基于矩阵乘法的多边形窗口线裁剪算法

黄文钧
(广西民族大学信息科学与工程学院 南宁 530006)

摘 要 提出一种任意多边形窗口线裁剪新方法,它不解方程而通过矩阵乘法得到窗口和线段的交点。对于一组待裁剪线段,该方法先做简单的包围盒预处理,将那些和包围盒无交的线段排除在求交之外;然后引进齐次坐标,构造一组仿射变换矩阵,通过矩阵乘法对任意多边形窗口和待裁剪线段实施连续仿射变换,完成窗口和直线求交操作并从矩阵中获得交点;经过交点排序、配对等过程,得到多边形裁剪线段的结果。经实验对比,该新方法有效,并且速度得以提高。

关键词 裁剪,仿射变换,矩阵,多边形,窗口
中图法分类号 TP39 **文献标识码** A

Matrix Multiplication for Line Clipping of Polygon

HUANG Wen-jun
(College of Information Science and Engineering, Guangxi University for Nationalities, Nanning 530006, China)

Abstract This paper proposed a new method for line clipping with a polygon. The method gets the intersection points of a polygon and a line by matrix multiplication. For a set of line segments, the algorithm of this paper tests it by a bounding box that included the polygon to discard the line segments which do not intersect the box, then the algorithm introduces homogeneous coordinates, and makes a group of matrixes and applies the matrixes multiplication to the polygonal window and the straight line to make the continuous affine transformations, and gets the intersection points of the window and straight line from the matrix. Having sorted and matched the points of intersection, the algorithm of this paper obtains the result of the polygon clipping the line segment. The experiment shows that the new method is effective, and the speed is improved.

Keywords Clipping, Affine transformation, Matrix, Polygon, Window

1 引言

计算机图形学的二维裁剪,其裁剪窗口有凸多边形和凹多边形。关于凸多边形窗口的线裁剪问题[1-5]已经得到充分讨论;任意多边形窗口的研究[6-12]也得到很大关注,一些文献[11,12]提出的算法新颖,效率比较高,裁剪速度比较快。文献[12]的方法新颖,裁剪速度很快,其新颖之处在于:将错切变换(一种仿射变换)应用于待裁剪线段和裁剪窗口,使待裁剪线段平行于 x 轴,然后进行 y 坐标检测,确定窗口与直线有交的边,再解方程组求出交点。该算法计算复杂度为 $O(n)$。虽然该文献方法新颖,但仍然要解方程才能求出交点。本文将提出一种新方法,不需要解方程,只需做矩阵乘法,即可得到任意多边形窗口裁剪直线的有效交点。

2 预备定理

以下定理在文献[13]已经提及或已经部分证明,但为了叙述连贯,特将它们列出如下:

定理1 一条线段,至多经过一次仿射变换,就可以变成一条水平或者竖直的线段。

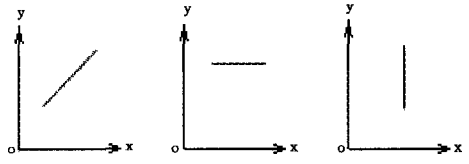


图1 一条线段经过一次仿射变换后变成一条水平或竖直的线段

证明:设一条线段为 $(x_0,y_0),(x_1,y_1)$,

(1)如果该线段的斜率为零或者不存在,这时定理已获得证明,否则我们设该线段的斜率为 k,然后

(2)取仿射变换

$x'=x$

$y'=y-kx$

将该仿射变换应用于上述线段,得到相应的线段:

$(x_0,-k*x_0+y_0),(x_1,-k*x_1+y_1)$

今将这条线段的两个端点作比较:

① $x_1-x_0\neq 0$;

②$(-k*x_0+y_0)-(-k*x_1+y_1)$

$=y_0-y_1+k*x_1-k*x_0$

$=y_0-y_1+k(x_1-x_0)$

到稿日期:2012-12-07 返修日期:2013-03-28

黄文钧(1958—),男,硕士,副教授,主要研究方向为真实感图形生成、图像分割,E-mail:hwjart@126.com。

$=y_0-y_1+(x_1-x_0)*(y_1-y_0)/(x_1-x_0)$

$=0$

故这条线段垂直于 y 轴，是一条水平线。

(3)取仿射变换

$y'=y$

$x'=x-(y/k)$

将该仿射变换应用于上述线段，得到相应的线段：

$(-(y_0/k)+x_0,y_0),(-(y_1/k)+x_1,y_1)$

今将这线段的两个端点作比较：

①$y_1-y_0\neq 0$；

②$(-(y_0/k)+x_0)-(-(y_1)/k+x_1)$

$=y_1/k-y_0/k+(x_0-x_1)$

$=(y_1-y_0)/k+(x_0-x_1)$

$=(y_1-y_0)(x_1-x_0)/(y_1-y_0)+(x_0-x_1)$

$=(x_1-x_0)+(x_0-x_1)$

$=0$

故这条线段垂直于 x 轴，是一条竖直线。

定理 2 平面上任意两条不重合直线，至多需要两次仿射变换，即可变成互相平行(两者都是水平线或者竖直线)或者互相垂直(一条水平线，另一条竖直线)。

证明：

①若两直线都是竖直线或者水平线，这时不需要任何变换。

②若其中一条是水平线，另一条是竖直线，这时也不需要任何变换。

③若其中一条是竖直线，另一条既不是竖直线，也不是水平线，其斜率 k 存在且 $k\neq 0$，这时作一次 y 轴方向的仿射变换：

$x'=x$

$y'=y-kx$

根据定理 1，后者变成水平线，而前者仍然是竖直线。

④若其中一条是水平线，另一条既不是竖直线，也不是水平线，其斜率 k 存在且 $k\neq 0$，这时作一次 x 轴方向的仿射变换：

$y'=y$

$x'=x-(y/k)$

根据定理 1，后者变成竖直线，而前者仍然是水平线。

⑤两条直线都不是水平线，也不是竖直线。这时先作 y (x)轴方向的仿射变换，使得其中一条线段变成水平线(竖直线)，这时变成情形②、④或者①。

若变成②或者①，则不需要任何变换；若变成④，则再作一次 x(y)轴方向的仿射变换，另一条线段则变成竖直线(水平线)。

定理 3 直线 $y=y_0$ 和 $x=x_0$ 的交点坐标为 (x_0,y_0)。

3 算法的基本思想

3.1 待裁剪线段 L 的斜率 $|k|\geq 1$ 的情形(见图 2(a))

(1)将待裁剪线段 L 和多边形窗口 W 平移(实施第一次仿射变换)，使 L 的一个端点和坐标系原点重合；

(2)对 L 和 W 实施 x 方向的错切变换(实施第二次仿射变换)，使待裁剪线段 L 与 y 轴重合；

(3)将那些和 y 轴有交的多边形 W 各边做 y 方向的错切变换(实施第三次仿射变换)，使它们和 y 轴垂直；直接写出它们和 y 轴的交点；

(4)将这些交点(这些交点在 y 轴上，对于第三次仿射变换是“不动点”)作第二次仿射变换和第一次仿射变换的反变换，即得到原窗口 W 裁剪原直线 L 的交点。

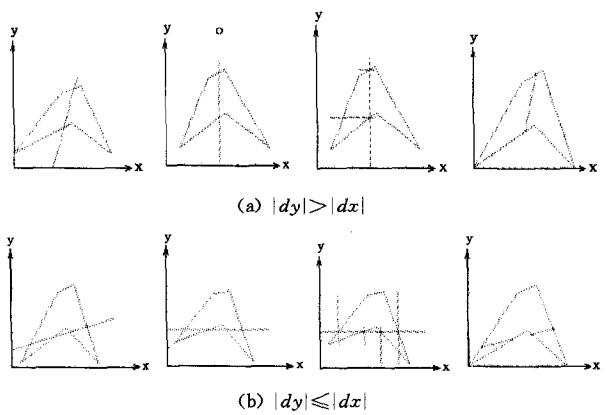
3.2 待裁剪线段 L 的斜率 $|k|<1$ 的情形(见图 2(b))

(1)将待裁剪线段 L 和多边形窗口 W 平移(实施第一次仿射变换)，使 L 的一个端点和坐标系原点重合；

(2)对 L 和 W 实施 y 方向的错切变换(实施第二次仿射变换)，使待裁剪线段 L 与 x 轴重合；

(3)将那些和 x 轴有交的多边形 W 各边做 x 方向的错切变换(实施第三次仿射变换)，使它们和 x 轴垂直；直接写出它们和 x 轴的交点；

(4)将这些交点(这些交点在 x 轴上，对于第三次仿射变换是“不动点”)作第二次仿射变换和第一次仿射变换的反变换，即得到原窗口 W 裁剪原待裁剪直线 L 的交点。



(a) $|dy|>|dx|$

(b) $|dy|\leq|dx|$

图 2

以上的每一步操作，都是实施仿射变换，而仿射变换可用矩阵表示。引进齐次坐标之后，以上的连续变换就可通过矩阵连乘实现。

4 算法步骤(用 MATLAB 语言书写)

1)构造多边形窗口矩阵

(1)随机生成任意 $n(n\geq 3)$ 边形窗口，将各边以规范齐次坐标形式存入矩阵 $P(2n,3n)$ 中：

```
定义矩阵  P=zeros(2n,3n);
for i=1:n/*第 n+1 个顶点和第一点重合*/
  j=2*(i-1)+1;
  jj=3*(i-1)+1;
    P(j,jj)=x_i;
    P(j,jj+1)=y_i;
    P(j,jj+2)=1;
  P(j+1,jj)=x_ii;
  P(j+1,jj+1)=y_ii;
  P(j+1,jj+2)=1;
end /*(x_i,y_i),(x_ii,y_ii)为第 i 边两个端点坐标*/
```

(2)构造矩阵 $kk(1,n)$ 和 $kk\_(1,n)$，分别依次存储窗口 $P(2n,3n)$ 的各边斜率和斜率的倒数，若斜率或斜率的倒数不存在，用 0 表示。

2)构造两个矩阵，用于取交点坐标

定义单位矩阵 $Y_0=eye(3n)$；然后

```
for i=1:n
  Y0(3i-1,3i-1)=0;
end
```

定义单位矩阵 $X_0=eye(3n)$;然后

```
for i=1:n
  X0(3i-2,3i-2)=0;
end
```

3)定义待裁剪线段的矩阵

随机生成 $m(m\geqslant 1)$条待裁剪线段,存入矩阵 $L(2,2m)$中。矩阵第一行 $L(1,2m)$存放线段端点 x 坐标,第二行 $L(2,2m)$存放线段端点 y 坐标,每一条线段占两列。

例如第一条线段(x_1,y_1)和(x_2,y_2)在矩阵中表示为:

$L(1,1)=x_1$;$L(1,2)=x_2$;

$L(2,1)=y_1$;$L(2,2)=y_2$。

4)构造活动矩阵及实施矩阵乘法

(0)构造一个平移矩阵 Txy,平移量为第 $j(1\leqslant j\leqslant m)$条线段的一个端点坐标;

从矩阵 $L(2,2m)$中取出第 j 条线段($j\geqslant 1$ &&$j\leqslant m$),设其端点为(x_1,y_1)和(x_2,y_2);

定义 $dx=x_2-x_1$;$dy=y_2-y_1$;

如果 $dx=0$ && $dy=0$,重新在矩阵 L 中取另外一条线段;

定义一个单位矩阵 $Txy=eye(3n,3n)$;

```
for i=1:n
  Txy(3i,3i-2)=-x1;
  Txy(3i,3i-1)=-y1;
  Txy(3i,3i)=1;
end
if abs(dx)≥abs(dy)
  k=dy/dx;
```

(1)构造待裁剪线段沿 y 方向的错切变换矩阵,以第 j 条直线的斜率 k 为矩阵元素:

定义单位矩阵 $Lafy=eye(3n)$;然后

```
for i=1:3:3(n-1)
  Lafy(i,i+1)=-k;
end
```

相应的反变换矩阵为 $Lafy\_=eye(3n)$,其构造方法和 $Lafy=eye(3n)$的相似,用 k 换$-k$ 即可:

```
for i=1:3:3(n-1)
  Lafy_(i,i+1)=k;
end
```

(2)构造裁剪窗口沿 x 方向的错切变换矩阵,以窗口 W 各边斜率 $kk(jj)(jj=1,2,3,\cdots,n)$和第 j 条线段斜率 k 为矩阵元素:

定义单位矩阵 $Wafx=eye(3n)$;然后做如下操作:

```
for i=1:3:3(n-1)
    jj=(i-1)/3+1;
  if 原窗口 W 的第 jj 边垂直于 x 轴|| kk(jj)==k
    Wafx(i+1,i)=0;
  else
    Wafx(i+1,i)=1/(k-kk(jj));
  end
end
```

(3)作矩阵连乘

```
if k==0
    mid=P*Txy *Wafx;
elseif k~=0
    mid=P*Txy*Lafy*Wafx;
end
```

到此,待裁剪线段已和 x 轴重合,并且窗口各边(除了和待裁剪线段平行的之外)已和 x 轴垂直。

(4)考察矩阵 mid,标注那些符合下列条件之一的边:

A. 两端点 y 坐标异号,则标注该边;/*表明该边和 x 轴有交*/

B. 两端点 y 坐标为 0,并且以这条边为邻边的两边 y 坐标异号,则标注其中的一条边;/*表明该边与 x 轴重合,两个交点只算一个*/

C. 两边的行序号连续,两边各有一个端点的坐标分量对等并且 y 坐标为 0,另两个端点 y 坐标异号,则标注其中的一条边;/*表明 x 轴过该顶点,并且该顶点所属的两边分在 x 轴两侧,这时两个交点只算一个*/

/*有多种标注方法,最简单的就是将第三维坐标改为-1*/

将标注过的矩阵仍然记为 mid。

(5)再做一次矩阵连乘(逆变换)

$LW=mid*Y_0*Lafy\_*Txy$;

此时矩阵 LW 中带有标注符号的点即为原窗口 W 和原待裁剪直线的交点,逐点取出,组成一个点集,参照待裁剪线段两端点 x 坐标次序,按 x 坐标排序,然后两两配对,与原待裁剪线段的 x 区间求交集。将所有交集连同相应的 y 坐标一起作并,其并集即为原窗口裁剪原待裁剪线段的结果。

```
elseif abs(dx)<abs(dy)
        k_=dx/dy;
```

(1)构造一个待裁剪线段沿 x 方向的错切变换矩阵,以第 j 条直线斜率的倒数 $k_$为矩阵元素:

定义单位矩阵 $Lafx=eye(3n)$;然后做如下操作:

```
for i=1:3:3(n-1)
  Lafx(i+1,i)=-k_;
end
```

相应的反变换矩阵为 $Lafx\_=eye(3n)$,其构造方法和 $Lafx=eye(3n)$的相似,将$-k_$换成 $k_$即可:

```
for i=1:3:3(n-1)
  Lafx_(i+1,i)=k_;
end
```

(2)构造裁剪窗口沿 y 方向错切变换矩阵,以窗口各边的斜率倒数 $kk\_(jj)$ $(jj=1,2,3,\cdots,n)$和第 j 条线段斜率倒数 $k_$为矩阵元素:

定义单位矩阵 $Wafy=eye(3n)$;然后作如下操作:

```
for i=1:3:3(n-1)
    jj=(i-1)/3+1;
  if 原窗口 W 的第 jj 边垂直于 y 轴|| kk_(jj)==k_
    Wafy(i,i+1)=0;
  else
    Wafy(i,i+1)=1/(k_-kk_(jj));
  end
end
```

(3)作矩阵乘法

```
  if  k_==0
      mid=P*Txy *Wafy;
    elseif k_~=0
```

mid=P * Txy * Lafx * Wafy;

end

到此，待裁剪线段已和 y 轴重合，并且窗口各边(除了和待裁剪线段平行的之外)已和 y 轴垂直。

(4)考察矩阵 mid，标注那些符合下列条件之一的边

A. 两端点 x 坐标异号，则标注该边；/ * 表明该边和 y 轴有交 * /

B. 两端点 x 坐标为 0，并且以这条边为邻边的两边 x 坐标异号，则标注其中的一条边；/ * 表明该边与 y 轴重合，两个交点只算一个 * /

C. 两边的行序号连续，两边各有一个端点的坐标分量对等，并且 x 坐标为 0，另两个端点 x 坐标异号，则标注其中的一条边；/ * 表明 y 轴过该顶点，并且该顶点所属的两边分在 y 轴两侧，这时两个交点只算一个 * /

将标注过的矩阵仍然记为 mid。

(5)再做一次矩阵连乘(逆变换)

$LW=mid * X_0 * Lafx\_ * Txy$;

此时矩阵 LW 中带有标注符号的点即为原窗口 W 和原待裁剪直线的交点，逐点取出，组成一个点集，参照待裁剪线段两端点 y 坐标次序，按 y 坐标排序，然后两两配对，与原待裁剪线段的 y 区间求交集。将所有交集连同相应的 x 坐标一起作并，其并集即为原窗口裁剪原待裁剪线段的结果。

end

5 算法分析

上述算法将待裁剪线段分为 $|dx|\geqslant|dy|$ 和 $|dx|<|dy|$ 两种情形来处理，目的是为了避免斜率不存在的讨论。

下面我们对 $|dx|\geqslant|dy|$ 的情形作讨论，其讨论过程和结论可简单类比到 $|dx|<|dy|$ 的情形。

算法中的第一次矩阵乘法：

$mid=P * Txy * Lafy * Wafx$

完成三步操作：首先将待裁剪线段平移到坐标原点，同时将多边形窗口 W 也一起平移，对应的矩阵乘法为：

$P * Txy$

接着将待裁剪线段变换，使其与 x 轴重合；对窗口 W 也做同样变换，对应的矩阵乘法为：

$(P * Txy) * Lafy$

若待裁剪线段斜率为 0，这步可跳过；接下来变换窗口各边，使它们(斜率与待裁剪线段相等的边除外，它们不受这一步变换影响)都垂直于 x 轴，对应的矩阵乘法为：

$mid=((P * Txy) * Lafy) * Wafx$

$$=\begin{bmatrix}
\ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & xa_1 & ya_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & xa_2 & ya_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & xb_1 & yb_1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & xb_2 & yb_2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & xc_1 & yc_1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & xc_2 & yc_2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ddots
\end{bmatrix}$$

到这一步，mid 中每一条边，除了斜率与待裁剪线段相等的边以外，都垂直于 x 轴，两端点的 x 坐标相等。根据端点的 y 坐标符号就可判断待裁剪直线和该边是否有交。图 3 是多边形窗口裁剪线段的典型例子：线段交多边形顶点的两类情形、线段和窗口的边相交、线段与窗口的边重合。这 3 种情形都可以从矩阵 mid 的数据反映出来。表 1－表 3 是矩阵 mid 的数据，记录多边形各边经过矩阵连乘之后的数据。

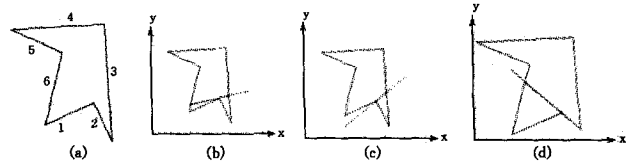


图 3 待裁剪线段过多边形窗口顶点及边的情形

表 1 对应图 3(b)的 mid 矩阵

x 坐标	y 坐标		多边形各边的序号	x 坐标	y 坐标		多边形各边的序号
80.0000	−20.0000	1.0000	1	829.4737	135.0000	1.0000	4
80.0000	0	1.0000		829.4737	163.5000	1.0000	
80.0000	0	1.0000	2	171.1475	163.5000	1.0000	5
80.0000	−77.5000	1.0000		171.1475	102.5000	1.0000	
106.3529	−77.5000	1.0000	3	4.8980	102.5000	1.0000	6
106.3529	135.0000	−1.0000		4.8980	−20.0000	−1.0000	

表 2 对应图 3(c)的 mid 矩阵

x 坐标	y 坐标		多边形各边的序号	x 坐标	y 坐标		多边形各边的序号
80.0000	30.0000	1.0000	1	251.2224	122.5000	1.0000	4
80.0000	0	−1.0000		251.2224	247.2500	1.0000	
80.0000	0	1.0000	2	128.9868	247.2500	1.0000	5
80.0000	−96.2500	1.0000		128.9868	133.7500	1.0000	
105.6000	−96.2500	1.0000	3	−8.6747	133.7500	1.0000	6
105.6000	122.5000	−1.0000		−8.6747	30.0000	1.0000	

表 3 对应图 3(d)的 mid 矩阵

x 坐标	y 坐标		多边形各边的序号	x 坐标	y 坐标		多边形各边的序号
80.0000	−120.0000	1.0000	1	−50.2439	160.0000	1.0000	4
80.0000	0	−1.0000		−50.2439	−4.0000	−1.0000	
80.0000	0	1.0000	2	−46.3636	−4.0000	1.0000	5
110.0000	0	1.0000		−46.3636	40.0000	−1.0000	
110.0000	0	1.0000	3	22.5000	40.0000	1.0000	6
110.0000	160.0000	1.0000		22.5000	−120.0000	−1.0000	

我们来看表 1 的数据，对应图 3(b)：①第 1 边和第 2 边端点的 x 坐标相等，说明这两边同在一条垂直于 x 轴的直线

上；②各有一个端点 y 坐标为 0，说明顶点在待裁剪直线上；③另两个端点 y 坐标同号，说明这两边在直线的同侧。对于这类“直线过顶点”情形，不计交点，不作任何标记。第 3 和第 6 边，其 y 坐标异号，说明待裁剪直线交这两边，在这两边各自的一个端点上作标记，记第三维坐标为 -1。

再来看表 2 的数据，对应图 3(c)：第 1 边和第 2 边端点的 x 坐标相等、各有一个端点 y 坐标为 0、另两个端点 y 坐标异号，说明直线过多边形的顶点，并且这两边在待裁剪直线的两侧。对于这类“直线过顶点”情形，计一个交点，在其中一边的一个端点第三维坐标记为 -1。

最后来看表 3 的数据，对应图 3(d)：①第 2 边两端点的 x 坐标不等，y 坐标为 0，说明该边在待裁剪直线上；②和该边相邻的两边，即第 1 边和第 3 边，它们各有一个端点 y 坐标为 0，另两个端点 y 坐标异号。对于这类“直线和边重合”情形，计一个交点，记一邻边的一个端点第三维坐标为 -1。第 4、5、6 三边，各边的两端点 y 坐标异号，说明它们和待裁剪直线有交，记各自的一个端点第三维坐标为 -1。

在矩阵 mid 中，待裁剪线段已重合于 x 轴，和 x 轴有交的多边形各边已垂直于 x 轴，本来可以直接写出多边形和待裁剪线段的交点，但是我们却先写出多边形和待裁剪直线的交点，为什么？因为我们不仅要得到交点，还要得到交点两两配对而形成的区间，而“交点两两配对”需要待裁剪线段所属直线和多边形的交点全体参与。

当矩阵 mid 完成了标注后，取多边形各边在 x 轴上的交点，对应矩阵乘法为：

$mid * Y_0$

然后作反第一次仿射变换，对应矩阵乘法：

$(mid * Y_0) * Lafy\_$

最后做反平移：

$LW=((mid * Y_0) * Lafy\_) * Txy$

矩阵 LW 中对应各边端点的数据，凡第三维坐标为 -1 的，皆表示该边和待裁剪直线的交点。用减法仅将交点的 x 坐标和待裁剪线段端点的 x 坐标作比较，即可判定该交点是否在待裁剪线段上。

6 算法优化

上述的算法步骤要进行多个矩阵相乘，计算量大。实际上，上述算法可分为两步：第一步，变换；第二步，逆变换。每一步对应一个矩阵，只需作一次矩阵乘法即可。下面讨论相应矩阵的构造方法。

1) $|dx|\geqslant|dy|$ 的情形

设多边形窗口的顶点集为 $\{(x_i,y_i)\mid i=1,2,\cdots n\}$，边的斜率集为 $\{kk(i)\mid i=1,2,\cdots,n\}$；待裁剪线段斜率为 k；平移量为 $\{-tx,-ty\}$；记 $ki=1/(kk(i)-k)$，$i=1,2,\cdots,n$；

(1)第一步

$mid=P * Txy * Lafy * Wafx$

$$=\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & 0 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -tx & -ty & 1 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -tx & -ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} *$$

$$\begin{bmatrix} 1 & -k & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -k & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -k1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -k2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

$$=\begin{bmatrix} x_1-tx & y_1-ty & 1 & 0 & 0 & 0 & 0 \\ x_2-tx & y_2-ty & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2-tx & y_2-ty & 1 & 0 \\ 0 & 0 & 0 & x_3-tx & y_3-ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} *$$

$$\begin{bmatrix} 1+k*k1 & -k & 0 & 0 & 0 & 0 & 0 \\ -k1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1+k*k2 & -k & 0 & 0 \\ 0 & 0 & 0 & -k2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

$=$

$$\begin{bmatrix} (1+k,k1)(x_1-tx)-k1(y_1-ty) & -k(x_1-tx)+y_1-ty & 1 \\ (1+k,k1)(x_2-tx)-k1(y_2-ty) & -k(x_2-tx)+y_2-ty & 1 \\ 0 & 0 & \ddots \end{bmatrix}$$

今将该矩阵(乘积的结果)的结构作如下改动：每边仅保留小序号端点的信息；每边占矩阵的前三列；于是该矩阵变成如下形式(仍然记为 mid)：

$mid=$

$$=\begin{bmatrix} (1+k,k1)(x_1-tx)-k1(y_1-ty) & -k(x_1-tx)+y_1-ty & 1 \\ (1+k,k1)(x_2-tx)-k1(y_2-ty) & -k(x_2-tx)+y_2-ty & 1 \\ (1+k,k3)(x_3-tx)-k3(y_3-ty) & -k(x_3-tx)+y_3-ty & 1 \\ \cdots & \cdots & \cdots \end{bmatrix}$$

这个矩阵维数为 $n*3$，其中 n 为多边形顶点数目。

(2)第二步

$Y_0LT=Y_0 * Lafy\_ * Txy$

$$=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} 1 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & k & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} *$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -tx & -ty & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -tx & -ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

$$= \begin{bmatrix} 1 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & k & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -tx & -ty & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -tx & -ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

$$= \begin{bmatrix} 1 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -tx & -ty & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & k & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -tx & -ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

该矩阵由 n 个相同的分块矩阵组成，今只取其中一个（仍然用记号 Y_0LT 标识）：

$$Y_0LT = \begin{bmatrix} 1 & k & 0 \\ 0 & 0 & 0 \\ -tx & -ty & 1 \end{bmatrix}$$

这个矩阵维数为 3 * 3。

所以，只需按照新规则构造两个矩阵 $mid(n,3)$ 和 $Y_0LT(3,3)$，当 mid 完成标注后，仅做一次矩阵乘法：

$LW = mid * Y_0LT$

$$= \begin{bmatrix} (1+k,k1)(x_1-tx)-k1(y_1-ty)-k(x_1-tx)+y_1-ty & 1 \\ (1+k,k1)(x_2-tx)-k2(y_2-ty)-k(x_2-tx)+y_2-ty & 1 \\ (1+k,k3)(x_3-tx)-k3(y_3-ty)-k(x_3-tx)+y_3-ty & 1 \\ \cdots & \cdots \end{bmatrix} *$$

$$\begin{bmatrix} 1 & k & 0 \\ 0 & 0 & 0 \\ -tx & -ty & 1 \end{bmatrix}$$

即可得到窗口 W 和直线 L 的交点坐标。

在矩阵 mid 中仅保留边的一个端点信息，也就是顶点信息，另一个端点信息含在相邻的顶点坐标里。例如顶点 i 和 $i+1$ 是相邻两个顶点，它们分别表示第 i 条边和第 $i+1$ 条边的端点信息；第 i 条边的另一个端点的 y 坐标和第 $i+1$ 顶点的 y 坐标相等。利用这一特点，仿照上述的矩阵 mid 标注方法，稍作改动，即可得到适合算法优化的矩阵 mid 标注方法：

A. 两相邻顶点 y 坐标异号，则标注小序号边；/ * 边和 x 轴有交 * /

B. 两相邻顶点 y 坐标为 0，并且分别以这两顶点为邻的另外两顶点，它们的 y 坐标异号，则标注其中的一个小序号顶点；/ * 边和 x 轴重合 * /

C. 一个顶点的 y 坐标为 0，和它相邻的两个顶点 y 坐标异号，则标注其中的小序号边；/ * 多边形顶点在 x 轴上，即交点在顶点上 * /

2) $|dy|>|dx|$ 的情形

设多边形窗口的顶点集为 $\{(x_i,y_i)|i=1,2,\cdots,n\}$，边的斜率倒数集为 $\{kk\_(i)|i=1,2,\cdots,n\}$；待裁剪线段斜率倒数为 $k\_$；平移量为 $\{-tx,-ty\}$；记 $ki\_=1/(kk\_(i)-k\_)$，$i=1,2,\cdots,n$；

(1)第一步

$mid = P * Txy * Lafx * Wafy$

$$= \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & 0 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -tx & -ty & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -tx & -ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} *$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -k\_ & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -k\_ & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} 1 & -k1\_ & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -k2\_ & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

$$= \begin{bmatrix} x_1-tx & y_1-ty & 1 & 0 & 0 & 0 & 0 \\ x_2-tx & y_2-ty & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2-tx & y_2-ty & 1 & 0 \\ 0 & 0 & 0 & x_3-tx & y_3-ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} *$$

$$\begin{bmatrix} 1 & -k1\_ & 0 & 0 & 0 & 0 & 0 \\ -k\_ & 1+k\_*k1\_ & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -k\_ & 1+k\_*k2\_ & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

$$\begin{bmatrix} x_1-tx-k\_(y_1-ty) & (1+k\_*k\_1)(y_1-ty)-k\_1-(x_1-tx) & 1 \\ x_2-tx-k\_(y_2-ty) & (1+k\_*k\_1)(y_2-ty)-k\_1-(x_2-tx) & 1 \\ 0 & 0 & \ddots \end{bmatrix}$$

今将该矩阵（乘积的结果）的结构作如下改动：每边仅保留小序号端点的信息；每边占矩阵的前三列；于是该矩阵变成如下形式（仍然记为 mid）：

$mid =$

$$\begin{bmatrix} x_1-tx-k\_(y_1-ty) & (1+k\_*k1\_)(y_1-ty)-k1\_(x_1-tx) & 1 \\ x_2-tx-k\_(y_2-ty) & (1+k\_*k2\_)(y_2-ty)-k2\_(x_2-tx) & 1 \\ x_3-tx-k\_(y_3-ty) & (1+k\_*k3\_)(y_3-ty)-k3\_(x_3-tx) & 1 \\ \cdots & \cdots & \cdots \end{bmatrix}$$

这个矩阵维数为 $n * 3$，其中 n 为多边形顶点数目。

(2)第二步

$X_0LT = X_0 * Lafx\_ * Txy$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ k\_ & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & k\_ & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} *$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -tx & -ty & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -tx & -ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ k\_ & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k\_ & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -tx & -ty & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -tx & -ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ k\_ & 1 & 0 & 0 & 0 & 0 & 0 \\ -tx & -ty & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k\_ & 1 & 0 & 0 \\ 0 & 0 & 0 & -tx & -ty & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix}$$

该矩阵由 n 个相同的分块矩阵组成，今只取其中一个（仍然用记号 X_0LT 标识）：

$$X_0LT = \begin{bmatrix} 0 & 0 & 0 \\ k\_ & 1 & 0 \\ -tx & -ty & 1 \end{bmatrix}$$

所以，只需按照新规则构造两个矩阵 $mid(n,3)$ 和 $X_0LT(3,3)$，当 mid 完成标注后，只做一次矩阵乘法：

$LW = mid * X_0LT$

$$= \begin{bmatrix} x_1 - tx - k\_(y_1 - ty) & (1 + k\_ * k1\_)(y_1 - ty) - k1\_(x_1 - tx) & 1 \\ x_2 - tx - k\_(y_2 - ty) & (1 + k\_ * k2\_)(y_2 - ty) - k2\_(x_2 - tx) & 1 \\ x_3 - tx - k\_(y_3 - ty) & (1 + k\_ * k3\_)(y_3 - ty) - k3\_(x_3 - tx) & 1 \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$\begin{bmatrix} 0 & 0 & 0 \\ k\_ & 1 & 0 \\ -tx & -ty & 1 \end{bmatrix}$ 即可得到窗口 W 和直线 L 的交点坐标。

对矩阵 mid 的标注方法和情形 $|dx| \geqslant |dy|$ 的类似：

A. 两相邻顶点 x 坐标异号，则标注小序号边；/ * 边和 y 轴有交 * /

B. 两相邻顶点 x 坐标为 0，并且分别以这两顶点为邻的另外两顶点的 x 坐标异号，则标注其中的一个小序号顶点；/ * 边和 y 轴重合 * /

C. 一个顶点的 x 坐标为 0，和它相邻的两个顶点 x 坐标异号，则标注其中的小序号边；/ * 多边形顶点在 y 轴上，即交点在顶点上 * /

7 算法加速及复杂度分析

对于 $|dx| \geqslant |dy|$ 的情形，算法分析指出：矩阵乘法“$(P * Txy) * Lafy$”完成之后，待裁剪直线和 x 轴重合，再做一次矩阵乘法“$((P * Txy) * Lafy) * Wafx$”，多边形窗口各边（除了和 x 轴平行的之外）和 x 轴垂直。事实上，没有必要将多边形各边都参与后一次的矩阵乘法，只要那些和 x 轴有交的边参与变换就可以了。因此，算法应该提前标注那些和 x 轴有交的边，仅让那些已被标注记号的边参与后续的矩阵乘法即可。对于 $|dx| < |dy|$ 的情形也做类似处理。

下面是加速算法：

1) $|dx| \geqslant |dy|$ 的情形

设多边形窗口的顶点集为 $\{(x_i, y_i) | i = 1,2,\cdots,n\}$，边的斜率集为 $\{kk(i) | i = 1,2,\cdots,n\}$；待裁剪线段斜率为 k；平移量为 $\{-tx, -ty\}$；记 $ki = 1/(kk(i) - k)$，$i = 1,2,\cdots,n$；

(1)构造如下 $n * 3$ 矩阵

$$mid = \begin{bmatrix} (x_1 - tx) & -k(x_1 - tx) + y_1 - ty & 1 \\ (x_2 - tx) & -k(x_2 - tx) + y_2 - ty & 1 \\ (x_3 - tx) & -k(x_3 - tx) + y_3 - ty & 1 \\ \cdots & \cdots & \cdots \end{bmatrix}$$

(2)按照“算法优化”中的标注规则，将该标注的边（譬如第 i 边）作如下处理且存入新矩阵 $midd$ 中；

$midd(j,1) = mid(i,1) - ki * mid(i,2)$；

$midd(j,2) = mid(i,2)$；

$midd(j,3) = -1$；/ * $j \geqslant 1$，记录矩阵 $midd$ 的行数 * /

(3)作矩阵乘法

$$midd = midd * \begin{bmatrix} 1 & k & 0 \\ 0 & 0 & 0 \\ -tx & -ty & 1 \end{bmatrix}$$

矩阵 $midd$ 中的坐标即为原多边形窗口 W 裁剪原直线 L 的交点。

2) $|dy| > |dx|$ 的情形

设多边形窗口的顶点集为 $\{(x_i, y_i) | i = 1,2,\cdots,n\}$，边的斜率倒数集合为 $\{kk\_(i) | i = 1,2,\cdots,n\}$；待裁剪线段斜率倒数为 $k\_$；平移量为 $\{-tx, -ty\}$；记 $ki\_ = 1/(kk\_(i) - k\_)$，$i = 1,2,\cdots,n$；

(1)构造如下 $n * 3$ 矩阵

$$mid = \begin{bmatrix} x_1 - tx - k\_(y_1 - ty) & (y_1 - ty) & 1 \\ x_2 - tx - k\_(y_2 - ty) & (y_2 - ty) & 1 \\ x_3 - tx - k\_(y_3 - ty) & (y_3 - ty) & 1 \\ \cdots & \cdots & \cdots \end{bmatrix}$$

(2)按照“算法优化”中的标注规则，将该标注的边（譬如第 i 边）作如下处理且存入新矩阵 $midd$ 中：

$midd(j,1) = mid(i,1)$

$midd(j,2) = mid(i,2) - ki\_ * mid(i,1)$

$midd(j,3) = -1$

(3)作矩阵乘法

$$midd = midd * \begin{bmatrix} 1 & k & 0 \\ 0 & 0 & 0 \\ -tx & -ty & 1 \end{bmatrix}$$

矩阵 $midd$ 中的坐标即为原多边形窗口 W 裁剪原直线 L 的交点。

3)复杂度分析

在加速算法的 3 个步骤中，步骤(1)处理 n 多边形每一个顶点，时间复杂度为 $O(n)$；步骤(2)也是考察 n 个顶点，复杂度仍为 $O(n)$；步骤(3)的矩阵 $midd$，其最高维数为 $n * 3$，而和它相乘的矩阵其维数为 $3 * 3$，故时间复杂度也是 $O(n)$。因此，加速算法的时间复杂度为 $O(n)$。

8 实验和对比

MATLAB语言擅长矩阵计算，而本文算法以矩阵乘法表述，故我们用MATLAB语言编制程序实现本文算法，与文献[12]算法对比。实验设备是一台笔记本电脑：Intel Pentium、1.60GHz处理器、191 MHz，252MB内存。在本实验中，用随机函数randint生成任意多边形的顶点坐标和待裁剪线段的端点坐标。为了使多边形能够闭合、不出现自交现象，对随机函数中的取值范围做了必要设计。

将实验分为5组，分别裁剪1000/2条、5000/2条、10000/2条、20000/2条、30000/2条线段。对于上述的每一组线段，分别用6边形窗口、60边形窗口、100边形窗口、500边形窗口、1000边形窗口、2000边形窗口、3000边形窗口裁剪，用函数cputime函数计时两种算法的执行时间。由于该函数计时不是很稳定，我们将算法在条件不变的情况下连续执行4次，取4个时间数据平均值作为算法在这一特定条件下的执行时间，保留在表4－表10中。

表4 两种算法执行时间对比(6边形窗口)

线段条数	1000/2	5000/2	10000/2	20000/2	30000/2
本文算法	0.0250	0.1227	0.1467	0.2128	0.3555
文献[12]算法	0.0951	0.4331	0.9939	3.1020	3.9907
文献[12]/本文	3.8040	3.5297	6.7751	14.5771	11.2256

表5 两种算法执行时间对比(60边形窗口)

线段条数	1000/2	5000/2	10000/2	20000/2	30000/2
本文算法	0.2378	0.5007	0.9689	1.9278	2.8791
文献[12]算法	0.2454	0.8488	1.5297	3.5802	5.553
文献[12]/本文	1.0320	1.6952	1.5788	1.8571	1.9287

表6 两种算法执行时间对比(100边形窗口)

线段条数	1000/2	5000/2	10000/2	20000/2	30000/2
本文算法	0.1578	0.5758	1.2243	2.3534	3.5526
文献[12]算法	0.2028	0.8437	1.6299	3.3223	5.4303
文献[12]/本文	1.2852	1.4653	1.3313	1.4117	1.5285

表7 两种算法执行时间对比(500边形窗口)

线段条数	1000/2	5000/2	10000/2	20000/2	30000/2
本文算法	0.2504	0.686	1.4346	2.9317	4.3187
文献[12]算法	0.2454	0.9664	2.0079	4.0208	6.0738
文献[12]/本文	0.9800	1.4088	1.3996	1.3715	1.4064

表8 两种算法执行时间对比(1000边形窗口)

线段条数	1000/2	5000/2	10000/2	20000/2	30000/2
本文算法	0.2529	0.8963	1.8251	3.6352	5.3978
文献[12]算法	0.3355	1.3344	2.7389	5.3202	7.979
文献[12]/本文	1.3266	1.4888	1.5007	1.4635	1.4782

表9 两种算法执行时间对比(2000边形窗口)

线段条数	1000/2	5000/2	10000/2	20000/2	30000/2
本文算法	0.2779	1.2693	2.5587	5.0422	7.6185
文献[12]算法	0.4907	2.1807	4.1109	8.1492	12.2401
文献[12]/本文	1.7657	1.7180	1.6066	1.6162	1.6066

表10 两种算法执行时间对比(3000边形窗口)

线段条数	1000/2	5000/2	10000/2	20000/2	30000/2
本文算法	0.4206	1.8977	3.9807	7.8112	11.6868
文献[12]算法	0.7711	3.2822	6.627	13.1163	19.5081
文献[12]/本文	1.8333	1.7296	1.6648	1.6792	1.6692

每一个表含有5列数据，每一列数据分别记录该组线段的数目、两个算法的执行时间、两个执行时间的比值。以表4的最后一列数据为例：第一行的数据表示该组有15000(即30000/2)条线段将被6边形窗口裁剪，第二行数据0.3555表示本文算法利用这个6边形窗口裁剪这15000条线段所耗费的时间，第三行数据3.9907表示文献[12]算法利用这个6边形窗口裁剪这15000条线段所耗费的时间，最后一行的数据11.2256表示文献[12]算法耗时3.9907和本文算法耗时0.3555之比。

表中的数据表明，本文算法的平均执行速度是文献[12]的1.2倍以上。

在窗口边数很大(譬如2000或3000)的情况下，本文算法的执行速度也是比较高的：从表9和表10最后一行数据可以看出，本文算法的平均执行速度是文献[12]的1.6倍以上。

对于窗口边数少(譬如6边)而被裁剪线段数目大(譬如10000或15000)的情形，本文算法的执行效率更显著：表4最后一行数据随着被裁剪线段数目的增大而变化，当被裁剪线段数目增加到10000或15000时，本文算法的执行速度比文献[12]算法快11倍以上。

下面，我们考察多边形窗口边的数目对算法执行速度的影响。

将表4－表10的最后一行数据逐列取出，并按取出次序组成一个有序数集，譬如最后一列数据组成的有序集为：{11.2256,1.9287,1.5285,1.4064,1.4782,1.6066,1.6692}。按照这一规则，可以得到5个有序集，每一个集合对应一个线段集；每个集合含7个元素，每一个元素对应一个多边形窗口。譬如上述的有序集，对应的那个线段集含有30000/2条线段；此有序集以及另外的4个有序集，其中的第一个元素对应6边形窗口、第二个元素对应60边形窗口、第三个元素对应100边形窗口，如此类推。

现在将7个多边形窗口置入一个有序集：{6,60,100,500,1000,2000,3000}，将此集合分别与上述的5个有序集作对应，将对应元素组成数对并在直角坐标系上描点，得到5条曲线。用不同的符号表示不同曲线上的点，譬如对应那个线段集(含1000/2条线段)的那条曲线，我们用符号“Δ”表示此曲线上的点，见图4。

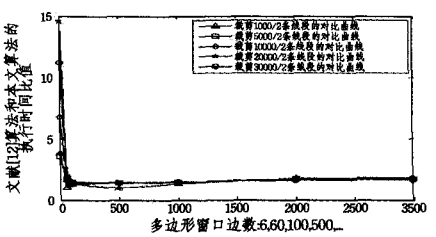


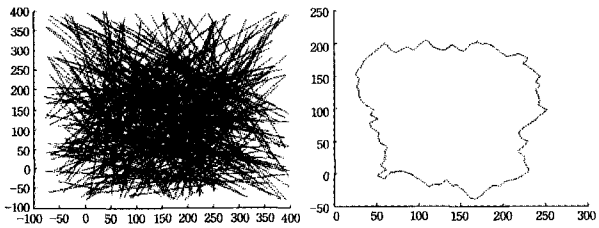
图4 算法执行时间比较

观察图4我们发现，窗口边数很小时，时间比值较大；当窗口边数为6时，时间比值达到3.5以上，甚至可达14.5。随着窗口边数的增加，时间比值随着下降，但最小值仍然大于1；随着窗口边数继续增加，时间比值缓慢上升；当窗口边数大于2000时，5条曲线趋于重合，曲线走势趋于平缓但仍有上升趋势，时间比值在1.6以上。

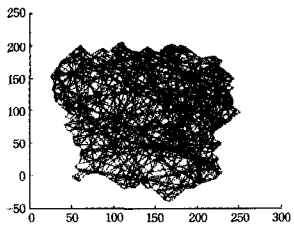
综合以上分析，我们得出结论：在裁剪窗口边数很少的情况下，受线段数目影响，本文算法执行速度比文献[12]算法快3.5～14.5倍；在裁剪窗口边数很大的情况下，几乎不受线段数目影响，本文算法执行速度比文献[12]算法快1.6倍以上；在一般的情况下，仍受线段数目影响，本文算法执行速度比文

献[12]算法快 1.03～1.85 倍。

本实验的效果如图 5 所示。



(a) 1000/2 条线段　　(b) 100 边多边形



(c) 裁剪结果

图 5　算法实验的一个例子

结束语　本文应用矩阵乘法实现连续变换，获得多边形窗口裁剪直线的交点，同时利用矩阵的规范格式快速获得交点的分类和计数，提高了效率。实验表明本文方法有效，并且算法执行速度得到了提高。

参 考 文 献

[1] Skala V. O(lgN) line clipping algorithm in E^2[J]. Computers & Graphics, 1994, 18(4): 517-527

[2] Skala V. A new approach to line and line segment clipping in Homogeneous Coordinates[J]. Visual computer, 2005, 21(11): 905-914

[3] 唐井林，张庆，孙惠学. 基于叉积法的凸多边形窗口裁剪算法[J]. 东北重型机械学院学报，1995，19(1)：23-25

[4] 孙燮华. 凸多边形窗口线裁剪的新算法[J]. 中国图像图形学报，2003，8(12A)：1475-1477

[5] 李伟青. 凸多边形窗口线裁剪的折半查找算法[J]. 计算机辅助设计与图形学学报，2005，17(5)：962-965

[6] 韩俊卿，葛永慧，张东升. 多边形窗口的矢量图形裁剪算法[J]. 太原理工大学学报，2005，36(2)：160-163

[7] 任洪海. 基于点区域分布的多边形窗口线裁剪算法[J]. 科学技术与工程，2009，9(16)：4833-4835

[8] 陆国栋，刑世海，彭群生. 基于顶点编码的多边形窗口线裁剪高效算法[J]. 计算机学报，2002，25(9)：987-993

[9] 李伟青. 基于扫描带的任意多边形窗口线裁剪算法[J]. 工程图学学报，2005(2)：35-40

[10] 孙春娟，王文成，李静，等. 基于凸片段分解的多边形窗口线裁剪算法[J]. 计算机辅助设计与图形学学报，2006，18(12)：1799-1805

[11] 李静，王文成，吴恩华. 基于凸剖分的多边形窗口线裁剪算法[J]. 计算机辅助设计与图形学学报，2007，19(4)：425-429

[12] Huang Y Q, Liu Y K. An algorithm for the clipping against a polygon based on shearing transformation[J]. Computer Graphics Forum, 2002, 21(4): 683-688

[13] Huang Wen-jun. The Line Clipping Algorithm Basing on Affine Transformation[J]. Intelligent Information Management, 2010, 2(6): 380-385

(上接第 282 页)

但易受光照等因素影响。Gabor 特征描述子和 HOG 特征描述子在识别率方面相差不大，但 Gabor 特征描述子特征提取速度慢，且计算复杂，全局 HOG 特征提取易受光照等因素的影响。综合以上因素，本文方法效率最高。

结束语　本文采用 HOG 特征描述子来提取人脸特征，并将全局 HOG 特征和局部关键部分 HOG 特征进行特征层融合形成最后分类特征，用随机森林分类器对其进行分类。在 FERET、CAS-PEAL-R1 和真实场景人脸库上的实验表明，本方法能够快速有效提取人脸特征，对姿势、光照等具有鲁棒性且有较高的人脸识别率。

参 考 文 献

[1] Perlibakas V. Measures for PCA-based Face Recognition [J]. Pattern Recognition Letters, 2004, 25(6): 711-724

[2] Xie Yong-lin. LDA and Its Application in Face Recognition [J]. Computer Engineering and Application, 2010, 46(19): 189-192

[3] Barlett M S, Movellan J R, Sejnowski T J. Face Recognition By Independent Component Analysis [J]. IEEE Transactions on Neural Networks, 2002, 13(6): 1450-1464

[4] Ahonen T, Hadid A, Pietikainen M. Face Description with Local Binary Patterns Application to Face Recognition [J]. IEEE Transactions on Pattern Analysis And Machine Intelligence, 2006, 28(12): 2037-2041

[5] 王庆军，张汝波. 基于 Log-Gabor 和正交等度规映射的人脸识别[J]. 计算机科学，2011，38(2)：274-276

[6] Cong Geng, Jiang Xu-dong. SIFT features for face recognition [C]// International Conference on Computer Science and Information Technology. Kiev, 2009: 598-602

[7] 江艳霞，王娟，等. 融合局部 Gabor 相位特征和全局本征脸的人脸识别算法[J]. 小型微型计算机系统，2012，33(9)：2091-2095

[8] Dalal N, Triggs B. Histograms of Oriented Gradients for Human Detection[C]// Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Los Alamitos, 2005: 886-893

[9] Déniz O, Bueno G, et al. Face recognition using histograms of oriented gradients [J]. Pattern Recognition Letters, 2011, 32 (12): 1598-1603

[10] 汪大任，刘慧玲，等. 人脸识别中 PCA，2DPCA 以及分块 PCA 的性能与比较[J]. 中国西部科技，2009，8(27)：14-16

[11] Yang Jing, Zhang D. Two-dimensional PCA: A new approach to appearance-based face representation and recognition [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004, 26(1): 131-137

[12] Tan X, Triggs B. Enhanced Local Texture Feature Sets for Face Recognition under Difficult Lighting Conditions [C] // Proceedings of the 2007 IEEE International Workshop on Analysis and Modeling of Faces and Gestures. LNCS 4778, 2007: 168-182

[13] 向征，谭恒良. 改进的 HOG 和 Gabor，LBP 性能比较[J]. 计算机辅助设计与图形学学报，2012，24(6)：787-792

[14] 王宪，张彦，等. 基于改进的 LBP 人脸识别算法[J]. 光电工程，2012，39(7)：109-114