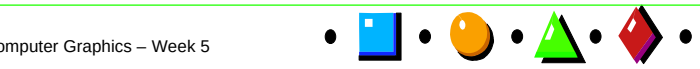
Computer Graphics - Week 5

Bengt-Olaf Schneider
IBM T.J. Watson Research Center

Questions about Last Week ?



Comments about the assignment

- **YOU SHOULD HAVE REALLY STARTED BY NOW !!**

- **The assignment is due next week.**
- **Use the office hours and the news group if you have questions or problems.**
  - Elias: Tue 12-1, Thu 12-1
  - Bengt: Wed 4-5 pm in Adjunct Office MUDD 460

Comments about the assignment (cont'd)

- **Non-uniform scaling squashes my planets into ellipsoids**
  - Well, then don't apply non-uniform scaling to the planets
  - Instead put them into their orbits "manually"

- **OK, I'm done with all the compulsory stuff in the assignment. How do I make my planets look nicer ?**
  - Good for you !!
  - Assign realistice colors (white Moon, yellow Sun, blue Earth, read Mars, etc.). To pick colors use one of the color mixers in Windows (e.g. to change the color of the desktop elements) or in X-Windows.
  - Apply textures. Read ahead in the OGL programming guide, learn how to do textures and find images of planets on the web.

Overview of Week 5

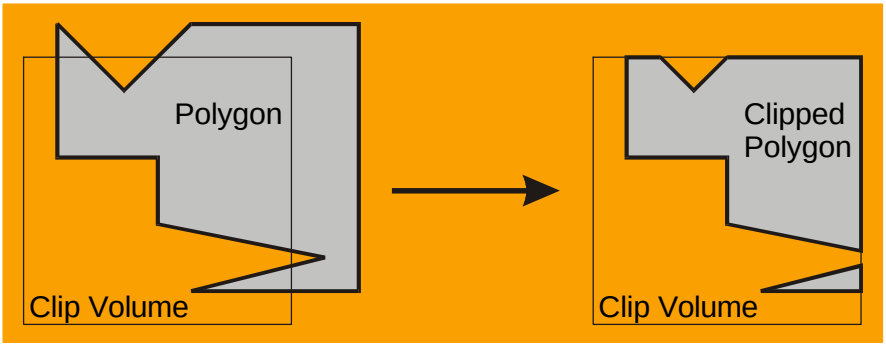
- Removing parts of the model
- Clipping for lines and polygons
- Scissoring

Clipping: Motivation

- Clipping intersects two geometric objects
  - In general, both objects can be of arbitrary shape
- Clipping against the view volume
  - Typically, convex view volume
  - Eliminates geometry outside of the view
  - Avoids warping of geometry from behind the eye to in front of the eye due to perspective projection
  - Removes parts too close to and too far away from the viewer
- We will only discuss clipping in 2D. Extension to 3D is straight-forward.

Clipping: Overview

- We will first consider clipping against a convex clip volume, delimited by clip planes
- Later we will look at clipping against more complex clip volumes

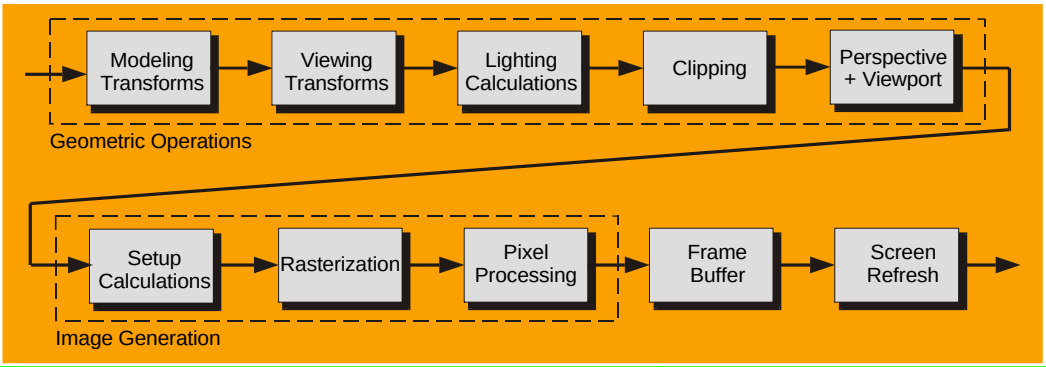


Clipping: Terminology

- **Clip Volume**
  - Closed volume of *n*-dimensional space that defines which parts of the model should be retained
  - We define that the boundaries of the clip volume are considered to be part of (i.e. inside) the clip volume
- **Clip Boundaries**
  - Closed surface in (n-1)-dimensional space separating inside and outside of the clip volume
  - Typically, defined using intersection of planes, e.g. view volume
- **Clip Planes**
  - Planes defining the clip boundary and delimiting the clip volume

Clipping as Part of the Rendering Pipeline

- Clipping is part of the geometry stage
- Clipping occurs in canonical coordinates



Lines and Planes

- Lines are described as multiples of a vector aligned with line. The vector is defined by two points.

$$\ell: \quad P = P_1 + t \cdot (P_2 - P_1)$$

- Planes are described as linear expressions in x, *y*, *z*.
  - The (normalized) expression defines the distance of a point (*x*,*y*,*z*) from the plane.
  - Points in the plane have distance 0.
  - Planes can be reoriented by multiplying the expression with -1.

$$\wp: \quad ax + by + cz + d = 0$$

Clipping of Primitives

- **We will only consider clipping of points, lines and polygons**
  - Clipping of line segments is part of polygon clipping
- **Higher order primitives are tesselated first, then clipped**

Clipping Points (1)

- **Let's first consider whether points are inside the clip volume**
- **To be inside the clip volume, a point must be on the inside of all clip planes**
  - We define "inside" as those points that have a negative distance from the clip plane

$$D_i = a_i x + b_i y + c_i z + d_i$$

A point is inside the clip volume, iff

$$D_i \leq 0 \quad \forall \; i$$

Clipping Points (2)

- **For canonical view volumes these criteria simplify to:**
  - Parallel view volume:

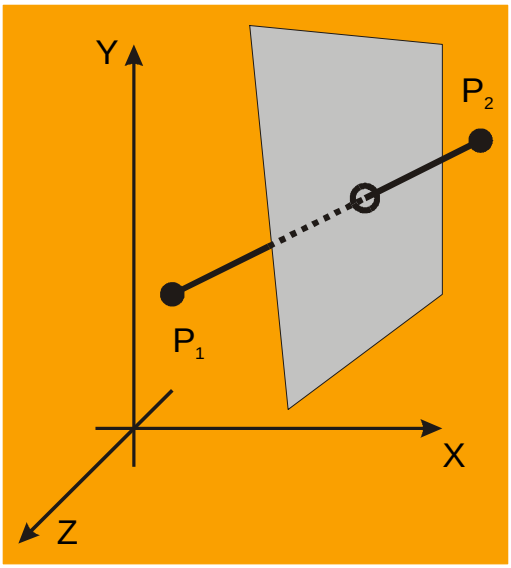
$$-1 \le x \le +1$$
$$-1 \le y \le +1$$
$$-1 \le z \le 0$$

  - Perspective view volume:

$$-z \le x \le +z$$
$$-z \le y \le +z$$
$$-1 \le z \le z_{MIN}$$

Clipping Lines

- **To clip a line against the clip volume, we must compute the intersection(s) of the line with the clip planes**
- **A line is defined by two end points P_1 and P_2**
- **A plane is defined by its plane equation *ax+by+cz+d* = 0.**



Line-Plane Intersection (1)

Line:

$$P_L = P_1 + t \cdot (P_2 - P_1)$$

Plane:

$$0 = ax + by + cz + d$$

$$\Rightarrow$$

$$0 = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \bullet \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = V \bullet P$$

Line - Plane Intersection:

$$\begin{aligned} 0 &= V \bullet P_L \\ &= V \bullet \left(P_1 + t \cdot (P_2 - P_1)\right) \\ &= V \bullet P_1 + t \cdot V \bullet (P_2 - P_1) \end{aligned}$$

$$\Rightarrow$$

$$t = -\frac{V \bullet P_1}{V \bullet (P_2 - P_1)}$$

$$= -\frac{ax_1 + by_1 + cz_1 + d}{a(x_2 - x_1) + b(y_2 - y_1) + c(z_2 - z_1)}$$

Intersection iff

$$0 \le t \le 1$$

Line-Plane Intersection (2)

- **Intersection calculation is expensive:**
  - 8 additions / subtractions
  - 6 multiplications
  - 1 division
  - 2 comparisons
- **Simplifications are important**
  - Canonical view volume
  - Trivial accept/reject

Line - Plane Intersection:

$$\begin{aligned} 0 &= V \bullet P_L \\ &= V \bullet \left(P_1 + t \cdot (P_2 - P_1)\right) \\ &= V \bullet P_1 + t \cdot V \bullet (P_2 - P_1) \end{aligned}$$

$$\Rightarrow$$

$$t = -\frac{V \bullet P_1}{V \bullet (P_2 - P_1)}$$

$$= -\frac{ax_1 + by_1 + cz_1 + d}{a(x_2 - x_1) + b(y_2 - y_1) + c(z_2 - z_1)}$$

Intersection iff

$$0 \le t \le 1$$

Line-Plane Intersection (3)

- **Canonical view volume**
  - Parallel projection:
  - *x*,*y*,*z* = const
- **Computational cost reduced to**
  - 2 additions / subtractions (1 if *d*=0)
  - 2 division
  - 2 comparisons

Line - Plane Intersection for a = b = 0,
$z = -d / c = const.$

$$t = -\frac{ax_1 + by_1 + cz_1 + d}{a(x_2 - x_1) + b(y_2 - y_1) + c(z_2 - z_1)}$$

$$t = -\frac{cz_1 + d}{c(z_2 - z_1)} = \frac{z_1 + d/c}{z_2 - z_1}$$

Intersection iff

$$0 \le t \le 1$$

Line-Plane Intersection (4)

- **Canonical view volume**
  - Perspective projection:
  - *x*,*y* = +/- *z*
- **Computational cost reduced to**
  - 4 additions / subtractions
  - 1 division
  - 2 comparisons

Line - Plane Intersection for
b = d = 0 and a = c,

$$x = -z$$

$$t = -\frac{ax_1 + by_1 + cz_1 + d}{a(x_2 - x_1) + b(y_2 - y_1) + c(z_2 - z_1)}$$

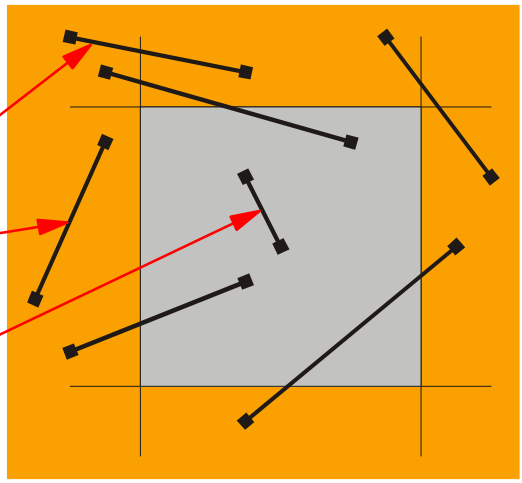
$$t = -\frac{x_1 + z_1}{(x_2 - x_1) + (z_2 - z_1)}$$

Intersection iff

$$0 \le t \le 1$$

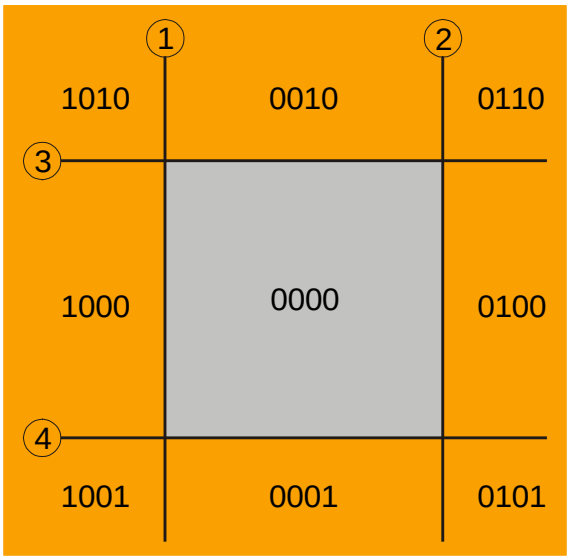
Line Clipping (1)

- **Two-step process:**
  - Trivial accept/reject of line segments
  - Clipping of remaining line segments
- **Trivial rejection**
  - Both endpoints are outside the same clip plane
- **Trivial accept**
  - Both endpoints lie inside the clip volume, i.e. inside all clip planes



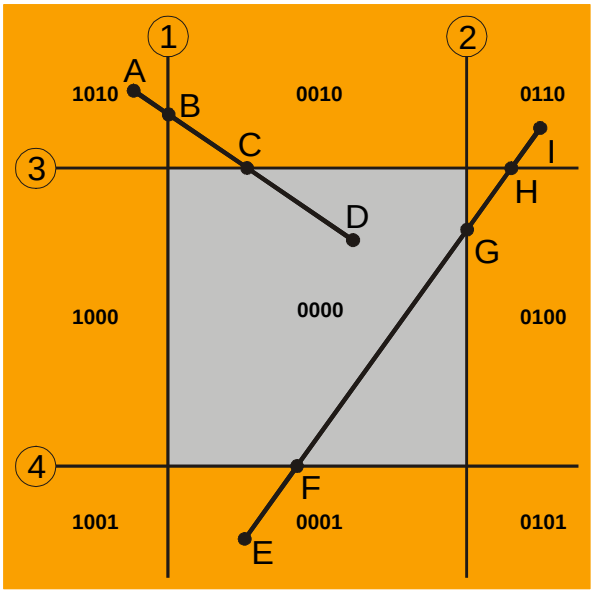
Line Clipping (2)

- **Clipcodes**
  - 1 bit per clip plane
  - Bit is set if the vertex is outside the clip plane
- **Trivial rejection**
  - Bitwise AND of clip codes for endpoints is non-zero
- **Trivial accept**
  - Clip codes for endpoints is 0
- **Works for all convex clip volumes !!**



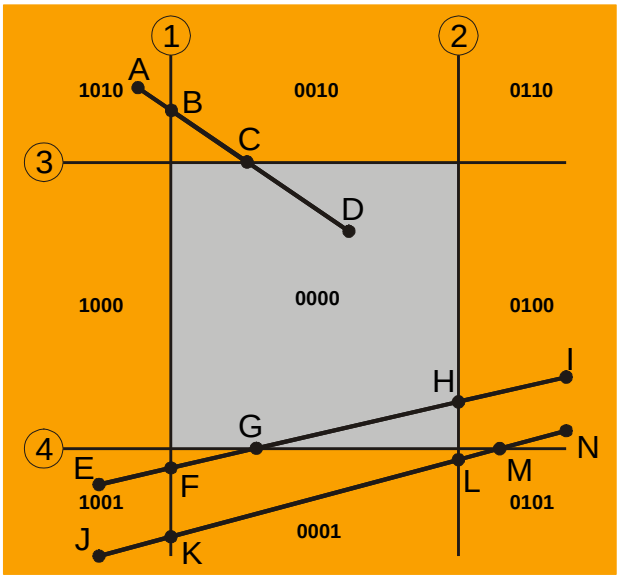
Cohen-Sutherland Line Clipping (1)

- If no trivially accept or reject, we must clip
- Successively clip line segment against clip planes, until remaining line segment can be either accepted or rejected trivially



Cohen-Sutherland Line Clipping (2)

- Line segment A-D
  - CP1: B-D
  - CP3: C-D trivial accept
- Line segment E-I
  - CP1: F-I
  - CP2: F-H
  - CP4: G-H trivial accept
- Line segment J-N
  - CP1: K-N
  - CP2: K-L trivial reject (4)



Cohen-Sutherland Line Clipping (3)

- **Procedure (for details see textbook !):**
  - Compute clip codes for both vertices
  - Test for trivial accept or reject → DONE
  - Pick a vertex outside of the clip volume (using the clip codes)
  - Clip against the first clip plane that intersects with the line segment (use the clip codes to determine that clip plane)
  - Compute intersection with that clip plane
  - Compute clip code for new vertex

Parametric Line Clipping (1)

- **Developed by Cyrus & Beck (1978) and later refined by Liang & Barsky (1984)**
- **More powerful trivial rejection algorithm, that detects lines that cross several outside clip regions**
  - Reduces the number of actual intersection calculations wrt Cohen-Sutherland algorithm
  - Less advantageous if many lines can be trivially accepted/rejected by Cohen-Sutherland algorithm

Parametric Line Clipping (2)

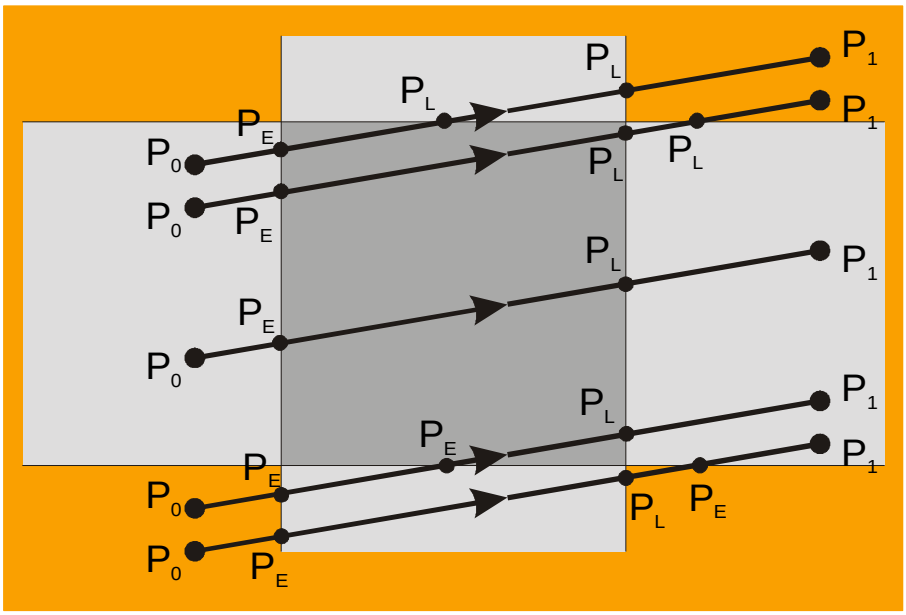
- **Recall the expression of computing a line-plane intersection**

$$t = -\frac{V \bullet P_1}{V \bullet (P_1 - P_0)}$$

- **Define intersections as "entering" or "leaving"**
  - Call such points P_E and P_L with associated t_E and t_L
  - For oblique lines there are 2 P_E and 2 P_L.
  - Investigate the sequence of P_E and P_L intersections
  - The edge intersect the clip volume iff

$$0 \le t \le 1 \quad \&\& \quad \max(t_{E1}, t_{E2}) < \min(t_{L1}, t_{L2})$$

Parametric Line Clipping (3)



Parametric Line Clipping (4)

- **Procedure**
  - Initialize $t_E = 0$ and $t_L = 1$
  - Compute parameters of interscetion of line and clip planes: t_{E1}, t_{E2}, t_{L1}, t_{L2}
  - Compute $t_E = \max(t_E, t_{E1}, t_{E2})$ and $t_L = \min(t_L, t_{L1}, t_{L2})$
  - If $(t_E > t_L)$ the edge does not intersect the clip volume: reject
  - If not rejected:
    Compute intersection coordinates for given parameters

Parametric Line Clipping (5)

- **Extends readily to convex 3D clip volume**
  - Compute line parameters for intersection with all clip planes
  - Compute maximum of entering intersections and minimum of leaving intersections, provided that parameter is between 0 and 1.
  - Reject edge if parameter of maximum entering intersection is greater the parameter of minimu leaving intersection

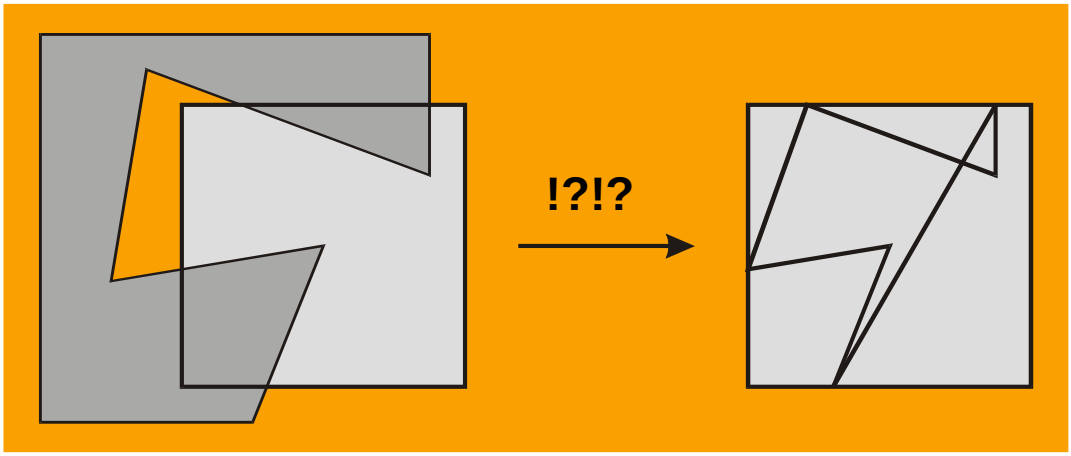
Parametric Line Clipping (6)

- **Summary:**
- **More powerful trivial rejection algorithm, that detects lines that cross several outside clip regions**
  - Reduces the number of actual intersection calculations wrt Cohen-Sutherland algorithm
  - Less advantageous if many lines can be trivially accepted/rejected by Cohen-Sutherland algorithm

Polygon Clipping

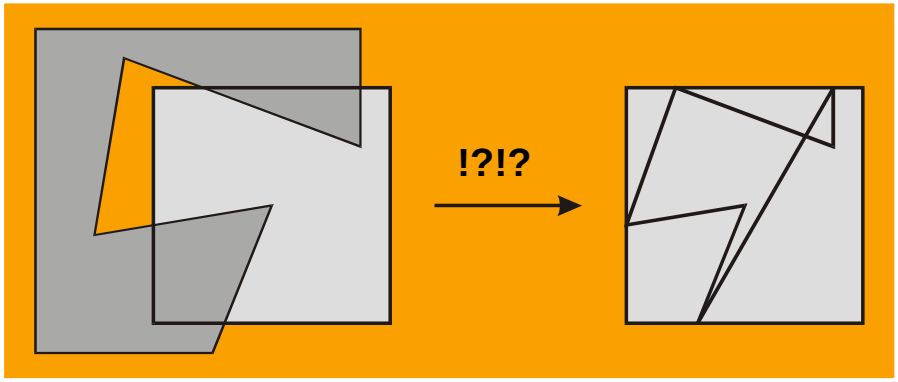
- **Polygons are connected line segments**
- **The interior of the polygon is delimited by the line segments**
- **Polygon clipping will retain those portions of the polygon that are inside the clip volume**

Polygon Clipping: How not to do it ! (1)



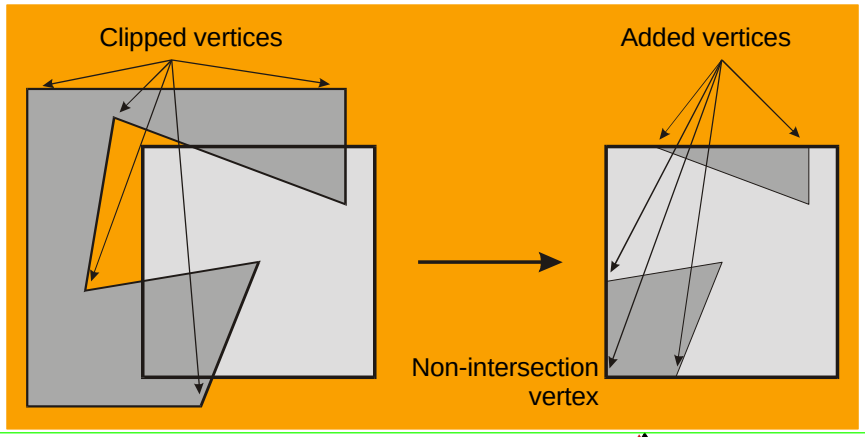
Polygon Clipping: How not to do it ! (2)

- **What went wrong ?**
  - The polygon was treated as a collection of line segments
  - The notion of "interior" was ignored
  - Naive connection of intersection points and inside vertices
  - Similar problems if the clip regions is fully enclosed by the polygon



Polygon Clipping

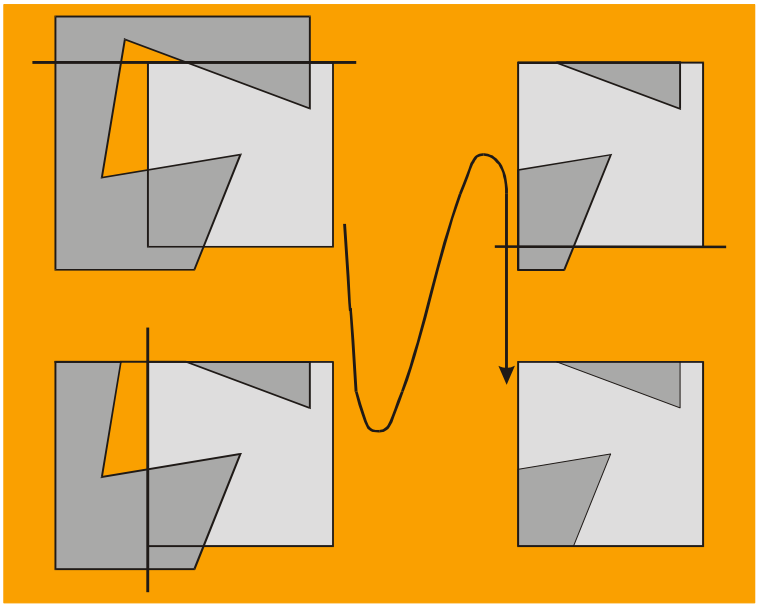
- **Polygon clipping clips these line segments to form the clipped polygon**
- **Different cases**
  - Remove vertices
  - Add vertices
  - Several polygons after clip



Sutherland-Hodgman Polygon Clipping (1)

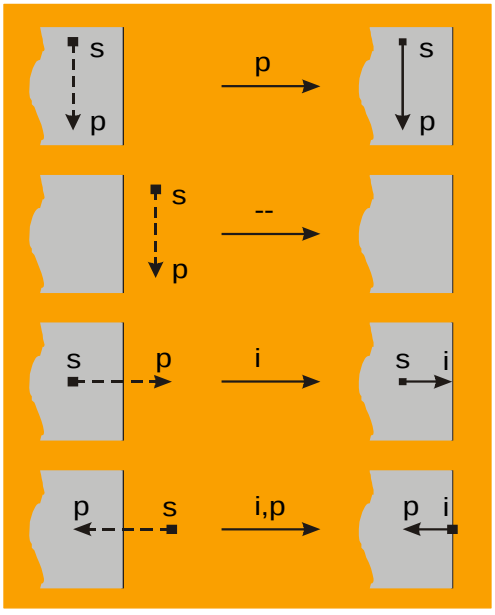
- **All these cases are properly handled**
- **The polygon is clipped successively against all clip planes**
  - The result of clipping the polygon against one clip plane is the input for clipping against the next clip plane
  - (This leads to pipelined implementations of the SH algorithm.)
- **The SH clipping algorithm generalizes readily to convex 3D clip volumes**

Sutherland-Hodgman Polygon Clipping (2)



Sutherland-Hodgman Polygon Clipping (3)

- **Procedure**
  - Input polygon and clipped polygon are lists of vertices
  - Clip each edge and generate 0, 1, or 2 output vertices:
    - Start and end vertex inside: **End vertex**
    - Start and end vertex outside: **none**
    - Only start vertex inside: **intersection**
    - Only end vertex inside: **intersection and end vertex**
  - If the start vertex s is inside it has been generated as output by the previous edge
  - If the first vertex is inside it will be generated as the end vertex of the last edge

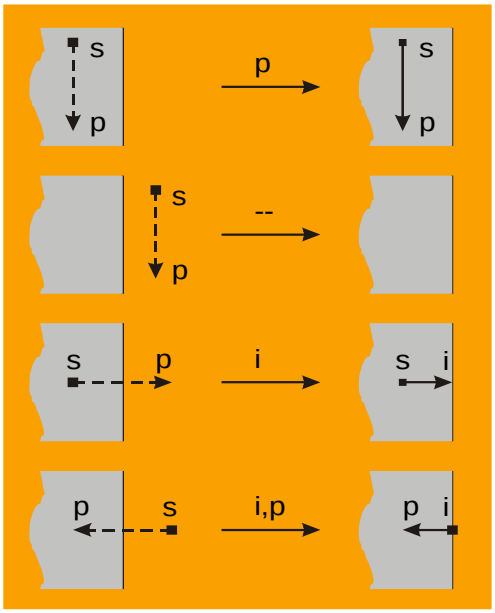


Sutherland-Hodgman Polygon Clipping (4)

- **Pipelining**
  - When clipping has produced a point, it can be used as input for clipping against the next clipping plane
  - Recursive call inside the clipping routine, e.g.

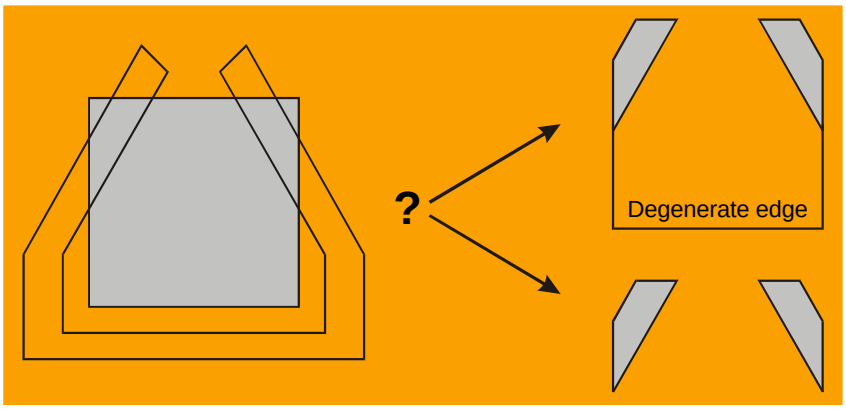
```
SHclip (plane, p)
{
  ... process vertex p ...
  SHclip (plane+1, p) ;
}
```

  - Pipelined hardware with separate stage for each plane



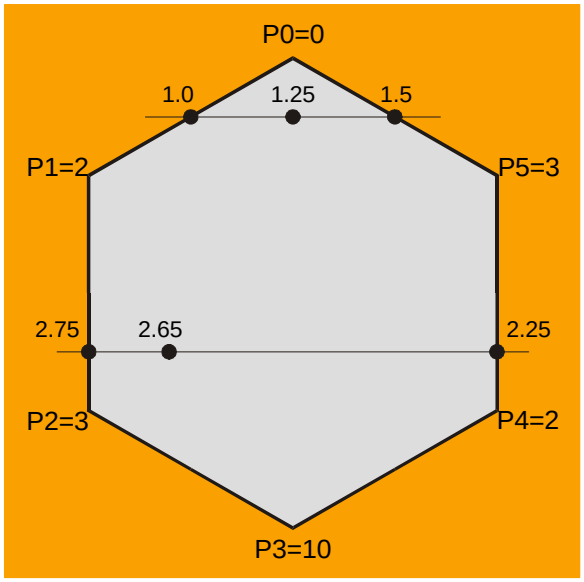
Sutherland-Hodgman Polygon Clipping (5)

- Does the connecting, degenerate edge belong to the clipped polygon or not ?
- Sutherland-Hodgman and Liang-Barsky algorithms generate the phantom edge
- Weiler algorithm produces disjoint polygons



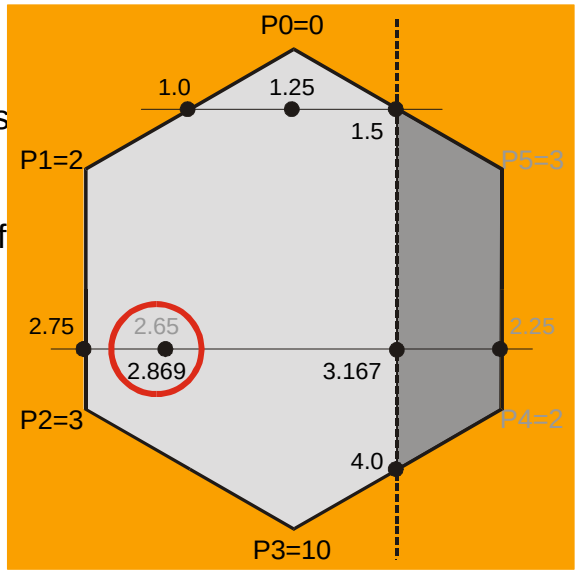
Polygon Clipping and Scan Conversion (1)

- **Scan Conversion (next week)**
  - Computes location of pixels covered by a primitive
  - Polygon scan conversion usually interpolates parameters (color, depth, texture, etc.) across the polygon
  - Bilinear interpolation:
    - Linear interpolation of parameters along edges
    - Linear interpolation of parameters along scanline between edges



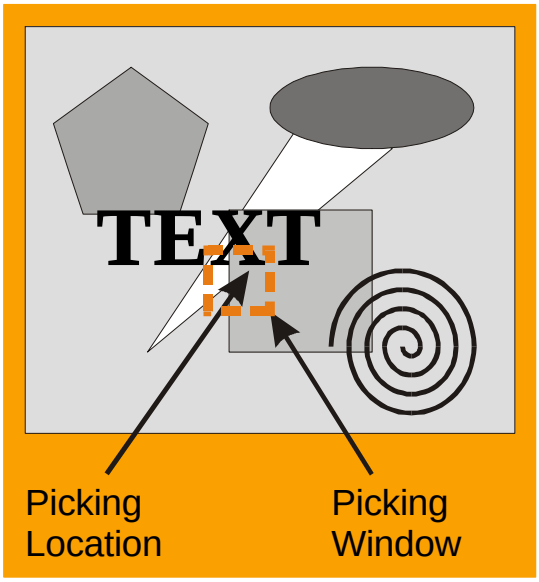
Polygon Clipping and Scan Conversion (2)

- **Clipping such polygons changes interior colors**
  - The effect of clipped vertices is reduced or eliminated
  - Even worse, this effect is dependent on the position of the clip plane and the orientation of the polygon
- **Problem is solved by triangulating before clipping**
  - Triangle provides true linear interpolation of parameters



Picking

- Picking identifies objects on screen
- One way to do that is to use clipping against a small pick volume
- All objects intersecting the pick volume are reported as being picked

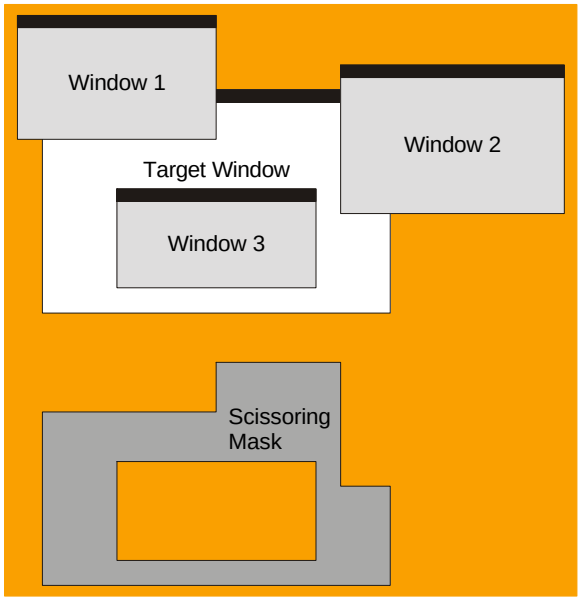


Scissoring (1)

- Clipping
  - Geometric determination of which portions of the scene are inside and outside of the clip region
  - Object-precision approach (calculations are performed with the precision of the underlying geometry)
  - Problematic for irregular clip regions: non-convex, unconnected
  - This occurs frequently in windowing systems, when several other windows overlap the target window
- Scissoring
  - Image-precision, i.e. clip decision is made at pixel resolution
  - Irregular clip regions are defined as pixel masks
  - Regular scissoring regions can also be defined by x/y extents

Scissoring (2)

- **Example: windowing system**
  - Several windows are overlapping the target window
  - Define a bit-mask that is set at visibile pixels
  - Only write frame buffer where bit-mask is set



Scissoring (3)

- **Typically geometric clipping and scissoring are combined**
  - Clipping removes objects outside of the window
  - Scissoring removes image pixels that are hidden by other windows and/or fall outside the screen
- **Advantages:**
  - Irregular clip regions
  - Conceptually easy to implement
- **Drawbacks:**
  - Effort for generating pixels that are ultimately discarded
  - Extra storage and pixel processing to process scissoring bitmask

Summary

- **Clipping algorithms for lines and polygons**
  - Coordinate system and canonical view volume
  - Cohen-Sutherland, Liang-Barsky
  - Sutherland-Hodgman
  - Clipping artifacts
- **Scissoring**

Homework

- **Review clipping algorithms in Foley**
- **Study scan-conversion algorithms**
  - Foley chapters 3.1 - 3.6

Next Week ...

- **Scan Conversion**