Journal of Zhejiang University SCIENCE
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn

JZUS

Line clipping against polygonal window algorithm based on the multiple virtual boxes rejecting*

WANG Jin (王 进)[†1], LU Guo-dong (陆国栋)[†‡1], PENG Qun-sheng (彭群生)[1], WU Xuan-hui (吴煊晖)[2]

([1]*State Key Lab. of CAD&CG, Zhejiang University, Hangzhou 310027, China*)

([2]*Department of Electrical and Computer Engineering, National University of Singapore, 117576, Singapore*)

[†]E-mail: dwjcom@cmee.zju.edu.cn; lugd@zju.edu.cn

Received Feb. 10, 2005; revision accepted July 10, 2005

Abstract: This paper presents a new algorithm for line clipping against a polygonal window by exploiting the local relationship between each line segment and the polygon. Firstly, a minimal enclosing box (MEB) of the polygon is adopted to reject the invisible line segments located outside the MEB. Secondly, a 45° rotated box is used to encode the endpoint of the line segment, and then reject a portion of the invisible segments crossing polygon corners. Finally, instead of encoding the endpoints of all line segments with respect to the polygonal window, each vertex of the polygon is encoded, taking the line segment to be clipped as reference. For efficient encoding of the polygon vertices, a new concept, termed with slope adaptive virtual box, is introduced regarding each line segment. Such a box can not only conveniently reject all totally invisible lines lying outside the MEB conveniently, but also precisely identify the edges of the polygon with which the line segment potentially intersects. With the summation of the vertex codes, it can be verified whether the line segment is separated from or potentially intersects the polygon window. Based on the product of the codes of adjacent vertices, singular cases of intersection can be solved accurately. Experimental results demonstrate the efficiency and stability of the new algorithm.

Keywords: Polygon vertex encoding, Adaptive virtual box, Line segment rejection, Line segment intersection, Line clipping
doi:10.1631/jzus.2005.AS0100 **Document code:** A **CLC number:** TP312

INTRODUCTION

Line clipping is a fundamental operation whose efficiency directly affects the performance of a whole graphics system. There are already many efficient line clipping algorithms against a rectangular window such as the classical Cohen-Sutherland algorithm (Newman and Sproull, 1979), the midpoint-subdivision algorithm (Sproull and Southerland, 1968), the LB algorithm (Liang and Barsky, 1984), the NLN algorithm (Nicholl *et al.*, 1987), the ELC algorithm (Wang *et al.*, 1998), and the FLC algorithm (Wang *et al.*, 1991), etc.

Given a rectangular clipping window, each edge and its extension divides the entire plane into two parts, one with the clipping window, and one without the window. Most algorithms for line clipping against a rectangular window commonly adopt an encoding technique for rapid rejection of invisible lines that locate completely in the half plane without the clipping window. Recently, a supplementary endpoint encoding technique based on a concept of virtual window was developed (Lu *et al.*, 2002; Lu and Wu, 2002). Invisible line segments crossing the corner of the window, which cannot be identified by the traditional endpoint encoding method, can now be rejected easily.

In contrast, algorithms for polygon clipping are rather few. A classic algorithm CB proposed by Cyrus and Beck (1978) employs a parameterized approach by examining the normal dot product. The cost of clipping against a convex window is almost constant

‡Corresponding author

*Project supported by the National Natural Science Foundation of China (No. 60021201), and the Research Fund for the Doctoral Program of Higher Education (No. 2002335093), China

in (Skala, 1993), and the algorithm efficiency (Skala, 1993) can be practically improved 2.5 to 3 times compared to CB, and can be improved further by cutting down the computational cost form $O(N)$ to $O(\lg N)$ (Bui and Skala, 1999). However, these algorithms are applicable for a convex polygonal window only. Recently, there appeared some algorithms for concave polygon clipping (Liu *et al.*, 1999; Liu and Liu, 1993). Similar to what is done in the case of a rectangular window, algorithms for polygon clipping commonly adopt a minimal enclosing box (MEB) enclosing the polygon window to encode the endpoints of line segments to be clipped. However, the technique of encoding with an enlarged virtual window only rejects a portion of line segments efficiently. Those line segments inside the MEB as well as other remaining ones have to be handled by later intersection operations.

Apparently, line clipping against a polygon, especially a concave polygon, is more difficult than that against a rectangle. Traditional methods for the clipping against a polygon are to some extent extensions of those against a rectangle. Therefore, it is necessary to study the unique characteristics of the clipping against a polygon, and develop some algorithms more adaptive to different clipping polygons.

The authors deeply studied the polygonal clipping window. With the aim of determining the local correlation of the line segment and the polygonal window, several virtual boxes, namely MEB virtual box, 45° rotated virtual box, and line slope adaptive virtual box, were used to reject different kinds of invisible line segments samples of which are shown in Fig.1. In addition, the line slope adaptive virtual box was used to encode the polygon vertices by taking the line segment as reference. The sum and product of the vertices' codes were used for further clipping and rapid intersection calculation, and handle singular cases accurately. As a result, the new algorithm achieves the following three goals of all clipping algorithms:

(1) Avoid unnecessary intersection calculations;

(2) Speed up the intersection process;

(3) Handle singular intersection cases accurately and efficiently.

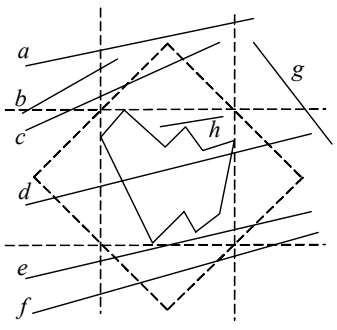


Fig.1 Line segments types

REJECTIONG MOST OF THE INVISIBLE LINE SEGMENTS WITH THREE DIFFERENT VIRTUAL BOXES

Rejecting totally invisible line segments with the MEB and 45° rotated virtual boxes

Fig.1 shows several different types of line segments such as *a* and *g* locating outside the MEB, which can be rejected by using the Cohen-Sutherland encoding algorithm shown in Fig.2.

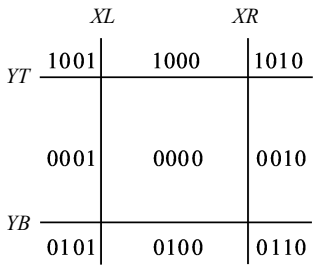


Fig.2 Cohen-Sutherland encoding

A large portion of invisible line segments such as *b* in Fig.1 intersecting the MEB can be rejected by the 45° rotated virtual box proposed by Lu *et al.*(2002) and shown in Fig.3. Comparison of Fig.2 and Fig.3 shows that the codes obtained by using the 45° rotated virtual box are identical to those obtained by using the MEB, so that the rejection of invisible lines segments is similar to that by using the Cohen-Sutherland (CS) algorithm.

Rejecting totally invisible line segments with slope adaptive virtual box

1. Adaptive virtual box construction according to the line slope

The traditional endpoint encoding technique takes the window as reference to encode the endpoints of the line segments. It is therefore difficult for this

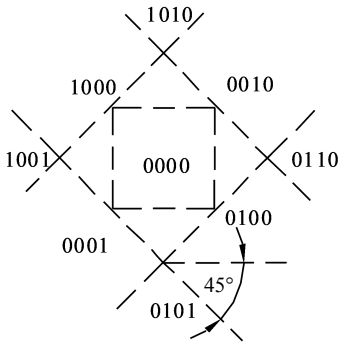


Fig.3 45° rotated virtual box encoding

kind of algorithms to reject totally invisible line segments such as *c* and *f* in Fig.1 that cross the corner of the MEB. We construct an adaptive virtual box for each line segment to be clipped in order to better exploit the local correlation between the line segment and the clipping polygon.

Note that, if a line segment is potentially visible within the polygon window, it has two intersection points with the MEB on the line segment or its extension. This suggests using the correlativity of these two intersection points and the polygon vertices to reject invisible line segments more efficiently. Specifically, if the slope of the line segment is k, and if $|k|\leq 1$, given P_1 and P_2 as two intersection points of the line segment or its extension against the left and right boundaries of the MEB, respectively, the slope adaptive virtual box is constructed in such a way that its diagonal points are P_1 and P_2, as shown in Figs.4a and 4b. If $|k|>1$, the diagonal points of the adaptive virtual box, also named as P_1 and P_2, are obtained by intersecting the line segment or its extension with the top and bottom boundaries of the MEB, as shown in Figs.4c and 4d. Note that, the slope adaptive virtual box is not the traditional enclosing box of the line segment, but the one adaptive to the MEB and the slope of the line segment to be clipped.

Note that if the line segment slope $k=\infty$, that is $P_1.x$ equals to $P_2.x$, P_1 and P_2 can be acquired without intersection operation.

2. Utilizing the adaptive virtual box to reject the invisible line segments

With the adaptive virtual box constructed above, many invisible lines that have no intersection with the MEB can be rejected. Clearly, when the adaptive virtual box is located outside the MEB, the line segment to be clipped is invisible and should be rejected. Specifically, as shown in Fig.5, if the slope of the line segment satisfies $|k|\leq 1$, and the adaptive virtual box is above or below the MEB, the line segment is invisible. A line segment is also invisible if the adaptive virtual box is totally on the left or right side of the MEB when $|k|>1$. The above rejection test can be performed by simple numerical value comparison according to Table 1. It can be found that, after the above rejection tests, all the remaining line segments either totally lie

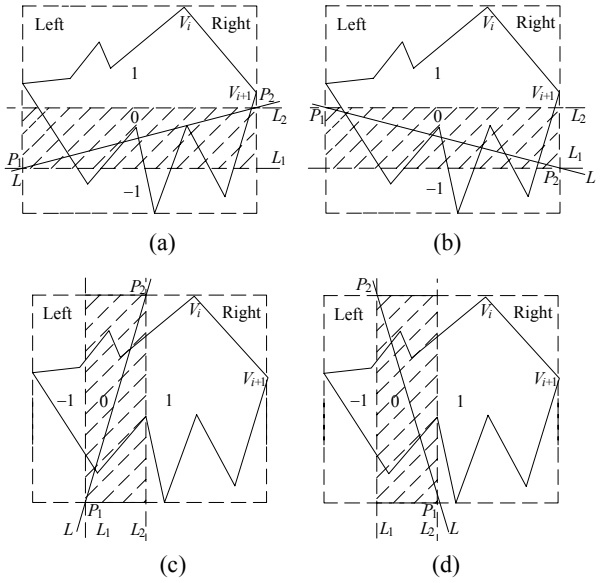


Fig.4 Adaptive virtual box construction
(a) $0\leq k\leq 1$; (b) $-1\leq k\leq 0$; (c) $k>1$; (d) $k<-1$

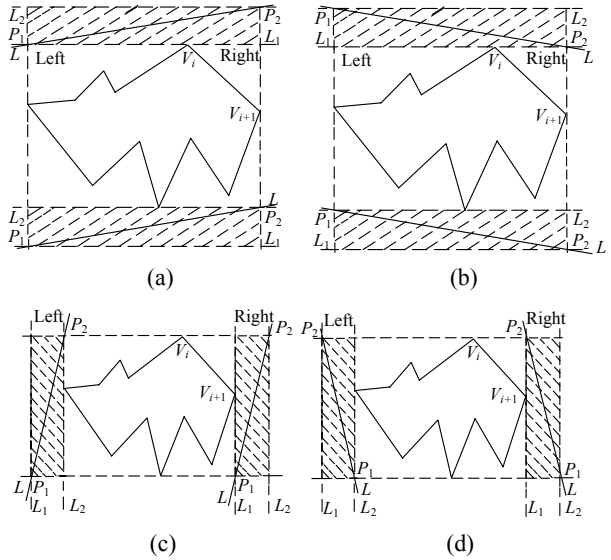


Fig.5 Adaptive virtual box rejects invisible line segments
(a) $0\leq k\leq 1$; (b) $-1\leq k\leq 0$; (c) $k>1$; (d) $k<-1$

inside the MEB or intersect it.

In Table 1, *LT* and *RB* are the top left corner and the bottom right corner vertices of the adaptive virtual box. *LT.x*, *LT.y*, *RB.x*, *RB.y* are the *x* and *y* coordinates of *LT* and *RB*.

Table 1 Simple conditions for rejecting invisible line segments

Line segment slope	Area where the invisible lines locate
$0\le k\le 1$	($P_1.y>LT.y$) or ($P_2.y<RB.y$)
$-1\le k\le 0$	($P_1.y<RB.y$) or ($P_2.y>LT.y$)
$k>1$	($P_1.x>RB.x$) or ($P_2.x<LT.x$)
$k<-1$	($P_1.x<LT.x$) or ($P_2.x>RB.x$)

POLYGON VERTICES ENCODING TACKING THE LINE SEGMENT AS REFERENCE

A line segment and its extension, and expression $ax+by+c=0$, divide the 2D plane into positive, negative, and zero regions, based on the sign of ax_0+by_0+c, where x_0 and y_0 are the coordinates of a point in the plane. With the line segment and its extension dividing the entire plane, any point in the plane including the vertices of the clipping window is thus encoded as 1, −1 or 0, if it is located in the positive, negative or zero region.

However, the encoding of a vertex by directly substituting the vertex coordinates into the line equation requires much computation. So the adaptive virtual box of the line segment is employed instead for initial point classification. As shown in Fig.4, the 2D plane is divided into positive, negative, and near-zero three regions by the adaptive virtual box (the near-zero region is labelled as code "0" and is shaded in Fig.4). The near-zero region is just within the adaptive virtual box and vertices within the near-zero region must undergo further classification. While those vertices located in the positive and negative regions with respect to the adaptive virtual box can be directly encoded as 1 and –1.

The conditions of judging whether a vertex is inside the positive or negative region during the initial test are straightforward, and listed in Table 2. V_i is an arbitrary vertex of the polygon, $V_i.x$ and $V_i.y$ are the coordinates of V_i, and f_i is the vertex code. P_1 and P_2 are the diagonal vertices of the adaptive virtual box, as shown in Fig.4. ($P_1.x$, $P_1.y$) and ($P_2.x$, $P_2.y$) are the coordinates of P_1 and P_2, respectively, assuming $P_2.y\ge P_1.y$.

Table 2 Encoding of the polygon vertices in positive and negative regions

Slope	Encoding	
	$f_i=1$	$f_i=-1$
$0\le k\le 1$	$V_i.y>P_2.y$	$V_i.y<P_1.y$
$-1\le k\le 0$	$V_i.y>P_1.y$	$V_i.y<P_2.y$
$k>1$	$V_i.x>P_2.x$	$V_i.x<P_1.x$
$k<-1$	$V_i.x>P_1.x$	$V_i.x<P_2.x$

FAST CLIPPING LINE SEGMENT BASED ON INDIVIDUAL VERTEX ENCODING AND CODE INFORMATION

Fast line segments rejecting based on the sum of the vertex codes

After the rejection of totally invisible line segments by making use of the MEB, the 45° rotated virtual box, and the slope adaptive virtual box, there are still some totally invisible line segments left, such as *e* shown in Fig.1. Traditional line clipping algorithms that take the polygon as reference to encode the line endpoints cannot easily reject such line segments.

The summation of the codes for all the polygon vertices can be used to further reject more totally invisible line segments. Let $F=\sum_{i=1}^{N} f_i$, where N is the number of the polygon vertices and f_i the code for the ith vertex, if $F=\pm N$, the line segment has no intersection with the polygon, such as line segments *e* in Fig.1, and can be rejected.

Fast intersection based on the product of the vertex codes

Line segments *d* and *h* or their extension intersect with the polygon, as shown in Fig.1. But they do not intersect with all the polygon edges, so the edges intersecting with the line segment should be determined efficiently.

Given an arbitrarily selected edge of the polygon V_iV_{i+1}, where V_i with the code f_i and V_{i+1} with the code f_{i+1}, are two endpoints of the edge. Obviously, there are three possible cases of their product as follows,

$$f_i f_{i+1}=1 \quad (1)$$
$$f_i f_{i+1}=-1 \quad (2)$$
$$f_i f_{i+1}=0 \quad (3)$$

If Eq.(1) is satisfied, the line segment has no intersection with the polygon. If Eq.(2) is satisfied, the line segment may intersect with the polygon. If Eq.(3) is satisfied, the line segment intersects with the polygon singularly, that is the segment passes through one or several vertices of the polygon and vertex V_i is an intersection point if f_i=0.

Note that, if the segment passes an arbitrary vertex V_i, we only need do intersection operation with one adjacent edge, $V_{i-1}V_i$ or V_iV_{i+1}.

Now, only the necessary intersection calculation is to be performed, with technical details are omitted. (If the slope of the line segment is infinite or the slope of the polygon edge is infinite, it must be dealt with by special codes if we use equation $y=kx+b$ to do intersection operation.)

Singular intersection handling based on the product of the vertex codes

Singular intersections should be handled properly as some of the intersection points may be invalid. Thus, further identification of the singular cases satisfying Eq.(3) is important.

Assuming an arbitrary polygon edge V_iV_{i+1}, with V_i's code f_i=0, there are three possible cases shown in Eqs.(4)~(6), where f_{i-1} and f_{i+1} are the codes of V_{i-1} and V_{i+1}, respectively.

$$f_{i-1}f_{i+1}=-1 \quad (4)$$
$$f_{i-1}f_{i+1}=1 \quad (5)$$
$$f_{i-1}f_{i+1}=0 \quad (6)$$

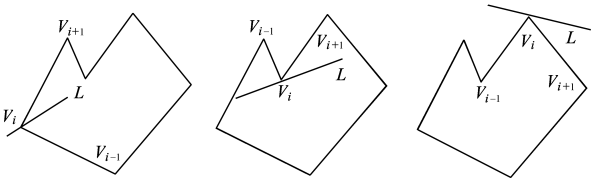
The two polygon edges $V_{i-1}V_i$ and V_iV_{i+1} locate at the two sides of the line segment when Eq.(4) is satisfied, as shown in Fig.6a. Save such point as V_i is a valid point of intersection and it should be recorded in DISPLAY-ARRAY, an array for recording all the valid intersection points for displaying the visible part of the line segment to be clipped.

The two polygon edges $V_{i-1}V_i$ and V_iV_{i+1} locate at the same side of the line segment when Eq.(5) is satisfied, as shown in Figs.6b and 6c. Such intersection point V_i is ignored as it does not affect the result of displaying the visible part of the line segment whether the line segment lies inside or outside the polygon.

If Eq.(6) is satisfied, the line segment passes through one of the polygon edges. Two different cases should be discussed separately.

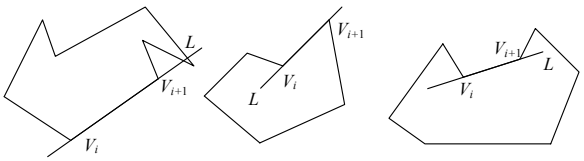
(1) The intersection point is a convex vertex, as points V_i and V_{i+1} in Fig.7a, and point V_{i+1} in Fig.7b. Such intersection points should be recorded in DISPLAY-ARRAY.

(2) The intersection point is a concave vertex, as point V_i in Fig.7b, points V_i and V_{i+1} in Fig.7c. Such intersection points should be ignored.



(a) (b) (c)

Fig.6 The line segment goes through the polygon vertex



(a) (b) (c)

Fig.7 The line segment goes through the polygon edge

Here is a method to determine whether the intersection point is a convex vertex or a concave one. Note that the leftmost, rightmost, topmost, and bottommost vertex of the polygon is a convex vertex. Arrange these points and make them into a polygon, and call this polygon EXTREM-POLYGON. If the product $R_iR_k>0$, point V_i is a convex point and should be saved. Otherwise, it is a concave point of intersection and should be rejected. R_i and R_k are cross products as shown in Eq.(7) and Eq.(8).

$$R_i=\boldsymbol{V}_{i-1}\boldsymbol{V}_i\times\boldsymbol{V}_i\boldsymbol{V}_{i+1} \quad (7)$$
$$R_k=\boldsymbol{V}_{k-1}\boldsymbol{V}_k\times\boldsymbol{V}_k\boldsymbol{V}_{k+1} \quad (8)$$

where $\boldsymbol{V}_{i-1}\boldsymbol{V}_i$ or $\boldsymbol{V}_i\boldsymbol{V}_{i+1}$ is a vector pointing from V_{i-1} to V_i, or from V_i to V_{i+1}; $\boldsymbol{V}_{k-1}\boldsymbol{V}_k$ or $\boldsymbol{V}_k\boldsymbol{V}_{k+1}$ is a vector pointing from V_{k-1} to V_k, or from V_k to V_{k+1}; V_{k-1}, V_k and V_{k+1} are the adjacent vertices in EXTREM-POLYGON.

SHOW THE VISIBLE PART OF THE LINE SEGMENT

Finally, we have to show the visible part of the line segment correctly.

(1) Sort the intersection points

All the intersection points should be sorted by x or y coordinate, if the slope of the line segment to be clipped is infinite, sort the intersection point in y coordinate.

(2) Discard the intersection points on the extension part of the line segment

Obviously, intersection points that locate on the extension of the line segment are invalid and should be rejected. Especially if all intersection points of the line segment against the polygon are invalid, such as line segment c in Fig.8, the line segment is completely invisible.

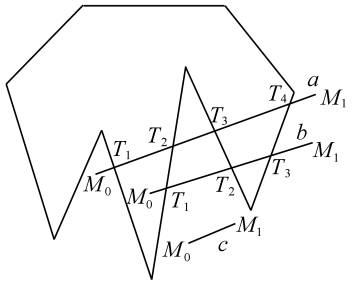


Fig.8 Display the line segment

(3) Test whether the endpoint of the line segment lies in the polygon

Let the left endpoint of the line segment be M_0, and the right endpoint be M_1, as shown in Fig.8. Let variable *Count*1 records the number of the intersection points to the right of M_0. If *Count*1 is odd, M_0 is in the polygon. Otherwise, it is outside the polygon. Whether the right endpoint lies inside the polygon can be determined in a similar way. If $M_0.x$ equals to $M_1.x$, let the M_0 be the bottom endpoint of the line segment and M_1 be the top endpoint, *Count*1 records the intersection points number that are on top of M_0.

(4) Show the visible section of the line segments correctly according to the positions of M_0 and M_1, as shown in Table 3.

IMPLEMENTATIONS AND DISCUSSION

The algorithm is implemented with Turbo C 2.0 on Pentium IV 2.0 GHz PC, and compared with Liu Yong-Kui algorithm, LIU for short. The flow chart is shown in Fig.9.

Table 4 lists the time durations in seconds of clipping 10000 line segments 1000 times by Liu algorithm and our algorithm. The time for sorting and displaying the intersection points are excluded in order to better compare the performance of the two

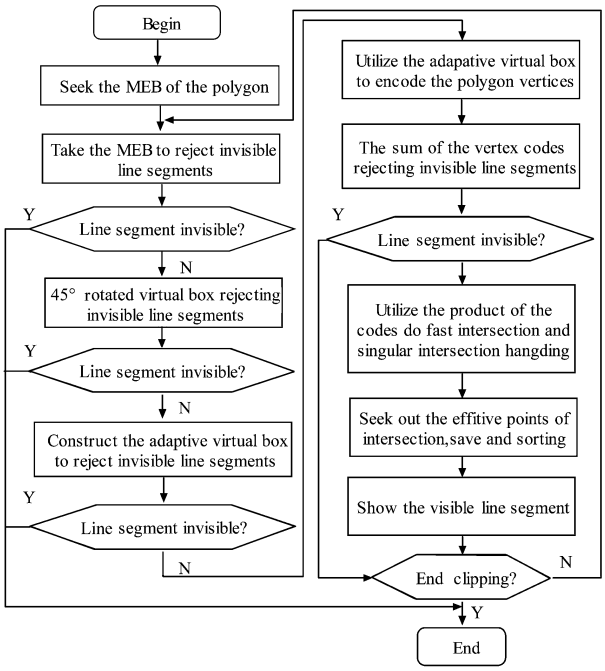


Fig.9 Flow chart of the algorithm

Table 3 Show the visible section of the line segment

Position of M_0 and M_1		Line segment array for displaying
M_0 is out of the polygon	M_1 is out of the polygon	$\{L(T_1,T_2);L(T_3,T_4);...;L(T_{2i+1},T_{2i+2})\}$
	M_1 is in the polygon	$\{L(T_1,T_2);L(T_3,T_4);...;L(T_{2i+1},M_1)\}$
M_0 is in the polygon	M_1 is out of the polygon	$\{L(M_0,T_1);L(T_2,T_3);...;L(T_{2i},T_{2i+1})\}$
	M_1 is in the polygon	$\{L(M_0,T_1);L(T_2,T_3);...;L(T_{2i}, M_1)\}$

algorithms.

Experiment conditions:

(1) The polygon size

The polygon size is described by perimeter and area. Assume that the dynamic ranges of the x and y coordinates for line segment endpoints are 800 and 600, respectively. The selection of such figures is based on the fact that the resolution of a normal computer screen is about 800×600 pixels, supposing the scale of the world coordinate system is equal to that of the screen coordinate system, so the line segments' range is regarded to be the same as that of a computer screen.

Assume $W_1=600/L$, and $W_2=600\times800/[(Y_{max}-Y_{min})\times(X_{max}-X_{min})]$, where L is the perimeter of the MEB, $[(Y_{max}-Y_{min})\times(X_{max}-X_{min})]$ is the area of the MEB. It is easy to know that the diagonal length of the 45° rotated virtual box is $L/2$. The bigger L is, the lower is the efficiency for 45° rotated virtual box to reject invisible line segments. Similarly, W_2 describes the efficiency of the 45° rotated virtual box to reject invisible line segments from the area aspect. So W_1 and W_2 are used to describe the polygon size. Table 4 lists the W_1 and W_2 for each test polygon window.

(2) Polygon shapes, are shown in Fig.10

Coefficient "*Gain*" is used for comparing the efficiency comparison coefficient between Liu's algorithm and our new algorithm, see Eq.(9).

$$Gain=(Liu-New2)\times100\%/New2 \quad (9)$$

where "*Liu*" means the time consumption of Liu's algorithm for clipping 10000 random line segments 1000 times, "*New*2" means the time consumption of our new algorithm with 45° rotated virtual box rejecting.

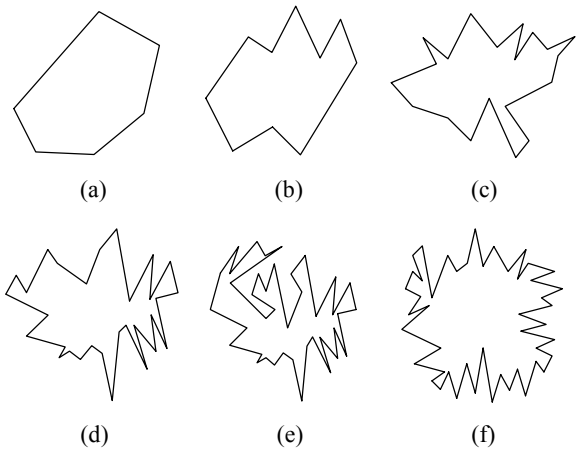


Fig.10 Polygon shapes in the experiment

Table 4 data show that the new algorithm has higher efficiency. Given W_1 and W_2 equal to each other in all the testing cases, and that when the new algorithm does not use the 45° rotated virtual box to reject invisible line segments, it is about 7.9%, obtained by Eq.(9), is faster than the Liu's algorithm when the polygon has six edges, and it goes up to 63.7% when the polygon has 50 edges. If the 45° rotated virtual box is used to reject invisible line segments, our algorithm is more efficient, and is about 25% faster, than the Liu's algorithm when the polygon has six edges. That figure goes up to 125%

Table 4 Time durations of clipping 10000 random line segments 1000 times

Polygon edges' number	Polygon sizes	*Liu*[1]	*New*1[2]	*New*2[3]	*Gain*[4]
6	W_1=2.76, W_2=40	8.56	7.36	6.32	34.81
	W_1=4, W_2=80	8.30	7.69	6.87	20.81
10	W_1=2.53, W_2=34.5	13.63	11.70	9.39	45.15
	W_1=4, W_2=80	10.66	8.73	7.91	34.77
20	W_1=4.22, W_2=96.8	16.10	10.10	7.91	103.54
	W_1=4, W_2=80	18.08	11.15	8.84	104.52
30	W_1=4, W_2=85.7	18.57	14.83	8.35	122.39
	W_1=4, W_2=80	21.15	12.74	9.78	116.26
40	W_1=3.57, W_2=68	25.06	21.15	11.54	117.16
	W_1=4, W_2=80	27.47	15.77	11.48	139.28
50	W_1=3.37, W_2=62.6	32.92	23.90	15.16	117.15
	W_1=4, W_2=80	31.20	19.07	13.85	125.27

[1]*Liu*: Liu's algorithm; [2]*New*1: New algorithm without 45° rotated virtual box rejecting; [3]*New*2: New algorithm with 45° rotated virtual box rejecting; [4]*Gain*=(*Liu*–*New*2)×100%/*New*2;

when the polygon has 50 edges.

If the clipping is only for convex polygon, the efficiency can be increased more by terminating the intersection operation after the determination of the second intersection point. Numerical experiments showed that the new algorithm can identify and handle the singular cases of intersection correctly.

CONCLUSION

Line clipping against a polygon is more complicated than that against a rectangle. It is a basic operation of polygon clipping against polygon. Compared to the line clipping against a rectangle, fewer studies have been made on the line clipping against a polygon, and excellent algorithms are rather few. This paper presents a new concept that takes the line segment as reference to encode the polygon vertices, and introduces an adaptive virtual box to efficiently reject more totally invisible line segments after the use of MEB virtual box and 45° rotated virtual box. The new algorithm thoroughly makes use of the correlative position relation of the line segment and the polygon. Experimental results showed that the new algorithm has high efficiency and good stability.

References

Bui, D.H., Skala, V., 1999. New Fast Line Clipping Algorithm in E^2 with $O(\lg N)$ Complexity. International Conferences SCCG'99, Budmerice, Slovak Republic, p.221-228.

Cyrus, M., Beck, J., 1978. Generalized two and three dimensional clipping. *Computers & Graphics*, **3**(1):23-28.

Liang, Y.D., Barsky, B.A., 1984. A new concept and method for line clipping. *ACM Trans. Graphics*, **3**(1):1-22.

Liu, Y.K., Liu, G.F., 1993. Line clipping against generalized polygon. *Journal of Computer-Aided Design & Computer Graphics*, **5**(4):269-274 (in Chinese).

Liu, Y.K., Yan, Y., Shi, J.Y., 1999. An efficient algorithm for the line clipping against a polygon. *Chinese Journal of Computers*, **22**(11):1210-1214 (in Chinese).

Lu, G.D., Wu, X.H., 2002. Midpoint-subdivision line clipping algorithm based on filtering technique. *Computer-Aided Design & Computer Graphics*, **14**(6): 513-517 (in Chinese).

Lu, G.D., Wu, X.H., Peng, Q.S., 2002. An efficient line clipping algorithm based on adaptive line rejection. *Computers & Graphics*, **3**(26):409-415.

Newman, W.M., Sproull, R.F., 1979. Principles of Interactive Computer Graphics. McGraw-Hill, New York.

Nicholl, T.M., Lee, D.T., Nicholl, R.A., 1987. An efficient new algorithm for 2D line clipping: Its development and analysis. *Computer Graphics*, **21**(4):253-262.

Skala, V., 1993. Efficient algorithm for line clipping by convex polygon. *Computers & Graphics*, **17**(4):417-421.

Sproull, R.F., Southerland, I.E., 1968. A Clipping Divider. Proceedings of the Fall Joint Computer Conference, **33**:765-776.

Wang, J., Liang, Y.D., Peng, Q.S., 1991. A 2-D line clipping algorithm with least arithmetic operations. *Chinese Journal of Computers*, **14**(7):495-504 (in Chinese).

Wang, H.H., Wu, R.X., Cai, S.J., 1998. A new efficient line clipping algorithm based on Geometric transformation. *Journal of Software*, **9**(10):728-733 (in Chinese).