3.12.4 A Parametric Line-Clipping Algorithm

The Cohen–Sutherland algorithm is probably still the most commonly used line-clipping algorithm because it has been around longest and has been published widely. In 1978, Cyrus and Beck published an algorithm that takes a fundamentally different and generally more efficient approach to line clipping [CYRU78]. The Cyrus–Beck technique can be used to clip a 2D line against a rectangle or an arbitrary convex polygon in the plane, or a 3D line against an arbitrary convex polyhedron in 3D space. Liang and Barsky later independently developed a more efficient parametric line-clipping algorithm that is especially fast in the special cases of upright 2D and 3D clip regions [LIAN84]. In addition to taking advantage of these simple clip boundaries, they introduced more efficient trivial rejection tests that work for general clip regions. Here we follow the original Cyrus–Beck development to introduce parametric clipping. Since we are concerned only with upright clip rectangles, however, we reduce the Cyrus–Beck formulation to the more efficient Liang–Barsky case at the end of the development.

Recall that the Cohen–Sutherland algorithm, for lines that cannot be trivially accepted or rejected, calculates the (x, y) intersection of a line segment with a clip edge by substituting the known value of x or y for the vertical or horizontal clip edge, respectively. The parametric line algorithm, however, finds the value of the parameter t in the parametric representation of the line segment for the point at which that segment intersects the infinite line on which the clip edge lies. Because all clip edges are in general intersected by the line, four values of t are calculated. A series of simple comparisons is used to determine which (if any) of the four values of t correspond to actual intersections. Only then are the (x, y) values for one or two actual intersections calculated. In general, this approach saves time over the Cohen–Sutherland intersection-calculation algorithm because it avoids the repetitive looping needed to clip to multiple clip-rectangle edges. Also, calculations in 1D parameter space are simpler than those in 3D coordinate space. Liang and Barsky improve on Cyrus–Beck by examining each t-value as it is generated, to reject some line segments before all four t-values have been computed.

The Cyrus–Beck algorithm is based on the following formulation of the intersection between two lines. Figure 3.42 shows a single edge E_i of the clip rectangle and that edge's outward normal N_i (i.e., outward to the clip rectangle[11]), as well as the line segment from P_0 to P_1 that must be clipped to the edge. Either the edge or the line segment may have to be extended to find the intersection point.

As before, this line is represented parametrically as

$$P(t) = P_0 + (P_1 - P_0)t,$$

where $t = 0$ at P_0 and $t = 1$ at P_1. Now, pick an arbitrary point P_{E_i} on edge E_i and consider the three vectors $P(t) - P_{E_i}$ from P_{E_i} to three designated points on the line from P_0 to P_1: the intersection point to be determined, an endpoint of the line on the inside halfplane of the edge, and an endpoint on the line in the outside halfplane of the edge. We can distinguish in which region a point lies by looking at the value of the dot product $N_i \cdot [P(t) - P_{E_i}]$. This value is negative for a point in the inside halfplane, zero for a point on the line containing the edge, and positive for a point that lies in the outside halfplane. The definitions of inside and outside halfplanes of an edge correspond to a counterclockwise enumeration of the edges of the clip region, a convention we shall use throughout this book. Now we can solve for the value of t at the intersection of P_0P_1 with the edge:

$$N_i \cdot [P(t) - P_{E_i}] = 0.$$

First, substitute for $P(t)$:

$$N_i \cdot [P_0 + (P_1 - P_0)t - P_{E_i}] = 0.$$

Next, group terms and distribute the dot product:

$$N_i \cdot [P_0 - P_{E_i}] + N_i \cdot [P_1 - P_0]t = 0.$$

Let $D = (P_1 - P_0)$ be the vector from P_0 to P_1, and solve for t:

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}. \tag{3.1}$$

Note that this gives a valid value of t only if the denominator of the expression is nonzero.

[11]Cyrus and Beck use inward normals, but we prefer to use outward normals for consistency with plane normals in 3D, which are outward. Our formulation therefore differs only in the testing of a sign.

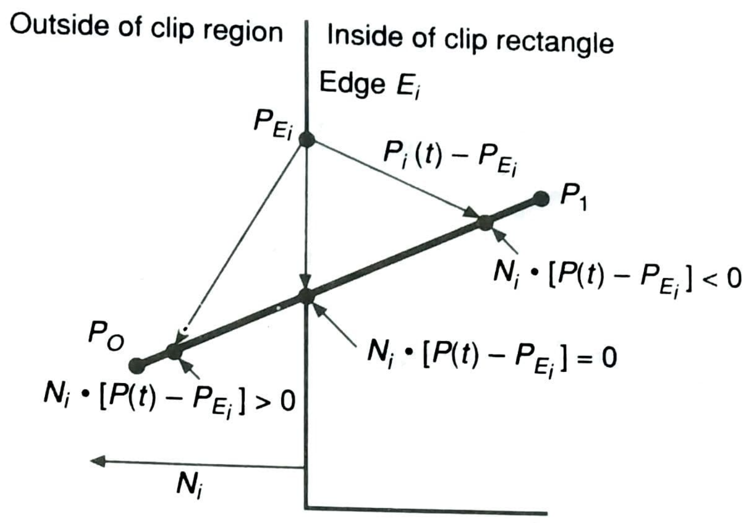
Outside of clip region
Inside of clip rectangle
Edge E_i
P_{E_i}
$P_i(t) - P_{E_i}$
P_1
$N_i \bullet [P(t) - P_{E_i}] < 0$
P_0
$N_i \bullet [P(t) - P_{E_i}] = 0$
$N_i \bullet [P(t) - P_{E_i}] > 0$
N_i

Fig. 3.42 Dot products for three points outside, inside, and on the boundary of the clip region.

For this to be true, the algorithm checks that

$N_i \neq 0$ (that is, the normal should not be 0; this could occur only as a mistake),

$D \neq 0$ (that is, $P_1 \neq P_0$),

$N_i \cdot D \neq 0$ (that is, the edge E_i and the line from P_0 to P_1 are not parallel. If they were parallel, there can be no single intersection for this edge, so the algorithm moves on to the next case.).

Equation (3.1) can be used to find the intersections between P_0P_1 and each edge of the clip rectangle. We do this calculation by determining the normal and an arbitrary P_{E_i}—say, an endpoint of the edge—for each clip edge, then using these values for all line segments. Given the four values of t for a line segment, the next step is to determine which (if any) of the values correspond to internal intersections of the line segment with edges of the clip rectangle. As a first step, any value of t outside the interval [0, 1] can be discarded, since it lies outside P_0P_1. Next, we need to determine whether the intersection lies on the clip boundary.

We could try simply sorting the remaining values of t, choosing the intermediate values of t for intersection points, as suggested in Fig. 3.43 for the case of line 1. But how do we distinguish this case from that of line 2, in which no portion of the line segment lies in the clip rectangle and the intermediate values of t correspond to points not on the clip boundary? Also, which of the four intersections of line 3 are the ones on the clip boundary?

The intersections in Fig. 3.43 are characterized as "potentially entering" (PE) or "potentially leaving" (PL) the clip rectangle, as follows: If moving from P_0 to P_1 causes us to cross a particular edge to enter the edge's inside halfplane, the intersection is PE; if it causes us to leave the edge's inside halfplane, it is PL. Notice that, with this distinction, two interior intersection points of a line intersecting the clip rectangle have opposing labels.

Formally, intersections can be classified as PE or PL on the basis of the angle between P_0P_1 and N_i: If the angle is less than 90°, the intersection is PL; if it is greater than 90°, it is PE. This information is contained in the sign of the dot product of N_i and P_0P_1:

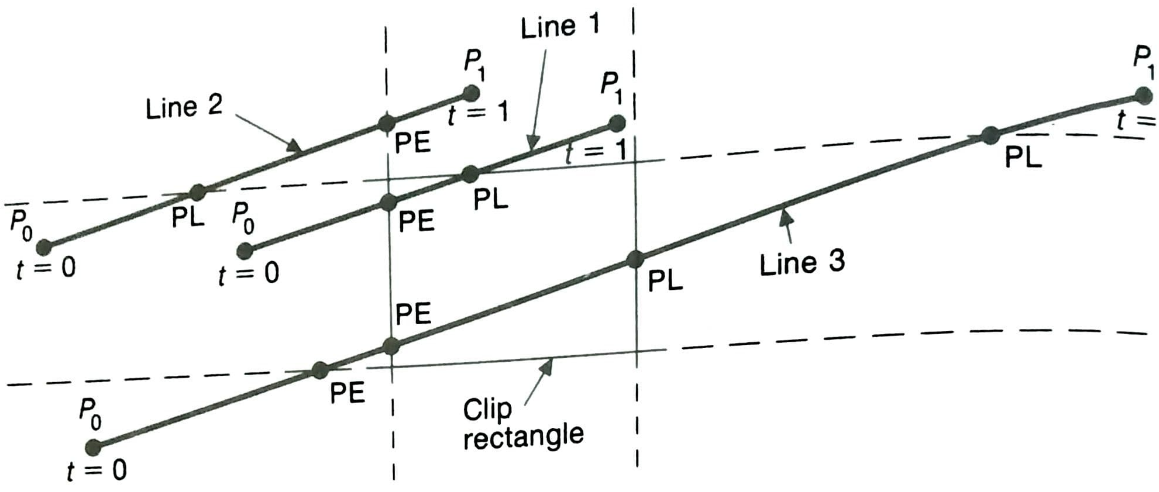

Fig. 3.43 Lines lying diagonal to the clip rectangle.

$$N_i \cdot D < 0 \Rightarrow \text{PE (angle greater than 90)},$$
$$N_i \cdot D > 0 \Rightarrow \text{PL (angle less than 90)}.$$

Notice that $N_i \cdot D$ is merely the denominator of Eq. (3.1), which means that, in the process of calculating t, the intersection can be trivially categorized.

With this categorization, line 3 in Fig. 3.43 suggests the final step in the process. We must choose a (PE, PL) pair that defines the clipped line. The portion of the infinite line through P_0P_1 that is within the clipping region is bounded by the PE intersection with the largest t value, which we call t_E, and the PL intersection with the smallest t value, t_L. The intersecting line segment is then defined by the range (t_E, t_L). But because we are interested in intersecting P_0P_1, not the infinite line, the definition of the range must be further modified so that $t = 0$ is a lower bound for t_E and $t = 1$ is an upper bound for t_L. What if $t_E > t_L$? This is exactly the case for line 2. It means that no portion of P_0P_1 is within the clip rectangle, and the entire line is rejected. Values of t_E and t_L that correspond to actual intersections are used to calculate the corresponding x and y coordinates.

The completed algorithm for upright clip rectangles is pseudocoded in Fig. 3.44. Table 3.1 shows for each edge the values of N_i, a canonical point on the edge, P_{E_i}, the vector $P_0 - P_{E_i}$ and the parameter t. Interestingly enough, because one coordinate of each normal is 0, we do not need to pin down the corresponding coordinate of P_{E_i} (denoted by an indeterminate x or y). Indeed, because the clip edges are horizontal and vertical, many simplifications apply that have natural interpretations. Thus we see from the table that the numerator, the dot product $N_i \cdot (P_0 - P_{E_i})$ determining whether the endpoint P_0 lies inside or outside a specified edge, reduces to the directed horizontal or vertical distance from the point to the edge. This is exactly the same quantity computed for the corresponding component of the Cohen-Sutherland outcode. The denominator dot product $N_i \cdot D$, which determines whether the intersection is potentially entering or leaving, reduces to $\pm dx$ or dy: if dx is positive, the line moves from left to right and is PE for the left edge, PL for the right edge, and so on. Finally, the parameter t, the ratio of numerator and denominator, reduces to the distance to an edge divided by dx or dy, exactly the constant of proportionality we could calculate directly from the parametric line formulation. Note that it is important to preserve the signs of the numerator and denominator instead of cancelling minus signs, because the numerator and denominator as signed distances carry information that is used in the algorithm.

```
precalculate $N_i$ and select a $P_{E_i}$ for each edge;

for (each line segment to be clipped) {
    if ($P_1 == P_0$)
        line is degenerate so clip as a point;
    else {
        $t_E = 0$; $t_L = 1$;
        for (each candidate intersection with a clip edge) {
            if ($N_i \bullet D$ != 0) {    /* Ignore edges parallel to line for now */
                calculate t;
                use sign of $N_i \bullet D$ to categorize as PE or PL;
                if (PE) $t_E$ = max ($t_E$, t);
                if (PL) $t_L$ = min ($t_L$, t);
            }
        }
        if ($t_E > t_L$)
            return NULL;
        else
            return $P(t_E)$ and $P(t_L)$ as true clip intersections;
    }
}
```

Fig. 3.44 Pseudocode for Cyrus–Beck parametric line clipping algorithm.

The complete version of the code, adapted from [LIAN84] is shown in Fig. 3.45. The procedure calls an internal function, CLIPt(), that uses the sign of the denominator to determine whether the line segment-edge intersection is potentially entering (PE) or leaving (PL), computes the parameter value of the intersection and checks to see if trivial rejection can be done because the new value of t_E or t_L would cross the old value of t_L or t_E, respectively. It also signals trivial rejection if the line is parallel to the edge and on the

TABLE 3.1 CALCULATIONS FOR PARAMETRIC LINE CLIPPING ALGORITHM*

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: $y = y_{\max}$	$(0, 1)$	$(x, y_{\max})$	$(x_0 - x, y_0 - y_{\max})$	$\frac{(y_0 - y_{\max})}{-(y_1 - y_0)}$

*The exact coordinates of the point P_{E_i} on each edge are irrelevant to the computation, so they have been denoted by variables x and y. For a point on the right edge, $x = x_{\min}$ as indicated in the first row, third entry.

```
void Clip2D (double *x0, double *y0, double *x1, double *y1, boolean *visible)
/* Clip 2D line segment with endpoints (x0, y0) and (x1, y1), against upright */
/* clip rectangle with corners at (xmin, ymin) and (xmax, ymax); these are */
/* globals or could be passed as parameters also. The flag visible is set TRUE. */
/* if a clipped segment is returned in endpoint parameters. If the line */
/* is rejected, the endpoints are not changed and visible is set to FALSE. */
{
  double dx = *x1 - *x0;
  double dy = *y1 - *y0;
  /* Output is generated only if line is inside all four edges. */
  *visible = FALSE;
  /* First test for degenerate line and clip the point; ClipPoint returns */
  /* TRUE if the point lies inside the clip rectangle. */
  if (dx == 0 && dy == 0 && ClipPoint (*x0, *y0))
     *visible = TRUE;
  else {
     double tE = 0.0;
     double tL = 1.0;
     if (CLIPt (dx, xmin - *x0, &tE, &tL))                /* Inside wrt left edge */
        if (CLIPt (-dx, *x0 - xmax, &tE, &tL))            /* Inside wrt right edge */
           if (CLIPt (dy, ymin - *y0, &tE, &tL))          /* Inside wrt bottom edge */
              if (CLIPt (-dy, *y0 - ymax, &tE, &tL)) {    /* Inside wrt top edge */
                 *visible = TRUE;
                 /* Compute PL intersection, if tL has moved */
                 if (tL < 1) {
                    *x1 = *x0 + tL * dx;
                    *y1 = *y0 + tL * dy;
                 }
                 /* Compute PE intersection, if tE has moved */
                 if (tE > 0) {
                    *x0 += tE * dx;
                    *y0 += tE * dy;
                 }
              }
  }
} /* Clip2D */

boolean CLIPt (double denom, double num, double *tE, double *tL)
/* This function computes a new value of tE or tL for an interior intersection */
/* of a line segment and an edge. Parameter denom is -(Ni • D), which reduces to */
/* ± Δx, Δy for upright rectangles (as shown in Table 3.1); its sign */
/* determines whether the intersection is PE or PL. Parameter num is Ni • (P0 - PEi) */
/* for a particular edge/line combination, which reduces to directed horizontal */
/* and vertical distances from P0 to an edge; its sign determines visibility */
/* of P0 and is used to trivially reject horizontal or vertical lines. If the */
/* line segment can be trivially rejected, FALSE is returned; if it cannot be, */
/* TRUE is returned and the value of tE or tL is adjusted, if needed, for the */
/* portion of the segment that is inside the edge. */
```

Fig. 3.45 *(Cont.)*.

```
{
    double t;

    if (denom > 0) {                /* PE intersection */
        t = num / denom;            /* Value of t at the intersection */
        if (t > tL)                 /* tE and tL crossover */
            return FALSE;           /* so prepare to reject line */
        else if (t > tE)            /* A new tE has been found */
            tE = t;
    } else if (denom < 0) {         /* PL intersection */
        t = num / denom;            /* Value of t at the intersection */
        if (t < tE)                 /* tE and tL crossover */
            return FALSE;           /* so prepare to reject line */
        else                        /* A new tL has been found */
            tL = t;
    } else if (num > 0)             /* Line on outside of edge */
        return FALSE;
    return TRUE;
} /* CLIPt */
```

Fig. 3.45 Code for Liang–Barsky parametric line-clipping algorithm.

outside; i.e., would be invisible. The main procedure then does the actual clipping by moving the endpoints to the most recent values of t_E and t_L computed, but only if there is a line segment inside all four edges. This condition is tested for by a four-deep nested **if** that checks the flags returned by the function signifying whether or not the line segment was rejected.

In summary, the Cohen–Sutherland algorithm is efficient when outcode testing can be done cheaply (for example, by doing bitwise operations in assembly language) and trivial acceptance or rejection is applicable to the majority of line segments. Parametric line clipping wins when many line segments need to be clipped, since the actual calculation of the coordinates of the intersection points is postponed until needed, and testing can be done on parameter values. This parameter calculation is done even for endpoints that would have been trivially accepted in the Cohen–Sutherland strategy, however. The Liang–Barsky algorithm is more efficient than the Cyrus–Beck version because of additional trivial rejection testing that can avoid calculation of all four parameter values for lines that do not intersect the clip rectangle. For lines that cannot be trivially rejected by Cohen–Sutherland because they do not lie in an invisible halfplane, the rejection tests of Liang–Barsky are clearly preferable to the repeated clipping required by Cohen–Sutherland. The Nicholl et al. algorithm of Section 19.1.1 is generally preferable to either Cohen–Sutherland or Liang–Barsky but does not generalize to 3D, as does parametric clipping. Speed-ups to Cohen–Sutherland are discussed in [DUVA90]. Exercise 3.29 concerns instruction counting for the two algorithms covered here, as a means of contrasting their efficiency under various conditions.