Generating Conic Sections Using an Efficient Algorithms

Abdul-Aziz Solyman Khalil

Department of Computer Science, College of Computers and Mathematical Sciences, University of Mosul, Mosul, Iraq

Abstract

Efficient integer algorithms for the fast generation of Conic Sections whose axes are aligned to the coordinate axes are described based on a Bresenham-like methodology and simulating midpoint algorithm concepts. 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. Also efficient techniques for generating hyperbola and parabola are designed.

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 [3][5]. We must derived efficient integer 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 the midpoint technique [1].

Our algorithms use integer arithmetic in a straight forward 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. 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 [3]. Our algorithms are very suitable for hardware implementation especially in view of increasing display resolutions. Our parabola algorithm in particular is suitable for very high resolutions due to the elimination of the calculation of a square factor.

The rest of this paper is organized 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 functions wrote using C++.

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[2][5].

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

$$d1 = R1^2 - R^2$$
$$= \left(y_i^2 + (x_i+1)^2\right) - \left(y^2 + (x_i+1)^2\right)$$
$$= y_i^2 - y^i \quad \ldots\ldots (1)$$
$$d2 = R^2 - R2^2$$
$$= \left(y^2 + (x_i+1)^2\right) - \left((y_i-1)^2 + (x_i+1)^2\right)$$
$$= y^2 - (y_i-1)^2 \quad \ldots\ldots\ldots (2)$$

Where R1 & R2 are the distances from the circle center to B & D pixels respectively.

We then take [2]:

$$d = d1 - d2 = \left(y_i^2 - y^2\right) - \left(y^2 - (y_i-1)^2\right) \ldots\ldots (3)$$

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

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

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

Derivation of the ellipse generating algorithm

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

$$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1 \quad \ldots\ldots\ldots\ldots (5)$$

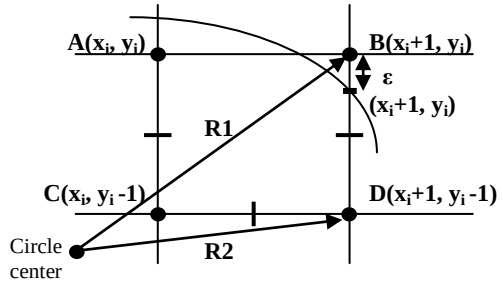


Fig.1 Candidate pixels B or D

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

d(1/2) =1/2 as the decision value:

if d ≤ 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 [4][6].

The ellipse has a 4-way symmetry and it is therefore only necessary to generate its arc in the first quadrant [1][2]. 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.

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 $(0,r_y)$. 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:

$$d1 = y_i^2 - y^2 \quad \ldots\ldots\ldots (6)$$

$$d2 = y^2 - (y_i - 1)^2 \quad \ldots\ldots (7)$$

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

$$d = r_x^2 (d1 - d2) \quad \ldots\ldots\ldots (8)$$

the following will hold (Fig. 1):

if $d \le r_x^2 / 2$ then pixel $B(x_i+1, y_i)$ is chosen

else pixel $D(x_i+1, y_i-1)$ is chosen: …….. (9)

The only reason for multiplying by r_x^2 is to facilitate its incremental derivation (see below).

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

$$d_{1,i} = r_x^2 (d1 - d2)$$

$$= r_x^2 y_i^2 + r_x^2 (y_i - 1)^2 - 2 r_x^2 y^2 \quad \ldots\ldots\ldots (10)$$

Given that $r_x^2 y^2 = r_x^2 r_y^2 - r_y^2 (x_i + 1)^2$ [equation of ellipse]

$$d_{1,i} = -2 r_x^2 r_y^2 + 2 r_y^2 (x_i + 1)^2 + r_x^2 y_i^2 + r_x^2 (y_i - 1)^2 \ldots (11)$$

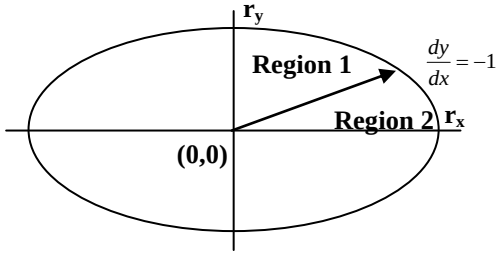


Fig. 2 Ellipse

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

$$d_{1,i+1} = -2 r_x^2 r_y^2 + 2 r_y^2 (x_{i+1} + 1)^2 + r_x^2 y_{i+1}^2 + r_x^2 (y_{i+1} - 1)^2$$

$$= -2 r_x^2 r_y^2 + 2 r_y^2 ((x_i + 1) + 1)^2 + r_x^2 y_{i+1}^2 + r_x^2 (y_{i+1} - 1)^2$$

since $x_{i+1} = x_i + 1$

$$= -2 r_x^2 r_y^2 + 2 r_y^2 (x_i + 1)^2 + 2 r_y^2 + 4 r_y^2 (x_i + 1)$$

$$+ r_x^2 y_{i+1}^2 + r_x^2 (y_{i+1} - 1)^2 \ldots\ldots\ldots (12)$$

But

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

therefore

$$d_{1,i+1} = d_{1,i} + r_x^2 y_{i+1}^2 + r_x^2 (y_{i+1} - 1)^2 - r_x^2 y_i^2$$

$$- r_x^2 (y_i - 1)^2 + 2 r_y^2 + 4 r_y^2 (x_i + 1) \ldots\ldots (13)$$

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

$$d_{1,i+1} = d_{1,i} + 2 r_y^2 - 4 r_x^2 (y_i - 1) + 4 r_y^2 (x_i + 1) \quad \ldots\ldots\ldots (14)$$

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

$$d_{1,i+1} = d_{1,i} + 2 r_y^2 + 4 r_y^2 (x_i + 1) \quad \ldots\ldots\ldots (15)$$

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

$$d_{1,0} = 2 r_y^2 + r_x^2 (1 - 2 r_y) \quad \ldots\ldots\ldots (16)$$

Transition from Region 1 to Region 2

The transition criterion based on midpoints algorithm 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:

1. Taking the tangent on integer coordinates rather than the true ellipse detect region change.
2. A drastic change of curvature can take place within a single pixel.

This method proposes a transition criterion, which combines the advantages of the above two methods.

The 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.

The error function d given in Equation (11) can be in terms of $\varepsilon = (y_i - y)$ in a manner similar to the circle (4):

$$d(\varepsilon) = -2 r_x^2 \varepsilon^2 + 4 r_x^2 y_i \varepsilon + r_x^2 - 2 r_x^2 y_i \quad \ldots\ldots\ldots (17)$$

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

$$d(3/2) = -2 r_x^2 (3/2)^2 + 4 r_x^2 y_i (3/2) + r_x^2 - 2 r_x^2 y_i$$

$$= 4 r_x^2 (y_i - 1) + r_x^2/2 \quad \ldots\ldots\ldots (18)$$

The transition criterion is as follows:

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

The above criterion is optimized and used in the code fragment in Appendix A. we have to note that the above criterion works correctly for $y_i \ge 1$. Square corners can be predicted easily in this algorithm using one copy of the pixel last printed in the first region.

The initial value of the decision variable d2,i for Region 2 (see Equation (23) below) can be calculated by adding to the final value of $d_{1,i}$ the difference $d_{2,i} - d_{1,i}$. From Equation (12) and Equation (23):

$$d_{2,i} = d_{1,i} - r_x^2 (2 y_i - 1) - r_y^2 (2 x_i + 1) \quad \ldots\ldots\ldots (19)$$

Decision variable for Region 2

In Region 2 the expressions for d1 and d2, as can be seen from Fig. 3, are:

$$d1 = (x_i + 1)^2 - x^2 \quad \ldots\ldots\ldots (20)$$

$$d2 = x^2 - x_i^2 \quad \ldots\ldots\ldots (21)$$

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 ≤ r_y^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 r_y^2) for Region 2 step i is defined to be:

$$d_{2,i} = r_y^2(d1 - d2)$$

$$= r_y^2(x_i + 1)^2 + r_y^2 x_i^2 - 2r_y^2 x^2 \quad \text{..... (22)}$$

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

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

$$+ r_y^2 (x_i + 1)^2 \text{.....(23)}$$

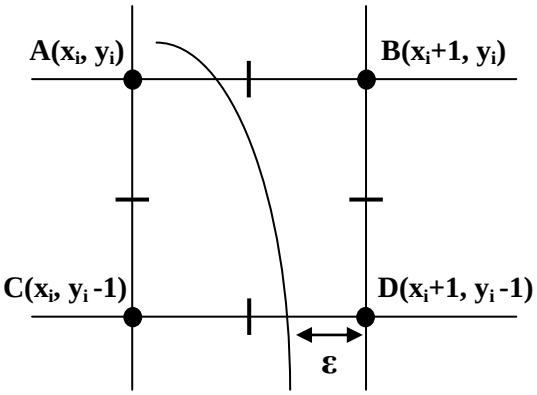


Fig. 3 Ellipse Construction (Region 2)

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

$$d_{2,i+1} = d_{2,i} + r_y^2 x_{i+1}^2 + r_y^2 (x_{i+1} + 1)^2 - r_y^2 x_i^2$$

$$- r_y^2 (x_i + 1)^2 + 2r_x^2 - 4r_x^2 (y_i - 1) \text{.....(24)}$$

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

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

$$d_{2,i+1} = d_{2,i} + 2r_x^2 - 4r_x^2 (y_i - 1) \text{ ..... (25)}$$

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

$$d_{2,i+1} = d_{2,i} + 4r_y^2 (x_i + 1) + 2r_x^2 - 4r_x^2 (y_i - 1) \text{ ..... (25)}$$

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.

Hyperbola

Figure 4 shows a hyperbola centered at (0,0), symmetric about the X and Y-axes, defined by the equation [12][23]

$$\frac{x^2}{r_x^2} - \frac{y^2}{r_y^2} = 1 \text{ ..... (26)}$$

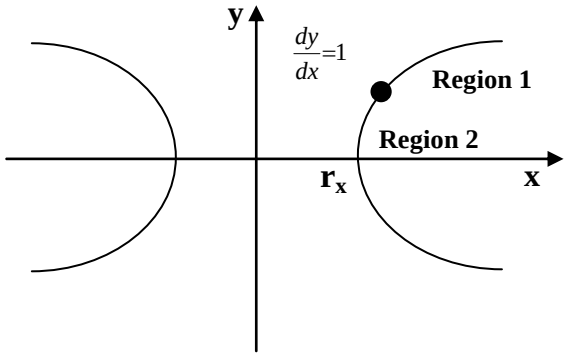


Fig. 4 Hyperbola

We consider here only the case $r_x > r_y$ in which the hyperbola has 2 regions, one in which the major axis of movement is Y (Region 1) i.e. Y will increment automatically, and another in which the major axis of movement is X (Region 2).

If $r_x <= r_y$ 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. 5), the expressions for a measure of the distance of the true hyperbola to the 2 nearest pixels are:

$$d1 = r_y^2 x^2 - r_y^2 x_i^2 \text{ ..... (27)}$$

$$d2 = r_y^2 (x_i + 1)^2 - r_y^2 x^2 \text{ ..... (28)}$$

Setting ε = x - x_i we get:

d(ε) = d1- d2

$$= 2r_y^2 \varepsilon^2 + 4r_y^2 x_i \varepsilon - r_y^2 - 2r_y^2 x_i \text{ ..... (29)}$$

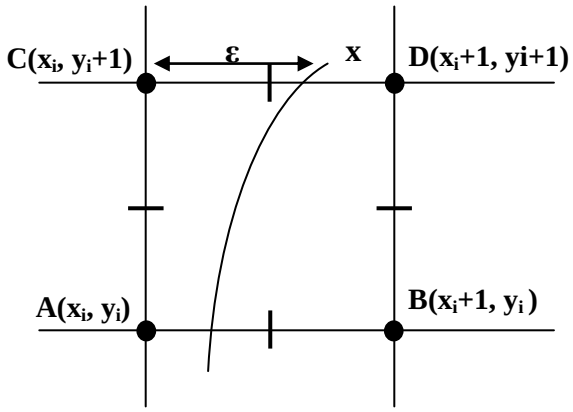


Fig. 5 Hyperbola Construction (Region 1)

The above expression is monotonically increasing in the interval $\varepsilon \in \left[-x_i, +\infty\right]$. Thus by noting that d (1/2) = - $r_y^2/2$, the following will hold (Fig. 5):

if d ≥ - $r_y^2/2$ then pixel D(x_i +1, y_i + 1) is chosen

else pixel C(x_i, y_i + 1) is chosen: ..... (30)

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

$d_{i,1}$ = d1 - d2

$$= 2r_x^2 r_y^2 + 2r_x^2 (y_i + 1)^2 - r_y^2 x_i^2 - r_y^2 (x_i + 1)^2 \text{ ..... (31)}$$

which can be incrementally derived to be:

if $d_{1,i} \geq -\ r_y^2/2$ then $x_{i+1}=x_i + 1$ by Equation (30), thus

$$d_{1,i+1} = d_{1,i} + 2r_x^2 + 4r_x^2(y_i + 1) - 4r_y^2(x_i + 1) \text{ ..... (32)}$$

if $d_{1,i} < -\ r_y^2/2$ then $x_{i+1} = x_i$ by Equation (30), thus

$$d_{1,i+1} = d_{1,i} + 2r_x^2 + 4r_x^2(y_i + 1) \text{ ..... (33)}$$

The initial value $d_{1,0}$ is determined by substituting the coordinates of the first pixel of Region 1 $(r_x,0)$ for (x_i, y_i) in the expression for $d_{1,i}$ in Equation (31):

$$d_{1,0} = 2r_x^2 - r_y^2(1 + 2r_x) \text{ ..... (34)}$$

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

$$d1 = r_x^2(y_i + 1)^2 - r_x^2 y^2 \text{ ..... (35)}$$

$$d2 = r_x^2 y^2 - r_x^2 y_i^2 \text{ ..... (36)}$$

The error term is:

$d_{2,i} = d1 - d2$

$$= 2r_x^2 r_y^2 - 2r_y^2(x_i + 1)^2 + r_x^2(y_i + 1)^2 + r_x^2 y_i^2 \text{ ..... (37)}$$

which can be incrementally derived to be:

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

$$d_{2,i+1} = d_{2,i} - 2r_y^2 - 4r_y^2(x_i + 1) + 4r_x^2(y_i + 1) \text{ ..... (38)}$$

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

$$d_{2,i+1} = d_{2,i} - 2r_y^2 - 4r_y^2(x_i + 1) \text{ ..... (39)}$$

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

if $d < 4\ r_y^2\ (x_i+1) - r_y^2/2$ 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:

$$d_{2,i} = d_{1,i} - r_y^2(1+2x_i) - r_x^2(1+2y_i) \text{ ..... (40)}$$

Parabola

Figure 6 shows a parabola centered at (0,0) symmetric about the X-axis defined by the equation [3][5]:

$y^2 = 2px$ ..... (41)

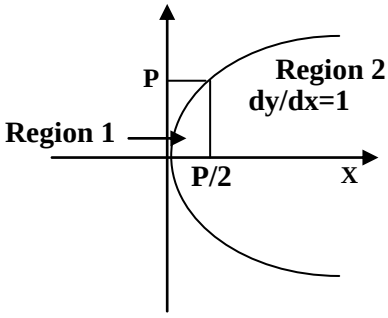


Fig. 6 Parabola

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:

$d1 = px - px_i$ ..... (42)

$d2 = p(x_i + 1) - px$ ..... (43)

The error term is:

$d_{1,i} = d1 - d2$

$$= (y_i + 1)^2 - px_i - p(x_i + 1) \text{ ..... (44)}$$

which can be incrementally derived to be:

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

$$d_{1,i+1} = d_{1,i} = 2(y_i + 1) + 1 - 2p \text{ ..... (45)}$$

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

$$d_{1,i+1} = d_{1,i} + 2(y_i + 1) + 1 \text{ ..... (46)}$$

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

$$d1 = (y_i + 1)^2 - y^2 \text{ ..... (47)}$$

$$d2 = y^2 - y_i^2 \text{ ..... (48)}$$

The error term is:

$d_{2,i} = d1 - d2$

$$= (y_i + 1)^2 + y_i^2 4p(x_i + 1) \text{ ..... (49)}$$

which can be incrementally derived to be:

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

$$d_{2,i+1} = d_{2,i} + 4(y_i + 1) - 4p \text{ ..... (50)}$$

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

$$d_{2,i+1} = d_{2,i} - 4p \text{ ..... (51)}$$

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

$$d_{2,i+1} = d_{2,i} + y_i^2 - p(2x_i + 3) \text{ ..... (52)}$$

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 $d_{2,i} = -4p + 1$, ... (53)

if $d_{1,i} = p + 1$ then $d_{2,i} = -2p + 1$, ... (54)

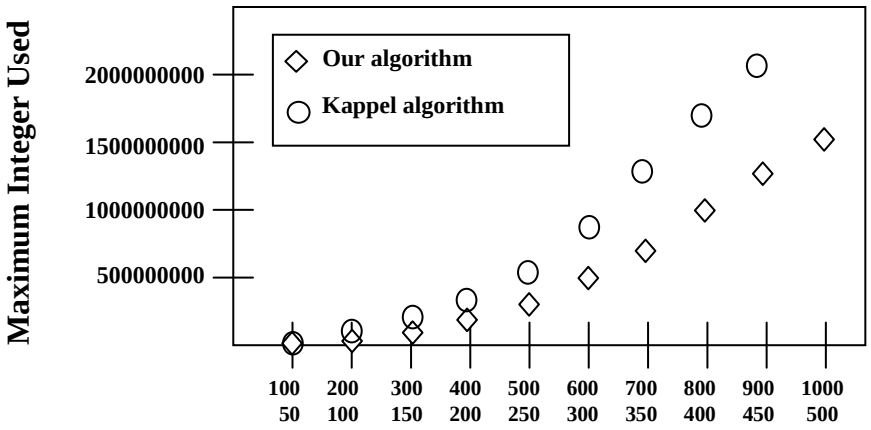
Results

The time performance of the new algorithm was compared against the algorithm described by Kappel [4], 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.

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.



Scheme 1. Maximum integer graph

References

1. D. Hearn & M. Baker, "Computer Graphics C Version", (2nd Ed.), 1997, Prentice-Hall, 97-112.
2. J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, "Computer Graphics: Principles and Practice", (2nd Ed.) , 1997, Addison-Wesley, 55-72.
3. Kappel M. R., "An Ellipse-Drawing Algorithm for Raster Displays", 1985, Springer-Verlag, Berlin, 115-135.
4. Rogers, David., "Procedural Elements for Computer Graphics", 1985, NY: McGraw-Hill Book Company, New York, 54-70.
5. Van Aken, J. R., "An efficient ellipse-drawing algorithm. CG&A", 1984, 22-45.

توليد مقاطع مخروطية باستخدام خوارزميات كفوءة

عبد العزيز سليمان خليل

قسم علوم الحاسبات، كلية علوم الحاسبات والرياضيات، جامعة الموصل، الموصل، العراق

الملخص

يقدم البحث الحالي وصفا لخوارزميات كفوءة لتوليد القطوع المخروطية ذات محاور موازية لمحور الإحداثيات، وذلك بالاعتماد على منهجية مشابهة لخوارزمية Bresenham ومحاكاة مفاهيم خوارزمية النقطة الوسطية. أظهرت نتائج التنفيذ، في حالة الشكل البيضوي، أنها تمتلك، على اقل تقدير، سرعة وكفاءة الخوارزميات الصحيحة المعروفة بالإضافة إلى كونها تتطلب مدى اقل من الأعداد الصحيحة في العمليات الحسابية وتؤدي دوما إلى انتقالات مناطقية صحيحة. كما تم في هذا البحث تصميم تقنيات كفوءة لتوليد القطع الناقص والقطع المكافئ.

Appendix A

Ellipse C++ function code

```
void ellipse1(int xc, int yc, long rx, long ry)
{
 long rx2,ry2,tworx2,twory2,x_slop,y_slop;
 long d,mida,midb;
 int x,y;
 x = 0; y = ry;
 rx2 = rx * rx;         ry2 = ry * ry;
 tworx2 = 2 * rx2;      twory2 = 2 * ry2;
 x_slop = 2 * twory2;   y_slop = 2 * tworx2 * ( y - 1 );
 mida = rx2 / 2;
 midb = ry2 / 2;
 d = twory2 - rx2 - y_slop / 2 - mida;
 while ( d <= y_slop )
  {
    draw(xc, yc, x, y);
    if ( d > 0 )
    {
     d -= y_slop;
     y--;
     y_slop -= 2 * tworx2;
    }
    d += twory2 + x_slop;
    x++;
    x_slop += 2 * twory2;
   }
  d -= ( x_slop + y_slop ) / 2 + ( ry2 - rx2 ) + ( mida -
midb );
   while( y >= 0 )
    {
    draw(xc, yc, x, y);
    if( d <= 0 )
     {
      d += x_slop;
      x++;
      x_slop += 2 * twory2;
     }
    d += tworx2 - y_slop;
    y--;
    y_slop -= 2 * tworx2;
   }
}
```

Appendix B

Hyperbola C++ function code

```
void hyperbola(int xc, int yc, long rx,long ry,int bound)
{
 long x,y,d,mida,midb;
 long tworx2,twory2,rx2,ry2;
 long x_slop,y_slop;
 x = rx;           y = 0;
 rx2 = rx * rx;    ry2 = ry * ry;
 tworx2 = 2 * rx2; twory2 = 2 * ry2;
 x_slop = 2 * twory2 * ( x + 1 );
 y_slop = 2 * tworx2;
 mida = x_slop / 2; midb = y_slop / 2;
 d= tworx2 - ry2 * ( 1 + 2 * rx ) + midb;
 while( ( d < x_slop ) && ( y<= bound ) )
  {
   draw(x,y);
   if( d >= 0 )
    {
     d -= x_slop;
     x++;
     x_slop += 2 * tworx2;
    }
   d += tworx2 + y_slop;
   y++;
   y_slop += 2 * tworx2;
  }
  d -= ( x_slop + y_slop ) / 2 + ( rx2 + ry2 ) – midb -
mida;
  if ( rx > ry )
  while( y <= bound )
   {draw(xc, yc, x, y);
    if( d <= 0 )
     {d += y_slop;
      y++;
      y += 2 * tworx2;
     }
    d -= twory2 - x_slop;
    x++;
    x_slop += 2 * twory2;
   }
 }
```

Appendix C

Parabola C++ Function code

```
void parabola(int xc, int yc, int p, int bound)
 {int x,y,d,p2,p4;
     p2 = 2 * p;       p4 = p2 * 2;
     x = 0 ; y = 0;
     d = 1 - p;
     while ( ( y < p ) && ( x <= bound ) )
        {draw(xc, yc, x, y);
         if( d >= 0 )
          { x++;
           d -= p2;
          }
         y++;
         d += 2 * y + 1;
        }
      if( d == 1 ) d = 1 - p4;
                else d = 1 - p2;
      while( x <= bound )
         {draw(xc, yc, x, y);
          if( d <= 0 )
           {y++;
              d += 4 * y;
           }
          x++;
          d -= p4;
         }
}
```