Why interval arithmetic is so useful

Younis Hijazi[1], Hans Hagen[1], Charles Hansen[2], and Kenneth I. Joy[3]

[1] University of Kaiserslautern & IRTG 1131
Department of Computer Sciences
D-67653 Kaiserslautern, Germany
http://www-hagen.informatik.uni-kl.de

[2] SCI Institute
University of Utah
http://www.cs.utah.edu/ hansen/

[3] Institute for Data Analysis and Visualization
Computer Science Department
University of California, Davis
http://graphics.cs.ucdavis.edu/ joy/

Abstract: *Interval arithmetic* was introduced by Ramon Moore [Moo66] in the 1960s as an approach to bound rounding errors in mathematical computation. The theory of interval analysis emerged considering the computation of both the exact solution and the error term as a single entity, i.e. the interval. Though a simple idea, it is a very powerful technique with numerous applications in mathematics, computer science, and engineering. In this survey we discuss the basic concepts of interval arithmetic and some of its extensions, and review successful applications of this theory in particular in computer science.

1 Introduction

In this paper we survey interval arithmetic—a very powerful technique for controlling errors in computations—and some of its extensions in the context of self-validated arithmetics. An immediate first-order extension of interval arithmetic (IA) is known as affine arithmetic (AA) and many algorithms using interval techniques compare these two approaches; in practice there is a trade-off between accuracy and speed. Moreover, there are a lot more interval-based alternatives to IA or AA such as Taylor-based arithmetics [Neu03] or extensions of AA that are numerically more stable (e.g. Messine [Mes02]) that we will briefly review in this paper. Our central interest is to show how these interval techniques can be used in practical applications in computer science to provide robust algorithms without necessarily sacrifying speed, as interactivity or real-time are often desired especially in computer graphics and visualization.

2 Interval arithmetic

2.1 A brief history

This section is largely inspired by G. W. Walster's article *Introduction to Interval Arithmetic* [Wal97], as it perfectly introduces how IA emerged.

Ramon E. Moore conceived interval arithmetic in 1957, while an employee of Lockheed Missiles and Space Co. Inc., as an approach to bound rounding errors in mathematical computation. Forty years later, at April 19, 1997 kick-off meeting of Sun Microsystems' interval arithmetic university R & D program, he explained his thinking as follows: in 1957 he was considering how scientists and engineers represent measurements and computed results as $\tilde{x} \pm \epsilon$, where $\tilde{x}$ is the measurement (or result) and ϵ is the error tolerance.

While representing fallible values using the $\tilde{x} \pm \epsilon$ notation is convenient, computing with them is not, even in a case as simple as calculating the area of a room. If the errors due to finite precision arithmetic are simultaneously taken into account, complexity increases further. Error analyses of large scientific, engineering and commercial algorithms are sufficiently complex and labor intensive that they are often not conducted. The result is that machine computing with floating-point arithmetic is not tightly linked to mathematics, science, commerce or engineering.

Moore had a better idea. He reasoned that since $\tilde{x} \pm \epsilon$ consists of two numbers, $\tilde{x}$ and ϵ, why not use two different numbers to represent exactly the same information? That is, instead of $\tilde{x} \pm \epsilon$, use $\tilde{x} - \epsilon$ and $\tilde{x} + \epsilon$, which define the endpoints of an interval containing the exact quantity in question, i.e. x. It was this simple, yet profound, idea that started interval arithmetic and interval analysis, the branch of applied mathematics developed to numerically analyze interval algorithms.

One of the most famous references on IA is probably Moore's *Interval Analysis* book [Moo66] but there are also several more recent surveys introducing IA, e.g. [CMN02, Wal97].

2.2 What is interval arithmetic?

In the same way classical arithmetic operates on real numbers, interval arithmetic defines a set of operations on intervals. We denote an interval as $\underline{\overline{x}} = [\underline{x}, \overline{x}]$, and the base arithmetic operations are as follows:

$$\begin{aligned}
\underline{\overline{x}} + \underline{\overline{y}} &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}],\\
\underline{\overline{x}} - \underline{\overline{y}} &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}],\\
\underline{\overline{x}} \times \underline{\overline{y}} &= [min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})].
\end{aligned}$$

Other operations such as division can be devised in a similar fashion; in the case of division by an interval containing zero, special care is needed, the same way as for real-number floating point arithmetic.

Three properties of intervals and interval arithmetic make it possible to precisely link the fallible observations of science and engineering to mathematics and floating-point arithmetic ([Wal97]):

1. Any contiguous set of real numbers (a continuum) can be represented by a containing interval.
2. Intervals provide a convenient and mechanical way to represent and compute guaranteed error bounds using fallible data.
3. All the important properties of infinite precision interval arithmetic can be preserved using finite precision numbers and directed rounding, which is commonly available on most computers (indeed, any machines supporting the IEEE 754 floating-point standard).

2.3 Why interval arithmetic?

There are usually three sources of error while performing numerical computations: rounding, truncation and input errors. In the following examples (taken from [CMN02]) we show how IA is meant to keep track of them.

- *Rounding errors*:
  Consider the expression $f(x) = 1-x+\frac{x^2}{2}$ with $x = 0.531$, i.e. with 10^{-3} precision. Computing this expression with classical arithmetic gives the result $f(x) = 0.610$. Now, if we perform the computations using IA, we get $f(x) = 0.469 + \frac{0.531^2}{2} \in 0.469 + \frac{[0.281, 0.282]^2}{2}$ and so $f(x) \in 0.469 + [0.140, 0.141] = [0.609, 0.610]$. This guarantees that the exact result is within the interval $[0.609, 0.610]$.

- *Truncation errors*:
  We are now interested in a Taylor series of the exponential function: $e^x = 1 + x + \frac{x^2}{2!}e^t$ where $t \in [0, x]$. For $x < 0$, $e^x \in 1 + x + \frac{x^2}{2!}[0, 1]$. In particular, with $x = -0.531$, we get $e^{-0.531} \in 1 - 0.531 + \frac{(-0.531)^2}{2!}[0, 1] = 0.469 + [0.140, 0.141][0, 1] = [0.469, 0.610]$. This example illustrates how IA keeps track of both, the rounding and the truncation errors.

- *Input errors*:
  Suppose that due to data uncertainty our input is $x \in [-0.532, -0.531]$. If we evaluate the previous expression we obtain $e^x \in 1+[-0.532, -0.531]+\frac{[-0.532, -0.531]^2}{2}[0, 1] = [0.468, 0.470] + \frac{[0.280, 0.284]^2}{2}[0, 1] = [0.468, 0.470] + [0, 0.142] = [0.468, 0.612]$. This final example illustrates how IA can keep track of all error types simultaneously.

Moreover IA does not suffer from any restriction to a particular class of functions that it can be applied to. Indeed Moore's fundamental theorem of interval arithmetic [Moo66] states that for any function f defined by an arithmetical expression, the corresponding interval evaluation function F is an *inclusion function* of f:

$$F(\underline{\overline{x}}) \supseteq f(\underline{\overline{x}}) = \{f(x) \mid x \in \underline{\overline{x}}\}.$$

Given an implicit function f and a n-dimensional bounding box B defined as a product of n intervals, we have a very simple and reliable rejection test for the box B not intersecting the image of the function f (e.g. surface or volume),

$$0 \notin F(B) \Rightarrow 0 \notin f(B).$$

As shown in Figure 1(a), "point sampling" fails as a robust rejection test on non-monotonic intervals. While many methods exist for isolating monotonic regions or approximating the solution, inclusion methods using interval or affine arithmetic are among the most robust and general.

The inclusion property can be used in ray tracing or mesh extraction for identifying and skipping empty regions of space. Note that although $0 \notin F(B)$ guarantees the absence of a root on an interval B, the converse does not necessarily hold: one can have $0 \in F(B)$ without B intersecting f. When $F(B)$ loosely bounds the convex hull, as in Figure 1(b), IA makes for a poor (though still reliable) rejection test. This overestimation problem is a well-known disadvantage, and is fatal for algorithms relying on iterative evaluation of non-diminishing intervals.

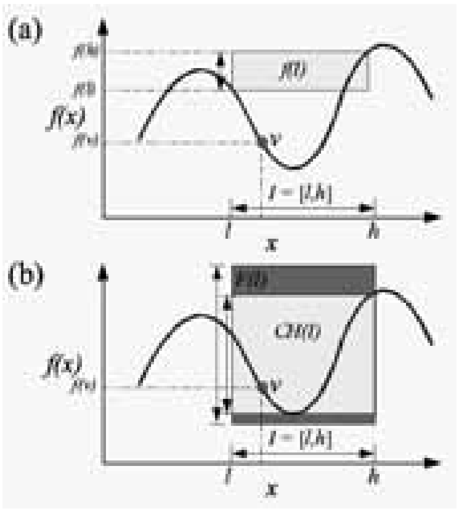


Figure 1: *Inclusion property of interval arithmetic.* (a) Floating point arithmetic is insufficient to guarantee a convex hull over the range. (b) IA is much more robust by encompassing all minima and maxima of the function within that interval. Ideally, $F(I)$ is equal or close to the bounds of the convex hull, $CH(I)$. Here the box B is simply the interval I.

Fortunately, the overestimation error is proportional to domain interval width; therefore IA guarantees (linear) convergence to the correct solution when interval domains diminish. This is the case in algorithms such as sweeping computation of hierarchically subdivided domains [Duf92, HB07], and ray tracing algorithms involving recursive interval bisection [Mit90, CHMS00, KHW$^+$07] as we will see in Section 5. Though the overestimation problem affects the efficiency of these algorithms, recursive IA methods robustly detect the zeros of a function, given an adequate termination criterion such as a sufficiently small precision ϵ over the domain, or tolerance δ over the range.

Effectively, it suffices to implement a library of these IA operators, and substitute them for the real operators, producing an *interval extension* F. If each component operator preserves the inclusion property, then arbitrary compositions of these operators will as well. As a result, literally any computable function may be expressed as interval arithmetic. Some operations are ill-defined, such as empty-set or infinite intervals. However, these are easily handled in a similar fashion to real-number floating point arithmetic.

In literature we often read "inclusion algebra" or "self-validated arithmetic" when referring to IA and the IA extension is often referred to as the *natural inclusion function*, but it is neither the only mechanism for defining an inclusion algebra, nor always the best. Particularly in the case of multiplication, it greatly overestimates the actual bounds of the range. To overcome this, it is necessary to represent intervals with higher-order approximations.

3 Extensions of interval arithmetic

In this section we discuss extensions of IA. We here focus on first-order arithmetics that can be alternatives to IA for example when higher accuracy in the computations is needed.

3.1 Affine arithmetic

Affine arithmetic was developed by Comba & Stolfi [CS93] to address the bound overestimation problem of IA. Intuitively, if IA approximates the convex hull of f with a bounding box, AA employs a piecewise first-order bounding polygon, such as a parallelogram (see Figure 2).

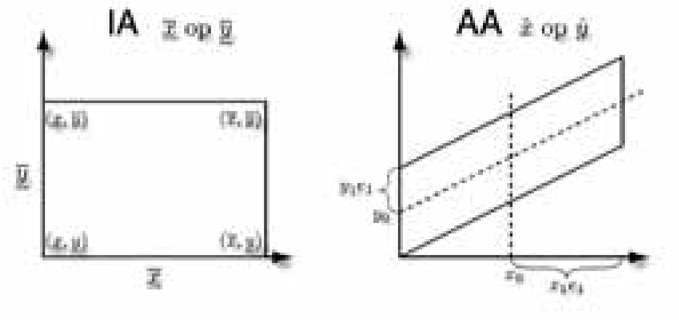


Figure 2: *Bounding box of IA versus AA.*

An affine quantity $\hat{x}$ takes the form

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i e_i$$

where the $x_i, \forall i \geq 1$, are the *partial deviations* of $\hat{x}$, and $e_i \in [-1, 1]$ are the *error symbols*.

Base affine operations in AA are as follows (where c is a real constant):

$$\begin{aligned}
\mathbf{c} \times \hat{x} &= \mathbf{c}x_0 + \mathbf{c}\sum_{i=1}^{n} x_i e_i,\\
\mathbf{c} \pm \hat{x} &= (\mathbf{c} \pm x_0) + \sum_{i=1}^{n} x_i e_i,\\
\hat{x} \pm \hat{y} &= (x_0 \pm y_0) \pm \sum_{i=1}^{n} (x_i \pm y_i) e_i.
\end{aligned}$$

However, *non-affine* operations in AA cause an additional error symbol e_z to be introduced. This is the case in multiplication between two affine forms,

$$\hat{x} \times \hat{y} = x_0 y_0 + \sum_{i=1}^{n} (x_i y_0 + y_i x_0) e_i + rad(\hat{x}) rad(\hat{y}).$$

Other operations in AA, such as square root and cosine, approximate the range of the IA operation using a first-derivative regression curve. These also compute a new error symbol.

The chief improvement in AA comes from maintaining correlated error symbols as orthogonal entities. This effectively allows error among correlated symbols to diminish, as opposed to always increasing monotonically in IA. Figure 3(b) shows how AA be much more accurate than IA (by providing tighter bounds) e.g. in curve approximation.

Conversion between IA and AA An affine form is created from an interval as follows:

$$\begin{aligned}
x_0 &= (\overline{x} + \underline{x})/2,\\
x_1 &= (\overline{x} - \underline{x})/2,\\
x_i &= 0, \;\; i > 1.
\end{aligned}$$

and can equally be converted into an interval $\underline{\overline{x}} = [x_0 - rad(\hat{x}), x_0 + rad(\hat{x})]$ where the *radius* of the affine form is given as

$$rad(\hat{x}) = \sum_{i=1}^{n} |x_i|.$$

Affine arithmetic is only one example of first-order interval-based approximation and this topic is still an active research area that concentrates on finding a trade-off between accuracy and computational cost. Other popular first-order arithmetics are E. R. Hansen's generalized interval arithmetic [Han75] and its centered form variant [Neu03], and first-order Taylor arithmetic [Neu03]. All these methods provide over-approximations; when combined with under-approximations, they can prove useful e.g. in static analysis [GP07].

3.2 Extended affine arithmetic

There are many possible extensions of IA or AA; here we will illustrate only one. In the following example we present the so-called AF1 formulation by Messine [Mes02] in which, for some constant n, a truncated affine form is given as

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i e_i + x_{n+1} e_{n+1}.$$

The arithmetic operations are given by

$$\mathbf{c} \pm \hat{x} = (\mathbf{c} \pm x_0) + \sum_{i=1}^{n} x_i e_i + |x_{n+1}| e_{n+1},$$

$$\hat{x} \pm \hat{y} = (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i) e_i + (x_{n+1} + y_{n+1}) e_{n+1},$$

$$\mathbf{c} \times \hat{x} = (\mathbf{c} x_0) + \sum_{i=1}^{n} \mathbf{c} x_i e_i + |c x_{n+1}| e_{n+1},$$

$$\hat{x} \times \hat{y} = (x_0 y_0) + \sum_{i=1}^{n} (x_o y_i + y_0 x_i) e_i + (|x_0 y_{n+1}| + |y_0 x_{n+1}| + rad(\hat{A}) rad(\hat{B})) e_{n+1}.$$

In practice we noticed that this formulation is far more efficient than pure AA, providing decent results even when using $n = 1$ or $n = 2$. Messine also developed an extension of AF1 denoted by AF2; for details see [Mes02].

Previous introduced interval-based approaches were all zero- or first-order. Note that there are also higher-order methods such as the quadratic form (QF) by Messine. [Mes02] (whose formulation becomes more and more complicated) or Taylor-based arithmetics (see Gavriliu's PhD thesis [Gav05]).

4 Discussion on IA-based techniques

After introducing these different interval algebras we might wonder which one should be used in practice. In this section we aim at providing answers to this question (based on experiments) and provide pointers to existing IA-like implementations.

In practice we noticed that despite providing less accurate results, IA is much more robust than AA and, in particular, has better numerical stability. As previously mentioned IA is especially bad when computing interval products. In this case, simple AA can achieve double the performance for the same cost. But for all the other operations the difference is not that important. An interesting approach might be having a hybrid IA/AA technique which would always use the optimal one. Indeed we often have to make a trade-off in practice between accuracy and efficiency. In short, IA has a linear convergence—which is considered slow in most applications—but is robust whereas AA algebras have a better accuracy at a more important computational cost and less stability than IA.

There are many IA-like libraries available and the most popular ones are written in C++, e.g. Filib++ [LTG$^+$06] for interval arithmetic and LibAffa [GCH00] for affine arithmetic. Those libraries are convenient for experimenting with IA—as one basically needs to replace `float` or `double` by `interval`—but they can be slow as carrying a lot of unnecessary operations for the desired application; in this case, one might prefer to implement our own library, e.g. as for the interactive ray tracing algorithm using IA [KHW$^+$07]. If completeness is required there are several references which provide a complete IA, e.g. [Kul08]. Also, if function derivatives are needed in a particular application, IA can be combined with automatic differentiation (AD) and thus evaluate the interval function and its interval derivative at the same time.

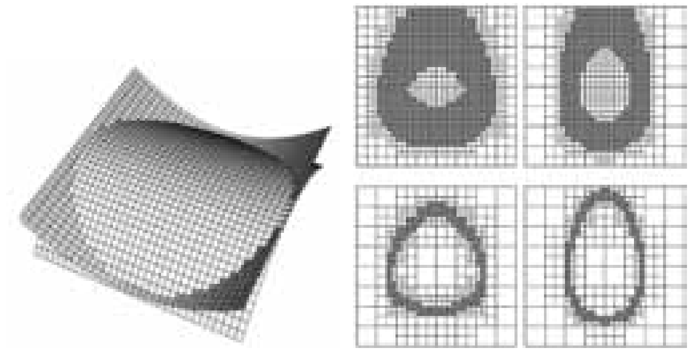
5 Successful applications in computer science

IA-based algorithms have been used successfully to address many problems. Here we review four selected applications which are pattern recognition, computational geometry, mesh extraction and finally ray tracing.

5.1 Pattern recognition and computational geometry

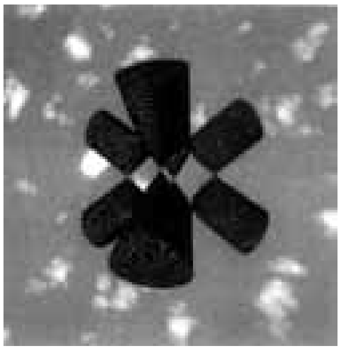
In computer vision, exploration of arrangements by subdivision methods has been used for computing globally optimal solutions to geometric matching problems under bounded error using interval arithmetic as a key ingredient in the algorithm [Bre03].

In computational geometry, e.g. for computing the intersection of curves or surfaces, IA has been employed for robustly finding those roots. In [dF96] de Figueiredo applies AA for robustly intersecting parametric surfaces 3(a). He uses a quadtree decomposition of the domain for the output. He also compares IA and AA in performance 3(b) and comes to the conclusion that AA is better suited for this particular task.



(a) (b)

Figure 3: [dF96]: (a) Surface intersection using AA. (b) IA (top) versus AA (bottom).


(a)

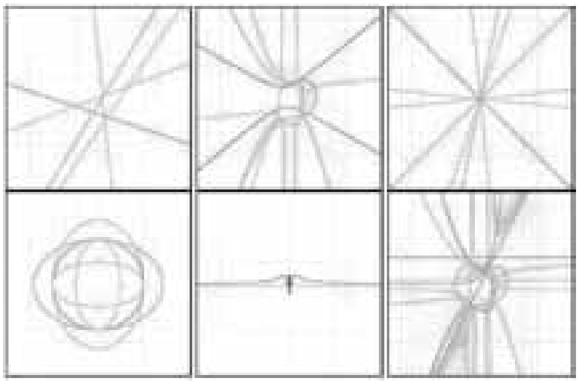

(b)

Figure 4: (a) [Duf92]: CSG ray tracing using IA. (b) [HB07]: Arrangement of curves using IA.

Other computational geometry examples using interval arithmetic-based recursive subdivisions are CSG operations for implicit surfaces [Duf92] (see Figure 4(a)) and arrangements of implicit curves [HB07] (see Figure 4(b)). In their paper, Hijazi et al. provide a method for computing arrangements of implicitly defined curves. The new method for computing arrangements is an adaptation of methods successfully used for the exploration of large, higher dimensional, non-algebraic arrangements in computer vision. While broadly similar to subdivision methods in computational geometry, its design and philosophy are different; for example, it replaces exact computations by subdivision and interval arithmetic computations and prefers data-independent subdivisions. It can be used (and is usually used in practice) to compute well-defined approximations of arrangements, but can also yield exact answers for specific problem classes. For a brief survey on arrangement of curves see [Hij06].

5.2 Mesh extraction

Lopes et al. [LJdF01] present an algorithm for computing a robust adaptive polygonal approximation of an implicit curve in the plane. The approximation is adapted to the geometry of the curve as the length of the edges varies with the curvature of the curve (see Figure 5). Robustness is achieved by combining interval arithmetic and automatic differentiation.

This work has been extended to robust surface approximation by Paiva et al. [PLLdF06], leading to a dual marching-cube octree-based algorithm using IA and providing topological guarantees (given a certain precision). Figure 6 illustrates this concept: green regions are topologically guaranteed whereas red regions are uncertain; in this illustration, the precision is pixel-based.

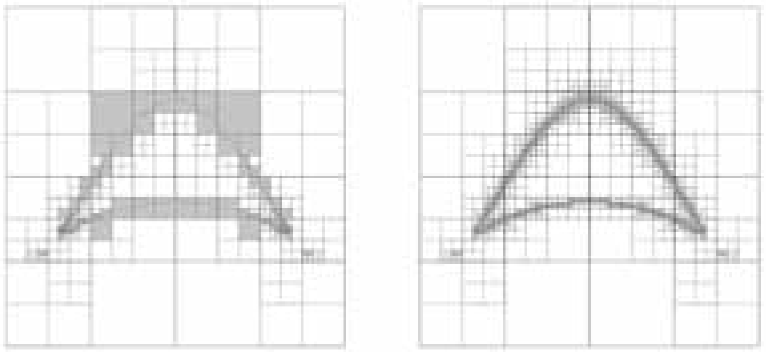


Figure 5: *[LJdF01]: Bicorn curve approximation using IA.*

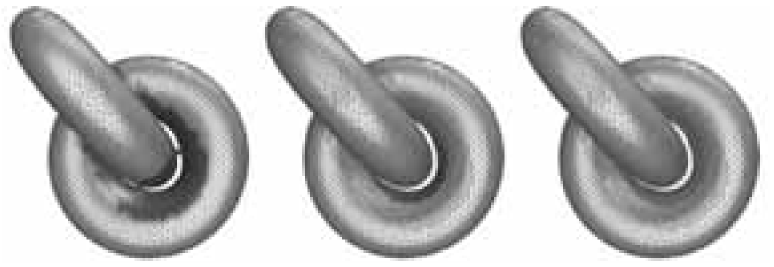
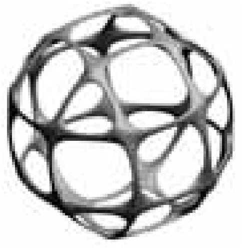
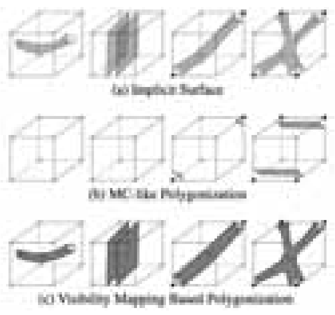


Figure 6: *[PLLdF06]: Linked tori approximation using IA.*

Another marching-cube like algorithm relying on IA and providing topological guarantees was proposed by Varadhan et al. [VKZM06]. A decocube obtained with this technique is shown in Figure 7(a). On top of interval arithmetic, the method uses a visibility map and dual contouring for extracting the mesh and compares the results with classical marching cubes (MC) in Figure 7(b).



(a)


(a) Implicit Surface
(b) MC-like Polygonization
(c) Visibility Mapping Based Polygonization

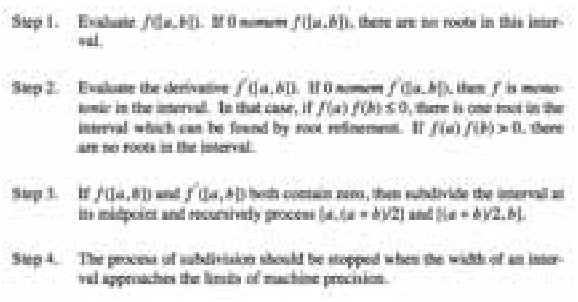
(b)

Figure 7: [VKZM06]: (a) Decocube. (b) Marching cubes.

5.3 Ray tracing

5.3.1 A brief overview

There is a large number of IA-based ray tracers and we only review some of them. The first ones who introduced IA techniques for ray tracing were Toth [JB86] and Mitchell [Mit90]. In his survey [Mit91] Mitchell applied Moore's algorithm (see Figure 8(a)) for root-finding using IA in the context of ray tracing. Indeed, ray tracing often reduces to a root-finding problem [Mit91]. This concept is illustrated by Figure 8(b) showing how ray tracing implicit surfaces reduces to solving a one-dimensional root-finding problem.



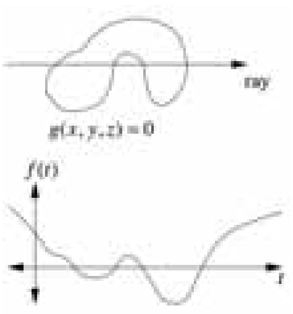
Step 1. Evaluate $f([a,b])$. If $0 \notin f([a,b])$, there are no roots in this interval.

Step 2. Evaluate the derivative $f'([a,b])$. If $0 \notin f'([a,b])$, then f is monotonic in the interval. In that case, if $f(a)\,f(b) \le 0$, there is one root in the interval which can be found by root refinement. If $f(a)\,f(b) > 0$, there are no roots in the interval.

Step 3. If $f([a,b])$ and $f'([a,b])$ both contain zero, then subdivide the interval at its midpoint and recursively process $[a,(a+b)/2]$ and $[(a+b)/2,b]$.

Step 4. The process of subdivision should be stopped when the width of an interval approaches the limits of machine precision.

(a)



(b)

Figure 8: [Mit91]: (a) Moore's root-finding algorithm. (b) Ray tracing and root-finding.

Since the 90s many other publications appeared in this area. Capriani et al. [CHMS00] combined interval bisection with various other iterative schemes, including the Interval Newton method. De Cusatis Junior et al. [dCJdFG99] used affine arithmetic to address the bound overestimation problem of pure interval arithmetic.

Sanjuan-Estrada et al. [SECG03] compared performance of two hybrid interval methods with implementations of the Interval Newton and a recursive point-sampling subdivision method in the POV-Ray framework. Figure 9 shows some results of the so-called MRFro (Minimal Root Finder reorder) algorithm (see [SECG03]). Heidrich & Seidel [HS98] used AA for ray tracing procedural displacement shaders.

5.3.2 Recent results

Recently, Gamito & Maddock [GM07] applied reduced affine arithmetic for ray casting implicit fractal surfaces. Though efficient, the proposed reduced affine arithmetic (RAA) method only preserves inclusion under specific circumstances and can only be applied to a certain class of functions. Figure 10 shows a procedural planet modeled using their technique.

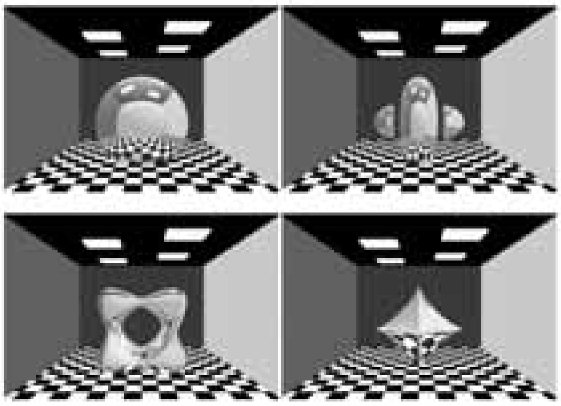


Figure 9: *[SECG03]: Up-left to down-right: sphere, Mitchell, tangle and super-ellipsoid.*

Figure 10: *[GM07]: Procedural modeling using RAA.*

Knoll & Hijazi et al. [KHW$^+$07] applied interval arithmetic for interactive ray tracing of arbitrary implicit functions. This is the first time robustness and interactivity have been jointly achieved for ray tracing implicits. Figures 11(a) and 11(b) show selected implicits ray-traced using Knoll & Hijazi et al.'s technique.

In their paper the authors present an efficient algorithm for interactive ray tracing of arbitrary implicit surfaces where IA is used for both robust root computation and guaranteed

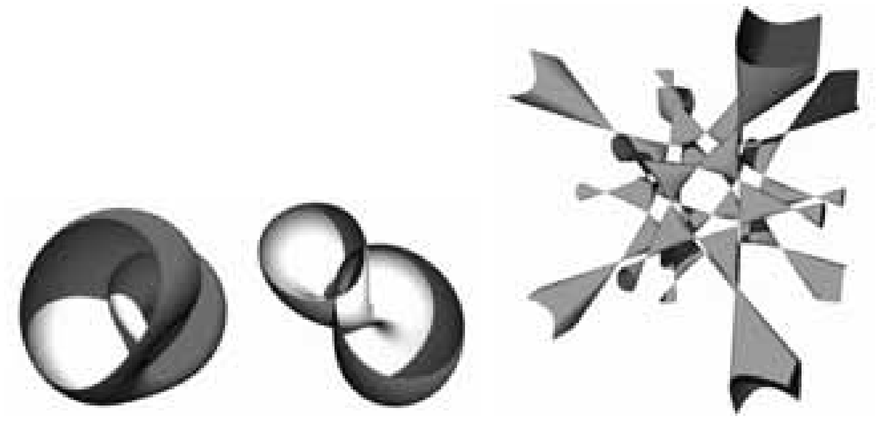

(a) (b)

Figure 11: [KHW+07]: (a) Klein bottle. (b) Barth-sextic implicit.

detection of topological features. In conjunction with ray tracing, this allows for rendering literally any programmable implicit function simply from its definition. The proposed method requires neither special hardware, nor preprocessing or storage of any data structure. Efficiency is achieved through SIMD optimization of both the interval arithmetic computation and coherent ray traversal algorithm, delivering interactive results even for complex implicit functions.

Because they neither pre-compute an explicit representation of the object, nor a physical acceleration structure in memory, they have great flexibility in rendering dynamically changing N-dimensional implicits. Figure 12 demonstrates an animated $4D$ implicit (morphing between a two-sheeted hyperboloid and a torus).

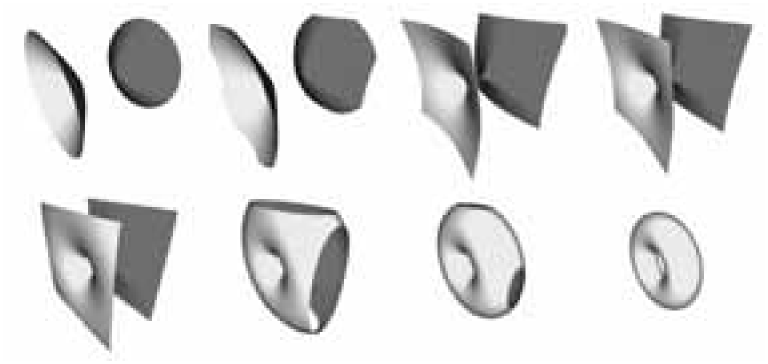


Figure 12: *[KHW+07]: Morphing performed "on the fly".*

6 Conclusion

As we tried to demonstrate in this brief survey, interval-based techniques can be very useful in computer science, e.g. in computer vision, computational geometry, mesh extraction or ray-tracing. Ten years ago, the first ray-tracing algorithms using IA as a key ingredient appeared and took advantage of IA's inherent robustness and adaptivity by applying it to concrete problems. Though robust, many of those algorithms suffered from a lack of speed and thus implying little interest in the graphics community. More recently, with the increasing computational power of CPUs and GPUs, interval techniques are gaining attention as being now able to provide both fast and robust algorithms. We hope that the computer science community, especially in computer graphics and visualization, will from now on tend to increasingly include IA techniques in their algorithms.

Acknowledgements This work was supported by the German Science Foundation (DFG IRTG 1131) as part of the International Research Training Group on *Visualization of Large and Unstructured Data Sets—Applications in Geospatial Planning, Modeling, and Engineering*.

References

[Bre03] T. M. Breuel. On the Use of Interval Arithmetic in Geometric Branch-and-Bound Algorithms. *Pattern Recognition Letters*, 24(9-10):1375–1384, 2003.

[CHMS00] O. Capriani, L. Hvidegaard, M. Mortensen, and T. Schneider. Robust and Efficient Ray Intersection of Implicit Surfaces. *Reliable Computing*, 6:9–21, 2000.

[CMN02] O. Caprani, K. Madsen, and H. B. Nielsen. Introduction to Interval Analysis. *Informatics and Mathematical Modelling*, Technical University of Denmark, Nov 2002.

[CS93] J. L. D. Comba and J. Stolfi. Affine arithmetic and its applications to computer graphics. In *Proc. SIBGRAPI'93 â VI Brazilian Symposium on Computer Graphics and Image Processing, 9â18*, 1993.

[dCJdFG99] A. de Cusatis Junior, L. de Figueiredo, and M. Gattas. Interval Methods for Raycasting Implicit Surfaces with Affine Arithmetic. In *Proceedings of XII SIBGRPHI*, pages 1–7, 1999.

[dF96] L. H. de Figueiredo. Surface intersection using affine arithmetic. In *Graphics Interface*, pages 168–175, May 1996.

[Duf92] T. Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 131–138, New York, NY, USA, 1992. ACM Press.

[Gav05] M. Gavriliu. *Towards more efficient interval analysis corner forms and a remainder interval Newton method.* PhD thesis, California Institute of Technology, Pasadena, California, 2005.

[GCH00] O. Gay, D. Coeurjolly, and N. J. Hurst. Libaffa—C++ Affine Arithmetic Library for GNU/Linux, 2000.

[GM07] M. N. Gamito and S. C. Maddock. Ray casting implicit fractal surfaces with reduced affine arithmetic. *Vis. Comput.*, 23(3):155–165, 2007.

[GP07] E. Goubault and S. Putot. Under-approximations of computations in real numbers based on generalized affine arithmetic. In *Proceedings of SAS'07*, Copenhagen, August 2007.

[Han75] E. R. Hansen. A Generalized Interval Arithmetic. In *Proceedings of the International Symposium on Interval Mathemantics*, pages 7–18, London, UK, 1975. Springer-Verlag.

[HB07] Y. O. Hijazi and T. M. Breuel. Computing Arrangements using Subdivision and Interval Arithmetic. In *P. Chenin, T. Lyche and L.L. Schumaker (Eds.), Proceedings of the Sixth International Conference on Curves and Surfaces, June 29-July 5, 2006, Avignon, France, Curve and Surface Design: Avignon 2006, Nashboro Press*, pages 173–182, 2007.

[Hij06] Y. O. Hijazi. Arrangements of Planar Curves. In *H. Hagen, A. Kerren, P. Dannenmann (Eds.), Visualization of Large and Unstructured Data Sets, Proceedings of the first workshop of DFG's International Research Training Group "Visualization of Large and Unstructured Data Sets—Applications in Geospatial Planning, Modeling, and Engineering", June 14-16, 2006, Dagstuhl, Germany, GI-Edition, Lecture Notes in Informatics, Seminars Series*, volume S-4, pages 59–68, 2006.

[HS98] W. Heidrich and H.-P. Seidel. Ray-Tracing Procedural Displacement Shaders. In *Graphics Interface*, pages 8–16, 1998.

[JB86] K. I. Joy and M. N. Bhetanabhotla. Ray tracing parametric surface patches utilizing numerical techniques and ray coherence. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 279–285, New York, NY, USA, 1986. ACM Press.

[KHW$^+$07] A. Knoll, Y. Hijazi, C. Hansen, I. Wald, and H. Hagen. Interactive Ray Tracing of Arbitrary Implicits with SIMD Interval Arithmetic. In *Proceedings of the 2nd IEEE/EG Symposium on Interactive Ray Tracing*, pages 11–18, 2007.

[Kul08] U. Kulisch. Complete Interval Arithmetic and its Implementation. In *Numerical Validation in Current Hardware Architectures*, Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[LJdF01] H. Lopes, O. Jo, and L. H. de Figueiredo. Robust Adaptive Approximation of Implicit Curves. In *SIBGRAPI '01: Proceedings of the XIV Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'01)*, page 10, Washington, DC, USA, 2001. IEEE Computer Society.

[LTG$^+$06] M. Lerch, G. Tischler, J. Wolff Von Gudenberg, W. Hofschuster, and W. Krämer. FILIB++, a fast interval library supporting containment computations. *ACM Trans. Math. Softw.*, 32(2):299–324, 2006.

[Mes02] F. Messine. Extentions of Affine Arithmetic: Application to Unconstrained Global Optimization. *Journal of Universal Computer Science*, 8(11):992–1015, 2002.

[Mit90] D. P. Mitchell. Robust Ray Intersection with Interval Arithmetic. In *Proceedings on Graphics Interface 1990*, pages 68–74, 1990.

[Mit91] D. P. Mitchell. Three applications of interval analysis in computer graphics. Course Notes for Frontiers in Rendering, Siggraph '91, 1991.

[Moo66] R. E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1966.

[Neu03] A. Neumaier. Taylor Forms-Use and Limits. *Reliable Computing*, 9:43–79(37), February 2003.

[PLLdF06] A. Paiva, H. Lopes, T. Lewiner, and L. H. de Figueiredo. Robust adaptive meshes for implicit surfaces. In *19th Brazilian Symposium on Computer Graphics and Image Processing*, pages 205–212, 2006.

[SECG03] J. F. Sanjuan-Estrada, L. G. Casado, and I. Garcia. Reliable algorithms for ray intersection in computer graphics based on interval arithmetic. In *XVI Brazilian Symposium on Computer Graphics and Image Processing, 2003. SIBGRAPI 2003.*, pages 35–42, 2003.

[VKZM06] G. Varadhan, S. Krishnan, L. Zhang, and D. Manocha. Reliable implicit surface polygonization using visibility mapping. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 211–221, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[Wal97] G. W. Walster. Introduction to Interval Arithmetic. unpublished note supporting Sun Microsystems compilers for interval programs, May 19, 1997.