Reliable Two-Dimensional Graphing Methods for Mathematical Formulae with Two Free Variables

Jeff Tupper[†]

University of Toronto

Abstract

This paper presents a series of new algorithms for reliably graphing two-dimensional implicit equations and inequalities. A clear standard for interpreting the graphs generated by two-dimensional graphing software is introduced and used to evaluate the presented algorithms. The first approach presented uses a standard interval arithmetic library. This approach is shown to be faulty; an analysis of the failure reveals a limitation of standard interval arithmetic. Subsequent algorithms are developed in parallel with improvements and extensions to the interval arithmetic used by the graphing algorithms. Graphs exhibiting a variety of mathematical and artistic phenomena are shown to be graphed correctly by the presented algorithms. A brief comparison of the final algorithm presented to other graphing algorithms is included.

CR Categories: G.1.0 [Numerical Analysis]: General—Interval Arithmetic; G.4 [Mathematical Software]: Reliability and Robustness; I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms

Keywords: interval arithmetic, Tupper interval arithmetic, interval analysis, implicit curves, algebraic curves, graphing, relation graphing, formula graphing, GrafEq

1 Introduction

The problem discussed in this paper is a familiar one to computer graphics researchers: given an equation in x and y, produce its graph. Given that this problem has been discussed for centuries, it is unsurprising that there is an abundance of partial solutions to this problem. It is, however, surprising that there is no published method capable of reliably solving this problem even if we restrict our attention to the simple equations encountered in introductory mathematics courses.

This paper presents a method that correctly graphs a wide variety of equations, including all of the equations encountered in typical introductory mathematics courses. Moreover, when confronted with a difficult equation that is beyond the capabilities of the presented method, the portions of the graph that are not known to be correct will be marked clearly.

Many students currently studying mathematics are using automated graphing tools that produce incorrect graphs for some of the equations discussed in their curricula. I have written this paper in the hope that, in the future, more students will have access to graphing tools that work correctly.

The methods described in this paper are used in GrafEq$^{\text{TM}}$ 2.10 [Ped]. Except for branch cut tracking, I first implemented these methods in 1992 and publically demonstrated their use with GrafEq 2.00 in 1993. My M.Sc. thesis [Tup96] provides more details about graphing with Tupper interval arithmetic[1] and includes a discussion of Tupper linear interval arithmetic.

2 Formula Syntax

To be general-purpose tools, our graphing methods must handle implicit equations of the form $f = g$, where f and g are given symbolically using standard mathematical operators, constants, and the variables x and y.

The operators that I have implemented for the graphing algorithms we will discuss include $+$,$-$, $\pm$, $\times$, $\div$, exponentiation, nth root, log, min, max, median, $\min_n$ (nth smallest), $\max_n$ (nth largest), $||$, $\lfloor\ \rfloor$, $\lceil\ \rceil$, !, Γ, seventy-two trigonometric (multi-)functions (the six basic functions sin, cos, tan, csc, sec, cot; their functional partial inverses Arcsin, Arccos, Arctan, Arccsc, Arcsec, Arccot; and their multi-functional inverses arcsin, arccos, arctan, arccsc, arcsec, arccot; for trigonometry based on the unit circle $x^2 + y^2 = 1$, the unit hyperbola $x^2 - y^2 = 1$, the unit diamond $|x| + |y| = 1$, and the unit square $\max(|x|, |y|) = 1$), sgn, mod, gcd, and lcm.

I have implemented the preceeding operators so that the user can directly enter a wide range of equations. Restricting the user to a special class of equations, such as algebraic equations, can allow the development of better special-purpose algorithms, but this paper presents general-purpose graphing algorithms. Arbitrarily removing some of the preceeding operators will not, in many cases, simplify the problem as the user may be able to emulate the missing operators with the remaining ones. For example, $\max(f, g) \equiv \frac{1}{2}(f + g + |f - g|)$, while $|f| \equiv \sqrt{f^2} \equiv \sqrt{ff}$.

My implementations of the graphing algorithms allow the user to enter inequalities as well as equations: any of the seven different comparisons $=$, $<$, $\leq$, $>$, $\geq$, $\lesseqgtr$, and $\lessgtr$ can be used; conjunctions, disjunctions, logical negations, and conditional definitions are also available to the user. This does not increase the true difficulty of the graphing problem

[†]The author may be reached at: Department of Computer Science, University of Toronto, 10 King's College Circle, Toronto ON M5S 1A4, Canada; or via email at mooncake@dgp.toronto.edu.

[1]I have been asked by members of the interval arithmetic community to refer to my generalization of interval arithmetic in this way to distinguish it from other generalizations of interval arithmetic.

ACM SIGGRAPH 2001, 12-17 August 2001, Los Angeles, CA, USA
© 2001 ACM 1-58113-374-X/01/08...$5.00

as these constructs can be emulated if they are not directly available,as $[f \geq 0] \equiv [f - |f| = 0]$, $[(g = 0) \wedge (h = 0)] \equiv [|g| + |h| = 0]$ and $[(g = 0) \vee (h = 0)] \equiv [gh = 0]$ if g and h are well-defined. Many other mathematical constructs can be emulated; for example, $[f \in \mathbb{Z}] \equiv [\sin(\pi f) = 0]$ and $[f(\pm x) = 0] \equiv [f(x)f(-x) = 0]$.

3 Formula Semantics

Any formula $r(x, y)$, when evaluated with specific real numbers x and y, is always either false (F) or true (T). Based on my experience discussing these topics with others, I would like to discuss a few of the rules I use to evaluate formulae before continuing so that the meaning of formulae are clear.

Over the years, I have asked many mathematicians to graph $y < \sqrt{x}$ and $y \geq \sqrt{x}$ over $[-1, 1] \times [-1, 1]$ and have always received a pair of graphs similar to those shown in figure 1. No points with a negative x coordinate have ever been included in either graph since neither $y < \sqrt{x}$ nor $y \geq \sqrt{x}$ is true when x is negative, regardless of the value of y.

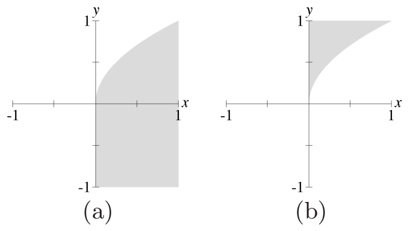


Figure 1: Graphs of (a) $y < \sqrt{x}$ and (b) $y \geq \sqrt{x}$.

My rules for evaluating expressions involving "undefined" quantities are:

1. a mathematical operator is "undefined" if any of its arguments are, and
2. the boolean result of comparing an "undefined" quantity to any other quantity, including another "undefined" quantity, using $=$, $<$, $\leq$, $>$, $\geq$, $\lessgtr$, or $\lesseqqgtr$ is F.

Here are three ways of defending these rules:

- A comparison evaluates to T if and only if the ordered pair being compared is a member of the set defining the comparison; for example, $=$ is defined by $\{(x, x) : x \in \mathbb{R}\}$. In Prolog parlance, we will make a *closed-world assumption*, namely that we have a complete understanding of all available mathematical operators. This assumption is reasonable as we have no obligation to consider any other model of arithmetic. When given with an "undefined" quantity, we cannot produce an ordered pair that is a member of the comparison's defining set, so the result of evaluation is F.
- With the mathematical operators we are considering, each partial operator can be rewritten using operators that are always well-defined. For example, the formula $y < \sqrt{x}$ may be expanded out to $\exists s[y < s] \wedge [s^2 = x] \wedge [s \geq 0]$; this formula is false when $x < 0$ and involves no undefined quantities. Rewriting all of the partial operators in this manner produces results equivalent to the above evaluation rules.
- Pragmatically, we want expressions such as $\left[y = \sqrt{x}\right] \vee \left[\left[y = \sqrt[3]{x}\right] \wedge [x < 0]\right]$ to be well-defined for all x; this occurs when using the rules above.

My rules for evaluating gcd and lcm are based on Euclid's definition. My rules for evaluating exponents with negative bases agree with those taught in many high-schools: for $a < 0$, a^b is well-defined if and only if b is equal to a rational number with an odd denominator; for $a < 0$, a^b is negative if and only if b is equal to a rational number with an odd numerator. These rules for evaluating exponents can be defended, but would take us too far afield. I do not know of any evaluation rules that prescribe different values. I accept that $0^0 = 1$. Nevertheless, the graphing techniques presented in this paper can be adapted to different evaluation rules.

4 Computed Graph Semantics

Given a mathematical formula r and a rectangular region $[L, R] \times [B, T]$, of the Cartesian plane $\mathbb{R}^2$, our graphing algorithm will produce an illustration that consists of a $W \times H$ rectangular array of pixels. We will refer to this illustration as a *computed graph* of r.

Each pixel of the computed graph represents a closed rectangular region of the plane. To simplify the ensuing discussion, we will often refer to points as being inside or outside of a pixel rather than the region the pixel represents.

It would be very natural to require correct computed graphs to satisfy the following two rules, with each pixel being either white or black:

- if a pixel is white, there are no solutions of r within the pixel; and
- if a pixel is black, there is at least one solution of r within the pixel;

A point (x, y) is a solution of r if $r(x, y)$ is true. If we place these semantics on computed graphs, users will naturally interpret computed graphs correctly. Unfortunately, these naïve semantics are unrealistic. Now that we have formalized the equation graphing problem, it can be shown that the problem, as formalized, is not computable: no fixed algorithm can produce correct black and white graphs of arbitrary formulae, even if given an arbitrary amount of time and memory. We will give our algorithms a way out by allowing them to color some pixels red:

- if a pixel is red, there may or may not be solutions of r within the pixel.

With these new semantics, computed graphs can be iteratively refined, as shown in figure 2. A graphing algorithm can begin by presenting the user with a solid red image, and then, over time, gradually reveal the behavior of r by replacing red pixels with either white pixels or black pixels. With this approach, information is presented to the user as it is discovered and the user will intuit that the red pixels show where the computer is "unsure" or "not yet finished." These precise semantics also allow us to numerically gauge the performance of different graphing algorithms.

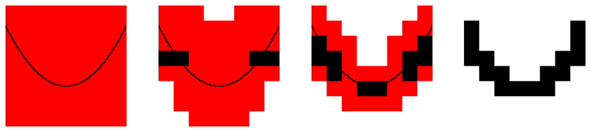


Figure 2: A sequence of computed graphs of $y = x^2 - \frac{1}{3}$ over $[-1, 1] \times [-1, 1]$ with an 8×8 pixmap; the thin black curve shows the true graph.

5 Interval Arithmetic

Our algorithms will use interval arithmetic [Moo66, Moo79, Tup96] with IEEE 754 floating-point [IEE85]. *I will use an analogous two-digit base-ten floating-point number system for numerical examples in this paper.*

The guiding principle behind interval arithmetic is to represent quantities, and perform computations, using only lower and upper bounds: a quantity q can be represented by any interval $\langle \mathtt{a}, \mathtt{b} \rangle$ with $\mathtt{a} \le q \le \mathtt{b}$, where $\mathtt{a}$ and $\mathtt{b}$ are floating-point values. I will use `a teletype font` for the members of intervals that are explicitly represented and processed by the computer. For example, π is best represented by $\langle 3.1, 3.2\rangle$, but can also be represented by $\langle 3.0, 4.0\rangle$ or even by $\langle -\infty, +\infty\rangle$; π is not represented by $\langle 3.2, 3.4\rangle$ or by $\langle 3.2, 3.1\rangle$. IEEE 754 provides both $-\infty$ and $+\infty$ as floating-point values.

Interval arithmetic routines work solely with the lower and upper bounds and do not require any other information about the actual quantity being represented, as illustrated by the following pseudo-code:

```
Subtract($\langle a, b\rangle$, $\langle A, B\rangle$)

1. return $\langle a -^{\downarrow} B, b -^{\uparrow} A\rangle$
```

The small arrows next to the floating-point subtraction operators specify whether the floating-point operation should round the true mathematical result down (towards $-\infty$) or up (towards $+\infty$). For example, Subtract($\langle 3.1, 3.2\rangle$, $\langle 0.11, 0.12\rangle$) returns $\langle 2.9, 3.1\rangle$ while the true mathematical result would be $\langle 2.98, 3.09\rangle$. IEEE 754 provides $+^{\downarrow}$, $+^{\uparrow}$, $-^{\downarrow}$, $-^{\uparrow}$, $\times^{\downarrow}$, $\times^{\uparrow}$, $\div^{\downarrow}$, $\div^{\uparrow}$, $\sqrt[\downarrow]{\ }$, and $\sqrt[\uparrow]{\ }$ through rounding mode control. After we have implemented interval arithmetic routines for all of the mathematical operators we allow, we can compute guaranteed bounds for arbitrary expressions; see the references cited at the start of this section for further information about implementing interval arithmetic routines.

We will use interval arithmetic with boolean values to represent, and process, the results of formula evaluations, as shown by the following pseudo-code: (GreaterThan takes floating-point intervals as arguments)

```
GreaterThan($\langle a, b\rangle$, $\langle A, B\rangle$)

1. return $\langle a > B, b > A\rangle$
```

```
And($\langle a, b\rangle$, $\langle A, B\rangle$)

1. return $\langle a \wedge A, b \wedge B\rangle$
```

Booleans are ordered with F < T; three boolean intervals are possible: $\langle \mathrm{F}, \mathrm{F}\rangle$, $\langle \mathrm{F}, \mathrm{T}\rangle$, and $\langle \mathrm{T}, \mathrm{T}\rangle$.

6 Pixel Boundaries

Since the rectangular graphing area is partitioned into a regular grid of rectangles, pixel (x, y) corresponds to the region $[L + x(R-L)W^{-1}, L + (x+1)(R-L)W^{-1}] \times [B + y(T-B)H^{-1}, B + (y+1)(T-B)H^{-1}]$, where pixel $(0,0)$ is the bottom-left pixel and pixel $(W-1, H-1)$ is the top-right pixel. As shown in figure 3, even if the bounds of the graphing area (L, R, B, and T) are given as floating-point numbers, the bounds of individual pixels may not be representable exactly using floating-point numbers.

Carrying out the calculations using floating-point, with round-to-nearest as the rounding mode, will show that the center-bottom pixel of figure 3 corresponds to $[0.33, 0.67] \times [0.0, 0.33]$ instead of $\left[\frac{1}{3}, \frac{2}{3}\right] \times \left[0, \frac{1}{3}\right]$ — some points that do belong have been excluded and some points that do not belong have been included. As this incorrect correspondence may easily cause an incorrect graph to be generated, we will *not* use this approach, which will often lead to significant errors when only a few floating-point numbers separate L and R or B and T.

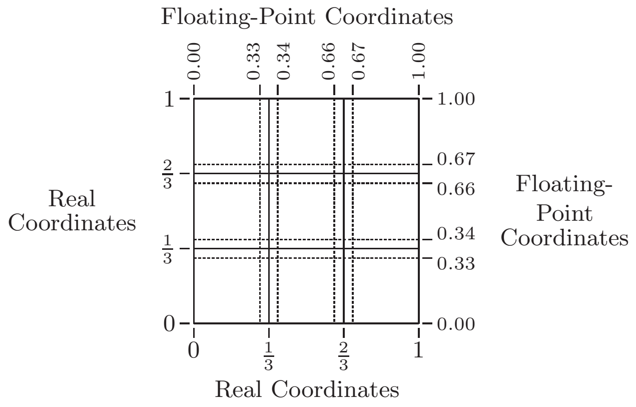


Figure 3: The graphing area $[0,1] \times [0,1]$ partitioned into a 3×3 array of pixels, not to scale.

We will instead use inner and outer bounds of the rectangular region that corresponds to each pixel.[2] The inner bounds of the center-bottom pixel of figure 3 are $[0.34, 0.66] \times [0.0, 0.33]$; the outer bounds are $[0.33, 0.67] \times [0.0, 0.34]$. Our algorithms will use the inner bounds to show the existence of solutions and the outer bounds to show the absence of solutions. Since we are using inner and outer bounds, it is natural to allow L, R, B, and T to be given using intervals; this allows $y = \sin x$ to be graphed over $[-\pi, \pi] \times [-1, 1]$ where $L = \langle -3.2, -3.1\rangle$, $R = \langle 3.1, 3.2\rangle$, $B = \langle -1, -1\rangle$, and $T = \langle 1, 1\rangle$.

7 Procedure 1

All of our graphing procedures have the same high-level structure that is shown in the pseudo-code for Graph below: the computed graph is first painted red and then a set of uncertain regions $\mathfrak{U}_k$ is maintained that keeps track of which regions of the graph are undecided.

```
Graph($r$, $L$, $R$, $B$, $T$, $W$, $H$)

1. PaintRed($[0, W]$, $[0, H]$)
2. $k \leftarrow \gcd(W, H)$
3. $k \leftarrow \lfloor \lg(\gcd(k, 2^{\lfloor \lg k \rfloor})) \rfloor$
4. $\mathfrak{U}_k \leftarrow \{ [a2^k, (a+1)2^k] \times [b2^k, (b+1)2^k]$
5.         $: (0 \le a2^k < W) \wedge (0 \le b2^k < H) \}$
6. while $(k \ge 0) \wedge (\mathfrak{U}_k \ne \emptyset)$
7.     $\mathfrak{U}_{k-1} \leftarrow$ RefinePixels($r$, $\mathfrak{U}_k$)
8.     $k \leftarrow k - 1$
```

The Graph pseudo-code given assumes that W and H are chosen / adjusted so that the starting $\mathfrak{U}_k$ has only a few regions. The RefinePixels procedure checks each of the regions in $\mathfrak{U}_k$ and colors portions of the graph black or white if the existence or absence of solutions of r can be established throughout an entire region; regions that remain undecided are subdivided and put into $\mathfrak{U}_{k-1}$ for processing later. With

[2]We could ensure that all pixel boundaries are exactly representable if we graph $r(L + x(R-L)W^{-1}, B + y(T-B)H^{-1})$ over $[0, W] \times [0, H]$ instead of graphing $r(x, y)$ over $[L, R] \times [B, T]$. But, as such preprocessing actually spreads the uncertainty that was confined to the pixel boundaries to the rest of the graphing area by introducing the inexact calculations into every evaluation of r, I have chosen to handle imprecise pixel boundaries directly.

each region $\mathfrak{u}$, r is evaluated over $\mathfrak{u}^{\uparrow} = (\langle \mathtt{l}, \mathtt{r}\rangle, \langle \mathtt{b}, \mathtt{t}\rangle)$ where, as discussed in the previous section, $[\mathtt{l}, \mathtt{r}] \times [\mathtt{b}, \mathtt{t}]$ is an outer boundary of the region represented by $\mathfrak{u}$.

```
RefinePixels($r$, $\mathfrak{U}_k$)
1. $\mathfrak{U}_{k-1} \leftarrow \emptyset$
2. for each $\mathfrak{u} \in \mathfrak{U}_k$
3.     if $(r(\mathfrak{u}^{\uparrow}) = \langle \mathrm{T}, \mathrm{T}\rangle)$ PaintBlack($\mathfrak{u}$)
4.         else if $(r(\mathfrak{u}^{\uparrow}) = \langle \mathrm{F}, \mathrm{F}\rangle)$ PaintWhite($\mathfrak{u}$)
5.         else $\mathfrak{U}_{k-1} \leftarrow \mathfrak{U}_{k-1} \cup$ FourSubsquares($\mathfrak{u}$)
6. return $\mathfrak{U}_{k-1}$
```

Figure 4 shows a sequence of computed graphs from procedure 1.

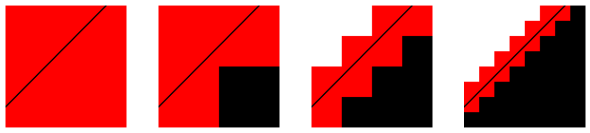


Figure 4: A sequence of computed graphs, from procedure 1, of $y < x + \frac{1}{3}$ over $[-1, 1] \times [-1, 1]$ with an 8×8 pixmap; the thin black line shows the true graph of $y = x + \frac{1}{3}$.

When graphing a formula with a one-dimensional solution set, procedure 1 only establishes the absence of solutions and not the presence of solutions, as shown in figure 5b.

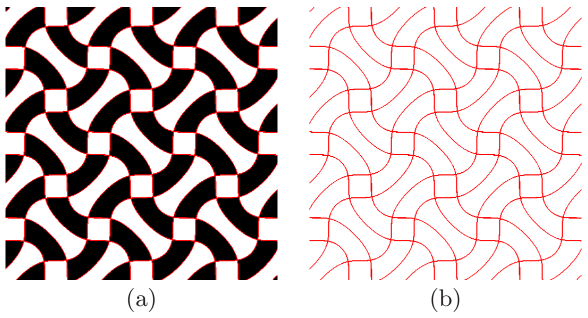


Figure 5: Computed graphs, from procedure 1, of (a) $f(x, y) > 0$ and (b) $f(x, y) = 0$, both over $[-10, 10] \times [-10, 10]$ with 512×512 pixmaps; $f(x, y) = \cos\cos\min(\sin x + y, x + \sin y) - \cos\sin\max(\sin y + x, y + \sin x)$.

One approach to removing the uncertain pixels that remain after $\mathfrak{U}_0$ has been exhausted is to declare, by fiat, that the remaining uncertain pixels are "close enough" and to color all remaining red pixels black. All other "interval" graphing approaches that I have seen take an approach similar to this — an unreliable heuristic is applied so that graphing may terminate. Although such approaches are certainly expedient, they fails to meet our standards of rigor. Instead, we will extend our procedure; but before we do, we will investigate the interval arithmetic system that provides the foundation for our procedures.

Although it may seem that our procedure is infallible, and that a correctness proof would be trivial, a little testing will reveal that we have overlooked a limitation of standard interval arithmetic. Consider the computed graph shown in figure 6; one of the elements of $\mathfrak{U}_1$ has been misclassified. The computation for that element, with pixel coordinates $[2, 4] \times [0, 2]$, is $r(\langle -0.48, 0.05\rangle, \langle -1.0, -0.47\rangle) \rightsquigarrow [\langle -1.0, -0.47\rangle < \sqrt{\langle -0.48, 0.05\rangle}] \rightsquigarrow [\langle -1.0, -0.47\rangle < \langle 0.0, 0.23\rangle] \rightsquigarrow \langle \mathrm{T}, \mathrm{T}\rangle$; the error is caused by the assumption that all quantities are well-defined. But when we take the square root of x, the result is not well-defined when $x < 0$ as we are modelling real arithmetic.

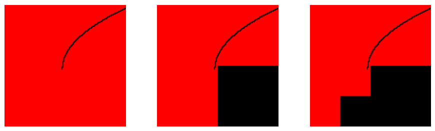


Figure 6: A sequence of computed graphs, from procedure 1, of $y < \sqrt{x}$ over $[-1, 1.1] \times [-1, 1.1]$ with an 8×8 pixmap; the thin black curve shows the true graph of $y = \sqrt{x}$. The computed graph on the right is incorrect.

When confronted with the task of computing the interval result of a mathematical operation that may not be well-defined, standard interval arithmetic libraries will halt, issue an exception, return an overly-wide result, or simply act as though the result is well-defined. Most libraries act, when possible, as though the result is well-defined, as we did above, under the assumption that only well-defined expressions will be evaluated. Returning an overly wide result, such as $\langle -\infty, +\infty\rangle$ would keep our procedure from incorrectly graphing the example above, but would not help with other graphs, such as $y < 0\sqrt{x}$, and would hinder the graphing of $y = \sqrt{x}$. Issuing an exception would not be of much direct help, as it is unclear what action our graphing procedure should then take. Halting is not a useful option for our application.

8 Domain Tracking; Algorithm 1.1

We will extend our interval arithmetic system to keep track of whether or not a quantity is well-defined. The interval $\langle \mathsf{val} \in \langle \mathtt{a}, \mathtt{b}\rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d}\rangle\rangle$ represents a quantity q

- that is well-defined if $\langle \mathtt{c}, \mathtt{d}\rangle = \langle \mathrm{T}, \mathrm{T}\rangle$,
- that is not well-defined if $\langle \mathtt{c}, \mathtt{d}\rangle = \langle \mathrm{F}, \mathrm{F}\rangle$, and
- that may or may not be well-defined if $\langle \mathtt{c}, \mathtt{d}\rangle = \langle \mathrm{F}, \mathrm{T}\rangle$;

if q is well-defined, $\mathtt{a} \leq q \leq \mathtt{b}$. The standard interval $\langle \mathtt{a}, \mathtt{b}\rangle$ corresponds to $\langle \mathsf{val} \in \langle \mathtt{a}, \mathtt{b}\rangle; \mathsf{def} \in \langle \mathrm{T}, \mathrm{T}\rangle\rangle$ if we assume that intervals always represent well-defined quantities. When evaluating r, leaf intervals have $\mathsf{def} \in \langle \mathrm{T}, \mathrm{T}\rangle$ as constants and the variables x and y are always well-defined. An interval with domain tracking is stored using two floating-point values and two boolean values; "val ∈" and "def ∈" are not stored as part of an interval and are used only as notational aides.[3] Here is example psuedo-code for a square-root routine with domain tracking:

```
SquareRoot($\langle \mathsf{val} \in \langle \mathtt{a}, \mathtt{b}\rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d}\rangle\rangle$)
1. if $(\mathtt{b} < 0)$ return $\langle \mathsf{val} \in \langle \mathtt{a}, \mathtt{b}\rangle; \mathsf{def} \in \langle \mathrm{F}, \mathrm{F}\rangle\rangle$
2.     else if $(0 \leq \mathtt{a})$ return $\langle \mathsf{val} \in \langle \sqrt[\downarrow]{\mathtt{a}}, \sqrt[\uparrow]{\mathtt{b}}\rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d}\rangle\rangle$
3.     else return $\langle \mathsf{val} \in \langle 0, \sqrt[\uparrow]{\mathtt{b}}\rangle; \mathsf{def} \in \langle \mathrm{F}, \mathtt{d}\rangle\rangle$
```

Keeping track of whether or not a represented quantity is well-defined is a simple example of *domain tracking*.

With standard interval arithmetic, procedure 1 will correctly graph any formula that is well-defined for all x and y. Algorithm 1.1 carries out the same steps as procedure 1, but uses an interval arithmetic with domain tracking. Algorithm 1.1 will graph any formula correctly.

[3] My M.Sc. thesis used a more compact notation that many have found difficult to remember and use.

9 Subpixel Computation; Algorithm 2

Our next algorithm, algorithm 2, will extend algorithm 1.1 by working with subpixel regions. We do this by adding the following three lines to Graph:

```
9.  while $\mathfrak{U}_k \neq \emptyset$
10.     $\mathfrak{U}_{k-1} \leftarrow$ RefineSubpixel($r$, $\mathfrak{U}_k$)
11.     $k \leftarrow k - 1$
```

The new routine, RefineSubpixel, is similar to RefinePixels, but it must take into account any other remaining subpixel elements.

```
RefineSubpixel($r$, $\mathfrak{U}_k$)

1.  $\mathfrak{U}_{k-1} \leftarrow \emptyset$
2.  for each $\mathfrak{u} \in \mathfrak{U}_k$
3.      $\mathfrak{p} \leftarrow$ PixelContaining($\mathfrak{u}$)
4.      if $(r(\mathfrak{u}^\uparrow) = \langle \mathrm{T}, \mathrm{T} \rangle \wedge (\mathfrak{u}^\uparrow \cap \mathfrak{p}^\downarrow \neq \emptyset))$
5.          then ShowSubpixelSolution($\mathfrak{p}$, $\mathfrak{U}_k$, $\mathfrak{U}_{k-1}$)
6.          else if $(r(\mathfrak{u}^\uparrow) = \langle \mathrm{F}, \mathrm{F} \rangle)$
7.                  then Remove($\mathfrak{u}, \mathfrak{U}_k$)
8.                          if (Absent($\mathfrak{p}, \mathfrak{U}_k$) $\wedge$ Absent($\mathfrak{p}, \mathfrak{U}_{k-1}$))
9.                              then PaintWhite($\mathfrak{p}$)
10.                 else $\mathfrak{U}_{k-1} \leftarrow \mathfrak{U}_{k-1} \cup$ FourSubsquares($\mathfrak{u}$)
11. return $\mathfrak{U}_{k-1}$
```

```
PixelContaining($[\ell, r] \times [b, t]$)

1.  return $[\lfloor \ell \rfloor, \lceil r \rceil] \times [\lfloor b \rfloor, \lceil t \rceil]$
```

```
ShowSubpixelSolution($\mathfrak{p}$, $\mathfrak{U}_k$, $\mathfrak{U}_{k-1}$)

1.  PaintBlack($\mathfrak{p}$)
2.  Remove($\mathfrak{p}$,$\mathfrak{U}_k$)
3.  Remove($\mathfrak{p}$,$\mathfrak{U}_{k-1}$)
```

Absent($\mathfrak{p}, \mathfrak{U}$) returns true if and only if no subpixel elements of $\mathfrak{p}$ are in $\mathfrak{U}$. Remove($\mathfrak{u}, \mathfrak{U}$) removes $\mathfrak{u}$ from $\mathfrak{U}$; Remove($\mathfrak{p}, \mathfrak{U}$) removes all subpixel elements of $\mathfrak{p}$ from $\mathfrak{U}$.

To determine $\mathfrak{u}^\uparrow$, we can use a grid that covers $\mathfrak{p}$, as shown in figure 7. We need not ensure that $\mathfrak{u}^\uparrow$ covers the region represented by $\mathfrak{u}$, but merely that the entire grid covers the region represented by $\mathfrak{p}$. Using such a grid will minimize the overlap between subpixel regions of $\mathfrak{p}$ and can consequently improve rendering times and rendering results.

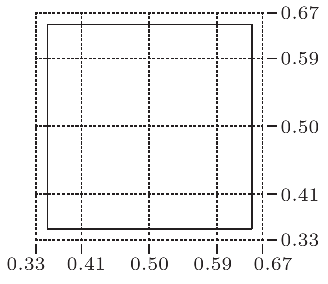
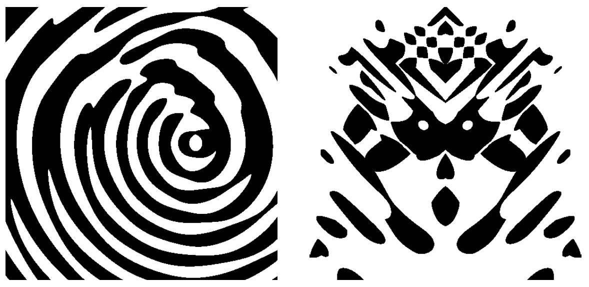


Figure 7: A 4×4 subpixel grid for the center pixel of figure 3, not to scale. With this grid, $\mathfrak{u}^\uparrow = \langle 0.33, 0.41 \rangle, \langle 0.33, 0.41 \rangle$ for $\mathfrak{u} = \left[1, 1\frac{1}{4}\right] \times \left[1, 1\frac{1}{4}\right]$ even though $[0.33, 0.41] \times [0.33, 0.41]$ does not cover $\left[\frac{1}{3}, \frac{5}{12}\right] \times \left[\frac{1}{3}, \frac{5}{12}\right]$, the region that $\mathfrak{u}$ represents.

For inequalities, algorithm 2 is quite capable of producing *finished* computed graphs, where no red pixels remain, as shown in figure 8; algorithm 2 will also finish the inequality of figure 5a.



(a) (b)

Figure 8: Finished 512×512 pixmap computed graphs, from algorithm 2, of (a) $(y-5)\cos\left(4\sqrt{(x-4)^2+y^2}\right) > x\sin\left(2\sqrt{x^2+y^2}\right)$ over $[-10, 10] \times [-10, 10]$ and (b) $\frac{1}{15}[6-y] + \frac{1}{6400000}\left[8x^2 + 4(y-3)^2\right]^3 + \cos\max(\beta\cos\alpha, \alpha\cos\beta) < \sin\min(\beta\sin\alpha, \alpha\sin\beta), \alpha = y - x, \beta = x + y$ over $[-5, 5] \times [0, 10]$.

10 Continuity Tracking; Algorithm 3

When applied to the equation of figure 5b, algorithm 2 will whiten all pixels that lack solutions, but will still leave red all pixels that contain solutions.

To extend algorithm 2 so that it can finish graphing one-dimensional sets, we will extend our interval system so that it provides *continuity tracking.* The interval $\langle \mathsf{val} \in \langle \mathtt{a}, \mathtt{b} \rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d} \rangle; \mathsf{cont} \in \langle \mathtt{e}, \mathtt{f} \rangle \rangle$ represents a quantity q

- that is continuous if $\langle \mathtt{e}, \mathtt{f} \rangle = \langle \mathrm{T}, \mathrm{T} \rangle$,
- that is not continuous if $\langle \mathtt{e}, \mathtt{f} \rangle = \langle \mathrm{F}, \mathrm{F} \rangle$, and
- that may or may not be continuous if $\langle \mathtt{e}, \mathtt{f} \rangle = \langle \mathrm{F}, \mathrm{T} \rangle$;

$\langle \mathtt{c}, \mathtt{d} \rangle$ states whether or not q is defined while $\langle \mathtt{a}, \mathtt{b} \rangle$ provides us with a bound on q when it is defined. Example psuedo-code for the addition and floor operators follow:

```
Add($\langle \mathsf{val} \in \langle \mathtt{a}, \mathtt{b} \rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d} \rangle; \mathsf{cont} \in \langle \mathtt{e}, \mathtt{f} \rangle \rangle$,
    $\langle \mathsf{val} \in \langle \mathtt{A}, \mathtt{B} \rangle; \mathsf{def} \in \langle \mathtt{C}, \mathtt{D} \rangle; \mathsf{cont} \in \langle \mathtt{E}, \mathtt{F} \rangle \rangle$)

1.  return $\langle \mathsf{val} \in \langle \mathtt{a} +^\downarrow \mathtt{A}, \mathtt{b} +^\uparrow \mathtt{B} \rangle; \mathsf{def} \in \langle \mathtt{c} \wedge \mathtt{C}, \mathtt{d} \wedge \mathtt{D} \rangle;$
            $\mathsf{cont} \in \langle \mathtt{e} \wedge \mathtt{E}, \mathrm{T} \rangle \rangle$
```

```
Floor($\langle \mathsf{val} \in \langle \mathtt{a}, \mathtt{b} \rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d} \rangle; \mathsf{cont} \in \langle \mathtt{e}, \mathtt{f} \rangle \rangle$)

1.  return $\langle \mathsf{val} \in \langle \lfloor \mathtt{a} \rfloor, \lfloor \mathtt{b} \rfloor \rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d} \rangle;$
            $\mathsf{cont} \in \langle \mathtt{c} \wedge (\lfloor \mathtt{a} \rfloor = \lfloor \mathtt{b} \rfloor), \mathtt{d} \rangle \rangle$
```

Algorithm 3 extends algorithm 2 by using continuity tracking to try to prove that solutions of r exist. Line 10 of RefineSubpixel is replaced with the following pseudo-code:

```
10. else if SubpixelSolutionExists($f$, $\mathfrak{p}^\downarrow$, $\mathfrak{u}^\uparrow$)
11.         then ShowSubpixelSolution($\mathfrak{p}$, $\mathfrak{U}_k$, $\mathfrak{U}_{k-1}$)
12.         else $\mathfrak{U}_{k-1} \leftarrow \mathfrak{U}_{k-1} \cup$ FourSubsquares($\mathfrak{u}$)
```

```
SubpixelSolutionExists($f$, $\mathfrak{p}^\downarrow$, $\mathfrak{u}^\uparrow$)

1.  if $(\mathfrak{p}^\downarrow \cap \mathfrak{u}^\uparrow = \emptyset)$ return F
2.  $[\ell, r] \times [b, t] \leftarrow \mathfrak{p}^\downarrow \cap \mathfrak{u}^\uparrow$
3.  if $(f(\mathfrak{u}^\uparrow)$.cont $\neq \langle \mathrm{T}, \mathrm{T} \rangle)$ return F
4.  return $(f(\langle \ell, \ell \rangle, \langle b, b \rangle) f(\langle r, r \rangle, \langle t, t \rangle) \leq 0) = \langle \mathrm{T}, \mathrm{T} \rangle$
```

The above pseudo-code assumes that r is given as $f = 0$. In general, when r is not given as a single equation, but is given as a logical combination of equations and inequalities, the above steps are carried out on the equations within r and any proof of a solution of an equation causes a $\langle \mathrm{T}, \mathrm{T} \rangle$ to be propagated up through the higher levels of r before any final decision can be made regarding $\mathtt{p}$. My implementations look for a sign change of f by evaluating f over several points in $[\ell, r] \times [b, t]$.

Snyder [Sny92a] discusses a "continuity operator" that seems to be similar to our continuity tracking: only a few details are given, but Snyder seems to use the information at a higher-level than we do. It is not clear if, or how, Snyder deals with domain tracking.

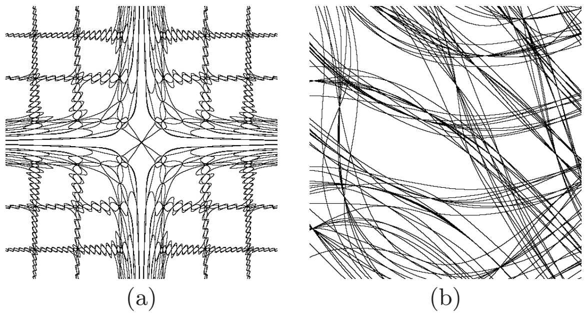


Figure 9: Finished computed graphs, from algorithm 3, of (a) $x \cos y \cos xy \pm y \cos x \cos xy \pm xy \cos x \cos y = 0$ over $[-10, 10] \times [-10, 10]$ and (b) $\sin((x \pm \sin y)(\sin x \pm y)) = \cos \sin((\sin x \pm \cos y)(\sin y \pm \cos x))$ over $[4, 6.5] \times [2, 4.5]$, both with 512×512 pixmaps.

11 Interval Sets; Algorithm 3.1

As shown in figure 10, algorithm 3 fails to whiten pixels that lie along discontinuities, even for the simple case of $y = \frac{1}{x}$. Algorithm 3 does not break discontinuous curves apart as the underlying interval arithmetic does not break discontinuous evaluations apart. With *interval sets*, our interval procedures can break discontinuous evaluations apart by returning a set of intervals whose union covers the true result, as shown by the following pseudo-code which replaces our previous Floor procedure:

```
Floor($\langle \mathsf{val} \in \langle \mathtt{a}, \mathtt{b} \rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d} \rangle; \mathsf{cont} \in \langle \mathtt{e}, \mathtt{f} \rangle \rangle$)
1. if ($\lfloor \mathtt{b} \rfloor - \lfloor \mathtt{a} \rfloor = 0$)
       return $\{\langle \mathsf{val} \in \langle \lfloor \mathtt{a} \rfloor, \lfloor \mathtt{b} \rfloor \rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d} \rangle; \mathsf{cont} \in \langle \mathtt{c}, \mathtt{d} \rangle \rangle\}$
2. if ($\lfloor \mathtt{b} \rfloor - \lfloor \mathtt{a} \rfloor = 1$)
       return $\{\langle \mathsf{val} \in \langle \lfloor \mathtt{a} \rfloor, \lfloor \mathtt{a} \rfloor \rangle; \mathsf{def} \in \langle \mathrm{F}, \mathtt{d} \rangle; \mathsf{cont} \in \langle \mathrm{F}, \mathtt{d} \rangle \rangle,$
               $\langle \mathsf{val} \in \langle \lfloor \mathtt{b} \rfloor, \lfloor \mathtt{b} \rfloor \rangle; \mathsf{def} \in \langle \mathrm{F}, \mathtt{d} \rangle; \mathsf{cont} \in \langle \mathrm{F}, \mathtt{d} \rangle \rangle\}$
3. return $\{\langle \mathsf{val} \in \langle \lfloor \mathtt{a} \rfloor, \lfloor \mathtt{b} \rfloor \rangle; \mathsf{def} \in \langle \mathtt{c}, \mathtt{d} \rangle; \mathsf{cont} \in \langle \mathrm{F}, \mathtt{d} \rangle \rangle\}$
```

To evaluate a mathematical operator with interval set arguments, our interval library can loop over all possible combinations of interval arguments and repeatedly invoke the appropriate interval evaluation routine.

Using this approach with binary operators, such as multiplication, can produce sets with many members since the product of an interval set with m elements and an interval set with n elements would be an interval set with mn elements. If an interval set contains too many elements, the set can be simplified by merging some of the interval elements together; after simplification, the set will use fewer memory but may also be less precise. Judicious simplification can minimize the loss of precision: for example, merging overlapping intervals with similar properties together will not affect precision.

I will clarify the role that domain tracking plays now that interval sets are available to our interval library. First, intervals with $\mathsf{def} \in \langle \mathrm{F}, \mathrm{F} \rangle$ can now be omitted; the empty set can be returned for quantities that are not well-defined. Second, an interval with $\mathsf{def} \in \langle \mathrm{T}, \mathrm{T} \rangle$ will only be returned if it is known that the quantity represented is well-defined *and* lies within the interval about to be returned. This approach is used for several reasons: it will generalize nicely when we later modify our underlying interval arithmetic; it allows multifunctions to be modelled directly; and it allows intervals to be considered individually rather than as part of a set — this improves modularity and simplifies the introduction of interval sets to an interval library. Other forms of property tracking, such as continuity tracking, are similarly applied to individual interval elements. If desired, property tracking can be applied to interval sets.

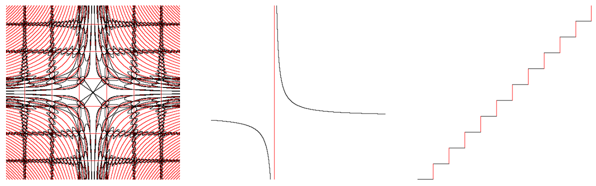


Figure 10: Computed graphs, from algorithm 3, of (a) $x \sec x \pm y \sec y \pm xy \sec xy = 0$ over $[-10, 10] \times [-10, 10]$, (b) $y = \frac{1}{x}$ over $[-4, 7] \times [-4, 7]$, and (c) $y = \lfloor x \rfloor$ over $[-4, 7] \times [-4, 7]$; all with 384×384 pixmaps. Compare figure 10a with figure 9a.

Algorithm 3.1 carries out the same procedure as algorithm 3, but uses interval sets when evaluating r. Given any of the graphing problems of figure 10, algorithm 3.1 produces a finished computed graph.

Other researchers have also used interval set approaches [Kah68, Han80, RR88]. For example, when evaluating rational polynomials, Kahan [Kah68] uses $\langle \mathtt{a}, \mathtt{b} \rangle$ with $\mathtt{b} < \mathtt{a}$ to represent $\{\langle -\infty, \mathtt{b} \rangle, \langle \mathtt{a}, +\infty \rangle\}$.

12 Branch Cut Tracking; Algorithm 3.2

By keeping track of the circumstances surrounding interval evaluations, our interval library can reduce the number of intervals kept in interval sets while simultaneously increasing the precision of computed bounds. Consider the following evaluation, which occurs while trying to decide one of the red pixels of figure 11a: (for clarity, property tracking has been omitted)

$$\begin{aligned}
&r(\langle 0.99, 1.0 \rangle, \langle 1.3, 1.4 \rangle) \\
\rightsquigarrow\ &\langle 1.3, 1.4 \rangle + \lfloor \langle 0.99, 1.0 \rangle \rfloor = \langle 0.33, 0.34 \rangle + \lfloor \langle 0.99, 1.0 \rangle \rfloor \\
\rightsquigarrow\ &\langle 1.3, 1.4 \rangle + \{\langle 0, 0 \rangle, \langle 1, 1 \rangle\} = \langle 0.33, 0.34 \rangle + \{\langle 0, 0 \rangle, \langle 1, 1 \rangle\} \\
\rightsquigarrow\ &\{\langle 1.3, 1.4 \rangle, \langle 2.3, 2.4 \rangle\} = \{\langle 0.33, 0.34 \rangle, \langle 1.3, 1.4 \rangle\} \\
\rightsquigarrow\ &\langle \mathrm{F}, \mathrm{T} \rangle.
\end{aligned}$$

The result of the evaluation is $\langle \mathrm{F}, \mathrm{T} \rangle$ as the routine testing for equality is unaware of the correlation between the two interval sets.

When an evaluation routine breaks a discontinuous evaluation apart into pieces, no information is kept that keeps track of which branch each piece belongs to. We will extend our interval library to keep track of which branch each interval belongs to by adding a partial function $\mathsf{branch} : \mathbb{Z} \to \mathbb{Z}$ to each interval. This function maps from

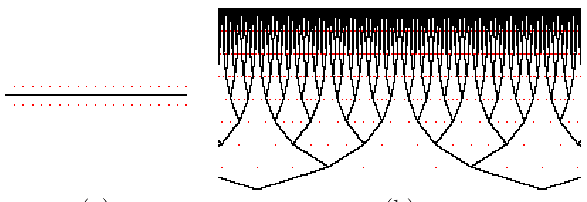


Figure 11: Computed graphs, from algorithm 3.1, of (a) $y + \lfloor x \rfloor = \frac{1}{3} + \lfloor x \rfloor$ over $[-10, 10] \times [-10, 10]$ with a 128×128 pixmap and (b) the bi-infinite binary tree $\sin\left(2^{\lfloor y \rfloor} x \pm \frac{\pi}{4}(y - \lfloor y \rfloor) - \frac{\pi}{2}\right) = 0$ over $[-8, 8] \times [-2, 6]$ with a 256×128 pixmap.

branch cut sites to chosen branches; branch cut sites are identified during formula preprocessing: each operator that can perform branch cuts that occurs more than once with the same arguments within the formula being evaluated is assigned a unique integer that identifies the branch cut site. This preprocessing occurs within the framework of common subexpression elimation. Pseudo-code for evaluating a floor operation with branch cut tracking follows:

```
Floor(⟨val ∈ ⟨a, b⟩; def ∈ ⟨c, d⟩; cont ∈ ⟨e, f⟩; branch = g⟩, site)

1. if (⌊b⌋ − ⌊a⌋ = 0)
       return {⟨val ∈ ⟨⌊a⌋, ⌊b⌋⟩; def ∈ ⟨c, d⟩; cont ∈ ⟨c, d⟩; branch = g⟩}
2. if (⌊b⌋ − ⌊a⌋ = 1)
       return {⟨val ∈ ⟨⌊a⌋, ⌊a⌋⟩; def ∈ ⟨F, d⟩;
                 cont ∈ ⟨F, d⟩; branch = g ∪ {site → 0}⟩,
               ⟨val ∈ ⟨⌊b⌋, ⌊b⌋⟩; def ∈ ⟨F, d⟩;
                 cont ∈ ⟨F, d⟩; branch = g ∪ {site → 1}⟩}
3. return {⟨val ∈ ⟨⌊a⌋, ⌊b⌋⟩; def ∈ ⟨c, d⟩; cont ∈ ⟨F, d⟩; branch = g⟩}
```

When evaluating a binary operation with interval sets, only intervals that can belong to the same branch are considered for evaluation, as the following pseudo-code shows: (the resulting intervals have branch $= g_i \cup G_j$)

```
Add({⟨val ∈ ⟨a, b⟩; def ∈ ⟨c, d⟩; cont ∈ ⟨e, f⟩; branch = g⟩_i : 1 ≤ i ≤ m},
    {⟨val ∈ ⟨A, B⟩; def ∈ ⟨C, D⟩; cont ∈ ⟨E, F⟩; branch = G⟩_j : 1 ≤ j ≤ n})

1. 𝔊 ← ∅
2. for i ← 1 to m
3.   for j ← 1 to n
4.     if (IsAFunction(g_i ∪ G_j))
5.       𝔊 ← 𝔊 ∪
6.           Add(⟨val ∈ ⟨a, b⟩; def ∈ ⟨c, d⟩; cont ∈ ⟨e, f⟩; branch = g⟩_i,
7.               ⟨val ∈ ⟨A, B⟩; def ∈ ⟨C, D⟩; cont ∈ ⟨E, F⟩; branch = G⟩_j)
8. return 𝔊
```

With my implementations, the branch function is represented using two bit-fields, cut and chosen; branch cut site i corresponds to the ith bit of each bit-field. cut remembers which cuts have been performed while chosen remembers which branch the interval belongs to. With this representation, the pseudo-code for IsAFunction is as follows:

```
IsAFunction(g, G)

1. m ← g.cut ∧ G.cut
2. return (g.chosen ∧ m) = (G.chosen ∧ m)
```

When evaluating r, leaf evaluation intervals, which correspond to constants and the variables x and y, have empty branch functions as they belong to all branches.

Algorithm 3.2 uses branch functions while evaluating r, but is otherwise the same as algorithm 3.1. Algorithm 3.2 finishes both graphs of figure 11; figure 12a shows a finished computed graph of the bi-infinite binary tree from figure 11b. Graphs based on enumerations, such as figure 12b, can be improved significantly by keeping tracking of branch cuts as discontinuous elements of r usually reoccur several times.

Figure 13 shows a computed graph based on enumerating all $j \times 17$ bi-level grids; a high-precision floating-point package must be used to finish this computed graph. The interval routines must also use the same branch cut sites for $\left\lfloor \frac{y}{17} \right\rfloor$ and $\mathrm{mod}\left(\lfloor y \rfloor, 17\right)$, which is possible as $\left\lfloor \frac{y}{17} \right\rfloor \equiv \left\lfloor \frac{\lfloor y \rfloor}{17} \right\rfloor$ and $\mathrm{mod}\left(\lfloor y \rfloor, 17\right) \equiv \lfloor y \rfloor - 17 \left\lfloor \frac{\lfloor y \rfloor}{17} \right\rfloor$. The formula of figure 13 can be modified to enumerate all possible $j \times k$ binary grids with both j and k varying.

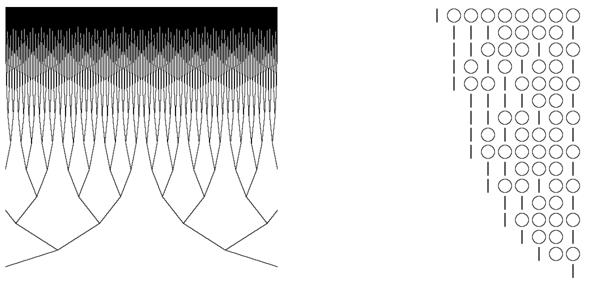


Figure 12: Computed graphs, from algorithm 3.2, of (a) the bi-infinite binary tree from figure 11 over $[-5.1, 5.1] \times [-2.1, 8.1]$ and (b) the integer squares in binary $r_b(\lfloor y \rfloor^2)$ over $[-15, 0] \times [1, 16]$, both with 512×512 pixmaps; $r_b(n) = \left[n \geq 2^{-\lceil x \rceil}\right] \wedge \left[\left(1 + 9^9 \left\lfloor \mathrm{mod}\left(n 2^{\lceil x \rceil}, 2\right)\right\rfloor\right) \left(x - \lfloor x \rfloor - \frac{1}{2}\right)^2 + \left(y - \lfloor y \rfloor - \frac{1}{2}\right)^2 = 0.15\right]$. The graph of $r_b(\lfloor y \rfloor) \wedge [y > 2] \wedge \left[\gcd\left(\lfloor y \rfloor, \left\lfloor \sqrt{2 \lfloor y \rfloor} - \frac{1}{2} \right\rfloor !\right) \leq 1\right]$ shows all prime numbers in binary.

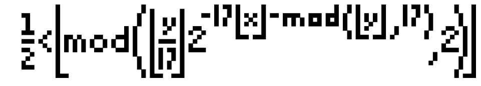
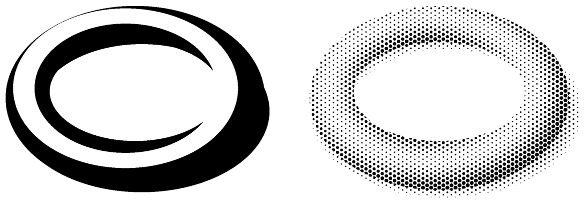


Figure 13: Finished computed graph, from algorithm 3.2, of $\frac{1}{2} < \left\lfloor \mathrm{mod}\left(\left\lfloor \frac{y}{17} \right\rfloor 2^{-17\lfloor x \rfloor - \mathrm{mod}(\lfloor y \rfloor, 17)}, 2\right) \right\rfloor$ with a 1696×272 pixmap over $[0, 106] \times [k, k+17]$, $k =$ 960 939 379 918 958 884 971 672 962 127 852 754 715 004 339 660 129 306 651 505 519 271 702 802 395 266 424 689 642 842 174 350 718 121 267 153 782 770 623 355 993 237 280 874 144 307 891 325 963 941 337 723 487 857 735 749 823 926 629 715 517 173 716 995 165 232 890 538 221 612 403 238 855 866 184 013 235 585 136 048 828 693 337 902 491 454 229 288 667 081 096 184 496 091 705 183 454 067 827 731 551 705 405 381 627 380 967 602 565 625 016 981 482 083 418 783 163 849 115 590 225 610 003 652 351 370 343 874 461 848 378 737 238 198 224 849 863 465 033 159 410 054 974 700 593 138 339 226 497 249 461 751 545 728 366 702 369 745 461 014 655 997 933 798 537 483 143 786 841 806 593 422 227 898 388 722 980 000 748 404 719 .

Affine arithmetic [CS93], which is another generalization of standard interval arithmetic, also uses common subexpression analysis to improve the bounds returned from interval evaluations. As with linear interval arithmetic [Tup96], affine arithmetic uses functions to bound quantities, but introduces a new dependent variable for each common subexpression.

13 Halftoning

Algorithm 3.2 can correctly graph a wide variety of formulae, as shown by figures 14 and 15.



(a) (b)

Figure 14: Computed graphs, from algorithm 3.2, of (a) the bi-level torus $2 > f_\circ(x,y)$ and (b) the hexagonally halftoned torus $h_6 > f_\circ(x,y)$, both over $[-41,43] \times [-42,42]$ with 1024×1024 pixmaps. For both formulae,

$$\begin{aligned} f_\circ(x,y) &= 1 + \left\{ \begin{array}{cc} \frac{3}{2}\sin\frac{1}{4}\sqrt{(x+3)^2+2(y-3)^2} & \text{if } d\leq 0 \\ 2\,\mathrm{Arctan}\left(\frac{1}{8}\sqrt{4(x-2)^2+10(y+4)^2}-9\right)^2 & \text{if } d>0 \end{array} \right\}, \\ d &= (x^2+2y^2-1600)(x^2+3(y-2)^2-700), \text{ and} \\ h_6 &= \cos 5x + \cos\tfrac{5}{2}\left(x-\sqrt{3}y\right) + \cos\tfrac{5}{2}\left(x+\sqrt{3}y\right). \end{aligned}$$

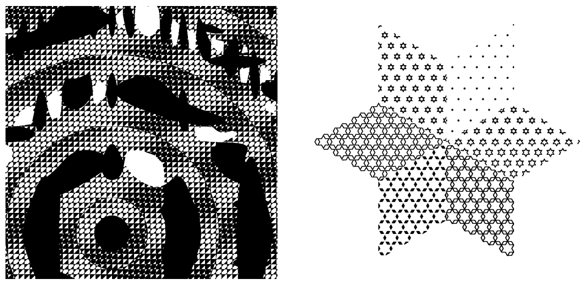
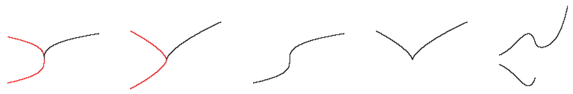


Figure 15: Computed graphs, from algorithm 3.2, of (a) $\tan[\sqrt{x^2+y^2}\,\mathrm{sgn}(\sin 12(x-y)\sin 13x \sin 14y)] < \max(\sin[x\cos y], \cos[y \sin x])$ over $[-4.7, 7.3] \times [-2, 10]$ and (b) the patterned star $\left[0.15 > \left|\mathrm{median}\left(\cos 8y, \cos 4\left(y-\sqrt{3}x\right), \cos 4\left(y+\sqrt{3}x\right)\right) - \cos\left\lfloor \frac{3}{\pi}p - 0.5\right\rfloor - 0.1\right|\right] \wedge \left[\mathrm{median}\left(|2x|, \left|x-\sqrt{3}y\right|, \left|x+\sqrt{3}y\right|\right) < 10\right], p = \mathrm{Arctan}(x,y)$; both with 1024×1024 pixmaps.

14 Exponentials; Algorithm 3.3

Still, a few formulae from introductory mathematics courses remain beyond the reach of our graphing algorithms; these remaining formulae involve exponentials; several are shown in figure 16.

Algorithm 3.2 will *not* finish graphing $y = x^{\frac{1}{3}}$ because the underlying interval arithmetic cannot decide if $x^{\langle 0.33, 0.34\rangle}$ is well-defined when $x < 0$. Clearly explaining all of the details involved with evaluating exponentials with interval arithemetic would unacceptably lengthen this paper and is a likely topic for a future presentation towards an audience focused on mathematical computation. It is sufficient, for many graphs, for our graphing algorithm to to analyse r symbolically before graphing and to add, when possible, a tag to each exponent: this tag would state the parity of the numerator and the denominator of the exponent when it is in lowest terms. This modification allows algorithm 3.3 to finish graphing $y = x^{\frac{1}{3}}$ as the interval exponentiation routine can determine that $x^{\langle 0.33, 0.34\rangle}$ is well-defined and negative when $x < 0$ if it knows that the exponent is a rational number with an odd numerator and denominator. Further preprocessing and tagging, based on noticing that $y = x^x$ is of the form $f(y) = g(x)^{g(x)}$ where g is a non-constant an-



(a) (b) (c) (d) (e)

Figure 16: Computed graphs, from algorithm 3.2, of (a) $y = x^{\frac{1}{3}}$ and (b) $y = x^{\frac{2}{3}}$ and finished computed graphs, from algorithm 3.3, of (c) $y = x^{\frac{1}{3}}$, (d) $y = x^{\frac{2}{3}}$, and (e) $y = x^x$; all over $[-2,3] \times [-2,3]$ with 128×128 pixmaps.

alytic function, allows algorithm 3.3 to manipulate $y = x^x$ into a form that it can use to generate finished computed graphs.

15 Optimization; Algorithm 3.4

I have not yet discussed the many ways in which our graphing algorithms can be optimized. Some optimization suggestions that will usually reduce the amount of memory and time required to graph a formula follow: (see [Sny92a] for additional hints)

1. Common subexpressions should be folded together and only evaluated at most once per evaluation of r. This is required for tracking branch cuts.

2. Constants in r should be computed once and then cached for future use; they should not be computed each time r is evaluated.

3. Evaluated subexpressions that depend only on x or y should be cached. As an example, consider figure 15a; while processing each $\mathfrak{U}_k$, $\sin 13x$ will be evaluated many times with the exact same interval for x.

4. Using a multigrid approach, as we have, for subpixel computations does not fully exploit our lack of interest in subpixel geometry. Recursively probing a pixel for solutions of r is generally much faster as we are interested in at most one solution of r.

5. A separate explicit graphing algorithm should be used when r can be expressed as either $y \diamond f(x)$ or $x \diamond f(y)$, where $\diamond$ is a comparison. More generally, any explicit disjuncts of r can be removed and handled by an explicit graphing algorithm while the remainder of r is handled by our implicit graphing algorithm.

6. Symbolic preprocessing should be done to simplify r; many of the examples used to disparage interval methods are based on symbolic reasoning. Appropriate symbolic preprocessing will nullify these examples. As a simple example, if $f - f$ occurs within r, and f is known to be well-defined, $f - f$ may be replaced with 0. Many algorithms have been developed to massage r into various well-behaved forms for certain classes of r. See [Arn83, Tau93] for when r is algebraic.

16 Comparisons with Other Work

Other researchers have presented algorithms for graphing formulae. The approach I present in this paper differs from these other approaches in the following ways:

1. I assign a clear meaning to the color used for each pixel. With this meaning, it is possible to prove that algorithms 1.1–3.4 are all correct and never produce incorrect graphs.

   Fateman [Fat92] argues for "honest" plotting that faithfully represents all significant features of a graph. This is desirable, but it is clearly impossible in general as our display can only present a limited amount of information; nevertheless, algorithms 1.1–3.4 can faithfully represent some features of a graph, such as local extrema, when the resolution of the graph is sufficient; algorithms 1.1–3.4 will never misrepresent any features of a graph, given our semantics behind computed graphs.

   Arnon [Arn83] argues for a "topologically reliable" display. Although this too, due to the limitations of our display device, is clearly impossible, even for algebraic curves, algorithms 1.1–3.4 will, for example, never show a computed graph that disconnects connected components of the graph being plotted.

   I have taken the liability of a limited-resolution display device and turned it into a valuable asset: its limitations provides a terminating criterion for our graphing algorithm. Approaches based on computing significant features or topological characteristics of a graph will be unable to handle graphs with infinite amounts of detail. Even smooth one-dimensional manifolds with no horizontal or vertical segments, the type of graphs that Snyder's "Implicit Curve Approximation Algorithm" is intended to address [Sny92b], can contain an infinite number of disjoint components and have an infinite number of local extrema, as exemplified by the graph of $\sin \frac{1}{x^2+y^2} = 0$. Algorithms 1.1–3.4 can, of course, be stymied by difficult formulae, but it is reassuring to know that, for every formula, there is a unique finished computed graph of that formula and that that finished computed graph can be produced by some reliable graphing algorithms, even if we have not yet implemented those algorithms.

2. Algorithms 1.1–3.4 correctly handle formulae that contain subexpressions that are undefined and/or discontinuous for some values of x and y.

   Approaches based on standard interval methods [Fat92, Sny92b, ABK95, HQvE00] will generally not be able to break discontinuous evaluations apart and will not break discontinuous curves apart, as discussed in section 11 and shown in figures 10 and 17a,b.

   Other approaches explicitly limit themselves to formulae which are well-defined, such as algebraic equations [Arn83, Tau93].

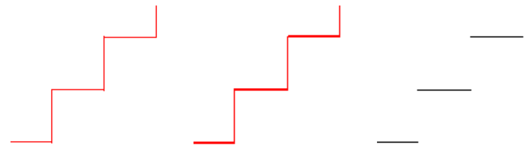


Figure 17: Computed graphs of $y = x - \operatorname{Arctan} \tan x$ over $[-4, 5] \times [-4, 5]$ from (a) Graphing Calculator 3.0.1, with a 145×145 pixmap; (b) IAsolver 0.1beta1, with a 128×128 pixmap; and (c) algorithm 3.4, with a 128×128 pixmap.

Table 1 summarizes how quickly graphing results are presented to the user, for two different algebraic equations, using several different graphing algorithms. The equations were chosen from [Tau94] as Taubin provided concrete timing results for the RecursivePaintZeros graphing algorithm he presented, along with the equations, pixmap dimensions, and graphing areas used to produce those results. I chose two equations of high degree; with Taubin's lower-degree examples, the differences between the methods are similar, but less pronounced. One notable example is the equation shown in his figure 13i[4], $x^2 + y^2 + y^3 = 0$, which contains a singularity: although algorithm 3.4 will generally not finish equations with singularities, it can finish this example as the singularity lies on a point with floating-point coordinates. As suggested in section 15, algorithm 3.4 could be improved by adding symbolic preprocessing.

Arnon [Arn83] presents some timing results as well, but the machine he uses, a VAX 11/780 minicomputer running UNIX™, is *quite* different than the Macintosh I am using to time algorithm 3.4; any comparisons are *tenuous*. A VAX 11/780 with a floating-point accelerator running UNIX™ is capable of around 0.11 Linpack MFlops/s [Don00]; a PowerMac G3/300 is capable of 70.7–77.1 Linpack MFlops/s [Met99, Mic00]. Arnon reports, for his example 3, a running time of around 8 minutes; algorithm 3.4, with a 512×512 pixmap representing $[-2.5, 2.5] \times [-2.5, 2.5]$, begins subpixel processing after 2.3 seconds and presents a finished computed graph after 37.8 seconds. The computed graph presented by algorithm 3.4 shows that the illustration given in Arnon's figure 3 misrepresents the geometry of the graph, although Arnon states that the illustrations given in his figures 1–3 were copies made by a draftsman working from the output of the algorithm he presents in his section 7. Arnon's figure 3 shows a graph with several long horizontal and vertical segments when there is more than enough resolution to show the gentle curves of the graph.

17 Conclusion and Future Work

Algorithm 3.4 correctly graphs the mathematical formulae encountered in introductory mathematics courses; when applied to a difficult formula that is beyond its capabilities, algorithm 3.4 clearly marks the pixels that it cannot decide. At no point does the algorithm use any approximations that may cause it to produce an incorrect graph.

Given that equation graphing, as formalized herein, is not computable, there is a never-ending stream of improvements that could be made to algorithm 3.4 that would allow it to produce finished graphs for an ever-larger set of formulae. Improvements in this direction can be made by introducing more features into the interval library underlying algorithm 3.4; with some thought, many symbolic techniques that could be applied to the formula prior to graphing can be adapted to work within our interval library: continuity tracking and domain tracking are both examples of this. Adapted techniques can often then be applied to a wider variety of situations and are available to other applications that are clients of the interval library.

The semantics we use for computed graphs are quite simple. Although this allows mathematics students, one of the primary users of this technology, to quickly gain a complete understanding of the information presented by computed

[4]The graph given in [Tau94] actually shows $y^2 + x^2 + x^3 = 0$; several of the equations and graphs in [Tau94] do not match up precisely.

Algorithm	First Update (s)	First Covering (s)	Graph Approx-imated (s)	Graph Finished (s)	Linpack (MFlop/s)
Algorithm 3.4	0	0	1.2	2.2	70.7–77.1
GCalc	0	1–2	1–2	∞	70.7–77.1
IAsolver	0	0–1	50	∞	28.6
RPZ$_2$	N/A	N/A	12.5	∞	15
RPZ$_{10}$	N/A	N/A	24.9	24.9*	15

(a)

Algorithm	First Update (s)	First Covering (s)	Graph Approx-imated (s)	Graph Finished (s)	Linpack (MFlop/s)
Algorithm 3.4	0	0	5.5	13.9	70.7–77.1
GCalc	0	6–7	6–7	∞	70.7–77.1
IAsolver	0	76	635	∞	28.6
RPZ$_2$	N/A	N/A	757.1	∞	15
RPZ$_{50}$	N/A	N/A	6,234.6	6,234.6*	15

(b)

Table 1: Timing results for graphing (a) $(2y - x \pm 1)(2x + y \pm 1)\prod_{j\in\{-1,0,1\}}\left((5x+2j)^2 + (5y+6j)^2 - 10\right) = 0$ over $[-5,5] \times [-5,5]$ and (b) $\prod_{(j,k)\in\{-2,-1,0,1,2\}^2}\left((x+j)^2 + (y+k)^2 - 0.4\right) = 0$ over $[-3,3] \times [-3,3]$, both with 512×512 pixmaps.

First Update gives the elapsed time, in seconds, until the algorithm first begins updating the display; algorithm 3.4, GCalc, and IAsolver all show graphing as it progresses. **First Covering** gives the elapsed time, in seconds, until the algorithm first covers every pixel with graphing information; a quick covering provides the user with an over-all view of the graph. **Graph Approximated** gives the elapsed time, in seconds, until the computed graph closely approximates the finished graph; with algorithm 3.4, this is when subpixel computations start. **Graph Finished** gives the elapsed time, in seconds, until the graph is finished. All elapsed times are measured from when graphing begins. **Linpack** gives the reported Linpack benchmark results, in megaflops per second, for each platform used [Don00, DWM00, Mic00, Met99].

Algorithm 3.4 was timed on a PowerMac G3/300. **GCalc** is Graphing Calculator [ABK95] 3.0.1, timed on a PowerMac G3/300. **IAsolver** is IAsolver [HQvE00] 0.1beta1, timed on a PowerMac G3/300 with MRJ 2.1. **RPZ$_k$** is the RecursivePaintZeros procecure [Tau94] used with an order k distance-approximation, timed on an IBM RS/6000 model 930; the timing results for RPZ$_k$ are from [Tau94].

*RPZ does not take into account the limited precision of floating-point.

graphs, a clear direction for future research is to enrich the semantics of computed graphs so that topological information and other significant features of the graph are also presented.

Another clear direction for future work is to develop algorithms that can interactively display three-dimensional formulae with the same degree of mathematical rigor.

Acknowledgements I would like to thank John Hughes, for his practical suggestions on restructing the presentation of the material; the paper reviewers, for their comments; and my supervisor Eugene Fiume, for his support and advice.

References

[ABK95] Ron Avitzur, Olaf Bachmann, and Norbert Kajler. From Honest to Intelligent Plotting. In A. H. M. Levelt, editor, *Proc. of the International Symp. on Symbolic and Algebraic Computation (ISSAC'95), Montreal, Canada*, pages 32 – 41. ACM Press, 1995.

[Arn83] Dennis S. Arnon. Topologically Reliable Display of Algebraic Curves. *Computer Graphics (SIGGRAPH 83 Conference Proceedings)*, 17(3):219–227, July 1983.

[CS93] J. Comba and J. Stolfi. Affine Arithmetic and its Applications to Computer Graphics. In *Anais do VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens (SIBGRAPI '93)*, pages 9–18, 1993.

[Don00] Jack J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Technical Report CS-89-85, University of Tennessee, 2000.

[DWM00] Jack Dongarra, Reed Wade, and Paul McMahan. Linpack Benchmark — Java Version. http : //www.netlib.org/benchmark/linpackjava, 2000.

[Fat92] R. Fateman. Honest Plotting, Global Extrema and Interval Arithmetic. In P. S. Wang, editor, *Proc. of the International Symp. on Symbolic and Algebraic Computation (ISSAC'92), Berkeley, USA*, pages 216 – 223. ACM Press, 1992.

[Han80] Eldon Hansen. Global Optimization Using Interval Analysis — The Multi-Dimensional Case. *Numerische Mathematik*, 34(3):247–270, 1980.

[HQvE00] Timothy J. Hickey, Zhe Qiu, and Maarten H. van Emden. Interval Constraint Plotting for Interactive Visual Exploration of Implicitly Defined Relations. *Reliable Computing*, 6(1):81–92, 2000.

[IEE85] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic.* IEEE, New York, NY, USA, August 1985. Revised 1990. A preliminary draft was published in the January 1980 issue of IEEE Computer, together with several companion articles. Also standardized as *IEC 60559 (1989-01) Binary floating-point arithmetic for microprocessor systems.*

[Kah68] W. M. Kahan. A More Complete Interval Arithmetic. Lecture notes prepared for a summer course at the University of Michigan, June 17–21, 1968.

[Met99] Metrowerks. Macintosh Linpack Benchmark. http://www.metrowerks.com/benchmarks/desktop/mac_linpack.html, 1999.

[Mic00] Tom Michiels. http://www.cs.kuleuven.ac.be/∼tomm/bench.html, 2000.

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

[Moo79] R. E. Moore. *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, 1979.

[Ped] Pedagoguery Software Inc. GrafEq$^{\mathrm{TM}}$. http://www.peda.com/grafeq.

[RR88] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* Ellis Horwood Ltd., Chichester, 1988.

[Sny92a] John M. Snyder. *Generative Modeling for Computer Graphics and CAD: Symbolic Shape Design Using Interval Analysis.* Academic Press, San Diego, 1992.

[Sny92b] John M. Snyder. Interval Analysis for Computer Graphics. *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, 26(2):121–130, July 1992.

[Tau93] Gabriel Taubin. An Accurate Algorithm for Rasterizing Algebraic Curves. In *Second Symposium on Solid Modeling.* ACM SIGGRAPH and IEEE Computer Society, May 1993.

[Tau94] Gabriel Taubin. Distance Approximations for Rasterizing Implicit Curves. *ACM Transactions on Graphics*, 13(1):3–42, January 1994.

[Tup96] Jeffrey Allen Tupper. Graphing Equations with Generalized Interval Arithmetic. Master's thesis, University of Toronto, 1996.