PERGAMON

Computers & Graphics 26 (2002) 409–415

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Best Papers of CAD and CB 2001

An efficient line clipping algorithm based on adaptive line rejection

Guodong Lu*, Xuanhui Wu, Qunsheng Peng

State Key Lab. of CAD&CG at Zhejiang University, Hangzhou, 310027, People's Republic of China

Abstract

Line clipping process often involves a lot of intersection calculations. One way for improving the efficiency of a line clipping algorithm is to save the unnecessary intersection calculations demanded by traditional algorithms either for rejecting some totally invisible lines or for clipping some partially visible lines. An adaptive clipping algorithm is presented here to achieve this goal. The clipping process of our new algorithm consists of three steps. Firstly, we adopt Encoding & Code checking technique of Cohen–Sutherland algorithm to accept the totally visible lines and reject the portion of invisible lines that lie completely on the outside of each boundary of the clip window. Secondly, we construct a diamond enclosing box of the original window called *S-Window* to reject another portion of invisible lines that cross the corner of the clip window. All the remaining lines are identified at the third step. For each line, we construct a virtual enclosing box of the original window, called *V-Window*, which aligns with the line's direction. All of invisible lines that fail to be rejected during the previous two steps are now easily picked up without any intersection calculations. The remaining partially visible lines are clipped against the boundary edges of the window that they do intersect. Theoretical analysis and experimental statistics demonstrate the high efficiency of our new algorithm.

Keywords: Line clipping; Encoding & Code checking technique; Adaptive enclosing box; Rejection test

1. Introduction

Clipping, which is a basic operation to several aspects of computer graphics, includes two elements: the clipping window which could be rectangle, circle, convex window, concave window or open window, and the object to be clipped which could be line, polygon, circle, character or irregular curves. Different combination of the two elements and different clipping strategies lead to various algorithms. Among all algorithms, line clipping against the rectangle window receives special attentions. Classical line clipping algorithms, such as *Cohen–Sutherland* algorithm and *Midpoint Subdivision* algorithm, adopt *Encoding & Code Checking* technique to trivially reject a portion of invisible lines and accept all of totally visible lines. The clipping of the rest lines, including partially visible lines and the remaining invisible lines, requires intersection calculations [1,2].

Line clipping is the process of classifying a line set. If we divide a general line set into four non-overlapping subsets, the clipping process can be described as in Fig. 1. Subset A contains all of the totally visible lines. Subset B contains invisible lines that can be trivially rejected by *Encoding & Code Checking* technique. Subset C contains the remaining invisible lines that are not included in Subset B. Subset D contains all lines partially visible.

According to the process illustrated in Fig. 1 and the discussion above, it is obvious that:

1. Subsets A and B can be efficiently identified with *Encoding & Code Checking* technique.
2. Subset D can be successfully clipped after intersection calculation.

*Corresponding author. Tel.: +86-571-87951273; fax: +86-571-87951899.

E-mail address: lugd@mail.hz.zj.cn (G. Lu).

0097-8493/02/$ - see front matter
PII: S0097-8493(02)00084-5

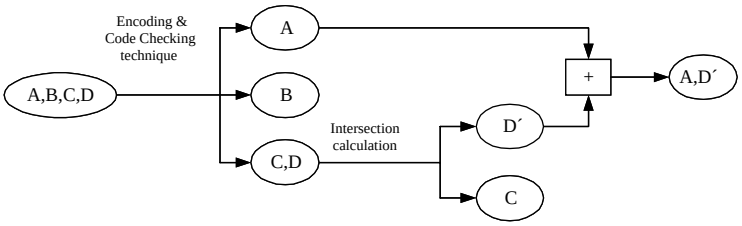

Fig. 1. Line clipping based on *Encoding & Code Checking* technique.

3. Subset *C*, which is called *Redundant Line Set*, is currently identified and rejected by the process of intersection calculation.

Intersection calculation is necessary for the clipping of subset *D*, since the points of intersection determine the visible part of the line. However, it is a waste of time to calculate the intersections when *Redundant Lines*, lines within subset *C* are clipped, because the points of intersection are of no use except for telling us that the line is totally invisible. The complexity in rejecting redundant lines leads to low efficiency of the whole clipping process [3–5].

One way to accelerate the clipping process is to reduce the calculations involved in identifying the *Redundant Line Set*. In this paper, we propose an efficient algorithm of line clipping by adaptively identifying subsets A, B, C, D with different windows. The clipping process of our new algorithm consists of three steps. In the first step, we adopt *Encoding & Code Checking* technique to identify subsets A and B against the original window. In the second step, we construct a 45° window which is an enclosing box of the original one, termed *S-Window*, to identify most of lines in subset C. The remaining line set consists of subset D and the other portion of subset C, and they are distinguished in the third step. To do this, we construct a virtual enclosing box of the original window for each line, called *V-Window*. Since the *V-Window* aligns with the direction of the line to be tested, the two groups of subsets C and D can be easily classified by sorting their intercepts on the *Y*-axis. In the following sections, we will describe the new algorithm step by step.

2. Cohen–Sutherland's clipping against the original window

The traditional *Encoding & Code Checking* technique divides the two-dimensional plane into nine non-overlapping regions as shown in Fig. 2, then uses a four-digit-code to indicate in which region the two endpoints of the line are located [6]. All of the lines totally visible and a part of invisible lines can be easily identified by the

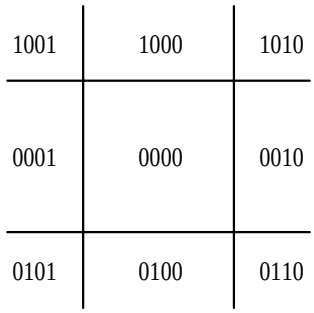

Fig. 2. Traditional *Encoding & Code Checking* technique.

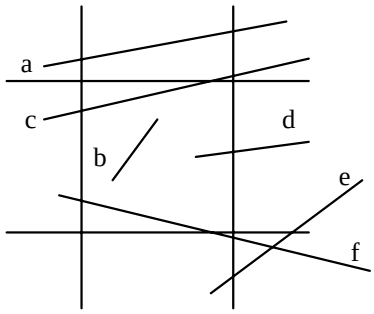

Fig. 3. Line examples.

following logical operations:

$$\text{code}(P1)|\text{code}(P2) = 0, \quad (1)$$

$$\text{code}(P1)\&\text{code}(P2) \neq 0, \quad (2)$$

where $P1$ and $P2$ are endpoints of the line, code( ) is the encoding operation, & denotes logical *and* operator and | denotes logical *or* operator. Line is totally visible if it satisfies Eq. (1), such as line *b* in Fig. 3; while it is totally invisible if it satisfies Eq. (2), such as line *a* in Fig. 3.

Note that if one endpoint of a line is inside the window and the other one is located outside, such as line *d* in Fig. 3, it must be partially visible. This case can be easily identified with *Encoding & Code Checking* technique. Because intersection calculations are necessary for this kind of lines, we adopt the procedure of *Cohen–Sutherland* algorithm to cope with them.

3. Redundant line clipping with S-window

The classical *Cohen–Sutherland* clipping algorithm can trivially reject some of the totally invisible lines such as line *a* in Fig. 3, but it does not work for line *e* in Fig. 3 though it is also invisible. In fact, it is awkward to perform the traditional *Encoding & Code Checking* method to deal with lines passing through the corner region of the original rectangular window. This situation can be turned over if we construct a diamond enclosing box of the original window as shown in Fig. 4. For description convenience, we call it *S-Window*. Note that, the *S-Window* also cuts the plane into nine regions. Each of them can be represented by a four-digit-code shown in Fig. 4. Then most of *Redundant Lines*, such as line *g* of Fig. 4, can be easily rejected by the logical operation of their second endpoint-codes. The encoding procedure for the second endpoint-codes is simple since *S-Window* is aligned 45° to the *X*-axis. Let the boundary of the original clipping window be left (XL), right (XR), top (YT), bottom (YB), the procedure can be described as follows:

```
#define LEFT 1
#define RIGHT 2
#define BOTTOM 4
#define TOP 8
code = 0;
If(x + y − YB < XL)        code = code|LEFT;
Else if(x + y − YT > XR)   code = code|RIGHT;
If(y − x + XR < YB)        code = code|BOTTOM;
Else if(y − x + XL > YT)   code = code|TOP;
Return code;
```

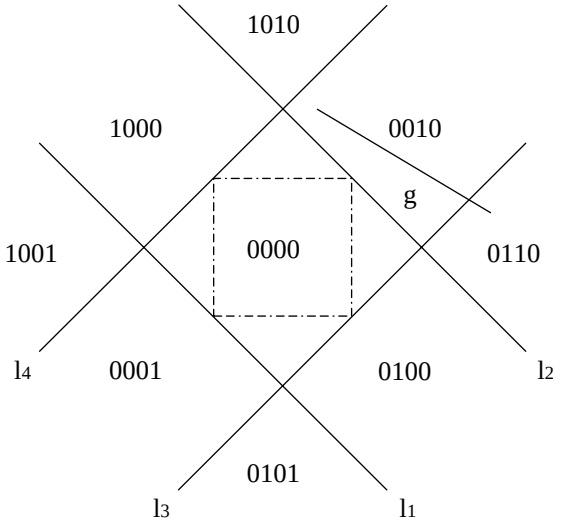

Fig. 4. *S-Window*.

Then, how much is the chance for lines to be rejected with this simple and fast procedure?

Assume that the endpoints of all lines are distributed evenly within a large square whose edge is of length tW, and the clipping window, whose boundary edge is of length W, locates exactly at the center of the large square, as shown in Fig. 5.

In that case, the application of *Encoding & Code Checking* technique against the original window can trivially reject lines located entirely inside any of the four regions: ABGL, BCIF, CDKH and DAEJ. The probability for lines located within any of them is

$$P1 = \left(\frac{(tW - W) \times tW}{2 \times tW \times tW}\right)^2 \times 100\% = \left(\frac{t-1}{2}\right)^2 \times 100\%. \quad (3)$$

Nevertheless, the regions mentioned above have four overlapping squares: AEML, BGNF, CIOH and DKPJ. The probability for lines located within any of them is

$$P2 = \left(\frac{(tW - W) \times (tW - W)}{2 \times 2 \times tW \times tW}\right)^2 \times 100\% = \left(\frac{t-1}{2t}\right)^4 \times 100\%. \quad (4)$$

Also note that the probability of totally visible lines, which locates entirely inside the square of MNOP, is

$$P3 = \left(\frac{W^2}{(tW)^2}\right)^2 \times 100\% = \left(\frac{1}{t}\right)^4 \times 100\%. \quad (5)$$

Then, according to Eqs. (3)–(5), the ratio of lines which can be classified by adopting the *Encoding & Code Checking* technique, against the original window, to all lines can be determined as follows:

$$M1 = 4P1 - 4P2 + P3 = \left[\left(1 - \frac{1}{t}\right)^2 - \frac{1}{4}\left(1 - \frac{1}{t}\right)^4 + \frac{1}{t^4}\right] \times 100\%. \quad (6)$$

Similarly, *S-Window* rejects lines of four portions trivially, i.e. lines whose two endpoints locate, respectively, in the regions of XLM and ESM, in the regions of QKP and JVP, in the regions of WIO and THO, and in the regions of RFN and UGN. Examples of such lines are *a–d* in Fig. 5. The ratio of lines within each portion to all lines is

$$Q1 = 2\left(\frac{(tW - W)^2}{8(tW)^2}\right)^2 \times 100\% = \frac{1}{32}\left(\frac{t-1}{t}\right)^4 \times 100\%. \quad (7)$$

Based on Eqs. (6) and (7), the ratio of lines rejected by *S-Window* to lines remaining after the first clipping step,

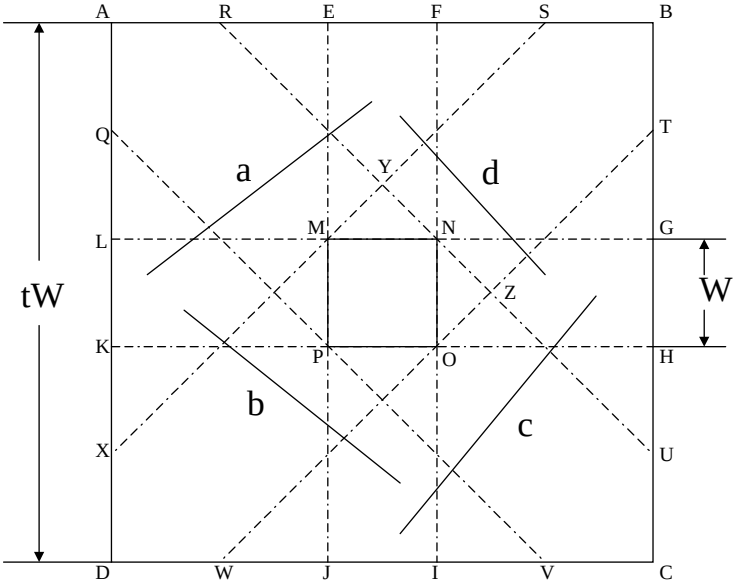

Fig. 5. The model of lines and the clipping window.

can be deduced as follows:

$$M2 = \frac{4Q1}{1 - M1} = \frac{\frac{1}{8}\left(1 - \frac{1}{t}\right)^4}{1 - \left(1 - \frac{1}{t}\right)^2 + \frac{1}{4}\left(1 - \frac{1}{t}\right)^4 - \frac{1}{t^4}} \times 100\%. \quad (8)$$

Table 1 lists the value of $M1$ and $M2$ when t is specified to different values. If $t > 100$, there are nearly a half of lines that can be trivially rejected by *S-Window*. The greater t is, the larger the ratio $M2$ will be. Apparently, *S-Window* does contribute to the high efficiency of line clipping.

Table 1
Efficiency of *S-Window* rejection

t	5	10	100	1000	Infinite
M1 (%)	53.9	64.6	74.0	74.9	75
M2 (%)	11.1	23.2	46.2	49.6	50

4. Remaining line clipping with V-window

After the previous two clipping steps against the original window and *S-Window*, all totally visible lines and most of invisible lines have been identified efficiently. The remaining line set consists of all partially visible lines and some invisible lines that escaped from the above two steps. To facilitate the classification, we construct a virtual window for each remaining line. In this paper, we only consider lines of positive-slop; the discussion on negative-slop lines is analogous.

4.1. Rejection of remaining redundant lines

The virtual window, or *V-Window* for short, is actually an enclosing box of the original clipping window, which aligns with the direction of the line to be clipped, as shown in Fig. 6. It is therefore easy to determine whether the line is totally invisible by means of comparing the y-intercept of the line with that of lines S1 and S2, which compose the *V-Window* and are parallel to the line to be clipped. Suppose b, b1, b2 are, respectively, the y-intercepts of the line to be clipped, line S1 and line S2 as shown in Fig. 6, the classification result can be made from below:

```
if((b<b1)||(b>b2))
   return FAILER;    /* Redundant Lines ,
                        such as lines a and c of Fig. 6*/
else
   {……}              /* lines partly visible,
                        such as line e of Fig. 6*/
```

Here b, b1 and b2 are calculated as following:

$$b = ya - kxa, \quad (9)$$

$$b1 = YB - kXR, \quad (10)$$

$$b2 = YT - kXL, \quad (11)$$

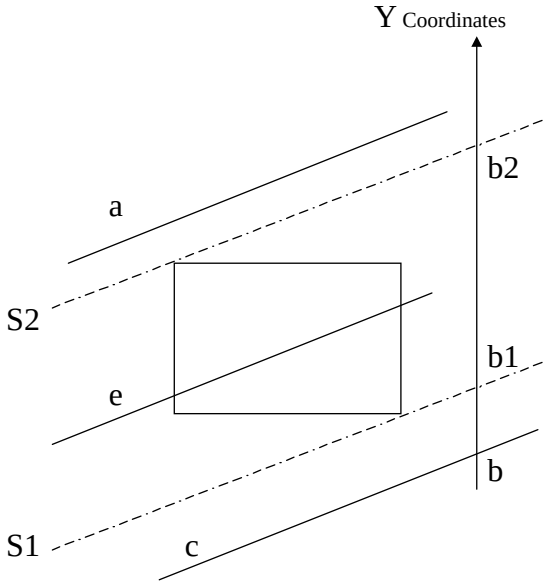

Fig. 6. Rejecting redundant lines by *V-Window*.

$$k = \frac{ya - yb}{xa - xb}, \tag{12}$$

where k is the slop of the line. By now, all redundant lines have been rejected.

4.2. Fast intersection calculation for partially visible lines

In the case of lines that intersect the clipping window, such as line e in Fig. 6, we reduce the size of the *V-Window* so as to find out which edges of the original clipping window may have intersections with the lines. Two cases arise depending on the slop of the line being clipped. Suppose k is the slop of the line and, *K_WINDOW* is the positive slop of the original window's diagonal, we select positive k for further test as follows:

```
if (k < = K_WINDOW)
   the broken lines in Fig. 7 constitute the reduced V-Window.
else
    the broken lines in Fig. 8 constitute the reduced V-Window.
```

By comparing the y-intercept of the line with that of lines S3 and S4, two edges of the reduced *V-Window* shown in Figs. 7 or 8, the exact edges of the original window that intersect with the line can be accurately determined as follows:

```
if (b< = b3)→line can only intersect with right edge and bottom edge (line c in Figs. 7 and 8)
else if (b> = b4)→line can only intersect with left edge and top edge (line a in Fig. 7 and 8)
else {
   if (k< = K_WINDOW) →line can only intersect with left edge and right edge (line e in Fig. 7)
    else {……}→line can only intersect with top edge and bottom edge (line e in Fig. 8)
}
```

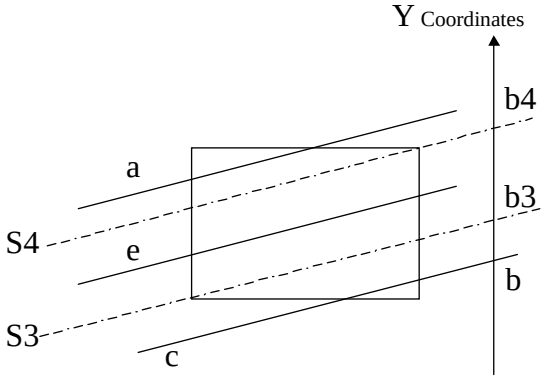

Fig. 7. Reduced *V-Window*-1.

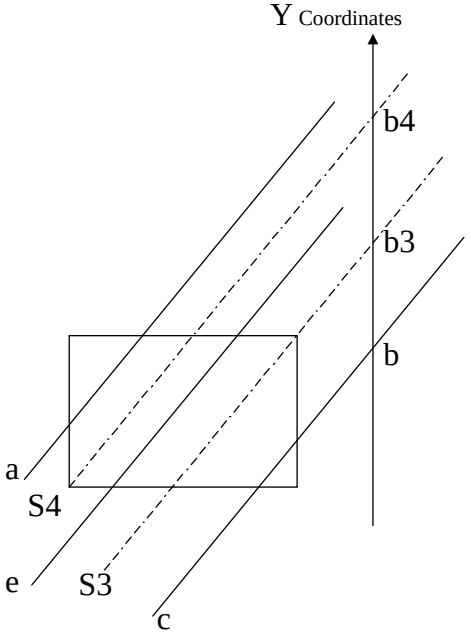

Fig. 8. Reduced *V-Window*-2.

Here, b3 and b4 are y-intercepts of S3 and S4, and are determined as follows:

$$\text{If } (k > = K\_WINDOW)\{$$
$$b3 = b1 + YT - YB; \tag{13}$$

$$b4 = b2 + YT - YB; \quad (14)$$

}
Else{

$$b4 = b1 + YT - YB; \quad (15)$$

$$b3 = b2 - YT - YB; \quad (16)$$

}

The results of the six cases of intersection calculation are listed bellow:

Line *a* of Figs. 7 and 8:
(XL, YT−b2+b), (XL+(b2−b)/k, YT)
Line *c* of Figs. 7 and 8:
(XR, YB+b−b1), (XR−(b−b1)/k, YB)
Line *e* of Fig. 7:
(XL, YB+b−b3), (XR, YT−b4+b)
Line *e* of Fig. 8:
(XL+(b4−b)/k, YB), (XR−(b−b3)/k, YT)

Note that, lines perpendicular to *X*-axis belong to a special subset, which need separate treating. Suppose (xa, ya), (xb, yb) are two endpoints of the line to be clipped, and *xa* is equal to *xb*, the endpoints of the line after clipping is (*xa*, *YB*) (*xb*, *YT*).

5. Implementations of the new clipping algorithm

We implemented the new algorithm on PC and compared its performance with that of *Cohen–Sutherland* algorithm, *C–S* for short. Our machine is based on Pentium II 350 MHz, and the compiler is Turbo C 2.0 running on Windows 2000. Time statistics of the two clipping algorithms is presented in Tables 2–5.

Lines are divided into four groups during the three steps of the new algorithm. The first group, of which lines are accepted or rejected by *Encoding & Code Checking* technique, is processed at the same efficiency either by *C–S* or by our new algorithm, so the results concerning this kind of lines are not listed here. Time costs of identifying the remaining three groups with *S-Window* and *V-Window*, respectively, are presented in Tables 2–4. For each case, we selected 6000 lines that satisfy the case specification and clipped them both with C–S algorithm and our new algorithm. To ensure a reliable result, the clipping process was repeated 1000 times.

The statistics listed in the above three tables show clearly that our new algorithm is much more efficient in clipping most kinds of lines than C–S algorithm.

Table 5 shows the result of a comprehensive experiment in which we choose 6000 lines which are selected according to the following criteria:

1. One thousand lines which can be accepted by traditional *Encoding & Code Checking* technique.
2. One thousand lines which can be rejected by traditional *Encoding & Code Checking* technique.
3. One thousand lines which can be rejected by *S-Window*.
4. One thousand lines which can be rejected by *V-Window*.
5. One thousand lines with one endpoint inside the clipping window and the other outside it.

Table 2
Time costs for lines that can be rejected by *S-Window* (in seconds)

Case specification	Performance		
	C–S algorithm (s)	New algorithm (s)	Ratio of time consuming, $\frac{New}{C-S}$
Six thousand redundant lines that can only be rejected by C–S algorithm after one time of intersection calculation	9.61	13.19	0.595
Six thousand redundant lines that can only be rejected by C–S algorithm after two times of intersection calculation	5.72	5.43	0.412

Table 3
Time costs for lines that can be rejected by *V-Window* (in seconds)

Case specification	Performance		
	C–S algorithm (s)	New algorithm (s)	Ratio of time consuming, $\frac{New}{C-S}$
Six thousand redundant lines that can only be rejected by C–S algorithm after one time of intersection calculation	9.01	9.00	1.00
Six thousand redundant lines that can only be rejected by C–S algorithm after two times of intersection calculation	11.98	7.99	0.667

Table 4
Time costs for partially visible lines that can be clipped by reduced *V-Window* (in seconds)

Case specification	Performance		
	C–S algorithm (s)	New algorithm (s)	Ratio of time consuming, $\frac{New}{C-S}$
Six thousand redundant lines that can only be rejected by C–S algorithm after two time of intersection calculation	12.12	13.50	1.11
Six thousand redundant lines that can only be rejected by C–S algorithm after three times of intersection calculation	17.17	13.64	0.794
Six thousand redundant lines that can only be rejected by C–S algorithm after four times of intersection calculation	21.69	14.45	0.666

Table 5
Comprehensive comparisons of the new algorithm with *C–S* algorithm (in seconds)

	Time costs for 1000 times repeatedly clipping
New algorithm (s)	9.23
C–S algorithm (s)	7.31
Ratio of time consuming, $\frac{New}{C-S}$	0.792

6. One thousand partially visible lines which can be clipped with reduced *V-Window*.
7. Statistics in Table 5 suggests the potentials of our new algorithm.

6. Summary

Line clipping is a fundamental operation of computer graphics. Until recently, most works are concentrated on accelerating the intersection calculation so as to improve the clipping efficiency [7–10]. In this paper, a new clipping algorithm is presented, which pays more attention to the efficiency of line rejection. With adaptively constructed *S-Window* and *V-Window*, our algorithm can quickly reject all invisible lines, which may otherwise incur unnecessary intersection calculations in traditional clipping algorithms.

Moreover, the exact edges of the original clipping windows, which intersect with a partially visible line can be easily identified by our algorithm with a reduced *V-Window*. It is expected that our strategy of adaptive line rejection with different windows can be applied to other clipping operations, and can then improve the clipping efficiency in a broader circumstance.

Acknowledgements

This work received support from the National Natural Science Foundation of China for Distinguished Young Scholar (Grant No. 69925204), and the Natural Science Foundation of China for Innovative Research Groups (Grant No. 60021201).

References

[1] Sproull RF, Sutherland IE. A clipping divider. In: Proceedings of the Fall Joint Computer Conference. Washington: Thompson Books, 1968. p. 765–75.
[2] Rogers DF. Procedural elements for computer graphics. New York: McGraw-Hill, 1985. p. 111–87.
[3] Andreev R, Sofianska E. New algorithm for two-dimensional line clipping. Computers and Graphics 1991;15(4):519–26.
[4] Nicholl TM, Lee DT, Nicholl RA. A efficient new algorithm for 2-D line clipping: its development and analysis. Computer Graphics 1987;21(4):253–62.
[5] Shi KJ, Edwards JA, Cooper DC. An efficient line clipping algorithm. Computers and Graphics 1990;14(2):297–301.
[6] Sobkow MS, Pospisil P, Yang YH. A fast two-dimensional line clipping algorithm via line encoding. Computers and Graphics 1987;11(4):459–67.
[7] Sharma NC, Manohar S. Line clipping revisited: two efficient algorithm based on simple geometric observations. Computers and Graphics 1992;16(1):51–4.
[8] Day JD. A new two dimensional ling clipping algorithm for small windows. Computer Graphics Forum 1992;11(4):241–5.
[9] Wang Haohong, Wu Ruixun, Cai Shijie. A new efficient clipping algorithm based on geometric transformation. Journal of Software 1998;9(10):728–33 (in Chinese).
[10] Wang Jun, Liang Youdong, Peng Qunsheng. A 2-D line clipping with the least arithmetic operations. Chinese Journal of Computers 1991;(7):495–504 (in Chinese).