Computer Graphics

Clipping

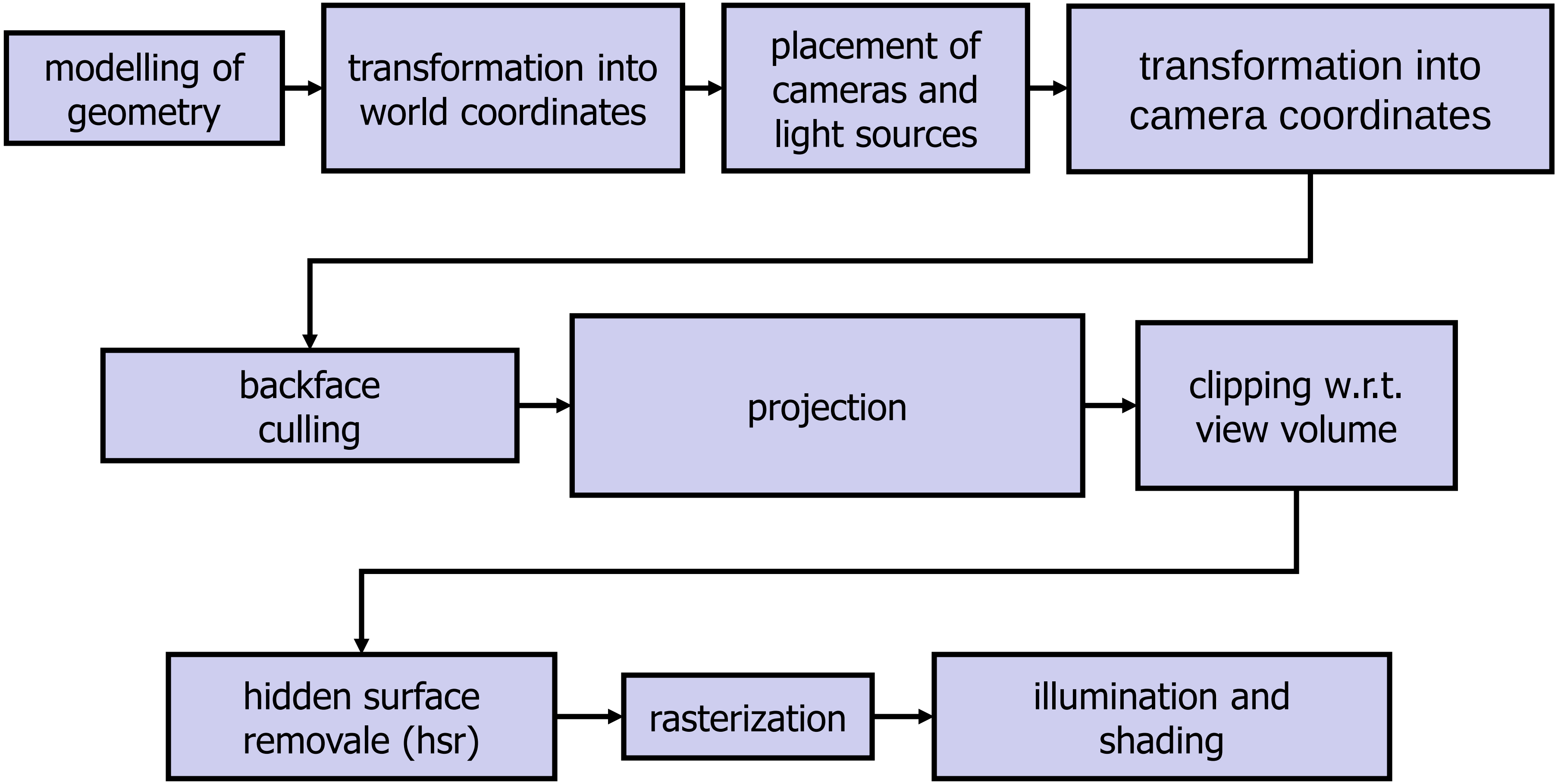
What do we need clipping for?

- rectangular screen
- shows a section of 3D world
- clipping for removing invisible parts
  - avoid processing – view volume clipping
  - avoid raster conversion – viewport clipping
  - avoid color computation – viewport clipping

  $\rightarrow$ faster rendering!
- also for exploration of volumetric data

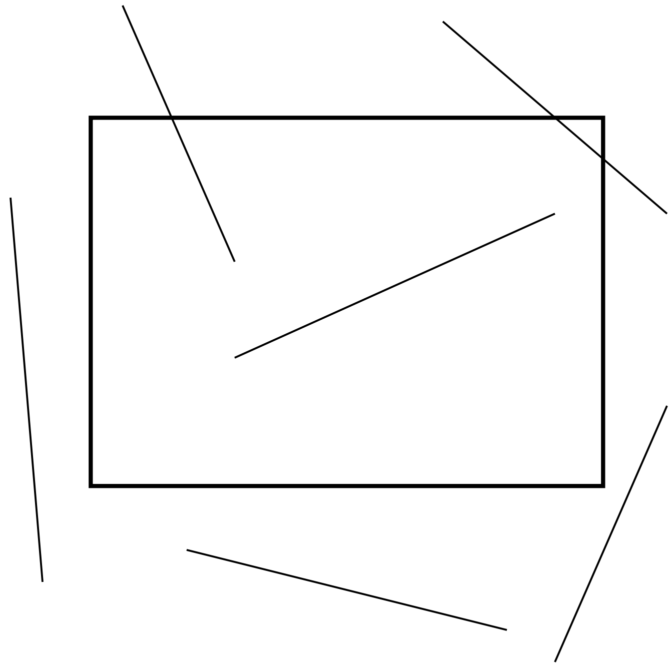
  $\rightarrow$ clip surfaces to view inside

Rendering Pipeline



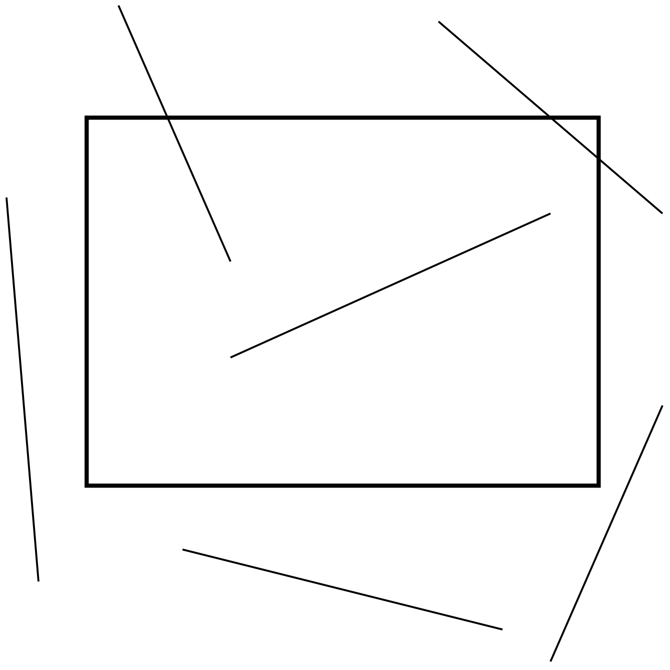
Basic Clipping Objectives

- simple case: clipping of lines on an axis-aligned rectangle in 2D
- decides for each line whether it
  - can be discarded
  - can be drawn in its entirety
  - has to be further examined
- efficiency!



Simple Algorithm

- classify endpoints of line segments
- both inside the rectangle
  → draw line segment
- one inside and
  one outside
  → find intersection
  → draw inside part
- both outside
  → not clear
  → could be partially inside



Clipping

Cohen-Sutherland Algorithm

Cohen-Sutherland Algorithm

- by Danny Cohen and Ivan Sutherland

image: Wikipedia author Kvgd

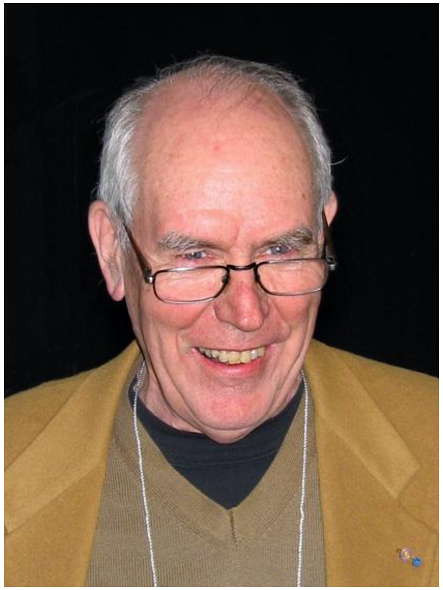
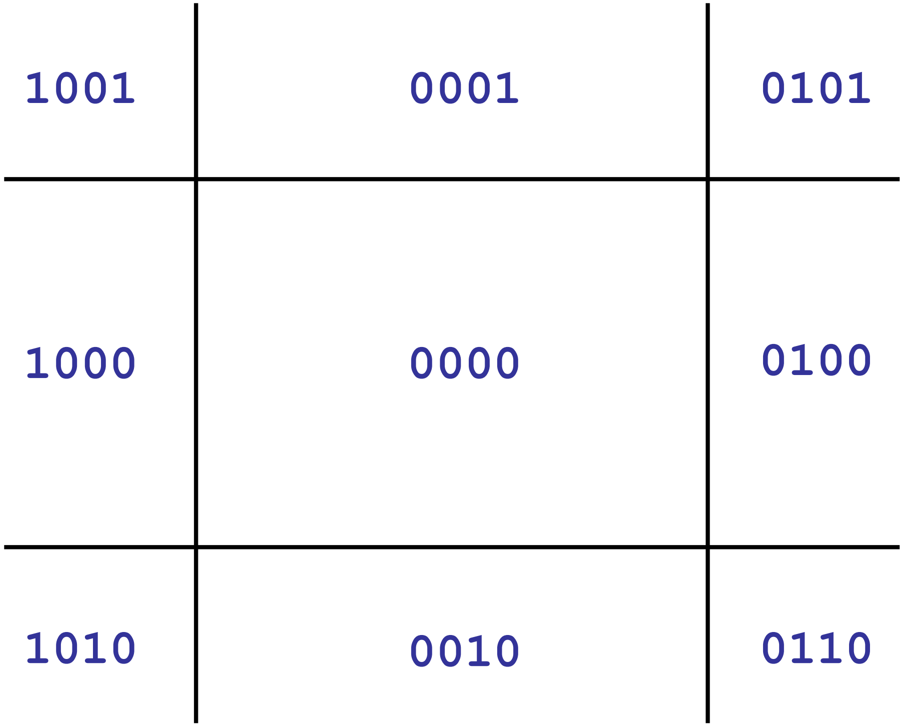


image: Dick Lyon

- created 1967 to realize a flight simulator
- Sutherland also created touch input, graphical UIs, much of computer graphics

Cohen-Sutherland Algorithm

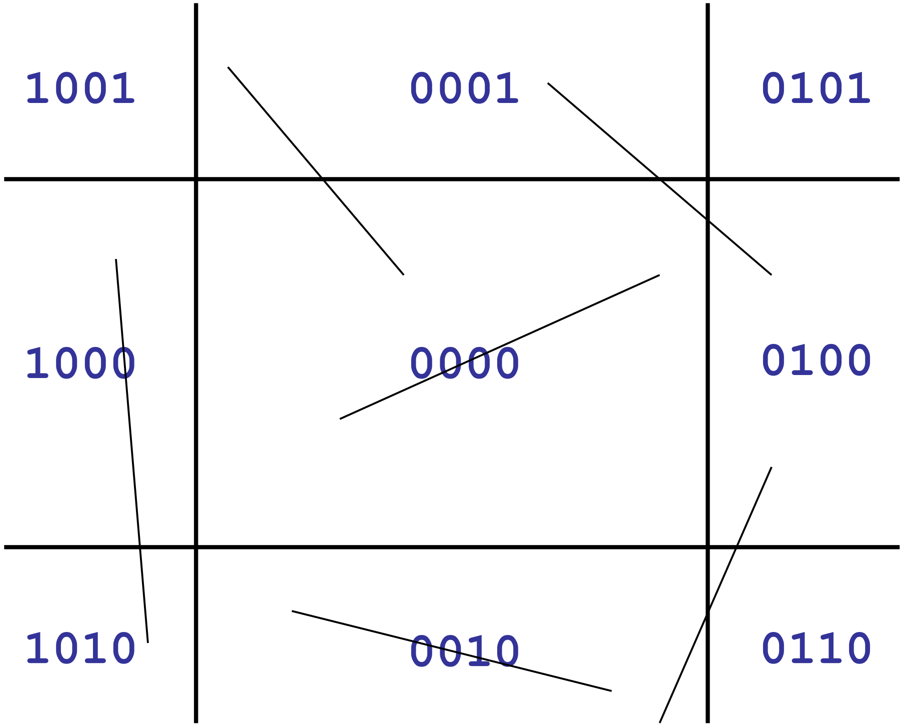
- general idea: 9 regions with binary codes
- each bit records whether point is outside a clip line
  - 1^{st} bit: $< x_{min}$
  - 2^{nd} bit: $> x_{max}$
  - 3^{rd} bit: $< y_{min}$
  - 4^{th} bit: $> y_{max}$
  - order of these does not matter



Cohen-Sutherland Algorithm

- all endpoints classified according to these *outcodes*
- using bit operations for efficient test

```
if (!(oc(P1) | oc(P2)) {
    accept(); return; }
if (oc(P1) & oc(P2)) {
    reject(); return; }
else {
    intersect_segment();
    return; }
```

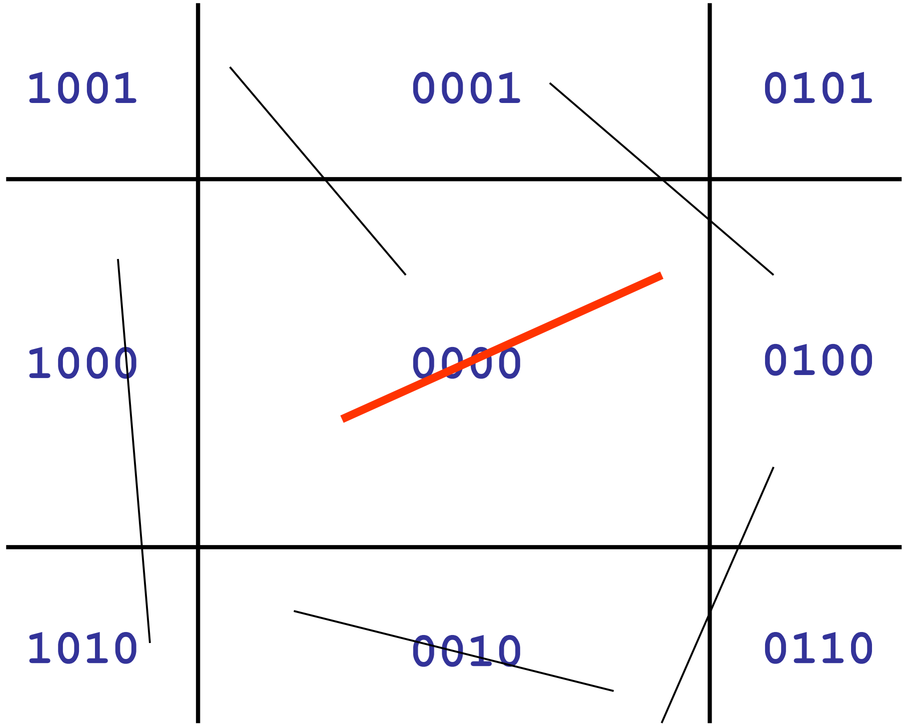


- only one logic operation per test needed!

Cohen-Sutherland Algorithm

Example 1

- $oc(P_1) = 0000$
- $oc(P_2) = 0000$
- first test:
  $oc(P_1) \mid oc(P_2) = 0000$
  $\rightarrow$ accept!
- no further test necessary



Cohen-Sutherland Algorithm

Example 2

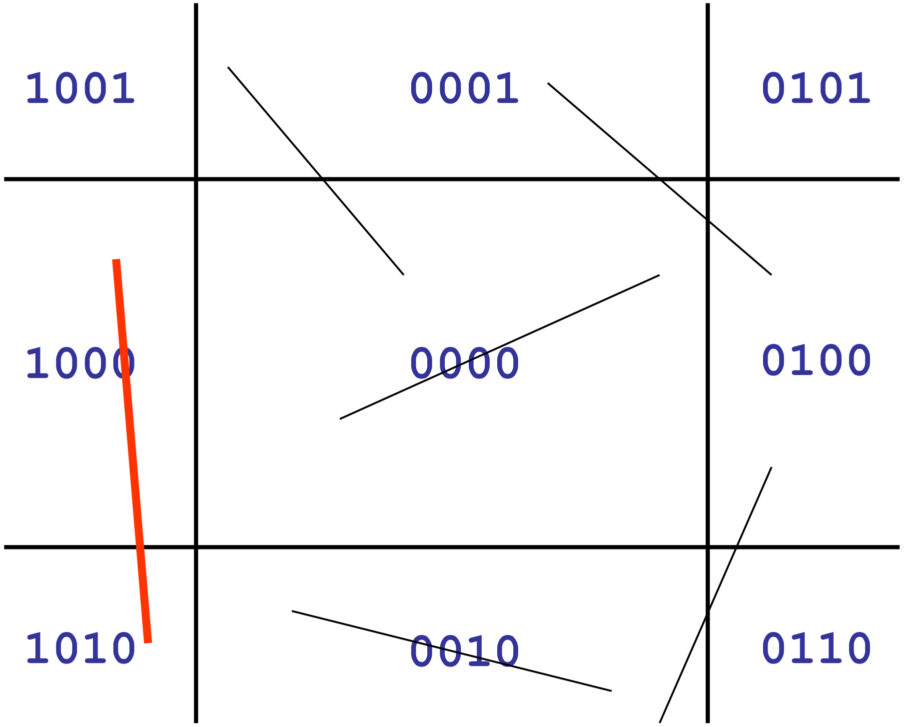
- $oc(P_1) = 1000$
- $oc(P_2) = 1010$
- first test:

  $oc(P_1) \mid oc(P_2) = 1010$

  $\rightarrow$ examine further
- second test:

  $oc(P_1) \,\&\, oc(P_2) = 1000$

  $\rightarrow$ reject!



Cohen-Sutherland Algorithm

Example 3

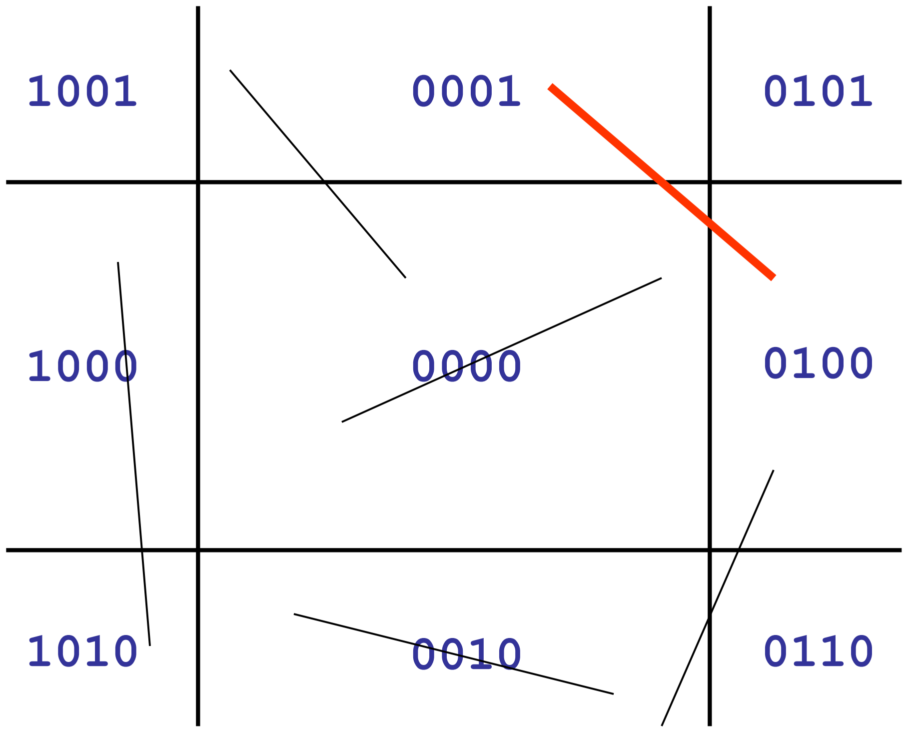
- $oc(P_1) = 0001$
- $oc(P_2) = 0100$
- first test:

  $oc(P_1) \mid oc(P_2) = 0101$

  → examine further
- second test:

  $oc(P_1)\ \&\ oc(P_2) = 0000$

  → intersect!



Cohen-Sutherland Algorithm

Example 4

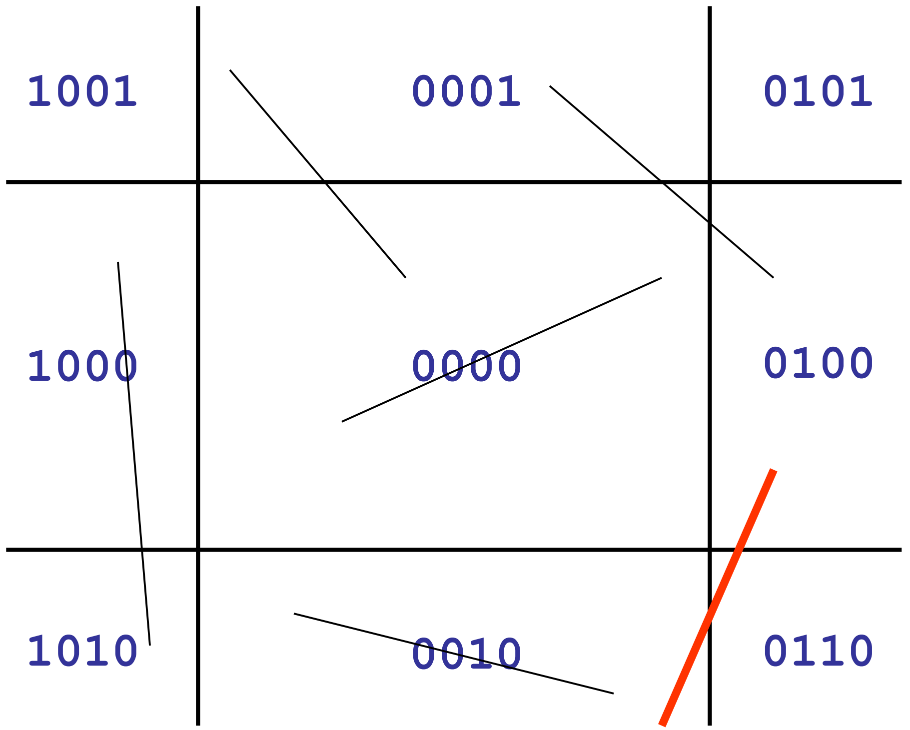
- $oc(P_1) = 0100$
- $oc(P_2) = 0010$
- first test:

  $oc(P_1) \mid oc(P_2) = 0110$

  → examine further
- second test:

  $oc(P_1) \,\&\, oc(P_2) = 0000$

  → intersect!



Clipping

Line Intersections with the Liang-Barsky Algorithm

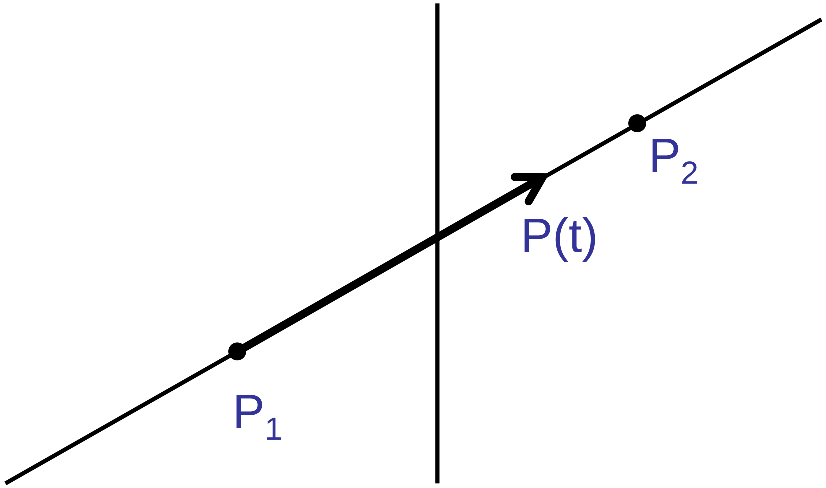
Line Intersections: Liang-Barsky

- by You-Dong Liang and Brian A. Barsky

- created in 1984

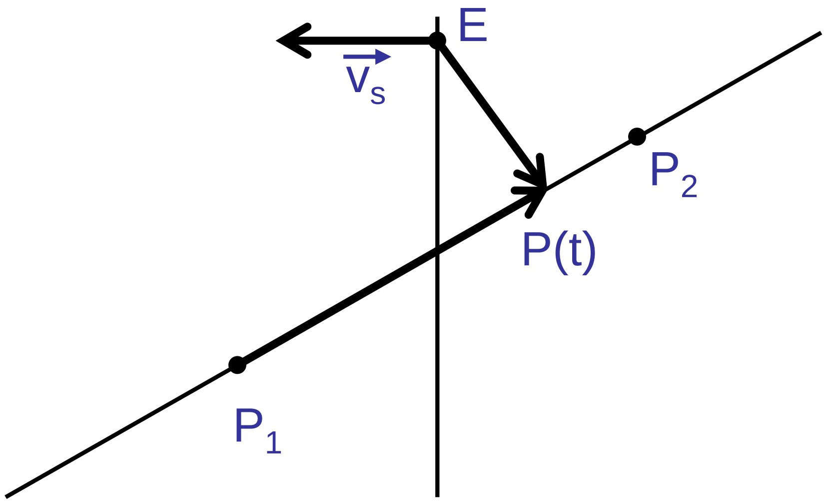
Line Intersections: Liang-Barsky

- find intersections of line segments with (axis-aligned) clip lines
- using the parametric line equation
  $P(t) = P_1 + \overrightarrow{(P_2 - P_1)}t$
- if $0 \leq t \leq 1$ then $P(t) \in$ line segment



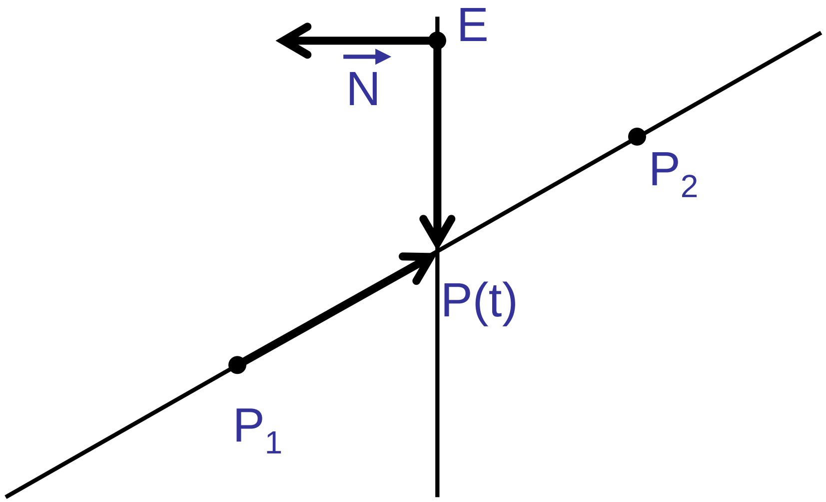
Line Intersections: Liang-Barsky

- additional point on the clip line: E
- consider the two vectors: $\overrightarrow{P(t) - E}$ and a vector $\perp$ to clip line $\rightarrow \vec{N}$
- P(t) is intersection point when both vectors perpendicular



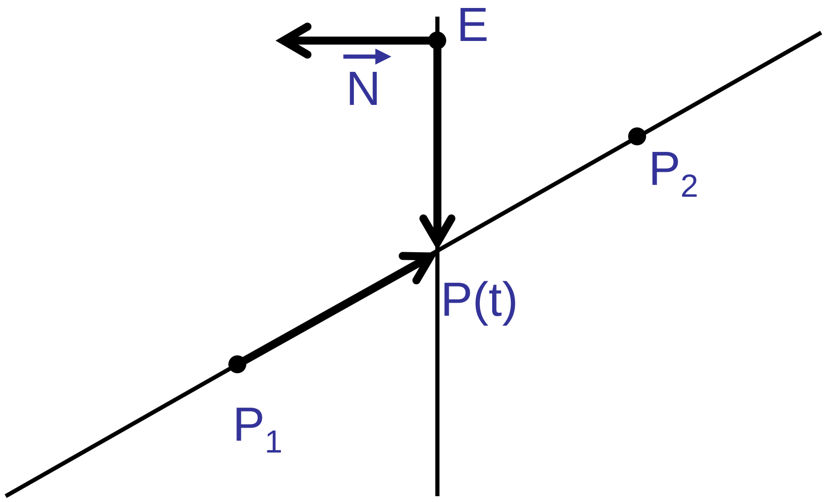
Line Intersections: Liang-Barsky

- additional point on the clip line: E
- consider the two vectors:
  $\overrightarrow{P(t) - E}$ and a vector $\perp$ to clip line $\rightarrow \vec{N}$
- P(t) is intersection point when both vectors perpendicular



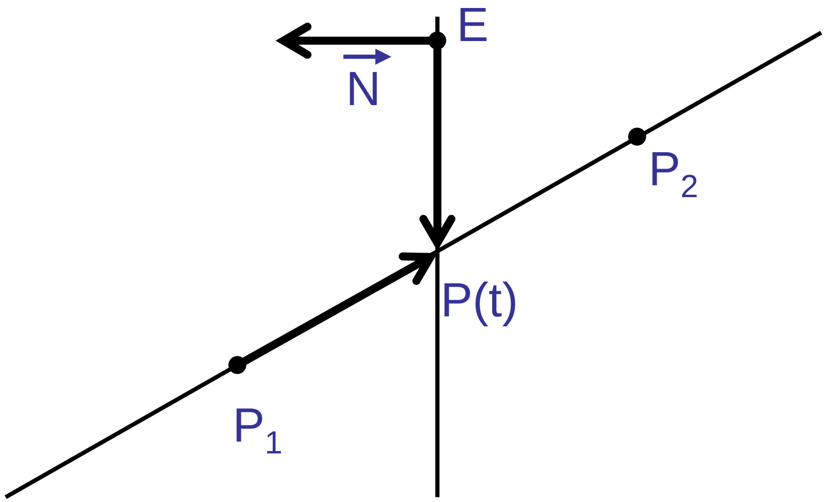
Line Intersections: Liang-Barsky

- $$\begin{aligned} 0 &= \langle \overrightarrow{N}, \overrightarrow{P(t) - E} \rangle \\ &= \langle \overrightarrow{N}, \overrightarrow{P_1 + (P_2 - P_1)t - E} \rangle \\ &= \langle \overrightarrow{N}, \overrightarrow{P_1 - E + (P_2 - P_1)t} \rangle \\ &= \langle \overrightarrow{N}, \overrightarrow{P_1 - E} \rangle + t \langle \overrightarrow{N}, \overrightarrow{(P_2 - P_1)} \rangle \end{aligned}$$



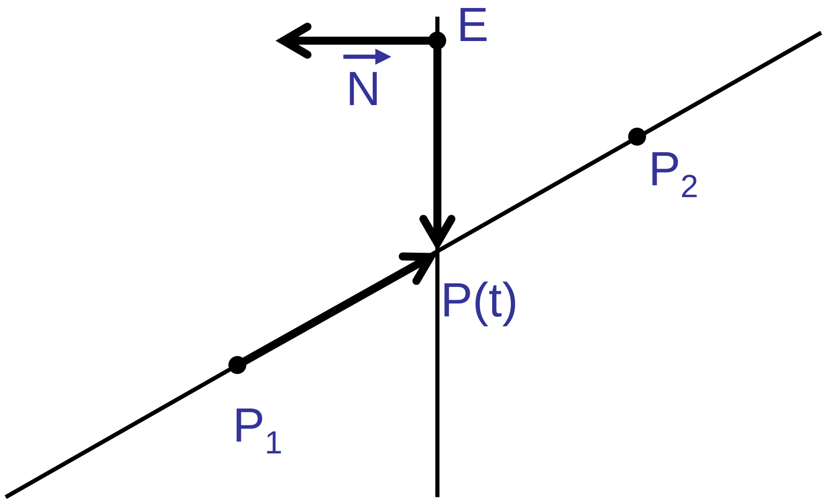
Line Intersections: Liang-Barsky

- $$t = \frac{\langle \vec{N}, \overrightarrow{E - P_1} \rangle}{\langle \vec{N}, \overrightarrow{(P_2 - P_1)} \rangle}$$
- efficiency? how many operations (2D)?
  - (2A to get vectors, 2M+1A for dot product) x 2
  - 6A + 4M + 1D in total



Line Intersections: Liang-Barsky

- $t = \dfrac{\langle \vec{N}, \overrightarrow{E - P_1} \rangle}{\langle \vec{N}, \overrightarrow{(P_2 - P_1)} \rangle}$ $\quad \vec{N} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
- efficiency? what if clip line is axis-aligned?
  – (1A to get vectors, nothing for dot product) x 2
  – 2A + 0M + 1D in total



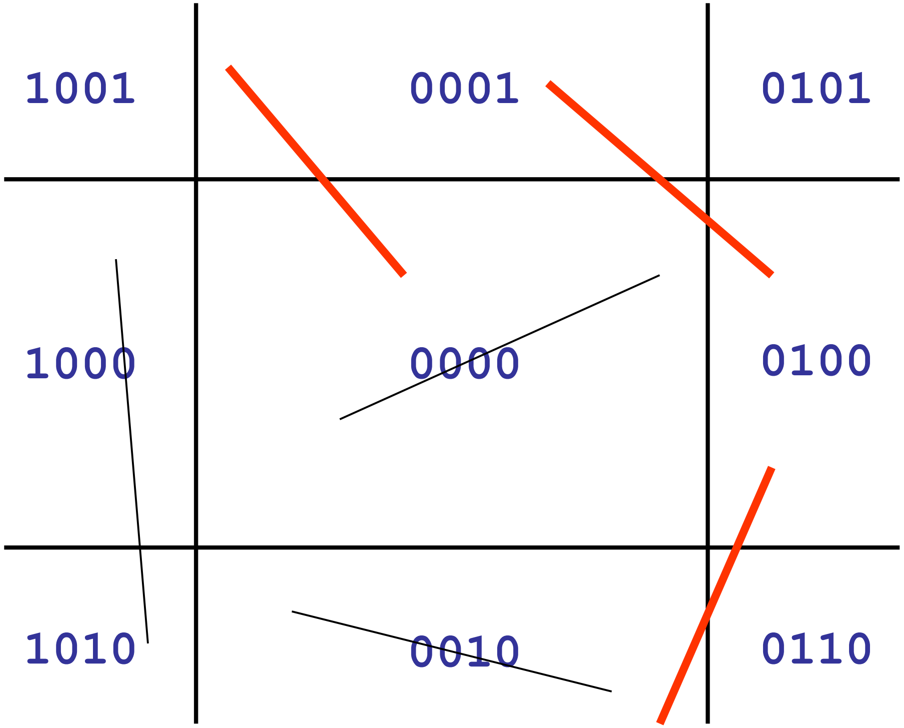
Clipping

Using both Algorithms:
Cohen-Sutherland + Liang-Barsky

Determining the Visible Parts

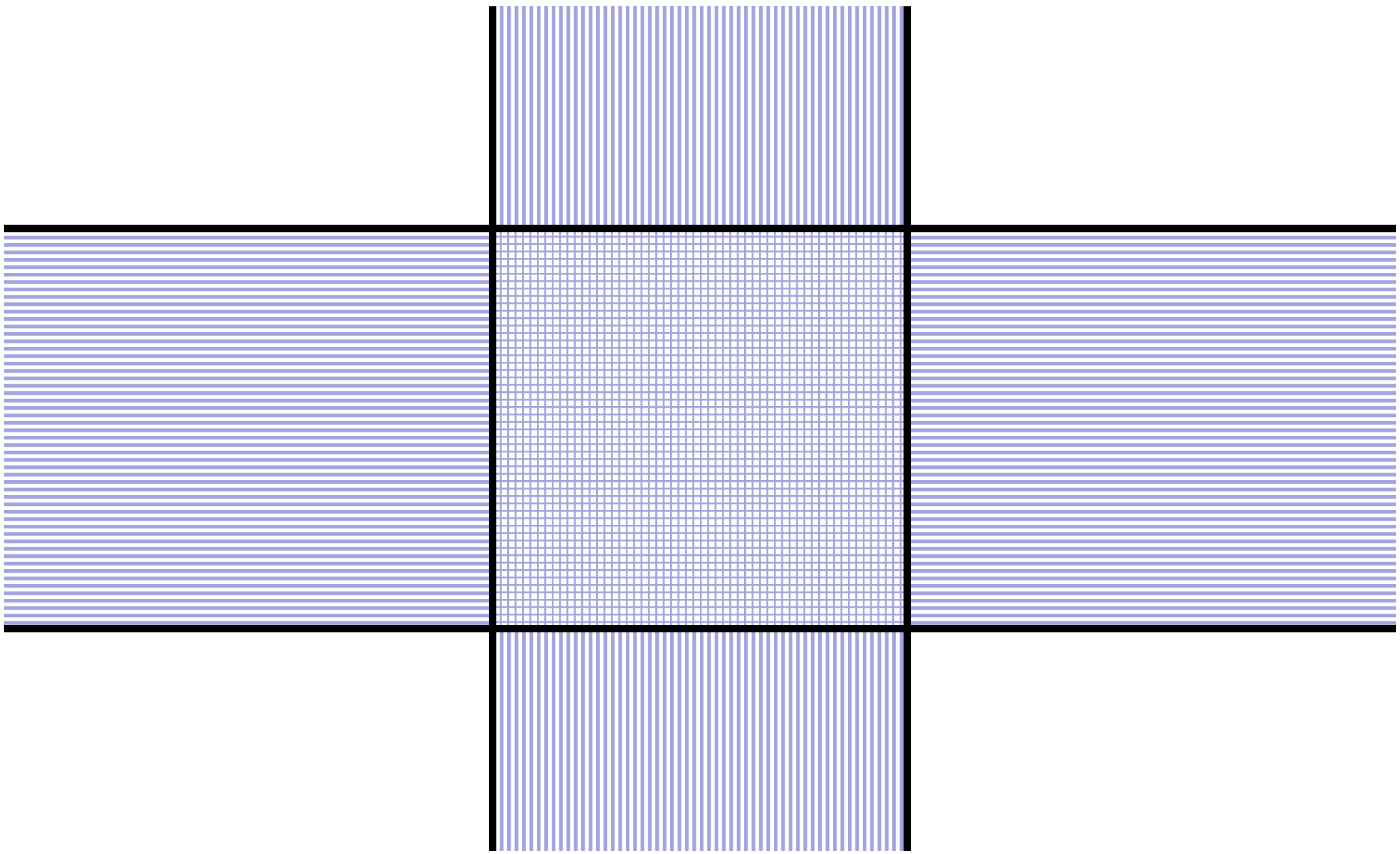
Having all intersections, what part to draw?

- determine **all 4** intersections, also for $t \notin [0, 1]$
- classify each intersection as entering or leaving (w.r.t. line direction)
- sort intersections depending on their t
- if 2nd is leaving & 3rd is entering then discard
- otherwise draw middle segment of line, $t \in [0, 1]$



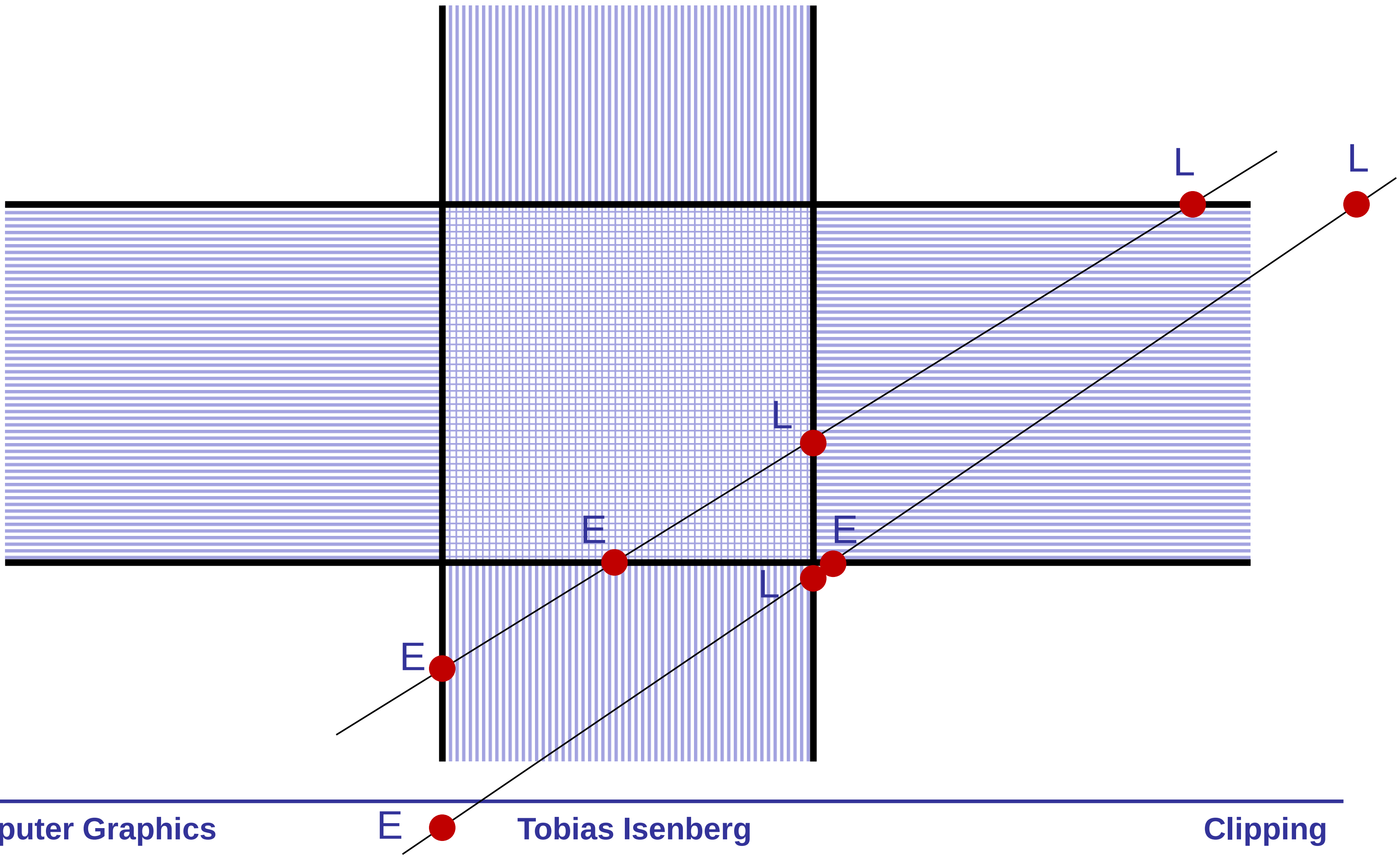
Determining the Visible Parts

- entering and leaving the “inside” zones



Determining the Visible Parts

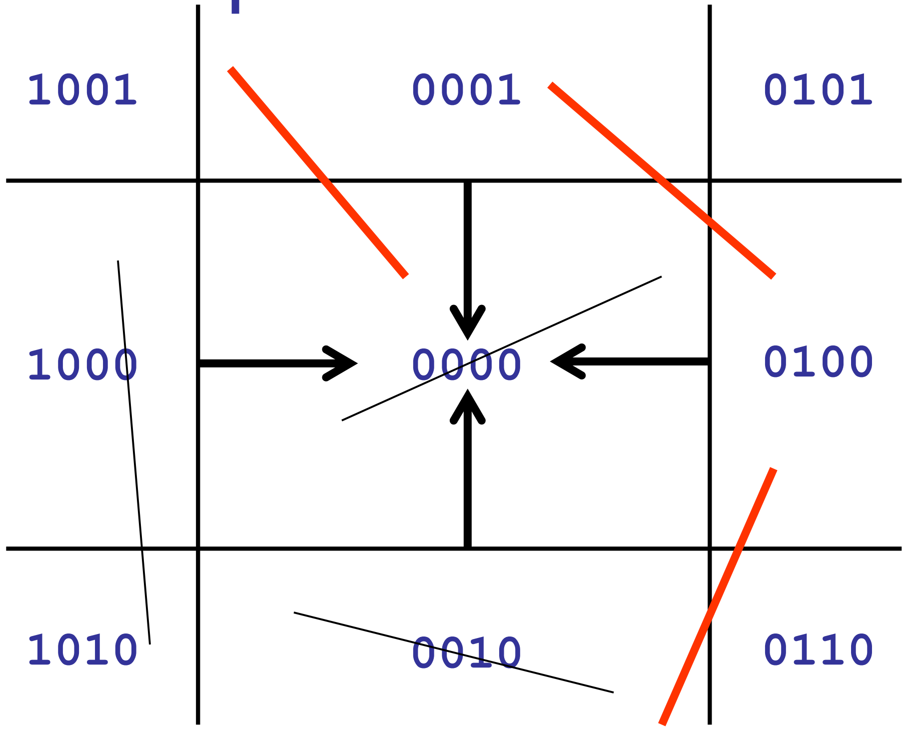
- comparison of both cases



Determining the Visible Parts

Classifying intersections:

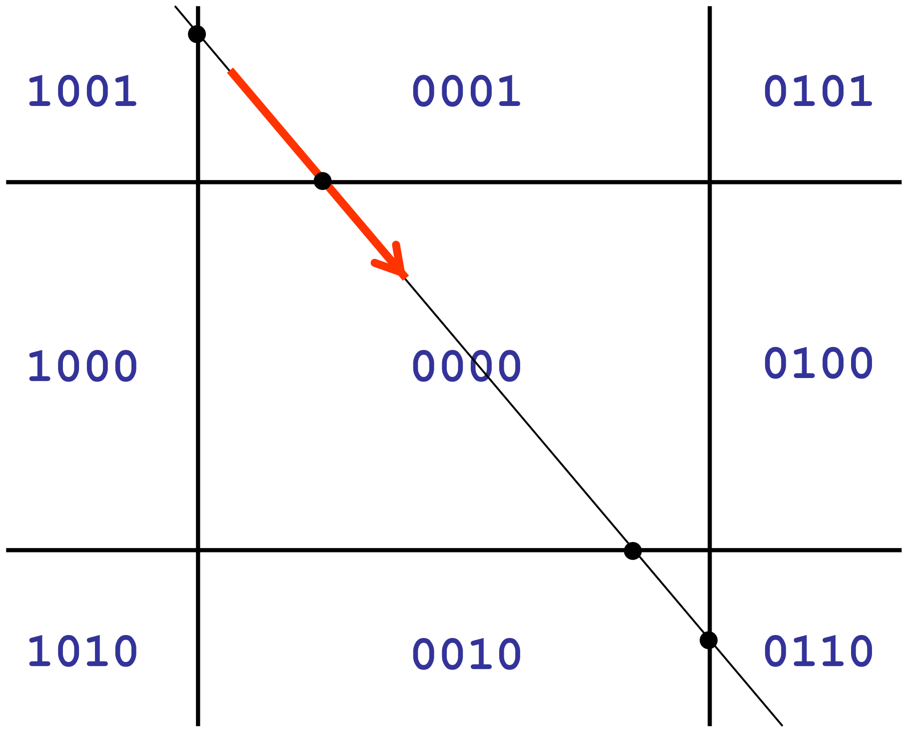
- using normal N for each clip line
- $\langle N, P_2 - P_1 \rangle > 0$ normal and line have same direction
- $\langle N, P_2 - P_1 \rangle < 0$ normal and line have different direction



Determining the Visible Parts

Example:

- 4 intersections:
  - $t_1 = -0.2$, entering
  - $t_2 = 0.5$, entering
  - $t_3 = 2.4$, leaving
  - $t_4 = 2.8$, leaving
- draw from t_2 to t_3
  $\rightarrow$ draw segment
  $t = 0.5$ to 1.0

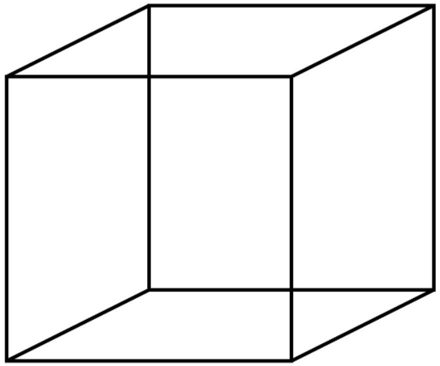


Overall Clipping Approach

1. Cohen-Sutherland (bitcodes) to trivially accept and reject some edges
2. Liang-Barsky to intersect the rest (derive 4 intersection points, t-values)
3. sort the points in ascending t order
4. draw if 2nd is entering and 3rd is leaving (only visible part, i.e. $t \in [0, 1]$)

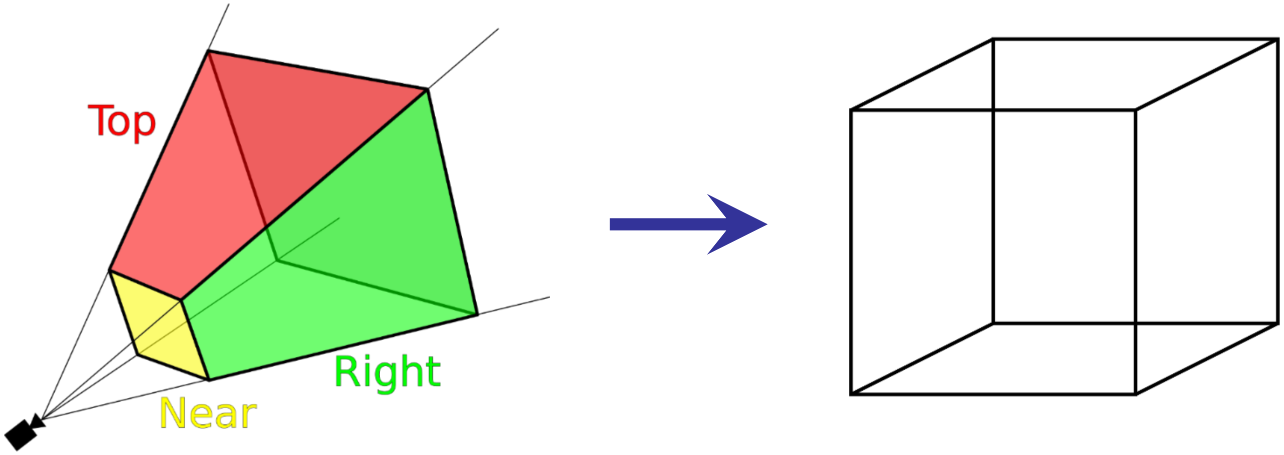
Cohen-Sutherland Algorithm in 3D

- extension to clipping in 3D easily possible
- six bit outcodes
  - previous four bits similar, now for peripheral clipping planes instead of edges (left, right, bottom, and top)
  - two more bits for front and back clipping planes



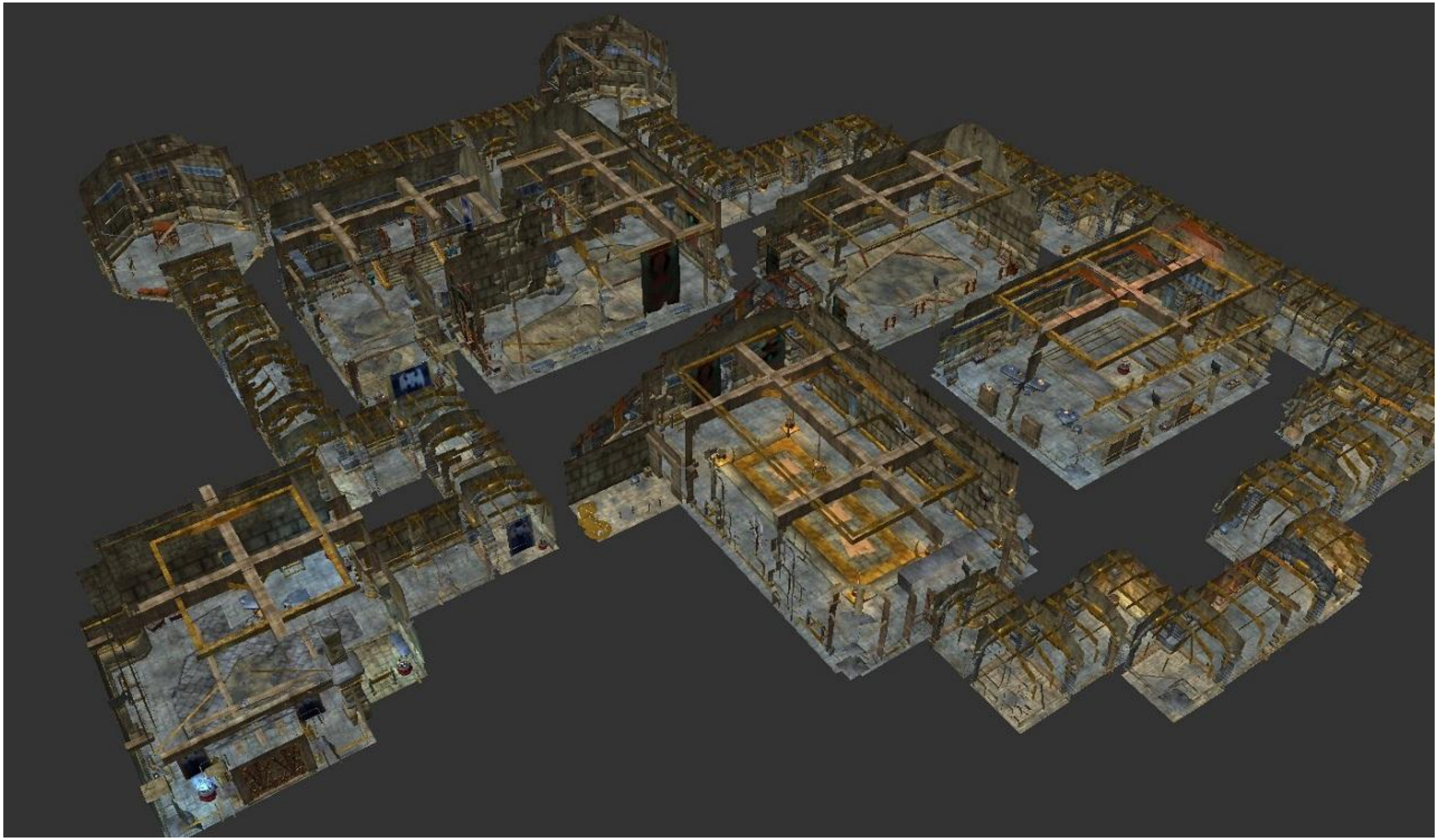
Clipping in 3D

- general vs. canonical view frustum: pyramid stump vs. axis-aligned box



  - 6 bits of Cohen-Sutherland for 6 sides of box
  - Liang-Barsky analogously to 2D: axis-aligned
  - thus whole algorithm analogous to 2D case

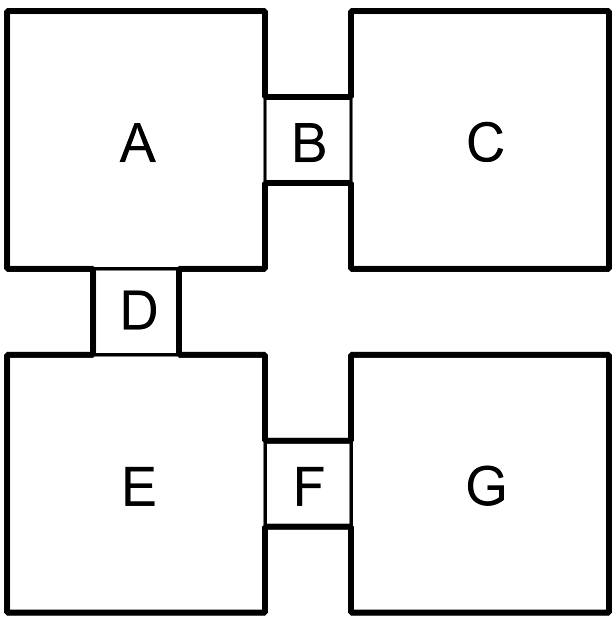
Advanced Clipping

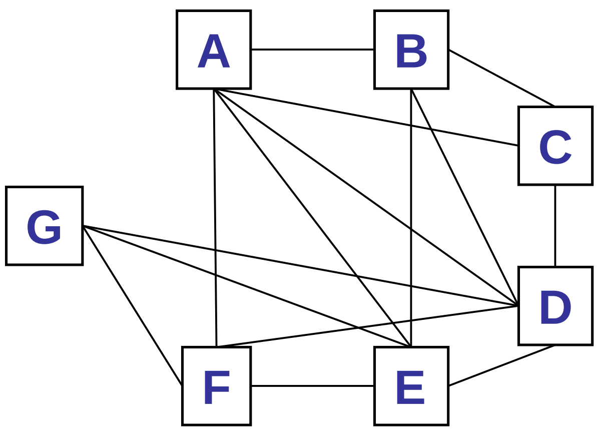


Advanced Clipping

Advanced Clipping: Portal Rendering

- core idea: easily reject large portions of geometry based on potential visibility (determined as pre-process)

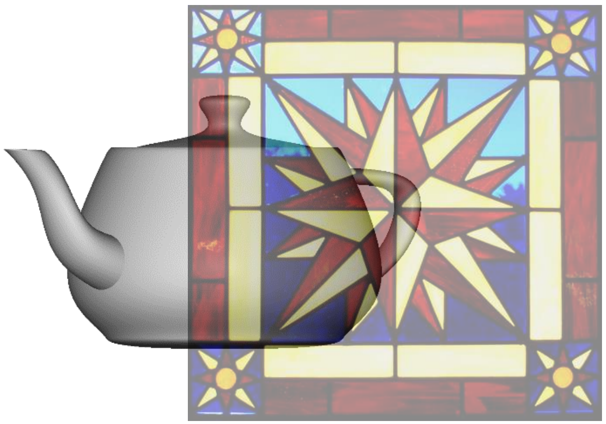


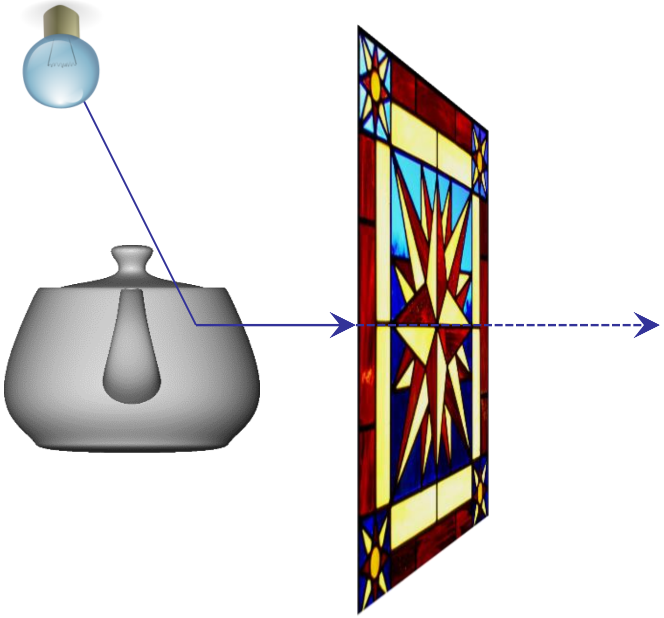


dynamic graph for fast look-up

Implication: Rendering transparency

- need to alpha-blend transparent object on top of objects behind it
- **sorting needed**! (z-buffer of transparent object prevents rendering behind it)

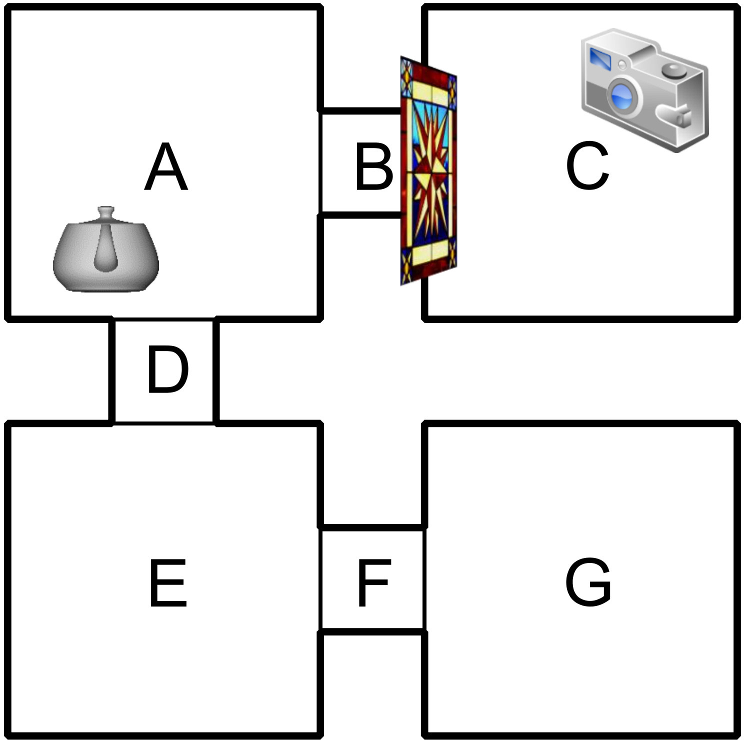




Implication: Rendering transparency

– sorting too expensive to do for all triangles

– also not needed: only behind / in front of



- use portal rendering for “sorting”:
  - render objects in A, B, D first (depending on portal status)
  - alpha-blend transparent object
  - render objects in C

Clipping

Polygon Clipping: Sutherland-Hodgman Algorithm

Sutherland-Hodgman algorithm

- by Ivan Sutherland and Gary W. Hodgman

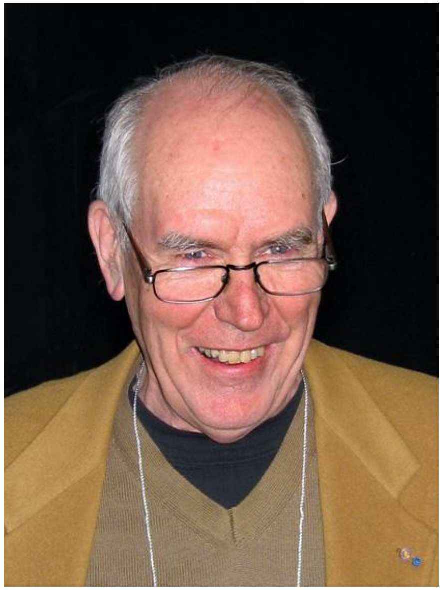
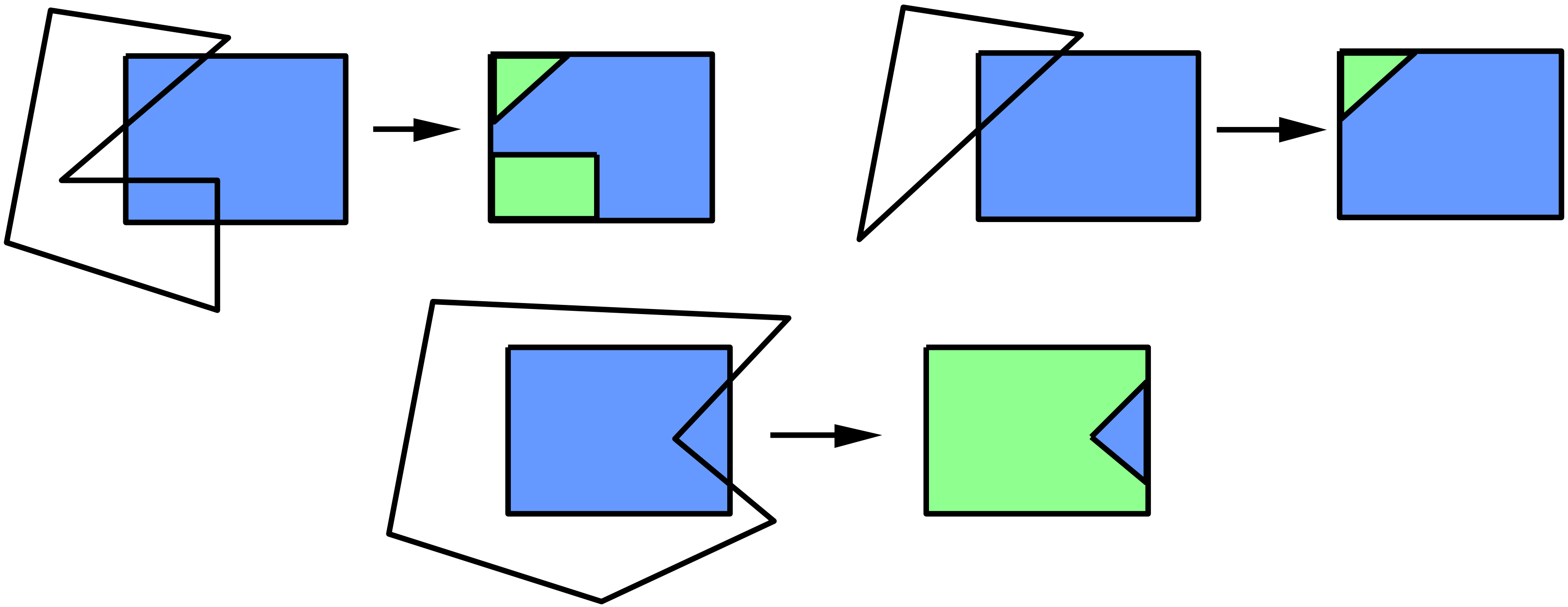


image: Dick Lyon

- created in 1974

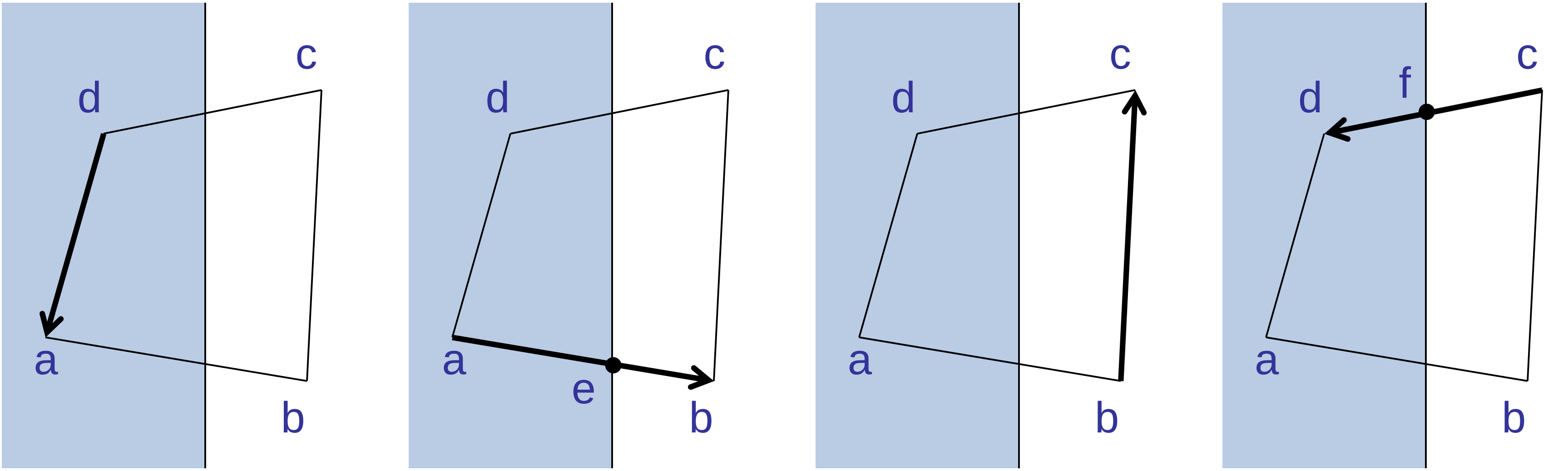
Sutherland-Hodgman Clipping

- arbitrary polygons can be clipped
- clip against each clip edge individually



Sutherland-Hodgman Algorithm

- four cases depending on edge location
- input vertex series → output vertex series
- # of vertices may change during process



output: a

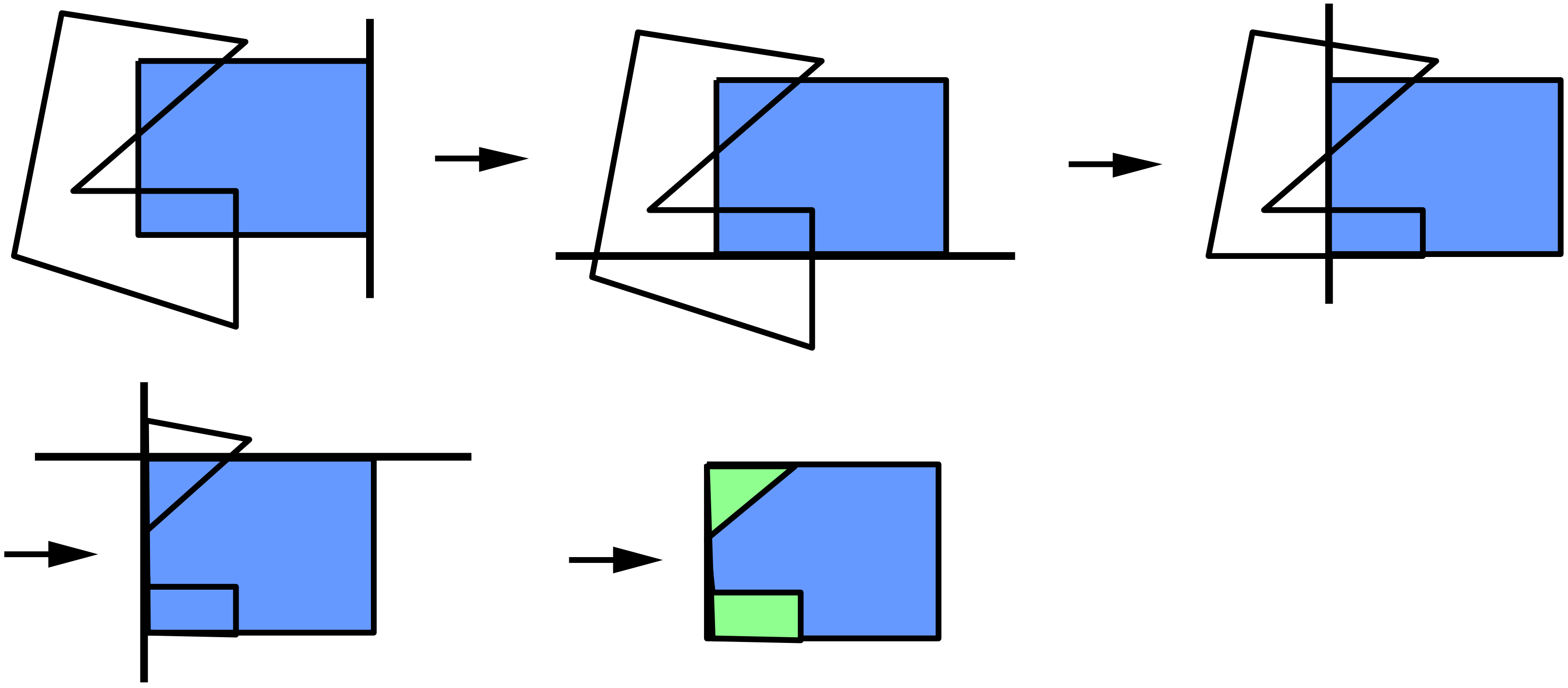
output: e

output: nothing

output: f, d

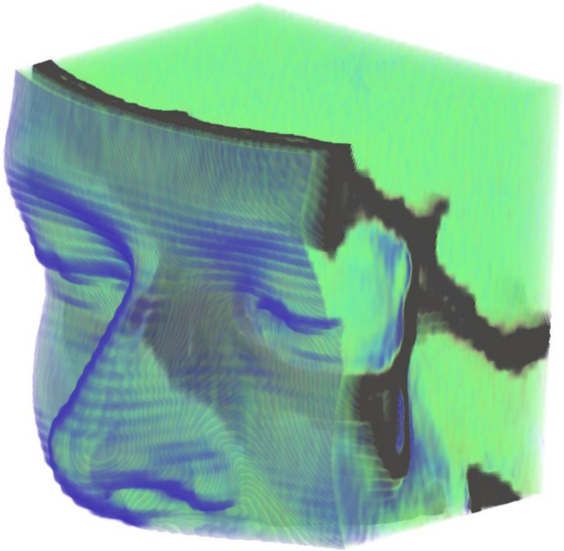
Sutherland-Hodgman Algorithm

- repeat process for each clip edge
- has been implemented in hardware



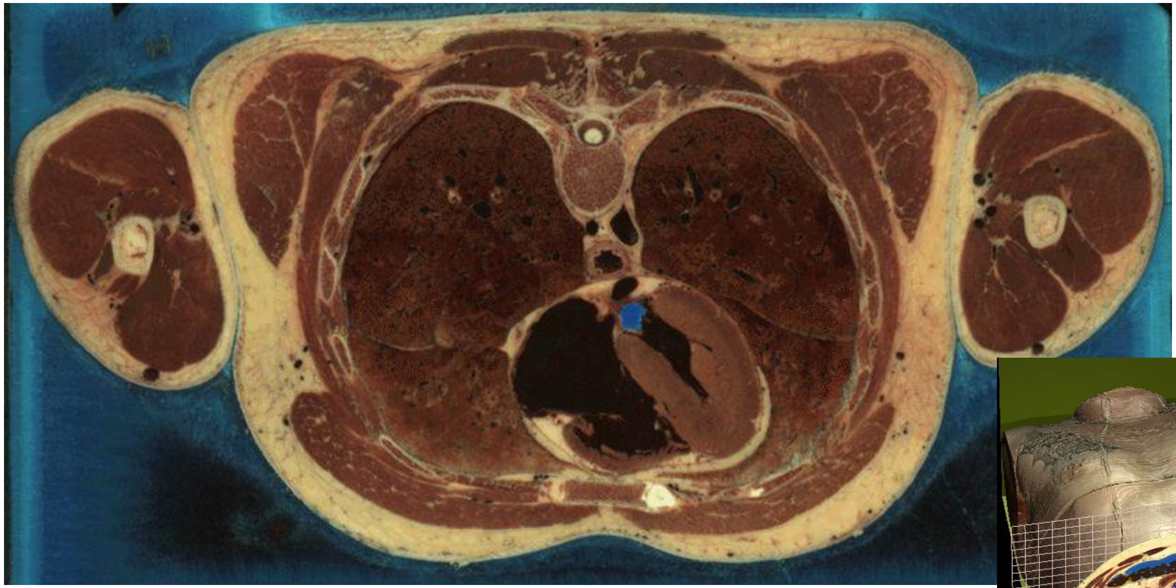
Related: Volume Data Clipping

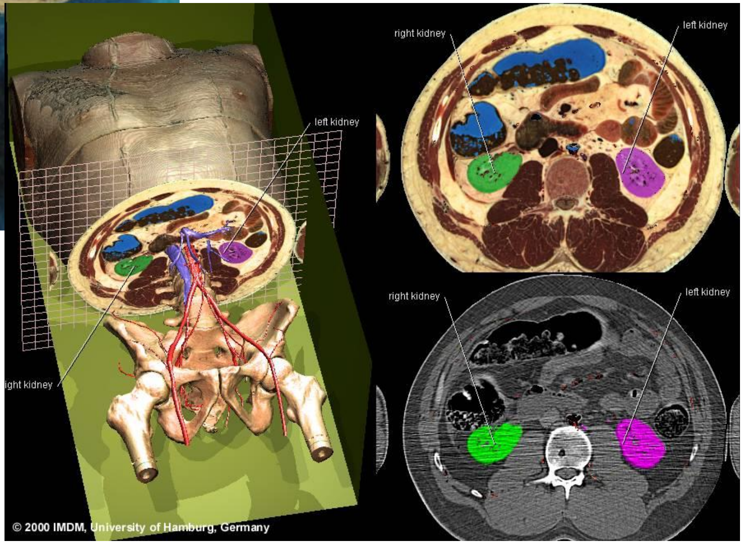
- showing insides of volumetric data sets
- MRI, CT, visual data
- clipping planes in 3D
- complete or partial clipping; 1–3 planes



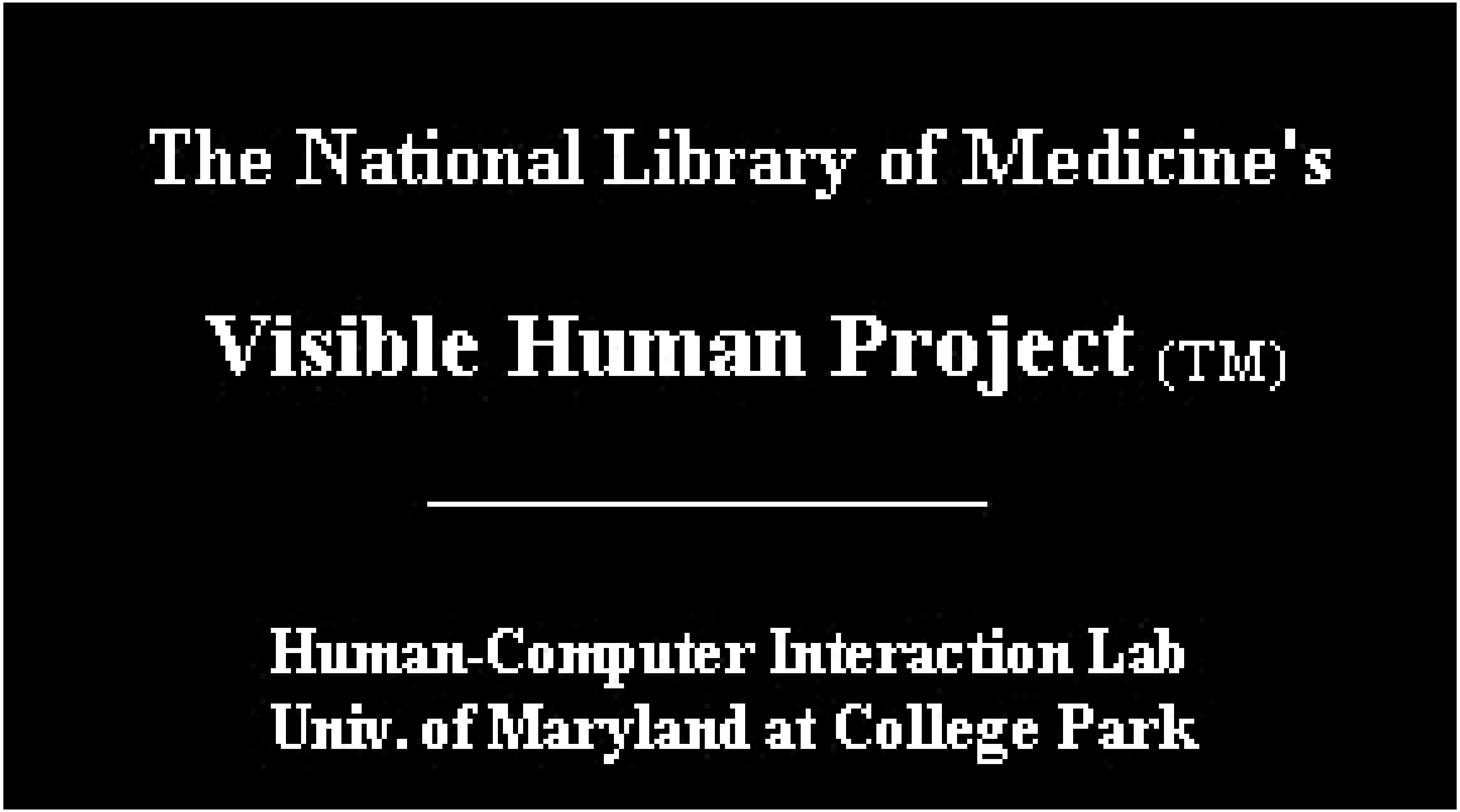
Related: Volume Data Clipping

- example: Visible Human Project



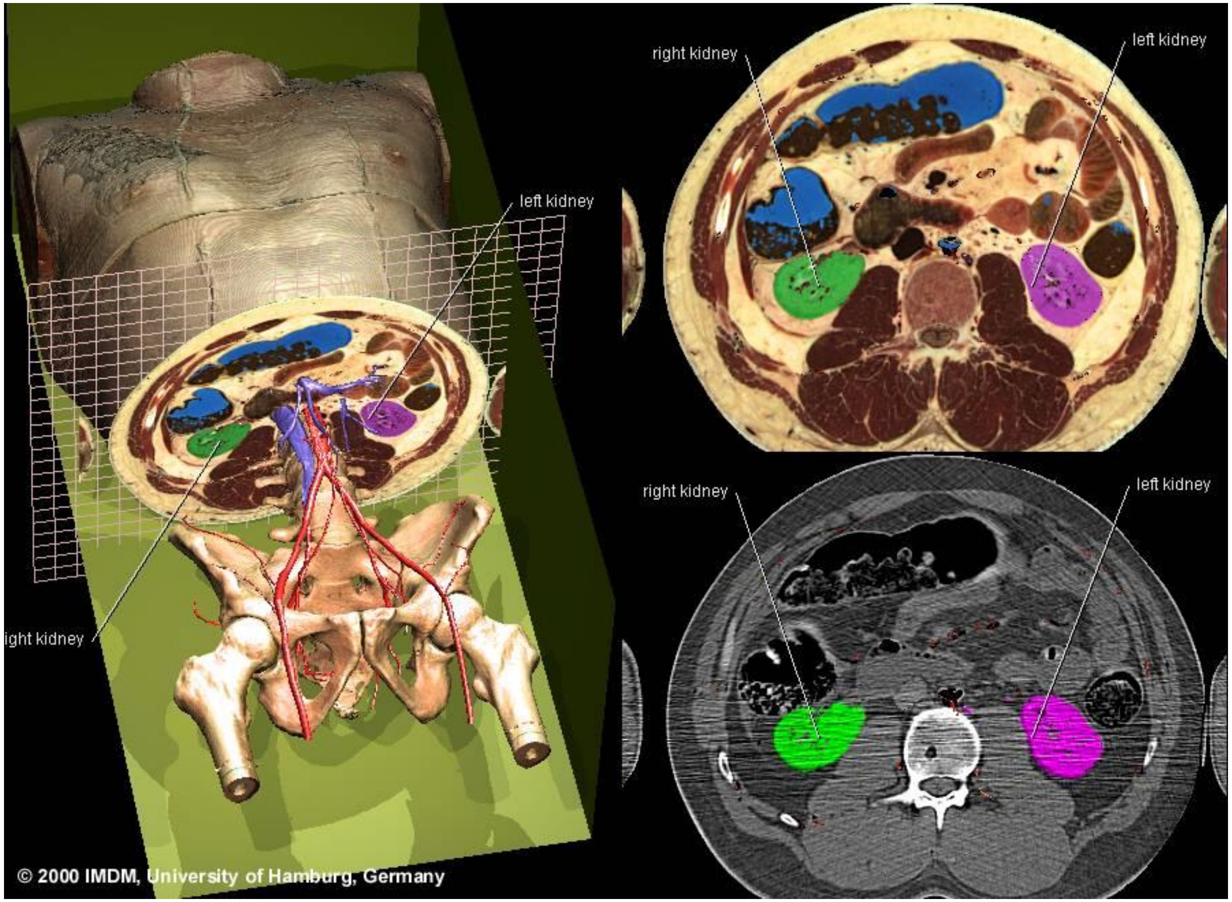


Related: Volume Data Clipping



Related: Volume Data Clipping

- example: Visible Human Project



© IMDM, University of Hamburg, Germany

Clipping Summary

- 3 step process for clipping line segments
  - trivially accept/reject segments using Cohen-Sutherland technique (outcodes)
  - determine all intersection points using Liang-Barsky technique
  - find part to be drawn using classifications
- clipping of complex polygons using Sutherland-Hodgman algorithm
- usage: clip 3D geometry on view frustum
- advanced clipping for larger scenes