

# A Recursive Taylor Method for Ray-Casting Algebraic Surfaces\*

Huahao Shou<sup>a</sup>, Wenhao Song<sup>b</sup>, Jie Shen<sup>c</sup>, Ralph Martin<sup>d</sup>, Guojin Wang<sup>b</sup>

<sup>a</sup> Department of Applied Mathematics, Zhejiang University of Technology, Hangzhou 310032, China.

<sup>b</sup> Department of Mathematics, Zhejiang University, Hangzhou 310027, China.

<sup>c</sup> Department of Computer & Information Science, University of Michigan-Dearborn, MI 48128, USA.

<sup>d</sup> School of Computer Science, Cardiff University, Wales CF24 3AA, UK.

## Abstract

In this paper, we propose a recursive Taylor method for ray-casting algebraic surfaces. The performance of this approach is compared with four other candidate approaches to ray-casting: using interval arithmetic on the power form, using interval arithmetic on centered forms, using affine arithmetic, and using modified affine arithmetic. Test results show that the recursive Taylor method compares favorably to the other methods.

**Keywords:** Interval arithmetic, ray-casting, algebraic surface

## 1 Introduction

An algebraic surface is defined as a point set  $S = \{(x, y, z) \mid h(x, y, z) = 0\}$ , where  $h(x, y, z)$  is a trivariate polynomial. Algebraic surfaces are widely used in computer aided geometric design and computer graphics.

A very important task for such surfaces is to render them, i.e., to compute an image of the set  $S$  as seen from a given point of view in  $R^3$ . One technique for rendering an algebraic surface is ray-casting, in which rays are cast from the viewpoint, through an image plane, until they first hit the surface  $S$ . Typically, at the intersection point, the normal to  $S$  is computed and used to calculate the color of the corresponding pixel in the image according to some illumination model. Ray-casting is a basic step in ray tracing, a more sophisticated technique that can handle reflections and refractions [3].

A ray from the viewpoint  $E$  (where the eye is located) in the direction  $v$  can be written in parametric form as  $r(t) = E + t \cdot v$ , where  $t \in [0, \infty)$ . Rendering via ray-casting depends crucially on being able to find reliably the smallest positive root of the nonlinear equation  $h(r(t)) = 0$ .

Reliable solution methods can be provided by range analysis [9], and interval arithmetic [7] is well-suited to that purpose. Interval methods have already been used successfully in ray-tracing [6] as well as for several other computer graphics problems [4, 13, 14].

In this paper, a one dimensional recursive Taylor method (RT) is investigated for finding roots of univariate polynomials in the particular context of ray-casting algebraic surfaces. The performance of this approach is compared with other four generally candidate approaches: using interval arithmetic on the power form of the polynomial (IAP) [7], using interval arithmetic on the centered form of the polynomial (IAC) [9], using affine arithmetic (AA) [2] and using modified affine arithmetic (MAA) [10]. Our experimental test results show that the recursive Taylor method compares favorably to these other methods.

---

\*Project supported jointly by the National Grand Fundamental Research 973 Program of China (No. 2004CB719400) and the National Science Foundation of USA (No. DMI-0514900).

## 2 Ray-surface intersection algorithm

As mentioned in Section 1, the key to ray casting algebraic surface is to reliably find the smallest positive root of the univariate nonlinear equation  $f(t) = h(r(t)) = 0$  on  $[0, T]$ , where  $T$  is some large value, corresponding to some far clipping plane. The basic idea is to evaluate  $f(t)$  over the desired parameter interval using a range analysis estimation method (IAP, IAC, AA, MAA or RT) giving a range  $[\underline{F}, \overline{F}]$  that is guaranteed to contain the exact range of  $f$  over the parameter interval. If the resulting interval does not contain 0, no root can be present in this parameter interval. If it does contain 0, we subdivide the parameter interval into two sub-intervals at its mid point, and recursively consider the pieces in turn. The interval containing lower values is considered first as we are interested in finding the smallest root. The process stops when the size of an parameter interval reaches a user-specified tolerance  $\epsilon$ . In such a case the mid-point of this parameter interval is considered to be the smallest positive root of  $f(t) = 0$ . In detail, we use the following algorithm:

```

Bisection ( $\underline{t}, \overline{t}$ ):
 $[\underline{F}, \overline{F}]$  = Range evaluation of  $f$  on  $(\underline{t}, \overline{t})$ ;
IF  $\underline{F} \leq 0 \leq \overline{F}$  THEN
  IF  $\overline{t} - \underline{t} < \epsilon$  THEN
     $t_0 = (\underline{t} + \overline{t})/2$ 
    RETURN  $t_0$ 
  ELSE
    Bisection ( $\underline{t}, t_0$ )
    Bisection ( $t_0, \overline{t}$ )

```

Figure 1: Subdivision algorithm for finding the first positive root of univariate polynomial

Here  $[\underline{F}, \overline{F}]$  is a conservative interval containing all values of  $f(t)$  over the parameter interval  $[\underline{t}, \overline{t}]$  computed using a chosen range analysis method such as IAP, IAC, AA, MAA or RT, and  $t_0$  is the mid-point of the parameter interval  $[\underline{t}, \overline{t}]$ .

As described in [3], the above simple algorithm does not miss any zeros of  $f$ , because it only discards intervals that cannot contain zeros. Moreover, since on recursion the subinterval with smaller values  $[\underline{t}, t_0]$  is always considered before the subinterval with larger values  $[t_0, \overline{t}]$ , the algorithm finds the smallest root, if any.

The above algorithm stops as soon as the smallest zero has been found. A small modification can be made to the algorithm to find all zeros of  $f$ , in order of size, and to collect them into a list. Although finding all zeros is not necessary for the application of ray-casting algebraic surfaces, it is required for certain other applications such as ray casting CSG models [4].

## 3 Interval Range Analysis Methods

### 3.1 Introduction

Interval arithmetic, which is widely used in scientific computation, is a natural tool for range analysis of a function over an interval. Interval arithmetic is a technique for numerical computation where each uncertain quantity is represented by an interval of floating-point numbers. These intervals are added, subtracted, multiplied, etc. in such a way that each computed interval is guaranteed to contain the unknown value of the quantity that it represents [7]. Details of standard interval arithmetic operations can be found in [7].

A significant property of interval arithmetic is that the form in which the polynomial is expressed can affect the result. We thus consider several different ways of expressing polynomials for evaluation in interval arithmetic.

In the following we describe five interval range analysis methods for univariate polynomial. They are interval arithmetic on the power form of the polynomial, interval arithmetic on its centered form, affine arithmetic, modified affine arithmetic and a recursive Taylor method, respectively.

### 3.2 Interval arithmetic on power form

The power form is the most commonly used polynomial form. We express the univariate polynomial  $f(t)$  as

$$f(t) = \sum_{i=0}^n a_i t^i, \quad t \in [\underline{t}, \bar{t}]$$

Interval arithmetic on the power form means that we substitute the interval  $[\underline{t}, \bar{t}]$  for the variable  $t$  in this power form to estimate the range  $[\underline{F}, \bar{F}]$  of  $f(t)$  on  $[\underline{t}, \bar{t}]$  using standard interval arithmetic rules.

### 3.3 Interval arithmetic on centered form

The centered form has been proven to often give tighter bounds than other polynomial forms when interval arithmetic is applied to a polynomial function [9].

The centered form

$$f(\tilde{t}) = \sum_{i=0}^n d_i \tilde{t}^i, \quad \tilde{t} \in [-1, 1]$$

can be obtained by applying a variable transformation to the power form in Section 3.2, where  $\tilde{t}$  is a new variable satisfying  $t = t_0 + t_1 \tilde{t}$ , where  $t_0 = (\bar{t} + \underline{t})/2$ ,  $t_1 = (\bar{t} - \underline{t})/2$ , and  $d_i$  can be calculated using

$$d_i = \sum_{j=i}^n a_j B_{ij}, \quad i = 0, \dots, n,$$

where

$$B_{ij} = \binom{j}{i} t_0^{j-i} t_1^i, \quad i = 0, \dots, j \text{ and } j = 0, \dots, n.$$

The range  $[\underline{F}, \bar{F}]$  of  $f(t)$  on  $[\underline{t}, \bar{t}]$  can be obtained as follows by applying standard interval arithmetic rules on this centered form:

$$\underline{F} = d_0 - \sum_{i=1}^n |d_i|, \quad \bar{F} = d_0 + \sum_{i=1}^n |d_i|.$$

### 3.4 Affine arithmetic

Affine arithmetic (AA), proposed by Comba and Stolfi [2], is an alternative approach to IA, and can be sometimes more resistant to over-conservatism due to its ability of keeping track of correlations between computed and input quantities. Details of standard affine arithmetic rules can be found in [2]. In a previous paper [11] we proved theoretically that interval arithmetic on the centered form is always more accurate than standard affine arithmetic.

### 3.5 Modified affine arithmetic

We also showed theoretically in [11] that modified affine arithmetic is similar to interval arithmetic on the centered form method, but enhanced by a proper consideration of the properties of even and odd powers of polynomial terms. In fact, the only difference between modified affine arithmetic

and interval arithmetic on the centered form is the last step of calculating  $[\underline{F}, \overline{F}]$ . In modified affine arithmetic  $[\underline{F}, \overline{F}]$  is calculated as

$$\overline{F} = d_0 + \sum_{i=1}^n \left\{ \begin{array}{ll} \max(0, d_i), & \text{if } i \text{ is even} \\ |d_i|, & \text{otherwise} \end{array} \right\},$$

and

$$\underline{F} = d_0 + \sum_{i=1}^n \left\{ \begin{array}{ll} \min(0, d_i), & \text{if } i \text{ is even} \\ -|d_i|, & \text{otherwise} \end{array} \right\},$$

As a result, modified affine arithmetic is always more accurate than interval arithmetic on the centered form.

### 3.6 Recursive Taylor method

In [12] we gave 2D and 3D recursive Taylor methods for evaluating ranges of polynomials in 2 and 3 variables. Here we give the corresponding recursive Taylor method for estimating the range of a univariate polynomial.

To estimate the bounds of a polynomial  $f(x)$  on  $[\underline{x}, \overline{x}]$ , we expand  $f(x)$  at the mid point  $x_0$  of the interval  $[\underline{x}, \overline{x}]$  using Taylor's formula:

$$f(x) = f(x_0) + hf'(x_0) + \frac{1}{2}h^2 f''(x_0 + \theta h)$$

where

$$x \in [\underline{x}, \overline{x}], \quad x_0 = \frac{\underline{x} + \overline{x}}{2}, \quad 0 < \theta < 1, \quad h = x - x_0 \in \left[-\frac{\overline{x} - \underline{x}}{2}, \frac{\overline{x} - \underline{x}}{2}\right] = \frac{\overline{x} - \underline{x}}{2}[-1, 1].$$

Suppose we know interval bounds  $B_{xx}$  of the second derivative  $f''(x)$  of the function  $f(x)$  over the region  $[\underline{x}, \overline{x}]$  such that  $f''(x) \in B_{xx}$ . Let  $x_1 = (\overline{x} - \underline{x})/2$ . Then the bounds  $[\underline{f}, \overline{f}]$  of  $f(x)$  over the region  $[\underline{x}, \overline{x}]$  can be expressed as

$$[\underline{f}, \overline{f}] = f(x_0) + x_1 f'(x_0)[-1, 1] + \frac{1}{2}x_1^2 B_{xx}[-1, 1].$$

When  $f(x)$  is a polynomial, the second derivative  $f''(x)$  is itself also a polynomial with a lower degree. Therefore we can use a recursive technique to estimate the bound of the second derivative, as given by the algorithm in Figure 2.

```

Bound ( $f, \underline{x}, \overline{x}$ ):
IF  $f \equiv c$  (a constant) RETURN Interval $[c, c]$ 
ELSE
 $x_0 = (\underline{x} + \overline{x})/2$ ;
 $x_1 = (\overline{x} - \underline{x})/2$ ;
 $[\underline{f}, \overline{f}] = f(x_0) + x_1 f'(x_0)[-1, 1] + \frac{1}{2}x_1^2 [0, 1] \text{Bound}(f'', \underline{x}, \overline{x})$ ;
RETURN Interval $[\underline{f}, \overline{f}]$ .

```

Figure 2: Recursive Taylor algorithm for univariate polynomial bounding

We first test if  $f$  is a constant, and if so terminates the recursion—the bound on a constant can be trivially computed. Recursion could also be stopped one step earlier, as it is easy to compute exact bounds for linear functions.

As an aside, we note that only in the case in which  $f$  is a polynomial can we guarantee that such recursion will terminate. For polynomials, successive differentiation must eventually result in a constant, which is not true for other functions.

The above proposed recursive Taylor method can be seen as a particular version of the first order Taylor form method [8]. The particularity comes in the way the Taylor form remainder interval is calculated using an automatic recursive technique.

Example	Surface	Polynomial
1	Sphere	$x^2 + y^2 + z^2 - 1$
2	Drop	$4x^2 + 4y^2 - 1 + 2z - 2z^3 + z^4$
3	Torus	$(x^2 + y^2 + z^2 + 3)^2 - 16(x^2 + y^2)$
4	Double Torus	$16x^4 - 32x^6 - 8x^2y^2 + 16x^8 + 8x^4y^2 + y^4 + z^2 - 0.25$
5	Six peak	$(3x^2 - y^2)^2y^2 - (x^2 + y^2)^4 - z$
6	Mitchell	$4(x^4 + (y^2 + z^2)^2) + 17x^2(y^2 + z^2) - 20(x^2 + y^2 + z^2) + 17$
7	Steiner	$x^2y^2 + y^2z^2 + z^2x^2 + xyz$
8	Kummer	$(x^4 + y^4 + z^4 + 1) - (x^2 + y^2 + z^2 + y^2z^2 + z^2x^2 + x^2y^2)$
9	Cusp	$z^3 + xz + y$
10	Blending	$(x^2 + y^2 - 4)(x^2 + z^2 - 4)(y^2 + z^2 - 4) - 4.0078$
11	Heart	$(2x^2 + y^2 + z^2 - 1)^3 - 0.1x^2z^3 - y^2z^3$
12	Cyclide	$90000(x^4 + y^4 + z^4) + 180000(x^2y^2 + x^2z^2 + y^2z^2) - 55000x^2 - 45000y^2 + 12600z^2 + 15600x - 459$

Table 1: Polynomial equations of the algebraic surfaces used for tests

## 4 Comparison of Interval Methods by Examples

In this Section we use various algebraic surfaces to explore experimentally what happens when the above five methods are applied to ray-casting each algebraic surface. We used *Visual C++ 6.0* and Windows XP on a laptop computer with a mobile Pentium 4 CPU 3.20 GHz and 512MB RAM for all the tests.

We rendered 12 carefully chosen algebraic surfaces listed in Table 1 to compare the speed of the five methods. Each example surface is represented by a trivariate polynomial equation  $h(x, y, z) = 0$ . Examples 1–7 were chosen from [3] (the coefficients for Example 3 are different from the ones in [3] which we believe were given incorrectly), Examples 8 and 9 were chosen from [5], Example 10 was chosen from [1], Example 11 was chosen from [15], and Example 12 was chosen from [12].

To compare the efficiency of the five methods, two quantities were measured for each example:

- The total CPU time used in ray-surface intersection calculations: the lower the better.
- The total number of subdivisions (bisections) involved in finding the first intersection point of ray and surface: the lower the better.

The CPU times taken are shown in Table 2 and the number of subdivisions for each method are recorded in Table 3 (rounded to the nearest 1000). In certain cases the AA method failed to produce a result due to memory overflow: these cases are indicated by symbol \*.

For reasons of space we only show here graphical outputs produced for the first four examples by the recursive Taylor method in Figures 3–6.

From Tables 2 and 3 we can clearly see that the performance of these interval methods in ray casting algebraic surface are generally in a sequence of RT, MAA, IAC, IAP, and AA. RT is always better than MAA, while MAA is always better than IAC. IAC is often better than IAP (although not always: see Examples 2, 5, 7, and 10), and IAP is always better than AA. We also notice that while usually the performance of IAP is poor, sometimes it gives very good results (see Example 10), if the polynomial form is well suited to standard interval arithmetic. However IAP is essentially an unstable method (see also Examples 4 and 11).

It is rather surprising how poor the performance of AA is, which is not in accordance with the test results reported in [3]. This was probably due to the particular implementation used, although for fairness and objectiveness we used the standard C++ affine arithmetic library (libaa) obtained from the Internet, and we did not intentionally optimize the C++ code of any particular method out of the five methods.

	RT	MAA	IAC	IAP	AA
Sphere	1	1	1	2	42
Drop	2	3	4	3	82
Torus	4	5	5	15	160
Double Torus	15	37	46	58	*
Six Peak	13	33	40	19	731
Mitchell	4	5	5	25	260
Steiner	4	5	6	5	168
Kummer	4	5	5	21	230
Cusp	1	2	2	3	53
Blending	10	14	15	9	*
Heart	11	19	22	89	*
Cyclide	4	5	6	8	288

Table 2: Times taken for each method, in seconds

	RT	MAA	IAC	IAP	AA
Sphere	93	93	117	86	132
Drop	101	103	135	99	147
Torus	164	165	170	273	184
Double Torus	372	388	481	1135	*
Six Peak	297	323	400	281	440
Mitchell	176	180	207	430	254
Steiner	164	171	225	142	264
Kummer	180	183	201	391	243
Cusp	135	134	137	222	211
Blending	252	236	253	150	*
Heart	245	277	342	600	*
Cyclide	153	160	214	115	237

Table 3: Thousands of subdivisions required for each method



Figure 3: Example 1–Sphere.

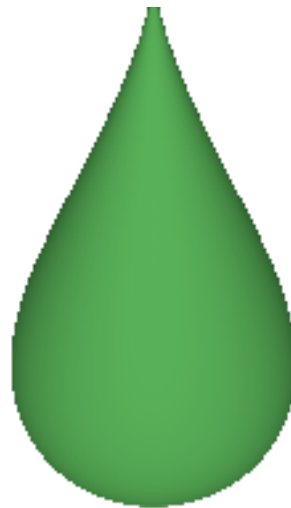


Figure 4: Example 2–Drop.

## 5 Conclusions

The above test results show that in general terms, but not without exception, the performances of the five interval methods when used for ray-casting algebraic surfaces are ranked from best to worst in a sequence of the recursive Taylor method, the modified affine arithmetic method, the interval arithmetic on centered form method, the interval arithmetic on power form method, and the affine arithmetic method. We thus conclude, both on grounds of its simplicity and efficiency, the recursive Taylor method is generally to be favoured compared to other range methods for ray-casting algebraic surfaces.

## References

- [1] Balsys, R.J., Suffern, K.G. “Visualisation of implicit surfaces”. *Computers & Graphics*, 25: 89–107, 2001.
- [2] Comba, J.L.D., Stolfi, J. “Affine arithmetic and its applications to computer graphics”. *Anais do VII SIBGRAPI*, 9–18, 1993.



Figure 5: Example 3–Torus.



Figure 6: Example 4–Double torus.

- [3] 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, 65–71, 1999.
- [4] Duff, T. “Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry”. *Computer Graphics*, 26 (2): 131–138, 1992.
- [5] Hanrahan, P. “Ray tracing algebraic surfaces”. *Computer Graphics*, 17 (3): 83–90, 1983.
- [6] Mitchell, D.P. “Robust ray intersection with interval arithmetic”. *Proceedings of Graphics Interface*, 68–74, 1990.
- [7] Moore, R.E. “Interval Analysis”. Prentice-Hall, 1966.
- [8] Neumaier, A. “Taylor forms - use and limits”. *Reliable Computing*, 9: 43–79, 2002.
- [9] Ratschek, H., Rokne, J. “Computer Methods for the Range of Functions”. Ellis Horwood, 1984.
- [10] 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): 77–80, 2002.
- [11] Shou, H., Lin, H., Martin, R., Wang, G. “Modified affine arithmetic is more accurate than centered interval arithmetic or affine arithmetic”. *Lecture Notes in Computer Science*, 2768: 355-365, 2003.
- [12] Shou, H., Martin, R., Wang, G., Bowyer, A., Voiculescu, I. “A recursive Taylor method for algebraic curves and surfaces”. In: Dokken, T., Jüttler, B., (Eds.), *Computational Methods for Algebraic Spline Surfaces*, Springer Verlag, 135-154, 2004.
- [13] Suffern, K.G., Fackerell, E.D. “Interval methods in computer graphics”. *Computers & Graphics*, 15 (3): 331–340, 1991.
- [14] Synder, J.M. “Interval analysis for computer graphics”. *Computer Graphics*, 26 (2): 121–130, 1992.
- [15] Taubin, G. “Rasterizing algebraic curves and surfaces”. *IEEE Computer Graphics & Applications*, 14: 14–23, 1994.