A Recursive Taylor Method for Algebraic Curves and Surfaces

Huahao Shou[1,2], Ralph Martin[3], Guojin Wang[1], Adrian Bowyer[4], and Irina Voiculescu[5]

[1] State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou, China
[2] Department of Applied Mathematics, Zhejiang University of Technology, Hangzhou, China
[3] School of Computer Science, Cardiff University, Cardiff, UK
[4] Department of Mechanical Engineering, University of Bath, Bath, UK
[5] Computing Laboratory, Oxford University, Oxford, UK

Abstract. This paper examines recursive Taylor methods for multivariate polynomial evaluation over an interval, in the context of algebraic curve and surface plotting as a particular application representative of similar problems in CAGD. The modified affine arithmetic method (MAA), previously shown to be one of the best methods for polynomial evaluation over an interval, is used as a benchmark; experimental results show that a second order recursive Taylor method (i) achieves the same or better graphical quality compared to MAA when used for plotting, and (ii) needs fewer arithmetic operations in many cases. Furthermore, this method is simple and very easy to implement. We also consider which order of Taylor method is best to use, and propose that second order Taylor expansion is generally best. Finally, we briefly examine theoretically the relation between the Taylor method and the MAA method.

1 Introduction

The aim of range analysis is to find the range of a function (usually a polynomial) in one or several variables over an input interval. In practice, finding an exact range is difficult, and it is more usual to find a range which includes the actual range. Information about the range of a function f, and related functions such as its partial derivatives, inverse, etc. are of considerable interest to people working in the fields of numerical and functional analysis, differential equations, linear algebra, approximation and optimization theory and other disciplines [7].

Range analysis has many important applications in CAGD and computer graphics, including the plotting and localisation of implicit curves and surfaces. Implicit surfaces are of direct use, for example, in CSG solid modelling, while implicit curves can be used to represent the intersection of two parametric surfaces, or the silhouette edges of a parametric surface with respect to a given view [10]. Many other geometric operations can also be performed by finding the simultaneous solution of a set of non-linear equations in several variables, and range analysis provides a means of localising such solutions [6]. Both as an interesting example in its own right, and as a *representative* problem, we thus consider in this paper the problem of solving $f(x, y) = 0$ in a rectangle or $f(x, y, z) = 0$ in a cuboid, and more particularly the problem of plotting this curve or surface into a set of pixels or voxels. Clearly, for other problems, e.g. finding the intersection of two surfaces, producing a pixel or voxel grid may not be appropriate, but our overall methodology and conclusions concerning localisation of implicit curves and surfaces remain valid.

Parametric curves or surfaces are very easy to plot. On the other hand, implicit curves or surfaces can not be plotted so readily. Implicit curve or surface plotting methods can be classified into two categories. The first are continuation methods [2–4], which are efficient. They find one or more seed cells (pixels or voxels) on a curve or a surface, and then trace the curve or surface continuously through appropriate adjacent cells—only cells containing the curve or surface are visited. However, continuation methods have one fundamental difficulty, that of finding a *complete* set of initial seed cells.

Subdivision methods [5, 8, 10–14] make up the second approach. These methods start with the whole plotting region itself as an initial cell. If a cell can be proven to be empty, it is discarded; otherwise it is subdivided into smaller cells, which are then visited recursively, until the cells reach pixel size. All pixels which contain the curve are thus guaranteed to be retained. In this way large portions of the plotting region can be discarded quickly and reliably at an early stage, leading to an efficient method. Such methods are generally based on ideas from interval arithmetic.

When $f(x, y)$ is a polynomial in two variables x and y, the curve is an *algebraic* curve. Similarly when $f(x, y, z)$ is a polynomial in three variables x, y and z, the surface is an *algebraic* surface. Algebraic curves or surfaces are a rich family, with several plotting methods [5, 8, 13] that exploit the properties of polynomials.

Taubin's method [13] is well known; we have shown in [5] that Taubin's method is equivalent to performing interval arithmetic on centered forms but without consideration of the even or odd properties of powers of polynomial terms. We have further shown that interval arithmetic on centered forms method is less accurate than a modified affine arithmetic method (MAA) which does take into consideration the even or odd properties.

In this paper we propose the use of a *recursive* Taylor method for function range evaluation and use it to plot algebraic curves and surfaces. We combine it with a point sampling technique and a subpixel (or subvoxel) technique to improve the results.

In our previous papers [5, 8] we showed that the modified affine arithmetic method is one of the best methods for polynomial evaluation over an interval, for use in recursive subdivision methods for plotting algebraic curves—we thus compare the Taylor method with that method. Our test results show that, when used for plotting algebraic curves and surfaces at a given resolution, the recursive Taylor method can give same or better graphical accuracy as the MAA method, and needs fewer arithmetic operations in most cases. Furthermore, this recursive Taylor method is simple and very easy to implement.

We also consider which order Taylor method to use, and show that 2nd order Taylor expansion seems to be best for general use.

Finally we examine theoretically the relation between the recursive Taylor method and the modified affine arithmetic method.

As noted above, the recursive Taylor technique presented in this paper is a general efficient method for computing bounds on a polynomial: its use here for algebraic curve and surface drawing is just an example application. The recursive Taylor method presented in this paper can be easily generalized to an arbitrary number of dimensions.

2 The Subdivision Algorithm

Subdivision algorithms for plotting implicit curves and implicit surfaces have much in common. We mainly focus on the case of plane implicit curves in this section.

In the following we use the standard notation that an interval A represents a range of real values between $\underline{a}$ and $\overline{a}$ such that $\underline{a} < \overline{a}$ and is written $[\underline{a}, \overline{a}]$.

The main idea of subdivision algorithms [11] for plotting implicit curves over a rectangular array of pixels is to consider various regions, initially the whole plotting region, $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$, and to estimate bounds $[\underline{f}, \overline{f}]$ guaranteed to contain all values of $f(x, y)$ over this region. This is done using some range analysis method to estimate the range of the function. If $0 \notin [\underline{f}, \overline{f}]$, this means that the curve cannot pass through region, which therefore can be discarded. Otherwise the region is subdivided horizontally and vertically at its mid point into four sub-regions, and the pieces are considered in turn. The process stops when any region not yet discarded reaches pixel size.

In a basic version of the algorithm, we may just plot this pixel as if it did contain the curve. This can result in a “fat” curve if the bounds on the function obtained by range analysis method are too conservative, i.e. extra pixels which are actually not on the curve are plotted. Later, we will consider how to process the pixel-sized regions further to remove some, but not all, of the extraneous pixels. The basic procedure is summarized in fig. 1.

```
PROCEDURE Plot_Curve($\underline{x}, \overline{x}, \underline{y}, \overline{y}$):
$[\underline{f}, \overline{f}] =$ Bound($f, \underline{x}, \overline{x}, \underline{y}, \overline{y}$);
IF $\underline{f} \leq 0 \leq \overline{f}$ THEN
  IF $\overline{x} - \underline{x} \leq$ Pixel_size AND $\overline{y} - \underline{y} \leq$ Pixel_size THEN
    Plot_Pixel($\underline{x}, \overline{x}, \underline{y}, \overline{y}$)
  ELSE Subdivide($\underline{x}, \overline{x}, \underline{y}, \overline{y}$).

PROCEDURE Subdivide($\underline{x}, \overline{x}, \underline{y}, \overline{y}$):
$x_0 = (\underline{x} + \overline{x})/2$;
$y_0 = (\underline{y} + \overline{y})/2$;
Plot_Curve($\underline{x}, x_0, \underline{y}, y_0$);
Plot_Curve($\underline{x}, x_0, y_0, \overline{y}$);
Plot_Curve($x_0, \overline{x}, y_0, \overline{y}$);
Plot_Curve($x_0, \overline{x}, \underline{y}, y_0$).
```

Fig. 1. Subdivision algorithm for curve plotting

The key step in subdivision algorithms of this type is to estimate the bounds $[\underline{f}, \overline{f}]$ on $f(x, y)$ over the region $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$; this is done using some range analysis method. Different range analysis methods for computing the bounds have different effects on accuracy and efficiency of the plotting algorithm [5]. Generally, the more accurate the estimate is, the better the graphical result will be, and also less subdivision will be required. However, more accurate estimates usually need more arithmetic operations, which reduces the efficiency of the plotting algorithm. Obviously, accuracy and efficiency are to some extent trade-offs. In the next section we will present a Taylor method for computing these bounds.

In order to reduce the uncertainties associated with the regions remaining at pixel level, which may or may not contain the curve, as noted above, we use two further techniques. Point sampling [12] is done for regions of pixel size by evaluating the values of $f(x, y)$ at the four corner points of the pixel. If they do not all have the same sign (or zero), then the pixel must be include the curve (as f is a continuous function); otherwise, the pixel may or may not be on the curve. Thus, after point sampling, all pixels in the plotting region belong to one of three classes: (i) pixels discarded by the basic subdivision method, which are surely not on the curve, (ii) pixels accepted by the point sampling technique, which are surely on the curve, and (iii) pixels whose status is still not clear, and may or may not be on the curve. We now further attempt to discard as many pixels as possible in the third class. To this end we use a subpixel technique [14]. We subdivide pixels in the third category into four subpixels. If all four subpixels can be discarded by the range method, we discard this pixel, otherwise we keep the pixel.

A major advantage of the subdivision algorithm presented above is that it finds *all* points on the curve, and can handle singularities with no special processing. Thus, it can handle problems where continuation methods may typically fail, including curves with multiple components, cusps, self-intersections, touching components, and isolated points.

The subdivision algorithm for plotting implicit surfaces is a direct generalisation to three variables of the planar implicit curve algorithm. Plotting implicit space curve cases can also readily be done by finding regions *simultaneously* containing zeros of *two* implicit functions in three variables.

3 Taylor Method for Bounds

Constructing the natural inclusion function [10] giving the exact range of a function over an interval is often not easy, and may be impossible for general functions $f(x, y)$. Here we use a simple Taylor method [1] for computing bounds of $f(x, y)$ over $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$, which can be combined with point sampling and subpixel techniques to solve the implicit curve plotting problem in a reliable, accurate and efficient way. For now, we assume the choice of a *second order* Taylor method, but we will return to the choice of order later. Suppose $f(x, y)$ has continuous second derivatives on $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$. In many practical applications in CAGD and computer graphics, the functions encountered satisfy this condition, at least piecewise. To estimate the bound of $f(x, y)$ on $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$, we expand $f(x, y)$ at the mid point (x_0, y_0) of the region $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$ using

Taylor's formula:

$$f(x,y) = f(x_0,y_0) + hf_x(x_0,y_0) + kf_y(x_0,y_0) + \frac{1}{2}h^2 f_{xx}(x_0+\theta h, y_0+\theta k)$$
$$+ \frac{1}{2}k^2 f_{yy}(x_0+\theta h, y_0+\theta k) + hkf_{xy}(x_0+\theta h, y_0+\theta k),$$

where

$$(x,y) \in [\underline{x},\overline{x}] \times [\underline{y},\overline{y}], \quad x_0 = \frac{\underline{x}+\overline{x}}{2}, y_0 = \frac{\underline{y}+\overline{y}}{2}, \quad 0 < \theta < 1,$$
$$h = x - x_0 \in [-\frac{\overline{x}-\underline{x}}{2}, \frac{\overline{x}-\underline{x}}{2}] = \frac{\overline{x}-\underline{x}}{2}[-1,1],$$
$$k = y - y_0 \in [-\frac{\overline{y}-\underline{y}}{2}, \frac{\overline{y}-\underline{y}}{2}] = \frac{\overline{y}-\underline{y}}{2}[-1,1].$$

Suppose we know the interval bounds B_{xx}, B_{yy}, B_{xy} of the three second derivatives $f_{xx}(x,y)$, $f_{yy}(x,y)$, $f_{xy}(x,y)$ of the function $f(x,y)$ over the region $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}]$ such that $f_{xx}(x,y) \in B_{xx}$, $f_{yy}(x,y) \in B_{yy}$, $f_{xy}(x,y) \in B_{xy}$. Let $x_1 = (\overline{x}-\underline{x})/2$, $y_1 = (\overline{y}-\underline{y})/2$. Then the bounds $[\underline{f},\overline{f}]$ of $f(x,y)$ over the region $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}]$ can be expressed as

$$[\underline{f},\overline{f}] = f(x_0,y_0) + x_1 f_x(x_0,y_0)[-1,1] + y_1 f_y(x_0,y_0)[-1,1]$$
$$+ \frac{1}{2}x_1^2 B_{xx}[-1,1] + \frac{1}{2}y_1^2 B_{yy}[-1,1] + x_1 y_1 B_{xy}[-1,1].$$

(To apply interval computation to the above formula, real numbers are converted where necessary to intervals with *equal* lower and upper bounds.)

The main potential limitation of this method is that we need estimates for the bounds B_{xx}, B_{yy}, B_{xy} of the three second derivatives $f_{xx}(x,y)$, $f_{yy}(x,y)$, $f_{xy}(x,y)$ of $f(x,y)$ on the region $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}]$. (Note that the first derivatives required need only be computed at a specific point, and thus can readily be found.) For general implicit curves, finding bounds on the second derivatives is a difficult problem. However, as we show in the next section, they can be readily computed for algebraic curves.

Similarly, for surface plotting, to estimate the bound of $f(x,y,z)$ on $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}] \times [\underline{z},\overline{z}]$, we may expand $f(x,y,z)$ at the mid point (x_0,y_0,z_0) of the region $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}] \times [\underline{z},\overline{z}]$ using Taylor's formula:

$$f(x,y,z) = f(x_0,y_0,z_0) + hf_x(x_0,y_0,z_0) + kf_y(x_0,y_0,z_0) + lf_z(x_0,y_0,z_0)$$
$$+\frac{1}{2}h^2 f_{xx}(x_0+\theta h, y_0+\theta k, z_0+\theta l) + \frac{1}{2}k^2 f_{yy}(x_0+\theta h, y_0+\theta k, z_0+\theta l)$$
$$+\frac{1}{2}l^2 f_{zz}(x_0+\theta h, y_0+\theta k, z_0+\theta l) + hkf_{xy}(x_0+\theta h, y_0+\theta k, z_0+\theta l)$$
$$+hlf_{xz}(x_0+\theta h, y_0+\theta k, z_0+\theta l) + klf_{yz}(x_0+\theta h, y_0+\theta k, z_0+\theta l)$$

where

$$(x,y,z) \in [\underline{x},\overline{x}] \times [\underline{y},\overline{y}] \times [\underline{z},\overline{z}], \quad x_0 = \frac{\underline{x}+\overline{x}}{2}, y_0 = \frac{\underline{y}+\overline{y}}{2}, z_0 = \frac{\underline{z}+\overline{z}}{2}, \ 0 < \theta < 1,$$

$$
\begin{aligned}
h &= x - x_0 \in [-\frac{\overline{x}-\underline{x}}{2}, \frac{\overline{x}-\underline{x}}{2}] = \frac{\overline{x}-\underline{x}}{2}[-1,1],\\
k &= y - y_0 \in [-\frac{\overline{y}-\underline{y}}{2}, \frac{\overline{y}-\underline{y}}{2}] = \frac{\overline{y}-\underline{y}}{2}[-1,1].\\
l &= z - z_0 \in [-\frac{\overline{z}-\underline{z}}{2}, \frac{\overline{z}-\underline{z}}{2}] = \frac{\overline{z}-\underline{z}}{2}[-1,1].
\end{aligned}
$$

Suppose we know the interval bounds $B_{xx}, B_{yy}, B_{zz}, B_{xy}, B_{xz}, B_{yz}$ of the six second derivatives $f_{xx}(x,y,z), f_{yy}(x,y,z), f_{zz}(x,y,z), f_{xy}(x,y,z), f_{xz}(x,y,z), f_{yz}(x,y,z)$ of the function $f(x,y,z)$ over the region $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}] \times [\underline{z},\overline{z}]$ such that $f_{xx}(x,y,z) \in B_{xx}$, $f_{yy}(x,y,z) \in B_{yy}$, $f_{zz}(x,y,z) \in B_{zz}$, $f_{xy}(x,y,z) \in B_{xy}$, $f_{xz}(x,y,z) \in B_{xz}$, $f_{yz}(x,y,z) \in B_{yz}$. Let $x_1 = (\overline{x}-\underline{x})/2$, $y_1 = (\overline{y}-\underline{y})/2$,, $z_1 = (\overline{z}-\underline{z})/2$,. Then the bounds $[\underline{F}, \overline{F}]$ of $f(x,y,z)$ over the region $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}] \times [\underline{z},\overline{z}]$ can be expressed as

$$
\begin{aligned}
[\underline{F},\ \overline{F}] &= f(x_0,y_0,z_0) + x_1 f_x(x_0,y_0,z_0)[-1,1] + y_1 f_y(x_0,y_0,z_0)[-1,1]\\
&+ z_1 f_z(x_0,y_0,z_0)[-1,1] + \frac{1}{2}x_1^2 B_{xx}[-1,1] + \frac{1}{2}y_1^2 B_{yy}[-1,1] + \frac{1}{2}z_1^2 B_{zz}[-1,1]\\
&+ x_1 y_1 B_{xy}[-1,1] + x_1 z_1 B_{xz}[-1,1] + y_1 z_1 B_{yz}[-1,1].
\end{aligned}
$$

As above, again we need estimates for the bounds B_{xx}, B_{yy}, B_{zz}, B_{xy}, B_{xz}, B_{yz}.

4 Finding Bounds on Derivatives

When $f(x,y) = 0$ represents an *algebraic* curve, $f(x,y)$ is a *polynomial* function of two variables. In this case the three second derivatives $f_{xx}(x,y)$, $f_{yy}(x,y)$, $f_{xy}(x,y)$ are themselves also polynomials in two variables with lower degrees in x or y or both. Therefore we can use a recursive technique to estimate the bounds of the second derivatives, as given by the algorithm in fig. 2. Here, "`IF` $f \equiv c$ `RETURN Interval`$[c,c]$"

```
Bound($f, \underline{x}, \overline{x}, \underline{y}, \overline{y}$) :
IF $f \equiv c$ RETURN Interval$[c,c]$
ELSE
    $x_0 = (\underline{x}+\overline{x})/2$;   $y_0 = (\underline{y}+\overline{y})/2$;   $x_1 = (\overline{x}-\underline{x})/2$;   $y_1 = (\overline{y}-\underline{y})/2$;
    $[\underline{f},\overline{f}] = f(x_0,y_0) + x_1 f_x(x_0,y_0)[-1,1] + y_1 f_y(x_0,y_0)[-1,1]$
            $+\frac{1}{2}x_1^2[0,1]$Bound($f_{xx}, \underline{x}, \overline{x}, \underline{y}, \overline{y}$) $+ \frac{1}{2}y_1^2[0,1]$Bound($f_{yy}, \underline{x}, \overline{x}, \underline{y}, \overline{y}$)
            $+x_1 y_1[-1,1]$Bound($f_{xy}, \underline{x}, \overline{x}, \underline{y}, \overline{y}$);
    RETURN Interval$[\underline{f},\overline{f}]$.
```

Fig. 2. Recursive Taylor algorithm for polynomial bounding

tests 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.)

Note that only in the case that 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.

A similar recursive technique can be used for trivariate polynomials.

5 Examples

In the above sections we proposed a recursive Taylor method combined with point sampling and a subpixel technique for plotting algebraic curves and surfaces. In this section we give some examples demonstrating the accuracy and efficiency of these methods.

Most of the examples we give involve low degree polynomials. While it is conceivable that somewhat different conclusions might be drawn for the cases of higher degree polynomials, other tests we have done on further higher degree polynomials support the conclusions here. Furthermore, in most CAGD applications, the polynomials used are generally of a low degree, justifying our choice of low degree test cases.

5.1 Algebraic Curves

Examples 1 to 10 are the same examples for plotting algebraic curves given in a recent survey of methods [5], with plotting region $[0,1]\times[0,1]$ and resolution 256×256 pixels. They were designed to test the efficiency and accuracy of range evaluation methods on a variety of problem cases, including curves with cusps, self-intersections, closely adjacent loops, and so on.

The corresponding figures produced by the new recursive Taylor (RT) method (including the use of point sampling and subpixel techniques, denoted RT++) are shown in Figures 3 to 12. The survey [5] showed that the modified affine arithmetic method (MAA) is one of the best methods for plotting algebraic curves. Therefore we have compared the recursive Taylor method with the MAA method. A detailed quantitative comparison of the MAA and RT methods, and also their variants MAA++ and RT++ which include point sampling and subpixel techniques, is given in Table 1 (*left*) for these examples.

Table 1 (*left*) shows, for each example, how many pixels are plotted by the different methods (the fewer, the more accurately the method has found the curve), the number of subdivisions used in the computation (the fewer, the better, as less stack operation overheads result), and the number of addition and multiplication operations used overall (the lower, the better).

The recorded number of additions and multiplications in Table 1 (*left*) does not include the arithmetic operations used to differentiate the polynomial. An implementation of the recursive Taylor method should calculate all necessary coefficients of the derivatives of the polynomial just once at the beginning, and store them in an array, to avoid differentiation of the polynomial during the subdivision process every time a derivative is needed. The number of arithmetic operations used to differentiate the polynomial once only is relatively small and can be neglected.

From Table 1 (*left*) we can see that in one case out of ten (Example 4), the recursive Taylor method produced better graphical quality than the modified affine arithmetic method (fewer pixels were plotted). The corresponding graphical output for the RT method is shown in fig. 14, where 801 pixels were plotted, and for the MAA method in fig. 13, where 816 pixels were plotted. (These two figures only differ in the lower left corner). In the other nine test cases the recursive Taylor method produced the same graphical quality as the modified affine arithmetic method.

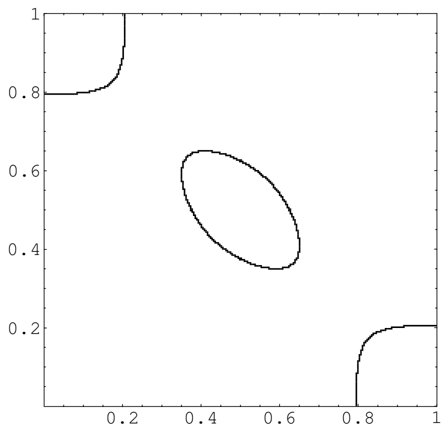

Fig. 3. Example 1: $\frac{15}{4} + 8x - 16x^2 + 8y - 112xy + 128x^2y - 16y^2 + 128xy^2 - 128x^2y^2 = 0$, plotted by RT++ method (522 pixels)

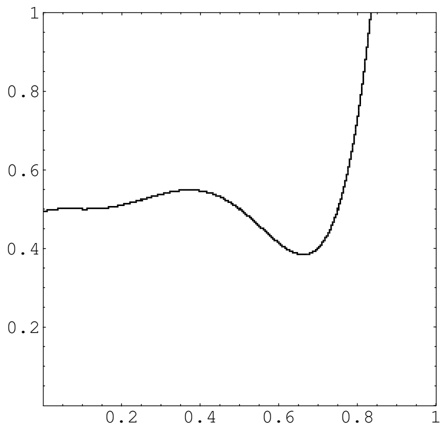

Fig. 4. Example 2: $20160x^5 - 30176x^4 + 14156x^3 - 2344x^2 + 151x + 237 - 480y = 0$, plotted by RT++ method (432 pixels)

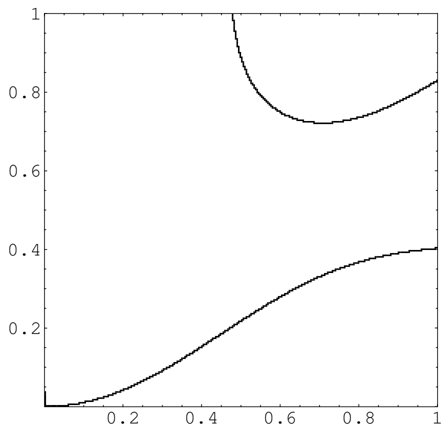

Fig. 5. Example 3: $0.945xy - 9.43214x^2y^3 + 7.4554x^3y^2 + y^4 - x^3 = 0$, plotted by RT++ method (601 pixels)

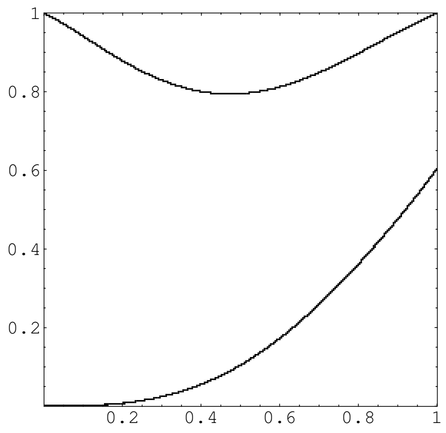

Fig. 6. Example 4: $x^9 - x^7y + 3x^2y^6 - y^3 + y^5 + y^4x - 4y^4x^3 = 0$, plotted by RT++ method (774 pixels)

In seven out of ten cases, the recursive Taylor method needed fewer arithmetic operations in total (the number of additions plus the number of multiplications) than the modified affine arithmetic method (Examples 2,4,6,7,8,9,10). In Examples 2,6,9,10 the number of arithmetic operations needed by the recursive Taylor method was much fewer than (less than half of) those for the modified affine arithmetic method. Although the recursive Taylor method needed more arithmetic operations than the modified affine

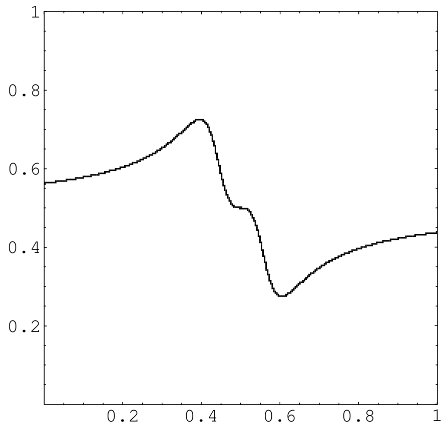

Fig. 7. Example 5. $-\frac{1801}{50}+280x-816x^2+1056x^3-512x^4+\frac{1601}{25}y-512xy+1536x^2y-2048x^3y+1024x^4y=0$, plotted by RT++ method (456 pixels)

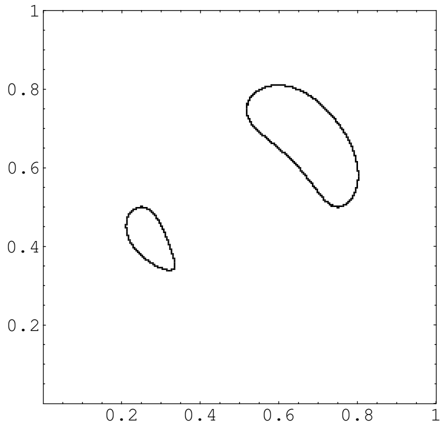

Fig. 8. Example 6: $\frac{601}{9}-\frac{872}{3}x+544x^2-512x^3+256x^4-\frac{2728}{9}y+\frac{2384}{3}xy-768x^2y+\frac{5104}{9}y^2-\frac{2432}{3}xy^2+768x^2y^2-512y^3+256y^4=0$, plotted by RT++ method (456 pixels)

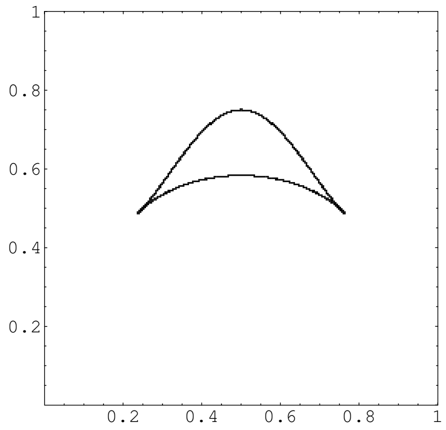

Fig. 9. Example 7: $-13+32x-288x^2+512x^3-256x^4+64y-112y^2+256xy^2-256x^2y^2=0$, plotted by RT++ method (460 pixels)

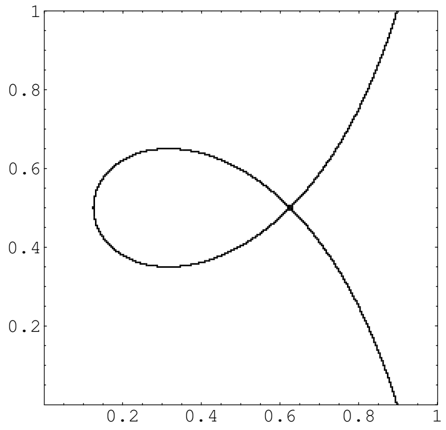

Fig. 10. Example 8: $-\frac{169}{64}+\frac{51}{8}x-11x^2+8x^3+9y-8xy-9y^2+8xy^2=0$, plotted by RT++ method (808 pixels)

arithmetic method for Examples 1,3,5, we note that the numbers of arithmetic operations needed by both methods for these examples were very similar.

One minor disadvantage of the recursive Taylor method is that it often needs a few more recursive operations than MAA.

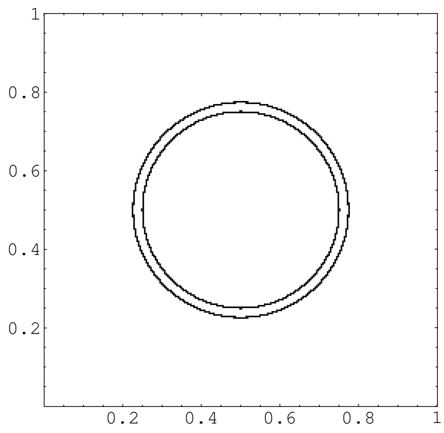

Fig. 11. Example 9: $47.6 - 220.8x + 476.8x^2 - 512x^3 + 256x^4 - 220.8y + 512xy - 512x^2y + 476.8y^2 - 512xy^2 + 512x^2y^2 - 512y^3 + 256y^4 = 0$, plotted by RT++ method (1088 pixels)

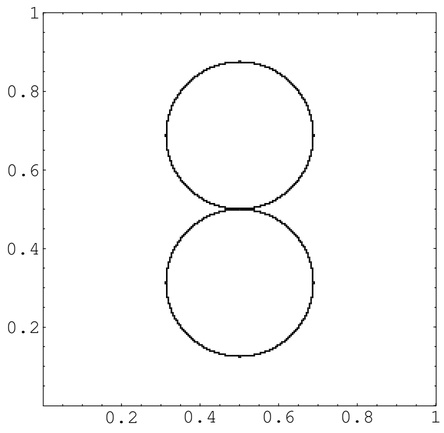

Fig. 12. Example 10: $\frac{55}{256} - x + 2x^2 - 2x^3 + x^4 - \frac{55}{64}y + 2xy - 2x^2y + \frac{119}{64}y^2 - 2xy^2 + 2x^2y^2 - 2y^3 + y^4 = 0$, plotted by RT++ method (772 pixels)

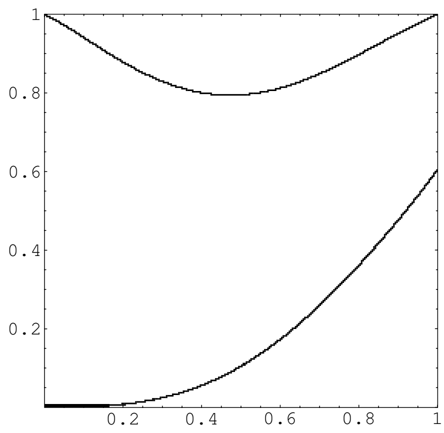

Fig. 13. Example 4, plotted by the MAA method (816 pixels)

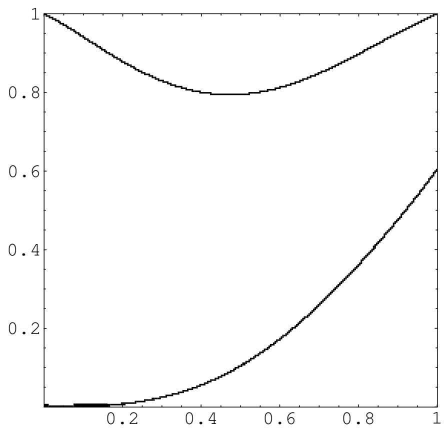

Fig. 14. Example 4, plotted by the RT method (801 pixels)

Point sampling and subpixel techniques further improved the graphical quality of RT and MAA methods, especially for Examples 4,7,9 where the improvements are significant. However, for Examples 1,2,3,5,6,8,10 the improvements only affected a few pixels and insignificant. Of course, the price to pay for these improvements is an increase in arithmetic operations: every pixel which cannot be discarded by the basic subdivision process needs to be examined further. We can however see from Table 1 (*left*) that the increased number of arithmetic operations is not greatly significant. This is

Table 1. Comparison of 2D (*left*) and 3D (*right*) RT, MAA, RT++, MAA++ methods

Ex.	Method	Pixels plotted	Subdi-visions	Addi-tions	Multipli-cations
1	RT	526	571	415688	343892
1	MAA	526	563	404262	171226
1	RT++	522	575	436316	385080
1	MAA++	522	567	421448	207820
2	RT	433	461	241581	205717
2	MAA	433	459	601510	407812
2	RT++	432	462	253193	234577
2	MAA++	432	460	611148	434354
3	RT	608	637	1116344	936757
3	MAA	608	634	1178329	646933
3	RT++	601	653	1143206	992682
3	MAA++	601	650	1202312	694836
4	RT	801	845	4662221	4461229
4	MAA	816	857	6773822	6302500
4	RT++	774	876	4844054	4748416
4	MAA++	774	903	7139018	6757864
5	RT	464	627	664231	575815
5	MAA	464	611	599656	339853
5	RT++	456	635	690161	630353
5	MAA++	456	619	621248	387781
6	RT	460	567	442025	414092
6	MAA	460	560	1329630	788830
6	RT++	456	573	469450	478064
6	MAA++	456	566	1362826	853306
7	RT	512	629	445039	386359
7	MAA	512	627	873923	476708
7	RT++	460	719	512886	472534
7	MAA++	460	717	986288	569061
8	RT	818	829	563844	422917
8	MAA	818	827	855337	397078
8	RT++	808	843	595997	476088
8	MAA++	808	841	886530	444873
9	RT	1144	1281	998825	935312
9	MAA	1144	1269	3012696	1787102
9	RT++	1088	1351	1106039	1131219
9	MAA++	1088	1339	3214325	2018571
10	RT	784	849	662153	609761
10	MAA	784	845	2006376	1190110
10	RT++	772	861	710484	710732
10	MAA++	772	857	2068693	1294219

Ex.	Method	Voxels plotted	Subdi-visions	Addi-tions	Multipli-cations
11	RT	1791	592	397403	229152
11	MAA	1791	592	326348	110727
11	RT++	1791	592	432100	278177
11	MAA++	1791	592	361045	159752
12	RT	3992	1353	918367	588609
12	MAA	3992	1353	3289042	1476259
12	RT++	3952	1401	1163930	953372
12	MAA++	3944	1401	3513741	1733406
13	RT	3712	1433	958102	589014
13	MAA	3712	1433	1084217	692199
13	RT++	3712	1433	1023606	713910
13	MAA++	3712	1433	1149721	817095
14	RT	3272	1129	756950	491169
14	MAA	3272	1129	2735177	1231875
14	RT++	3176	1249	1145079	1038878
14	MAA++	3192	1249	3048966	1515888
15	RT	2192	985	4455080	3265689
15	MAA	2144	985	13108130	11792931
15	RT++	1904	1337	5603358	4948007
15	MAA++	1920	1289	16804088	15499281
16	RT	2376	1153	5232146	3831834
16	MAA	2344	1121	14953054	13456863
16	RT++	2148	1497	6291409	5435377
16	MAA++	2104	1433	18908261	17434612
17	RT	5276	1841	7081323	4483139
17	MAA	5256	1837	6948311	5854917
17	RT++	4896	2265	8662097	6658461
17	MAA++	4976	2241	8576707	7662017
18	RT	9424	2865	12975392	9497889
18	MAA	9376	2769	36866234	33149195
18	RT++	7236	5313	21000658	20340451
18	MAA++	7792	5169	64451180	59649509
19	RT	1832	961	4290881	3139995
19	MAA	1816	961	12656417	11259579
19	RT++	1572	1249	5248699	4536725
19	MAA++	1624	1233	15851779	14407101
20	RT	3428	1197	3139078	2100954
20	MAA	3416	1169	3739482	4352652
20	RT++	3288	1425	3913112	3195292
20	MAA++	3288	1385	4474204	5400382

because the RT and MAA methods already provide close to the best possible graphical quality at the given resolution, and thus the numbers of pixels left to be examined further by point sampling and subpixel techniques are relatively small.

5.2 Algebraic Surfaces

We have also experimented with algebraic surface plotting, as outlined below. Note that our main purpose in this paper is to compare our new range analysis method with existing methods, in this case for *localising* the surface to specific regions (voxels). We only use voxel plotting as a *representative* application; the graphical results of surface plotting shown at a resolution of $32 \times 32 \times 32$ are clearly crude. Such an approach is not meant to be a useful surface rendering algorithm in itself. A realistic surface plotting

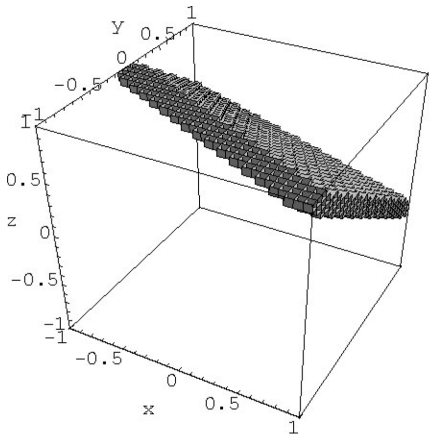

Fig. 15. Example 11: The plane plotted by 3D RT++ method (1791 voxels)

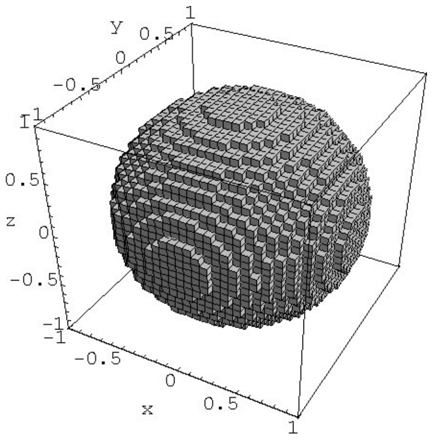

Fig. 16. Example 12: The sphere plotted by 3D RT++ method (3952 voxels)

algorithm would, for example, attempt to find a linear fit to the surface and estimate its normal in each region where the surface has been localised.

Example 11: this plots the plane $f(x, y, z) = x + 2y + 3z - 2$ inside the box box $[-1, 1] \times [-1, 1] \times [-1, 1]$, with resolution $32 \times 32 \times 32$ voxels. Figure 15 shows the plane plotted by the 3D recursive Taylor method using point sampling and subpixel techniques. A total of 1791 voxels were plotted.

Example 12: this plots the sphere $f(x, y, z) = 100x^2 + 100y^2 + 100z^2 - 81$ inside the box $[-1, 1] \times [-1, 1] \times [-1, 1]$, with resolution $32 \times 32 \times 32$ voxels. Figure 16 shows the sphere plotted by the recursive Taylor method using point sampling and subpixel techniques. A total of 3952 voxels were plotted.

Example 13: this plots the cylinder $f(x, y, z) = 100x^2 + 100y^2 - 81$ inside the box $[-1, 1] \times [-1, 1] \times [-1, 1]$, with resolution $32 \times 32 \times 32$ voxels. Figure 17 shows the cylinder plotted by the recursive Taylor method using point sampling and subpixel techniques. A total of 3712 voxels were plotted.

Example 14: this plots the cone $f(x, y, z) = 100x^2 + 100y^2 - 81z^2$ inside the box $[-1, 1] \times [-1, 1] \times [-1, 1]$, with resolution $32 \times 32 \times 32$ voxels. Figure 18 is the cone plotted by the recursive Taylor method using point sampling and subpixel techniques. A total of 3176 voxels were plotted.

Example 15: this plots the torus $f(x, y, z) = 64 - 500x^2 + 625x^4 - 500y^2 + 1250x^2y^2 + 625y^4 + 400z^2 + 1250x^2z^2 + 1250y^2z^2 + 625z^4$ inside the box $[-1, 1] \times [-1, 1] \times [-1, 1]$ with resolution $32 \times 32 \times 32$ voxels. Figure 19 is the torus plotted by the recursive Taylor method using point sampling and subpixel techniques. A total of 1904 voxels were plotted.

Example 16: this plots the cyclide $f(x, y, z) = -459 + 15600x - 55000x^2 + 90000x^4 - 45000y^2 + 180000x^2y^2 + 90000y^4 + 12600z^2 + 180000x^2z^2 + 180000y^2z^2 + 90000z^4$ inside the box $[-1, 1] \times [-1, 1] \times [-1, 1]$, with resolution $32 \times 32 \times 32$ voxels.

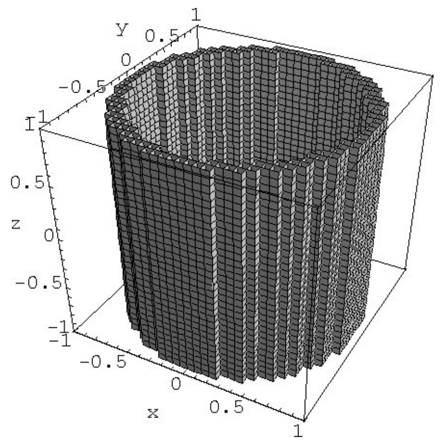

Fig. 17. Example 13: The cylinder plotted by 3D RT++ method (3712 voxels)

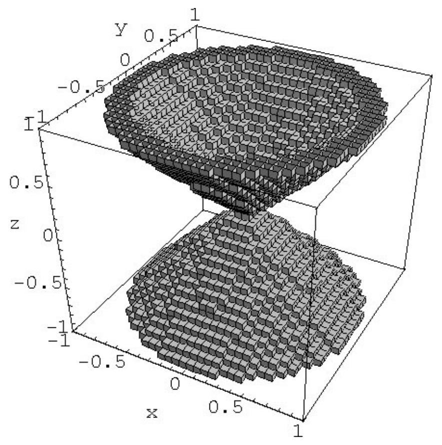

Fig. 18. Example 14: The cone plotted by 3D RT++ method (3176 voxels)

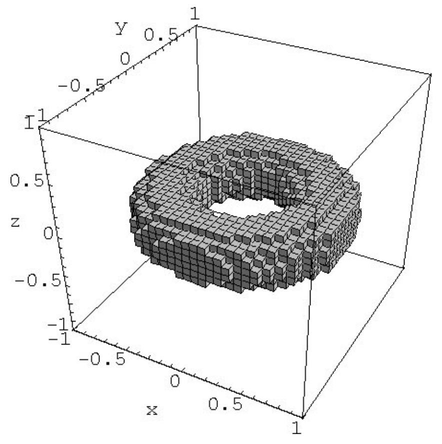

Fig. 19. Example 15: The torus plotted by 3D RT++ method (1904 voxels)

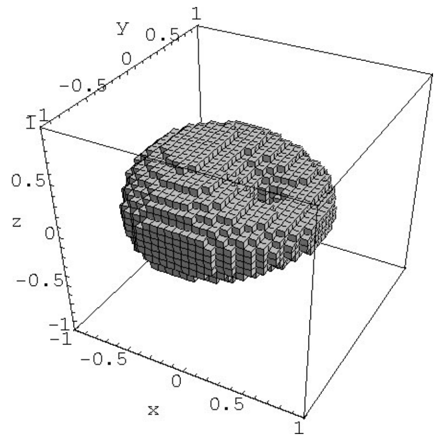

Fig. 20. Example 16: The cyclide plotted by 3D RT++ method (2148 voxels)

Figure 20 is the cyclide plotted by the recursive Taylor method using point sampling and subpixel techniques. A total of 2148 voxels were plotted.

Example 17: this plots a self-intersecting surface $f(x,y,z) = 16 - 32x - 25x^2 + 50x^3 - 25y^2 + 50xy^2 - 25z^2 + 50xz^2$ inside the box $[-1,1] \times [-1,1] \times [-1,1]$, with resolution $32 \times 32 \times 32$ voxels. Figure 21 is the self-intersecting surface plotted by the recursive Taylor method using point sampling and subpixel techniques. A total of 4896 voxels were plotted.

Example 18: this plots a pair of parallel surfaces $f(x,y,z) = 1296 - 3625x^2 + 2500x^4 - 3625y^2 + 5000x^2y^2 + 2500y^4 - 3625z^2 + 5000x^2z^2 + 5000y^2z^2 + 2500z^4$

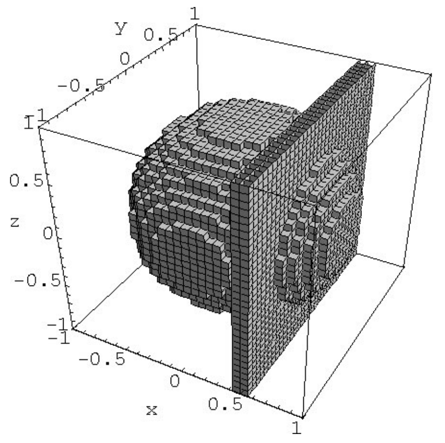

Fig. 21. Example 17: The self-intersecting surface plotted by 3D RT++ method (4896 voxels)

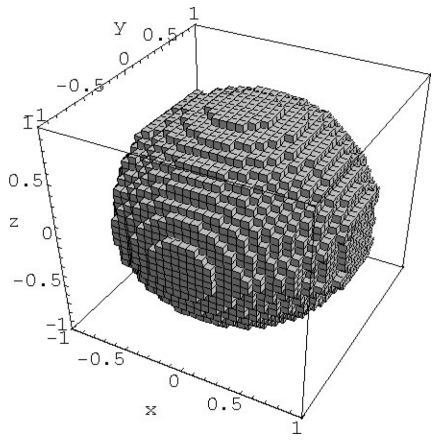

Fig. 22. Example 18: The pair of parallel surfaces plotted by 3D RT++ method (7236 voxels)

inside the box $[-1,1]\times[-1,1]\times[-1,1]$ with resolution $32\times32\times32$ voxels. Figure 22 is the pair of parallel surfaces plotted by the recursive Taylor method using point sampling and subpixel techniques. A total of 7236 voxels were plotted.

Example 19: this plots a pair of just-touching surfaces (two spheres sharing a tangent plane) $f(x,y,z) = -16x^2 + 25x^4 + 50x^2y^2 + 25y^4 + 50x^2z^2 + 50y^2z^2 + 25z^4$ inside the box $[-1,1] \times [-1,1] \times [-1,1]$, with resolution $32 \times 32 \times 32$ voxels. Figure 23 is the pair of tangent spheres plotted by the recursive Taylor method using point sampling and subpixel techniques. A total of 1572 voxels were plotted.

Example 20: this plots a cone-like surface with a line singularity $f(x,y,z) = -1 + 4x - 4x^2 + 2y^2 - 8xy^2 + 8x^2y^2 + 8z^2$ inside the box $[-1,1] \times [-1,1] \times [-1,1]$, with resolution $32 \times 32 \times 32$ voxels. Figure 24 is the cone-like surface plotted by the recursive Taylor method using point sampling and subpixel techniques. A total of 3288 voxels were plotted.

Table 1 (*right*) gives a detailed quantitative comparison for these surface examples of the 3D MAA and 3D RT methods, and also of their improvements which include point sampling and subpixel techniques, 3D MAA++ and 3D RT++.

From Table 1 (*right*) we can see that:

- In 4 out of 10 cases (Examples 11–14) the RT method plotted the same number of voxels as the MAA method. In the other 6 cases (Examples 15–20) the RT method plotted slightly more voxels than the MAA method.
- In 9 out of 10 cases (all but Example 11) the RT method needed fewer arithmetic operations than the MAA method.
- In 5 out of 10 cases (Examples 14,15,17–19) the RT++ method plotted fewer voxels than the MAA++ method. In 3 cases (Examples 11,13,20) the RT++ method plotted the same number of voxels as the MAA++ method. In the other 2 cases (Examples 12,16) the RT++ method plotted slightly more voxels than the MAA++ method.

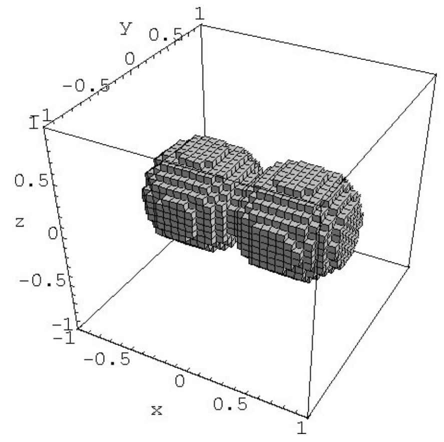

Fig. 23. Example 19: The pair of just touching surfaces plotted by 3D RT++ method (1572 voxels)

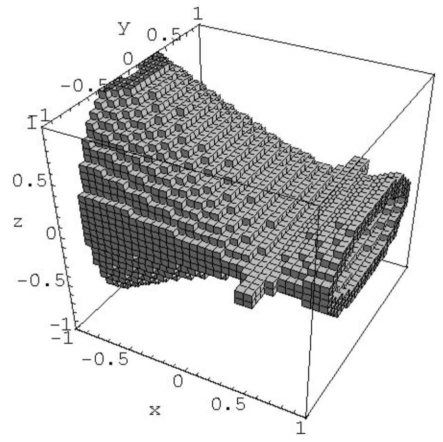

Fig. 24. Example 20: The cone like surface with a line singularity plotted by 3D RT++ method (3288 voxels)

– In 9 out of 10 cases (all but Example 11) the RT++ method needed fewer arithmetic operations than the MAA++ method.

Overall we may probably conclude that the 3D RT++ method is the best choice in terms of accuracy and efficiency.

6 Why use Order Two Taylor Expansion?

In sect. 4 we proposed an order 2 recursive Taylor method for finding the bound of a polynomial, and in sect. 5 we gave some examples to show that this method works well. Clearly, however, we could have chosen to use some other order for our Taylor expansion, so we will now justify why we use a second order expansion rather than some other order, particularly order 1, 3 or 4. To do so we give an experimental comparison between recursive Taylor methods of orders 1–4.

We first begin by explicitly stating order 1, 3 and 4 recursive Taylor algorithms for evaluating a bivariate polynomial $f(x, y)$. An order 1 recursive Taylor algorithm is given in fig. 25, while an order 3 recursive Taylor algorithm is given in fig. 26, and an order 4 recursive Taylor algorithm is given in fig. 27.

Using the same curves from Examples 1–10 as before, we compared the accuracy and efficiency of order 1, 2, 3 and 4 recursive Taylor methods, using the same criteria of assessment as before. The test results are shown in Table 2 (*left*).

From Table 2 (*left*) we can see that:

– The order 1 recursive Taylor method is less accurate than order 2, 3 and 4 recursive Taylor methods.
– Usually, but not always, the order 1 method needs more arithmetic operations than order 2, 3 and 4 methods (Example 2 is a counterexample).

```
Bound($f, \underline{x}, \overline{x}, \underline{y}, \overline{y}$):
IF $f \equiv c$ RETURN Interval$[c, c]$,
ELSE
    $x_0 = (\underline{x} + \overline{x})/2$;   $y_0 = (\underline{y} + \overline{y})/2$;   $x_1 = (\overline{x} - \underline{x})/2$,   $y_1 = (\overline{y} - \underline{y})/2$;
    $[\underline{f}, \overline{f}] = f(x_0, y_0) + x_1$Bound$(f_x, \underline{x}, \overline{x}, \underline{y}, \overline{y})[-1,1] + y_1$Bound$(f_y, \underline{x}, \overline{x}, \underline{y}, \overline{y})[-1,1]$;
    RETURN Interval$[\underline{f}, \overline{f}]$.
```

Fig. 25. Order 1 recursive Taylor algorithm

```
Bound($f, \underline{x}, \overline{x}, \underline{y}, \overline{y}$):
IF $f \equiv c$ RETURN Interval$[c, c]$,
ELSE
    $x_0 = (\underline{x} + \overline{x})/2$;   $y_0 = (\underline{y} + \overline{y})/2$;   $x_1 = (\overline{x} - \underline{x})/2$,   $y_1 = (\overline{y} - \underline{y})/2$;
    $[\underline{f}, \overline{f}] = f(x_0, y_0) + x_1 f_x(x_0, y_0)[-1,1] + y_1 f_y(x_0, y_0)[-1,1]$
            $+\frac{1}{2}x_1^2[0,1]f_{xx}(x_0, y_0) + \frac{1}{2}y_1^2[0,1]f_{yy}(x_0, y_0) + x_1 y_1[-1,1]f_{xy}(x_0, y_0)$
            $+\frac{1}{6}x_1^3[-1,1]$Bound$(f_{xxx}, \underline{x}, \overline{x}, \underline{y}, \overline{y} + \frac{1}{6}y_1^3[-1,1]$Bound$(f_{yyy}, \underline{x}, \overline{x}, \underline{y}, \overline{y})$
         $+\frac{1}{2}x_1^2 y_1[-1,1]$Bound$(f_{xxy}, \underline{x}, \overline{x}, \underline{y}, \overline{y}) + \frac{1}{2}x_1 y_1^2[-1,1]$Bound$(f_{xyy}, \underline{x}, \overline{x}, \underline{y}, \overline{y})$;
    RETURN Interval$[\underline{f}, \overline{f}]$.
```

Fig. 26. Order 3 recursive Taylor algorithm

```
Bound($f, \underline{x}, \overline{x}, \underline{y}, \overline{y}$):
IF $f \equiv c$ RETURN Interval$[c, c]$,
ELSE
    $x_0 = (\underline{x} + \overline{x})/2$;   $y_0 = (\underline{y} + \overline{y})/2$;   $x_1 = (\overline{x} - \underline{x})/2$,   $y_1 = (\overline{y} - \underline{y})/2$;
    $[\underline{f}, \overline{f}] = f(x_0, y_0) + x_1 f_x(x_0, y_0)[-1,1] + y_1 f_y(x_0, y_0)[-1,1]$
            $+\frac{1}{2}x_1^2[0,1]f_{xx}(x_0, y_0) + \frac{1}{2}y_1^2[0,1]f_{yy}(x_0, y_0) + x_1 y_1[-1,1]f_{xy}(x_0, y_0)$
            $+\frac{1}{6}x_1^3[-1,1]f_{xxx}(x_0, y_0) + \frac{1}{6}y_1^3[-1,1]f_{yyy}(x_0, y_0)$
            $+\frac{1}{2}x_1^2 y_1[-1,1]f_{xxy}(x_0, y_0) + \frac{1}{2}x_1 y_1^2[-1,1]f_{xyy}(x_0, y_0)$
            $+\frac{1}{24}x_1^4[0,1]$Bound$(f_{xxxx}, \underline{x}, \overline{x}, \underline{y}, \overline{y}) + \frac{1}{24}y_1^4[0,1]$Bound$(f_{yyyy}, \underline{x}, \overline{x}, \underline{y}, \overline{y})$
            $+\frac{1}{6}x_1^3 y_1[-1,1]$Bound$(f_{xxxy}, \underline{x}, \overline{x}, \underline{y}, \overline{y})$
            $+\frac{1}{6}x_1 y_1^3[-1,1]$Bound$(f_{xyyy}, \underline{x}, \overline{x}, \underline{y}, \overline{y})$
            $+\frac{1}{4}x_1^2 y_1^2[0,1]$Bound$(f_{xxyy}, \underline{x}, \overline{x}, \underline{y}, \overline{y})$;
    RETURN Interval$[\underline{f}, \overline{f}]$.
```

Fig. 27. Order 4 recursive Taylor algorithm

- In 9 out of 10 cases the order 2 recursive Taylor method has the same accuracy as order 3 and 4 methods. In the other case (Example 4) the order 2 method is more accurate than the order 3 and 4 methods.
- In 6 out of 10 cases, the order 2 recursive Taylor method needs fewer arithmetic operations than the order 3 method (Examples 1,2,6,7,9,10).
- In all cases the order 4 recursive Taylor methods has the same accuracy as the order 3 method.
- In 6 out of 10 cases order 4 recursive Taylor method needs fewer arithmetic operations than the order 3 method (Examples 1,4,6,7,9,10).

Table 2. Comparison of order 1, 2, 3 and 4 RT methods under resolution 256×256 (*left*) and resolution 16×16 (*right*)

Ex.	Method	Pixels plotted	Subdivisions	Additions	Multiplications
1	1	550	631	795049	536562
1	2	526	571	415688	343892
1	3	526	567	460441	429975
1	4	526	563	252186	287257
2	1	438	497	248387	191938
2	2	433	461	241581	205717
2	3	433	460	246584	240250
2	4	433	459	334228	357296
3	1	619	681	1771000	1265762
3	2	608	637	1116344	936757
3	3	608	636	793926	808037
3	4	608	634	887844	1000846
4	1	843	952	12534981	9330145
4	2	801	845	4662221	4461229
4	3	816	860	3767717	4179094
4	4	816	857	2149817	3043237
5	1	484	803	1171467	869116
5	2	464	627	664231	575815
5	3	464	615	518665	535267
5	4	464	611	691345	764062
6	1	492	710	1053137	762808
6	2	460	567	442025	414092
6	3	460	560	743610	707035
6	4	460	560	281964	357439
7	1	562	755	990114	684256
7	2	512	629	445039	386359
7	3	512	627	644351	600905
7	4	512	627	273019	327424
8	1	846	895	612153	402862
8	2	818	829	563844	422917
8	3	818	827	258064	246520
8	4	818	827	337480	352408
9	1	1336	1625	2410713	1745518
9	2	1144	1281	998825	935312
9	3	1144	1269	1685062	1601793
9	4	1144	1269	639200	809781
10	1	844	997	1479305	1059079
10	2	784	849	662153	609761
10	3	784	845	1122246	1056562
10	4	784	845	425760	529126

Ex.	Method	Voxels plotted	Subdivisions	Additions	Multiplications
1	1	58	57	72137	48662
1	2	48	49	35848	29648
1	3	44	49	39969	37331
1	4	44	45	20266	23077
2	1	36	52	26059	20168
2	2	32	36	18977	16167
2	3	32	34	18348	17878
2	4	32	33	24200	25868
3	1	55	57	148840	106370
3	2	43	48	84512	70927
3	3	43	47	58950	60007
3	4	43	45	63340	71404
4	1	76	68	898473	668713
4	2	63	53	293765	281053
4	3	63	52	228897	253830
4	4	62	50	126073	178387
5	1	156	85	124747	92240
5	2	88	77	81927	70915
5	3	84	77	65225	67207
5	4	82	77	87465	96562
6	1	100	84	125089	90484
6	2	57	74	57845	54202
6	3	55	72	95878	91179
6	4	55	72	36344	46095
7	1	80	67	88282	60928
7	2	58	51	36311	31467
7	3	58	49	50663	47181
7	4	58	49	21507	25708
8	1	64	59	40545	26662
8	2	58	51	34876	26137
8	3	58	49	15400	14676
8	4	58	49	20128	20980
9	1	108	85	126601	91558
9	2	88	73	57193	53472
9	3	88	69	92038	87393
9	4	88	69	34976	44181
10	1	92	85	126537	90535
10	2	68	65	50905	46849
10	3	64	65	86646	81562
10	4	64	65	32880	40846

Obviously the order 1 recursive Taylor method is not as good as the order 2, 3 or 4 methods in accuracy or speed. On the other hand, we note that the order 3 and 4 recursive Taylor methods are not always at least as accurate as the order 2 method (see Example 4), or as efficient (see Example 2). While it is clear that the order 1 method can be rejected on grounds of poor performance, choice amongst the higher order methods is less clear-cut. Unsurprisingly, in most cases, using higher-order recursive Taylor methods leads to fewer recursive operations, but the decrease between using orders 1 and 2 is much greater than between using orders 2 and 3, and between higher orders. Overall the above results suggest using an order 2 recursive Taylor method as the best compromise between accuracy and efficiency, and ease of implementation.

However, a word of warning is necessary. This judgement strictly applies only to 256×256 resolution. If we reduce the resolution to 16×16 we get the results shown in Table 2 (*right*). This Table shows that under these conditions, a second order expansion need neither be most accurate (see Examples 1,4,5,6,10), nor most efficient. (The accuracy of the second order method is still quite close to that of the third and fourth order methods in all cases, however). Clearly, these results show that a theoretical proof that *any* particular order expansion is the best choice is not possible.

7 Theoretical Connection Between Taylor Method and MAA

In this section we briefly consider a theoretical relation between the Taylor method and the modified affine arithmetic method. It only concerns the intervals output by a direct (i.e. non-recursive) Taylor method and the MAA method; furthermore, it does not say how many operations are needed by each method.

Theorem 1 *Given a degree n polynomial, suppose $m > n$, and we perform an order m Taylor method. The output interval is equivalent to that produced by the modified affine arithmetic method.*

Proof We only prove the theorem here in the univariate case. The proofs for multivariate cases are similar.

Let $f(x) = \sum_{i=0}^{n} a_i x^i$ be the degree n polynomial in one variable whose range we wish to estimate over $[\underline{x}, \overline{x}]$. Let $x_0 = (\underline{x} + \overline{x})/2$, and $x_1 = (\overline{x} - \underline{x})/2 > 0$. Then the centered form of $f(x)$ on $[\underline{x}, \overline{x}]$ is

$$f(x) = f(x_0) + \sum_{i=1}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i. \tag{1}$$

It is known that the modified affine arithmetic method produces the same results as carrying out interval arithmetic on the centred form method with proper consideration of even and odd properties of polynomial terms [9]. If we evaluate $f(x)$ on $[\underline{x}, \overline{x}]$ using the modified affine arithmetic method we get

$$f_{MAA}[\underline{x}, \overline{x}] = f(x_0) + \sum_{i=1}^{n} \frac{f^{(i)}(x_0)}{i!} x_1^i \times \left\{ \begin{array}{l} \text{[0,1], if } i \text{ is even} \\ \text{[-1,1], if } i \text{ is odd} \end{array} \right\} \tag{2}$$

On the other hand, when $m > n$, the degree m Taylor form of $f(x)$ on $[\underline{x}, \overline{x}]$ is the same as eq. 1, because for any integer $i > n$, $f^{(i)}(x) = 0$ when $f(x)$ is a degree n polynomial. Therefore if we evaluate $f(x)$ on $[\underline{x}, \overline{x}]$ using a degree m Taylor method, we get the same interval as in eq. 2.

More work is needed to compare theoretically the intervals produced by the recursive Taylor method with those from MAA, and also to compare the numbers of operations. We intend to study these issues in the near future.

8 Conclusions

From the above experiments we can see that recursive Taylor methods can produce at least as good graphical results as the modified affine arithmetic method, and often need fewer arithmetic operations. Furthermore, the recursive Taylor method is simple and very easy to implement. One minor disadvantage of the recursive Taylor methods are that they often need a few more recursive operations than MAA. Repeating our earlier conclusions, overall we suggest using the *second order* recursive Taylor method as the best compromise (in terms of order) between accuracy and efficiency, and ease of implementation.

Acknowledgements

We wish to thank Dr. Stephen Cameron of Oxford University for insightful comments. This work was supported jointly by the National Natural Science Foundation of China (Grant No.60173034), the National Natural Science Foundation for Innovative Research Groups (No.60021201) and the Foundation of State Key Basic Research 973 Item (Grant No.2002CB312101).

References

1. Berz, M., Hoffstatter, G., Computation and application of Taylor polynomial with interval remainder bounds, *Reliable Computing*, 1998, 4: 83–97.
2. Cai, Y. Z., *Positive-Negative Method for Numerical Control Plot*, Zhejiang University Publishing House, Hangzhou, 1990 (in Chinese).
3. Chandler, R. E., A Tracking Algorithm for Implicitly Defined Curves, *IEEE Computer Graphics & Applications*, 1988, 8(2): 83–89.
4. Jin, T. G., T-N method for curve approximation, *Journal of Zhejiang University, Proceedings of Computational Geometry Conference*, 1982, 150–176 (in Chinese).
5. Martin R., Shou H., Voiculescu I., Bowyer A., Wang G., Comparison of Interval Methods for Plotting Algebraic Curves, *Computer Aided Geometric Design*, 2002, 19(7): 553-587.
6. Patrikalakis, N. M., Maekawa T., *Shape Interrogation for Computer Aided Design and Manufacturing*, Springer Verlag, 2002.
7. Ratschek, H., Rokne, J., *Computer Methods for the Range of Functions*, Ellis Horwood Ltd., 1984.
8. 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*, 2002, 12(1): 77–80.
9. Shou H., Lin H., Martin R., Wang G., Modified affine arithmetic is more accurate than centered interval arithmetic or affine arithmetic, In: Michael J. Wilson, Ralph R. Martin (Eds.), Lecture Notes in Computer Science 2768, *Mathematics of Surfaces*, Springer-Verlag Berlin Heidelberg New York 2003, 355-365.
10. Snyder, J. M., Interval Analysis for Computer Graphics, *Computer Graphics (SIGGRAPH'92 Proceedings)*, 1992, 26(2): 121–130.
11. Suffern, K. G., Quadtree Algorithms for Contouring Functions of Two Variables, *The Computer Journal*, 1990, 33: 402–407.

12. Taubin, G., Distance Approximations for Rasterizing Implicit Curves, *ACM Transactions on Graphics*, 1994, 13(1): 3–42.
13. Taubin, G., Rasterizing Algebraic Curves and Surfaces, *IEEE Computer Graphics and Applications*, 1994, 14: 14–23.
14. Tupper, J., Realiable two-dimensional graphing methods for mathematical formulae with two free variables, *Proceedings of SIGGRAPH'2001*.