Robust and Efficient Evaluation of Functionals on Parametric Surfaces

Leif Kobbelt

Computer Sciences Department (IMMD IX), University of Erlangen - Nürnberg
Am Weichselgarten 9, 91058 Erlangen, Germany
kobbelt@informatik.uni-erlangen.de

Quality control is an important issue in computational geometry. Since analytical properties like the differentiability of a boundary representation are often not sufficient to fully judge the quality of an object, additional criteria measured in terms of scalar valued functionals have to be verified. Energy functionals in the context of *fairing* are just one example where the global smoothness of a surface is rated by one single value, e.g., the total amount of (squared) curvature. More "industrial" examples are the computation of *physical properties* like the surface area or the volume of an object, its center of gravity, or the moments of inertia.

Let an object X of arbitrary topology be given by a closed boundary representation $\Phi : \{\Omega_i\} \to \mathbb{R}^3$. The functionals we consider, have the form

$$P(X) \;:=\; \sum_i \int_{\Omega_i} F(\Phi, \Phi_u, \Phi_v)\;du\,dv \qquad (1)$$

where $F : \mathbb{R}^3 \to \mathbb{R}$ does not depend on the geometry of X. Hence, once the functional P characterizing the property under investigation is found, the problem reduces to the numerical approximation of a bivariate integral. A closed form solution for P is usually not known – even if X is a polyhedron with Φ being piecewise linear.

In this paper we suggest solutions for the key steps of an algorithm which performs the task of approximately evaluating (1) on a parametric surface. The efficiency of the algorithm results from the adaption of powerful quadrature techniques to the particular geometric setting. The robustness of the algorithm is due to exact arithmetic operations.

Non-uniform Romberg-quadrature

Iterative algorithms to approximately evaluate (multivariate) integrals use an *adaptive* refinement approach. The integrand function is sampled on an initial uniform grid and a simple cubature formula is applied. A posteriori error estimates decide whether the grid has to be refined in order to meet prescribed error tolerances[Str71]. The different approximations to the exact solution which one obtains by this iterative process can be subject to an extrapolation scheme that allows to further reduce the approximation error [Sto83].

Romberg-quadrature [Bul67] efficiently computes the Richardson-extrapolation by a Neville-type scheme. If a polynomial expansion of the approximation error as a function of the grid's step width h exists, the value of this polynomial at $h = 0$ can be computed by extrapolating approximate results T_i with step widths h_i. Romberg-quadrature specifically uses the trapezoid quadrature rule and the step widths $h_i = 2^{-i}$. From $T_{i,i} := T_i$ we recursively compute a tableau of values $T_{i,j}$ by

$$T_{i,j} \;:=\; \frac{4^{j-i}\; T_{i+1,j} - T_{i,j-1}}{4^{j-i} - 1}. \qquad (2)$$

[Bul66] shows that the error of the extrapolated value $T_{0,k}$ with respect to the exact solution T_∞ can be bounded by

$$|T_{0,k} - T_\infty| \;\le\; \sigma\,|T_{1,k} - T_{0,k-1}| \qquad (3)$$

for an appropriate $0 < \sigma \le 1$.

The major draw-back of this approach is the exponentially increasing number of function evaluations that are necessary to obtain an approximation on the next refinement level. Slightly reducing the prescribed error tolerance may increase the running time by the factor 4. Hence, one introduces *selective* refinement, i.e., only those parameter regions are refined where large local errors occur. Although this contradicts the assumption of nested uniform grids which is crucial for extrapolation schemes [Slu82], we combine the two opposite concepts in a scheme that allows to use selectively refined grids for a modified Romberg-quadrature.

Consider a triangular parameter domain Ω. The exact trapezoidal sums $T_{i,i}$ stem from a uniform subdivision of Ω into 4^i sub-triangles. Selectively refining only *some* sub-triangles yields an approximate value $\widetilde{T}_{i+1,i+1}$ which is the sum of local results from different grid levels. Our goal is to bound the error of the extrapolated values $\widetilde{T}_{0,k} \approx T_{0,k}$. This allows the application of (2) with (3) replaced by a modified error estimate.

Due to the exponential increase of parameter triangles, most of the effort is spent on the finest level. Reducing the complexity on this level has the heaviest impact on the total running time of the algorithm. The geometric background of the functionals ensures a regular behavior of the integrand function. Since the local convergence of the trapezoid rule is $O(h^4)$, the probability that a selective refinement algorithm picks a triangle from level $i+1$ before all level i triangles are subdivided, is low (the local error would have to differ by more than the factor 16).

Assume that the values $T_{i,i}$ with $i = 0, \ldots, k-1$ are exact and $\widetilde{T}_{k,k} = T_{k,k} + \varepsilon$ since not all triangles from generation $k-1$ have been subdivided yet. Applying (2) yields

$$\widetilde{T}_{0,k} \;=\; T_{0,k} + \left(\prod_{i=1}^{k} \frac{4^i}{4^i - 1} \right) \varepsilon.$$

Hence, no matter what k, $\widetilde{T}_{0,k}$ deviates from $T_{0,k}$ by less than $\frac{3}{2}\,\varepsilon$ and the analogue holds for $\widetilde{T}_{1,k}$. Taking into account that $\widetilde{T}_{0,k-1} = T_{0,k-1}$, we have

$$|\widetilde{T}_{0,k} - T_\infty| \;\leq\; \sigma\,|\widetilde{T}_{1,k} - \widetilde{T}_{0,k-1}| + 3\,\varepsilon \tag{4}$$

This estimate allows to use the result $\widetilde{T}_{0,k}$ extrapolated from a selectively refined grid to approximate the exact solution T_∞. The bound (4) gives an estimate of the current approximation error while ε is bounded by summing the local approximation errors over those parameter triangles which are still from generation $k-1$. Inverse Richardson-extrapolation allows to compute the difference $|\widetilde{T}_{1,k} - \widetilde{T}_{0,k-1}|$ without actually generating the table entries $\widetilde{T}_{j,k}$ [Kob92].

Exact Arithmetics

The reliable evaluation of algebraic expressions is an important issue in current research on computational geometry [For96], [Gui96]. Many computational geometers have proposed algorithms to perform exact arithmetic operations. [For93] and [She96] observe that most geometric *predicates* can be formulated as determinants. [Dek71] and [FW93] consider more elementary operations which allow to build code for the exact evaluation of fairly general formulae (assuming exact floating-point input data). However, the complexity of these solutions increases faster than the complexity of the formulae themselves.

When evaluating the functionals (1) either by a uniform or a selective refinement approach, the most unstable operation is the summation (*destillation*) of the local results. Both approaches, yield an increasing number of *local* estimates which have to be added up in order to obtain the *global* result. Eventually, one has to accumulate a huge number of local approximations, each within some prescribed tolerance to the exact value. Obviously, the rounding errors during this step can lead to invalid answers because they are usually not covered by error bounds emerging from numerical integration theory.

In practice one encounters situations where further refinement of the grid makes the approximation of the integral worse because of this effect. Consider a selective refinement cubature algorithm. The current approximation T is stored as the accumulated sum of the local values T_i. Local refinement means replacing one local value T_i by $T'_{i,1}, \ldots T'_{i,r}$ and updating

$$T \;\leftarrow\; T - T_j + T'_{j,1} + \ldots + T'_{j,r}.$$

Hence, with every refinement step, the uncertainty of T increases. Notice that during the computation of the integral, this update operation might be performed several million times! Otherwise, if the sum T is not updated in every step but only computed once in the end, the current approximation is not available for extrapolation and error estimation.

There are several approaches to *destillation* which turns out to be an important step in almost any numerical algorithm. A whole family of algorithms perform a more or less strict pre-sorting step to order the elements by their magnitude and compute the sum by exploiting this special configuration [Kul81, Pri92]. These algorithms always have a $O(n\,\log n)$ time complexity. Another draw-back is that in order to apply such algorithms, all n elements have to be stored and cannot be processed on the fly.

A second class of algorithms uses special high precision fixed point arithmetics to compute the sum and project the result back into floating point representation [Boh90]. Those approaches are usually machine dependent – in the extreme they are designed for hardware implementation. More portable algorithms should be formulated relative to a specific floating point arithmetics standard (e.g. IEEE 754) and can work on any computer providing this standard [Kob94].

We present a simple algorithm which allows to compute arbitrarily large sums without any rounding errors by just using standard IEEE arithmetic. The scheme has linear time complexity $O(n)$ with a small constant $1+\varepsilon$ and does not depend on custom designed long integer arithmetics. It is an improved version of [Kob94]. Its software implementation only needs a small amount of intermediate storage and its simplicity makes it also a candidate for hardware realization.

The basic idea of the proposed algorithm can be described as follows. A floating point number is given by a sign, a p-bit mantissa, and an exponent (all being integer). It can be observed that the addition (or subtraction) of two floating point numbers does not produce any roundoff error if both have the same exponent and both mantissae have the same parity. For subtraction this is trivial since subtraction on the same scale is simply integer arithmetic plus normalization. Adding two p-bit integers yields a $(p+1)$-bit integer which is subject to rounding (back to p-bits). However, since both addends have the same parity, their sum will be an even number and the rounding step cuts off a zero.

We can use the exponent of a floating point number combined with its mantissa's parity bit as a hashing key. Storing all incoming addends into a hash table, we know that if a collision occurs, the two numbers can be processed without rounding error. The table has to be large enough to hold two times the number of valid exponents of floating point numbers. In IEEE standard this means, we need 2 KByte (single precision) or 32 KByte (double precision) to store the whole accumulator table.

```
insert(value x,table t)
  { i = parity(x) + 2 * exponent(x)
    while (t[i] != 0.0)
      { x    = x + t[i]
        t[i] = 0.0
        i    = parity(x) + 2 * exponent(x) }
    t[i] = x }
```

The hashing takes constant time per access: Parity and exponent of a floating point number x can be isolated by masking out the corresponding bit-fields. Once all addends are inserted, we have a table, with a few remaining entries orderd according to their exponent's magnitude. In [Dek71] it is shown that for two floating point numbers a and b with $|a| \geq |b|$ the rounding error of the floating point operation $a+b$ can be computed exactly (in floating point arithmetic) by $b - ((a+b) - a)$. This remains true as long as the exponent of a is not smaller than the exponent of b. Hence we can add the hashtable's two top entries and store the result as well as the exact rounding error back into the table. By repeating this operation, we obtain the best approximation to the true sum which is representable as a floating point number together with an expansion of the approximation error (the remaining entries).

```
sum(table t)
  { i = search_downwards_from(TOP)   % first
    j = search_downwards_from(i-1)   % second
    while (j>=BOTTOM)
      { if (t[i] + t[j] != t[i])
          { a = t[i] + t[j]
            b = t[j] - (a - t[i])
            delete t[i],t[j]
            insert(a,t)
            insert(b,t)
            i = search_downwards_from(i+3) }
        else
          { i = j }
        j = search_downwards_from(i-1) }
    i = search_downwards_from(TOP)
    j = search_downwards_from(i-1)
    return t[i]+t[j] }
```

The destillation algorithm has linear complexity since no auxilliary additions are necessary. To sum up n numbers, a minimum of $n-1$ additions is always necessary. While hashing the addends we occasionally perform additions and end up with m remaining numbers in the table. Hence, $n-m$ additions have taken place so far. The only computational overhead stems from the evaluation of the hashing function. The number m does not depend on n but rather on the orders of magnitude which are covered by the addends. Summing up the remaining numbers is an $O(m)$ process with $m \ll n$.

Reformulation of Physical Properties as Functionals

The Divergence Theorem provides a tool to transform the volume based physical properties into proper integral expressions of the form (1):

$$\int_V \nabla G \; dx\, dy\, dz \;=\; \int_{\partial V} G^T N \; d\sigma \qquad (5)$$

where $N : \partial V \to \mathbb{R}^3$ is the unit-length outer normal vector on ∂V and $G : \mathbb{R}^3 \to \mathbb{R}^3$ is an arbitrary vector field (cf. [TS80]). Replacing N by the cross product of the partial derivatives of Φ and transforming (5) into an integral over the parameter domain $\Omega = \bigcup_i \Omega_i$ yields

$$\int_V \nabla G \; dx\, dy\, dz \;=\; \int_\Omega G(\Phi)^T \left[\Phi_u \times \Phi_v\right] du\, dv.$$

which reduces the reformulation of physical properties to the construction of a function G such that ∇G fits to the volume integral which defines the property.

The volume of an object is given by the integral over $\nabla G = 1$ and a suitable (symmetric) choice is $G := \frac{1}{3}(x, y, z)$. Analogously, we may chose $G := \frac{1}{2}(x^2, 0, 0)$ for the center of gravity's x-coordinate ($\nabla G = x$) and for the other coordinates respectively. To compute the moments of inertia with respect to the z-axis, we use $G := \frac{1}{3}(x^3, y^3, 0)$, i.e., $\nabla G = x^2 + y^2$, and for the x- and y-axis respectively.

Conclusion

The ingredients presented in this paper yield a framework to implement a robust and efficient algorithm for the numerical evaluation of functionals on surfaces. Efficiency is obtained through a modification of the Romberg-quadrature which does not require to subdivide the domain uniformly but allows to use a selective refinement strategy. Volume based physical properties are reformulated as surface integrals which strongly reduces the complexity. The implementation becomes reliable and robust by including an algorithm to sum up the intermediate results without rounding errors. This exact arithmetic does not affect the performance of the algorithm significantly but it prevents the quadrature algorithm from accumulating rounding errors. This is important for engineering applications since rounding errors are not covered by error bounds emerging from numerical integration theory.

References

[Boh90] G. Bohlender, *What do we need beyond IEEE-arithmetics?*, Comp. Arith. and Self-validating Numerical Methods, pp. 1–32, Academic Press, New York 1990

[Bul66] R. Bulirsch / J. Stoer, *Asymptotic Upper and Lower Bounds for Results of Extrapolation Methods*, Numer. Math. 8(1966) pp. 93-104

[Bul67] R. Bulirsch / J. Stoer, *Numerical Quadrature by Extrapolation*, Numer. Math. 9(1967) pp. 271-278

[Dek71] T. Dekker, *A floating-point technique for extending the available precision*, Numerische Mathematik 18 (1971), pp. 224–242

[Gui96] L. Guibas, *Implementing Geometric Algorithms Robustly*, Proc. Workshop on Applied Computational Geometry, pp. 15–22, Springer Verlag, 1996

[For93] S. Fortune, *Progress in Computational Geometry*, Directions in Geometric Computing, R.R. Martin ed. pp. 81-128, Information Geometers Ldt. 1993

[For96] S. Fortune, *Robustness issues in geometric algorithms*, Proc. Workshop on Applied Computational Geometry, pp. 9–14, Springer Verlag, 1996

[FW93] S. Fortune / J. Van Wyk, *Efficient Exact Arithmetics for Computational Geometry*, Proc. of the 9th annual symposium on Computational Geometry, pp. 163–172, ACM 1993

[Kob92] L. Kobbelt, *Iterative Berechnung metrischer Eigenschaften*, Thesis, University of Karlsruhe, Germany

[Kob94] L. Kobbelt, *A fast dot-product algorithm with minimal rounding errors*, Computing 52 (1994), pp. 355–369, Springer Verlag 1994

[Kul81] U. Kulisch / W. Miranker, *Computer Arithmetic in Theory and Practice*, Academic Press, New York, 1981

[Pri92] D. Priest, *On properties of floating point arithmetics*, Thesis, University of California at Berkeley, 1992

[She96] J. Shewchuk, *Adaptive precision floating point arithmetic and fast robust geometric predicates*, Report CMU-CS-96-140, Carnegie Mellon University, Pittsburgh, PA

[Slu82] A. van der Sluis, *Asymptotic expansions for quadrature errors over a simplex*, Numerical Integration, proceedings of the conference held at the Mathematisches Forschungsinstitut Oberwolfach Birkhäuser Verlag 1982 pp. 222-240

[Sto83] Josef Stoer, *Einführung in die Numerische Mathematik I*, Springer-Verlag 1983

[Str71] A. H. Stroud, *Approximate Calculation of Multiple Integrals*, Prentice-Hall 1971

[TS80] Timmer / Stern, *Computation of global geometric properties of solid objects*, Computer Aided Design 12(1980) pp. 301-304