Computer Graphics 6 - View Transformation and Clipping

Tom Thorne

Slides courtesy of Taku Komura
www.inf.ed.ac.uk/teaching/courses/cg

Overview

View transformation

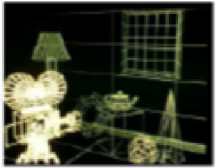
- Homogeneous coordinates recap
- Parallel projection
- Persepctive projection
- Canonical view volume

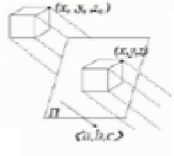
Clipping

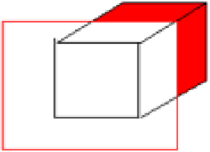
- Line and polygon clipping

Outline of procedure

1. Transform from world to *camera* coordinates
2. Project into *view volume* or *screen coordinates*
3. Clip geometry that falls outside of the *view volume*

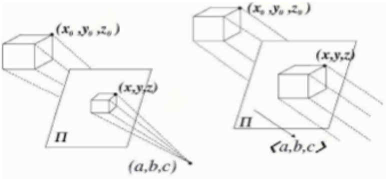


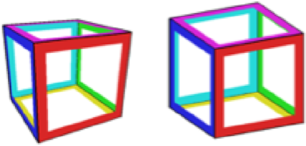




View projection

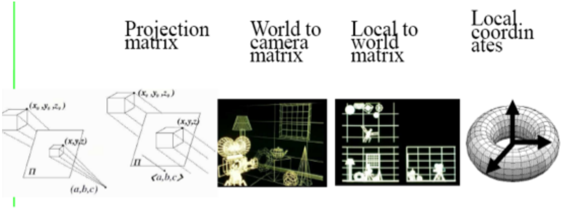
- Homogeneous transformation
- Parallel projection
- Perspective projection
- Canonical view volume





Homogeneous transformations

$$v = M_{proj} M_{c \leftarrow g} M_{g \leftarrow l} v_l$$

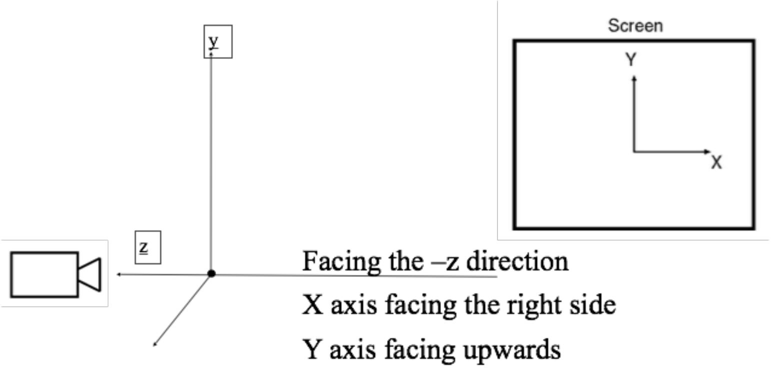


Homogeneous coordinates

- Allows us to represent translation, rotation and scaling by matrix multiplication
- Coordinates (x, y, z, w) and (cx, cy, cz, cw) represent the same point
- Return to Cartesian coordinates using $(x', y', z') = (x/w, y/w, z/w)$

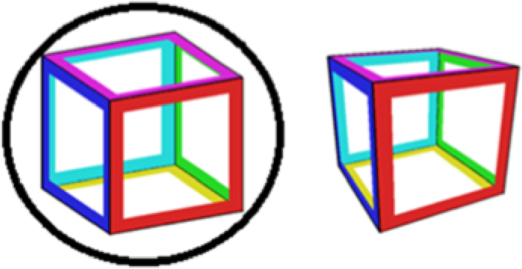
Camera coordinate system

- As used by OpenGL



Parallel (orthographic) projection

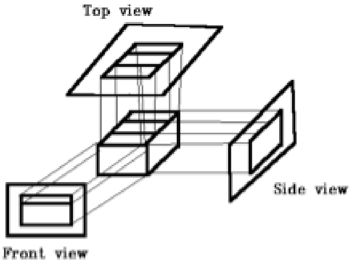
- Specified by a direction of projection
- No change in size of objects



Parallel projection

Project coordinates onto plane at $z = 0$

$x' = x$, $y' = y$, $z' = 0$

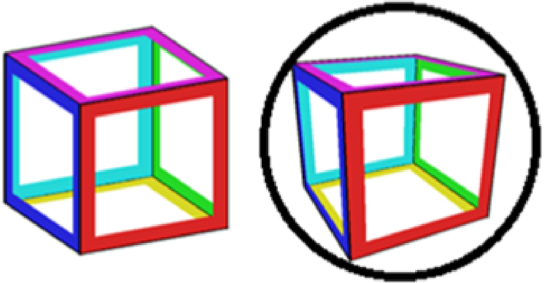


$$M_{orth} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

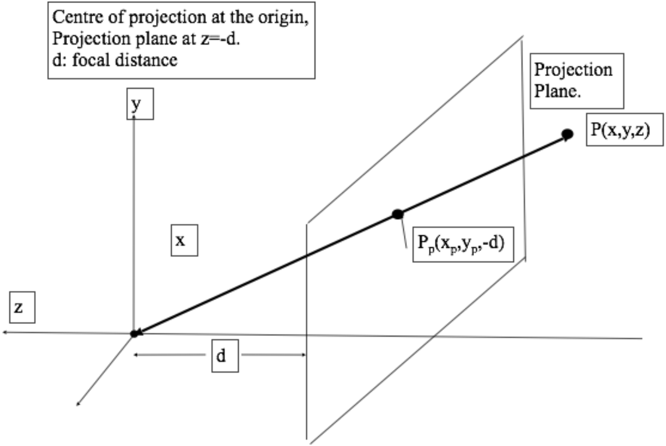
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix}$$

Perspective projection

- Far away objects appear smaller
- Specified by a center of projection and focal distance



Perspective projection

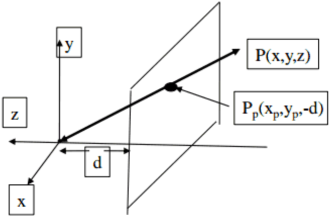


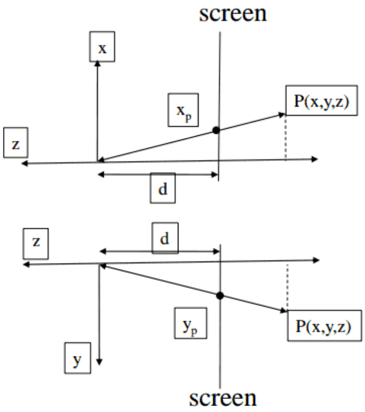
Perspective projection - simple case

From similar triangles:

$$\frac{x_p}{d} = \frac{x}{-z};\ \frac{y_p}{d} = \frac{y}{-z}$$

$$x_p = \frac{d \cdot x}{-z} = \frac{x}{-z/d};\ y_p = \frac{d \cdot y}{-z} = \frac{y}{-z/d}$$





Perspective projection

$$\begin{pmatrix} x' \\ y' \\ -d \\ 1 \end{pmatrix} = \begin{pmatrix} -d\frac{x}{z} \\ -d\frac{y}{z} \\ -d \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ -z/d \end{pmatrix}$$

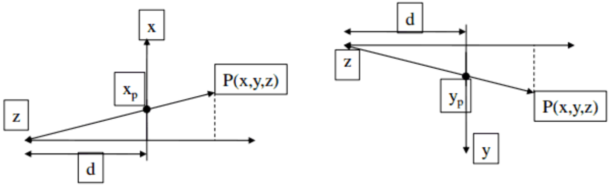
Represented as a matrix multiplication:

$$M_{proj} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{d} & 0 \end{pmatrix}$$

Perspective projection

$$V_p = M_{proj} V_c = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{d} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ -\frac{z}{d} \end{pmatrix} = \begin{pmatrix} \frac{x}{-z/d} \\ \frac{y}{-z/d} \\ -d \end{pmatrix}$$

Alternative formulation



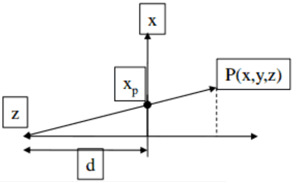
Projection plane at z = 0
Centre of projection at
z = d

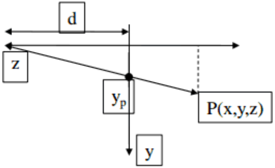
$$\frac{x_p}{d} = \frac{x}{-z+d}; \quad \frac{y_p}{d} = \frac{y}{-z+d}$$

Multiply by d

$$x_p = \frac{d \cdot x}{-z+d} = \frac{x}{(-z/d)+1}; \quad y_p = \frac{d \cdot y}{-z+d} = \frac{y}{(-z/d)+1}$$

Alternative formulation





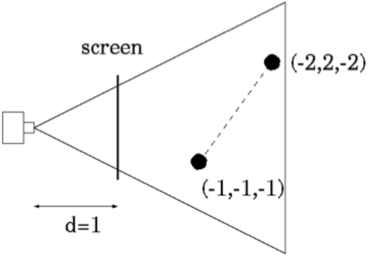
Projection plane at z = 0,
Centre of projection at
z = d

Now we can allow d→∞

$$M_{proj} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-1}{d} & 1 \end{pmatrix}$$

Exercise

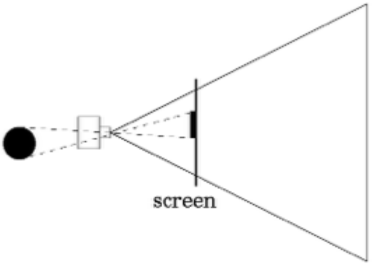
Where will the two points be projected?



$$M_{proj} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{d} & 1 \end{pmatrix}$$

Problems

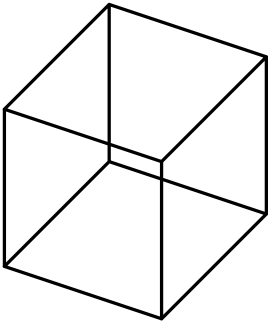
- After projection depth information is lost (this is needed for hidden surface removal)
- Objects behind the camera are projected infront of the camera!

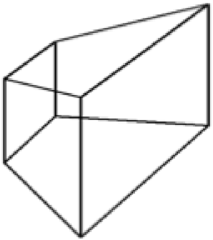


3D view volume

Define a volume in which objects are visible

- For parallel projection, a box
- For perspective projection, a frustum
- Surfaces outside this volume are clipped or removed and not drawn

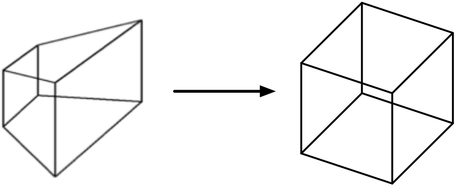




Canonical view volume

It can be computationally expensive to check if a point is inside a frustum

- Instead transform the frustum into a normalised canonical view volume
- Uses the same ideas a perspective projection
- Makes clipping and hidden surface calculation much easier

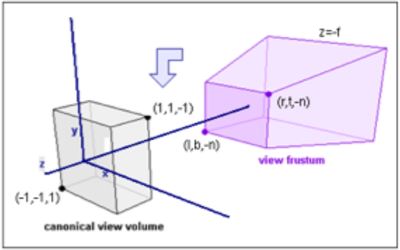


Transforming the view frustum

The frustum is defined by a set of parameters, l, r, b, t, n, f:

l	Left x coordinate of near plane
r	Right x coordinate of near plane
b	Bottom y coordinate of near plane
t	Top y coordinate of near plane
n	Minus z coordinate of near plane
f	Minus z coordinate of far plane

With $0 < n < f$.



Transforming the view frustum

We can transform the perspective canonical view volume to the parallel view volume using:

If $z \in [-n, -f]$ *and* $0 < n < f$ *then*

$$M_{can} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Final steps

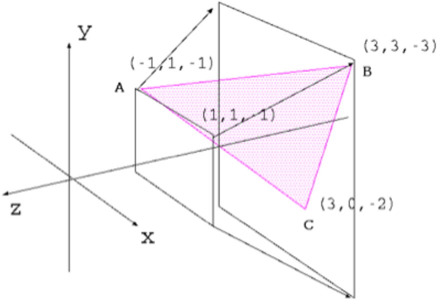
- After transformation to canonical view volume, divide by w to produce 3D Cartesian coordinates
- Perform clipping in the canonical view volume:

$$-1 \leq x \leq 1, -1 \leq y \leq 1, -1 \leq z \leq 1$$

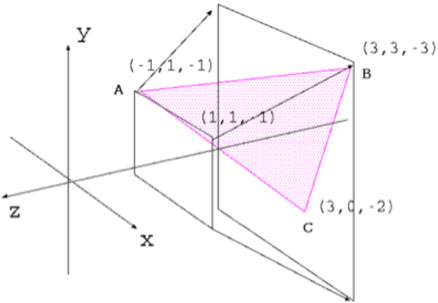
Example

What does ABC look like after projection?

$$M_{can} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$



Example



$M_{can} =$

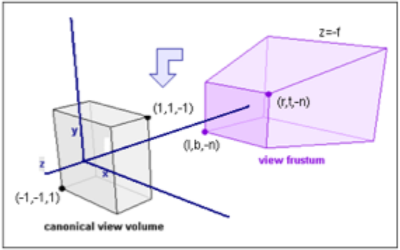
$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & -3 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$n = 1, f = 3, r = 1, l = -1, t = 1, b = -1$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & -3 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 0 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} \frac{3}{2} \\ 0 \\ \frac{1}{2} \\ 1 \end{pmatrix}$$

Projection summary

- Parallel and perspective projection
- Projection matrices transform points to 2D coordinates on the screen
- Canonical view volumes can be used for clipping



Overview

View transformation

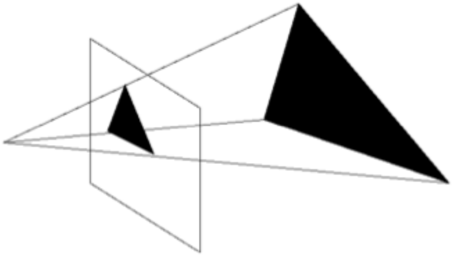
- Homogeneous coordinates recap
- Parallel projection
- Persepctive projection
- Canonical view volume

Clipping

- Line and polygon clipping

Projection of polygons and lines

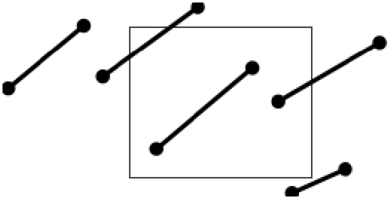
- Lines in 3D become lines in 2D
- Polygons in 3D become polygons in 2D

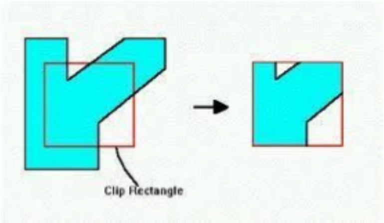


Clipping

They may intersect the canonical view volume, then we need to perform clipping:

- Clipping lines (Cohen-Sutherland algorithm)
- Clipping polygons (Sutherland-Hodgman algorithm)

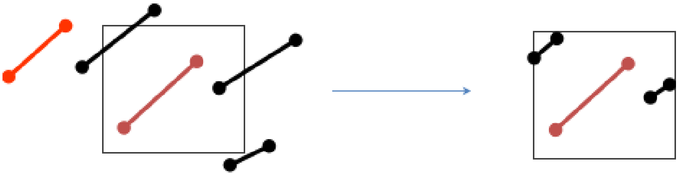




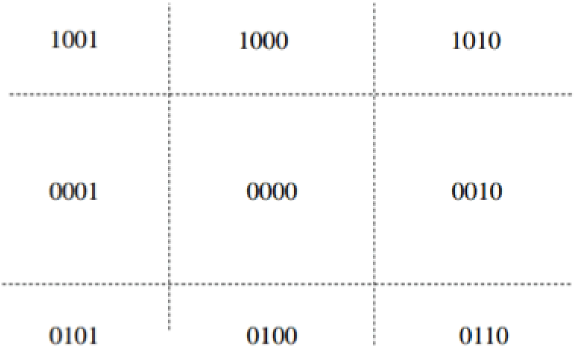
Cohen-Sutherland algorithm

Input Screen and 2D line segment

Output Clipped 2D line segment



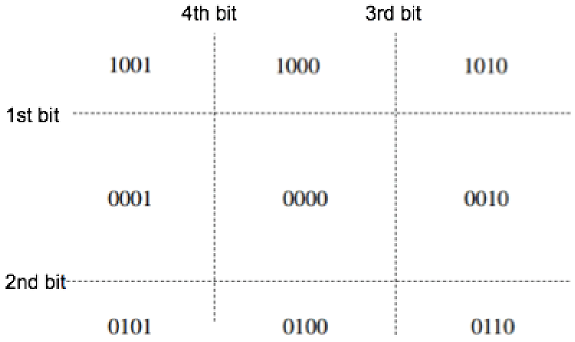
Cohen-Sutherland algorithm



Outcodes:

- Space is split into nine regions
- The center corresponds to the area visible on the screen
- Each region is encoded by four bits

Cohen-Sutherland algorithm



Outcodes:

1st bit	Above top of screen ($y > y_{max}$)
2nd bit	Below bottom of screen ($y < y_{min}$)
3rd bit	Right of right edge of screen ($x > x_{max}$)
4th bit	Left of left edge of screen ($x < x_{min}$)

Cohen-Sutherland algorithm

```
Data: Line from coordinate A to B
Result: Clipped line
while true do
    Calculate outcodes of endpoints;
    if trivial accept or trivial reject then
        return accepted line or reject;
    else
        Find which endpoint is outside clipping rectangle;
        Clip outside endpoint to edge of first non-zero outcode bit
        and update endpoint coordinates;
    end
end
```

Bitwise AND and OR

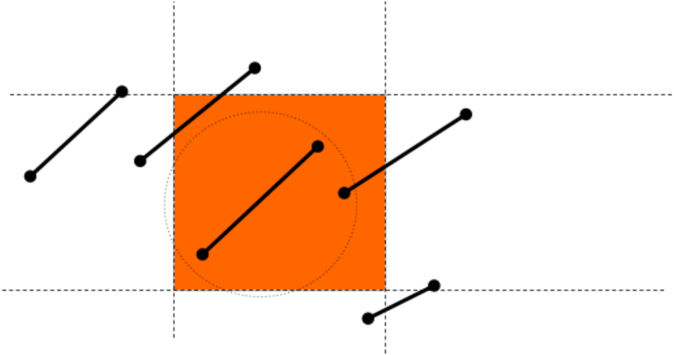
$$\begin{array}{ccccc} 0 & \text{OR} & 0 & = & 0 \\ 0 & \text{OR} & 1 & = & 1 \\ 1 & \text{OR} & 0 & = & 1 \\ 1 & \text{OR} & 1 & = & 1 \end{array}$$

$$\begin{array}{ccccc} 0 & \text{AND} & 0 & = & 0 \\ 0 & \text{AND} & 1 & = & 0 \\ 1 & \text{AND} & 0 & = & 0 \\ 1 & \text{AND} & 1 & = & 1 \end{array}$$

Trivial accept

All line endpoints are inside the screen

- Apply an OR to outcodes of the line endpoints, test for equal to 0



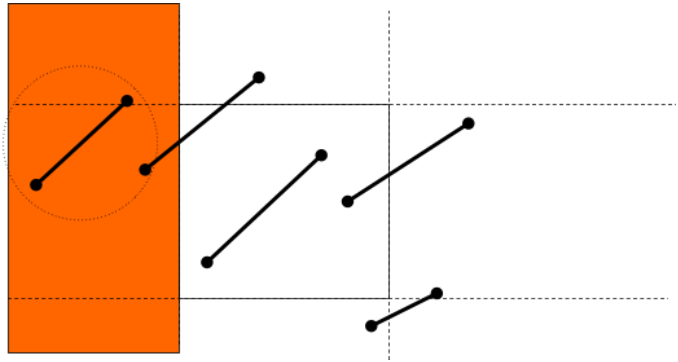
Cohen-Sutherland algorithm

```
Data: Line from coordinate A to B
Result: Clipped line
while true do
    Calculate outcodes of endpoints;
    if trivial accept or trivial reject then
        return accepted line or reject;
    else
        Find which endpoint is outside clipping rectangle;
        Clip outside endpoint to edge of first non-zero outcode bit
        and update endpoint coordinates;
    end
end
```

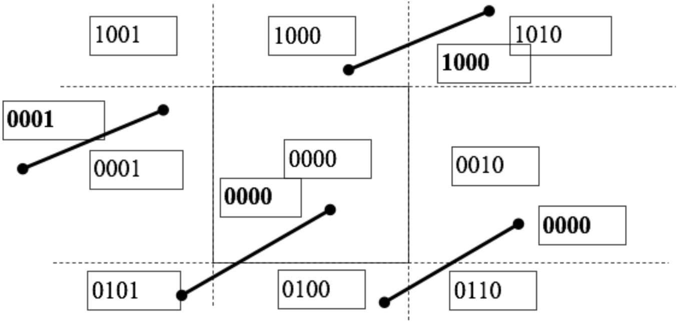
Trivial reject

Both line endpoints are outside of the screen on the same side

- Apply AND operation to the two endpoints and reject if not 0

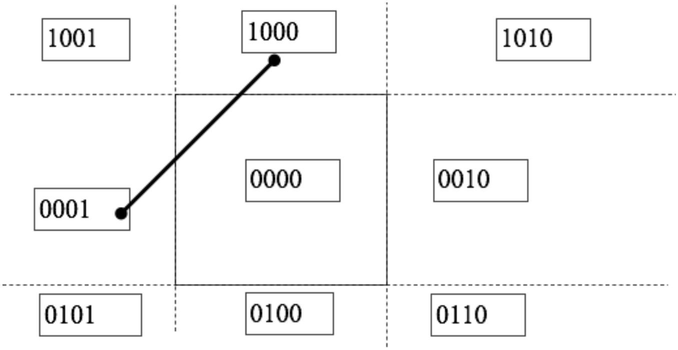


Cohen-Sutherland algorithm



Logical AND between codes for 2 endpoints,
Reject line if non-zero – trivial rejection.

Cohen-Sutherland algorithm



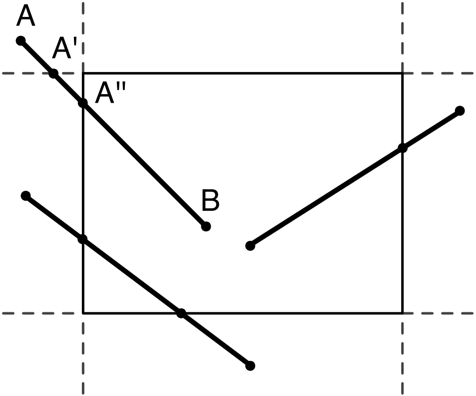
Logical AND between codes for 2 endpoints,
Reject line if non-zero – trivial rejection.

Cohen-Sutherland algorithm

```
Data: Line from coordinate A to B
Result: Clipped line
while true do
    Calculate outcodes of endpoints;
    if trivial accept or trivial reject then
        return accepted line or reject;
    else
        Find which endpoint is outside clipping rectangle;
        Clip outside endpoint to edge of first non-zero outcode bit
        and update endpoint coordinates;
    end
end
```

Line intersection

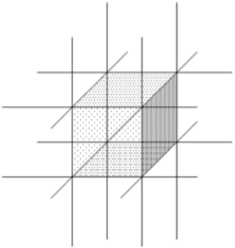
- Need to clip a line to the edges of the rectangle
- Find an endpoint which is outside the rectangle
- Select edge to clip based on the outcode, split the line and feedback into algorithm



Cohen-Sutherland algorithm

Extension to 3D

- Clip lines to near and far planes of view volume
- Needs more bits for the outcode



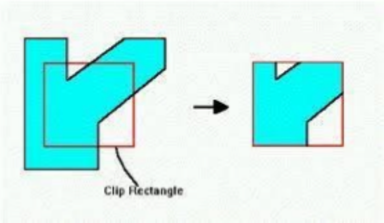
Polygon clipping

Sutherland-Hodgman algorithm

Input 2D polygon

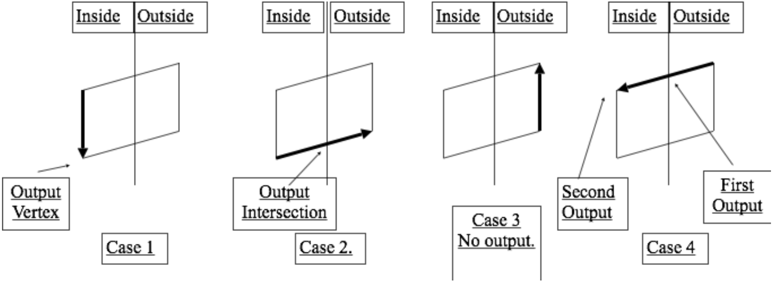
Output List of clipped vertices

Traverse the polygon and clip at each edge of the screen



Sutherland-Hodgman algorithm

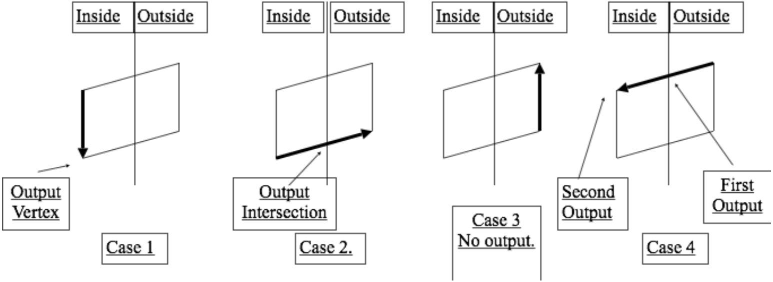
Traverse edges and divide into four types:



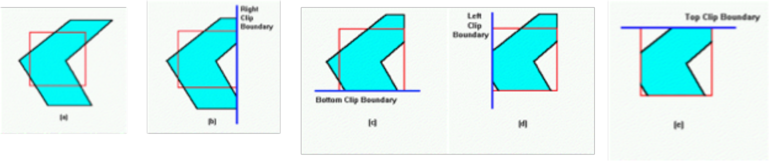
Sutherland-Hodgman algorithm

For each edge of the clipping rectangle:

- For each polygon edge between v_i and v_{i+1}



Example



(a)
Right
Clip
Boundary
(b)
Bottom Clip Boundary
(c)
Left
Clip
Boundary
(d)
Top Clip Boundary
(e)

Summary

Projection

- Parallel and perspective
- Canonical view volumes

Clipping

- Cohen-Sutherland algorithm
- Sutherland-Hodgman algorithm

References

View transformation

- Shirley, Chapter 7
- Foley, Chapter 6

Clipping

- Foley Chapter 3.12, 3.14
- `http://www.cc.gatech.edu/grads/h/Hao-wei.Hsieh/Haowei.Hsieh/mm.html`