FIXED-POINT ELLIPSE DRAWING ALGORITHM[1]

Ramón Mollá, Roberto Vivó

Department of Computer Systems and Computation
Polytechnical University of Valencia, Camino de Vera, 14
46022 Valencia
Spain
e-mail: rmolla@dsic.upv.es, rvivo@dsic.upv.es

ABSTRACT

This algorithm draws ellipses with integer centres and decimal radii on discrete devices using fixed-point arithmetic. These ellipses have both X and Y axis parallel to the coordinate axes. It uses forward differences to diminish its cost. It has a low computational complexity while the error is lower than traditional algorithms. This algorithm works in the squared R^2 space (fixed-point) and translates directly the decimal points to the Z^2 natural screen space.

Keywords: Fixed-point arithmetic, ellipse-drawing, scan conversion.

1. INTRODUCTION

An ellipse is a basic graphic primitive in computer graphics. It is a common shape in many graphics applications and appears naturally when viewing circles from a lateral position or when a circle primitive has to be drawn into a non-isotropic device. This primitive is defined by the following equations

$$\frac{X^2}{B^2} + \frac{Y^2}{A^2} = 1$$

$$Y^2 = A^2\left(1-\frac{X^2}{B^2}\right)$$

$$X^2 = B^2\left(1-\frac{Y^2}{A^2}\right)$$

Where A and B, are the vertical and horizontal ellipse axes respectively parallel to the grid lines. No zero axis is allowed also, since this ellipse degenerates into a line. We are concerned with approximating an ellipse by lighting pixels on a bitmap. In order to draw this primitive incrementally, it is commonly accepted that the ellipse is divided into two areas. So all the algorithms use two internal drawing loops. One for each area. See the Figure 1.

Taking advantage of the primitive symmetry, it is normally used the 4 points algorithm in order to accelerate the drawing process. So, for clarity purposes, this paper analyses only the drawing problem for the first quarter (the right up one). The other three ones may be obtained from it obviously.

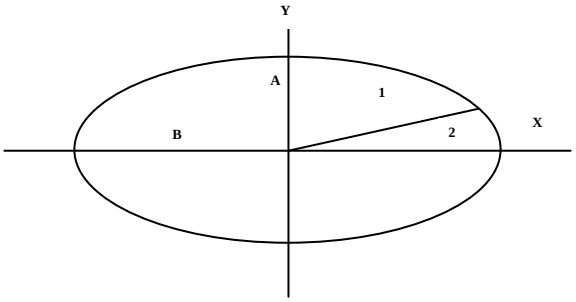

Figure 1. Parallel ellipse. "A" means the vertical ellipse axis, "B" the horizontal one and 1 and 2 represent the two areas that divide the ellipse in two drawing loops.

The major concern of this article is the time complexity of ellipse algorithms, since fast scan conversion is crucial both for real-time interaction and for animation in computer graphics, specially in low-cost or slow devices like PDAs, palmtops, microcontrollers, e-books, WAP devices, low price printers,… where there is processing power shortage and where accurated algorithms of low complexity are also of great significance. In these cases, simple algorithms are mandatory, especially for basic primitives like lines, circles or ellipses. The algorithm presented in this paper takes advantage of all the

[1] This paper has been developed partially thanks to CICYT TIC9-0510-C02-01

previously ideas introduced by the bibliography: it is an incremental algorithm based on the second order differences [Foley92], it uses an error function based on the middle point [Fellner94], it uses the 4 point symmetry and it uses fixed-point arithmetic [Fellner93]. It uses 32 and 64 bits mantissas. It works with decimal numbers radii. Although the operations works internally with decimal numbers, they use fixed-point arithmetic [Marven94], so there is no floating point penalty. The format change overhead (integer to fixed-point and vice versa) is practically avoided, so no significant penalty is noticed in practice compared to other algorithms that uses fixed-point arithmetic also.

2. PREVIOUS WORK

All the scan conversion algorithms work on the screen discrete space. This space is a quantum grid where only some given movements are allowed. This restriction forces to calculate only some points which X or Y coordinate are previously known and not all the infinite possible points. An order relationship is set between all these points since two neighbours are separated by one unit. This situation let many scan conversion algorithms to be based on incremental techniques. A ellipse is a primitive that works with squared values, many algorithms [Pitteway67] [Bresenham77] use the second order differences principle (Foley 1992) to allow easy incremental techniques. Diophantine equations are also used to improve speed-up [Andres94]. The incremental technique [Kappel85] is based on an error function adapted to ellipses that determines the decision for every loop step. The error function is based in the proximity between the primitive and the discrete screen grid. Although for circles, whatever criterion is equivalent: the function residual value, the orthogonal distance or the vertical distance [Bresenham85] [McIlroy83], it is not really true for ellipses [McIlroy92]. A better study of this problem may be seen at [Fellner94]. For these reasons, there are some approximations that can manage with extreme cases [Wu87] or on the accuracy of the algorithms [vanAken84]. Many ways have been proposed in order to increase performance, for instance, drawing several points at a time [Wu89], using partial differences [daSilva89]. This is a general algorithm for drawing conics. Although it may generate a lower cost solution, the algorithm is very complicated to develop and debug since this is a general solution. Other improvements draw this primitive as a collection of horizontal lines of different length [Hsu93]. If these lines are drawn using incremental loops, time savings may be between 25 and 40% comparing to Bresenham's if the radius is higher than 16 pixels. Another way to reduce the cost is to diminish the amount of I/O operations to draw each segment [Chengfu95]. Other approximation departs radically from the traditional approaches using the Lissahaus figures to produce not only circles but whatever kind of elliptical arcs or complete figures [Fellner93]. It requires an intermediate filter function to eliminate the amount of points accumulated at parts of increasing curvature. All the previously presented algorithms change speed-up by precision, since the parameters (decimal radius length and decimal position) are approximated to the screen grid (integer) before drawing the primitive. The later the conversion is made, the more precise the drawing is. The FPE algorithm presented in this paper maintains the decimal numbers even during the calculation phase. It only approximate the values to the grid when the values are sent to the raster, at the very last moment. So the representation error is the lowest possible as it will be seen in the following points.

3. BACKGROUND

Initially if $X_0 = 0$, then

$$Y_0^2 = A^2\left(1-\frac{X_0^2}{B^2}\right)=A^2\left(1-\frac{0^2}{B^2}\right)=A^2$$

, that is to say, $Y_0 = A$.

Since at the initial point, the tangent is null, lower than one, the drawing sweep has to start through the X axis. Following an incremental approach, $X_1 = X_0 + 1$, and so

$$Y_1^2 = A^2\left(1-\frac{X_1^2}{B^2}\right)=A^2\left(1-\frac{(X_0+1)^2}{B^2}\right)=A^2\left(1-\frac{(X_0^2+2X_0+1)}{B^2}\right)$$

$$Y_1^2 = A^2\left(1-\frac{X_0^2}{B^2}-\frac{(2X_0+1)}{B^2}\right)= Y_0^2 - A^2\frac{(2X_0+1)}{B^2}$$

Let K be the constant element in the last equation,

$K = A^2\frac{(2X_0+1)}{B^2}$, and so $Y_1^2 = Y_0^2 - K$

In the next step,

$$Y_2^2 = A^2\left(1-\frac{X_2^2}{B^2}\right)=A^2\left(1-\frac{(X_1+1)^2}{B^2}\right)=A^2\left(1-\frac{(X_1^2+2X_1+1)}{B^2}\right)$$

$$Y_2^2=A^2\left(1-\frac{X_1^2}{B^2}-\frac{(2X_1+1)}{B^2}\right)=Y_1^2-A^2\frac{(2X_0+1+2)}{B^2}=Y_1^2-K-A^2\frac{2}{B^2}$$

Let M be the new constant element appeared, where $M = 2A^2/B^2$, and $Y_2^2 = Y_1^2 - K - M$

On the next iteration,

$$Y_3^2 = A^2\left(1-\frac{X_3^2}{B^2}\right)=A^2\left(1-\frac{(X_2+1)^2}{B^2}\right)=A^2\left(1-\frac{(X_2^2+2X_2+1)}{B^2}\right)$$

$$Y_3^2 = A^2\left(1-\frac{X_2^2}{B^2}-\frac{(2X_2+1)}{B^2}\right) = Y_2^2 - A^2\frac{(2X_0+1+4)}{B^2} = Y_2^2 - K - A^2\frac{4}{B^2} = Y_2^2 - K - 2M$$

Generically, it may be affirmed that $Y_N^2 = Y_{N-1}^2 - K - (N-1)M; \quad \forall N > 0$

If $K_N' = K - (N-1)*M; \quad \forall N > 0$, in a incremental algorithm, $Y_N^2 = Y_{N-1}^2 - K_N'; \quad \forall N > 0$, where $K'_N = K'_{N-1} + M; \quad \forall N > 0$ being $K'_0 = \frac{A^2}{B^2} = M/2$ and $M = \frac{2A^2}{B^2}$

Similarly, when drawing the second area, $P_0 = (X_0,Y_0) = (B,0)$. $X_N^2 = X_{N-1}^2 - K_N'; \quad \forall N > 0$, where $K'_N = K'_{N-1} + M; \quad \forall N > 0$ being $K'_0 = \frac{B^2}{A^2} = M/2$ and $M = \frac{2B^2}{A^2}$

That is to say, the squared coordinates of all the ellipse points may be obtained in a incremental way using second order differences. The objective is to obtain the pixel integer coordinate from its squared value. So the algorithm works with the decimal point squared position P_i^2 and the middle point squared decimal position P_m^2. As soon as $P_i^2 < P_m^2$, the next point to light on the screen will be the just below, that is to say P(X,Y) = (X,Y-1).

That is to say, the squared coordinates can be obtained from the previous values in a incremental fashion. In the practice, the previous equations work on the squared R^2 space. The problem now is how to deduce the integer value on the screen from its corresponding squared real number. Given a discrete drawing grid like the one that appears on the next illustration, there is a primitive segment that pass through a pixel which coordinates are $P_s(X,Y)$.

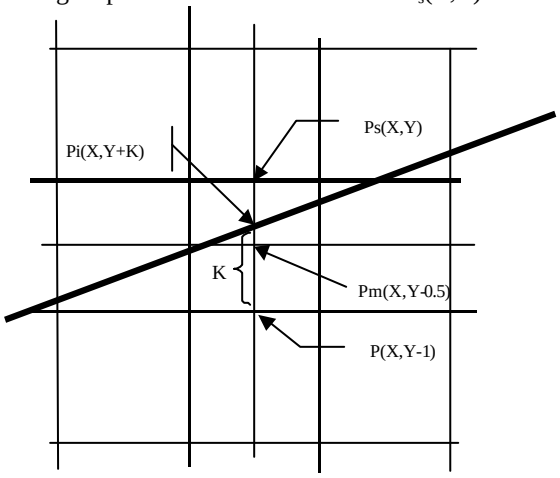

Figure 2. Decimal line approximation on a discrete grid

This primitive intersects on the vertical axis of the pixel at the coordinate $P_i(X, Y+K)$. If K >= 0.5, then the cutting point is nearer the upper point $P_s(X,Y)$, rather than to the lower one P(X,Y-1), and vice-versa. That is, the pixel P_s is lighted if and only if $P_m <= P_i <= P_s$; or Y-0.5 <= Y+K <= Y, or -0.5 <= K <= 0

On the other hand, P is lighted only when $P <= P_i <= P_m$; that is Y-1 <= Y+K <= Y-0.5, or what is the same -1 <= K <= -0.5. If these inequations are raised to the squared, P_s will be lighted if and only if $P_m^2 <= P_i^2 <= P_s^2$; that is $(Y-0.5)^2 <= P_i^2 <= Y^2$. On the contrary, P will be lighted when $P^2 <= P_i^2 <= P_m^2$; that is $(Y-1)^2 <= P_i^2 <= (Y-0.5)^2$. Starting the algorithm loop from the ellipse intersection with the vertical axis, $P_0(X,Y) = (0,A)$, then $Pm_0^2 = Y^2 - Y + 0.25 = A^2 - A + 0.25$

While the ellipse points P_i are being drawn, the squared Y coordinate of the calculated points will get closer to P_m^2. For all those points, their screen Y coordinate will have always the same value: A. As soon as the condition $P_i^2 < P_m^2$ is met, the point to draw will be just the lower one that is P(X,Y) = (X,A-1). Going on the same X loop, the points Y coordinate will get lower, so when $P_i^2 < Pm_1^2$, the coordinate A-2 will be lighted. So,

$Pm_1^2 = (Y - 1.5)^2 = Y^2 - 3Y + 2.25 = A^2 - A + 0.25 - 2(A - 1) = Pm_0^2 - 2(A - 1)$, where whatever point to draw will meet the condition $P_1 = (X,A-1)$, if and only if $Pi_1^2 >= Pm_1^2$. Generalising, it may be affirmed that $Pm_N^2 = (Y - 0.5 - N)^2 = Y^2 - (2N+1)Y + N^2 + N + 0.25$. In a similar way, $Pm^2_{N-1} = (Y - 0.5 - N + 1)^2 = (Y - (N - 0.5))^2 = Y^2 - (2N-1)Y + N^2 - N + 0.25$. So, $Pm_N^2 = Pm^2_{N-1} - 2Y + 2N = Pm^2_{N-1} - 2(Y - N) = Pm^2_{N-1} - 2(A - N)$, where all those screen points will meet the condition $P_N = (X_N,Y_N) = (X_N,A-N)$, if and only if $Pi_N^2 >= Pm_N^2$. That is, $Pm_N^2 = Pm^2_{N-1} - 2\,Y_N$.

4. IMPLEMENTATION

The algorithm code is provided in C language. The algorithm requires to calculate internally a squared value. Since the algorithm parameters are 32 bits Q15 fixed point arithmetic numbers [Marven94], their squared values can reach to 2^{64}. So these amounts cannot be supported by 32 bits fixed point numbers. That is the reason why 64 bits mantissas were used. So, the algorithm support ellipses with radii up to 2^{15} pixels, that is, enough to draw primitives on A0 papers using 600 ppi resolutions. These formats may be understood also as two signed and unsigned chained numbers with half the length (16 or 32 bits). The implementation is showed at the appendix A.

The drawing functions used in the algorithm are based on the 4 points algorithm in order to take advantage from the ellipse symmetry in a very similar way as the circle algorithm does with the eight points algorithm. In this way, the algorithm uses a 2 point array to draw a primitive in whatever screen position with the same computational cost. The increment routines only touches two points also as seen in the Appendix A.

INITIALISATION

The algorithm performs in the first stage the squared radii calculation and some format conversions from integer to fixed point formats. The next operations try to calculate the squared difference between the initial decimal height and the nearest middle point. The incremental constants K and M are also calculated. They are the second order differences and they will be used later in the loop phase. It is also very important to calculate the break point where change the swept direction. That is, where the first loop (X swept) is finished and the second one (Y swept) starts. The four points are initialised and drawn.

X AXIS SWEPT

The X loop starts at $P_0 = (X_0,Y_0) = (0,A)$. This is the area one as seen in Figure 1. While the primitive tangent is higher than -1, X is incremented in one unit, the gradient is recalculated and the squared of this coordinate. If this squared value is lower than the squared decimal low limit, then the coordinate is decreased in one unit, the gradient is recalculated again and the new squared low limit. Using the four point symmetry, four points are drawn.

Y AXIS INITIALISATION

In a first place the second order differences are calculated. It is also calculated the squared difference between the initial point and the nearest middle point. The break point reached in the first loop is now the one used to determine the end of this one. The four points are initialised again and drawn.

Y AXIS SWEPT

The Y loop starts at $P_0 = (X_0,Y_0) = (B,0)$. This is the area two as seen in Figure 1. The last point Y coordinate obtained in the previous loop determines the last point height in this second loop. The same K and M constants are calculated symmetrically for this loop. The four starting points are initialised and drawn. In the same way as the first loop, while the point Y coordinate is lower than the last point Y coordinate, it is increased in one unit. The X^2 is recalculated incrementally. When X^2 surpasses $(X - 0.5)^2$, X is decreased one unit and $(X - 1.5)^2$ is recalculated. For every step, four points are drawn.

5. COMPUTATIONAL COST

The aim in this point is to compare the cost given by this fixed point based algorithm to a paradigm like the Middle Point (MP) algorithm [Foley92]. In order to avoid original the MP implementation penalties, an optimised incremental second order differences version was used. This version used temporal variables to avoid redundant calculations and fixed-point arithmetic in order to diminish even more the computational cost and to allow the best possible comparison, since the original algorithm implementation is clearly worse. Both algorithms has two symmetrical loops with a similar computational cost. The point of change is reached when

$$Y = K_y = \frac{A^2}{\sqrt{B^2 + A^2}} \quad \text{and} \quad X = K_x = \frac{B^2}{\sqrt{B^2 + A^2}}$$

that is to say, the first loop is performed Kx iterations and the second one Ky. Adding both constants,

$$K_y + K_x = \frac{A^2}{\sqrt{B^2 + A^2}} + \frac{B^2}{\sqrt{B^2 + A^2}} = \frac{A^2 + B^2}{\sqrt{B^2 + A^2}}$$

$$K_y + K_x = \frac{\left(\sqrt{B^2 + A^2}\right) * \left(\sqrt{B^2 + A^2}\right)}{\sqrt{B^2 + A^2}} = \sqrt{B^2 + A^2} = C$$

This means that the ellipse algorithms always draw as many points as the hypotenuse that links the axis ends. The total initialisation cost (both loops) for the FPE, MP and McIlroy algorithms is

Operation	**FPE**	**MP**	**McIlroy**
Division	3	0	0
Product	5	12	5
Add/Subs Inc/Dec Comp	12	10	13
Binary shift	54	7	10

Without taking into account the pixel operations that are common for all the algorithms and adding both main loops, the exact FPE algorithm cost is showed in the next table. All the operations use integer arithmetic

Opertion	**1st loop**	**2nd loop**	**Total**
Comp.	2Kx	2Ky	2C
Add/Sub	3Kx+A-Ky	2Ky+B-Kx	C+A+B+Kx
Inc/Dec	2(A-Ky)	B-Kx +Ky	2A+B-C

The next table shows the computational cost comparison for the three algorithms

Opertion	**FPE**	**McIlroy**	**MP**
Add/Sub	C+A+B+Kx	10C+A	Kx+3A-Ky+2B
Comp	2C	6C+A	2C
Logical		3C	
Product		2C+A	
Inc	2A+B-C		A+B

Notice that if the fixed-point arithmetic should be hardware supported, almost all binary shifts could be avoided, reducing even more the cost for the FPE algorithm. In this situation, McIlroy's algorithm has the fastest initialisation phase, followed by the FPE and finally the MP. Nevertheless, these costs are very similar. Notice that we have avoided the original floating point arithmetic implementation of the MP algorithm and we have improved it using second order differences. Even though, the loop cost is clearly better for the FPE algorithm.

ERROR ANALYSIS

This point compares the brute force algorithm based on floating-point arithmetic to the MP [Foley92] and FPE algorithms. It has been chosen the MP algorithm because it is well know by the scientific community and it is representative (error and performance) of most traditional algorithms. The test drew all possible ellipses with X and Y radius in the range [1,1023] using a double nested loop from 1 pixel to 1023. The amount of primitives drawn was over half a million. The brute force algorithm calculated the points giving a floating-point result (without rounding). There were taken into account integer radii and centres or decimal radii and integer centres. This comparison took into account the vertical difference (height) between the value provided by the brute force algorithm and the ones given by the algorithms analysed. When comparing the rounded integer results when using integer radii and centres, all the algorithms provided practically the same values. No visual difference was detected. The absolute error value provided by both algorithms can be seen in Figure 3 compared to the radii aspect ratio and compared to the number of pixel drawn in a quadrant in Figure 5. The absolute difference between both algorithms can be seen in Figure 4 and in Figure 6.

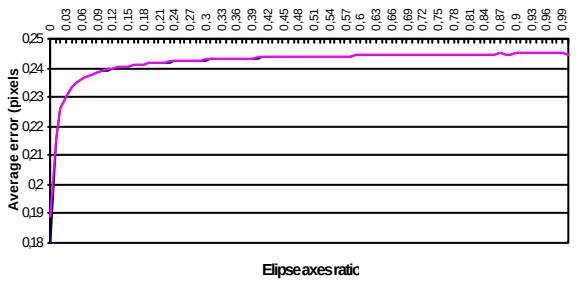

Figure 3. Average error distribution depending on the radii aspect ratio

The absolute average error presented by both algorithms was 0.2420 pixels. The FPE increased the error scarcely 0,00015 units. The error difference is normally between one thousandth and one ten thousandth for almost all the studied cases.

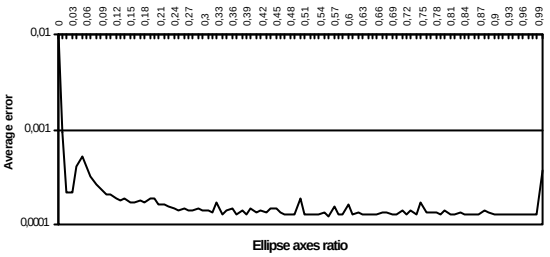

Figure 4. Average error difference distributed by axis ratio

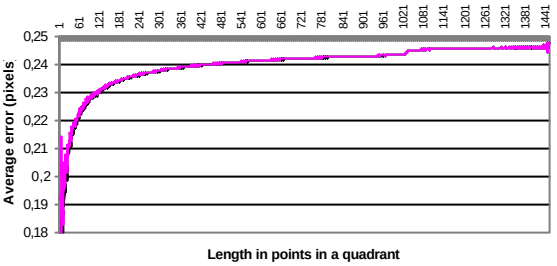

Figure 5. Average error distribution depending on perimeter length of a ellipse quadrant

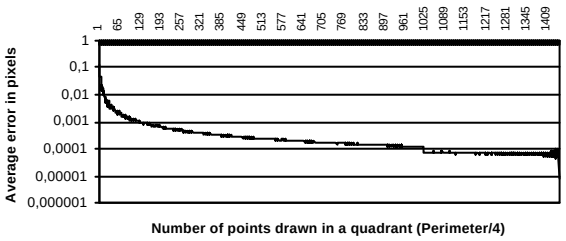

Figure 6. Average error difference distribution depending on perimeter length of a ellipse quadrant

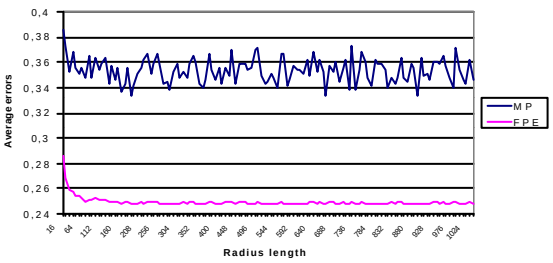

Figure 7. Average error when compared to the brute force algorithm with decimal radii and integer centres

Although the error differences increase for very flat ellipses or very small ones, the absolute error diminishes in these cases.The average difference between the integer algorithms, including the FPC, and the brute force one, is always under 0.25 pixels. This means that the maximum difference between the geometrical centre of a pixel and the real point was never higher than 0.5 pixels. So, the algorithms provide a very good approximation. This behaviour is similar to the one presented by the FDDA [Molla92] for line drawing. When the FPE algorithm takes into account the decimal radii, the average error provided by traditional algorithms increase up to 0.35 while the FPE remains on 0.25. On average, traditional integer algorithms for drawing ellipses and conics in general, have accuracy error around 45% higher than the FPE algorithm since they use integer radii that lose the decimal part. It may be proved empirically that:

- Independently from the error distribution (length or axes ratio), the error average for the considered algorithms is generally around 0.25 pixels.
- Only for very small radii or exaggerated radii ratios the error goes down significantly.
- The error difference between both algorithms is, in general, between a thousandth and a ten thousandth of pixel for the majority of the studied cases.

6. CONCLUSION

In this paper we have introduced a new algorithm for ellipse drawing, an algorithm based on fixed point arithmetic. We can affirm that

- The drawing loops are independent. So, the drawing can be performed in parallel from the extremes towards the break points.
- Additionally, it uses no floating-point operations while still supporting decimal numbers.
- The cost difference is always favourable to the FPE algorithm.
- The errors incurred by this algorithm are small enough to allow its use in a lot of applications from digital laser controllers to graphic coprocessors.

The fixed-point arithmetic may be applied to whatever kind of problem where the variation range of the input/output variables and the intermediate results is very limited. These problems need a very fast arithmetic although not a high precision. So computer graphics is the candidate for this kind of arithmetic. This arithmetic may be applied to line-drawing algorithms [Molla92], circle-drawing algorithms, ellipses, clipping, ray-tracing, simulation, scene description languages and so on.

The application of this arithmetic provides fast algorithms. They require low resources like registers, silicon surface on graphics coprocessors, lower operators' complexity, etc. In many cases, they allow hardware/software parallelisation [Molla93] and they can use also the speed-up techniques used for other algorithms. The algorithm presented in this paper draw ellipses with integer centres and decimal radii using the four points algorithm. It can be easily upgraded to support decimal centres. In this case there is only a two points symmetry. Nevertheless, using a hardware operator, all these calculations may be done in parallel, reducing the temporal cost to the one provided by the algorithm that uses the eight points algorithm. The accuracy provided by this algorithm is worth this handicap.

7. REFERENCES

[Andres94] Andres E (1994) Discrete Circles, Rings And Spheres. C&G 18(5):695-706

[Bresenham77] Bresenham, J.E., "A linear algorithm for incremental digital display of circular arcs", Communications of the ACM, Vol. 20, No. 2, Pp.100-106, Febrero 1977

[Bresenham85] Bresenham , Jack E (1985) Algorithm for circle arc generation, Fundamental Algorithms for Computer Graphics, R.A. Earnshaw, ed., NATO ASI Series, Vol. F17, Springer Verlag, Berlin, Pp. 197-218

[Chengfu95] Chengfu Y, Rokne JG (1995) Hybrid Scan-Conversion Of Circles. IEEE Trans V&CG, 1(4):311-318, ISSN: 1077-2626

[daSilva89] Da Silva, D., "Raster algorithms for 2D Primitives", Master´s Thesis, Computer Science Department, Brown University, Providence, RI, 1989.

[Fellner93] Fellner DW, Helmberg C (1993) Robust rendering of general ellipses and elliptical arcs, ACM Transactions on Graphics.12(3):251-276

[Fellner94] Fellner DW, Helmberg C (1994) Best Approximate General Ellipses on Integer Grids, C&G 18(2):143-151

[Foley92] Foley JD, van Dam A, Feiner SK, Hughes JF (1992) Computer Graphics, Principles And Practice. Addison-Wesley

[Hsu93] Hsu SY, Chow LR, Liu HC (1993) A New Approach For The Generation Of Circles. CGF 12(2):105-109

[Kappel85] Kappel, Michael R., "An Ellipse-Drawing Algorithm for Raster Displays", Earnshaw, R., ed. Fundamental Algorithms for Computer Graphics, NATO ASI Series, Springer-Verlag, Berlin, Vol. F17, Pp. 257-280, 1985

[Marven94] Marven, C and Ewers G (1994) a simple approach to Digital Signal Processing, Texas Instruments, Pag. 197-207, ISBN: 0-904-047-00-8

[McIlroy83] McIlroy, MD (1983) Best Approximate Circles on Integer Grids, ACM Trans. on Graph., Vol. 2, No. 4, Pp. 237-263

[McIlroy92] McIlroy M. Douglas (1992) Getting raster Ellipses Right. ACM Trans. On Graph. 11(3):259-275

[Molla92] Mollá R., Quirós R., Vivó R. "Fixed-point Digital Differential Analyser" Proceedings of Compugraphics 92. Pag. 1-5, 14-17, Dec. 1992.

[Molla93] Mollá R., Vivó R. “Parallel Fixed-point Digital Differential Analyser with Antialiasing”, Parallel and Distributed Computing Practices,

[Pitteway67] Pitteway ML.V (1967) Algorithm for Drawing Ellipses or Hiperbolae with a Digital Plotter, Computer Journal, 10(3):282–289

[vanAken84] Van Aken, J.R., "An efficient Ellipse-Drawing Algorithm.", IEEE Comput. Graph. & Appl., Vol. 4, No. 9, Pp. 24-35, Sept. 1984

[Wu87] Wu X, Rokne JG (1987) Double-step Incremental Generation of Lines and Circles, Comp. Graphics & Image Procc. 37(4):331-344

[Wu89] Wu X, Rokne JG (1989) Double-step Generation of Ellipses, IEEE CG&A, May:56-69

[Skala94a] Skala,V.: O(lg N) Line Clipping Algorithm in E^2, *Computers & Graphics*, *Pergamon Press*, Vol.18, No.4, pp.517-524, 1994.

8. APPENDIX A

Some definitions

```
#define DecrX4Points()  {P[0].X --;P[1].X ++;}
#define IncrY4Points()  {P[0].Y ++;P[1].Y --;}
#define IncrX4Points()  {P[0].X ++; P[1].X --;}
#define DecrY4Points()  {P[0].Y --;P[1].Y ++;}
```

Ellipse algorithm source code. Data types.

```
typedef union {
        __int64 V;      // 64 bits integer
        struct {
                unsigned long   Dec;
                long            Int;
        };
        struct {
                short int       DecLow;
                long Q15; //FPQ15
                short int       IntHigh;
        };
    }FPQ31;
    union {
        long    V;      //32 bits integer
        struct {
                unsigned short int Dec;
                short int Int;
                };
    } FPQ15 ;
```

The main algorithm for drawing ellipses is this following one:

```
Elipse1516 (int CX, int CY, FPQ15 A, FPQ15 B)
{
FPQ31   A2fp3132,       //A*A
        B2fp3132,       //B*B
        A2mB2,          //A*A + B*B
        Aux;
FPQ15   Dif2, K, M, Rint,
        Xx2,    //2X
        Yx2,    //2Y
__int64 A2fp4716, B2fp4716, DifH24716; //64b
int     Y;

B2fp3132.V  = B.V;
B2fp3132.V *= B.V;
B2fp4716= B2fp3132.V  >> 16;   //16 lsb not signf.

A2fp3132.V  = A.V;
A2fp3132.V *= A.V;
A2fp4716= A2fp3132.V  >> 16;
Rint = A.Round(); //The closest integer radius
Aux.V = Rint.V - 16384;   //int(A) – 0.5 fixed point
Aux.V *= Aux.V;
Aux.V -= A2fp3132.V;
Dif2.V = Aux.Q15;
M.V = (A2fp3132.V << 1) / B2fp4716;
K.V = (M.V + 1 ) >> 1;
Aux.V = A2fp3132.V  / (A2fp4716 + B2fp4716);
Aux.V *= A2fp4716;
Aux.V = A2fp3132.V - Aux.V;
DifH24716 = Aux.V >> 16;
Y = Rint.Int;
Yx2.V = Rint.V << 1;
Init4Points (CX, CY, 0, Y);
while (DifH24716 >= 0)  //First swept on the X axis
{
        IncX4Points();
        DifH24716 -= K.V;
        Dif2.V += K.V;
        K.V  += M.V;
        if ((0 < Dif2.V) && Y)
        {
                Y--;
                DecY4Points();
                Yx2.V -=131072L;
                Dif2.V-= Yx2.V;
        }
        Draw4Points ();
}
M.V = (B2fp3132.V << 1) / A2fp4716; //2nd swept
K.V = (M.V + 1 ) >> 1;
Rint = B.Round();
Aux.V = Rint.V - 16384;   //int(B) – 0.5 fixed point
Aux.V *= Aux.V;
Aux.V -= B2fp3132.V;
Dif2.V = Aux.Q15;
Xx2.V = Rint.V << 1;
Init4Points (CX, CY, Rint.Int, 0);
while (--Y >= 0)
{
        IncY4Points();
        Dif2.V += K.V;
        K.V  += M.V;
        if (0 < Dif2.V)
        {
                DecX4Points();
                Xx2.V -= 131072L;
                Dif2.V -= Xx2.V;
        };
        Draw4Points ();
}
}
```