Pergamon

PII: S0097-8493(98)00069-7

Comput. & Graphics, Vol. 22, No. 5, pp. 621–628, 1998
© 1998 Elsevier Science Ltd. All rights reserved
Printed in Great Britain
0097-8493/98/$ - see front matter

Technical Section

EFFICIENT INTEGER ALGORITHMS FOR THE GENERATION OF CONIC SECTIONS

A. AGATHOS, T. THEOHARIS† and A. BOEHM✠

Department of Informatics, University of Athens, Panepistimioupolis, TYPA Buildings, 157 71 Athens, Greece

Abstract—Efficient integer 8-connected algorithms for the fast generation of Conic Sections whose axes are aligned to the coordinate axes are described based on a Bresenham-like methodology. Performance results show that in the case of the ellipse, the algorithm is at least as fast as other known integer algorithms but requires lower integer range and always performs correct region transitions. Antialiasing is easily incorporated. © 1998 Elsevier Science Ltd. All rights reserved

1. INTRODUCTION

Conic sections (ellipse, hyperbola and parabola) are important geometric primitives and, after the straight line and circle, have received much attention from the computer graphics community. A number of algorithms have appeared in the literature for the generation of these primitives [1–7].

We have derived efficient *integer* 8-connected algorithms for the generation of ellipses hyperbolas and parabolas whose axes are aligned with the axes of the plane, using a Bresenham-like methodology [2] simulating, in effect, the midpoint technique [1]. Our algorithms use integer arithmetic in a straightforward manner without any scaling and do not lack in performance with regard to any previous algorithm. Spacewise, they require a small constant number of integer variables. They are also *symmetric* as to the number of arithmetic operations per pixel generated in each octant. Thus they are highly suitable for teaching. *Erroneous pixels* are *not* generated at region boundaries due to a better region transition criterion. In the case of the ellipse our algorithm requires *lower integer ravage* than Kappel's integer ellipse drawing algorithm. Our algorithms are very suitable for hardware implementation especially in view of increasing display resolutions and the availability of high-resolution plotters. Our parabola algorithm in particular, is suitable for very high resolutions due to the elimination of the calculation of a square factor. We are also able to exploit the value of the decision variable for antialiasing.

The rest of this paper is organised as follows: Section 2 presents a modified Bresenham circle algorithm. Section 3 describes the derivation of the ellipse generating algorithm. Section 4 briefly describes the parabola and hyperbola algorithm derivations. Appendix A, Appendix B and Appendix C give the ellipse, hyperbola and parabola Pascal procedures.

2. REFORMATTING THE BRESENHAM CIRCLE ALGORITHM

We develop a small variation to Bresenham's circle generating algorithm which concerns the criterion for next pixel selection and octant change detection. We shall later use this small change in a generalisation of the algorithm to the more complex conic sections; the integer algorithms derived in this way are correct and efficient.

Consider the second octant of a circle of integer Radius R (Fig. 1). As in Bresenham's algorithm, having chosen pixel A, we define:

$$d1 = R1^2 - R^2 = (y_i^2 + (x_i+1)^2) - (y^2 + (x_i+1)^2)$$
$$= y_i^2 - y^2,$$

$$d2 = R^2 - R2^2$$
$$= (y^2 + (x_i+1)^2) - ((y_i-1)^2 + (x_i+1)^2)$$
$$= y^2 - (y_i-1)^2$$

in a manner similar to Fellner [8]. We then take:

$$d = d1 - d2 = (y_i^2 - y^2) - (y^2 - (y_i-1)^2),$$

where d is the decision variable for the selection of the next pixel between the two candidates B and D.

At this point we take a diversion from Bresenham's algorithm by setting $\varepsilon = y_i - y$ to get:

$$d(\varepsilon) = -2\varepsilon^2 + 4y_i\varepsilon + 1 - 2y_i. \qquad (1)$$

†Corresponding author. Tel.: +0030-1-7275106; Fax: +0030-1-7231569; E-mail: theotheo@di.uoa.gr.

✠We would like to dedicate this paper to the memory of our dear friend Alexandros Boehm who died on May 1st 1998.

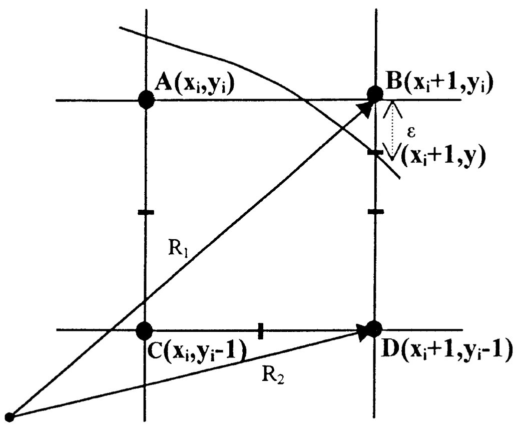


Fig. 1. Candidate Pixels B or D

The above expression is monotonically increasing in the interval $\varepsilon \in (-\infty, y_i]$. We can therefore use the value of $d(\varepsilon)$ at $\varepsilon = 1/2$ i.e. $d(1/2) = 1/2$ as the decision value:

if $d \leq 1/2$ then pixel B is chosen
else pixel D is chosen.

Note that since d is the integer, we can replace the 1/2 by 0, without affecting the semantics. Note that what we really accomplish here is to simulate a midpoint-type technique [1].

Due to the 8-way symmetry of the circle, we need not consider another octant; in the case of the ellipse, which has 4-way symmetry, we need to consider 2 regions which make up one-quarter of the ellipse.

3. DERIVATION OF THE ELLIPSE GENERATING ALGORITHM

Consider an ellipse centered at the origin of the 2D cartesian space defined by:

$$x^2/a^2 + y^2/b^2 = 1$$

The ellipse has a 4-way symmetry and it is therefore only necessary to generate its arc in the first quadrant. Here we distinguish two regions separated by the point on the ellipse where $dy/dx = -1$. In the first region the axis of major movement is X and in the second Y (Fig. 2).

We must derive an incremental expression for the decision variables in Regions 1 and 2, the initial values of the decision variables and a condition to detect the transition from Region 1 to Region 2.

3.1. *Decision variable for Region 1*

Let us begin by considering the 1st region of Fig. 2, where the X-axis is the major axis of movement. Assume that the ellipse is generated in a clockwise manner starting from the point $(0,b)$. At each step in the generation the X value is therefore always incremented. It must be determined whether the Y value should be decremented or not. This region corresponds to the 2nd octant of the circle (Fig. 1) and we define:

$$d1 = y_i^2 - y^2$$

$$d2 = y^2 - (y_i - 1)^2,$$

as in the case of the circle. Taking as decision variable:

$$d = a^2(d1 - d2),$$

the following will hold (Fig. 1):

if $d \leq a^2/2$ then pixel $B(x_i + 1, y_i)$ is chosen
else pixel $D(x_i + 1, y_i - 1)$ is chosen. (2)

The only reason for multiplying by a^2 is to facilitate its incremental derivation (see below). Of course the division of the integer value a^2 by 2 can be replaced by one Right Shift of a^2; since d is an integer the result is semantically equivalent.

We next derive the incremental computation of the decision variable; its value for the ith step of the algorithm in Region 1 is:

$$\begin{aligned} d_{1,i} &= a^2(d1 - d2) \\ &= a^2y_i^2 + a^2(y_i - 1)^2 - 2a^2y^2 \end{aligned}$$

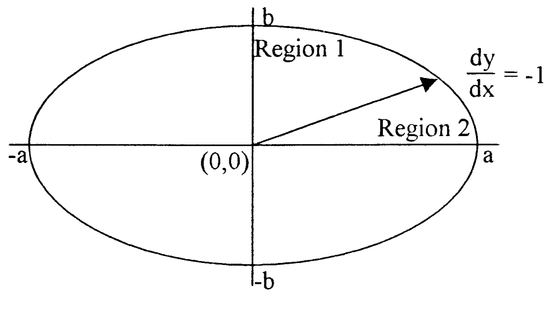


Fig. 2. Ellipse

Given that $a^2y^2 = a^2b^2 - b^2(x_i+1)^2$ [equation of ellipse],

$$d_{1,i} = -2a^2b^2 + 2b^2(x_i+1)^2 + a^2y_i^2 + a^2(y_i-1)^2. \qquad (3)$$

We shall now define $d_{1,i+1}$ in terms of $d_{1,i}$:

$$\begin{aligned} d_{1,i+1} &= -2a^2b^2 + 2b^2(x_{i+1}+1)^2 + a^2y_{i+1}^2 \\ &\quad + a^2(y_{i+1}-1)^2 \\ &= -2a^2b^2 + 2b^2((x_i+1)+1)^2 + a^2y_{i+1}^2 \\ &\quad + a^2(y_{i+1}-1)^2 \{x_{i+1} = x_i + 1\} \\ &= -2a^2b^2 + 2b^2(x_i+1)^2 + 2b^2 \\ &\quad + 4b^2(x_i+1) + a^2y_{i+1}^2 + a^2(y_{i+1}-1)^2 \qquad (4) \end{aligned}$$

But $-2a^2b^2 + 2b^2(x_i+1)^2 = d_{1,i} - a^2y_i^2 - a^2(y_i-1)^2$, therefore

$$\begin{aligned} d_{1,i+1} &= d_{1,i} + a^2y_{i+1}^2 + a^2(y_{i+1}-1)^2 \\ &\quad - a^2y_i^2 - a^2(y_i-1)^2 + 2b^2 + 4b^2(x_i+1) \end{aligned}$$

If $d_{1,i} > a^2/2$ then $y_{i+1} = y_i - 1$ by Equation (2), thus

$$d_{1,i+1} = d_{1,i} + 2b^2 + 4b^2(x_i+1) - 4a^2(y_i-1).$$

if $d_{1,i} \leq a^2/2$ then $y_{i+1} = y_i$ by Equation (2), thus

$$d_{1,i+1} = d_{1,i} + 2b^2 + 4b^2(x_i+1).$$

The initial value $d_{1,0}$ is determined by substituting the coordinates of the first pixel of Region 1 $(0,b)$ for (x_i, y_i) in the expression for $d_{1,i}$:

$$d_{1,0} = 2b^2 + a^2(1-2b)$$

3.2. *Transition from Region 1 to Region 2*

Van Aken [3] proposed a transition criterion based on midpoints which gives correct results but is computationally expensive. Kappel's method [4] is efficient but there exist cases where an erroneous pixel can arise at region boundaries. This is because:

i. Region change is detected by taking the tangent on integer coordinates rather than the true ellipse as acknowledged in Kappel's paper [4] and

ii. A drastic change of curvature can take place within a single pixel.

We propose a transition criterion which combines the advantages of the above two methods (see Fig. 3).

Our integer algorithm uses a correct criterion which is based on the value of the error function in the next column of pixels. In particular, the value of the error function d at the point $(x_i+1, y_i-3/2)$ is considered; note that if the ellipse 'passes under' this point then an octant transition is required.

We can express the error function d given in Equation (3) in terms of $\varepsilon(=y_i - y)$ in a manner similar to the circle (1):

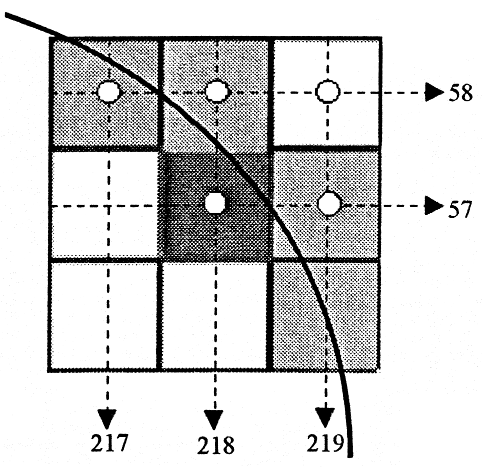
$$d(\varepsilon) = -2\,a^2\,\varepsilon^2 + 4\,a^2y_i\varepsilon + a^2 - 2\,a^2y_i$$

setting $\varepsilon = 3/2$ we get:

$$\begin{aligned} d(3/2) &= -2\,a^2(3/2)^2 + 4\,a^2y_i(3/2) + a^2 - 2\,a^2y_i \\ &= 4\,a^2(y_i-1) + a^2/2. \end{aligned}$$

The transition criterion is as follows:

if $d \leq 4a^2(y_i-1) + a^2/2$ then we remain in the same region



a=245 b=126

Already Estimated Grid Point		=	*(218,58)*
Approximate Slope		=	-0.9941
True Slope		=	-1.0027
Next Grid Point :			
	Y – Coordinate	=	*57*
Kappel	X-Coordinate	=	219
Our Algorithm	X-Coordinate	=	218
Van Aken	X-Coordinate	=	218
True	X-Coordinate	=	218.4971

Fig. 3. Transition from Region 1 to Region 2

else we change region.

As we have already mentioned above the division $a^2/2$ can be replaced by a semantically equivalent Right Shift. The above criterion is optimised and used in the code fragment in Appendix A. We have to note that the above criterion works correctly for $y_i \geq 1$. Square corners, as mentioned by McIlroy [6], can be predicted easily in our algorithm using one copy of the pixel last printed in the first region. Other degenerate cases, such as McIlroy's long thin ellipses, have been tested. It is heuristically believed that no degenerate cases will cause the algorithm to fail.

The initial value of the decision variable $d_{2,i}$ for Region 2 (see Equation (5) below) can be calculated by adding to the final value of $d_{1,i}$ the difference $d_{2,i} - d_{1,i}$. From Equation (4) and Equation (5):

$$d_{2,i} = d_{1,i} - a^2(2y_i - 1) - b^2(2x_i + 1).$$

3.3. *Decision variable for Region 2*

In Region 2 the expressions for $d1$ and $d2$, as can be seen from Fig. 4, are:

$$d_1 = (x_i + 1)^2 - x^2$$
$$d_2 = x^2 - x_i^2$$

Having chosen pixel $A(x_i, y_i)$, the difference $d = d1 - d2$ determines which of the 2 pixels in the next row of pixels ($y = y_i - 1$) is closer to the real ellipse:

if $d \leq b^2/2$ then pixel $D(x_i + 1, y_i - 1)$ is selected

else pixel $C(x_i, y_i - 1)$ is selected.

The decision variable (scaling again by b^2) for Region 2 step i is defined to be:

$$\begin{aligned} d_{2,i} &= b^2(d1 - d2) \\ &= b^2(x_i + 1)^2 + b^2x_i^2 - 2b^2x^2 \qquad (4) \end{aligned}$$

Given that $b^2x^2 = a^2b^2 - a^2(y_i - 1)^2$ [equation of ellipse],

$$\begin{aligned} d_{2,i} = &- 2a^2b^2 + b^2x_i^2 + 2a^2(y_i - 1)^2 \\ &+ b^2(x_i + 1)^2 \qquad (5) \end{aligned}$$

An incremental expression for $d_{2,i}$ can be derived in a similar manner to $d_{1,i}$ to be:

$$\begin{aligned} d_{2,i+1} = &\, d_{2,i} + b^2x_{i+1}^2 + b^2(x_{i+1} + 1)^2 - b^2x_i^2 \\ &- b^2(x_i + 1)^2 + 2a^2 - 4a^2(y_i - 1) \end{aligned}$$

which can be simplified, depending on the value of x_{i+1}, as follows:

if $d_{2,i} > b^2/2$ then $x_{i+1} = x_i$, thus

$$d_{2,i+1} = d_{2,i} + 2a^2 - 4a^2(y_i - 1),$$

if $d_{2,i} < b^2/2$ then $x_{i+1} = x_i + 1$ thus

$$d_{2,i+1} = d_{2,i} + 2a^2 + 4b^2(x_i + 1) - 4a^2(y_i - 1).$$

3.4. *Antialiasing and results*

In the past a linear antialiasing function for conic sections has been proposed [5]. A similar function could be applied to our algorithm (specifically the function $(d(\varepsilon) - d(0))/(d(1) - d(0))$), but unfortunately such linear approximation only works correctly for very few bits of colour ([9], page 971).

We have incorporated a very fast version of the box-filtering antialising technique [10] in the ellipse drawing algorithm, achieving satisfactory results, see Fig. 5.

Since it is computationally expensive to compute the reverse of $d(\varepsilon)$ function, we perform a binary search of the given value of d in the space of values $d(t/n, y_i)$, $t = 0 \ldots n$ where n is the number of grey levels available to determine the required grey level (in the second region we would use $d(t/n, x_i)$). The

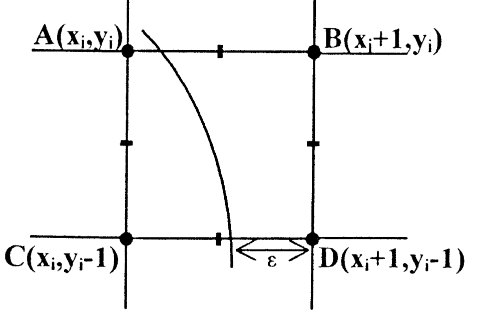


Fig. 4. Ellipse Construction (Region 2)

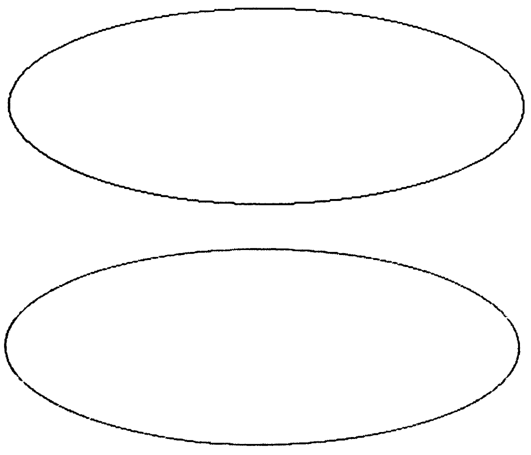


Fig. 5. Midpoint and Antialiased ellipse (4 Bits per pixel)

values of the function $d(t/n, y_i)$ can be precomputed. Note that in the case of antialiasing we do not follow the midpoint philosophy as it is always necessary to keep the 2 pixels above and below the true y-intercept for Region 1. The y value of these 2 pixels is decremented when $d \geq d(1,y_i)$ and an octant change occurs when $d \geq d(2,y_i)$.

Our 8-connected algorithm produces equally good antialiased results as 4-connected algorithms [5], but an 8-connected algorithm is advantageous in the case of a single bit per pixel since it provides regions of constant thickness.

The time performance of the new algorithm was compared against the algorithm described by Kappel [4] as well as an integer version of Kappel's algorithm which we derived by suitably scaling by 4 its variables in order to achieve the best possible performance. The integer Kappel algorithm exhibits similar performance to our ellipse algorithm; this should be expected because the integer version of Kappel we derived is very similar in structure to our algorithm. However, the integer Kappel produces arithmetic overflow quicker than ours. It also requires a greater integer range as can be seen in Scheme 1 which compares the two algorithms in terms of the maximum integer value required, as ellipse size increases. The maximum integer arises in the calculation of y_slope in both of the algorithms.

It must be restated here that Kappel's algorithm can give rise to erroneous pixels at the 4 region boundaries as pointed out by Kappel (see Fig. 3). Our integer algorithm does not exhibit this problem.

4. THE HYPERBOLA AND PARABOLA ALGORITHMS

In a similar manner to the ellipse, one can derive incremental error expressions for the construction of our hyperbola and parabola generating algorithms.

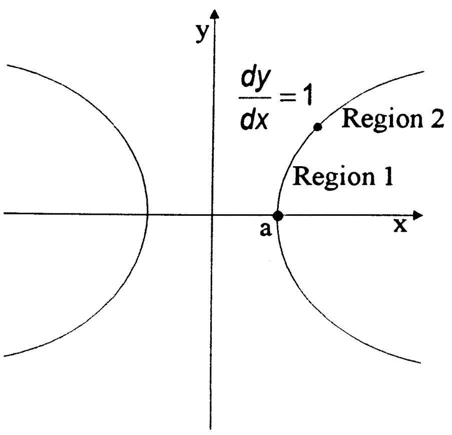


Fig. 6. Hyperbola.

4.1. *Hyperbola*

Figure 6 shows a hyperbola centered at (0,0), symmetric about the X and Y-axes, defined by the equation

$$x^2/a^2 - y^2/b^2 = 1$$

We consider here only the case $a > b$ in which the hyperbola has 2 regions, one in which the major axis of movement is Y (Region 1) and another in which the major axis of movement is X (Region 2). If $a < = b$ there is no Region 2. The two regions are separated by the point where the tangent to the hyperbola has slope $dy/dx = 1$.

In Region 1 (see Fig. 7), the expressions for a measure of the distance of the true hyperbola to the 2 nearest pixels are:

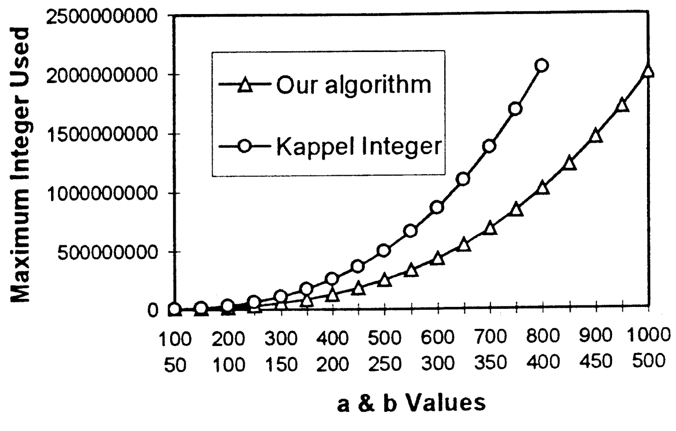
$$d1 = b^2x^2 - b^2x_i^2$$

$$b2 = b^2(x_i + 1)^2 - b^2x^2$$

Setting $\varepsilon = x - x_i$ we get:

$$d(\varepsilon) = d1 - d2$$

$$= 2b^2\varepsilon^2 + 4b^2x_i\varepsilon - b^2 - 2b^2x_i. \quad (5)$$



Scheme 1. Maximum integer graph.

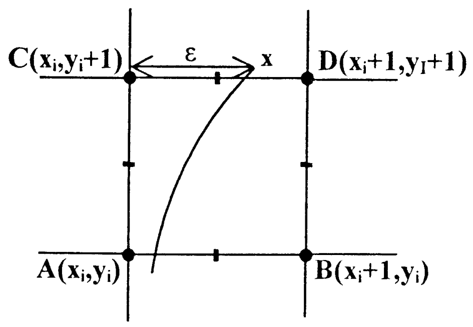


Fig. 7. Hyperbola construction (Region 1).

The above expression is monotonically increasing in the interval $\varepsilon \in [-x_i, +\infty)$. Thus by noting that $d(1/2) = -b^2/2$, the following will hold (Fig. 7):

if $d \geq -b^2/2$ then pixel $D(x_i+1, y_i+1)$ is chosen
else pixel $C(x_i, y_i+1)$ is chosen. (6)

We next derive the incremental computation of the decision variable whose value for the ith step of the algorithm in Region 1 is:

$$d_{1,i} = d1 - d2$$
$$= 2a^2b^2 + 2a^2(y_i+1)^2 - b^2x_i^2 - b^2(x_i+1)^2$$

which can be incrementally derived to be:

if $d_{1,i} \geq -b^2/2$ then $x_{i+1} = x_i + 1$ by Equation (6), thus

$$d_{1,i+1} = d_{1,i} + 2a^2 + 4a^2(y_i+1) - 4b^2(x_i+1),$$

if $d_{1,i} < -b^2/2$ then $x_{i+1} = x_i$ by Equation (6), thus

$$d_{1,i+1} = d_{1,i} + 2a^2 + 4a^2(y_i+1).$$

The initial value $d_{1,0}$ is determined by substituting the coordinates of the first pixel of Region 1 $(a,0)$ for (x_i, y_i) in the expression for $d_{1,i}$ in Equation (7):

$$d_{1,0} = 2a^2 - b^2(1+2a).$$

In Region 2, expressions for a measure of the distance of the true parabola to the 2 nearest pixel centers are:

$$d1 = a^2(y_i+1)^2 - a^2y^2$$
$$d2 = a^2y^2 - a^2y_i^2$$

The error term is:

$$d_{2,i} = d1 - d2$$
$$= 2a^2b^2 - 2b^2(x_i+1)^2 + a^2(y_i+1)^2 + a^2y_i^2$$

which can be incrementally derived to be:

if $d_{2,i} \leq a^2/2$ then $y_{i+1} = y_i + 1$, thus

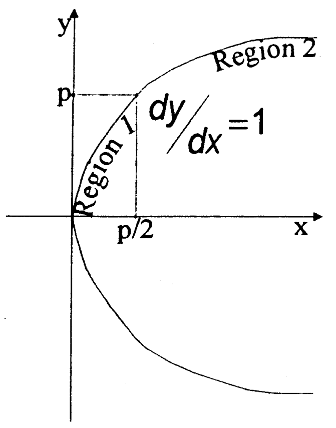


Fig. 8. Parabola.

$$d_{2,i+1} = d_{2,i} - 2b^2 - 4b^2(x_i+1) + 4a^2(y_i+1),$$

if $d_{2,i} > a^2/2$ then $y_{i+1} = y_i$, thus

$$d_{2,i+1} = d_{2,i} - 2b^2 - 4b^2(x_i+1)$$

In a manner similar to the ellipse, the transition criterion from Region 1 to Region 2 is as follows:

if $d < 4\,b^2(x_i+1) - b^2/2$ then we remain in the same region else we change region.

The expression for the initial value of the error term in Region 2 can then be derived:

$$d_{2,i} = d_{1,i} - b^2(1+2x_i) - a^2(1+2y_i)$$

4.2. *Parabola*

Figure 8 shows a parabola centered at (0,0) symmetric about the X-axis defined by the equation

$$y^2 = 2px$$

In Region 1 the axis of major movement is Y while in Region 2 it is X. The two regions meet at $x = p/2$, $y = p$ where the tangent to the parabola has slope $dy/dx = 1$.

In Region 1 the expressions for a measure of the distance of the true parabola to the 2 nearest pixels are:

$$d1 = px - px_i$$
$$d2 = p(x_i+1) - px$$

The error term is:

$$d_{1,i} = d1 - d2$$
$$= (y_i + 1)^2 - px_i - p(x_i + 1)$$

which can be incrementally derived to be:

if $d_{1,i} \geq 0$ then $x_{i+1} = x_i + 1$, thus

$$d_{1,i+1} = d_{1,i} + 2(y_i + 1) + 1 - 2p,$$

if $d_{1,i} < 0$ then $x_{i+1} = x_i$, thus

$$d_{1,i+1} = d_{1,i} + 2(y_i + 1) + 1,$$

In Region 2, expressions for a measure of the distance of the true parabola to the 2 nearest pixel centers are:

$$d1 = (y_i + 1)^2 - y^2$$
$$d2 = y^2 - y_i^2$$

The error term is:

$$d_{2,i} = d1 - d2$$
$$= (y_i + 1)^2 + y_i^2 4p(x_i + 1)$$

which can be incrementally derived to be:

if $d_{2,i} < 0$ then $y_{i+1} = y_i + 1$, thus

$$d_{2,i+1} = d_{2,i} + 4(y_i + 1) - 4p,$$

if $d_{2,i} > 0$ then $y_{i+1} = y_i$, thus

$$d_{2,i+1} = d_{2,i} - 4p,$$

The expression for the error, when making the transition from Region 1 to Region 2 can be derived to be:

$$d_{2,i} = d_{1,i} + y_i^2 - p(2x_i + 3),$$

The square in the calculation of $d_{2,i}$ gives rise to large integers and is unsuitable for hardware implementation. We have proved and verified experimentally that the final value of $d_{1,i}$ will be 1 or $p + 1$ and:

if $d_{1,i} = 1$ then $d_{2,i} = -4p + 1$,
if $d_{1,i} = p + 1$ then $d_{2,i} = -2p + 1$,

5. CONCLUSIONS

Despite years of research into basic graphics algorithms, new algorithms still emerge. The integer algorithms for conic sections described in this paper have straightforward Bresenham-like symmetric derivations, are at least as fast as previous integer algorithms, require lower integer arithmetic precision and do not set erroneous pixels at region boundaries, thus incorporating the advantages of well-known previous algorithms. They are very suitable for high performance applications and teaching. Fast antialiasing can also be incorporated.

REFERENCES

1. Pitteway, M. L. V., Algorithms for drawing ellipses or hyperbolae with a digital plotter. *Computer J.*, 1967, **10**(3), 282–289.
2. Bresenham, J. E., A linear algorithm for incremental digital display of circular arcs. *CACM*, 1977, **20**(2), 100–106.
3. Van Aken, J. R., An efficient ellipse-drawing algorithm. *CG&A*, 1984, **4**(9), 24–35.
4. Kappel M. R., An Ellipse-Drawing Algorithm for Raster Displays. In Earnshaw R. (ed) *Fundamental Algorithms for Computer Graphics*, NATO ASI Series, Springer-Verlag, Berlin, 1985, pp. 257–280.
5. Pitteway M. L. V. and Ebadollah Banissi, Soft Edging Fonts. Computer Graphics Technology and Systems. In *Proceedings of the conference held at Computer Graphics '87*, London, October 1987.
6. McIlroy, M. D., Getting Raster ellipses right. *ACM TOG*, 1992, **11**(3), 259–275.
7. Da Silva D., Raster Algorithms for 2D Primitives. Master's Thesis, Computer Science Department, Brown University, Providence, R.I., 1989.
8. Fellner W. D., Computer Grafik. Bibliografisches Institut, Zuerich, 1992.
9. Foley J. D. *et al.*, Computer Graphics, Principles and Practice, 2nd Edn. Addison-Wesley, 1990.
10. Wu, X., An efficient antialiasing technique. *Computer Graphics*, 1991, **25**(4), 143–152.

APPENDIX A

Ellipse Pascal Code

```
Procedure Ellipse(a,b:longint);
var a_sqr,b_sqr,a22,b22,a42,b42,x_slope,y_slope:longint;
    d,mida,midb:longint;
    x,y:integer;

begin
    x:=0;
    y:=b;
    a_sqr:=sqr(a);
    b_sqr:=sqr(b);
    a22:=a_sqr + a_sqr;
    b22:=b_sqr + b_sqr;
    a42:=a22 + a22;
    b42:=b22 + b22;
    x_slope:=b42;       {x_slope = (4*b^^2)*(x + 1) always}
    y_slope:=a42*(y-1); {y_slope = (4*a^^2)*(y - 1) always}
    mida:=a_sqr SHR 1; {a^^2 div 2}
    midb:=b_sqr SHR 1; {b^^2 div 2}
    d:=b22 - a_sqr - y_slope SHR 1 - mida;
    {subtract a^^2 div 2 to optimise}

    {Region 1}
    while d <= y_slope do
    begin
        Draw(x,y);
        if d > 0 then
        begin
            d:=d - y_slope;
            y:=y - 1;
            y_slope:=y_slope - a42;
        end;
        d:=d + b22 + x_slope;
        x:=x + 1;
        x_slope:=x_slope + b42;
    end;

    d:=d-(x_slope+y_slope) SHR 1 +(b_sqr-a_sqr)+(mida-midb)
    {Optimised region change using x_slope , y_slope}
```

```
      {Region 2}
      while y >= 0 do
      begin
          Draw(x,y);
          if d <= 0 then
          begin
              d:=d + x_slope;
              x:=x + 1;
              x_slope := x_slope + b42;
          end;
          d:=d + a22 - y_slope;
          y:=y-1;
          y_slope:=y_slope - a42;
      end;
  end;
```

APPENDIX B

Hyperbola Pascal Code

```
Procedure Hyperbola(a,b:longint;bound:integer);
{bound limits the hyperbola in y}
var x,y,d,mida,midb:longint;
    a22,b22,a_sqr,b_sqr:longint;
    a42,b42:longint;
    x_slope,y_slope:longint;
begin
    x:=a;
    y:=0;
    a_sqr:=sqr(a);
    b_sqr:=sqr(b);
    a22:=a_sqr+a_sqr;
    b22:=b_sqr+b_sqr;
    a42:=a22+a22;
    b42:=b22+b22;
    x_slope:=b42*(x+1); {x_slope = (4*b^^2) * (x + 1) always }
    y_slope:=a42;       {y_slope = (4*a^^2) * (y + 1) always }
    mida:=a_sqr shr 1;{a^^2 div 2}
    midb:=b_sqr shr 1;{b^^2 div 2}
    d:=a22 - b_sqr * (1+2*a) + midb; {add b^^2 div 2 to optimize}

    {Region 1}
    while (d < x_slope) and (y<=bound) do
    begin
        Draw(x,y);
        if d >= 0 then
        begin
            d:=d - x_slope;
            x:=x + 1;
            x_slope:=x_slope + b42;
        end;
        d := d + a22 + y_slope;
        y := y+1;
        y_slope := y_slope + a42;
    end;

    d:=d - (x_slope + y_slope) shr 1 + (a_sqr+b_sqr) - midb - mida;
    {optimised region change using x_slope , y_slope}

    {Region 2}
    if a>b then
    while y <= bound do
    begin
        Draw(x,y);
        if d<=0 then
        begin
            d:=d+y_slope;
            y:=y+1;
            y_slope:=y_slope + a42;
        end;
        d:=d - b22 - x_slope;
        x:=x + 1;
        x_slope:=x_slope + b42;
    end;
end;
```

APPENDIX C

Parabola Pascal Code

```
Procedure Parabola(p,bound:integer);
{bound limits the parabola in x}
var x,y,d:integer;
    p2,p4:integer;
begin
    p2 := 2*p;
    p4 := 2*p2;
    x  := 0;
    y  := 0;
    d := 1 - p ;

    {Region 1}
    while (y < p) and (x <= bound) do
    begin
        Draw(x,y);
        if d >= 0 then
        begin
            x := x + 1;
            d := d - p2;
        end;
        y := y + 1;
        d := d + 2*y + 1;
    end;
    if d = 1 then d := 1 - p4
    else d := 1 - p2;

    {Region 2}
    while x <= bound do
    begin
        Draw(x,y);
        if d <=0 then
        begin
            y := y + 1;
            d := d + 4*y;
        end;
        x := x + 1;
        d := d - p4;
    end;
end;
```