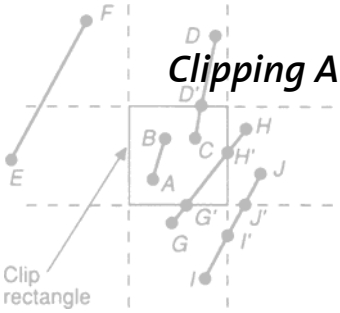
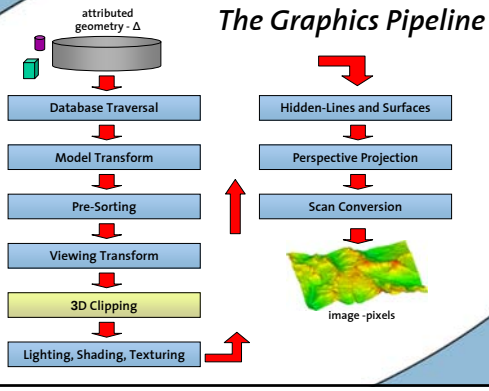
Clipping Algorithms



The Graphics Pipeline

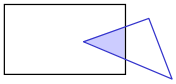


Clipping

- 3D clipping to viewing frustum

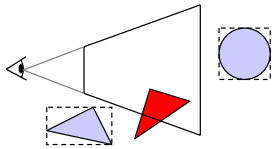
Images courtesy of MIT

- 2D clipping to windows and viewports



- Today: Line clipping, polygon clipping

Culling and Clipping



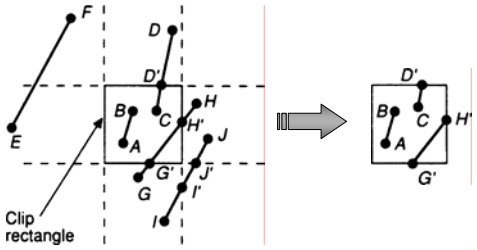
Culling:

- Rejects via bounding volumes
- Bounding volume hierarchies for entire scenes

Clipping:

- Partially visible objects need to be clipped against convex polygon (polytope)

Line Clipping in 2D



Cases

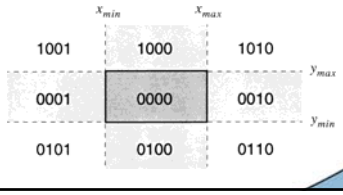
- Cases:
  - Both vertices inside ⇒ ***no clipping***
  - Both vertices outside ⇒ ***clipping may be necessary***
  - One in one out ⇒ ***clipping algorithm***
- Inside condition for point (x, y)

$$x_{min} \le x \le x_{max} \qquad y_{min} \le y \le y_{max}$$

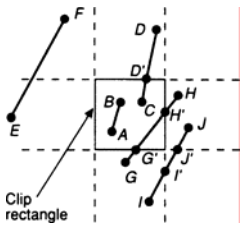
- Brute force: explicit computation of all segment-segment intersections

Cohen-Sutherland Algorithm

- Classical line clipper
- Trivial accept and trivial reject
- Iterative subdivision strategy
- Plane partitioning using 4 bit code



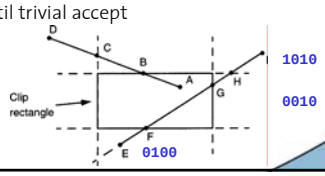
Outcodes – Example



Line	*Code 1*	*Code 2*	*C1 ∧ C2*	*C1 ∨ C2*
A⇒B	0000	0000	0000	0000
C⇒D	0000	1000	0000	1000
E⇒F	0001	1001	0001	1001
G⇒H	0100	0010	0000	0110
I⇒J	0100	0010	0000	0110

Iterative Subdivision

- Select a vertex (outside)
- Priority ranking of window edges
- Divide line at intersection with edge of highest priority
- Delete line segment (point ⇒ intersection)
- Compute outcode
- Iterate until trivial accept



C-Code

```
#define NIL     0
#define LEFT    1
#define RIGHT   2
#define BOTTOM  4
#define TOP     8

short CompOutCode(float x, float y) {
  short outcode;
  outcode = NIL;
  if (y > ymax)
    outcode |= TOP;
  else if (y < ymin)
    outcode |= BOTTOM;
  if (x > xmax)
    outcode |= RIGHT;
  else if (x < xmin)
    outcode |= LEFT;
  return outcode;
}
```


C-Code

```
void CohenSutherlandLineClipAndDraw(float x0, float y0, float x1, float y1) {
  int accept = FALSE, done = FALSE;
  float x, y;
  short outcode0, outcode1, outcodeOut;

  outcode0 = CompOutCode(x0, y0); outcode1 = CompOutCode(x1, y1);
  do {
    if (!(outcode0 | outcode1)) {       /* trivial innerhalb */
      accept = TRUE; done = TRUE;
    }
    else if ((outcode0 & outcode1))     /* trivial ausserhalb */
      done = TRUE;
    else {
      if (outcode0) outcodeOut = outcode0; else outcodeOut = outcode1;
      if (outcodeOut & TOP) { x = x0 + (x1 - x0)*(ymax - y0)/(y1 - y0); y = ymax; }
      else if (outcodeOut & BOTTOM) { x = x0 + (x1 - x0)*(ymin - y0)/(y1 - y0); y = ymin; }
      else if (outcodeOut & RIGHT) { y = y0 + (y1 - y0)*(xmax - x0)/(x1 - x0); x = xmax; }
      else if (outcodeOut & LEFT) { y = y0 + (y1 - y0)*(xmin - x0)/(x1 - x0); x = xmin; }
      if (outcodeOut == outcode0) { x0 = x; y0 = y; outcode0 = CompOutCode(x0, y0); }
      else { x1 = x; y1 = y; outcode1 = CompOutCode(x1, y1); }
    }
  } while (!done);
  if (accept) Line(x0, y0, x1, y1);
}
```

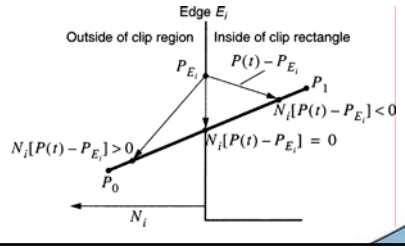

Parametric Clipping

- Cohen-Sutherland good for small and large windows
- Poor worst case performance
- Computes many unnecessary intersections

  ⇒ ***Parametric Clipping*** (Liang-Barsky Algorithm)

Idea

- Compute intersection in 1D parameter space of the line $\mathbf{P}(t) = \mathbf{P}_0 + (\mathbf{P}_1 - \mathbf{P}_0)t \qquad t \in [0,1]$



Parametric Form

- Parametric form of a line

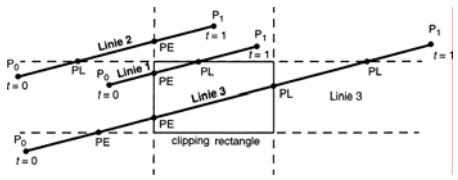
  $\mathbf{P}(t) = \mathbf{P}_0 + (\mathbf{P}_1 - \mathbf{P}_0)t \qquad t \in [0,1]$
- Intersection point

  $$\boldsymbol{N}_i \cdot \left(P(t) - P_{E_i}\right) = 0$$
- Insert

  $$t = \frac{\boldsymbol{N}_i \cdot \left(P_0 - P_{E_i}\right)}{-\boldsymbol{N}_i \cdot \boldsymbol{D}}$$
- $\boldsymbol{D} = \boldsymbol{P}_1 - \boldsymbol{P}_0$

Classification of Intersections

- Classification of entry points (*PE*) and leaving points (*PL*)



Tests and Conditions

- Test of all $t \in [0,..,1]$
- Entry and leaving points computed by

  $\boldsymbol{N} \cdot \boldsymbol{D} < 0 \Rightarrow PE \qquad \boldsymbol{N} \cdot \boldsymbol{D} > 0 \Rightarrow PL$
- Compute maximum and minimum

  $t_E = max(t_{E_i}, P_{E_i}) \qquad t_L = min(t_{L_i}, P_{L_i})$
- If $t_E > t_L \Rightarrow$ no relevant intersection

Look-up Table

Clip edge$_i$	Normal N_i	P_{E_i}	$P_0 - P_{E_i}$	$t = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot D}$
left: $x = x_{min}$	(-1, 0)	(x_{min}, y)	$(x_0 - x_{min}, y_0 - y)$	$\frac{-(x_0 - x_{min})}{(x_1 - x_0)}$
right: $x = x_{max}$	(1, 0)	(x_{max}, y)	$(x_0 - x_{max}, y_0 - y)$	$\frac{(x_0 - x_{max})}{-(x_1 - x_0)}$
bottom: $y = y_{min}$	(0, -1)	(x, y_{min})	$(x_0 - x, y_0 - y_{min})$	$\frac{-(y_0 - y_{min})}{(y_1 - y_0)}$
top: $y = y_{max}$	(0, 1)	(x, y_{max})	$(x_0 - x, y_0 - y_{max})$	$\frac{(y_0 - y_{max})}{-(y_1 - y_0)}$

Due to the zeros in the normal vector the point P_{E_i} is canceled out in the final equations

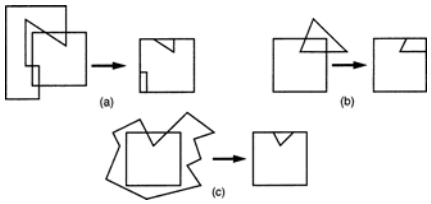
C-Code

```
void Clip2D (float *x0, float *y0, float *x1, float *y1, boolean *visible)
{
  float tE, tL;
  float dx, dy;
  dx = *x1 - *x0;
  dy = *y1 - *y0;
  *visible = FALSE;
  if (dx == 0 && dy == 0 && ClipPoint(*x0, *y0))
    *visible = TRUE;
  else {
    tE = 0;
    tL = 1;
    if (CLIPt(dx, xmin - *x0, &tE, &tL))
      if (CLIPt(-dx, *x0 - xmax, &tE, &tL))
        if (CLIPt(dy, ymin - *y0, &tE, &tL))
          if (CLIPt(-dy, *y0-ymax, &tE, &tL)) {
            *visible = TRUE;
            if (tL < 1) {
              *x1 = *x0 + tL * dx;
              *y1 = *y0 + tL * dy;
            }
            if (tE > 0) {
              *x0 = *x0 + tE * dx;
              *y0 = *y0 + tE * dy;
            }
          }
  }
}
```

C-Code

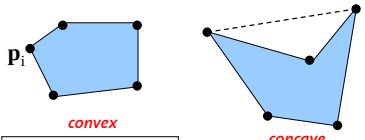
```
boolean CLIPt (float denom, float num, float *tE, float *tL) {
  float t;
  boolean accept;
  accept = TRUE;
  if (denom > 0) {            /* entry */
    t = num/denom;
    if (t > *tL)
      accept = FALSE;
    else if (t > *tE)
      *tE = t;
  }
  else if (denom < 0) {       /* leave */
    t = num/denom;
    if (t < *tE)
      accept = FALSE;
    else if (t < *tL)
      *tL = t;
  }
  else                        /* parallel */
    if (num > 0)
      accept = FALSE;
  return accept;
}
```


Polygon Clipping in 2D



2D Polygon Clipping

- Definition of a convex polygon $P = \{\mathbf{p}_1,..,\mathbf{p}_n\}$



$$\boxed{\forall i, j \in [1,n] : \overline{\mathbf{p}_i\mathbf{p}_j} \in P}$$

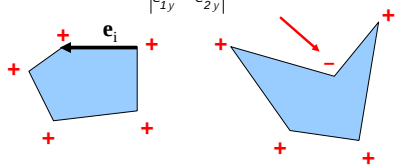
- Convex combination

$$conv(P) := \{ \sum_{i=1}^{n} \lambda_i \mathbf{p}_i \,|\, \lambda_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1 \}$$

Convexity Test

- Use determinant of adjacent edge vectors

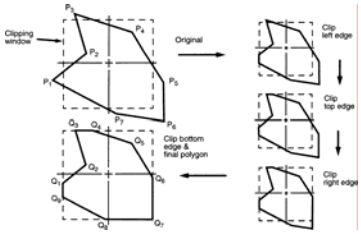
$$\begin{vmatrix} e_{1x} & e_{2x} \\ e_{1y} & e_{2y} \end{vmatrix}$$



signs uniform signs mixed

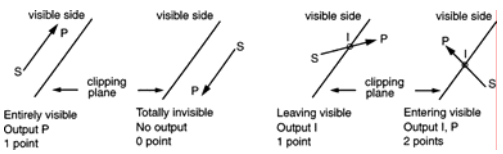
Sutherland-Hodgeman Algorithm

- Iterative subdivision (clipping window convex)
- Produces intermediate polygon lists $\{P_i\}$



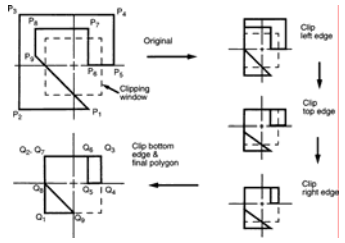
Clipping of Polygon Edges

- Reduces to clipping of individual polygon edges



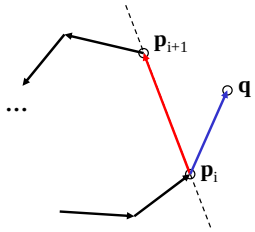
Example

- Polygon with unit square



Inside-Outside Test

- Define polygon counterclockwise

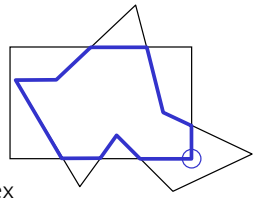


$\mathbf{c} = (\mathbf{q} - \mathbf{p}_i) \times (\mathbf{p}_{i+1} - \mathbf{p}_i)$

outside if $c_z > 0$

Liang-Barsky Algorithm

- Parametric clipping
- Walk around polygon
- For each edge output endpoints of visible part
- Sometimes turning vertex is needed!
- When?



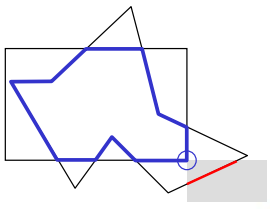
2D Regions

- Uses subdivision of the window plane

inside 2	*inside 3*	*inside 2*
inside 3	*inside all 4*	*inside 3*
inside 2	*inside 3*	*inside 2*

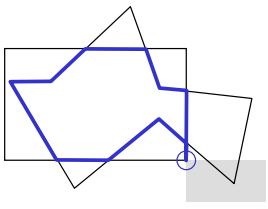
Liang-Barsky Algorithm

- Turning vertex needed when polygon touches inside 2 region
- Each time an edge enters "inside 2" region add turning vertex



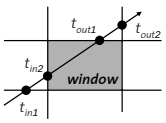
Unnecessary Corners

- Unnecessary vertices do not cause artifacts inside visible window

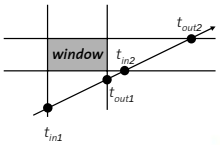


Two Cases

- Segment in parametric form $\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$
- In general corresponding line cuts 4 times:
- $t_{in1} < t_{in2}$, $t_{out1} < t_{out2}$

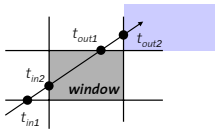


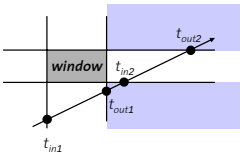
$t_{in2} < t_{out1}$



$t_{in2} > t_{out1}$

Two Cases





- Part of segment visible: $t_{out1} > 0$ and $t_{in2} \leq 1$
- Enter "inside 2" region: $0 < t_{out2} \leq 1$

- Segment not visible
- Enter "inside 2" region: $0 < t_{out1} \leq 1$ $0 < t_{out2} \leq 1$

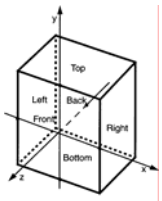
C-Code

```
for each edge e {
  compute tin1, tin2, tout1, tout2;
  if  (tin2 > tout1) { /* kein sichtbares Segment */
    if (0 < tout1 <= 1) Output_vert(turning vertex);
  }
  else {
    if ((0 < tout1) && (1 >= tin2)) {  /* sichtbares Segment vorhanden */
      if (0 <= tin2)
        Output_vert(appropriate side intersection);
      else
        Output_vert(starting vertex);
      if (1 >= tout1)
        Output_vert(appropriate side intersection);
      else
        Output_vert(ending vertex);
    }
  }
  if (0 < tout2 <= 1) Output_vert(appropriate corner);
}
```

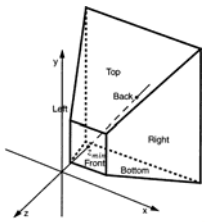
Horizontal and vertical lines!
About twice as fast as Sutherland-Hodgeman

3D Clipping

- Definition of clipping planes and viewing frustum in 3D



a) Parallelprojektion



b) Perspektivische Projektion