Modified Affine Arithmetic Is More Accurate than Centered Interval Arithmetic or Affine Arithmetic

Huahao Shou[1,2], Hongwei Lin[1], Ralph Martin[2], and Guojin Wang[1]

[1] State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou, China
[2] Department of Applied Mathematics, Zhejiang University of Technology Hangzhou, China
[3] Department of Computer Science, Cardiff University, Cardiff, UK
Ralph.Martin@cs.cf.ac.uk

Abstract. In this paper we give mathematical proofs of two new results relevant to evaluating algebraic functions over a box-shaped region: (i) using interval arithmetic in centered form is always more accurate than standard affine arithmetic, and (ii) modified affine arithmetic is always more accurate than interval arithmetic in centered form. Test results show that modified affine arithmetic is not only more accurate but also much faster than standard affine arithmetic. We thus suggest that modified affine arithmetic is the method of choice for evaluating algebraic functions, such as implicit surfaces, over a box.

1 Introduction

Affine arithmetic (AA) was first introduced by Comba and Stolfi in 1993 [3] as an improvement to interval arithmetic (IA). When used for finding the range of a multivariate polynomial function over a box, AA can be more resistant to over-conservatism due to its ability to keep track of correlations between various quantities, doing so using a linear series of "noise terms".

AA has been successfully applied as a replacement for IA in many geometric and computer graphics applications such as surface intersection [5], adaptive enumeration of implicit surfaces [6], ray tracing procedural displacement shaders [7], sampling procedural shaders [8], ray casting implicit surfaces [4], linear interval estimations for parametric objects [2] and in a CSG geometric modeller [1].

However, standard AA still has an over-conservatism problem because it uses an imprecise approximation in the multiplication of two affine forms, and we have shown how it can be further improved to give so-called modified affine arithmetic (MAA). MAA uses a matrix form for bivariate polynomial evaluation which keeps all noise terms without any unprecise approximation [9,11]. Of course, this more precise MAA involves more complex formulas.

In practical applications such as curve drawing, typically recursive methods are used to locate the curve. The extra accuracy provided by MAA means that fewer recursive calls are likely to be needed—some regions of the plane can be rejected as not containing the curve when using MAA which would need further subdivision when using ordinary AA. However, the amount of work to be done for each recursive call is greater for MAA than AA, and so it is still not clear whether MAA's advantage in accuracy is worth the extra cost and complexity in terms of overall algorithm performance. We thus give an empirical comparison between standard AA and MAA used in a curve drawing algorithm, as well as a theoretical proof that over a given interval, MAA is more accurate. Our test results in Section 3 show that MAA is not only more accurate but also much faster than standard AA in a curve drawing application.

We also demonstrate that MAA is actually the same as interval arithmetic on the centered form (IAC) [10] together with a consideration of the properties of even or odd powers of polynomial terms. In detail, we show in Section 2 that IAC is always more accurate than standard AA for polynomial evaluation, and that the MAA method is always somewhat more accurate than the IAC method. Overall, we conclude that the MAA method is better than the IAC method, and the IAC method is better than the standard AA method. These results hold in one, two and three dimensions.

The subdivision quadtree based implicit curve plotting algorithm described in [9] can be easily generalized to a subdivision octree based implicit surface plotting algorithm. The MAA in matrix form method proposed in [11] for bivariate polynomial evaluation and algebraic curve plotting problem can also be readily generalized to an MAA in tensor form method for trivariate polynomial evaluation and algebraic surface plotting. Thus, a 3D subdivision based algebraic surface plotting method and the 2D subdivision based algebraic curve plotting method have many similarities. Although further experiments are needed, due to these similarities we believe that the experimental conclusions we draw from 2D algebraic curve plotting problem are also applicable to the 3D algebraic surface plotting problem.

This paper is organized as follows. In Section 2 we theoretically prove two new results: that IAC is more accurate than AA, and MAA is more accurate than IAC, for evaluation of multivariate polynomials over a box-shaped region. In Section 3 we use some examples to test whether the MAA method is more efficient than the AA method when used in a practical curve drawing application. Finally in Section 4 we give some conclusions.

2 Why MAA Is More Accurate than AA

In this Section we prove theoretically two new results: that IAC is more accurate than AA, and MAA is more accurate than IAC. For definitions and explanations of how to evaluate functions using IAC, AA and MAA, see [9].

Theorem 1: IAC is more accurate than AA for bounding the range of a polynomial.

Proof: We only prove the theorem here the one dimensional case to avoid much more complex formulae needed in the 2D and 3D cases. However, the same basic idea works in all dimensions. Suppose we wish to find bounds on the range of $f(x)$ over some interval $x \in [\underline{x}, \overline{x}]$.

Let

$$f(x) = \sum_{i=0}^{n} a_i x^i.$$

Let $\hat{x} = x_0 + x_1\varepsilon_1$ be the affine form of the interval $[\underline{x}, \overline{x}]$, where ε_1 is the noise symbol whose value is unknown but is assumed to be in the range $[-1, 1]$, $x_0 = (\underline{x} + \overline{x})/2$, and $x_1 = (\overline{x} - \underline{x})/2 > 0$.

Using AA we may write:

$$f(\hat{x}) = \sum_{i=0}^{n} a_i x_0^i + \sum_{i=1}^{n} i a_i x_0^{i-1} x_1 \varepsilon_1 + \sum_{k=2}^{n} (\sum_{i=k}^{n} a_i x_0^{i-k}) x_1 [(|x_0| + x_1)^{k-1} - |x_0|^{k-1}] \varepsilon_k, \tag{1}$$

where $\varepsilon_k, k = 2, 3, \cdots, n$ are also noise symbols whose values are assumed to be in the range $[-1, 1]$.

Therefore the upper bound of $f(\hat{x})$ computed using AA is:

$$\begin{aligned}\overline{x}_{AA} &= \sum_{i=0}^{n} a_i x_0^i + |\sum_{i=1}^{n} i a_i x_0^{i-1}| x_1 + \sum_{k=2}^{n} |\sum_{i=k}^{n} a_i x_0^{i-k}| x_1 [(|x_0| + x_1)^{k-1} - |x_0|^{k-1}] \\ &= \sum_{i=0}^{n} a_i x_0^i + |\sum_{i=1}^{n} i a_i x_0^{i-1}| x_1 + \sum_{l=2}^{n} (\sum_{k=l}^{n} C_{k-1}^{l-1} |x_0|^{k-l} |\sum_{i=k}^{n} a_i x_0^{i-k}|) x_1^l\end{aligned}$$

Using IAC we may write:

$$f(\hat{x}) = \sum_{i=0}^{n} a_i x_0^i + \sum_{i=1}^{n} i a_i x_0^{i-1} x_1 \varepsilon_1 + \sum_{k=2}^{n} (\sum_{i=k}^{n} a_i C_i^k x_0^{i-k}) x_1^k \varepsilon_1^k \tag{2}$$

Therefore the upper bound of $f(\hat{x})$ computed using IAC is:

$$\begin{aligned}\overline{x}_{IAC} &= \sum_{i=0}^{n} a_i x_0^i + |\sum_{i=1}^{n} i a_i x_0^{i-1}| x_1 + \sum_{k=2}^{n} |\sum_{i=k}^{n} a_i C_i^k x_0^{i-k}| x_1^k \\ &= \sum_{i=0}^{n} a_i x_0^i + |\sum_{i=1}^{n} i a_i x_0^{i-1}| x_1 + \sum_{l=2}^{n} |\sum_{k=l}^{n} C_{k-1}^{l-1} x_0^{k-l} (\sum_{i=k}^{n} a_i x_0^{i-k})| x_1^l\end{aligned}$$

Since the first term and the second term of $\overline{x}_{AA}$ and $\overline{x}_{IAC}$ are the same, while for the third term it always holds that

$$|\sum_{k=l}^{n} C_{k-1}^{l-1} x_0^{k-l} (\sum_{i=k}^{n} a_i x_0^{i-k})| \leq \sum_{k=l}^{n} C_{k-1}^{l-1} |x_0|^{k-l} |\sum_{i=k}^{n} a_i x_0^{i-k}|,$$

we thus obtain that $\overline{x}_{IAC} \leq \overline{x}_{AA}$.

In a similar way we can prove that the lower bounds of AA and IAC satisfy $\underline{x}_{IAC} \geq \underline{x}_{AA}$.

Therefore we have proved that the bounds provided by IAC are more accurate than those provided by AA when evaluating a univariate polynomial over a range.

In addition, we can clearly see from equations (1) and (2) that the expression which must be evaluated in AA is actually more complicated and contains more arithmetic operations than the corresponding expression in IAC. We therefore conclude that IAC is always to be preferred to AA for polynomial evaluation.

Theorem 2: MAA is more accurate than IAC for bounding the range of a polynomial.

Proof: We only prove the theorem here in the 2D case. The proof is similar in the 1D and 3D cases. Let

$$f(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{m}a_{ij}x^iy^j,\quad (x,y)\in[\underline{x},\overline{x}]\times[\underline{y},\overline{y}].$$

Let $\hat{x}=x_0+x_1\varepsilon_x,\quad \hat{y}=y_0+y_1\varepsilon_y$ be the affine forms of the intervals $[\underline{x},\overline{x}]$ and $[\underline{y},\overline{y}]$ respectively, where $\varepsilon_x,\varepsilon_y$ are noise symbols whose values are unknown but are assumed to be in the range $[-1,1]$, $x_0=(\underline{x}+\overline{x})/2,\quad x_1=(\overline{x}-\underline{x})/2>0$, and $y_0=(\underline{y}+\overline{y})/2,\quad y_1=(\overline{y}-\underline{y})/2>0$. Let

$$f(\hat{x},\hat{y})=\sum_{i=0}^{n}\sum_{j=0}^{m}d_{ij}\varepsilon_x^i\varepsilon_y^j$$

be the centered form of the polynomial.

Using MAA, the upper bound for the range of the function over this region is

$$\overline{x}_{MAA}=d_{00}+\sum_{j=1}^{m}\left\{\begin{array}{cc}\max(0,d_{0j}), \text{ if } j \text{ is even}\\ |d_{0j}|, \quad\quad \text{otherwise}\end{array}\right\}$$
$$+\sum_{i=1}^{n}\sum_{j=0}^{m}\left\{\begin{array}{cc}\max(0,d_{ij}), & \text{if } i,j \text{ are both even}\\ |d_{ij}|, & \text{otherwise}\end{array}\right\}.$$

Using IAC, the upper bound for the range of the function over this region is

$$\overline{x}_{IAC}=d_{00}+\sum_{j=1}^{m}|d_{0j}|+\sum_{i=1}^{n}\sum_{j=0}^{m}|d_{ij}|.$$

Since it always holds that $\max(0,d_{0j})\le|d_{0j}|$ and $\max(0,d_{ij})\le|d_{ij}|$ we get that $\overline{x}_{MAA}\le\overline{x}_{IAC}$.

in a similar way, we can prove that the lower bounds obtained using MAA and IAC satisfy $\underline{x}_{MAA}\ge\underline{x}_{IAC}$.

Therefore we have proved that MAA provides more accurate bounds on the range of a bivariate polynomial over a rectangular region than does IAC.

The weak point of standard AA is that it uses a new noise symbol with a conservative coefficient to replace the quadratic term generated when multiplying two affine forms. This error due to conservativism is accumulated and magnified during long chains of multiplication operations, resulting in an "error explosion" problem, well known to also arise in standard IA. Thus, while standard AA is aimed at reducing this tendency of IA, and does so to some extent as shown in the examples in Section 3, it is possible to better with MAA.

As proved above, the MAA method provides more accurate bounds on a polynomial function over a range than the standard AA method. Whether the more accurate MAA method is also faster than the standard AA in such algorithms as one for recursive curve plotting over a region is not so obvious—while less subdivision is needed as some parts of the plane can be discarded sooner due to the higher accuracy of MAA, the amount of computation needed for each range evaluation is greater using MAA. Whether the advantages outwiegh the disadvantages must be determined by experiment. In the following section we give some examples to see what happens when AA and MAA are applied to the same bivariate polynomial evaluation and subdivision based algebraic curve plotting problem. Also see [9] for further results of this kind.

3 Experimental Comparison of AA and MAA for Curve Plotting

In this section we use ten carefully chosen example curves to compare the relative speed of AA and MAA, and to confirm the theoretical results concerning relative accuracy. Each example consists of plotting an implicit curve $f(x, y) = 0$ using the algorithm given in [9] on a grid of 256×256 pixels. We recursively compute whether a region can contain the curve by computing a bound on the range of the function over the region. If the range does not contain zero, the curve cannot be present in the region, and is discarded. If the range does contain zero, the region is subdivided in x and y, and retested. We continue down to 1×1 regions, which are plotted in black if they still potentially contain the curve. We used Visual C++ 6.0 running on Windows 2000 on a Pentium IV 2.00GHz computer with 512MB RAM for all the tests.

Overall, these examples were chosen to illustrate curves of varying polynomial degree, with differing numbers of both open and closed components, and include cusps, self-intersections and tangencies as special cases. Obviously, no finite set of test cases can establish universal truths, but we have aimed to capture a range of curve behaviour with these test cases, to at least give some hope that any conclusions we draw are relevant to many practical cases of interest.

We not only show the generated graphical output for these examples, but also present in tabular form an analysis of accuracy and computational load for each example. When comparing the performance and efficiency of AA and MAA methods, a number of quantities were measured:

– The number of pixels plotted, the fewer the better: plotted pixels may or may not contain the curve in practice.
– The CPU time used, the less the better.
– The number of subdivisions involved, the fewer the better.

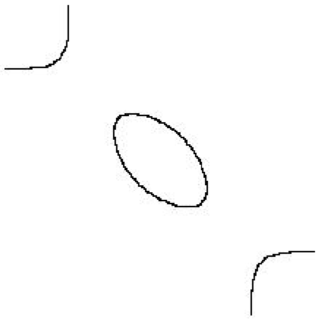

Fig. 1. Example 1. $\frac{15}{4} + 8x - 16x^2 + 8y - 112xy + 128x^2y - 16y^2 + 128xy^2 - 128x^2y^2 = 0$, plotted using AA.

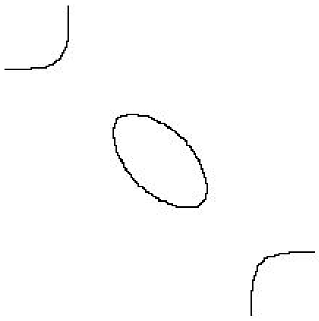

Fig. 2. Example 1. $\frac{15}{4} + 8x - 16x^2 + 8y - 112xy + 128x^2y - 16y^2 + 128xy^2 - 128x^2y^2 = 0$, plotted using MAA.

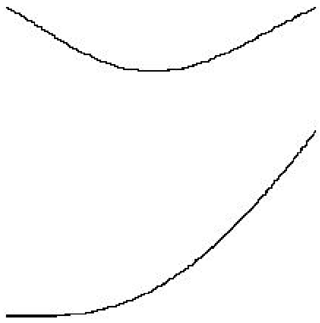

Fig. 3. Example 2. $20160x^5 - 30176x^4 + 14156x^3 - 2344x^2 + 151x + 237 - 480y = 0$, plotted using AA.

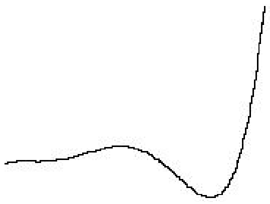

Fig. 4. Example 2. $20160x^5 - 30176x^4 + 14156x^3 - 2344x^2 + 151x + 237 - 480y = 0$, plotted using MAA.

The ten algebraic curve examples we used here for comparison of AA and MAA are all chosen from [9]. The graphical outputs for all these ten curve examples using AA and MAA methods respectively are shown in Figure 1 to Figure 20. The tabulated results are presented in Table 1.

From Figures 1–20 and Table 1 we can see that in general the MAA method is much more accurate *and* quicker than AA method. The AA method is particularly bad in Examples 5, 6, 9, and 10. In Example 5, AA completely fails to reveal the shape of the curve while MAA successfully reveals it. In Example 6 the curve generated by AA is much thicker than the one generated by MAA. In Example 9 AA is unable to distinguish two concentric circles of very similar radii, while MAA can do this. In Example 10 AA has an overconservativism problem near the tangency point of two circles which MAA does not. MAA is slightly more accurate than IAC, and MAA takes almost the same CPU time as IAC. Overall, the performance of MAA is slightly better than IAC.

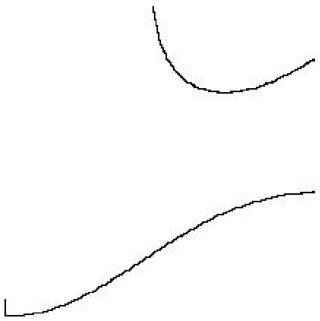

Fig. 5. Example 3. $0.945xy - 9.43214x^2y^3 + 7.4554x^3y^2 + y^4 - x^3 = 0$, plotted using AA.

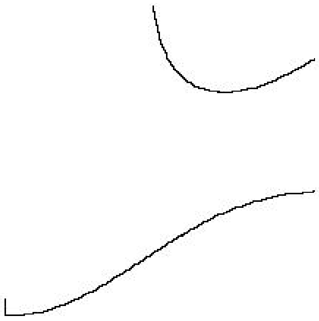

Fig. 6. Example 3. $0.945xy - 9.43214x^2y^3 + 7.4554x^3y^2 + y^4 - x^3 = 0$, plotted using MAA.

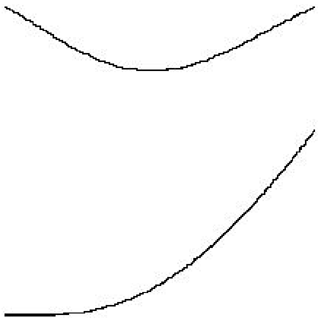

Fig. 7. Example 4. $x^9 - x^7y + 3x^2y^6 - y^3 + y^5 + y^4x - 4y^4x^3 = 0$, plotted using AA.

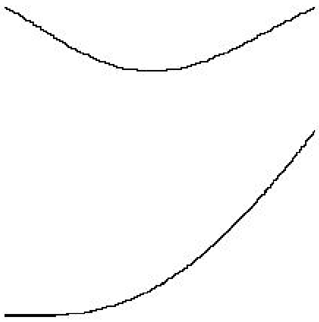

Fig. 8. Example 4. $x^9 - x^7y + 3x^2y^6 - y^3 + y^5 + y^4x - 4y^4x^3 = 0$, plotted using MAA.

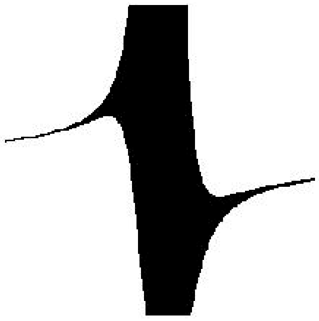

Fig. 9. Example 5. $-\frac{1801}{50} + 280x - 816x^2 + 1056x^3 - 512x^4 + \frac{1601}{25}y - 512xy + 1536x^2y - 2048x^3y + 1024x^4y = 0$, plotted using AA.

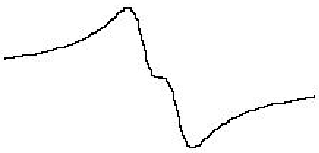

Fig. 10. Example 5. $-\frac{1801}{50} + 280x - 816x^2 + 1056x^3 - 512x^4 + \frac{1601}{25}y - 512xy + 1536x^2y - 2048x^3y + 1024x^4y = 0$, plotted using MAA.

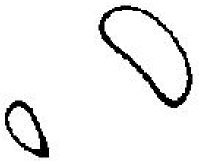

Fig. 11. Example 6. $\frac{601}{9} - \frac{872}{3}x + 544x^2 - 512x^3 + 256x^4 - \frac{2728}{9}y + \frac{2384}{3}xy - 768x^2y + \frac{5104}{9}y^2 - \frac{2432}{3}xy^2 + 768x^2y^2 - 512y^3 + 256y^4 = 0$, plotted using AA.

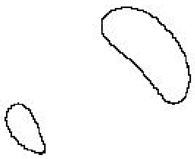

Fig. 12. Example 6. $\frac{601}{9} - \frac{872}{3}x + 544x^2 - 512x^3 + 256x^4 - \frac{2728}{9}y + \frac{2384}{3}xy - 768x^2y + \frac{5104}{9}y^2 - \frac{2432}{3}xy^2 + 768x^2y^2 - 512y^3 + 256y^4 = 0$, plotted using MAA.

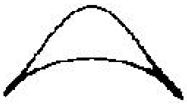

Fig. 13. Example 7. $-13 + 32x - 288x^2 + 512x^3 - 256x^4 + 64y - 112y^2 + 256xy^2 - 256x^2y^2 = 0$, plotted using AA.

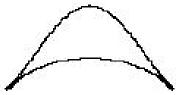

Fig. 14. Example 7. $-13 + 32x - 288x^2 + 512x^3 - 256x^4 + 64y - 112y^2 + 256xy^2 - 256x^2y^2 = 0$, plotted using MAA.

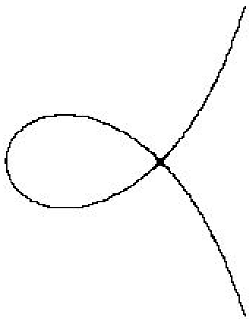

Fig. 15. Example 8. $-\frac{169}{64} + \frac{51}{8}x - 11x^2 + 8x^3 + 9y - 8xy - 9y^2 + 8xy^2 = 0$, plotted using AA.

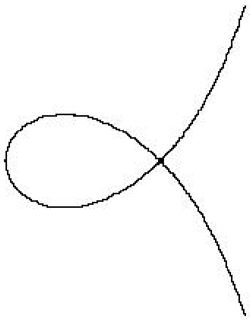

Fig. 16. Example 8. $-\frac{169}{64} + \frac{51}{8}x - 11x^2 + 8x^3 + 9y - 8xy - 9y^2 + 8xy^2 = 0$, plotted using MAA.

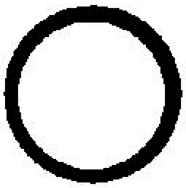

Fig. 17. Example 9. $47.6 - 220.8x + 476.8x^2 - 512x^3 + 256x^4 - 220.8y + 512xy - 512x^2y + 476.8y^2 - 512xy^2 + 512x^2y^2 - 512y^3 + 256y^4 = 0$, plotted using AA.

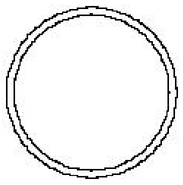

Fig. 18. Example 9. $47.6 - 220.8x + 476.8x^2 - 512x^3 + 256x^4 - 220.8y + 512xy - 512x^2y + 476.8y^2 - 512xy^2 + 512x^2y^2 - 512y^3 + 256y^4 = 0$, plotted using MAA.

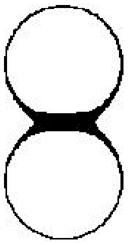

Fig. 19. Example 10. $\frac{55}{256} - x + 2x^2 - 2x^3 + x^4 - \frac{55}{64}y + 2xy - 2x^2y + \frac{119}{64}y^2 - 2xy^2 + 2x^2y^2 - 2y^3 + y^4 = 0$, plotted using AA.

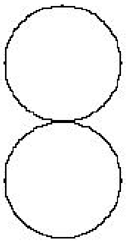

Fig. 20. Example 10. $\frac{55}{256} - x + 2x^2 - 2x^3 + x^4 - \frac{55}{64}y + 2xy - 2x^2y + \frac{119}{64}y^2 - 2xy^2 + 2x^2y^2 - 2y^3 + y^4 = 0$, plotted using MAA.

Table 1. Comparison of AA, MAA and IAC methods by examples.

Example	Method	Pixels plotted	Subdivisions involved	CPU time used
1	AA	604	900	1.047 sec
1	IAC	530	587	0.047 sec
1	MAA	526	563	0.047 sec
2	AA	513	815	1.219 sec
2	IAC	435	471	0.063 sec
2	MAA	433	459	0.063 sec
3	AA	625	715	1.187 sec
3	IAC	609	638	0.094 sec
3	MAA	608	634	0.094 sec
4	AA	832	934	4.969 sec
4	IAC	819	880	0.547 sec
4	MAA	816	857	0.562 sec
5	AA	15407	9027	49.468 sec
5	IAC	470	659	0.063 sec
5	MAA	464	611	0.062 sec
6	AA	1287	2877	10.266 sec
6	IAC	466	596	0.109 sec
6	MAA	460	560	0.110 sec
7	AA	933	1409	1.766 sec
7	IAC	532	675	0.078 sec
7	MAA	512	627	0.078 sec
8	AA	891	989	0.938 sec
8	IAC	838	853	0.078 sec
8	MAA	818	827	0.078 sec
9	AA	5270	4314	13.75 sec
9	IAC	1208	1373	0.250 sec
9	MAA	1144	1269	0.250 sec
10	AA	2071	2796	8.625 sec
10	IAC	812	905	0.172 sec
10	MAA	784	845	0.171 sec

The reasons why MAA is much faster than AA are as follows. Firstly, the most crucial reason which we can clearly see from Section 2 is that the expression used in AA is actually more complicated than that used in IAC or MAA. Therefore AA in fact involves more arithmetic operations than IAC or MAA. Secondly AA is less accurate than MAA, and therefore AA needs more subdivisions. Furthermore, many more incorrect pixels cannot be discarded, which increases the computation load of the AA method. Thirdly, although MAA looks more complicated, actually MAA only contains matrix manipulations which are easy to implement using loops, while AA, although looking simple, requires dynamic lists to represent affine forms with varying numbers of noise symbols. AA operations $(+, -, *)$ must be performed through insertion and deletion of elements of the lists, which are not as efficient the simper arithmetic operations in MAA.

4 Conclusions

From the above theoretical proofs and the experimental test results we conclude that the MAA method for estimating bounds on a polynomial over a range is not only more accurate but also much faster than the standard AA method. We also have demonstrated that the MAA method is very similar to the IAC method but also takes into consideration the special properties of even and odd powers of polynomial terms. Therefore the MAA method is always at least as or slightly more accurate than the IAC method. We have also shown that the IAC method is more accurate than the standard AA method. In conclusion we strongly recommend that the MAA method is used instead of standard AA method in geometric computations on implicit curves and surfaces.

Acknowledgements

This work was supported jointly by the National Natural Science Foundation of China (Grant 60173034), National Natural Science Foundation for Innovative Research Groups (Grant 60021201) and the Foundation of State Key Basic Research 973 Item (Grant 2002CB312101).

References

1. Bowyer, A., Martin, R., Shou, H., Voiculescu, I.: Affine intervals in a CSG geometric modeller. In: Winkler J., Niranjan, M. (eds.): Uncertainty in Geometric Computations. Kluwer Academic Publisher (2002) 1-14
2. Bühler, K.: Linear interval estimations for parametric objects theory and application. Computer Graphics Forum 20(3) (2001) 522–531
3. Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. Anais do VII SIBGRAPI (1993) 9–18
Available at http://www.dcc.unicamp.br/~stolfi/

4. De Cusatis, A., Jr., De Figueiredo, L.H., Gattass, M.: Interval Methods for Ray Casting Implicit Surfaces with Affine Arithmetic. XII Brazilian Symposium on Computer Graphics and Image Processing (1999) 65–71
5. De Figueiredo, L.H.: Surface intersection using affine arithmetic. Proceedings of Graphics Interface (1996) 168–175
6. De Figueiredo, L.H., Stolfi, J.: Adaptive enumeration of implicit surfaces with affine arithmetic. Computer Graphics Forum 15(5) (1996) 287–296
7. Heidrich, W., Seidel, H.P.: Ray tracing procedural displacement shaders. Proceedings of Graphics Interface (1998) 8–16
8. Heidrich, W., Slusallek, P., Seidel, H.P.: Sampling of procedural shaders using affine arithmetic. ACM Trans. on Graphics 17(3) (1998) 158–176
9. Martin, R., Shou, H., Voiculescu, I., Bowyer, A., Wang, G.: Comparison of interval methods for plotting algebraic curves. Computer Aided Geometric Design 19(7) (2002) 553–587
10. Ratschek, H., Rokne, J.: Computer Methods for the Range of Functions. Ellis Horwood (1984)
11. Shou, H., Martin, R., Voiculescu, I., Bowyer, A., Wang, G.: Affine arithmetic in matrix form for polynomial evaluation and algebraic curve drawing. Progress in Natural Science 12(1) (2002) 77–80