A Rational Rotation Method for Robust Geometric Algorithms

John Canny * **Bruce Donald** † **Eugene K. Ressler**†

1 Introduction

Algorithms in computational geometry often use the real-RAM model of computation. This model assumes that exact real numbers can be stored in memory and retreived in constant time, and that field operations (+, −, ∗, /) and certain other operations, square root, sine, and cosine for instance, are exact, and can be applied in constant time.

These assumptions are often difficult to discharge at implementation time. Even well-understood algorithms, like line-sweep for polygon union [PS85], present much trouble. Why? Such algorithms obtain good combinatorial complexity bounds by exploiting *geometric orders* on the input. These relations are often implicit, so an algorithm can only probe them pointwise with a collection of predicate functions answering, e.g., "Is point p left of edge e?". In implementations, the reply must depend on arithmetic with finitely represented numbers. The trouble originates here.

*Computer Science Division, University of California at Berkeley, Berkeley, California 94702

†Computer Science Department, Cornell University, Ithaca, NY 14853. This paper describes research done in the Robotics and Vision Laboratory at Cornell University. Support for our research is provided in part by the National Science Foundation (NSF) under grants No. IRI-8802390, IRI-9000532 and by a Presidential Young Investigator award to Bruce Donald, and in part by the Air Force Office of Sponsored Research (AFOSR), the Mathematical Sciences Institute, and AT&T Bell Laboratories. Initial discussions of this topic began at the Saratoga Workshop on the Integration of Numerical and Symbolic Computing Methods, Saratoga, NY (1990), which was funded by the NSF and the AFOSR.

8th Annual Computational Geometry, 6/92, Berlin, Germany
©1992 ACM 89791-518-6/92/0006/0251 ..... $1.50

In *limited precision* paradigms, numbers are restricted to some maximum size by rounding. Time for a single arithmetic operation is thus limited to a constant, and the real-RAM tends to model performance well. Unfortunately, a predicate employing limited precision is likely to give the wrong answer when important digits are rounded away, whereupon a geometric order is misrepresented to the overlying algorithm, and it fails. This is not a "mere engineering difficulty." Failures arise frequently in practice, and they cannot be fixed by using tolerances *ad hoc* in numerical comparisons, which is sufficient in some kinds of programs.

One recourse is to manage roundoff in clever ways. Generally speaking, we have predicates say "I don't know," when neither "yes" nor "no" is certain to be correct, then restructure the overlying algorithm to deal with this third reply. Some recent variations on this robust approach to limited precision are in [MN90, AFW88, HHK88, SI89].

The alternate approach, which concerns this paper, is to use *rational arithmetic* of arbitrary precision, storing all numbers exactly. Clearly, the real-RAM can be a very poor model here if numbers become long during computation. The *bit-complexity* model, which considers the cost of computing each bit (base 2 numeral) of the number representation, is more appropriate. A natural goal is then to show that an algorithm with real-RAM complexity $O(f(n))$ has bit-complexity $O(k \cdot f(n))$, where k is the maximum number of bits in any input number. This says that there is nothing asymptotically less efficient in using precise arithmetic than in rounding. Many algorithms, line-sweep for instance, have this property when carefully specified. Hence, on the face of it, rational arithmetic seems to solve our problem.

However, many applications from scientific computation and artificial intelligence necessarily mix such well-behaved algorithms with operations that destroy the exactness of the rational representation. One such

operation is rotation, which is the primary concern of this paper. For example, suppose we have two sets of polygons, A and B that model rigid objects in a 2D world. Suppose we *rotate* A by θ radians (in the plane), to obtain $A(\theta)$; i.e., for each vertex (x, y) of A we compute a new one (x', y') by the familiar transformation:

$$\begin{aligned} x' &= x \cdot \cos\theta - y \cdot \sin\theta \\ y' &= x \cdot \sin\theta + y \cdot \cos\theta \end{aligned} \quad (1)$$

This models a physical rotation of the object modeled by A, but here a problem arises. For almost all θ, one of θ, $\sin\theta$, or $\cos\theta$ is irrational (in fact, transcendental, by the Gelfand-Schneider theorem). Furthermore, for almost all θ, one of θ, $\sin\pi\theta$, or $\cos\pi\theta$ is irrational. Similarly, for almost all θ, one of $\sin\theta$ or $\cos\theta$ is irrational. The last is obvious, because almost all real numbers are irrational. Of course, arbitrary irrationals cannot be represented in computing machines. We will have to approximate somewhere, but where?

We might simply approximate the irrational sines and cosines in equations (1) by rationals within a certain tolerance, selectively introducing limited precision. The resulting transformation is typically not simply a rotation, but, instead, a rotation and scaling. In our example, instead of $A(\theta)$ we obtain $(1 + \delta_s) \cdot A(\theta + \delta_\theta)$, where δ_θ and δ_s are small constants. This has two distinguishable effects. First, the rotated versions of A are not rotated exactly the desired amount. We regard this as a good kind of approximation because it still models a nearby configuration of A and B in the world. Second, the rotated polygons of A have changed size with respect to those of B. This is a bad approximation because it is inconsistent with A and B as models of rigid objects.

Hence the thrust of this paper is to find rational rotation coefficients for some angle close to the one desired that have small representations and do not introduce scaling (i.e., we want $\delta_s = 0$). We call these *pure rotations* and give efficient algorithms to find them. More precisely, for given angle θ and tolerance ϵ_θ, our final algorithm returns a rational S with the following properties:

1. $\delta_\theta = |\sin^{-1} S - \theta| < \epsilon_\theta$
2. The corresponding cosine, $\sqrt{1 - S^2}$, is also rational.
3. S has at most one more bit than the shortest rational satisfying 1. and 2.

Shortness of sines is vital for applications, because, roughly speaking, the length of a rotated coordinate is the sum of its unrotated length and the length of the longest rotation coefficient. Moreover, storage for numbers may easily comprise the bulk of implementation storage requirements, and per-number-bit computations may easily be a performance bottleneck.[1]

To ease further discussion, then, let us adopt some conventions. The *length* of a rational is the magnitude of its denominator. Rational t is *shorter* (resp. *longer*) than rational u if the denominator of t has smaller (resp. greater) magnitude than that of u. The *number of bits* of a rational is the number of digits in the base-2 representation of its denominator with no leading zeros. Without loss of generality, we restrict discussion in this paper to $0 \leq \theta \leq \pi/4$; thus sines are positive and sign bits are of no concern.

Excepting the exhaustive search described in section 4, our algorithms are iterative and terminate in $O(n)$ iterations where $n = \log(1/\epsilon_\theta)$. Each iteration requires $O(M(n))$ time where $M(n)$ is the time to multiply two integers of n bits. With Schönhage-Strassen multiplication, this yields an overall complexity of $O(n^2 \log n \log\log n)$. It is practical to implement the algorithms entirely with rational arithmetic and thus achieve arbitrary precision. However, we concentrate on simple codes that use double precision floating point arithmetic to compute error terms. These perform well for $\epsilon_\theta \geq 10^{-10}$ and thus may be regarded as constant time operations. In practice, they run in a few milliseconds.

A secondary result of this paper concerns the variant of Euclid's algorithm often employed to approximate arbitrary numbers by some rational within a given tolerance. An example is the `rationalize` function of Common Lisp implementations (where the tolerance is machine floating point precision). We show this algorithm does not always return the shortest possible rational and give a fixup so it does.

1.1 Application: configuration space obstacles

The primary advantage of pure rotations is that they are distance-preserving (i.e., they are isometries). Moreover, they are trivially invertible, hence the inverse has the same bit-complexity. To illustrate why the former is important (the latter is covered in section 9), we now discuss a fundamental algorithm in robot motion planning where distance-preserving rotations are essential. Suppose we wish to compute the configuration space obstacle $CO_A(B)$ for a moving polygon A due to a stationary polygon B (see [LoP83]). $CO_A(B)$ can be characterized using the

[1] At least this holds in several of our Lisp implementations of robotics algorithms.

Minkowski sum $B \ominus A = \{b - a \mid a \in A, b \in B\}$. [LoP83] provides a linear time (optimal) algorithm for the case where A and B are convex, and this algorithm has been generalized by Guibas to the non-convex case. In practice, a non-convex A (or B) is often represented as a set of (possibly overlapping) convex polygons; these convex polygons are then pairwise convolved to find a set of configuration space polygons whose union is $CO_A(B)$. This union is often computed using sweep-line techniques. Suppose now that we wish to rotate A by θ and then compute $CO_{A(\theta)}(B)$. $B \ominus A(\theta)$ is clearly a set in which rotated and non-rotated edges must be simultaneously processed. Using rational rotations, the effect on the motion plan would be that the robot rotates to some θ' (the output of our algorithm) within a tolerance ϵ of θ instead. Finally, the computation of $CO_{A(\theta')}(B)$ is exact, whereas, for most θ, $CO_{A(\theta)}(B)$ cannot be exactly represented.

2 Outline

The final algorithm is short and simple, belying the rich structure of the problem it solves and the assortment of techniques available to attack it. As such, we chronicle various approaches that lead to the final results. In section 3, we couch the problem in convenient terms. In section 4, we describe a brute force approach that yields a few rational sines. Section 5 gives our first iterative algorithm, with encouraging insights. Section 6 applies well-known results on Diophantine equations to show that finding sines is equivalent to finding rational approximations to arbitrary numbers. Section 7 shows a variant of Euclid's algorithm to perform this approximation. Section 8 analyzes the Diophantine technique and Euclid and finishes with the algorithm that satisfies the claims above. Section 9 gives more application examples. Section 10 relates this work to algebraic geometry and suggests extensions.

3 Setting Up the Problem

Let $s = \sin\theta$ and $c = \cos\theta$, then we are interested in rational solutions to:

$$s^2 + c^2 = 1 \tag{2}$$

Picking such a rational solution is clearly equivalent to picking θ such that $\sin\theta$ and $\cos\theta$ are both rational, but, as mentioned above, such θ values are often irrational, so we are not free to compute them directly; we must be more subtle.

Define a *rational sine* to be any rational solution for s in (2). An alternate definition will also be helpful: Let $\Omega(x) = \sqrt{1 - x^2}$. Then a rational sine is any rational number S such that $\Omega(S)$ is also rational.[2] What is wanted is an algorithm with this specification:

Inputs: Angle θ, tolerance ϵ_θ.

Output: Rational sine S such that:

(a) $|\sin^{-1} S - \theta| < \epsilon_\theta$

(b) S is as short as possible.

4 Simple-minded Search

Our first attempt is purely pragmatic. In implementing planners for paths of robots that rotate, one approach is to consider only a finite, uniformly spaced set of rotation angles in $[0, 2\pi)$. The first job of the planner is then to compute configuration space obstacles for this set of angles as described above in sec. 1.1. For this purpose, it appears natural to see if there are enough short rational sines to fill a table with reasonable density, perhaps one per degree. Considering the equation

$$\left(\frac{a}{b}\right)^2 + \left(\frac{c}{d}\right)^2 = 1 \quad \Rightarrow \quad b^2 - a^2 = \left(\frac{bc}{d}\right)^2$$

it is not hard to see that we are looking for all pairs of integers (a, b) such that $b^2 - a^2$ is a perfect square. The pragmatic approach is to search exhaustively for all of these with fewer than some small number of digits. It is surprising to find 159 pairs where $0 \leq a \leq b < 1000$ and $a/b < \sqrt{2}/2$, corresponding to sines for angles between 0 and $\pi/4$. Moreover, they are distributed evenly enough so that for most integral degrees, a sine is available within 0.3 degrees, and usually much closer. The exceptions are angles within 2 degrees of $90 \cdot k$ for integer k, where no 3 digit sines exist. Searching for 4 digit sines is feasible, especially to fill the empty spaces at 1 and 2 degrees, and this yields many more pairs. The 46 sines closest to whole degrees found in this manner are given in Figure 1. However, this is about the practical limit of brute force search. We have compromised on (a) of the specification to ensure (b).

5 Inspired Iteration

We next seek a way to iteratively refine the entries of the table to achieve arbitrarily precise results, satisfying (a) of the specification. The result is given

[2] Of course $\Omega(S)$ is also a rational sine, as $\Omega(\Omega(S)) = S$.

0 – 0	10 – 92/533	20 – 51/149	30 – 451/901	40 – 88/137
1 – 115/6613	11 – 93/485	21 – 135/377	31 – 180/349	41 – 48/73
2 – 57/1625	12 – 76/365	22 – 372/997	32 – 28/53	42 – 65/97
3 – 39/761	13 – 156/685	23 – 348/877	33 – 432/793	43 – 429/629
4 – 29/421	14 – 205/853	24 – 231/569	34 – 161/289	44 – 555/797
5 – 23/265	15 – 69/269	25 – 36/85	35 – 228/397	45 – 697/985
6 – 19/181	16 – 7/25	26 – 39/89	36 – 504/865	
7 – 32/257	17 – 120/409	27 – 300/661	37 – 3/5	
8 – 129/929	18 – 57/185	28 – 8/17	38 – 580/941	
9 – 100/629	19 – 12/37	29 – 189/389	39 – 341/541	

Figure 1: Short rational sines for 0 to 45 degrees.

1. Retrieve S, the closest rational sine to $\sin\theta$ from a precalculated table.
2. If $|\sin^{-1} S - \theta| < \epsilon$, return S.
3. Compute a correction angle $\delta_\theta = \sin^{-1} S - \theta$.
4. Compute a correction factor
$$k = \frac{2c-1}{2c^2-2c+1} \quad \text{where} \quad c = \left\lfloor \frac{1}{|\delta_\theta|} \right\rfloor$$
5. Set $S := S\Omega(k) \pm \Omega(S)k$, where the sign is opposite the sign of δ_θ. Go to Step 2.

Figure 2: The first algorithm

in Figure 2. The following is a brief description of the non-obvious steps. Step 5 is based on the sum of sines identity $\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$ and the fact that for small k, $\sin k \cong k$. The job of Step 4 then is to find a small k related to δ_θ that ensures convergence and has $\Omega(k)$ rational so that Step 5 always yields a rational number. This it does. The algorithm returns reasonably short sines, about twice the optimal number of bits, very quickly, but we have yielded on (b) of the specification to get (a). It remains to show that we can come quite close to achieving both at the same time.

6 Invoking Diophantus...

The thing to notice about the algorithm of Figure 2 is that finding a suitable k, an approximation of the sine of a correction angle, is much the same as our overall requirement. If the formula in Step 4 had the power to generate all possible corrections k such that $\Omega(k)$ is rational, it would also be able to generate all possible rational sines by iterating over all integers c. It is not clear what can be said about the sine-generating powers of the formula in Step 4—it was obtained *ad hoc*—but we can derive a similar form which is close to optimal for our purposes.

All solutions to the Diophantine equation

$$a^2 + b^2 = c^2$$

with b even and a, b, and c relatively prime are of the form

$$a = m^2 - n^2 \quad b = 2mn \quad c = m^2 + n^2$$

where m and n are integers. Rearranging, we have

$$\left(\frac{a}{c}\right)^2 + \left(\frac{b}{c}\right)^2 = 1$$

Thus all a/c and b/c are rational sines. Solving for b/c, we have

$$\frac{b}{c} = \frac{2mn}{m^2+n^2} = \frac{2}{\frac{m}{n}+\frac{n}{m}} = \frac{2}{t+1/t}$$

where t is any rational number. Thus the expression $2/(t+1/t)$ exactly characterizes *all* rational sines.[3]

[3]The restriction of the Diophantine solutions to even b is of no concern. If some rational sine b'/c' has b' odd, then the solution with $b = 2b'$, $c = 2c'$ accounts for it.

$e_0, p_0, q_0, e_1, p_1, q_1 := x, 0, 1, -1, 1, 0;$
repeat
  $r := \lfloor e_0/e_1 \rfloor;$
  $e_0, p_0, q_0, e_1, p_1, q_1 :=$
    $e_1, p_1, q_1, e_0 - re_1, p_0 - rp_1, q_0 - rq_1;$
until $|p_1/q_1 - x| < \epsilon$

Figure 3: Rational approximation algorithm

The outline of an algorithm immediately suggests itself:

Guess rational t such that $S = 2/(t + 1/t)$ satisfies $|\sin^{-1} S - \theta| < \epsilon$. Return S. (3)

Thus the problem is reduced to a search for t, but the search has a very specific goal. We can solve the equation $\sin\theta = 2/(x + 1/x)$ for x:

$$x = 1/\sin\theta + \sqrt{1/\sin^2\theta - 1} \quad (4)$$

Then x is the exact value of t we need, except that it is probably irrational. It remains to approximate x with a nearby rational number. We initially did this using bisection between $\lfloor x \rfloor$ and $\lceil x \rceil$, and this gave answers with one third fewer bits than the algorithm of Figure 2. However, we can do better.

7 ...And Then Euclid

There is a classical algorithm to approximate any number x with a series of successively closer (and longer) rationals $t_0, t_1, \ldots$. What makes it especially desirable for our needs is that the t_i are significantly shorter than the estimates we got by bisection; often they are the shortest possible. Figure 3 is the algorithm. The assignment statements perform all assignment of right hand side expression results to respective left hand side variables simultaneously. Variable x holds the number we are approximating. The approximations t_i are the values of p_1/q_1 in successive iterations. Iteration stops when this value is within ϵ of x. The astute reader will see embedded in this algorithm Euclid's method for finding the GCD of two integers. In effect, it is finding the "GCD" of x and 1.

The existence of this algorithm should not surprise Common Lisp users! The library function `rationalize` commonly uses it to get a rational nearby to a floating point number; e.g., `(rationalize .1111111111111111)` returns 1/9 in many implementations.

Incorporating Figure 3 and the discussion of Section 6 yields a complete algorithm. Figure 4 shows Common Lisp code to compute rational sines in $[0, \pi/4)$. Recall that `do` loops have simultaneous assignment semantics as the notation in Figure 3 above. The first line of the *do* evaluates the right hand side of (4) in double precision arithmetic to obtain the "exact" value of x to be approximated. Iteration proceeds as in Figure 3, with the stopping criterion from (3). The last line computes the current most accurate value of the rational sine due to the current most accurate rational approximation of x, namely `p1/q1`.

8 Analysis

How short are the answers of Figure 4? There are two things to consider. First, we developed this algorithm on the intuition that if we find a short t, it will indeed yield a short value of $2/(t+1/t)$. This intuition needs verification. Second, though Euclid's algorithm is appealing, we do not know if it really produces shortest answers for a given ϵ. We deal with these matters in order.

To address the first point, suppose S^* is the shortest rational sine for angles within ϵ_θ of θ, i.e. $|\sin^{-1} S^* - \theta| < \epsilon_\theta$. We need to show the following:

Claim: *Let t be the shortest rational such that $|\sin^{-1} S - \theta| < \epsilon_\theta$, where $S = 2/(t + 1/t)$. Then S is at most one bit longer than S^*.*

For this, we need some simple lemmas.

Lemma 1: *Given p and q relatively prime,* $\gcd(2pq,\ p^2 + q^2) \leq 2$.
Proof: If 2 divides $p^2 + q^2$, then 2 is certainly a common divisor. Now suppose $d > 1$ is another prime divisor. Then d divides $2pq$, so it divides either p or q but not both. Thus one of p^2 or q^2 is congruent to zero (mod d) but the other is not. This implies $p^2 + q^2$ is not divisible by d, a contradiction.

Lemma 2: *Let $t = p/q$ be the shortest rational in interval I (p, q relatively prime). Then $t' = p'/q' \in I$ ($p, q \in \omega$, $t' \neq t$) implies $p < p'$ and $q < q'$.*
Proof: $q < q'$ is immediate from the shortness of t, and $p < p'$ follows by an easy analysis with the Farey sequences discussed below.

Proof of claim: If $S = S^*$, then we are done. Otherwise S is longer than S^*, and we must determine how much longer it might be.

Let $t = p/q$, with p and q relatively prime. Then $S = (2pq)/(p^2 + q^2)$ by substitution. Similarly, we know $S^* = (2p'q')/(p'^2 + q'^2)$ for some relatively prime p' and q'. From lemma 2 we have

$$p < p' \quad \text{and} \quad q < q'. \quad (5)$$

```
(defun ssine (ang &optional (eps 0.01))
  (let ((s (sin ang)))
    (if (< ang eps) 0
      (do ((e0 (+ (/ 1 s) (sqrt (- (/ 1 (* s s)) 1))) e1)
           (p0 0 p1)
           (q0 1 q1)
           (e1 -1 (- e0 (* rat e1)))
           (p1 1 (- p0 (* rat p1)))
           (q1 0 (- q0 (* rat q1)))
           (rsin 1)
           rat)
          ((< (abs (- ang (asin (coerce rsin 'double-float)))) eps) rsin)
        (setq rat (truncate e0 e1)
              rsin (/ (* 2 p1 q1) (+ (* p1 p1) (* q1 q1))))))))
```

Figure 4: Code for a better sine algorithm

Let $d = \gcd(2pq, p^2+q^2)$ and $d' = \gcd(2p'q', p'^2+q'^2)$. From lemma 1, we have

$$d \geq d'/2 \tag{6}$$

and wish to prove $(p^2+q^2)/d < 2(p'^2+q'^2)/d'$, i.e., the denominator of S is less than twice that of S^*, and thus at most a bit longer. Then

$$\frac{p^2+q^2}{d} \overset{(5)}{<} \frac{p'^2+q'^2}{d} \overset{(6)}{\leq} \frac{p'^2+q'^2}{d'/2} = 2\frac{p'^2+q'^2}{d'}. \quad \square$$

To summarize, this shows that if we really *do* find the shortest t, then $2/(t+1/t)$ is either the shortest sine, or another sine only a bit longer.

That leaves the second point: Does the algorithm of Figure 3 produce shortest answers? The answer is *no*, but it is quite close to one that does. To see what's going on, we need a simple tool from number theory, Farey Sequences (see e.g. [Rad84]). The Farey Sequence of order N is just the ascending sequence of canonically reduced fractions between zero and one with denominators not exceeding N. There are two fundamental theorems concerning them:

Theorem 1. Every reduced positive rational less than one appears in a Farey sequence.

Theorem 2. Every element of the Farey sequence of order $N+1$ that is not in the sequence of order N is an N-mediant.

An N-mediant is a number $(a+b)/(c+d)$, where a/c and b/d are adjacent elements of the order N sequence. Noting these theorems, we claim there is an obvious algorithm to find the sequence of fractions that approximate any x, $0 \leq x < 1$ in ascending order of length. We start with the order 1 sequence $0/1, 1/1$ and add the mediants closest to x in succession. Refining this idea a little, we obtain the algorithm of Figure 5. Why have we bothered with Euclid's algorithm at all when this is clearly correct and optimal

$p_0, q_0, p_1, q_1, := 0, 1, 1, 1;$
loop
  $a, b := p_0 + p_1, q_0 + q_1;$
  if $x < a/b$ then
    $p_1, q_1 := a, b$
    if $|p_1/q_1 - x| < \epsilon$ then return p_1/q_1
  else
    $p_0, q_0 := a, b;$
    if $|p_0/q_0 - x| < \epsilon$ then return p_0/q_0

Figure 5: Farey approximations to x

in the length of results? Unfortunately, Figure 5 requires $O(d)$ iterations, where d is the *denominator* of the returned value. Our version of Euclid's algorithm requires only $O(\log d)$ iterations.

Thus we cannot use Figure 5 directly, but we *can* use it for insight into Euclid's algorithm in Figure 3. Consider maximal sequences of adjacent iterations of the loop in Figure 5 where the "if" condition tests *true* (resp. *false*). Call these T (resp. F) sequences. Then it is not hard to see that the effect of a T sequence is to add $r(p_0/q_0)$ to p_1/q_1 where $r = |T|$. Respectively, F sequences add $r'(p_1/q_1)$ to p_0/q_0 where $r' = |F|$. This suggests that we can speed up Figure 5 by calculating r and r' for adjacent T and F sequence pairs, replacing all iterations for a T, F pair by exactly one. We have done this in Figure 6.

The summary result is this: A tedious algebraic analysis (not given here) shows that Figure 6 yields the same answers as Figure 3, at least for $0 < x \leq 1/2$. Thus Euclid's algorithm effectively simulates the successively tighter bracketing by Farey mediants of Figure 5, but does it much faster.

Despite this interesting equivalence, we cannot conclude that Figures 3 and 6 give shortest answers. Why? Precisely because entire T and F sequences

$p_0, q_0, p_1, q_1 := 0, 1, 1, 1;$
loop
  $r := \lfloor (p_1 - q_1 x)/(q_0 x - p_0) \rfloor$
  $p_1/q_1 := (rp_0 + p_1)/(rq_0 + q_1)$
  if $|p_1/q_1 - x| < \epsilon$ then return p_1/q_1
  $r' := \lfloor (xq_0 - p_0)/(p_1 - q_1 x) \rfloor$
  $p_0/q_0 := (r'p_1 + p_0)/(r'q_1 + q_0)$
  if $|p_0/q_0 - x| < \epsilon$ then return p_0/q_0

Figure 6: Speedy Farey approximations to x

are replaced by single iterations in Figure 6. Each iteration of Figure 5 increases the length of the answer, and the first approximation close enough to x to satisfy the stopping criterion may be generated *in the middle* of a T or F sequence. Figure 6 applies the stopping criterion only at the *ends* of T and F sequences.

Thus we have at once characterized the cases where Euclid's algorithm produces non-shortest answers and suggested how to solve the problem. We need to ensure that the calculation of r and r' in Figure 6 (resp. r in Figure 3) does not go too far, producing a too big value that steps past the shortest mediant. Incorporating this idea, Figure 7 is the final code with separate functions to approximate numbers and give rational sines. With $\epsilon_\theta = 10^{-4}$, the sine function gives answers one to three bits shorter that Figure 4 about a third of the time for random angles. Interestingly, about one in twenty returns is a bit *longer* than Figure 4. This occurs when a non-shortest t generated by Figure 3 happens to yield a one bit shorter sine than the shortest t. This possibility is allowed by our earlier claim, and we are thus assured that it occurs in fact. Though this implementation is sufficient for perhaps most applications, it is worth noting that all the double precision arithmetic can be replaced by rational arithmetic to achieve any precision. E.g., we can use the factorial series expansion of the sine and Newton iteration for the square root.

We now briefly discuss the fundamental complexity of the final algorithm. It is easy to verify that the size of the result denominator grows by at least a bit in each iteration. The final size is $n = \log(1/\epsilon)$ bits, and none of the other operands in the calculation are significantly longer. Assuming Schönhage-Strassen[4] multiplication, doing arithmetic on them takes $O(n \log n \log\log n)$ time. There are $O(n)$ iterations, which means $O(n^2 \log n \log\log n)$ time. A remaining detail is the complexity of computing rational approximations to the sine and square root to initialize the iteration. We merely note that Newton's method converges quadratically and the sine series linearly (as our algorithm). Since these are executed only twice each, and there is also no problem with excessively long intermediate operands, these calculations do not increase the overall complexity. Finally, we note there are faster GCD algorithms than our Farey derivative that are very well known (see [AHU74], Thm. 8.20.). These algorithms are $O(n \log^2 n \log\log n)$ and extend to finding rationals. However, whether our size-optimality analysis and modifications extend as well remains open.

[4]E.g. MIT SCHEME and Kyoto COMMONLISP use this method, so it is not a merely theoretical invocation.

9 Applications

This section applies the rational method to examples of the many geometric algorithms that require rotations to remove degeneracies, simplify intersection calculations, represent rotations of modeled objects, etc. These algorithms are often presented and analyzed using the real-RAM model, which allows rotation to arise in subtle forms. For instance, the statement "assume the direction of sweep is parallel to the x axis" masks a prescription for rotation. The more realistic bit-complexity model demands greater precision, and the work described here lets us provide it; we can now write, "Construct a rotation matrix that brings the sweep direction to the positive x axis... Approximate the entries with rationals... Apply the resulting transformation to the input..." or, "Choose a pure rotation within ϵ of θ." This validates the algorithm's correctness and usefulness in practical settings without reliance, implicit or otherwise, on robust, limited precision techniques for implementation. Thus, we believe these examples elaborate a significant *lacuna* in robust geometric algorithms.

The first two applications involve rotating the input as a pre-process to an algorithm, and then "unrotating" the output. One is a line-sweep in an arbitrary direction. The other is a perturbation of input to remove vertical degeneracies. Note that pure rotations are not optimal for these algorithms in their basic form for this reason: If all we need is the finally unrotated output, then we can find coefficients that rotate to within the desired tolerance *and* introduce an arbitrary scaling with fewer bits than a pure rotation matrix (about half as many suffice). This is possible because the inverse of such a transformation also has rational coefficients—the scaling introduced in rotating can be exactly undone while unrotating—and scaling does not affect correctness by changing topology. Scaling *does* cause problems, though, if

```
;;; Find the smallest rational between x0 and x1.
(defun rat (x0 x1)
  (let ((i (ceiling x0)) (i0 (floor x0)) (i1 (ceiling x1)))
    (if (>= x1 i) i
      (do ((p0 i0 (+ p1 (* r p0)))
           (q0 1 (+ q1 (* r q0)))
           (p1 i1 p0)
           (q1 1 q0)
           (e0 (- i1 x0) e1p)
           (e1 (- x0 i0) (- e0p (* r e1p)))
           (e0p (- i1 x1) e1)
           (e1p (- x1 i0) (- e0 (* r e1))) r)
        ((<= x0 (coerce (/ p0 q0) 'double-float) x1) (/ p0 q0))
      (setq r (min (floor e0 e1) (ceiling e0p e1p)))))))

;;; Return a small rational sine for an angle in [a0,a1], 0 <= a0 < a1 < pi/2
(defun rat-sin (a0 a1)
  (if (zerop a0) 0
    (let* ((s0 (sin a0)) (s1 (sin a1))
           (tt (rat (+ (/ s1) (sqrt (1- (/ (* s1 s1)))))
                    (+ (/ s0) (sqrt (1- (/ (* s0 s0))))))))
      (/ 2 (+ tt (/ tt))))))
```

Figure 7: Shortest rational approximations and short sines

ever we must augment the algorithms to compare a distance, area, or volume in the rotated problem to one in the original. It is not hard to imagine such a requirement.[5] We conclude that while a pure rotation may not be theoretically necessary for these and similar algorithms, it will often be highly desirable in practice.

The third application is computing visibility graphs for an environment containing a rotating robot where pure rotations are essential for consistency. Finally, we propose an application in computer graphics.

9.1 Rotating line sweep problems

To employ the Canny–Donald[6] line sweep projection algorithm in a multistep compliant motion (robot) planner with uncertainty in sensing and control, it is sufficient to compute projections for a finite set of robot velocity angles. For each, we rotate the environment and robot so that the velocity vector points along the negative x axis as shown in Figure 8. The sweep line then moves in the positive x direction and computation of intersections and distances is greatly simplified.[7] We can use the rational rotation method

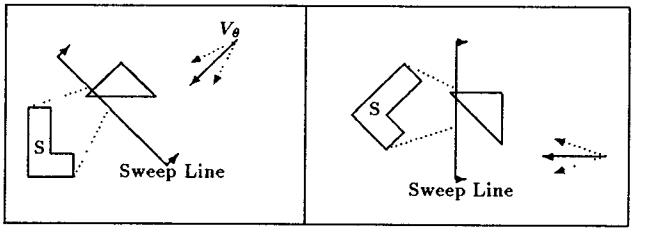

Figure 8: Rotating a projection problem.

presented here to do this rotation of the environment.

9.2 Tweaking line sweep problems

The characterization of rational sines by $2/(t + 1/t)$ with t rational has possibilities beyond the algorithm of Figure 7. Many line sweep algorithms are oblivious to the direction of the sweep line. Another common feature is that segments parallel to the sweep line must be treated as special cases. It might be much simpler and even more efficient to get rid of these cases by rotating the problem so there are no such segments. We can use our new understanding of rational sines to compute short rotation coefficients for this purpose: Generate the canonically reduced rationals $0 \leq t \leq 1$ in ascending order of length and pick the first one such that $2/(t + 1/t)$ is the sine for a rotation angle with the desired effect. The t values are just the stream of N-mediants of Farey sequences in increasing order of N. We omit further details.

[5] To make this work without pure rotations, we would have to compute the square of the scale introduced by the impure matrix (i.e. its determinant; the scale itself is usually irrational) and use it in every comparison.

[6] See any of [Don89, Don90, Lat91]. For readers unfamiliar with this algorithm, the same issues arise in any sweep-line algorithm, e.g., [PS85].

[7] Rotating the mountain to Mohammed, if you will.

9.3 Visibility graphs

Suppose we compute the visibility graph G of a set of polygons $A \cup B$. Denote by E_A the (set of) edges of A. Note that as a graph, $E_A \subset G$. Now, compute the visibility graph $G(\theta)$ of $A(\theta) \cup B$. Define $E_A(\theta)$ to be edges of $A(\theta)$, so $E_A(\theta) \subset G(\theta)$. Unless pure rotations are used, the lengths of the edges in E_A will be from the edges in $E_A(\theta)$. Most disturbing! This application arises in motion planning for a robot that can rotate, where A models the robot and B models the environment.

9.4 Graphics transformations

Many computer graphics applications are characterized by large databases of polygons with integer vertex coordinates. Homogeneous transformations must be applied to these very rapidly, perhaps many times a second. Floating point coefficients are a possibility, but another approach that has been successful for extracting good performance from inexpensive integer-only processing units is to use rational transformation coefficients with a restricted number of bits, say n. In the 3D case, the transformation matrix M is 4×4, and the desired transformation of a vertex $[v_0, v_1, v_2]$ is as follows:

$$v_i' = \text{round}\left(\frac{M_{i3} + \sum_{j=0}^{2} M_{ij} v_j}{w}\right),$$

$$\text{where } w = M_{33} + \sum_{j=0}^{2} M_{3j} v_j$$

Careful rewriting allows this to be calculated with 15 integer multiplies and three divides. Additional economies are possible when the structure of M is restricted.

Our algorithms are useful for forming M. Typically, it will be the concatenation of several transformations, say rotation, scaling, and perspective. We create a pure rotation first using some heuristic choice for its accuracy, perhaps $2^{-n/4}$ radians. Then we concatenate the desired scaling and perspective in a floating point or arbitrary precision matrix, and approximate its entries as accurately as possible without overflowing n bits in the result. A conservative tolerance is not hard to derive.

10 Relation to Algebraic Geometry

Algebraic geometry points to generalizations of this work: eq. (2) is a genus 0 curve, which implies it can be parameterized in one variable t. Our characterization of rational sines results from the choice $x = t(y-1)$ (i.e., substitution in (2) yields $y = (t^2-1)/(t^2+1) = \Omega(2/(t+1/t))$). All quadratic curves have similar parameterizations. These yield algorithms for short rational hyperbolic sines and cosines (for example). [Sha77] gives a general parameterization method and also explores its limitations for algebraic functions of higher degree. Note that this technique will not work in general; for example it won't work for elliptic functions!

Then the following interesting question arises: Recall the "one-bit-from-optimal" property we prove for the rational sine parameterization. We ask: *Do such parameterizations of all quadratic curves have this property?*

In partial answer, we believe they do not. The question devolves to answering the following:

Question: *Given an algebraic curve C defined by $f(x,y) = 0$, of total degree 2 and genus 0.*

Pick a rational point (x_0, y_0) on the curve. Consider a line $l(t)$ through (x_0, y_0) with rational slope t. Then the other intersection (x, y) of $l(t)$ with C is a rational point. Hence, t parameterizes all such rational points (x, y). Moreover, the choice of (x_0, y_0) is canonical.

So let t be the rational p/q, and consider the equation $f(x, y_0 + (p/q)(x - x_0)) = 0$. Solve (by quadratic formula) to get $x = g(p,q)/h(p,q)$, where g and h are relatively prime integer polynomials.

Then, for any relatively prime p, q is $\gcd(g(p,q), h(p,q)) \leq 2$?

We believe that $\gcd(g(p,q), h(p,q)$ can be as large as the product of the coefficients of f. It remains to demonstrate this as an upper or lower bound.

References

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, 1974.

[Bri89] Briggs, A. An Efficient Algorithm for One-Step Compliant Motion Planning with Uncertainty. In *Proc. of the 5th ACM Symp. on Computational Geometry*, Saarbrucken, BRD, pp 187–196, 1989.

[BM89] Brost, R. and M. Mason. Graphical Analysis of Planar Rigid Body Dynamics with Multiple Frictional Contacts. In *Proc. 5th Intl. Symp. on Robotics Research*, Tokyo, 1989, pp. 367–374.

[Can89] Canny, J.F. On The Computability of Fine-Motion Plans. In *Proc. IEEE ICRA*, Scottsdale, Arizona, 1989.

[Don89] Donald, B. *Error Detection and Recovery in Robotics, Lecture Notes in CS, Vol. 336.* Springer-Verlag, New York, 1989.

[Don90] Donald, B. The Complexity of Planar Compliant Motion Planning with Uncertainty. *Algorithmica*, Vol. 5, No. 3, 1990, pp. 353–382.

[DP90] Donald, B. and D. Pai. On The Motion of Compliantly-Connected Rigid Bodies in Contact: A System for Analyzing Designs for Assembly. In *Proc. IEEE ICRA*, Cincinatti, Ohio, May 1990.

[Erd86] Erdmann, M. Using Backprojections for Fine Motion Planning with Uncertainty. *IJRR*, Vol. 5 No. 1, 1986.

[AFW88] Aronov, B., S.J. Fortune and G. Wilfong. The furthest-site geodesic Voronoi diagram. In *Proc. 4^{th} ACM Symp. on Computational Geometry*, 1988, pp. 229-240.

[HHK88] Hoffmann, C.M., J.E. Hopcroft and M.S. Karasick. Towards implementing robust geometric computations. In *Proc. 4^{th} ACM Symp. on Computational Geometry*, 1988, pp. 106-117.

[LoP83] Lozano–Pérez, T. Spatial Planning: A Configuration Space Approach. *IEEE Trans. on Computers*, Vol. C–32, 1983, pp. 108–120.

[Lat91] Latombe, J.-C. *Robot Motion Planning.* Kluwer, 1991.

[LM90] Li, Z. and V. Milenkovic. Constructing strongly convex hulls using exact or rounded arithmetic. In *Proc. 6^{th} ACM Symp. on Computational Geometry*, 1990, pp. 235–243.

[MN90] Milenkovic, V. and L.R. Nackman. Finding compact coordinate representations for polygons and polyhedra. In *Proc. 6^{th} ACM Symposium Computational Geometry*, 1990, pp. 244–252.

[PS85] Preparata, F. and M. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[Rad84] Rademacher, H. *Lectures on Elementary Number Theory.* Robert E. Krieger, Malabar, Florida, 1984.

[Sha77] Shafarevich, I.R. *Basic Algebraic Geometry.* Springer-Verlag, Berlin, 1977.

[SI89] Sugihara, K. and M. Iri. A Solid Modelling System Free from Topological Inconsistency. *Jour. of Information Processing*, Vol. 12, No. 4, 1989, pp. 380–393.

Acknowledgments. We are especially grateful to Robert Freimer and Klara Kedem, who read drafts of this paper closely, and provided many corrections and insights on both presentation and substance. Amy Briggs, Dinesh Pai, and Rich Zippel also provided helpful comments, for which we are very grateful.