A Rational Rotation Method for Robust Geometric Algorithms (Extended Abstract)

John Canny*
Bruce Donald**
Gene Ressler***

TR 91-1247
December 1991

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Computer Science Division, University of California at Berkeley, Berkeley, CA 94702
**Computer Science Department, Cornell University, Ithaca, NY 14853
***For other work on robust geometric algorithms, see, for example, [Milenkovic; Fortune; Hopcroft, Hoffman and Karasick.]

A Rational Rotation Method for Robust Geometric Algorithms (Extended Abstract)

John Canny[*] **Bruce Donald**[†] **Gene Ressler**[†]

December 1, 1991

1 Introduction

Algorithms in computational geometry often use the real-RAM model of computation. In particular, this model assumes that exact real numbers can be stored in and retreived from memory in constant $O(1)$ time, and that field operations ($+$, $-$, $*$, $/$) and certain other operations (like square root, sine, and cosine) are also "exact," and can be applied in constant time.

This assumption makes theoretical computational geometry algorithms—even well-understood algorithms like plane-sweep for polygon union [PS] difficult to implement and numerically unstable. These algorithms obtain good combinatorial complexity bounds by exploiting *ordering* properties of the edges, vertices, and intersections. Floating point implementations are subject to numerical problems; for example, the "symbolic" ordering properties and the "numerical" ordering properties do not agree, and the algorithms fail. This is not a "mere engineering difficulty"—it prevents these algorithms from being used in practice. One possible solution is to use *rational arithmetic*, which is exact. For certain algorithms (eg plane-sweep), one can show that in a careful implementation, the "size" (number of bits) of the rational numbers at most doubles. Hence, on the face of it, it would seem that rational arithmetic helps solve our problem.[1]

However, many applications from scientific computation and artificial intelligence require that these algorithms be mixed, or interveaved, 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 $A(\theta)$; ie, for each vertex (x, y) of A we compute a new one (x', y') by the familiar transformation:

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

This models a physical rotation of the object modeled by A with respect to B, 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

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

[†]Computer Science Department, Cornell University, Ithaca, NY 14853

[1]For other work on robust geometric algorithms, see, for example, [Milenkovic; Fortune; Hopcroft, Hoffman, and Karasick.]

real numbers are irrational. Of course, arbitrary irrationals cannot be represented in computing machines. We will have to approximate somewhere, but where?

One idea is simply to approximate the irrational sines and cosines in (1) with "nearby" rationals. However the resulting transformation is typically no longer orthonormal — no longer 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 in that it is inconsistent with A and B as models of rigid objects.

Hence the thrust of this paper is to find approximate rotation coefficients that do not introduce unwanted scaling. We give efficient algorithms to accomplish this. More precisely, for given angle θ and tolerance ϵ_θ, our final algorithm returns a rational S with the following properties:

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

For our purposes, the *length* of a rational is the magnitude of its denominator. Thus rational P is *shorter* (resp. *longer*) than rational Q if P's denominator has smaller (resp. greater) magnitude. The *number of bits* of a rational is the number of bits in its denominator not counting leading zeros.

Shortness of sines is vital for applications, as in general, the number of bits in the rotation coefficients is added to the bits in each point in obtaining the rotated points.

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 rationals of n bits. With Schönhage-Strassen multiplication, this yields overall complexity of $O(n^2 \log n \log\log n)$. Though it is practical to implement the algorithm entirely with rational arithmetic and thus achieve arbitrary precision, 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 used to approximate arbitrary numbers with nearby rationals to within given ϵ. An example is the `rationalize` function of Common Lisp implementations (where ϵ is the 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 the rational method is that it is distance-preserving (i.e., pure rotations are isometries), and it has an inverse of the same complexity. To further illustrate why this is important, we now discuss a fundamental algorithm in robot motion planning where scale-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 [LoP]). $CO_A(B)$ can be characterized using the Minkowski sum $B \ominus A = \{\, b - a \mid a \in A, b \in B \,\}$. [LoP] 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 exactly 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 computed or represented using rational arithmetic.

2 Outline

The final algorithm is short and simple, belying the rich structure of the problem it solves 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 10 has some notes to place this work in context, 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 have not the freedom 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:

Specification: Rational sine

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

[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.

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

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

it is not hard to see that we are looking for all pairs of integers (a, b) such that $a^2 - b^2$ 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 $0 \leq b \leq a < 1000$ 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. However, this is about the practical limit of brute force search. We have compromised on (a) of the specification to ensure (b). The result is given in Figure 1.

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 is given in Figure 5. The following is a brief description of the non-obvious steps. Step 5 is apparently 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 $\delta\theta$ 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 5 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

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
$$k = \frac{2c-1}{2c^2-2c+1} \quad \text{where} \quad c = \lfloor 1/\delta\theta \rfloor$$
5. Set $S := S\Omega(k) \pm kS$, where the sign depends on the sign of $\delta\theta$ in the obvious way. Go to Step 2.

Figure 2: The first algorithm

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

$$x^2 + y^2 = z^2$$

with y even and x, y, and z relatively prime are of the form

$$x = m^2 - n^2 \quad y = 2mn \quad z = m^2 + n^2$$

where m and n are integers. Rearranging, we have

$$\left(\frac{x}{z}\right)^2 + \left(\frac{y}{z}\right)^2 = 1$$

Thus all x/z and y/z are rational sines. Picking y/z because the math is a bit simpler, we have

$$\frac{y}{z} = \frac{2mn}{m^2+n^2} = \frac{2}{\frac{m}{n}+\frac{n}{m}} = \frac{2}{x+1/x}$$

where x is any rational number. Thus the expression $2/(x+1/x)$ exactly characterizes *all* rational sines.[3] The outline of an algorithm immediately suggests itself:

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

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

$$x' = 1/\sin\theta + \sqrt{1 - 1/\sin^2\theta} \tag{4}$$

[3]The restriction of the Diophantine solutions to even y is of no concern. If some rational sine p/q has an odd numerator, then the solution with $y = 2p$, $z = 2q$ 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

Then x' is the exact value of x 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 5. However, we can do better.

7 ...And Then Euclid

There exists an iterative algorithm to approximate any number x with a series of successively closer rationals $R_0, R_1, \ldots$. It terminates, returning x in canonical reduced form, if and only if x is rational. What makes it especially desirable for our needs is that the R_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 R_i are the values of p_1/q_1 in successive iterations. Iteration stops when this value is within ϵ of x. The estute 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 4 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 x, it will indeed yield a short value of $2/(x + 1/x)$. This intuition needs verification. Second, though Euclid's algorthm 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. $|\sin^{-1}\theta - S^*| < \epsilon_\theta$.

```
(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 the final algorithm

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

Lemma: *Given p and q relatively prime,* $\gcd(2pq,\ p^2+q^2) \le 2$.

Proof of lemma: 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.

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 $x = p/q$. Then p and q are relatively prime by the shortness of x. Next, note $S = (2pq)/(p^2+q^2)$. Similarly, we know $S^* = (2p'q')/(p'^2+q'^2)$ for some relatively prime p' and q' longer than p and q. By the lemma, this can only occur if $\gcd(2p'q',\ p'^2+q'^2) = 2$ and $\gcd(2pq,\ p^2+q^2) = 1$. We conclude the denominator of S is less than two times that of S^*, thence S is at most one bit longer than S^*. $\square$

To summarize, we have shown that if we really *do* find the shortest x, then $2/(x+1/x)$ 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. 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 then 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, $0 \le x < 1$ in ascending order of length. We start with the order

```
$p_0, q_0, p_1, q_1, a, b := 0, 1, 1, 1, 0, 1;$
loop
  $a, b := p_0 + p_1, q_0 + q_1;$
  if $x < a/b$ then
    $p_1, q_1 := a, b$
    if $p_1/q_1$ is close enough to $x$, return $p_1/q_1$
  else
    $p_0, q_0 := a, b;$
    if $p_0/q_0$ is close enough to $x$, return $p_0/q_0$
```

Figure 5: Farey approximations to x

```
$p_0, q_0, p_1, q_1 := 0, 1, 1, 1;$
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 $p_1/q_1$ is close enough to $x$, 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 $p_0/q_0$ is close enough to $x$, return $p_0/q_0$
```

Figure 6: Speedy Farey approximations to x

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 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 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 it 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 result of all this is the following: Figure 6 transforms (by tedious algebraic manipulation not given here) to the algorithm of Figure 3! Thus Euclid's algorithm effectively simulates the successively tighter bracketing by Farey mediants of Figure 6, but does it much faster.

Yet we cannot jump to a conclusion that Figures 3 and 6 give shortest answers. Why? Precisely because entire T and F sequences are replaced by single iterations in Figure 6. Iterations of Figure 5 increase 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 exactly when 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. The sine function gives answers one to three bits shorter that Figure 4 about a third of the time. Interestingly, about one in twenty returns is a bit *longer* than Figure 4. This occurs when a non-shortest x generated by Figure 3 happens to yield a one bit shorter sine than the shortest x. This possibility is allowed by our earlier Claim, and we are thus assured that it occurs in fact.

```
;;; 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]
;;; with a0<a1 and both in [0..pi/4].
(defun rat-sin (a0 a1)
  (if (zerop a0) 0
    (let* ((s0 (sin a0)) (s1 (sin a1))
           (x (rat (+ (/ s1) (sqrt (1- (/ (* s1 s1)))))
                   (+ (/ s0) (sqrt (1- (/ (* s0 s0))))))))
      (/ 2 (+ x (/ x))))))
```

Figure 7: Shortest rational approximations and short sines

Though this implementation is sufficient for many applications, it is worth noting that all the double precision calculations can be replaced by rational approximations to achieve arbitrary precision. Eg, we can use the well-known factorial series expansion of the sine and Newton iteration for the square root calculations.

We now discuss the overall complexity of the rational method. The size of the integers/rationals in Euclid is growing linearly. Their final size is $n = \log(1/\epsilon)$ bits. Assuming Schönhage-Strassen[4] multiply, 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. Of course, there are faster GCD algorithms that are very

[4]MIT SCHEME and Kyoto COMMONLISP use this method, so we will assume it.

well known;[5] these algorithms are $O(n \log^2 n \log\log n)$. However, the analysis with Farey sequences does not extend in a trivial manner to the faster GCD algorithms, so if we bring them in, we lose size optimality unless we can extend the proof and algorithms.

One 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 there is also no problem with excessively long intermediate operands, these calculations do not change the stated complexity.

9 Applications

We now consider two applications in which our method is practical but not the most efficient solution known. We present them because some algorithms in computational geometry require "rotation" of geometric objects—eg, to remove degeneracies, or to simplify intersection calculations. Some of these requirements are subtle: eg, a statement like "Assume the direction of sweep is parallel to the x axis" may actually mask a prescription for rotation. The development of robust geometric algorithms analyzed in the bit-complexity model demands that such prescriptions be made precise. Hence, we would rather see this "rotation" phrased as "Construct the rotation matrix for θ. Approximate the entries with rationals using Euclid's algorithm. Apply the resulting transformation to the environment..." or, "Use the rational method to a choose a rational rotation within ϵ of θ ..." Short of this precision, we provide the examples below to show how geometries can be exactly rotated, using the rational method, as prescribed by the algorithms. Hence, these examples in a sense demonstrate the correctness of Real-RAM algorithms that prescribe rotation or perturbation of the input, even when implemented using rational arithmetic. Furthermore, they demonstrate that the resulting rational implementations have the same complexity bounds as the Real-RAM idealizations, when analyzed in the bit-complexity model. It is our belief that these examples elaborate a significant *lacuna* in robust geometric algorithms.

Both these examples involve "rotating" the environment as a pre-process to an algorithm, and then "unrotating" the output of the algorithm. The first example is a line-sweep in an arbitrary direction. The second is a "perturbation" of the environment to remove vertical degeneracies. Both arise frequently in geometric algorithms. The rational method is not optimal for these problems, since it suffices to rotate/perturb the environment by an affine rational matrix that is the concatenation of a rotation and a scaling. Such a matrix can be obtained by simply approximating each entry in the (initial) rotation matrix using Euclid's method (or our optimal version). The topology of the transformed environment is not different, and hence this approximation method would suffice. Nevertheless, we feel it is interesting to see that these operations (problem rotation and perturbation) can be done using pure rational rotation matrices, and that is why we include these examples.[6]

[5] See, eg, [Aho, Hopcroft, and Ullman] Thm 8.20.

[6] A third example involves shortest paths. 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 an exact rational rotation method is used, the lengths of the edges in E_A will be *different* from the edges in $E_A(\theta)$. Most disturbing! This application arises in motion planning under rotations.

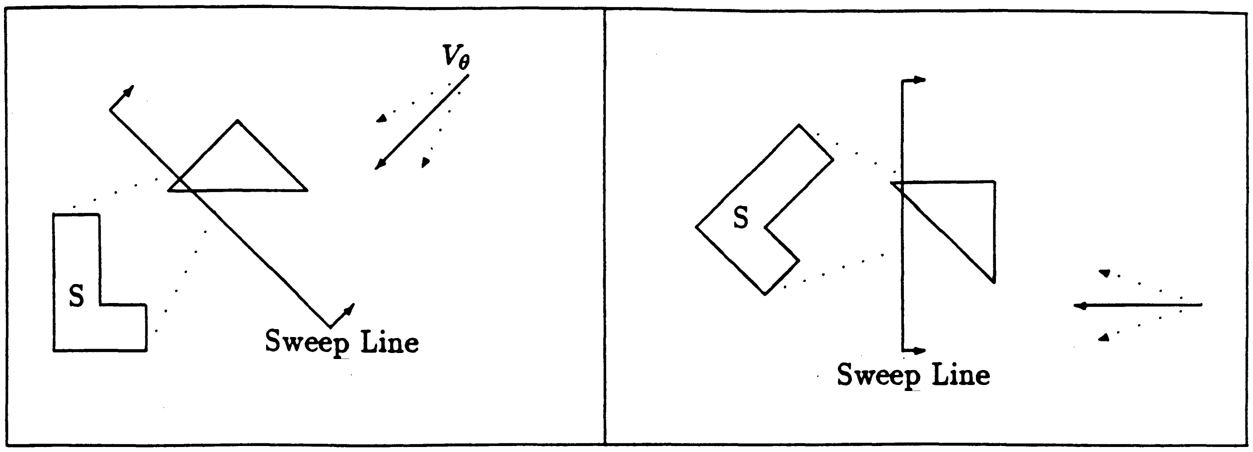

Figure 8: Rotating a projection problem.

9.1 Rotating Line Sweep Problem Instances

To employ the Canny–Donald[7] 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.[8] We can use the rational rotation method presented here to do this rotation of the environment.

9.2 Tweaking Line Sweep Problem Instances

The characterization of rational sines by $2/(x + 1/x)$, x rational, has another possible application. Many line sweep algorithms in the computational geometry literature are oblivious to the direction of the sweep line. Another common feature is that segments parallel to the sweep line must often 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 $0 \leq x \leq 1$ in ascending order of length and pick the first one such that $2/(x+1/x)$ is the sine for a rotation angle with the desired effect. One approach to generating the x values is to consider the stream of N-mediants of Farey sequences in increasing order of N. Details of an efficient algorithm are left as an exercise. ☺

10 ... About Algebraic Geometry

Finally, we note we are looking for rational solutions of eq. (2). Since this is a genus 0 curve, its solutions can be parameterized in terms of one variable (see Shaferavitch, first section of first

[7]See [Don; D2; Lat]. For readers unfamiliar with this algorithm, the same issues arise in any sweep-line algorithm, eg, [PS].

[8]This is rotating the mountain to Mohammed, if you will.

chapter). This parameterization immediately gives you $2x/(x^2+1)$. It follows that it should be possible to use these methods to produce short rational hyperbolic sines and cosines as well.

Finally, this technique generalizes and shows why we can't succeed if we want to use it on elliptic functions.

References

[Bri] **Briggs, Amy.** (1989) *An Efficient Algorithm for One-Step Compliant Motion Planning with Uncertainty,* 5th ACM Symp. Comp. Geom., Saarbrucken, BRD

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

[Can] **Canny, J.F.** *On The Computability of Fine-Motion Plans,* Proc. IEEE ICRA Scottsdale, AZ (1989).

[Don] **Donald, B. R.** *Error Detection and Recovery in Robotics,* Springer-Verlag (New York): Lecture Notes in CS, vol. 336 (1989).

[D2] **Donald, B.** "The Complexity of Planar Compliant Motion Planning with Uncertainty", *Algorithmica,* 5 (3), 1990 pp. 353-382.

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

[Erd] **Erdmann, M.** *Using Backprojections for Fine Motion Planning with Uncertainty,* IJRR Vol. 5 no. 1 (1986).

B. Aronov, S.J. Fortune and G. Wilfong *The furthest-site geodesic Voronoi diagram,* In Proc. 4th Ann. ACM Sympos. Comput. Geom. 229-240. (1988).

C.M. Hoffmann, J.E. Hopcroft and M.S. Karasick *Towards implementing robust geometric computations,* In Proc. 4th Ann. ACM Sympos. Comput. Geom., 106-117. (1988).

[LoP] **Lozano-Pérez, T.** *Spatial Planning: A Configuration Space Approach,* IEEE Trans. on Computers (C-32):108-120 (1983a).

[Lat] **Latombe, J.-C.** *Robot Motion Planning,* Kluwer (1990).

Z. Li and V. Milenkovic *Constructing strongly convex hulls using exact or rounded arithmetic,* In Proc. 6th Ann. ACM Sympos. Comput. Geom., 235-243. (1990).

V. Milenkovic and L.R. Nackman *Finding compact coordinate representations for polygons and polyhedra,* In Proc. 6th Ann. ACM Sympos. Comput. Geom. 244-252. (1990).

[PS] **Preparata F. & Shamos M.** *Computational Geometry: An Introduction,* Springer-Verlag (1985).

Acknowledgments. We would like to thank Rich Zippel and Dinesh Pai for their comments and suggestions, which greatly improved the substance and presentation of this paper. We are very grateful to them for their generosity with their time and ideas. We would also like to thank Amy Briggs for her comments and suggestions.