Exact Geometric Computation in LEDA

C. Burnikel* J. Könemann† K. Mehlhorn* S. Näher‡ S. Schirra* C. Uhrig*

Almost all geometric algorithms are based on the Real-RAM model. Implementors often simply replace the exact real arithmetic of this model by fixed precision arithmetic, thereby making correct algorithms incorrect. Two approaches have been taken to remedy this situation. The first approach is redesigning geometric algorithms for fixed precision arithmetic. The redesign is difficult and can inherently not lead to exact results. Moreover it prevents application areas from making use of the rich literature of geometric algorithms developed in computational geometry. The other approach advocates the use of exact real arithmetic.

In the LEDA platform of combinatorial and geometric computation [4] the new data-type **real** facilitates exact computation for many geometric problems. Every integer is a **real** and **real**s are closed under the operations addition, subtraction, multiplication, division and squareroot. All comparison operators $\{>, \geq, <, \leq, =\}$ are *exact*. In order to determine the sign of a real number x the data type first computes a *separation bound* q such that $|x| \leq q$ implies $x = 0$ and then computes an approximation of x of sufficient precision to decide the sign of x. The user may assist the data type by providing a separation bound q. The data type also allows to evaluate real expressions with arbitrary precision. Figure 1 shows (part of) the LEDA manual page for **real**s.

reals provide exact computation in a convenient way. In an implementation of a geometric algorithm in C++, **real**s can be used like **double**s. The following example arises in the computation of Voronoi diagrams of line segments [2]. For i, $1 \leq i \leq 3$, let $l_i : a_i x + b_i y + c_i = 0$ be a line in two–dimensional space and let $p = (0,0)$ be the origin. The circles passing through p and touching l_1 and l_2 have centers $(x_v/z_v, y_v/z_v)$ where

$$\begin{aligned} x_v &= a_1c_2 + a_2c_1 \pm \sqrt{2c_1c_2(\sqrt{N} + D)} \\ y_v &= b_1c_2 + b_2c_1 \pm \mathrm{sign}(S)\sqrt{2c_1c_2(\sqrt{N} - D)} \\ z_v &= \sqrt{N} - a_1a_2 - b_1b_2 \end{aligned}$$

with $S = a_1b_2 + a_2b_1$, $N = (a_1^2 + b_1^2)(a_2^2 + b_2^2)$, and $D = a_1a_2 - b_1b_2$.

The following C++ procedure tests whether l_3 intersects, touches or misses one of these circles by comparing the distances of $(x_v/z_v, y_v/z_v)$ and p and $(x_v/z_v, y_v/z_v)$ and l_3, see [2].

```
int INCIRCLE(real a1, real b1, real c1, real a2, real b2,
             real c2, real a3, real b3, real c3, int pm)
{
  real RN = sqrt((a1 * a1 + b1 * b1) * (a2 * a2 + b2 * b2));
  real A = a1 * c2 + a2 * c1;
  real B = b1 * c2 + b2 * c1;
  real C = 2 * c1 * c2;
  real D = a1 * a2 - b1 * b2;
  real S = a1 * b2 + a2 * b1;
  real xv = A + pm * sqrt(C * (RN + D));
  real yv = B + pm * S.sign() * sqrt(C * (RN - D));
  real zv = RN - (a1 * a2 + b1 * b2);
  real P = a3 * xv + b3 * yv + c3 * zv;
  real R = P * P - (a3 * a3 + b3 * b3) * (xv * xv + yv * yv);
  return R.sign();
}
```

If a_i, b_i, c_i are known to be $2k$-bit integers the sign computation R.sign() can be replaced by R.sign$(48 * k)$, see [2]. Without this assistance the sign computation can be extremely slow.

The implementation of **real**s is based on LEDA data types **integer** and **bigfloat** which are arbitrary precision integers and floating point numbers with exponents

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. Supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

†Fachbereich 14, Informatik, Universität des Saarlandes, 66041 Saarbrücken, Germany.

‡Martin-Luther-Universität Halle, Fachbereich Mathematik und Informatik, 06099 Halle, Germany.

0

The arithmetic operations $+$, $-$, $*$, $/$, $+=$, $-=$, $*=$, $/=$, $-$(unary), the comparison operations $<$, $<=$, $>$, $>=$, $==$, $!=$ and the stream operations all are available.

int	x.sign()	returns -1 if (the exact value of) $x < 0$, 1 if $x > 0$, and 0 if $x = 0$.
int	x.sign(*integer* a)	as above. Precondition: if $\vert x\vert \leq 2^{-a}$ then $x = 0$.
void	x.improve(*integer* a)	(re-)computes the approximation of x such that its final quality is bounded by a, i.e., $x.get\_precision() \geq a$ after the call $x.improve(a)$.
void	x.compute_in(*long* k)	(re-)computes the approximation of x; each numerical operation is carried out with k binary places.
void	x.compute_up_to(*long* k)	(re-)computes an approximation of x such that the error of the approximation lies in the k-th binary place.
double	x.todouble()	returns the current double approximation of x.
double	x.get_double_error()	returns the quality of the current double approximation of x, i.e., $\vert x - x.todouble()\vert \leq x.get\_double\_error() * \vert x.todouble()\vert$.
integer	x.get_precision()	returns the quality of the current internal approximation $x.num$ of x, i.e., $\vert x - x.num\vert \leq 2^{-x.get\_precision()}$.
real	sqrt(*real* x)	squareroot.
real	fabs(*real* x)	absolute value of x.

Figure 1: (Part of) the LEDA manual page for **real**s

and mantissa of arbitrary length. Similar to [1, 6] a **real** is represented by the expression which defines it. Furthermore it has a **double** approximation $\hat{x}$ together with a relative error bound ϵ_x. In the sign computation a *floating-point filter* is used as suggested in [3]. When the double test is inconclusive the quality of the approximation is repeatedly doubled (using calls of improve) until a decision is possible. A decision is possible if either the absolute value of the approximation is less than its absolute error or the separation bound is reached. The latter bound is either provided by the user or computed as in [5, 6].

The INCIRCLE procedure has been tested with incircle tests (of this special type) that arised in a randomized incremental construction of Voronoi diagrams of line segments. For these realistic data (all were nondegenerate) the exact test with **real**s was about 20 times slower than the equivalent test with **double**s.

References

[1] M.O. Benouamer, P. Jaillon, D. Michelucci, and J.-M. Moreau. A "lazy" solution to imprecision in computational geometry. 5th CCCG, pp. 73–78, 1993.

[2] C. Burnikel, K. Mehlhorn, and S. Schirra. How to compute the Voronoi diagram of line segments: Theoretical and experimental results. ESA'94, pp. 227–239, 1994.

[3] S. Fortune and C. van Wyk. Efficient exact arithmetic for computational geometry. *Proc. of the 9th Symp. on Computational Geometry*, pages 163–171, 1993.

[4] K. Mehlhorn and S. Näher. LEDA: A platform for combinatorial and geometric computing. CACM, Jan. 1995.

[5] M. Mignotte. *Mathematics for Computer Algebra.* Springer Verlag, 1992.

[6] C.K. Yap and T. Dube. The exact computation paradigm. In *Computing in Euclidean Geometry II*, 1995.