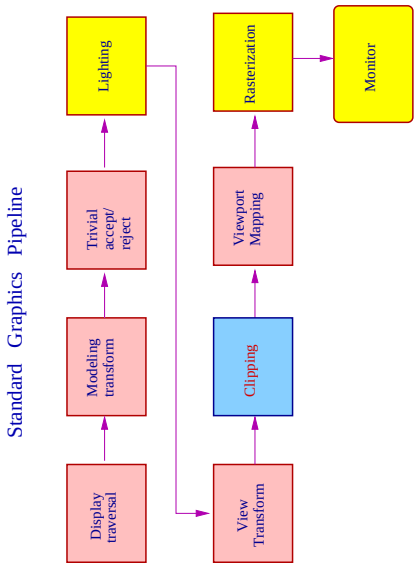
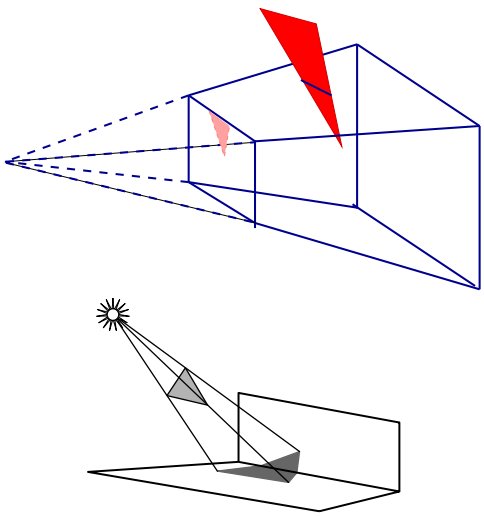
Graphics Pipeline: Clipping



Slide 1

Clipping: Introduction



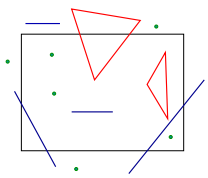
- ☆ Clip the objects within the view volume
- ☆ Clip the objects inside other regions
- ☆ Shadow generation

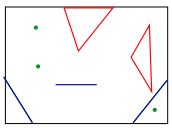
Slide 2

2D Clipping

W: clipping window

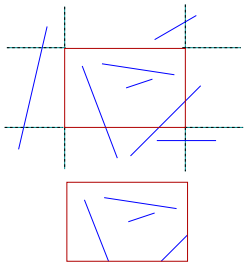
- ☆ Point clipping
  - $x_L \le x \le x_R$,
  - $y_L \le y \le y_R$.
- ☆ Line clipping
  - Cohen-Sutherland
  - Liang-Barsky
- ☆ Polygon clipping
  - Sutherland-Hodgeman
  - Weiler-Atherton
  - Weiler





Slide 3

Line Clipping



e: Segment with endpoints p and q.

$p \in W, q \in W$: Accept e.

$p \in W, q \notin W$: Compute $\sigma = e \cap \partial W$, accept $p\sigma$.

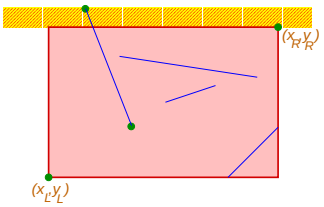
$p \notin W, q \in W$: Compute $\sigma = e \cap \partial W$, accept σq.

$p \notin W, q \notin W$:

- ☆ p, q lie oustide the same boundary line of W, reject e.
- ☆ Otherwise, a more complicated test.

Slide 4

Cohen-Sutherland Algorithm



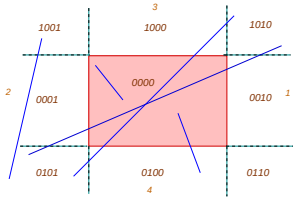
ℓ: Line supporting an edge of W,
outer halfplane: Halfplane bounded by ℓ and not containing W.

- ☆ A 4-bit *outer code* $OC(p)$ $a_4a_3a_2a_1$ for each point p.
- ☆ One bit for each outer halfplane.
- ☆ Bit is 1 if p lies in that outer halfplane.

bit 1	a_1	left	$x < x_L$	ℓ_1
bit 2	a_2	right	$x > x_R$	ℓ_2
bit 3	a_3	bottom	$y < y_L$	ℓ_3
bit 4	a_4	top	$y > y_R$	ℓ_4

Slide 5

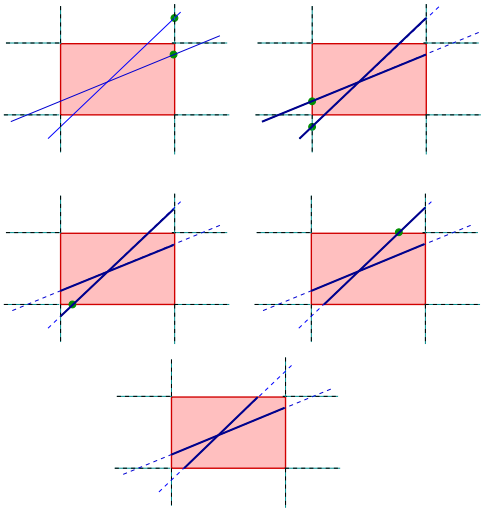
Cohen-Sutherland Algorithm



- ☆ $OC(p)$ OR $OC(q) = 0 \Rightarrow$ accept e.
- ☆ $OC(p)$ AND $OC(q) \neq 0 \Rightarrow$ reject e.
- ☆ Otherwise, do the following
  - If $OC(p) = 0$, then `swap` (p, q)
  - Find the rightmost bit $OC_i(p) = 1$
  - Compute $\sigma = e \cap \ell_i$
  - Set $p = \sigma$, compute $OC(p)$
- ☆ Repeat the above steps until e accepted/rejected.

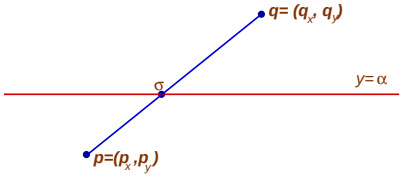
Slide 6

Cohen-Sutherland Algorithm



Slide 7

Computing the Intersection



Parametric representation of a line

$$\begin{aligned} \mathbf{e} &= \mathbf{p} + t(\mathbf{q} - \mathbf{p}) \quad t \in \mathbb{R}. \\ x &= p_x + t(q_x - p_x), \\ y &= p_y + t(q_y - p_y) \end{aligned}$$

$p : t = 0 \quad q : t = 1.$

$\sigma = (\sigma_x, \sigma_y)$: Intersection point of L and $y = \alpha$.

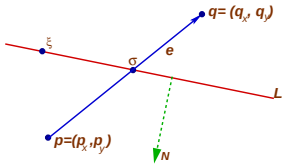
$\sigma_y = \alpha$

$$\alpha = p_y + t(q_y - p_y) \;\Rightarrow\; t = \frac{\alpha - p_y}{q_y - p_y}$$

$$\sigma_x = p_x + (\alpha - p_y) \cdot \frac{q_x - p_x}{q_y - p_y}.$$

Slide 8

Computing the Intersection



e: Segment with endpoints p and q

$$e(t) = p + (q - p)t.$$

L: Line with outward normal N and a point $\xi \in L$.

$$(\mathbf{x} - \xi) \cdot \mathbf{N} = 0$$

σ: Intersection point of e and L.

Since $\sigma = L \cap e$, $\exists t_\sigma \in \mathbb{R}$,

$$\sigma = p + (q - p)t_\sigma \quad \& \quad (\sigma - \xi) \cdot N = 0,$$

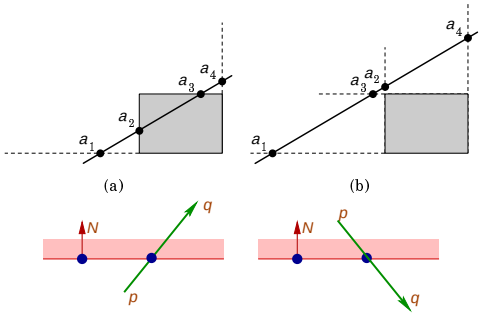
$$(p + (q - p)t_\sigma - \xi) \cdot N = 0 \Rightarrow$$
$$t_\sigma (q - p) \cdot N = (\xi - p) \cdot N$$

$$t_\sigma = \frac{(\xi - p) \cdot N}{(q - p) \cdot N}, \quad \sigma = p + (q - p)\left[\frac{(\xi - p) \cdot N}{(q - p) \cdot N}\right]$$

Slide 9

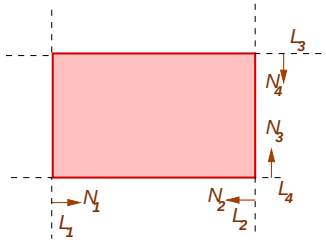
Liang-Barsky Algorithm

- ☆ Cohen-Sutherland algorithm works well when few lines appear inside the window
- ☆ Computing interscetion points explicitly is expensive
- ☆ Use parametric representation of segments:
  - $e(t) = (1 - t)p + tq$
  - Compute $t_1, t_2 \in [0, 1]$ s.t. $W \cap e = e(t_1)e(t_2)$



Slide 10

Liang-Barsky Algorithm



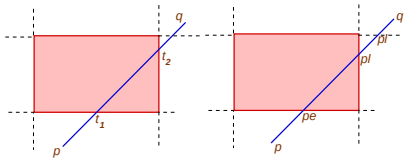
L_i: Line supporting the i-th boundary of W
N_i: Inward normal of L_i (pointing toward W)

$N_1 = (1, 0) \quad N_2(-1, 0)$
$N_3 = (0, 1) \quad N_2(0, -1)$

σ_i: Intersection point of e and L_i.

Slide 11

Liang-Barsky Algorithm



- ☆ Potentially entering (PE) at σ_i if $(q-p) \cdot N_i > 0$ (i.e., entering the inner halfplane).
- ☆ Potentially leaving (PL) if $(q-p) \cdot N_i < 0$ (i.e., leaving the inner halfplane).
- ☆ $(q-p) \cdot N_1 = q_x - p_x$, $(q-p) \cdot N_2 = -(q_x - p_x)$
- ☆ σ_i is PE $\Rightarrow$ update t_1
- ☆ σ_i is PL $\Rightarrow$ update t_2

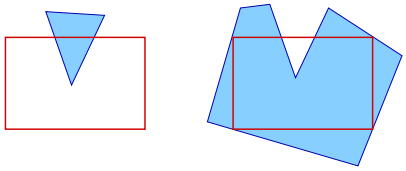
$$t = (\xi_i - p) \cdot N_i \Big/ (q - p) \cdot N_i$$

if $(q - p) \cdot N_i > 0$
  then $t_1 = \max\{t_1, t\}$
  else $t_2 = \min\{t_2, t\}$

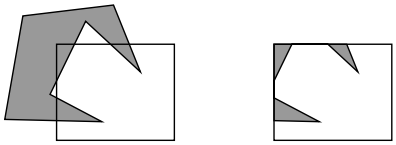
Slide 12

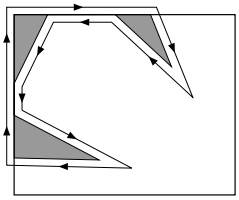
Polygon Clipping

Clipping the edges is not enough.



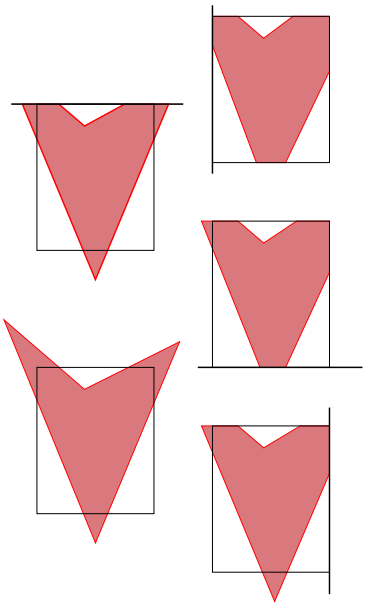
Clipped portion might be disconnected





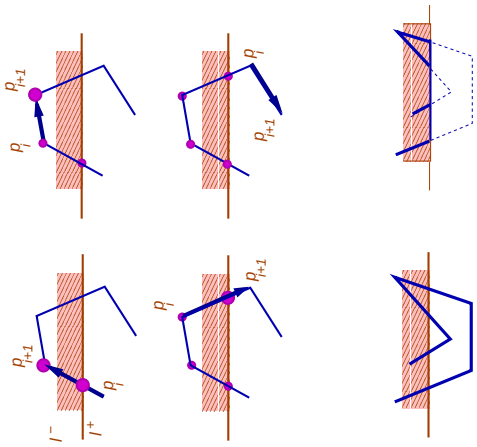
Slide 13

Sutherland-Hodgeman Algorithm



Slide 14

Sutherland-Hodgeman Algorithm



Slide 15

Sutherland-Hodgeman Algorithm

- ☆ Follow the boundary of P in the counter-clockwise direction.
- ☆ For each edge e of P, compute the vertices of the clipped polygon lying on e.

Processing an edge p_ip_{i+1}:

$p_i \in \ell_i^+, p_{i+1} \in \ell_i^-$:

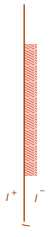
- ☆ Compute $\sigma_i = p_ip_{i+1} \cap \ell$.
- ☆ Output σ_i and then p_{i+1}.

$p_i \in \ell_i^+, p_{i+1} \in \ell_i^+$: Do nothing.

$p_i \in \ell_i^-, p_{i+1} \in \ell_i^+$:

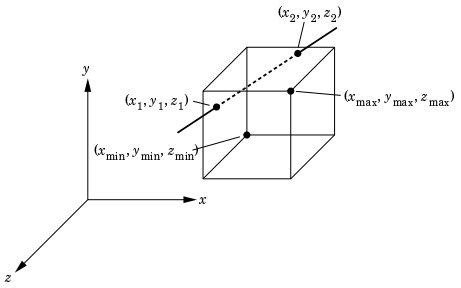
- ☆ Compute $\sigma_i = p_ip_{i+1} \cap \ell$.
- ☆ Output σ_i.

$p_i \in \ell_i^-, p_{i+1} \in \ell_i^-$: Output p_{i+1}.



Slide 16

3D Clipping



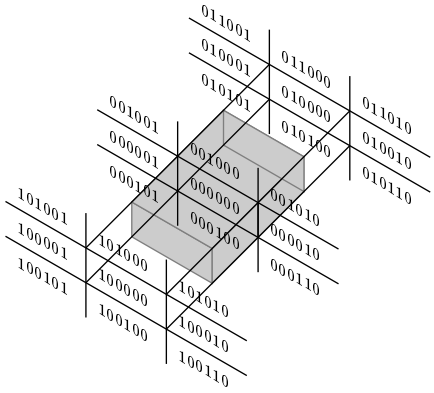
Extend the 2D algorithms.

Cohen-Sutherland Algorithm:
Maintain a six-bit code.

bit 5	front	$z < z_L$
bit 6	back	$z > z_R$

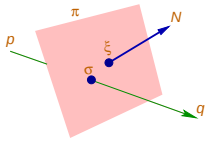
Slide 17

3D Cohen-Sutherland Algorithm



Computing plane-segment intersection:

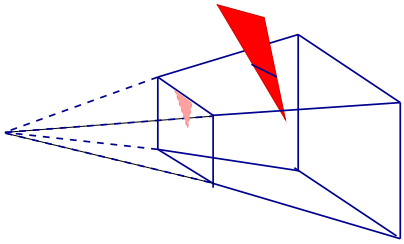
$e : e(t) = (1-t)p + tq$
$\pi : \mathbf{N}(\mathbf{x} - \xi) = 0$
$t = \dfrac{N \cdot (\xi - p)}{N(q - p)}$



Slide 18

3D Clipping

Perspective view volume



- ☆ Apply the above approach to the perspective view volume directly.
  Computing segment-plane intersection is difficult.
- ☆ Transform a perspective view frustum to an orthographic view frustum.
  - *Need only one precedure for clipping.*
  - Simplifies the hardware.
- ☆ Clipping after rasterization

Slide 19

OpenGL Commands

OpenGL provides six additional clipping planes
GL_CLIP_PLANE1, ... GL_CLIP_PLANE6
`glClipPlane`(*plane, *equation*);

- ☆ plane: GL_CLIP_PLANEi ($i = 1, \ldots, 6$)
- ☆ equation: Four coefficients of the plane equation

$$Ax + By + Cz + D = 0$$

- ☆ `glEnable`(GL_CLIP_PLANEi)
- ☆ `glDisable`(GL_CLIP_PLANEi)

Slide 20

Display Lists

```
drawCircle ()
  {
    GLfloat i;
    GLfloat cosine, sine;
    glBegin(GL_POLYGON);
    for (i = 0; i < 100; i++)
        {
          cosine = cos(i * 2.0 × π/100.0);
          sine = sin(i * 2.0 × π/100.0);
          glVertex2f (cosine, sine);
          }
    glEnd();
    }
```

Slide 21

Display Lists

```
#define MY_CIRCLE 1
buildCircle ()
  {
    GLfloat i;
    GLfloat cosine, sine;
    glNewList(MY_CIRCLE, GL_COMPILE);
    glBegin(GL_POLYGON);
    for (i = 0; i < 100; i++)
        {
          cosine = cos(i * 2.0 × π/100.0);
          sine = sin(i * 2.0 × π/100.0);
          glVertex2f (cosine, sine);
          }
    glEnd();
    glEndList();
    }
        glCallList(MY_CIRCLE);
```

Slide 22