

# Polynomial Evaluation using Affine Arithmetic for Curve Drawing

Qijiang Zhang      Ralph R. Martin

## Abstract

Affine arithmetic (AA) is a more sophisticated version of interval arithmetic (IA) which can provide more accurate and tighter interval results than IA. A new affine arithmetic technique is described for polynomial evaluation. A program was developed to draw algebraic curves in the form  $f(x, y) = 0$  in a specified rectangular interval, using both affine arithmetic and interval arithmetic. Tests show that curves drawn with AA are of higher quality and need less divisions than curves drawn with IA.

## 1 Introduction

Solving  $f(x, y) = 0$  in a rectangular interval  $[x_a, x_b] \times [y_a, y_b]$  on the  $x - y$  plane is a problem with many practical applications in computer graphics and CAD. One such example is drawing the algebraic curve represented by  $f(x, y) = 0$ ; other applications include surface-surface intersection [1, 2, 3, 4, 5, 6]. Given a rectangular interval  $[x_a, x_b] \times [y_a, y_b]$  on the  $x - y$  plane, and a curve in implicit polynomial form  $f(x, y) = 0$ , then the existence of the curve in this specified region can be tested using interval arithmetic. This can be combined with recursive subdivision of the interval as a method of drawing the curve.

Let  $\bar{x}$  and  $\bar{y}$  be intervals in the  $x$  and  $y$  directions,  $\bar{x} = [x_a, x_b]$ ,  $\bar{y} = [y_a, y_b]$ , and let the function  $f(x, y)$  be given. We can then compute  $f(\bar{x}, \bar{y})$ . The value of  $f(\bar{x}, \bar{y})$  is also an interval which can be evaluated using interval arithmetic. If the value of  $f(\bar{x}, \bar{y})$  in the region  $[x_a, x_b] \times [y_a, y_b]$  does not contain zero, we can say the curve definitely does not pass through that region. Otherwise, we can only say the curve *may* be present. The region may then be divided into sub-regions for further evaluation until pixel level is reached. A pixel is treated as a unit region. If at pixel level, the value of  $f(\bar{x}, \bar{y})$  contains 0, then the pixel is plotted. Any sub-region in which the value of  $f(\bar{x}, \bar{y})$  does not contain 0 is discarded.

A problem arises with this approach. As we will see in later sections, standard interval arithmetic often produces a ‘thick’ curve as the result, as even if the interval corresponding to a pixel contains zero, this does not guarantee that the curve passes through the pixel.

A second problem is that the interval for  $f(\bar{x}, \bar{y})$  returned by interval arithmetic may be much wider than the true interval in which  $f(x, y)$  lies. This may cause the curve drawing method to unnecessarily subdivide intervals which will ultimately be found to be empty.

The development of affine arithmetic [7] provides an alternative to interval arithmetic. We discuss how to use affine arithmetic to evaluate bivariate polynomials in this paper, and show examples which demonstrate that our method when used in the same curve drawing strategy (a) requires less subdivision, and (b) produces ‘thinner’ curves.

## 2 Interval arithmetic

Interval arithmetic (IA) is a technique for numerical analysis and computation [8, 9]. Since the 1980s, IA has been used in computer graphics applications such as ray tracing, solid modelling and so on [1, 2, 3, 4, 5].

An interval is defined as a closed bounded set of ordered ‘real numbers’. If  $x.lo$  and  $x.hi$  are real numbers with  $x.lo \leq x.hi$ , then  $\bar{x}$  is an interval,  $\bar{x} = [x.lo, x.hi]$ . A real number  $x = a$  can be considered to be a degenerate interval  $\bar{x} = [a, a]$ .

In IA, each quantity is represented by an interval of real numbers and computations are always performed on intervals rather than on single real numbers. Standard IA always assumes that the uncertainty in value of a variable may vary in its given interval independently of uncertainty in other variables, and IA operations work based on this assumption.

For any real number operation  $f(x, y, \dots)$ , there is a corresponding interval operation  $\bar{f}(\bar{x}, \bar{y}, \dots)$ . IA operations (+, −, ×, ÷, power, root, etc) are performed in such a way that each computed interval result must contain all the possible result values corresponding to all the possible original values in those intervals, and the range of the result should be as small as possible.

Thus, some basic IA operations are defined as below:

$$\bar{x} + \bar{y} = [x.lo + y.lo, x.hi + y.hi] \quad (1)$$

$$\bar{x} - \bar{y} = [x.lo - y.hi, x.hi - y.lo] \quad (2)$$

$$\bar{x} \times \bar{y} = [\min(x.lo \times y.lo, x.lo \times y.hi, x.hi \times y.lo, x.hi \times y.hi), \max(x.lo \times y.lo, x.lo \times y.hi, x.hi \times y.lo, x.hi \times y.hi)] \quad (3)$$

Powers can be generally computed as multiplication of the same variable, with attention paid to even powers as a special case. For example, given  $\bar{x} = [-2, 2]$ ,  $\bar{x}^2$  should give a result  $[0, 4]$  rather than  $[-4, 4]$ .

IA’s weakness is its over-conservatism [7, 10, 11]. The intervals IA produces are often much wider than the true range of the corresponding quantities. For example, given  $\bar{x} = [a, b]$ , the operation  $\bar{x} - \bar{x}$  gives the result  $[a - b, b - a]$  rather than  $[0, 0]$ .

Generally, in IA, the relative accuracy of the computed intervals decreases as an exponential rate. This is called the ‘error explosion’ problem and has attracted the attention of various researchers.

### 3 Affine arithmetic

Affine arithmetic (AA) was proposed by Stolfi and others in the early 1990s with the aim of tackling the ‘error explosion’ problem caused by standard IA [7]. Like IA, affine arithmetic can be used to manipulate imprecise values and to evaluate functions over intervals. While like IA, it provides guaranteed bounds for computed results, AA takes into consideration the correlations or dependencies between the sources of error. In this way it is able to produce much tighter and more accurate intervals than IA, especially in long chains of computations.

AA has been used as a replacement for IA in various computer graphics applications, such as ray tracing, intersection testing, enumeration of implicit surfaces, and sampling for procedural shaders [7, 10, 11, 12]. It is its tighter intervals of computed results that makes it possible to draw algebraic curves more efficiently and with higher quality, as we will show later.

In AA, an uncertain quantity  $x$  is represented by an affine form  $\hat{x}$  that is a first-degree polynomial of a set of noise symbols  $\varepsilon_i$ .

$$\hat{x} = x_0 + x_1\varepsilon_1 + \dots + x_n\varepsilon_n = x_0 + \sum_{i=1}^n x_i\varepsilon_i \quad (4)$$

Here the values of noise symbols  $\varepsilon_i$  are unknown but assumed to be in the interval  $[-1, +1]$ . The corresponding coefficient  $x_i$  is a real number that determines the magnitude and sign of  $\varepsilon_i$ . Each  $\varepsilon_i$  stands for an independent source of error or uncertainty that contributes to the total uncertainty in the quantity  $x$ . The source of error may be input data uncertainty, formula truncation errors, arithmetic rounding errors, etc. If the same noise symbol  $\varepsilon_i$  appears in two or more affine forms, e.g.  $\varepsilon_1$  appears in both  $\hat{x}$  and  $\hat{y}$ , it indicates that some dependencies and correlations exist between the underlying quantities  $x$  and  $y$ .

Let  $z$  be a function of variables  $x$  and  $y$ ,  $z(x, y)$ . Computing with affine forms is a matter of replacing each operation  $z(x, y)$  with an adequate operation on affine forms. This operation must take into account the relationships between the noise symbols in  $x$  and  $y$ .

General rules on polynomial operation can be applied to affine forms. Some simple operations are as below:

$$\hat{x} \pm \hat{y} = (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \dots + (x_n \pm y_n)\varepsilon_n \quad (5)$$

$$\alpha \pm \hat{x} = (\alpha \pm x_0) + x_1\varepsilon_1 + \dots + x_n\varepsilon_n \quad (6)$$

$$\alpha \times \hat{x} = (\alpha x_0) + (\alpha x_1)\varepsilon_1 + \dots + (\alpha x_n)\varepsilon_n \quad (7)$$

From Equations (5),(6) and (7), it can be seen that, in AA, we have  $\hat{x} - \hat{x} = 0$  and  $(2\hat{x} + \hat{y}) - \hat{x} = \hat{x} + \hat{y}$ . AA's more precise results and tighter intervals come from taking into account the source of uncertainty.

Multiplication of two affine forms  $\hat{x} \cdot \hat{y}$  will produce a quadratic polynomial on the noise symbols  $\varepsilon_i$ :

$$\hat{x} \cdot \hat{y} = (x_0 + \sum_{i=1}^n x_i \varepsilon_i) \cdot (y_0 + \sum_{i=1}^n y_i \varepsilon_i) \quad (8)$$

Stolfi et al. have suggested the best affine approximation method for multiplication of affine forms [7, 10, 11]. Expanding (8), we get

$$\begin{aligned} \hat{x} \cdot \hat{y} &= x_0 y_0 + \sum_{i=1}^n (x_0 y_i + y_0 x_i) \varepsilon_i + \left( \sum_{i=1}^n x_i \varepsilon_i \right) \cdot \left( \sum_{i=1}^n y_i \varepsilon_i \right) \\ &= A(\varepsilon_1, \dots, \varepsilon_n) + Q(\varepsilon_1, \dots, \varepsilon_n) \end{aligned} \quad (9)$$

where  $A(\varepsilon_1, \dots, \varepsilon_n)$  is the first two terms in (9) which is an affine form of first-degree in each  $\varepsilon_i$ , while  $Q(\varepsilon_1, \dots, \varepsilon_n)$  is the last term in (9) which is quadratic in the  $\varepsilon_i$ . The quadratic term is handled in such a way that it will produce another affine form containing a new noise symbol  $\varepsilon_k$  with coefficient  $\mu\nu$ . Let

$$\mu = \sum_{i=1}^n |x_i|, \quad \nu = \sum_{i=1}^n |y_i|$$

Thus  $\hat{x} \cdot \hat{y}$  can be expressed as an affine combination of first-degree polynomial on  $\varepsilon_i$  plus new noise symbol  $\varepsilon_k$  whose value is still between  $[-1, 1]$ :

$$\hat{x} \cdot \hat{y} = x_0 y_0 + (x_0 y_1 + x_1 y_0) \varepsilon_1 + \dots + (x_0 y_n + x_n y_0) \varepsilon_n + \mu \nu \varepsilon_k \quad (10)$$

Using this approximation may produce a resulting interval up to twice as wide as the exact range of the quadratic term.

Conversions between affine forms and intervals are defined as below [7, 10, 11]:

—Given an ordinary interval  $\bar{x} = [a, b]$  that represents a quantity  $x$ , then its affine form can be written as  $\hat{x} = x_0 + x_k \varepsilon_k$  with

$$x_0 = \frac{(a+b)}{2}, \quad x_k = \frac{(b-a)}{2} \quad (11)$$

—Given an affine form  $\hat{x} = x_0 + x_1 \varepsilon_1 + \dots + x_n \varepsilon_n$ , then the range of possible values of the corresponding interval  $\bar{x}$  is

$$\bar{x} = [x_0 - \xi, x_0 + \xi], \quad \xi = \sum_{i=1}^n |x_i| \quad (12)$$

## 4 Handling multiplication, power operation and conversions

In our application of algebraic curve drawing, we wish to evaluate  $f(x, y) = 0$  over intervals in  $x$  and  $y$ , where  $f$  is a polynomial of two variables. This will be done using both IA and AA later for comparison purposes. We will now consider the AA version of the problem in more detail.

When represented in AA, each variable  $x$  and  $y$  will have a representation containing just one noise symbol. We wish to compute terms of the form  $x^p y^q$  and add them. This could be done directly using general affine arithmetic multiplication and addition using Equations (5) and (10), but better results can be obtained by taking more care, as we now show in the rest of this section. This method is not explicitly given in the existing literature on affine arithmetic.

To evaluate a bivariate polynomial over two intervals, we first convert it to affine form, then compute the powers, next multiply the variables in each term, and finally add the terms.

### 4.1 Conversion from interval to affine form

Given a region in which the polynomial is to be tested,  $[x_a, x_b] \times [y_a, y_b]$ , then the intervals  $\bar{x} = [x_a, x_b]$  and  $\bar{y} = [y_a, y_b]$  can be converted to

$$\hat{x} = x_0 + x_1 \varepsilon_x, \quad \hat{y} = y_0 + y_1 \varepsilon_y \quad (13)$$

where  $x_0 = (x_b + x_a)/2$ ,  $x_1 = (x_b - x_a)/2$ ,  $y_0 = (y_b + y_a)/2$  and  $y_1 = (y_b - y_a)/2$ , using equation(11).

### 4.2 Power operation

Generally, a power operation on an affine form will produce a more complicated  $p$ -degree polynomial on noise symbol  $\varepsilon_i$  where  $p$  is power index.

$$\hat{x}^p = (x_0 + \sum_{i=1}^n x_i \varepsilon_i)^p \quad (14)$$

However, in our problem, the affine form produced from conversion actually only contains a single noise symbol:  $\hat{x} = x_0 + x_1 \varepsilon_x$ . Raising this to power  $p$  will produce a  $p$ -degree polynomial of  $(p + 1)$  terms:

$$\begin{aligned} (x_0 + x_1 \varepsilon_x)^p &= x_0^p + C_p^1 x_0^{p-1} x_1 \varepsilon_x + C_p^2 x_0^{p-2} x_1^2 \varepsilon_x^2 + \dots + x_1^p \varepsilon_x^p \\ &= x_0^p + \sum_{i=1}^p C_p^i x_0^{p-i} x_1^i \varepsilon_x^i \end{aligned} \quad (15)$$

where  $C_p^i$  are binomial coefficients.

Note that, in above polynomial, when  $i$  is an odd number,  $\varepsilon_x^i$  will vary between  $[-1, 1]$ , and when  $i$  is an even number,  $\varepsilon_x^i$  will vary between  $[0, 1]$ . In other words, the noise  $\varepsilon_x^i$  in every other term varies in the same range.

Our technique is to express all  $\varepsilon_x^i (i = 1, 3, 5, \dots)$  as a new symbol  $\varepsilon_{od}$ , and add their signed coefficients together to get  $x_{od}\varepsilon_{od}$ . Similarly, we do the same for all  $\varepsilon_x^i (i = 2, 4, 6, \dots)$  to get  $x_{ev}\varepsilon_{ev}$ . Then the result of a power operation can be simplified from a  $p$ -degree polynomial of  $(p+1)$  terms to a first degree polynomial of three terms with two noise symbols,  $x_0^p + x_{od}\varepsilon_{od} + x_{ev}\varepsilon_{ev}$ .

Note that while this does not lead to the narrowest possible interval for the polynomial  $x^p$  over the range, it does provide quite tight intervals in many cases. Furthermore, the noise symbols produced are compatible with those produced in other terms of  $f(x, y)$ .

### 4.3 Multiplication

To compute each product of the form  $x^p \cdot y^q$  in  $f(x, y)$ , powers as computed above are multiplied. Thus two polynomials of three terms each are multiplied to produce a polynomial of nine terms. Let  $x_0^p + x_{od}\varepsilon_{xod} + x_{ev}\varepsilon_{xev}$  be the polynomial result after  $\hat{x}$  is raised to the power  $p$ , and  $y_0^q + y_{od}\varepsilon_{yod} + y_{ev}\varepsilon_{yev}$  be the polynomial result after  $\hat{y}$  is raised to the power  $q$ . Then we have

$$\begin{aligned}
& (x_0^p + x_{od}\varepsilon_{xod} + x_{ev}\varepsilon_{xev}) \cdot (y_0^q + y_{od}\varepsilon_{yod} + y_{ev}\varepsilon_{yev}) \\
= & x_0^p y_0^q + (x_{od} y_0^q) \varepsilon_{xod} + (x_{ev} y_0^q) \varepsilon_{xev} + (x_0^p y_{od}) \varepsilon_{yod} \\
& + (x_{od} y_{od}) \varepsilon_{xod} \varepsilon_{yod} + (x_{ev} y_{od}) \varepsilon_{xev} \varepsilon_{yod} + (x_0^p y_{ev}) \varepsilon_{yev} \\
& + (x_{od} y_{ev}) \varepsilon_{xod} \varepsilon_{yev} + (x_{ev} y_{ev}) \varepsilon_{xev} \varepsilon_{yev}
\end{aligned} \tag{16}$$

We treat  $\varepsilon_{xod}\varepsilon_{yod}$  and similar quantities in the above as independent noise quantities. After power operations and multiplication, all polynomial terms in the curve function  $f(x, y)$  will be in the same form as (16), and addition and subtraction of these terms is straightforward.

### 4.4 Conversion from polynomial form to interval

The result of polynomial evaluation will be in the same form as (16). Let this final result be  $\tilde{r}$  where,  $r_0$  is the value of the first constant term, and  $r_1, \dots, r_8$  the coefficients of the noise in each other term. Then Equation (16) can be rewritten as:

$$\begin{aligned}
\tilde{r} = & r_0 + r_1 \varepsilon_{xod} + r_2 \varepsilon_{xev} + r_3 \varepsilon_{yod} + r_4 \varepsilon_{xod} \varepsilon_{yod} \\
& + r_5 \varepsilon_{xev} \varepsilon_{yod} + r_6 \varepsilon_{yev} + r_7 \varepsilon_{xod} \varepsilon_{yev} + r_8 \varepsilon_{xev} \varepsilon_{yev}
\end{aligned} \tag{17}$$

The products  $\varepsilon_{xod}\varepsilon_{yod}$ ,  $\varepsilon_{xev}\varepsilon_{yod}$  and  $\varepsilon_{xod}\varepsilon_{yev}$  will still vary between  $[-1, 1]$  as does a single  $\varepsilon_{xod}$  or  $\varepsilon_{yod}$ , and the product  $\varepsilon_{xev}\varepsilon_{yev}$  will vary between  $[0, 1]$  as does a single  $\varepsilon_{xev}$  or  $\varepsilon_{yev}$ . Thus Equation (17) becomes

$$\begin{aligned}\bar{r} = & r_0 + r_1[-1, 1] + r_2[0, 1] + r_3[-1, 1] + r_4[-1, 1] \\ & + r_5[-1, 1] + r_6[0, 1] + r_7[-1, 1] + r_8[0, 1]\end{aligned}\quad (18)$$

Converting  $\tilde{r}$  to  $\bar{r}$  is performed as below:

$$\bar{r} = [r_0 - \xi_1, r_0 + \xi_2], \quad (19)$$

where

$$\begin{aligned}\xi_1 = & |r_1| + |\min(0, r_2)| + |r_3| + |r_4| + |r_5| \\ & + |\min(0, r_6)| + |r_7| + |\min(0, r_8)|, \\ \xi_2 = & |r_1| + \max(0, r_2) + |r_3| + |r_4| + |r_5| \\ & + \max(0, r_6) + |r_7| + \max(0, r_8)\end{aligned}$$

## 5 Method and algorithm

We now outline how IA or AA as appropriate is used to draw an algebraic curve  $f(x, y) = 0$  in a given rectangular interval.

The basic strategy is that we evaluate  $f(\bar{x}, \bar{y})$  over the desired interval using IA or AA. If the resulting interval does not contain 0, no curve is present. If it does contain 0, we divide the interval horizontally and vertically at its mid point, and consider the pieces in turn. The process stops when an interval consisting of a single pixel is left.

### 5.1 Algorithms for recursive subdivision

The above strategy may be written as a recursive algorithm as below:

```
PROCEDURE interval-quadtree( $x_{min}, x_{max}, y_{min}, y_{max}$ ):
  result = AA or IA evaluation( $x_{min}, x_{max}, y_{min}, y_{max}$ );
  if resultmin ≤ 0 ≤ resultmax then
    if  $x_{max} - x_{min} < 1$  AND  $y_{max} - y_{min} < 1$  then
      plot pixel(round( $(x_{min} + x_{max})/2$ ), round( $(y_{min} + y_{max})/2$ ))
    else subdivide( $x_{min}, x_{max}, y_{min}, y_{max}$ ).
```

```
PROCEDURE subdivide( $x_{min}, x_{max}, y_{min}, y_{max}$ ):
   $x_{mid} = \text{round}((x_{min} + x_{max})/2)$ ;
   $y_{mid} = \text{round}((y_{min} + y_{max})/2)$ ;
  interval-quadtree( $x_{min}, x_{mid}, y_{min}, y_{mid}$ );
  interval-quadtree( $x_{min}, x_{mid}, y_{mid} + 1, y_{max}$ );
  interval-quadtree( $x_{mid} + 1, x_{max}, y_{mid} + 1, y_{max}$ );
  interval-quadtree( $x_{mid} + 1, x_{max}, y_{min}, y_{mid}$ ).
```

## 5.2 Algorithm for polynomial evaluation

We now give the algorithm for computing the interval produced by AA evaluation of  $f(x, y) = \sum_{i=1}^n c_i x_i^{p_i} y_i^{q_i}$ .

```

PROCEDURE  AA evaluation:
  convert  $\bar{x}$  and  $\bar{y}$  to  $\hat{x}$  and  $\hat{y}$  using Equation (13);
  initialise  $r = 0$ ;
  loop from  $i = 1$  to  $n$ 
     $temp_1 = \hat{x}_i^{p_i}$  calculated as in Section 4.2;
     $temp_2 = \hat{y}_i^{q_i}$  calculated as in Section 4.2;
     $temp_3 = temp_1 * temp_2$  multiplied using Equation (16);
     $temp_4 = c_i * (temp_3)$  multiplied using Equation (7);
     $r = r + temp_4$  using Equation (5);
  convert  $r$  to interval form  $result$  as in Equations (17) to (19).

```

The IA algorithm for polynomial evaluation is done by using standard rules for IA as in Equations (1) to (3).

## 6 Examples

To compare the relative merits of IA and AA for curve drawing, we now present some practical examples in Figures 1–4. The curve functions for Figure 1 and Figure 2 are from Stolfi et al. [7, 11] and Seggern [13]. The curves are drawn in a rectangular interval  $[-2, 2] \times [-2, 2]$ . The curve functions for Figure 3 and 4 are given by Voiculescu [14] and the curves are drawn in  $[0, 1] \times [0, 1]$ .

From these figures, it can be seen that

—Curves drawn with AA are finer (‘thinner’) than with IA, that is to say, AA provides better drawing quality than IA, because it provides tighter intervals. A dramatic improvement can be seen in the particular case of Figure 4.

—The number of divisions is less when drawing with AA (see Figures 1–4). Improvement increases with the complexity of the curve function. Again this is because of tighter intervals.

Running times for AA and IA drawing were also measured. Table 1 shows average timings that are obtained by drawing each curve ten times. Due to the complexity of AA computation, AA drawing speed is not significantly faster than IA drawing speed for simple curves but improvement is observed in certain cases — see Figure 4. Note that in such cases, we *simultaneously* achieve better performance *and* a better graphical result.

Note that a version of the program which simply uses the AA rules in Section 3 would not have the advantages noted above. These results rely on the extra coherence obtained by the careful method of bivariate polynomial evaluation we give in Section 4.

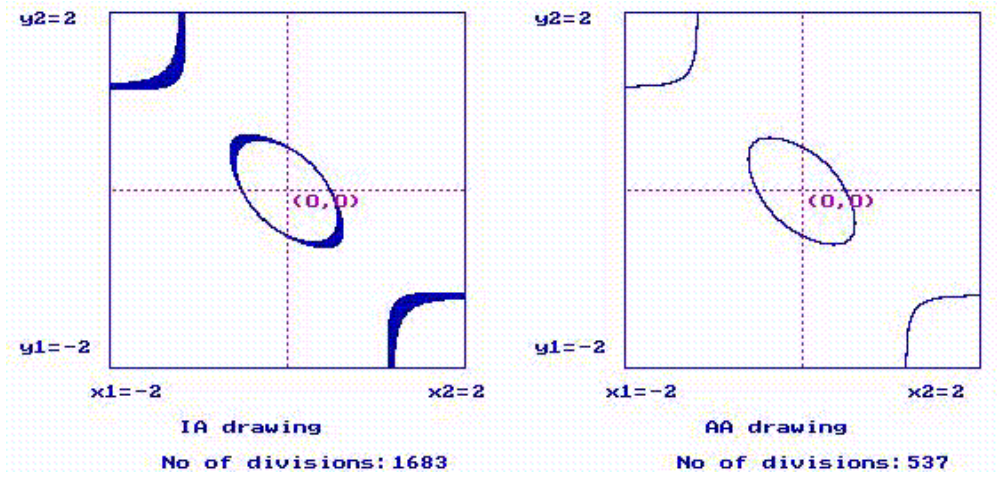


Figure 1: Drawing curve  $x^2 + y^2 + xy - 0.5x^2y^2 - 0.25 = 0$

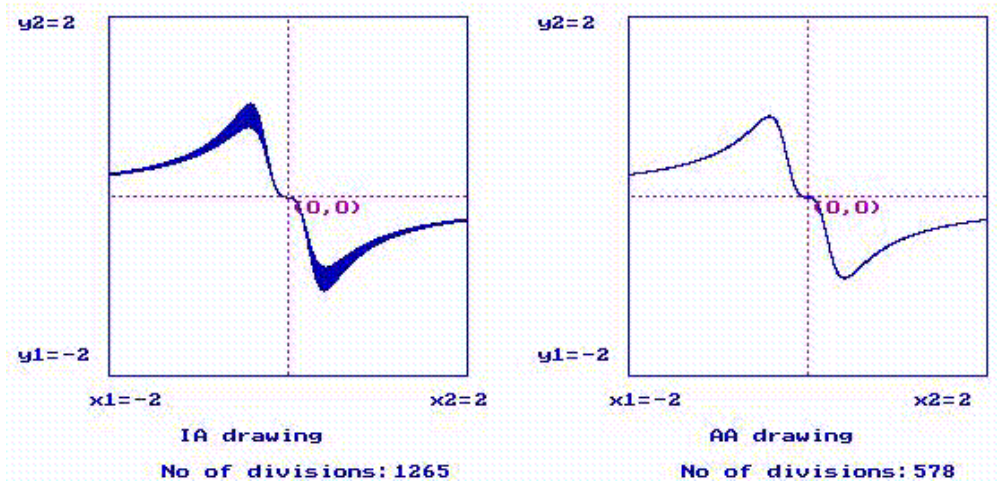


Figure 2: Drawing curve  $0.01y + x^4y + 0.5x^3 = 0$

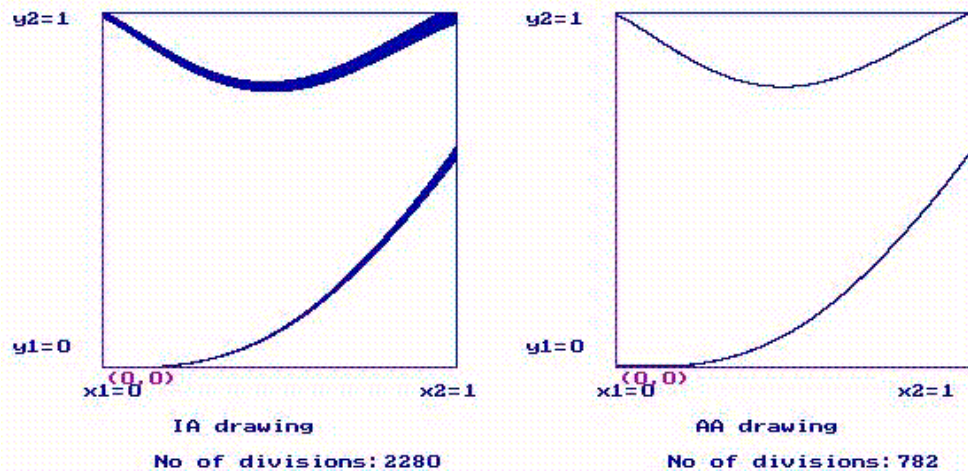


Figure 3: Drawing curve  $x^9 - x^7y + 3x^2y^6 - y^3 + y^5 + y^4x - 4y^4x^3 = 0$

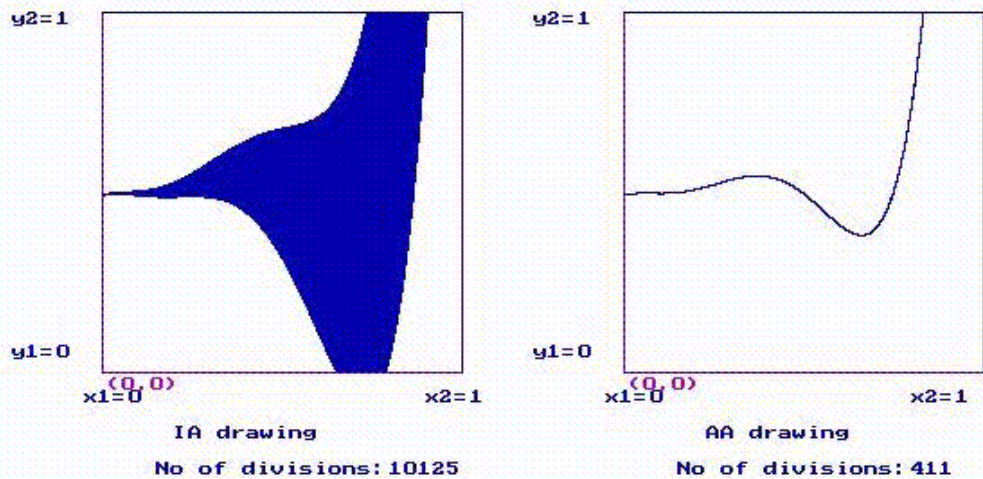


Figure 4: Drawing curve  $20160x^5 - 30176x^4 + 14156x^3 - 2344x^2 + 151x + 237 - 480y = 0$

Curve fuctions	AA drawing	IA drawing
$x^2 + y^2 + xy - 0.2x^2y^2 - 0.25 = 0$	89	115
$0.01y + x^4y + 0.5x^3 = 0$	60	65
$x^9 - x^7y + 3x^3y^7 - y^3 + y^5 + y^4x - 4y^4x^3 = 0$	235	209
$20160x^5 - 30176x^4 + 14156x^3 - 2344x^2 + 151x + 237 - 480y = 0$	99	769

Table 1: Running time comparison between AA and IA drawing (in ms)

## 7 Conclusions

Overall, we conclude that when AA is used with our careful polynomial evaluation method to obtain extra coherence,

1. The curves drawn by the AA method are better (i.e. thinner, with less pixels filled through which the curve does not pass) than those drawn by IA, as the intervals produced during polynomial evaluation are tighter.
2. The AA method achieves its results with less subdivided regions being considered, again because of tighter intervals.
3. Although the previous result might lead to an expectation that the AA method would also be quicker, this is not necessarily so because the AA calculations are more complex than IA ones. However, speed advantages are present in some cases when IA performs particularly badly (e.g. as in the case shown in Figure 4).

Finally, we may remark that we fully expect the benefits shown in curve drawing to also be applicable to other uses of solutions to  $f(x, y) = 0$ , such as surface intersection.

## References

- [1] A. Glassner, *Space Subdivision for Fast Ray Tracing*, IEEE Computer Graphics & Applications, 10, 15-22, (1984)
- [2] S. P. Mudur, P. A. Koparkar, *Interval Methods for Processing Geometric Objects*, IEEE Computer Graphics and Applications 4(2), 7-17, (1984)
- [3] K. G. Suffern, E. D. Fackerell, *Interval Methods in Computer Graphics*, Computer & Graphics 15(3), 331 - 340, (1991)
- [4] T. Duff, *Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry*, Proceedings of SIGGRAPH'92, in ACM Computer Graphics, 26(2), 131-138, (1992)
- [5] J. M. Snyder, *Interval Analysis for Computer Graphics*, Proceedings of SIGGRAPH'92, in ACM Computer Graphics, 26(2), 121-130, (1992)

- [6] A. Geisow, *Surface Interrogations*, PhD Thesis, University of East Anglia, (1983)
- [7] J. L. D. Comba, J. Stolfi, *Affine Arithmetic and its Applications to Computer Graphics*, Proceedings of Anais do VII SIBGRAPI, 9-18, (1993)
- [8] R. E. Moore, *Interval Analysis*, Prentice-Hall, (1966)
- [9] R. E. Moore, *Methods and Applications of Interval Analysis*, SIAM, Philadelphia (1979)
- [10] L. H. de Figueiredo, *Surface Intersection Using Affine Arithmetic*, Proceedings of Graphics Interface'96, 168-175, (1996)
- [11] L. H. de Figueiredo, J. Stolfi, *Adaptive Enumeration of Implicit Surfaces with Affine Arithmetic*, Computer Graphics Forum, 15(5), 287-296, (1996)
- [12] W. Heidrich, P. Slusallek H. P. Seidel, *Sampling of Procedural Shaders Using Affine Arithmetic*, ACM Transaction on Graphics, (1998)
- [13] D. H. von Seggern, *CRC Handbook of Mathematical Curves and Surfaces*, CRC Press, Inc. (1990)
- [14] I. Voiculescu, *personal communication*