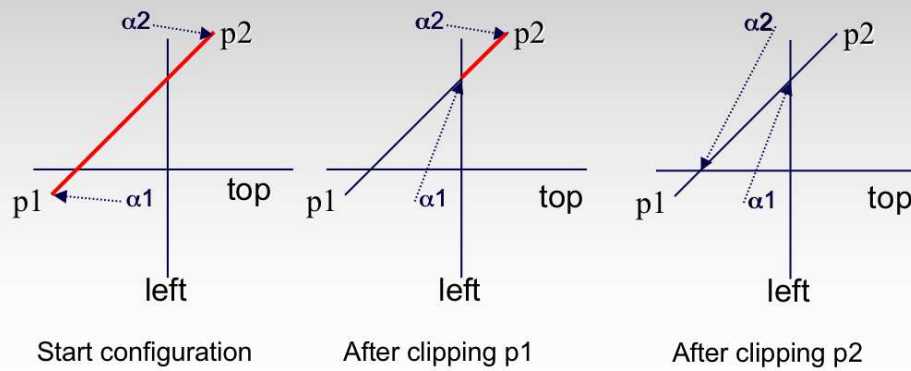


© Wolfgang Heidrich



Line Clipping

Another Example



© Wolfgang Heidrich



Line Clipping in 3D

Approach:

- Clip against parallelepiped in NDC (*after* perspective transform)
- Means that the clipping volume is always the same!
 - OpenGL: $x_{min}=y_{min}=-1, x_{max}=y_{max}=1$
- Boundary lines become boundary planes
 - *But outcodes and WECs still work the same way*
 - *Additional front and back clipping plane*
 - ♣ $z_{min}=0, z_{max}=1$ in OpenGL

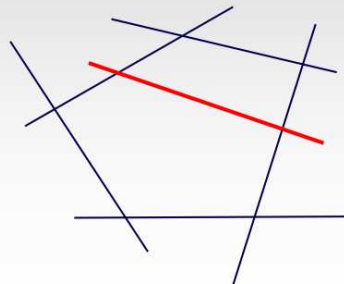
© Wolfgang Heidrich



Line Clipping

Extensions

- Algorithm can be extended to clipping lines against
 - *Arbitrary convex polygons (2D)*
 - *Arbitrary convex polytopes (3D)*

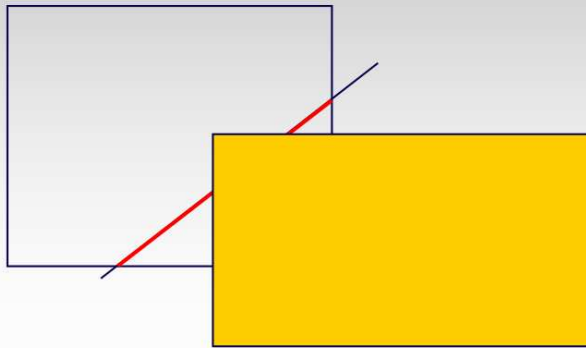


© Wolfgang Heidrich

Line Clipping

Non-convex clipping regions

- E.g.: windows in a window system!

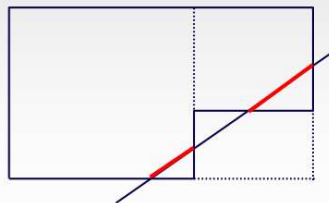


© Wolfgang Heidrich

Line Clipping

Non-convex clipping regions

- Problem: arbitrary number of visible line segments
- Different approaches:
 - *Break down polygon into convex parts*
 - *Scan convert for full window, and discard hidden pixels*



© Wolfgang Heidrich



Polygon Clipping

Objective

- 2D: clip polygon against rectangular window
 - Or general convex polygons
 - Extensions for non-convex or general polygons
- 3D: clip polygon against parallelepiped

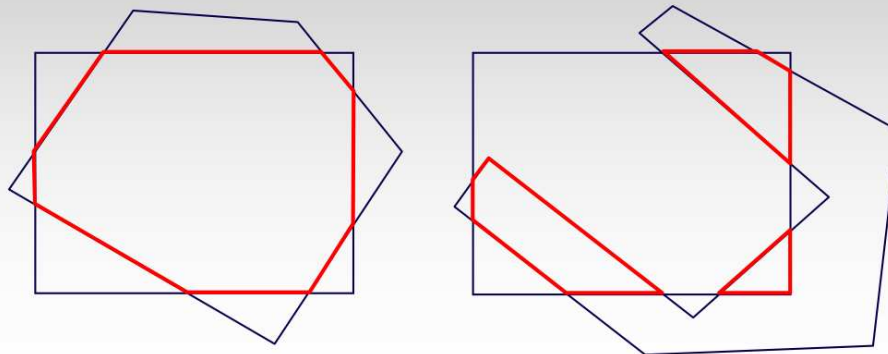
© Wolfgang Heidrich



Polygon Clipping

Not just clipping all boundary lines

- May have to introduce new line segments



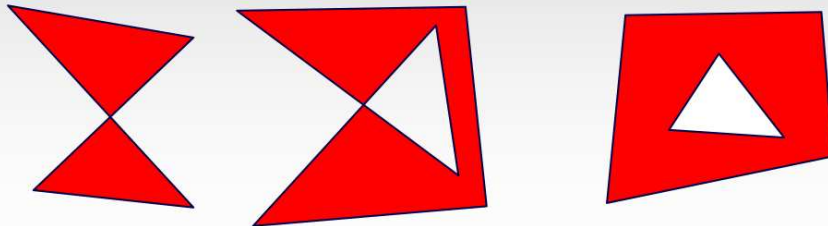
© Wolfgang Heidrich



Polygon Clipping

Classes of Polygons

- Triangles
- Convex
- Concave
- Holes and self-intersection



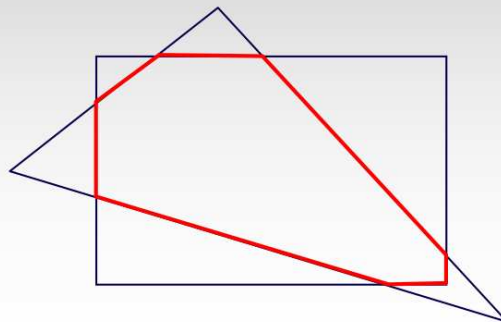
© Wolfgang Heidrich



Polygon Clipping

Sutherland/Hodgeman Algorithm ('74)

- Arbitrary convex or concave object polygon
 - *Restriction to triangles does not simplify things*
- Convex subject polygon (window)



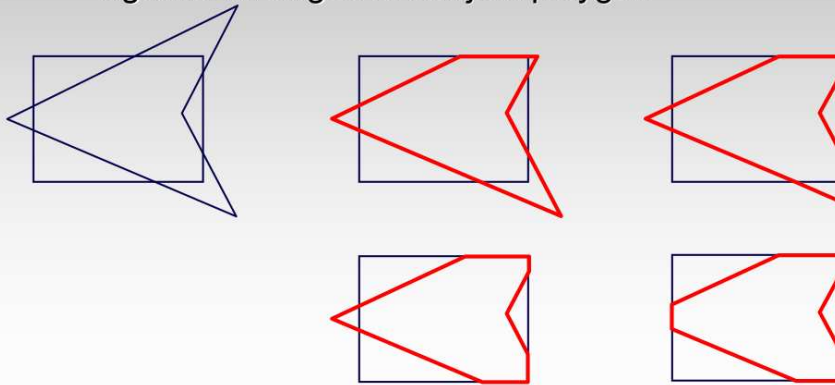
© Wolfgang Heidrich



Polygon Clipping

Sutherland/Hodgeman Algorithm ('74)

- Approach: clip object polygon independently against all edges of subject polygon



© Wolfgang Heidrich



Polygon Clipping

Clipping against one edge:

```
clipPolygonToEdge( p[n], edge ) {  
  for( i= 0 ; i< n ; i++ ) {  
    if( p[i] inside edge ) {  
      if( p[i-1] inside edge ) // p[-1]= p[n-1]  
        output p[i];  
      else {  
        p= intersect( p[i-1], p[i], edge );  
        output p, p[i];  
      }  
    } else...  
  }  
}
```

© Wolfgang Heidrich



Polygon Clipping

Clipping against one edge (cont)

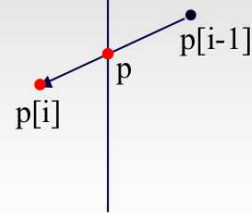
- $p[i]$ inside: 2 cases

inside | outside



Output: $p[i]$

inside | outside



Output: $p, p[i]$

© Wolfgang Heidrich



Polygon Clipping

Clipping against one edge (cont)

```
...
else { // p[i] is outside edge
  if( p[i-1] inside edge ) {
    p= intersect(p[i-1], p[I], edge );
    output p;
  }
} // end of algorithm
```

© Wolfgang Heidrich

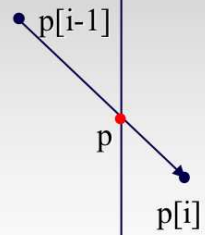


Polygon Clipping

Clipping against one edge (cont)

- $p[i]$ outside: 2 cases

inside | outside



Output: p

inside | outside



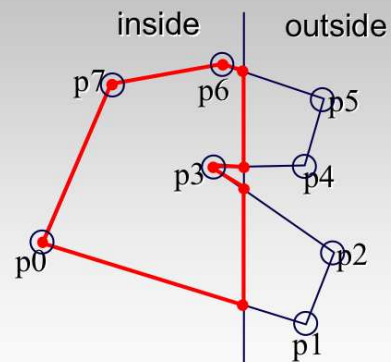
Output: nothing

© Wolfgang Heidrich



Polygon Clipping

Example



© Wolfgang Heidrich



Polygon Clipping

Sutherland/Hodgeman Algorithm

- Inside/outside tests: outcodes
- Intersection of line segment with edge: window-edge coordinates
- Similar to Cohen/Sutherland algorithm for line clipping

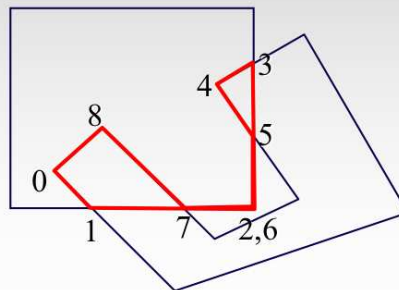
© Wolfgang Heidrich



Polygon Clipping

Sutherland/Hodgeman Algorithm

- Discussion:
 - *Works for concave polygons*
 - *But generates degenerate cases*



© Wolfgang Heidrich



Polygon Clipping

Sutherland/Hodgeman Algorithm

- Discussion:
 - *Clipping against individual edges independent*
 - Great for hardware (pipelining)
 - *All vertices required in memory at the same time*
 - Not so good, but unavoidable
 - Another reason for using triangles only in hardware rendering

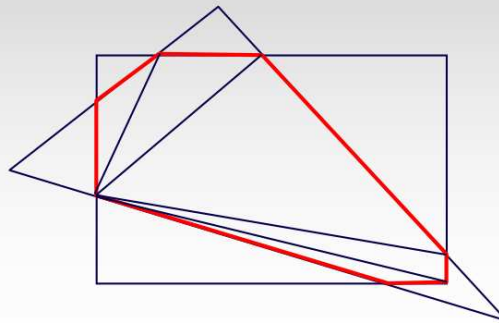
© Wolfgang Heidrich



Polygon Clipping

Sutherland/Hodgeman Algorithm

- For Rendering Pipeline:
 - *Re-triangulate resulting polygon (can be done for every individual clipping edge)*



© Wolfgang Heidrich



Polygon Clipping

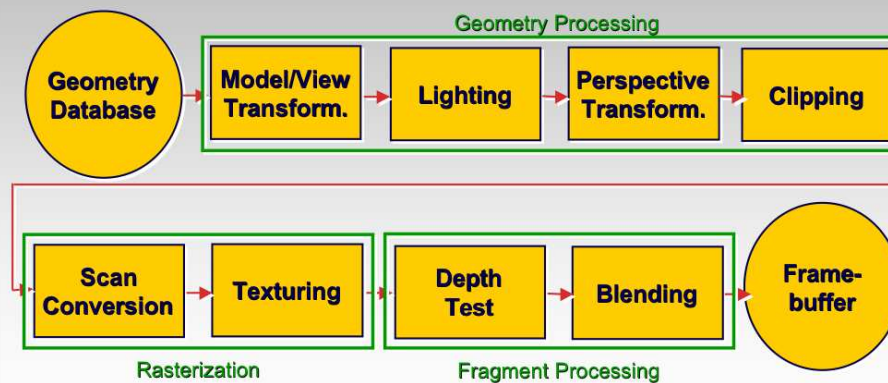
Other Polygon Clipping Algorithms

- Weiler/Aetherton '77:
 - *Arbitrary concave polygons with holes both as subject and as object polygon*
- Vatti '92:
 - *Self intersection allowed as well*
- ... many more
 - *Improved handling of degenerate cases*
 - *But not often used in practice due to high complexity*

© Wolfgang Heidrich



The Rendering Pipeline



© Wolfgang Heidrich



Coming Up:

Tuesday, Oct 9:

- Hidden surface removal / visibility

Thursday, Oct 11:

- Scan Conversion