FPAvisual: A Tool for Visualizing the Effects of Floating-Point Finite-Precision Arithmetic

Yi Gu, Nilufer Onder, Ching-Kuang Shene, Chaoli Wang
Michigan Technological University

Abstract

Many students in science and engineering do not realize how program correctness may be impacted when floating-point finite-precision arithmetic is used. In this paper, we present FPAvisual, a visualization tool that helps instructors teach the reasons for the inaccuracies caused by floating-point arithmetic (FPA), their impact and significance in programs, and the techniques to improve the accuracy. FPAvisual contains four components, namely, Roots, Pentagon, Associative Law, and Sine Function. Roots shows that the solution for a quadratic equation will be incorrect when two numbers that need to be subtracted are very close in magnitude or when one is much larger than the other. The program presents possible solutions to these subtraction problems. Pentagon demonstrates that accumulation of errors emanating from finite precision in geometric computation may result in large positional errors. Associative Law demonstrates how algebraically equivalent formulas computed by changing the order of operations can yield different results. Sine Function shows that results vary when the same infinite series for sine is used but computed in different ways. These four components allow the users to set up the parameters of the specific problem represented, trace the results step by step, see when the differences in results start to occur, and visualize how errors accumulate. They help students understand the ubiquity of issues with FPA, realize the significance of FPA in a multitude of contexts, and compare the methods to minimize the negative effects of FPA. FPAvisual has been classroom tested and evaluated by computer science students. We report our findings in this paper.

1. Introduction

Due to binary representation and finite precision, the floating-point data type in computer programs is not equivalent to real numbers in mathematics[11, 15]. Major differences occur when real numbers are converted into binary and are truncated to contain only the maximum number of digits allowed by the specific representation and precision. Programs that contain floating-point variables may produce incorrect results, or worse, may contain assumptions that will produce unpredictable results when the computing environment or the program parameters change. However, many students in science and engineering do not realize how program correctness may be impacted when floating-point arithmetic is used. Therefore, it is important to include floating-point concepts in computing curricula.

The ACM curricula recommendations include data representation or computer arithmetic among the core topics for all computing-related disciplines, namely, computer science, computer engineering, information systems, and software engineering. However, within already intense curricula there is little time left, if any, to extend the coverage of floating-point representation to show students how to effectively work with floating-point numbers. Based on this observation,

we developed FPAvisual, a visualization tool that helps instructors teach the reasons for the anomalies caused by floating-point arithmetic (FPA), their impact and significance in programs, and the techniques to increase the accuracy of FPA.

The main guiding principle in the design of FPAvisual is ease and effectiveness of use for both students and instructors. The specific design objectives are threefold. The first objective is to create a flexible learning environment. It is desirable to deliver fundamental concepts through visualization without requiring extensive work or time commitment from the user. It is further desirable to provide opportunities for an in-depth exploration of the concepts. The second objective is to clearly depict not only what anomalies occur, but how and why they occur. The final objective is to show students how to deal with inaccuracies caused by FPA using known solutions.

The remainder of this paper is organized as follows. The next section contains a brief survey of the research in teaching FPA and background information about the basic reasons behind FPA anomalies. We then present FPAvisual, the software we developed. Afterwards, we discuss the results of the formative assessment as conducted in Spring 2013 and Fall 2013 classes. We conclude with a summary and a description of future work.

2. Related Work

Dealing with unexpected inaccuracies caused by floating-point arithmetic is an important topic in many fields of engineering and computer science, most notably, numerical analysis[8,10,12] and computer systems[7]. It is important to note that the relevance of FPA extends beyond scientific computing and computer systems. In a recent paper, Polhill[14] et al. gave several examples of inaccuracies or misleading results in agent-based modeling of systems in a multitude of domains such as sociology, environmental and ecosystem management, resource management, and artificial stock market.

While there are several initiatives in teaching FPA to students, we give examples from approaches similar in spirit to FPAvisual. In his 1991 paper, Scott[15] gave a rich set of examples regarding FPA as well as recommendations to teach them to students of computing. Garzón[6] et al. described an initiative for teaching computer arithmetic. Within this initiative, practical exercises have been designed in the areas of integer representation, integer arithmetic, floating-point representation, floating-point arithmetic, the anomalies introduced by floating-point representation, and the IEEE 754 floating-point standard. Horvath and Verhoeff[11] discussed the issues in incorporating tasks involving FPA into the Annual International Informatics Competition, which is a programming competition for talented high-school students. The discussion includes how to prepare test cases that will bring out the degree of numerical insight possessed by competitors, how to design tasks with varying degrees of FPA challenges, how to prepare students for FPA awareness, and how to deal with platform differences. Fernández[4] et al. developed a course and a web-based tool on FPA. Their tool is different from FPAvisual in two aspects. First, it emphasizes the floating-point representation while FPAvisual was designed to show the appearance and significance of FPA errors. Second their tool presents examples in textual format while FPAvisual uses animated and interactive graphical displays. Allison[1] reported on a recent junior-level course that was prepared to increase awareness of FPA issues

among computer science students.

In the remainder of this section, we present brief background information on FPA issues shown in FPAvisual.

2.1 Rounding and the Failure of the Associative Law

Computers represent floating-point numbers using a finite number of bits, called the precision. When a number contains more digits than allowed by the hardware, it is rounded. The rounded number is an approximation of the original number. Consider an addition operation in base 10, where the precision is 3 significant digits, i.e., 3 digits without leading zeroes can be stored.

$123 + 2.46$	$=$	125.46	The original numbers have at most three significant digits. The exact result is 125.46.	
	$=$	125	The result is rounded to 3 significant digits.	(1)

In the above example, the low order digits of the second number were lost due to rounding. Consider another example:

$123 + 0.46$	$=$	123.46	The original numbers have at most three significant digits. The exact result is 123.46.	
	$=$	123	The result is rounded to 3 significant digits.	(2)

This time, the addition operation did not have an effect due to rounding, all digits of the second number were lost.

Due to rounding, changing the computation order of a mathematical expression may lead to different results[2]. For example, assuming 3 significant digits with base 10, the result of $0.121 \times 0.345 \times 4.21$ can be calculated using two different orders:

$(0.121 \times 0.345) \times 4.21$	$=$	$(0.041745) \times 4.21$	The exact result is 0.041745	
	$=$	0.0417×4.21	0.041745 is rounded to 3 digits	
	$=$	0.175557	The exact result is 0.175557	
	$=$	0.176	0.175557 is rounded to 3 digits	(3)
$0.121 \times (0.345 \times 4.21)$	$=$	0.121×1.45245	The exact result is 1.45245	
	$=$	0.121×1.45	1.45245 is rounded to 3 digits	
	$=$	0.17545	The exact result is 0.17545	
	$=$	0.175	0.17545 is rounded to 3 digits	(4)

By changing the computation order, Equations 3 and 4 produce two different results. In other words, the associative law fails.

2.2 Cancellation

Cancellation occurs when subtracting two nearly equivalent numbers and leads to the loss of significant digits. For example, when solving a quadratic equation, cancellation may occur in computing $b^2 - 4ac$. Assuming the number of significant digits is 3 with base 10, consider $a = 1$, $b = 1.23$ and $c = 0.374$, then

b^2	$=$	1.23×1.23	
	$=$	1.5129	The exact result is 1.5129

$= 1.51$ 1.5129 is rounded to 3 digits (5)

$4ac = 4 \times 1 \times 0.374$

$= 1.496$ The exact result is 1.496

$= 1.50$ 1.496 is rounded to 3 digits (6)

Therefore,

$b^2 - 4ac = 1.51 - 1.50$ Computation using rounding

$= 0.01$ (7)

$b^2 - 4ac = 1.5129 - 1.496$ Actual computation

$= 0.0169$ (8)

The original numbers are accurate in all 3 digits while the result is only accurate in the first non-zero digit. This computation loses almost all significant digits. Besides that, the actual result is 0.0169 which is quite different from the computed result 0.01.

2.3 Computing Iterative Summations

Errors due to rounding, cancellation, or overflow may occur when calculating the sum of an array of floating-point values. Two widely known solutions to avoid these errors are the positive-negative algorithm[9] and Kahan[12] summation algorithm. The positive-negative algorithm (Algorithm 1) reduces the number of subtractions thus reducing the possibility of cancellation errors. Kahan summation algorithm (Algorithm 2) uses a separate variable to store the error and utilizes it in the next iteration to improve the accuracy.

Algorithm 1 Positive-Negative Summation Algorithm

$S = 0$ // S is the sum of all values
$S_p = 0$ // S_p is the sum of all positive values
$S_n = 0$ // S_n is the sum of all negative values
for $i = 0$ to *num*−1 **do** // there are *num* values to be added
    if $array[i] \geq 0$ **then** // if the *i*th value is nonnegative
        $S_p = S_p + array[i]$
    else $term_i < 0$ // if the *i*th value is negative
        $S_n = S_n + array[i]$
    end if
end for
$S = S_p + S_n$

Algorithm 2 Kahan Summation Algorithm

$S = 0$ // S is the sum of all values
$c = 0$ // A variable to store the lost low-order bits
for $i = 0$ to *num* −1 **do**
    $y = array[i] - c$
    $t = S + y$ // If *S* is big and *y* is small, low-order digits of *y* are lost
    $c = (t - S) - y$ // *c* recovers the low-order digits of *y*
    $S = t$
end for

3. FPAvisual Software

FPAvisual was developed to provide engaging visualizations that show the inaccuracies caused by FPA, their significant influence on programs, and the techniques to increase the accuracy. It has Windows and Linux versions. FPAvisual consists of four components: Roots, Pentagon, Associative Law, and Sine Function. Roots solves a quadratic equation with four approaches, shows the incorrect results due to the cancellation, and provides solutions to increase the accuracy. Pentagon demonstrates that rounding may cause inaccurate calculations of geometric shapes. Associative Law reveals the fact that FPA can cause the associative law to fail. Sine Function shows the differences between twelve approaches that compute a sine value using the Taylor series. It helps students understand overflow and underflow errors and learn how to reduce them. In addition, students also learn how to sum an array of floating-point values accurately.

To provide students an easy start, FPAvisual contains several pre-constructed examples in each component. The animations in Pentagon, Associative Law and Sine Function components were designed to attract students' attention and to provide striking displays of when and how errors occur. The colors were selected to easily track different computations. Students can save the results of computations into *.txt or *.csv files for further studying with external tools or software. They can choose their preferred colors in each component for differentiation and tracking.

In the following subsections, we describe in detail each individual component and its implementation.

3.1 Roots Component

Solving a quadratic equation is a common problem in many fields of science and engineering. However, rounding and cancellation may affect the results[5]. Roots demonstrates their effect and provides possible solutions using four approaches. The first and the third solutions are the "naive" floating-point implementations of the standard formula for the two roots in single precision and double precision, respectively. The second is the cancellation solution with single-precision floating-point format that improves the result in one case that is described below. The last one is the solution using a high-precision floating-point format, which is treated as a ground-truth solution. We describe these approaches in the following.

Given a quadratic equation:

$$ax^2 + bx + c = 0 \quad (9)$$

The formula for the two roots is:

$$x_{1,2} = (-b \pm \sqrt{b^2 - 4ac})/2a \quad (10)$$

When the above formula is computed using floating-point variables, incorrect roots can be calculated when cancellation causes an error in the numerator's value. This can happen in two ways. First, if $b^2 \gg 4ac$, due to rounding, then $\sqrt{b^2 - 4ac}$ is approximately $|b|$. Then, cancellation occurs in $-b + \sqrt{b^2 - 4ac}$ if $b \geq 0$, and in $b - \sqrt{b^2 - 4ac}$ if $b < 0$. Therefore, one of the roots is 0. We can avoid this particular cancellation easily by avoiding subtraction which is

referred to as the cancellation solution:
If $b \geq 0$,

$$x_1 = (-b - \sqrt{b^2 - 4ac})/2a \quad (11)$$

$$x_2 = c/(ax_1) \quad (12)$$

If $b < 0$,

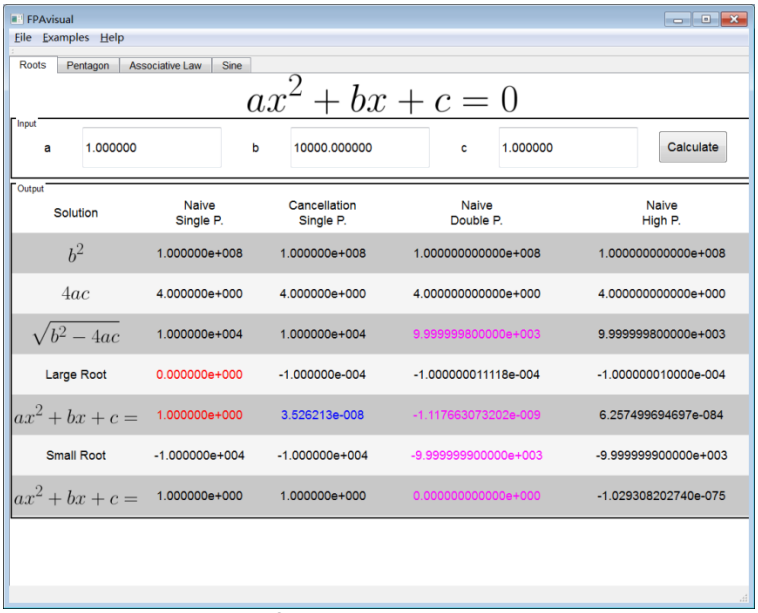
$$x_1 = (-b + \sqrt{b^2 - 4ac})/2a \quad (13)$$

$$x_2 = c/(ax_1) \quad (14)$$

Second, cancellation occurs when $b^2 \approx 4ac$. There is no easy way to overcome this situation. It might be addressed using double-precision arithmetic.

To give students an easy start, Roots provides three examples: a "regular" example where cancellation does not occur, an example where $b^2 \gg 4ac$, and an example where $b^2 \approx 4ac$. By studying these three examples, students can observe the issues caused by FPA and learn about ways to address them. Different colors are used to highlight the differences between the first three solutions so that students can easily notice where the differences are.

Figure 1 is a screenshot of Roots, where $a = 1$, $b = 10000$ and $c = 1$, which is the case where $b^2 \gg 4ac$. With the visual aid of colors, the differences appearing in $\sqrt{b^2 - 4ac}$ are highlighted, where the naive solution with double-precision is more accurate than the first two. Between the first two, the cancellation solution generates a more accurate result. After calculating the large root, the naive approach with single floating-point format gets a zero while the second approach gets a result close to the ground-truth result.



Solution	Naive Single P.	Cancellation Single P.	Naive Double P.	Naive High P.
b^2	1.000000e+008	1.000000e+008	1.000000000000e+008	1.000000000000e+008
$4ac$	4.000000e+000	4.000000e+000	4.000000000000e+000	4.000000000000e+000
$\sqrt{b^2 - 4ac}$	1.000000e+004	1.000000e+004	9.999999800000e+003	9.999999800000e+003
Large Root	0.000000e+000	-1.000000e-004	-1.000000011118e-004	-1.000000010000e-004
$ax^2 + bx + c =$	1.000000e+000	3.526213e-008	-1.117663073202e-009	6.257499694697e-084
Small Root	-1.000000e+004	-1.000000e+004	-9.999999900000e+003	-9.999999900000e+003
$ax^2 + bx + c =$	1.000000e+000	1.000000e+000	0.000000000000e+000	-1.029308202740e-075

Figure 1: An example of Roots, where $b^2 \gg 4ac$.

3.2 Pentagon Component

The Pentagon component illustrates the inaccuracies caused by rounding. We chose the pentagon problem in geometric computation because it provides an easy way to understand problems in an

interactive environment. The pentagon problem involves defining a pentagon with its five vertices and iteratively using "in" and "out" operations to get back to the original pentagon. The "in" operation computes a smaller pentagon by computing the intersection points of the lines joining the vertices of the original pentagon. This operation can be repeated on the new pentagon to form an even smaller one. The "out" extends the opposite sides of the pentagon to get the intersection points and form a larger pentagon. In Figure 2, the result of the "in" operation on the red pentagon is the blue one while the result of "out" operation on the blue pentagon is the red one. At each iteration, the vertices used for computation are calculated from the previous iteration. In theory, as long as the numbers of "in"s and "out"s are equal, the new pentagon should be exactly the same as the initial one. However, because of rounding, this may not always be true. The "in" and "out" operations require computing the intersection point of two lines. If the computation is inaccurate, the error may propagate to the next iteration, which will make the resulting pentagon dramatically different from the original one. This difference allows students to see the significance of accumulated floating-point errors and the difficulty of calculating the intersection point of two almost parallel lines.

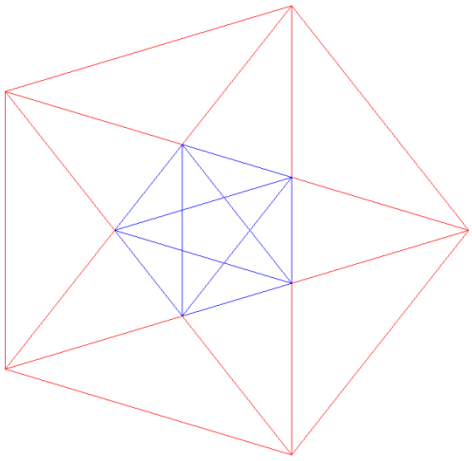


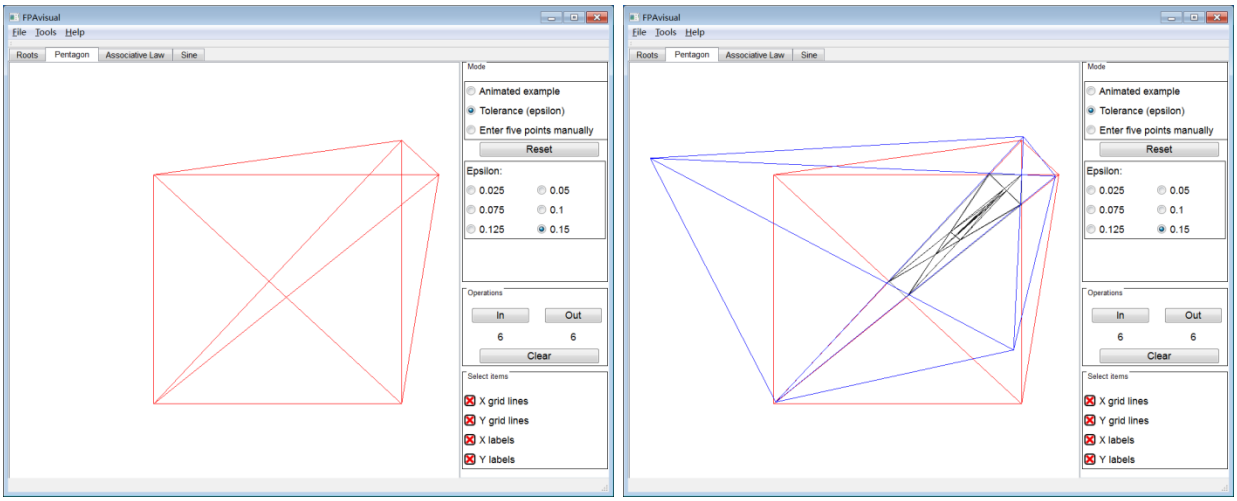
Figure 2: The "in" operation on the red pentagon will generate the blue one and the "out" operation on the blue pentagon will generate the red one.

Hoffmann[10], Dobkin and Silver[3], and Masotti[13] studied the problem of calculating the intersection of two lines accurately. Computing the intersection of two almost parallel lines is not easy. This is because the intersection (x, y) is very far away. Hence, x and y can be very large. As a result, errors due to rounding cannot be avoided. In other words, this system is ill-conditioned. Since a line is represented by a linear equation of two variables, the intersection point is found by solving a linear system of two equations. To see this problem, consider a pentagons with vertices (0, 0), (0, 1), (1, 0), (1, $1+\varepsilon$) and ($1+\varepsilon$, 1), where ε is a small value. Because ε is small, the edge defined by (0, 0) and (1, 0) and the edge defined by (0, 1) and (1, $1+\varepsilon$) are nearly parallel. Finding the intersection point can be represented by the following linear system:

$$\begin{bmatrix} \varepsilon & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad (15)$$

The solution is $(-1/\varepsilon, 0)$. Therefore, if ε is small, $|-1/\varepsilon|$ is large and rounding can make $-1/\varepsilon$ inaccurate, or even overflow or underflow. Since the value will be used in the next iteration, by accumulating the errors, the result can get even worse. Figure 3 (a) shows the initial pentagon in

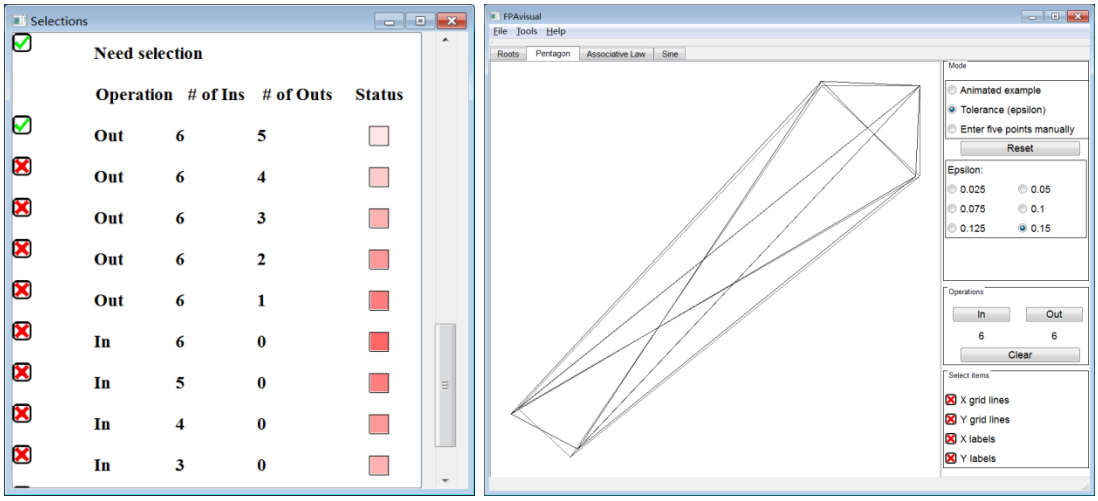
red. After six “in”s followed by six “out” operations, the result is the pentagon in blue shown in Figure 4 (b).



(a) (b)

Figure 3: Starting from the pentagon shown in (a), six “in”s and six “out”s are applied. (b) shows all the pentagons generated. An obvious difference between the blue final pentagon and the red initial pentagon can be observed.

The program allows the comparison of any intermediate pentagons to show the detail of differences. Figure 4 (a) is the selection window that allows students to select any pentagon for comparison. It lists all the pentagons along with their status, such as their corresponding operations and the numbers of “in” and “out” operations. The rectangles under “Status” illustrate the balances of “in” and “out” operations. Figure 4 (b) shows the result of comparing two pentagons: the first is the resulting pentagon after one “in” operation on the initial pentagon and the second is the one leading to the final pentagon with one “out” operation. The small difference between these two pentagons leads to the large difference shown in Figure 3 (b).



(a) (b)

Figure 4: (a) is the selection window that allows students to select any number of pentagons for comparison. (b) is the corresponding result of comparing two pentagons.

3.3 Associative Law Component

Associative Law demonstrates a situation when changing the order of computations of algebraically equivalent formulas may yield very different results. FPAvisual uses one example from Colonna[2]'s paper to visualize this problem:
Given a value R (the growing rate) and an initial population X_0, the following formula iteratively calculates the population of a group at time n:

$$X_n = (\mathrm{R}+1) \times X_{n-1} - R \times X_{n-1} \times X_{n-1} \quad (16)$$

If R is larger than 2.57, the failure of the associative and distributive laws can dramatically influence the accuracy of the results.

Due to the associative and distributive laws, Equation 16 is equivalent to the following five:

$$X_n = (R+1) \times X_{n-1} - R \times (X_{n-1} \times X_{n-1}) \quad (17)$$
$$X_n = (R+1) \times X_{n-1} - (R \times X_{n-1}) \times X_{n-1} \quad (18)$$
$$X_n = ((R+1) - R \times X_{n-1}) \times X_{n-1} \quad (19)$$
$$X_n = R \times X_{n-1} + (1 - R \times X_{n-1}) \times X_{n-1} \quad (20)$$
$$X_n = X_{n-1} + R \times (X_{n-1} - X_{n-1} \times X_{n-1}) \quad (21)$$

Since these five formulas are mathematically equivalent, they should yield the same set of X_n values. However, it is not the case when using FPA. Table 1 lists the values at several iterations, where R is 3 and X_0 is 0.5. Figure 5 is the corresponding plot. In this plot, the top row on the canvas shows the iteration number. There are small differences in the first 21 iterations. However, when the errors accumulate, these approaches start to have slight differences and begin to diverge dramatically at iteration 24.

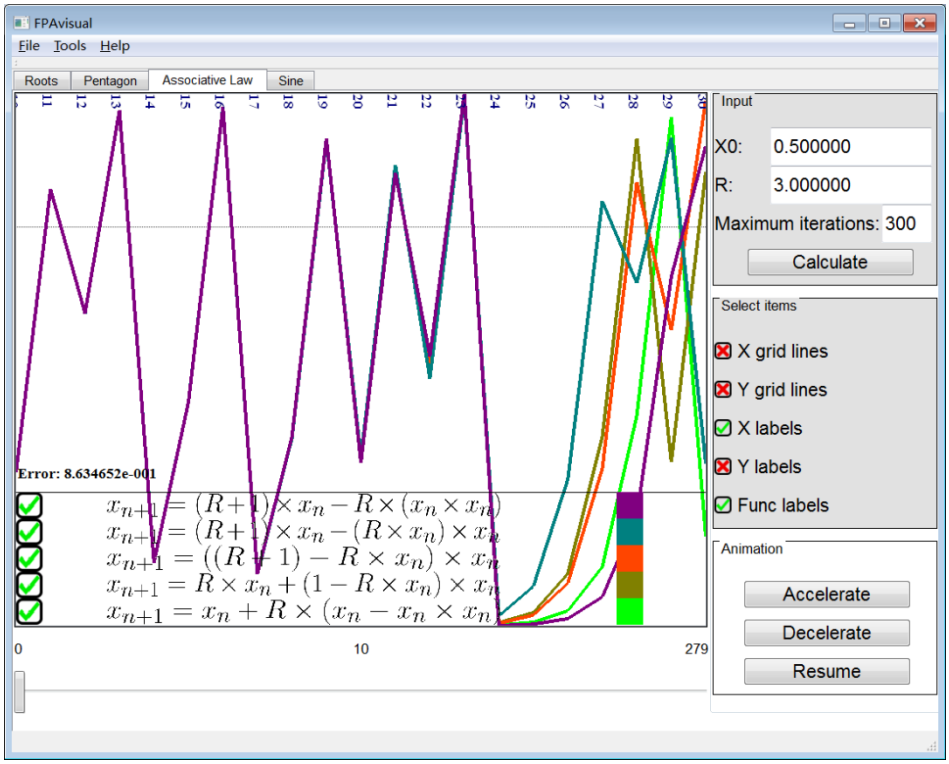
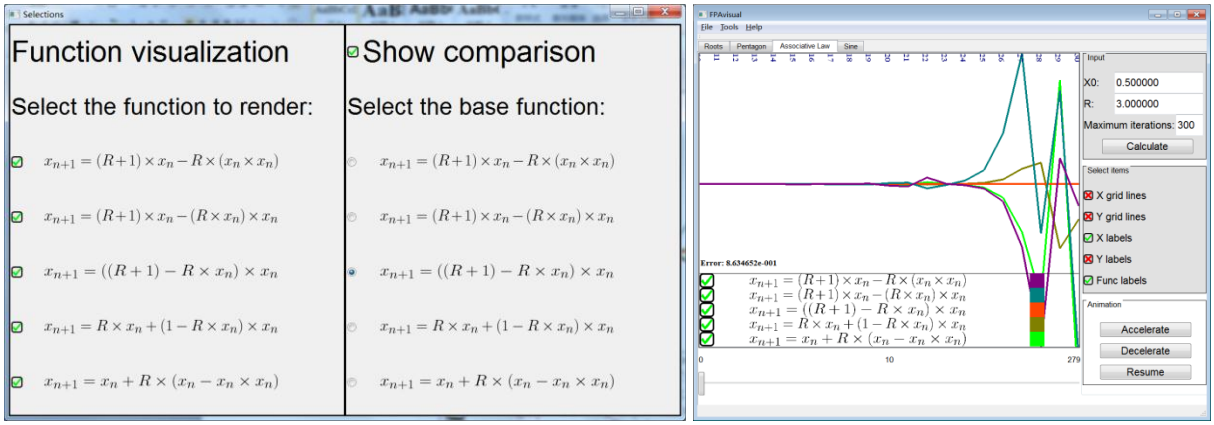


Figure 5: Associative Law. The differences become apparent at iteration 24.

Iteration	0	8	20	24	40	60	80	100
Equation 17	0.50000	0.85315	0.40927	0.00116	1.28944	1.33298	0.14510	0.29079
Equation 18	0.50000	0.85314	0.42221	0.02530	1.10737	0.01124	0.13318	1.29752
Equation 19	0.50000	0.85314	0.41709	0.00688	1.30471	1.33288	0.04450	1.21970
Equation 20	0.50000	0.85314	0.41770	0.00847	0.52484	0.86257	1.32618	1.31700
Equation 21	0.50000	0.85314	0.41482	0.00240	0.29500	0.71097	1.05449	0.04980

Table 1: X_ns at eight selected iterations.



(a) (b)

Figure 6: (a) is the selection window, in which students are allowed to choose any approaches to view and compare the approaches based on one of them. (b) shows the comparison result based on the third approach shown in orange.

FPAvisual provides a mechanism for students to see the relative error with respect to a base approach as shown in Figure 6 (a). Figure 6 (b) shows the comparison result based on the third approach. The five approaches have small differences between iterations 14 and 20, slight differences between iterations 20 and 24, and dramatic differences after iteration 24.

3.4 Sine Function Component

In this component, FPAvisual uses an infinite series approximation to the sine function to reveal some critical concepts in iterative summations of floating-point numbers. The sine function $\sin(x)$ can be evaluated using the infinite Taylor series as follows:

$$\begin{aligned} \sin(x) &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \\ &= \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}, \end{aligned} \quad (22)$$

where x is in radian. If x is very large or small, x^{2n+1} may overflow or underflow when n is large. One way to overcome this problem is to reduce the user input x (in degrees) to the $[0,90)$ interval using the following trigonometric identities:

(1) Since $\sin(x) = -\sin(-x)$, if $x < 0$, we just use $-\sin(|x|)$.

(2) Since $\sin(x)$ has a period of 360, we can reduce x to $[0,360)$ by letting $x = x\%360$.

(3) Since $\sin(x + 180) = -\sin(x)$, if $x \geq 180$, we may reduce x to $[0,180)$ by letting $x = x -$

180 and changing the sign of the computed result.
⑷ Since $\sin(180 - x) = \sin(x)$, if x is in $[90,180)$, we may reduce x to $[0,90)$ by letting $x = 180 - x$.
While the range of x can be reduced further, FPAvisual uses $[0,90)$ because it serves our purpose well.

There is another place where overflow or underflow may occur. If $\sin(x)$ is computed in a naive way by directly evaluating each term $x^{2n+1}/[(2n+1)!]$, it is very likely that $(2n+1)!$ will quickly cause integer overflow. In the implementation, we utilized a floating-point value to store it. A safer approach is to update a term from the previous one:

$$\frac{x^{2n+1}}{(2n+1)!} = \frac{x^{2n-1}}{(2n-1)} \times \frac{x^2}{(2n)(2n+1)} \tag{23}$$

The third potential error occurs in the summation of terms with alternating sign values due to cancellation. To solve this problem, FPAvisual utilizes the positive-negative and Kahan summation algorithms discussed in Section 2.3.

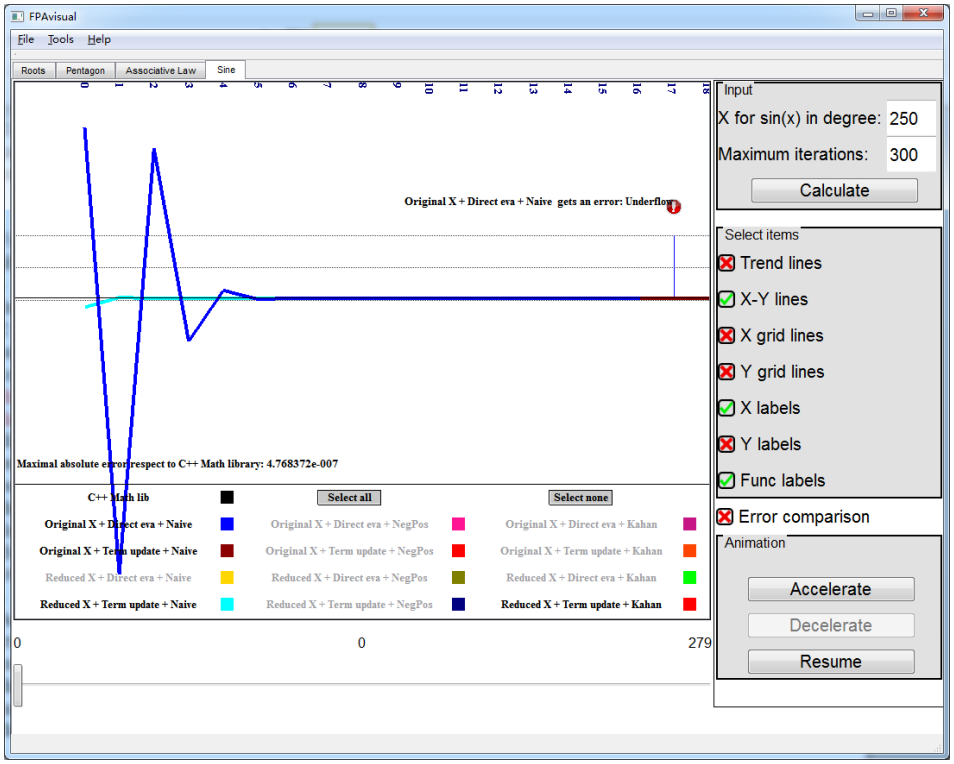


Figure 7: Four ways of calculating a sine value of the Taylor series and the way using the sin() library are selected for comparison.

Including the direct way in each aspect, we have 2 ways of using *x* (i.e., original *x* and reduced *x*), 2 ways of calculating the terms (i.e., direct evaluation and term update) and 3 ways for summation (i.e., direct, positive-negative and Kahan), therefore we have $2 \times 2 \times 3 = 12$ ways. For comparison, FPAvisual adds the result of the sin(*x*) library function. The 12 ways are shown in thick lines and the sin() library function is shown in thin line for differentiation. In addition, the

dashed lines are added to show the convergences of the 12 ways. Since visualizing 13 different approaches at the same time will be very cluttered and confusing, FPAvisual begins with only five ways. The first is the value calculated by the sin() library function which is used as the base for comparison. The second utilizes the original x, directly calculates the terms, and sums the terms in the usual way. The third updates the terms to show the benefits of term update. The fourth reduces x to $[0,90)$. The last one uses Kahan summation algorithm. Figure 7 compares the five ways. The error sign indicates that directly calculating the term causes underflow at iteration 16. All these approaches converge after iteration 7.

As shown in Figure 8, students can zoom in to see the details more clearly. However, in this example, the first approach using sin() library (in black) and the second (in blue) overlap with each other while the fourth using reduced x (in cyan) is below the first one and the fifth using the Kahan summation algorithm (in red) is even below the fourth. Table 2 lists the sine values using the five approaches and an online calculator (keisan.casio.com/calculator). The fifth way is the most accurate among the five.

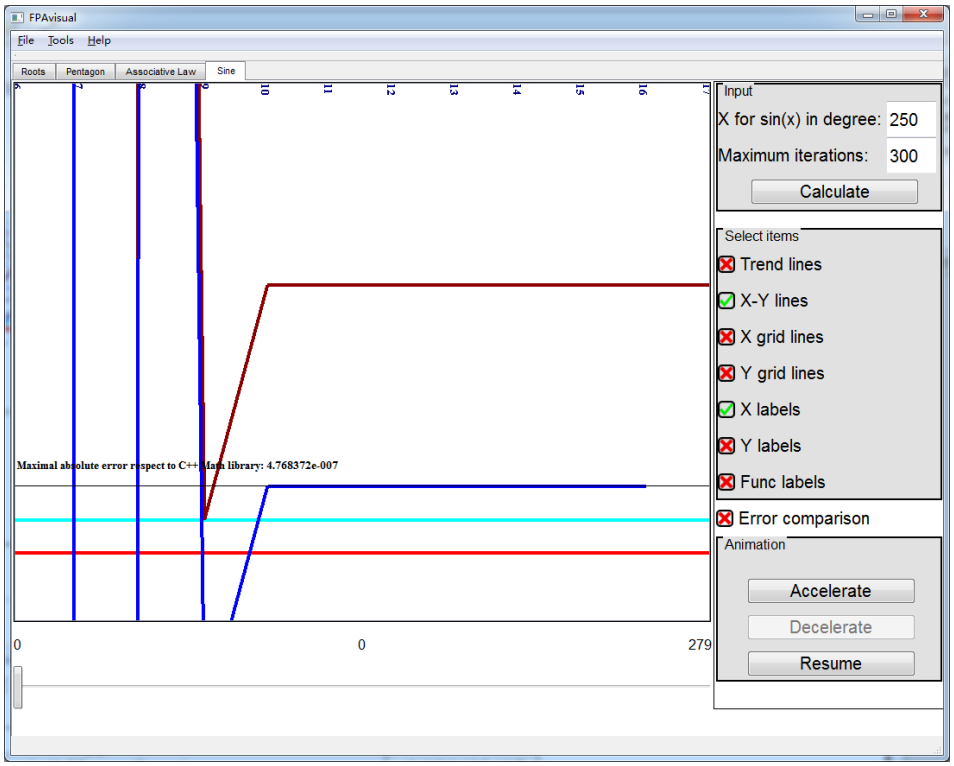


Figure 8: The result of zooming in to the iterations between 6 and 17.

Approach			Sine value
sin() library function			-0.939692497
Original x	Direct term evaluation	Naive summation	-0.939692497
Original x	Term update	Naive summation	-0.939692020
Reduced x	Term update	Naive summation	-0.939692557
Reduced x	Term update	Kahan summation	-0.939692616
Online high-precision calculator			-0.939692595

Table 2: The sine values calculated by the five ways and an online high-precision calculator.

4. Evaluation

The first version of FPAvisual was developed in Spring 2013 and contained the Roots, Associative Law, and Pentagon components. The Sine function component was added in Fall 2013. FPAvisual was tested and evaluated by the students of the Formal Models of Computation class. This is a required sophomore to junior level class which studies the hierarchy of automata and languages. The class prepares students for the Programming Language course by showing automata as tools. The class also studies model equivalency formally, e.g., regular expressions and finite automata are equivalent models for regular languages. At the end of the semester, one week consisting of three class hours was reserved to FPA topics to illustrate why floating-point variables are not a model for real numbers in mathematics. The FPAvisual software was demonstrated in the classroom. All the students were assigned a paper-pencil homework on FPA. In addition, they were asked to run the software and fill out an online survey. The survey was administered through the class management system.

The first survey given in Spring 2013 focused on the user interface and was taken by 19 students in a class of 30. Based on student input, the user interface was improved, features to save raw data were developed, and pre-constructed examples were added. To aid in the design of the next survey, we conducted an ethnographic study with volunteer graduate students who had taken the Formal Models class earlier. These students had not worked with FPAvisual before because it was not available at the time they took the class. After a brief introduction, the students were asked to use and explore the software while thinking aloud. The observers frequently reminded them to utter what they are doing and why they are doing it. These sessions helped us quickly uncover bugs and understand how users prefer to navigate in FPAvisual. Based on this information we updated the software and developed the Fall 2013 survey, which focused on the usefulness, usability, and the ease-of-use of FPAvisual. This survey was taken by 45 students in a class of 50. We collected 44 forms that included answers to all the survey questions.

		μ	σ
Q1	The “Roots” component helped me understand the effects of floating-point errors.	3.9	0.7
Q2	The “Roots” component helped me understand how to compute the roots of a quadratic equation more accurately.	3.6	0.8
Q3	The “Pentagon” component helped me understand the effects of floating-point errors.	3.9	1.1
Q4	The “Pentagon” component helped me understand that calculating the intersection points of two almost parallel lines can lead to noticeable errors.	3.9	0.9
Q5	The “Associative Law” component helped me understand how executing floating-point operations in different orders affects the computed results.	4.2	0.7
Q6	The “Associative Law” component helped me understand that there are no general techniques to detect and correct the errors coming from the failure of the associative law.	4.2	0.7
Q7	The “Sine” component helped me understand the effects of floating-point errors.	3.5	1.0
Q8	The “Sine” component helped me compare the effects of reducing X to [0,90), using the term update method, and using Kahan summation algorithm.	3.3	1.1
Q9	FPAvisual was a useful complement to the material presented in class.	3.8	0.6

Table 3: Questions evaluating the usefulness of FPAvisual.

The FPAvisual survey has two major components, 26 survey questions (Tables 3 to 5) and 9 write-in comments (Table 6). The first nine questions (Q1-Q9 in Table 3) evaluate the usefulness of FPAvisual, the next eleven questions (Q10-Q20 in Table 4) investigate its usability and the

last six questions (Q21-Q26 in Table 5) evaluate how easy it is to use FPAvisual. All the survey questions have the same set of choices on a 5-point Likert-scale:
1:strongly disagree, 2:disagree, 3:neutral, 4:agree, and 5:strongly agree.

		μ	σ
Q10	The example inputs provided in the "Roots" component helped me to see what kind of input values cause noticeable floating-point errors.	3.8	0.8
Q11	The animated examples in the "Pentagon" component helped me compare the results of in-out operations for differently shaped pentagons.	3.9	0.9
Q12	In the "Roots" component, seeing the results of computations in different colors helped me notice the differences between the approaches.	4.0	0.9
Q13	Being able to select pentagons for comparison was useful for me to see the accumulated floating-point errors.	3.8	0.9
Q14	The animations in the "Associative Law" component were useful for me to gain an impression of the effect of floating-point errors.	4.1	0.7
Q15	The animations in the "Sine" component helped me track the trend of different approaches.	3.7	0.9
Q16	The color encoding in the "Associative Law" component was useful for me to track the trend of the five computations.	4.3	0.7
Q17	Overall, I am satisfied with the color encoding.	4.2	0.6
Q18	The "error" icon in the "Sine" component helped me easily notice and identify the errors that occur during the computations.	3.7	0.9
Q19	The functionality that allows saving data into *.txt or *.csv files was useful for me to see the exact results of computations.	3.5	0.7
Q20	The freedom of manual input was useful to select inputs that cause noticeable floating-point errors.	4	0.7

Table 4: Questions evaluating the usability of FPAvisual.

		μ	σ
Q21	FPAvisual provided me adequate control.	3.9	0.7
Q22	FPAvisual is easy to use.	3.6	0.8
Q23	I was able to operate FPAvisual with minimal memory load, i.e., without having to remember a lot of commands or menu items.	4.1	0.6
Q24	FPAvisual required fewest steps to accomplish my tasks.	3.9	0.7
Q25	FPAvisual gave a way to review and return to previous contexts, e.g., seeing previous iterations or results.	3.6	0.7
Q26	Overall, I am satisfied with the support information (messages, documentation).	3.5	0.9

Table 5: Questions evaluating FPAvisual's ease of use.

Q27	Did the "Roots" component enable you to observe any new information pertaining to floating-point arithmetic? Please explain.
Q28	Did the "Pentagon" component enable you to observe any new information pertaining to floating-point arithmetic? Please explain.
Q29	Did the "Associative Law" component enable you to observe any new information pertaining to floating-point arithmetic? Please explain.
Q30	Did the "Sine" component enable you to observe any new information pertaining to floating-point arithmetic? Please explain.
Q31	Which components of FPAvisual offer the most interesting visualizations? Please explain.
Q32	Please list two or three of the most positive aspects of FPAvisual.
Q33	Please list two or three of the most negative aspects of FPAvisual.
Q34	Did you have problems when downloading, installing, or running FPAvisual? Please explain.
Q35	Do you have any suggestions for improvements or additional features? Please explain.

Table 6: Write-in Questions.

4.1 General Discussion

Tables 3 to 5 show the mean (μ) and standard deviation (σ) of each question. For usefulness questions, the highest scores (4.2) were given to Q5 and Q6, indicating that Associative Law helped students understand that different orders will affect the computation results and there are no general techniques to detect and correct the errors. Except for Q8, the other questions were rated in the range of 3.5 to 4.0, still above the neutral rating (3.0). The lowest rating (3.3) of Q8 indicates that Sine Function slightly helped them compare the different approaches to calculate the sine value using the Taylor series.

Of the usability questions, the highest score (4.3) was given to Q16, indicating that the animation of Associative Law helped students get an impression of the errors generated by FPA. During the class demonstration of the software, the students reacted with a loud "Wow" when the lines for different orderings of the formula started diverging without any pattern. Q12, Q14, Q17 and Q20 all received scores larger than 4.0, illustrating that the color encoding, animation, and freedom of manual input were helpful. The lowest rating (3.5) was given to Q19, suggesting that saving data files slightly helped students see the values. This is because there were no follow-up assignments requiring the use of the raw values. The results show that our software was able to assist students in understanding FPA better.

For ease-of-use questions, the ratings were in [3.5,4.1] with the standard deviation in [0.6,0.9], indicating that FPAvisual was easy to use. The highest rating (4.1) was given to Q23, showing that students did not need to remember a lot of commands or menu items to operate FPAvisual.

4.2 Student Comments

The set of nine write-in questions is designed to allow students to write comments for further improvement. We mainly focus on the following issues: which component has the most interesting visualization, what are the most positive or negative aspects.

The comments showed that Pentagon and Associative Law were the most visually interesting components because they were able to show the differences clearly. For example,
"*I realized that the errors introduced by rounding errors in the associative law were unpredictable. By watching the drastically diverged simulation emphasized the point.*"
"*I didn't even know there were problems with the associative law before this.*"
"*The pentagon function really shows how far floating point calculations skewed. The associative law also shows the progressive decay of accuracy*."
"I *liked the pentagon visualization the most, since it directly relates to the drawing of images in a program.*"
"*Pentagon shows one of the impacts on everyday calculations that you might experience. Associative Law shows how unpredictable error can make results.*"
The feedbacks of the positive aspects of FPAvisual covered almost all of the main functions. The visualization got comments such as:
"*The visualizations were extremely helpful*."
"*Good for being able to see things happen as opposed to just looking at math.*"

There were comments related to the animation. For example,
"*Easy to see errors and easy to track changes.*"
There were comments related to the manual input. For example,
"*Being able to input my own data was useful to emphasize the material.*"
Students also liked the color encoding and interface. For example,
"*The color schemes are very nice.*"
"*The interface is easy to use.*"
Students also mentioned that the saving function was helpful. For example,
"*The fact that you can save data to a file so you can look at it is also useful.*"

There were a few students who indicated that they would not benefit from FPAvisual as a study aid. This is not unexpected because not every individual is a visual learner. In addition, the color encoding was not helpful to color-blind students.

Besides the comments related to the positive aspects, FPAvisual also received suggests for improvement. Some students would like to see what kinds of error occurred and in which computation the errors occurred. For example,
"*I have no idea where the rounding errors came in. I just know that errors did happen somewhere.*"
Some students complained about the visual clutter, for example,
"*The visuals are a little clutter, but that can be overlooked due to its usefulness.*"
One student suggested adding more specific hints about the manual input and operations, such as what would happen if a button was clicked and what may be a good input range to play with. Students also mentioned that FPAvisual should support MacOS and a web version.

5. Conclusions

This paper presents a visualization tool FPAvisual for teaching and learning the concepts of floating-point arithmetic. With this tool, instructors are able to present the effects of different types of floating-point errors and the methods to reduce them. It also complements the lectures by helping students see the magnitude of the inaccuracy. The evaluation results suggested that FPAvisual is a useful complement to class teaching.

While nearly all students indicated that they understood the significance of the influence, we are currently looking at ways to show the details of computations leading to errors in the final results. Based on the student comments, the most needed extensions are visualizing what the errors are and where they occur, making the Sine Function component more understandable by distinguishing between the 12 approaches, adding detailed explanation text for the components, and developing a MacOS version.

Our future work includes expanding the type and number of the examples in the program and conducting a summative assessment of the software. The FPAvisual tools, evaluation forms, and user guides will be released online for public access.

Acknowledgements

This work was supported in part by the U.S. National Science Foundation through grants IIS-1017935, DUE-1105047, CNS-1229297, and IIS-1319363. We would like to thank all the students who participated in the user evaluation.

References

[1] C. Allison. Where did all my decimals go? *Journal of Computing Sciences in Colleges*, 21(3): 4-59, 2006.

[2] J.-F. Colonna. Kepler, von Neumann and God (More rounding-off error visualizations). *The Visual Computer*, 12(7): 346-349, 1996.

[3] D. Dobkin and D. Silver. Applied computational geometry: Towards robust solutions of basic problems. *Journal of Computer and System Sciences*, 40(1): 70-87, 1990.

[4] J.-J. Fernández, I. Garcia, and E. M. Garzón. Educational issues on number representation and arithmetic in computers: An undergraduate laboratory. *IEEE Transactions on Education*, 46(4): 477-485, 2003.

[5] G. E. Forsythe. How do you solve a quadratic equation? Technical Report AD0639052, Stanford University, 1966.

[6] E. M. Garzón, I. Garcí a, and J.-J. Fernández. An approach to teaching computer arithmetic. *In Proceedings of International Conference on High Performance Computing for Computational Science*, pages 269-283, 2003.

[7] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1): 5-48, 1991.

[8] N. J. Higham. Accuracy and stability of numerical algorithms. *Society for Industrial and Applied Mathematics (SIAM)*, Philadelphia, PA, second edition, 2002.

[9] N. J. Higham. The accuracy of floating point summation. *SIAM Journal on Scientific Computing*, 14(4): 783-799, 1993.

[10] C. M. Hoffmann. The problems of accuracy and robustness in geometric computation. *IEEE Computer*, 22(3): 31-39, 1989.

[11] G. Horváth and T. Verhoeff. Numerical difficulties in pre-university informatics education and competitions. *Informatics in Education*, 2(1): 21-38, 2003.

[12] W. Kahan. Further remarks on reducing truncation errors. *Communications of the ACM*, 8(1): 40-42, 1965.

[13] G. Masotti. Floating-point numbers with error estimates. *Computer-Aided Design*, 25(9): 524-538, 1993.

[14] J. G. Polhill, L. R. Izquierdo, and N. M. Gotts. What every agent-based modeler should know about floating point arithmetic. *Environmental Modelling and Software*, 21(3): 283-309, 2006.

[15] T. J. Scott. Mathematics and computer science at odds over real numbers. In *Proceedings of ACM SIGCSE Technical Symposium on Computer Science Education*, pages 130-139, 1991.