An Intersecting Modification to the Bresenham Algorithm

Vaclav Skala*

Introduction

The solution of many engineering problems have as a result functions of two variables. that can be given either by an explicit function description, or by a table of the function values. The functions have been usually plotted with respect to visibility. The subprograms for plotting the functions of two variables were not so simple1,2 although visibility may be achieved by the relatively simple algorithm at the physical level of the drawing, if we assume raster graphics devices are used. The Bresenham algorithm for drawing line segments can be modified in order to enable the drawing of explicit functions of two variables with respect to the visibility.

Though the order of curve drawing is essential for the method used the algorithm has not been published before. Williamson2 solved the problem by fixing the position of the view-point, Watkins1 only pointed out that some rotation angles can cause wrong hidden-line elimination and Boutland's method3 can use only one angle.

An algorithm to ensure the right order of the curve's drawing is therefore presented here.

1. Problem Specification

Let us have an explicit function of two variables x and Y

$$z = f(x,y)$$

where:

$$x \in <ax,bx> \text{ and } y \in <ay,by>$$

and we want to display that function using a raster graphics display or plotter. For many scientific problems it is enough to show the behaviour of the function by drawing the function slices according to the x and y axes, e.g. curves

$$z = f(x,y_i) \quad i = 1,\ldots,n$$

where:

$$x \in <ax,bx> \text{ and } ay = y_1 < y_2 < \ldots < y_n = by$$

and curves:

$$z = f(x_j,y) \quad j = 1,\ldots,m$$

where:

$$y \in <ay,by> \text{ and } ax = x_1 < x_2 < \cdots < x_m = bx$$

The given function can be represented either by a function specification or by a table of the function values for the grid points in the x-y plane. If the function is complex it can be very difficult to imagine the function behaviour because some parts are in the reality invisible. The problem has been solved by Watkins,1 Williamson2 and Boutland3 very successfully. The principle of the solution is generally very simple. If we have drawn the first two slices parallel to the x axis we have produced two curves and the space between them is the strip of invisibility. Let us suppose that we draw the lines in the direction from foreground to background.

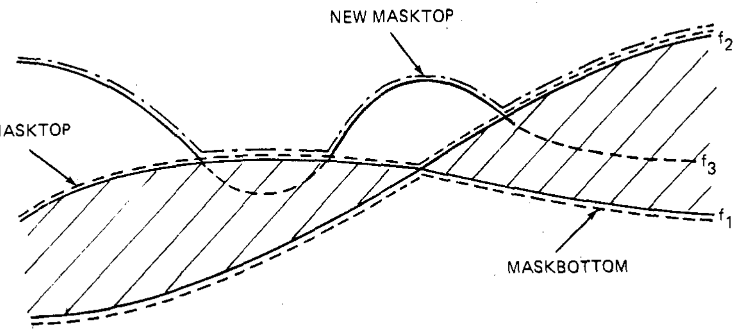

Figure 1.

Now if we want to draw the third curve it is obvious that those parts which are passing through the strip of invisibility are invisible and therefore ought not to be drawn, see figure 1.

*Department of Technical Cybernetics
Technical University
Nejedleho Sady 14
306 14 Plzen, Czechoslovakia

If we analyse the problem in detail we will realize that we need to represent the borders of the strip of invisibility. It can be done by the MASKTOP and MASKBOTTOM functions. The actual representation of the MASKTOP and MASKBOTTOM functions is described later, Now the problem of drawing curves with respect to the visibility becomes simple (see algorithm 1), because we will draw the next function slice if and only if the curve points are outside of the strip of invisibility.

The invisibility problem was solved by Watkins1 by introducing mask vectors for the representation of the MASKTOP and MASKBOTTOM bounds. Several problems had to be solved because all computation was done in the floating point representation:

- the first step is to decide if we have set up MASK[i] or MASK[i+1] if the coordinate x is between value i and i+1 e.g. $i<x<i+1$.
- the second problem is to set up the MASKTOP and MASKBOTTOM arrays for all points on the curve. This means that an interpolation procedure has to be employed, with some suitable interpolation step length.
- the third step is to handle the special case when the curve is parallel with the z axis, and the usual line segment slope computation can fail.

Algorithm 1.

```
MASKTOP := $-\infty$
MASKBOTTOM := $+\infty$
k := 1
WHILE k $\leq$ n DO BEGIN
   DRAW FUNCTION f(x, $y_k$)
       with respect to the strip
       of invisibilty for x $\in$ <ax, bx>
   MASKTOP := max {MASKTOP, f(x, $y_k$)}
   MASKBOTTOM := min {MASKBOTTOM, f(x, $y_k$)}
   k := k+1
END
```

2. Proposed Method

ln Watkins1 the functions MASKTOP and MASKBOTTOM are represented by vectors with values in floating point representation. we can imagine the whole process of hidden-line drawing as follows in figure 2.

Now we ask ourselves if there is any possibility of increasing the efficiency of the hidden-line solution. One possibility is to combine the complete Watkin's algorithm with the Bresenham algorithm directly at the physical level. Because we are dealing with the raster devices at the physical level we have got rid of all these above mentioned problems.

The solution of the hidden-line problem is now relatively very simple, because we have to change only the procedure DRAW-STEP that generates code for the physical movement, in order to take account of the strip of invisibility. Because DRAW-STEP draws only one step we have to check only if the next end-point in the raster is inside of the strip of invisibility or not. The structure of the proposed method is shown on figure 3. It is obvious that we need only integer representation for the MASKTOP and MASKBOTTOM masking arrays. The simplified solution is shown in algorithm 2.

It was found that the lines (on the physical level) which are parallel to y axis cause some problems with setting of the masks arrays, see figure 4. Suppose that we have defined the strip of invisibility and we want to draw the line segment x_1x_2. The problem is that if we want to draw the segment between the points 1 and 2 we have to change the strip of invisibility, so the future points 3 and 4 become inner points in the strip of invisibility; but that is not true. Therefore in the complete algorithm the content of the mask's arrays is changed only if dx<>0. The whole algorithm can be found in an earlier paper? where the clipping is realized too.

Watkin's original method and proposed solution have one common previously unpublished problem. Because of rotation sometimes the foreground and background can be altered and the order in which the

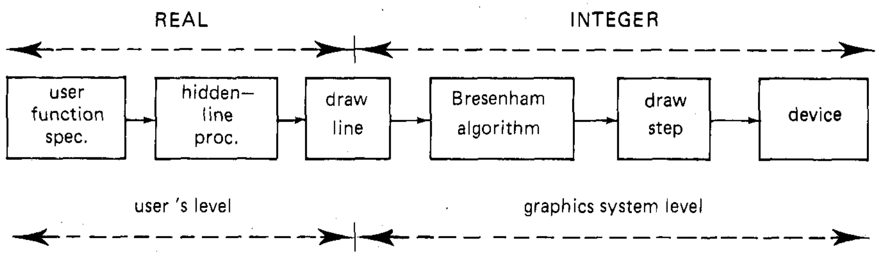

Figure 2.

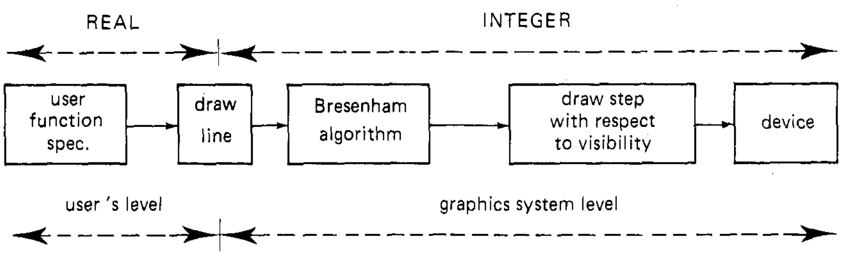

Figure 3.

curves are drawn cause a violation of the masking criteria. The second problem is how to select the scales for scaling in order not to lose any part of the picture and use the full screen area. The first problem seems to be more complicated and it is more fundamental. The proposed solution is presented below. The second problem can be solved easily by finding maximal and minimal values for the screen coordinates.

Algorithm 2.

```
{ GLOBAL VARIABLES }
VAR x0,y0: REAL;
    masktop,maskbottom:
        ARRAY [0..1024] OF INTEGER;

PROCEDURE draw (dx,dy: INTEGER );
VAR flag5: BOOLEAN;
BEGIN x0 := x0+dx; y0 := y0+dy;
  flag5 := FALSE;
  IF masktop[x0] <= y0 THEN
  BEGIN flag5 := TRUE;
      masktop[x0] := y0;
  END;
  IF maskbottom[x0] >= y0 THEN
  BEGIN flag5 := TRUE;
      maskbottom[x0] := y0;
  END;
  IF flag5 THEN
      physline(dx,dy)
  ELSE physmove(dx,dy)
END;

PROCEDURE bresenham (u,v: INTEGER);
VAR j,d,a,b: INTEGER;
BEGIN a := v+v; d := a-u; b := a-u-u;
  FOR j := 1 TO u DO
      IF d<0 THEN
      BEGIN draw(1,0); d:=d+a; END
      ELSE BEGIN draw(1,1); d:=d+b; END
END;
```

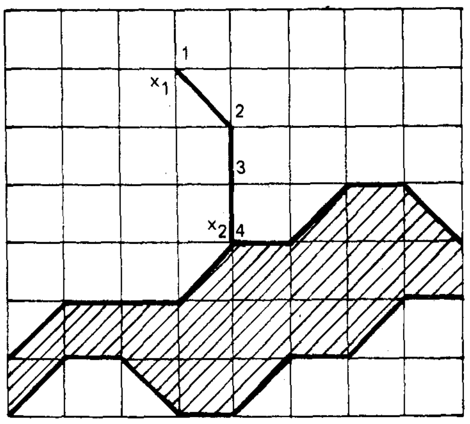

Figure 4.

3. Design of the Drawing Order

If we rotate the function or have a look at it from different points, we have to keep the basic drawing rules. First we have to draw the function slices that are nearer. Watkins1 pointed out this problem, but the problem solution has not been previously published and many users have real difficulties to ensure correct solutions. Therefore when the function is rotated many pictures are drawn incorrectly. To find the solution assume that the points:

$$X_1 = (ax, ay, 0) \quad X_3 = (bx, by, 0)$$
$$X_2 = (bx, ay, 0) \quad X_4 = (ax, by, 0)$$

are the corner-points of the grid in the x-y plane. We want to know the order of the drawing slices.

Assumes that the points:

$$X'_1 = T(X_1) \quad X'_3 = T(X_3)$$
$$X'_2 = T(X_2) \quad X'_4 = T(X_4)$$

are the corner points of the grid after the rotation transformation. Now we have to pick up two margins from which we will start to draw the picture. We have to select the end-points of these margins that has the smallest z' coordinates. We will mark that point by the

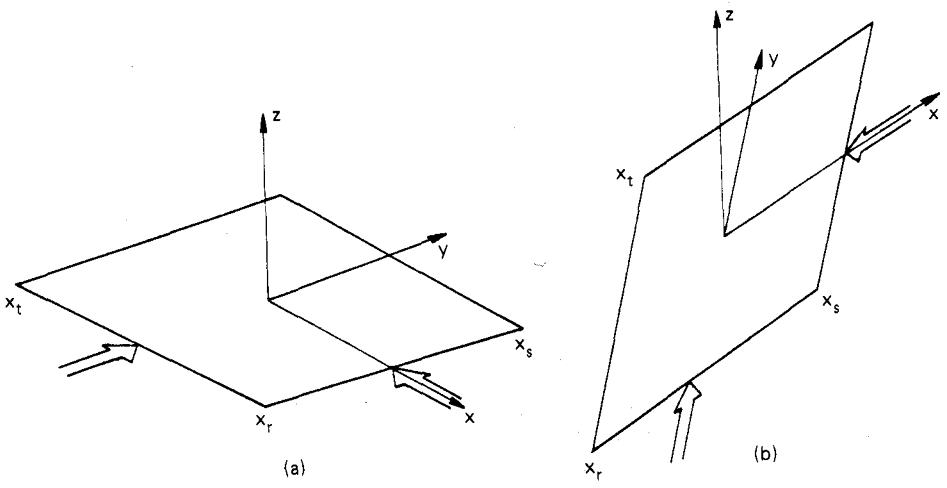

Figure 5.

index r. In general there are two basic possibilities, shown on figure 5.

The direction in which the slices are to be drawn are marked by $\Leftarrow$.

In case a) we can see that the margins from which we will start to draw line segments belong to the end-points X_rX_s and X_tX_r. In case b) we can see that we will fail. Therefore we have to test if

$$x'_t \leqslant x'_r \leqslant x'_s \text{ or } x'_s \leqslant x'_r \leqslant x'_t$$

If the boolean expression has value false then we have to find the second point which has minimal z' coordinate and which is different from the original point. The new point will be remarked by the index r.

The whole procedure can be described by the algorithm 3.

Algorithm 3.

```
1.  Find the index $r \in <1,4>$ so that
        $z'_r = \min\{z'_i\}\ i = 1,...,4$
2.  Find the indices of neighbours and mark them by
    indices t,s.
3.  If condition
        $x'_t \leqslant x'_r \leqslant x'_s$ OR $x'_s \leqslant x'_r \leqslant x'_t$
    has value FALSE then
    begin
        Find index $u \in <1,4>$ so that
            $z'_u = \min\{z'_i\}$  i = 1,...,4 and $i \neq r$
        r := u
        Find the indices of neighbours and mark them
            by indices t,s
    end
```

Now we can draw the function by drawing the slices according to the selected margins, which are defined by line segments with the end-points X_rX_t and X_rX_s.

But if we draw a function whose behaviour is wild enough then we obtain an incorrect picture (figure 6). It seems to be more convenient in this situation to apply the Zig-zag method and we will then obtain the correct result (figure 7).

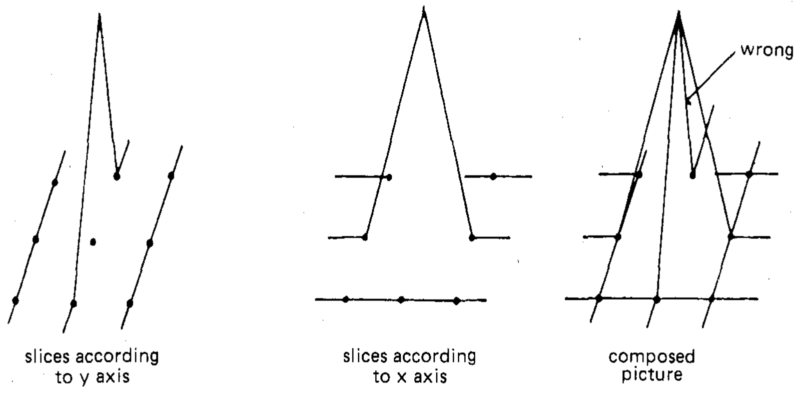

Figure 6.

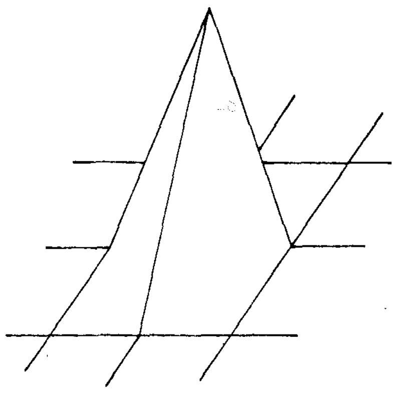

Figure 7.

The Zig-zag method can be described by:

1. Initialize the mask's arrays.
2. Draw the margins that are defined by the end-points $X'_r X'_s$ and $X'_r X'_t$ (steps 1,2).
3. Draw the function values according to the grid and according to the directions on figure 8 (steps 3.12).

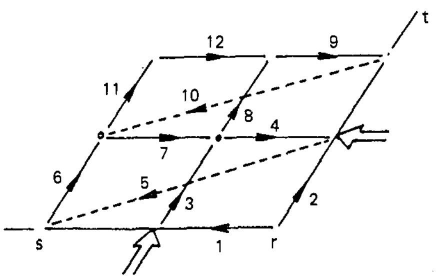

Figure 8.

Let us suppose that the function is given by the values

$$f[i,j] \quad i=1,...,n \text{ and } j=1,...,m$$

on the grid-points those coordinates are given by values

$$x[j] \quad j=1,...,m$$

$$x[i] \quad i=1,...,m$$

Then after transformation (rotation, translation) we receive values

$$x'[j],\ y'[i],\ f'[i,j],\ i=1,...,n \text{ and } j=1,...,m$$

Now the whole process of drawing can be made in the integer representation without using a floating point processor.

Conclusion

The algorithm presented for drawing functions of two variables with respect to visibility is intended for the use with microcomputers. Because the basic algorithm for visibility resolution can be realized by about ten assembly instructions it seems to be convenient to build it directly into the algorithm for a drawing straight lines. Now we can see that the basic graphics menu can be extended by the operations:

- initialize mask's arrays
- draw line with respect to the visibility

This means that the intelligence of graphic devices can be easily and significantly improved by adding some assembly instructions into the algorithm for the drawing lines. If a graphics display with grey scale is used the algorithm can be easily improved by using algorithms for drawing straight lines.

We can ask ourselves whether primitive-level instruction for drawing lines which takes account of visibility, should be a part of any basic graphics software system, e.g. GKS.

Acknowledgement

I would like thank Prof. L.M.V. Pitteway and Dr. J.P.A. Race for their many helpful discussions and suggestions that enabled me to finish this project successfully.

References

1. S.L. Watkins, "Masked three-dimensional plot program with rotation," Comm. of ACM 17(9), pp. 520-523 (September 1974).
2. H. Williamson, "Hidden-line plotting program," Comm. of ACM 15(2), pp. 100-103 (February 1972).
3. J. Boutland, "Surface drawing made simple," Computer Aided Design 11(1), pp. 19-22 (January 1979).
4. V. Skala, Hidden-line processor, CSTR/29, Computer Science Dept., Brunel University, Uxbridge, Middlesex (1984).
5. M. Pitteway and D. Watkinson, "Bresenham's algorithm with Grey scale," Comm. of ACM 23(11), pp. 625-626 (November 1980).
6. J.E. Bresenham, "Algorithm for computer control or digital plotter," IBM Syst. J. 4(1), pp. 25-30 (1965).
7. W.T. Sowerbutts, "A surface-plotting program suitable for microcomputers," Computer Aided Design 15(6), pp. 324-327 (November 1983).