ACM Symposium on Solid Modeling and Applications (2004)
G. Elber, N. Patrikalakis, P. Brunet (Editors)

Residual iteration and accurate polynomial evaluation for shape-interrogation applications

C. M. Hoffmann[†], G. Park[‡], J.-R. Simard[§] and N. F. Stewart[¶]

Abstract

Surface interrogation and intersection depend crucially on good root-finding algorithms, which in turn depend on accurate polynomial evaluation. Conventional algorithms for evaluation typically encounter difficulties near multiple roots, or roots that are very close, and this may lead to gross errors in the geometric computation, or even catastrophic failure. In this paper we study the cost and accuracy of several approaches to polynomial evaluation, explaining the reasons for non-convergence of certain methods, and supporting our subsequent conclusions with the results of benchmarking experiments.

Categories and Subject Descriptors (according to ACM CCS): G.1 [Numerical Analysis]: I.3.5 [Computational Geometry and Object Modeling]: J.6 [Computer-Aided Engineering]:

1. Introduction

Many important problems in CAD and CAGD reduce directly to solving systems of nonlinear polynomial equations of the form

$$p(x) = 0,$$

where p and x are in Euclidean spaces of possibly different dimension. Such problems include the calculation of intersections, distance functions, and curvature extrema [PMS 02].

The usual difficulties solving nonlinear polynomial equations are exacerbated when dealing with multiple roots. Such cases arise at surface intersections that are tangential or singular. Even the evaluation of polynomials near multiple roots turns out to be of reduced accuracy. Inaccuracies in these fundamental tasks can lead to gross errors in the geometric computation, or even catastrophic failure [Hof 01].

As a part of a project which includes finding more robust surface intersection evaluators [CARGO] we are re-examining these issues from the ground up. Arguing from first principles, it is clear that accurate root finding cannot succeed without accurate polynomial evaluation. Focusing specifically on this subproblem, we study in this paper the cost and feasibility of accurate floating-point evaluation using various approaches that have been proposed before.

Tangential or singular surface intersections arise often by design in fairing operations and in blending operations. Curvature-continuous surface intersections are not uncommon in ship-hull design. In such demanding situations, some authors attempt to increase the certainty of the numerical evaluation of the geometry by employing interval-arithmetic enclosure methods, *e.g.* [Pat 03]. Here, a common problem near multiple roots is that the enclosure algorithm cannot exclude the presence of a root in many intervals near the true root because the polynomial evaluation is not accurate enough. Thus, accurate polynomial evaluation has significant impact on CAD in a number of critical situations.

CAD systems create complex geometry with tangential and curvature-continuous contacts based on user specifications and proprietary algorithms. As long as the created geometry remains under the control of the CAD system, the difficulties of tangential and near-tangential intersection computations can be controlled by retaining additional information such as the rail curves of blending surfaces. This may motivate a perception that increasing the ac-

[†] Department of Computer Science, Purdue University; partially supported by NSF grants DMS-0138098 and CCR-9902025.
[‡] Department of Computer Science, Purdue University, supported by NSF grants DMS-0138098 and CCR-9902025.
[§] Département IRO, Université de Montréal, CP6128, Succ. CentreVille, H3C 3J7, Canada.
[¶] Département IRO, Université de Montréal, CP6128, Succ. CentreVille, H3C 3J7, Canada. The research of the fourth author was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada and by NSF grant DMS-0138098.

© The Eurographics Association 2004.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

curacy of intersection computations is a minor matter in practice. However, when the geometry is exported to other CAD systems or analysis programs for additional manipulation, then the additional information that simplified difficult intersections for the original CAD system is removed. Thus the problem we sketched has a practical dimension.

Böhm [Boe 83] and others [Ham+95] have described an algorithm for the evaluation of univariate polynomials

$$p(t) = \sum_{i=0}^{n} p_i t^i, \tag{1}$$

as well as [Kra 91] bivariate polynomials

$$p(x,y) = \sum_{j=0}^{m} \sum_{i=0}^{n} p_{ij} x^i y^j$$

where the coefficients p_i or p_{ij} are assumed to be exactly representable in the computer. Kramer's extension can be used for polynomials in more than two variables as well. Although the matrices generated can be large, they are highly structured and do not have to be represented explicitly.

Böhm's algorithm, which will be described in the next section, is based on Horner's method. Part of the algorithm is closely related to the classical method of *iterative improvement*, which is used to improve computed solutions to systems of linear equations [Wil 65]. The method of iterative improvement generally performs very well: indeed, Wilkinson states [Wil 65]p. 256 that "... the performance of the ... process does not fall far short of that corresponding to exact iteration provided only that inner-products are accumulated [in double precision] in the computation of the residuals." Nevertheless, there are difficulties with the method in the context of certain of the problems considered here. These difficulties can be explained by careful examination of the analysis of iterative improvement given in [Wil 63]. Our purpose is to provide this explanation, and to describe experimental results obtained with the iterative-improvement method and, more generally, with the method of Böhm. Moreover, we contrast the residual iteration approach with Priest's multi-precision arithmetic approach [Priest92].

2. Residual iteration in polynomial evaluation

2.1. The method of Böhm for evaluating polynomials

We restrict our attention to the case of the single-variable polynomial, as in equation (1). Rewriting $p(t)$ as in reference [Ham+95], in the Horner nested-multiplication form

$$p(t) = (\ldots(p_n t + p_{n-1})t + \ldots + p_1)t + p_0,$$

and defining

$$\begin{aligned} x_n &= p_n \\ x_i &= x_{i+1} t + p_i, \quad i = n-1, \ldots, 0, \end{aligned}$$

we obtain the linear system

$$Ax = p \tag{2}$$

where

$$A_{(n+1)\times(n+1)} = \begin{bmatrix} 1 & & & & & \\ -t & 1 & & & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & -t & 1 \end{bmatrix},$$

$$x = \begin{bmatrix} x_n \\ x_{n-1} \\ \cdot \\ \cdot \\ \cdot \\ x_0 \end{bmatrix}, \text{ and } p = \begin{bmatrix} p_n \\ p_{n-1} \\ \cdot \\ \cdot \\ \cdot \\ p_0 \end{bmatrix}$$

The desired solution to the problem of evaluating $p(t)$ can be obtained by taking the last component of x: $x_0 = p(t)$. Indeed, the process of computing an initial approximation $x^{(1)}$ to this linear system by direct forward substitution gives

$$\begin{aligned} x_n^{(1)} &= p_n \\ x_i^{(1)} &= x_{i+1}^{(1)} t + p_i, \quad i = n-1, \ldots, 0, \end{aligned}$$

which is exactly Horner's method.

2.2. Iterative improvement

Given the initial approximation $x^{(1)}$ to the solution of (2), the classical method of iterative improvement [Wil 65, Wil 63] can be applied to obtain a sequence of vectors $x^{(s)}$ which converge, under certain conditions, to the exact solution x of (2). Since in our special case the matrix A is lower triangular, the process [Wil 63]p. 121 simplifies: for $s \geq 1$, we first compute the residual $r^{(s)}$ from

$$r^{(s)} = p - Ax^{(s)}, \tag{3}$$

and then the correction $x^{(s+1)} - x^{(s)}$ from

$$x^{(s+1)} - x^{(s)} = A^{-1} r^{(s)}; \tag{4}$$

this last is computed by means of the same forward-substitution process that was used above to compute $A^{-1}p$, with p replaced by $r^{(s)}$. If guaranteed solutions are required, then the calculation of

the residual in (3), and the correction vector in (4), can be computed in interval arithmetic [Ham+95]. Alternatively, initial iterations $x^{(1)}, x^{(2)}, \ldots, x^{(s_0)}$ can be computed using ordinary floating-point arithmetic, with double-precision calculation of residuals, until convergence, followed by a final iteration $x^{(s_0+1)}$, using interval arithmetic, to obtain an interval bound.

It is essential, however, that the residual in (3) be computed to high accuracy [Wil 63]p. 126. If this is done, then under normal conditions the iterative-improvement procedure provides excellent results [Wil 65]p. 256. On the other hand, there are exceptional cases, one of which arises in the use of this procedure for the evaluation of polynomials with multiple roots.

Consider for example the polynomial

$$p(t) = \sum_{i=0}^{n} p_i t^i$$

where

$$p_i = \binom{n}{i}(-1)^{n-i}, i = 0, \ldots, n,$$

so that

$$p(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (-1)^{n-i} = (t-1)^n.$$

We might take $n = 12$ and consider evaluating the polynomial at $t = 1 + 10^{-6}$, so that $p(t) = 10^{-72}$. Since

$$A^{-1} = \begin{bmatrix} 1 & & & & & \\ t & 1 & & & & \\ t^2 & t & 1 & & & \\ . & & & . & & \\ . & & & & . & \\ t^{12} & t^{11} & . & . & t & 1 \end{bmatrix}$$

we see that x_0, which is equal to the last component of $A^{-1}p$, is equal to

$$\sum_{i=0}^{12} t^i p_i = \sum_{i=0}^{12} t^i \binom{n}{i}(-1)^{12-i} = (t-1)^{12},$$

which is indeed $p(t)$.

The matrix A has $\|A^{-1}\|_\infty = \sum_{i=0}^{12} t^i = \frac{t^{13}-1}{t-1} \cong 13$, and $\|A\|_\infty = 2 + 10^{-6}$, so that its condition number $\|A^{-1}\|_\infty \cdot \|A\|_\infty$ is approximately 26. This is well within the bound [Wil 63](38.1) required for convergence using "block floating" arithmetic. The iterative-improvement procedure fails to converge using floating-point arithmetic, however, even when the residual is computed in double precision [Wil 63]p. 126. (This fact is easily verified empirically by writing a short computer program.)

The reason for this failure is as follows. Suppose that $|x_0^{(s)}|$ has been reduced to a value below $2^{-53} \cong 10^{-17}$; then, using IEEE double-precision floating-point arithmetic [IEEE] the result of calculating

$$r_0^{(s)} = ((p_0 - x_0^{(s)}) + x_1^{(s)} t)$$

in double precision gives a result that does not depend on $x_0^{(s)}$, since $p_0 = 1$, and subtracting $x_0^{(s)}$ from 1 in double precision produces exactly 1. Since this is the only place at which $x_0^{(s)}$ enters the calculation, it can be seen that once $|x_0^{(s)}|$ is below the threshold mentioned, subsequent iterations do not even depend on $x_0^{(s)}$.

The convergence analysis given in [Wil 63] is for block-floating arithmetic—floating-point arithmetic is discussed only informally [Wil 63]p. 126. If we try to imitate the analysis, to obtain an analysis for floating-point arithmetic, we encounter a problem at a fairly early stage. Block-floating arithmetic produces an exact representation of the residual $r^{(s)}$ which can be converted to a standardized block-floating vector with small relative error [Wil 63](38.11), and this is a crucial step in the convergence proof. Although double-precision calculation of the residual will *usually* produce an estimate with small relative error, this cannot be guaranteed, as the example described above shows. In such cases we are therefore forced to compute the residual using extended precision, as is done for the inner-product operator $\diamond$ in [Ham+95], and to use this throughout the process of iterative improvement.

2.3. The Inner-Product Operator $\diamond$

Given floating-point numbers a_k and b_k, $k = 1, \ldots, n$, the operator $\diamond$ delivers as result a floating point number S' such that there is no representable floating point number S'' properly between $S = \sum_{k=1}^{n} a_k b_k$ and S'. The operation can be implemented in several ways.

Hammer *et al.* [Ham+95] implement an accumulator that holds, as a fixed-point number, the mantissa of products $a_k b_k$ and their summations. At a size of 544 bytes, the accumulator is large enough to represent all partial sums that do not overflow or underflow with the a_k and b_k in double precision. Thus, an accumulator can be used to compute accurate summations S and extract the leading bits as S'. Hence, S' is accurate to within less than one unit of least precision (ulp).

Kobbelt [Kob 97] implements an accumulator as an array of floating-point numbers indexed by their exponents. For each ex-

ponent value, there is an array element. Kobbelt gives an algorithm for summations using the array with appropriate carry. The final result can be extracted by combining partial sums recursively.

Because Kobbelt's summation delays the construction of the leading bits of S, his implementation does better when large summations are needed with few result extractions. Hammer's implementation is better suited for general-purpose situations.

2.4. Distillation

Studied by Priest [Pri 92], several algorithms have been developed that effectively carry out multiple-precision floating-point computations without explicitly manipulating extended mantissae. These techniques rest on the concept of compensating summation and distillation. Intuitively, a distillation of a sum

$$S = \sum_{k=1}^{n} a_k$$

of n floating-point numbers calculates p floating point numbers b_k such that

$$\sum_{k=1}^{n} a_k = \sum_{k=1}^{p} b_k$$

exactly and the mantissae of the b_k are disjoint, i.e.,

$$|b_{k+1}| < \mathrm{ulp}(|b_k|).$$

It is then clear that b_1 is an approximation to S that is accurate to within 1 ulp. By rounding according to the leading bit of b_2, S' can be accurate to within 1/2 ulp.

Distillation can be used to implement floating-point summation exactly. Additionally, products of distillations again can be represented exactly as sums of partial products, so implementing floating-point products exactly. This provides a third way for implementing the inner-product operator $\diamond$.

Since Priest's algorithms can compute multiplication and addition directly, it is also possible to use them to evaluate polynomials directly, without iteration. We include below data to show how efficient the direct evaluation is.

3. Experimental Assessment of Evaluation Methods

We have implemented polynomial evaluation in the following ways:

1. Simple Horner: The polynomial is evaluated using the usual Horner method. This provides us a baseline datum for assessing accuracy and speed.
2. Residual iteration using the Hammer *et al.* implementation of $\diamond$.
3. Direct evaluation of the polynomial using distillation.
4. Residual iteration using distillation of sums and products.

All evaluations were done in the fully expanded power-base form; no factorization is considered. The polynomials used to benchmark polynomial evaluation are the following:

- $p_1(x) = (x-2)^4$, $x = 2 + 10^{-8}$
- $p_2(x) = (x-2)^6$, $x = 2 + 10^{-8}$
- $p_3(x) = (x-2)^8$, $x = 2 + 10^{-8}$
- $p_4(x) = 0.886339x^6 - 0.163394x^5 + 1.38273x^4 + 4.28066x^3 - 0.390834x^2 - 1.38172x - 0.00221897$, $x = 0.00221897$
- $p_5(x) = -6.18121x^7 - 4.04466x^6 - 0.353717x^5 + 1.557x^4 + 0.846107x^3 + 4.10773x^2 + 0.307834x + 1.03226$, $x = -1.03226$
- $p_6(x) = 1.41423x^8 - 0.70459x^7 - 0.156364x^6 + 3.41544x^5 + 7.70459x^4 - 2.04128x^3 - 0.522173x^2 - 0.529087x - 1.5827$, $x = -1.06724$
- $p_7(x) = -0.7934x^9 - 0.257333x^8 - 0.402707x^7 + 1.67564x^6 + 0.442567x^5 + 1.58694x^4 + 0.110229x^3 - 0.777947x^2 - 0.440497x + 1.5179$, $x = -0.257333$

Polynomials p_1 through p_3 have been chosen to test the evaluation near a multiple root. Despite the high multiplicity, the coefficients of these polynomials have low bit complexity. The remaining four polynomials have simple roots but have coefficients of high bit complexity. They were generated randomly.

The table shows observed evaluation times for the four methods. All computations were done using 64-bit double-precision arithmetic on a Pentium 4 PC. In each case, the CPU time of a total of 3000 (repeated) evaluations was measured. All times are in milliseconds.

Polyn.	p_1	p_2	p_3	p_4	p_5	p_6	p_7
Degree	4	6	8	6	7	8	9
Iterat'ns	3	4	5	1	1	1	1
Horner	63	78	94	78	79	94	94
Hammer	390	1125	1938	328	375	406	453
Direct	1906	5563	10720	7137	10360	13938	17517
Distill	1531	11922	32062	609	672	766	859

Depending on the number of iterations, Hammer's method costs a factor of between 1:4 and 1:20 compared to Horner, with the number of iterations the main cost parameter. For multiple roots, more

than one iteration is needed when evaluating in its vicinity. Direct evaluation, using Priest's method, is unattractive, with cost factors as high as 1:180. Here, the bit complexity of the coefficients is the main factor leading to large vectors in the distillations. The cost can be reduced by using residual iteration with sum distillation, yet the computing times observed are always higher than using Hammer's inner-product implementation. Here, the number of iterations has a big impact.

The accuracies that are achieved over Horner's method are summarized as follows:

- p_1 : Horner evaluates to exactly zero. The other evaluation methods obtain the value 10^{-32} (hex 3949 f623 cb12 02b2), with an interval width of 3 ulp(10^{-32}) after 3 iterations.
- p_2 : Horner evaluates $8.53 \cdot 10^{-14}$, whereas the accurate evaluation is 10^{-48} with an enclosure width of 1 ulp after 4 iterations.
- p_3 : Horner evaluation yields $1.02 \cdot 10^{-12}$, accurate evaluation 10^{-64}, with an enclosure width of 1 ulp.
- p_4 : The Horner value is 2.64044, the accurate value is also. The distance between the two, observing the binary mantissae, is 1 ulp (4005 1f9d 29c7 a5da vs 4005 1f9d 29c7 a5db).
- p_5–p_7: The Horner value differs from the exact value by no more than 2 ulp.

The accuracy of evaluating p_4 through p_7 reflects the fact that the argument and the coefficients do not generate sums with terms of vastly different magnitude. Thus, digit cancellation does not create large errors.

4. Conclusions

Accurate polynomial evaluation is a key prerequisite for good root-finding algorithms which, in turn, are fundamental to surface interrogation and intersection. These operations typically encounter difficulties near multiple roots or near roots that are very close. Conventional evaluation algorithms are then a fundamental obstacle since they are inherently imprecise in such cases. Accurate evaluation is possible, but the experiments show that the algorithms available to us today are expensive. Residual iteration in particular is an algorithm capable of delivering accurate values, but the implementation of the $\diamond$ operator is expensive.

As we have seen, the difference between the accurately rounded value and the value delivered by Horner can be more than 30 orders of magnitude. In such situations, conventional evaluation delivers no information near the root. This has led to inefficiencies in shape-interrogation algorithms [Pat 03].

Against this backdrop, we have demonstrated that several techniques for accurate evaluation are rather costly and should not be used to simply displace conventional evaluation. Conventional evaluation using classical iterative improvement, conventionally implemented, may lead to non-convergence in cases of importance. These insights provide strong justification for the certificate approach of computational geometry pioneered by Fortune [FVW 93]. Indeed, as shown by evaluations of p_4 through p_7, simple evaluation can be efficient *and* accurate.

It remains to be seen whether novel hardware algorithms such as the *fused floating-point multiply and add* (fma) instruction on the Intel Itanium can make significantly lower the cost of accurate polynomial evaluation; see also [Nie 03]. The fma instruction computes the expression $d = a * b + c$ with all intermediate bits, and thus can deliver the floating-point value for d to the result precision rather than to the precision of $max(|a * b|, |c|)$. It is conceivable that it allows us to push the boundary of meaningful routine calculation closer to the vicinity of multiple roots.

Brep solid representations of curved solids contain geometric data that is intrinsically inexact. This raises the question how useful it is to improve the accuracy of evaluation and intersection computations. It seems to us, in this context, that precise evaluation must be combined with a semantic conceptualization of what the imprecise input data means in a mathematical sense. Clearly, without such a semantics highly precise evaluation is a luxury with unclear purpose. But conversely, given such a semantics, the inability to evaluate geometry with high precision at reasonable cost may be equally crippling.

The algorithms we have examined evaluate polynomials given in power-base representation. Corresponding techniques that work with the Bernstein-Bezier basis or with a B-spline representation would be desirable.

References

[PMS 02] Patrikalakis, N. M. and Maekawa, T. Shape Interrogation for Computer Aided Design and Manufacturing. Springer, 2002.

[Hof 01] Hoffmann, C. M. Robustness in geometric computa-

tions. *J. Computing and Information Science in Engineering* (1), 143-156, 2001.

[CARGO] CARGO Project "ITANGO", www.cse.uconn.edu/~biscegli/itango/, 2001.

[Pat 03] Patrikalakis, N. M. Shape Interrogation. Keynote Address, Proc. 8th ACM Symposium on Solid Modeling and Applications, 2003.

[Boe 83] Böhm, W. Berechnung von Polynomnullstellen und Auswertung arithmetischer Ausdrücke. PhD Thesis, Universität Karlsruhe, 1983.

[Ham+95] Hammer, R., *et al*, Chapter 4 "Evaluation of polynomials". In C++ Toolbox for Verified Computing, Springer, 1995.

[Kra 91] Krämer, W. Evaluation of polynomials in several variables with high accuracy. In COMPUTER ARITHMETIC, Scientific Computation and Mathematical Modelling, E. Kaucher, S. M. Markov and G. Mayer (editors), J. C. Baltzer AG, Scientific Publishing Co., IMACS, 239-249, 1991.

[Wil 65] Wilkinson, J. H. The Algebraic Eigenvalue Problem. Oxford, 1965.

[Wil 63] Wilkinson, J. H. Rounding Errors in Algebraic Processes. Prentice-Hall, 1963.

[IEEE] IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std 754-1985, IEEE, New York, 1985.

[Kob 97] Kobbelt, L. Robust and efficient evaluation of functionals on parametric surfaces. 13th ACM Symposium on Computational Geometry, ACM Press, 376-378, 1997.

[Pri 92] Priest, D. On properties of floating point arithmetics: numerical stability and the cost of accurate comutations. PhD Thesis, UC Berkeley, 1992.

[FVW 93] Fortune, S. J. and Van Wyk, C. Efficient Exact Arithmetic for Computational Geometry. Proc. Ninth Annual Symposium on Computational Geometry, 163-172, 1993.

[Nie 03] Nievergelt, Y. Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit. *ACM Trans. on Math. Software* (29), No. 1, 27-48, 2003.