Robustness in CGAL

Monique Teillaud

INRIA SOPHIA-ANTIPOLIS
Robustness issues

- Algorithms → explicit treatment of degenerate cases

  Symbolic perturbation for 3D dynamic Delaunay triangulations
  [Devillers Teillaud SODA’03]

- Kernel and arithmetics → Numerical robustness
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = sqrt( NT(2) );

Kernel::Point_2 p(0,0), q(sqrt2,sqrt2);
Kernel::Circle_2 C(p,2);

assert( C.has_on_boundary(q) );

OK if NT gives exact sqrt assertion violation otherwise
Orientation of 2D points

\[
\text{orientation}(p, q, r) = \text{sign} \left( \det \begin{bmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{bmatrix} \right) \\
= \text{sign} \left( (q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x) \right)
\]
\( p = (0.5 + x.u, 0.5 + y.u) \)
\( 0 \leq x, y < 256, \ u = 2^{-53} \)
\( q = (12, 12) \)
\( r = (24, 24) \)

\( \text{orientation}(p, q, r) \)
evaluated with double

\( 256 \times 256 \) pixel image

\[ > 0, = 0, < 0 \]

\( \rightarrow \text{inconsistencies} \) in predicate evaluations

[Kettner, Mehlhorn, Pion, Schirra, Yap, ESA’04]
Predicates and Constructions

Input

Predicates

< 0 = 0 > 0

Combinatorial Structure

Constructions

Geometric embedding

Robustness in
Delaunay triangulation only predicates are used
orientation, in_sphere

Voronoi diagram constructions are needed
circumcenter
Arithmetic filters
Numerical Robustness in

imprecise numerical evaluations $\rightarrow$ non-robustness

combinatorial result

Exact Geometric Computation

\[ \neq \]

exact arithmetics
Most expected cases: easy, to be optimized first

Control rounding errors of floating point computation
⇒ exact computation, expensive, not often used

In good cases, exact geometric computation
but cost ≃ cost of floating point computation.
Filtering Predicates

\[ \text{sign} \left( P(x) \right) \]

Approximate evaluation \( P^a(x) \) + Error \( \varepsilon \)

\[ |P^a(x)| > \varepsilon \]

\[ \text{sign} \left( P(x) \right) = \text{sign} \left( P^a(x) \right) \quad \text{Exact computation} \]
Dynamic filters: interval arithmetic

Floating point operation replaced by

operations on **intervals** of floating point values $[x; \bar{x}]$

encoding rounding errors.

**Inclusion property:**
at each operation, the interval contains the exact value of $X$. 
Operations on intervals

Rounding modes IEEE 754

Addition / substraction

\[
X + Y \rightarrow [x+y; \overline{x+y}]
\]

\[
X - Y \rightarrow [x-y; \overline{x-y}]
\]

Optimization:

\[
X + Y \rightarrow [-((-x)\overline{y}); \overline{x+y}]
\]

(fewer changes of rounding modes)
Operations on intervals

**Multiplication** :

\[ X \times Y \longrightarrow \left[ \min(x \times y, \ x \times \bar{y}, \ \bar{x} \times y, \ \bar{x} \times \bar{y}); \ max(x \times y, \ x \times \bar{y}, \ \bar{x} \times y, \ \bar{x} \times \bar{y}) \right] \]

In practice: comparisons for different cases before performing multiplications.

**Division** : similar

Handling of division by 0.
Comparisons

Inclusion property

if

\[ [x; x] \cap [y; y] = \emptyset \]

then

we can decide whether \( X < Y \) or \( X > Y \)

else

we cannot decide.

\[ \implies \text{Filter failure} \]
**Static filters**

**Static analysis** of error propagation on evaluation of a polynomial expression, assuming **bounds on the input data**.

$x$ being a positive floating point value, and $y$ the smallest floating point value greater than $x$

$$\text{ulp}(x) = y - x$$

(Unit in the Last Place).

Remark 1: $\text{ulp}(x)$ is a power of 2 (or $\infty$).
Remark 2: In normal cases: $\text{ulp}(x) \simeq x.2^{-53}$
$x$ real, $x$ value computed in double, $e_x$ and $b_x$ doubles such that

\[
\begin{cases}
  e_x \geq |x - x| \\
  b_x \geq |x|
\end{cases}
\]

Initially, value rounded to closest (if values cannot be represented by a double)

\[
\begin{cases}
  b_x = |x| \\
  e_x = \frac{1}{2} \text{ulp}(x)
\end{cases}
\]

For $+, -, \times, \div, \sqrt{}$, rounding error on result $r$ smaller than

- $\frac{1}{2} \text{ulp}(r)$ for rounding to nearest mode
- $\text{ulp}(r)$ otherwise.
Addition and substraction

Propagation of error on an addition $z = x + y$:

$$
\begin{align*}
\begin{cases}
b_z &= b_x + b_y \\
e_z &= e_x + e_y + \frac{1}{2}\text{ulp}(z)
\end{cases}
\end{align*}
$$

Indeed:

$$
|z - z| = \left| (z - (x + y)) + ((x + y) - (x + y)) + ((x + y) - z) \right|
= \underbrace{0}_{=0} + \underbrace{e_x + e_y}_{\leq e_x + e_y} + \underbrace{\frac{1}{2}\text{ulp}(z)}_{\leq \frac{1}{2}\text{ulp}(z)}
\leq e_x + e_y + \frac{1}{2}\text{ulp}(z)
$$
Multiplication

Propagation of error on a multiplication \( z = x \times y \):

\[
\begin{align*}
\{ 
\text{b}_z &= \text{b}_x \times \text{b}_y \\
\text{e}_z &= \text{e}_x \times \text{e}_y + \text{e}_x \times |x| + \text{e}_y \times |y| + \frac{1}{2} \text{ulp}(z)
\}
\end{align*}
\]

Indeed:

\[
\begin{align*}
|z - \overline{z}| &= \left| \left( z - (x \times y) \right) + \left( (x \times y) - (x \times y) \right) \right| \\
&= 0 + \left( x - x \right) \times \left( y - y \right) - \left( x - x \right) \times y - \left( y - y \right) \times x \\
&\leq \frac{1}{2} \text{ulp}(z) \\
&\leq \text{e}_x \times \text{e}_y + \text{e}_x \times y + \text{e}_y \times x + \frac{1}{2} \text{ulp}(z)
\end{align*}
\]
Application: *orientation* predicate

Approximate non guaranteed version

```c
int orientation(double px, double py,
                double qx, double qy,
                double rx, double ry)
{
    double pqx = qx - px, ppy = qy - py;
    double prx = rx - px, pry = ry - py;

    double det = pqx * pry - pqy * prx;

    if (det > 0) return 1;
    if (det < 0) return -1;
    return 0;
}
```
Application: orientation predicate

Code with static filtering (for entries bounded by 1):

```cpp
int filtered_orientation(double px, double py,
                          double qx, double qy,
                          double rx, double ry)
{
    double pqx = qx - px, pqy = qy - py;
    double prx = rx - px, pry = ry - py;

    double det = pqx * pry - pqy * prx;

    const double E = 1.33292e-15;
    if (det > E) return 1;
    if (det < -E) return -1;

    ... // can’t decide => call the exact version
}
```
int filtered_orientation(double px, double py,
    double qx, double qy,
    double rx, double ry)
{
    double b = max_abs(px, py, qx, qy, rx, ry);

double pqx = qx - px, pqy = qy - py;
double prx = rx - px, pry = ry - py;

double det = pqx * pry - pqy * prx;

    const double E = 1.33292e-15;

    if (det >  E*b*b) return 1;
    if (det < -E*b*b) return -1;

    ... // can’t decide => call the exact version
}
Theoretical study: [Devillers-Preparata-99]

Input data \textit{uniformly distributed} in a unit square/cube

static filtering

<table>
<thead>
<tr>
<th>Orientation</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>orientation 2D</td>
<td>$10^{-15}$</td>
</tr>
<tr>
<td>orientation 3D</td>
<td>$5.10^{-14}$</td>
</tr>
<tr>
<td>in_circle 2D</td>
<td>$10^{-11}$</td>
</tr>
<tr>
<td>in_sphere 3D</td>
<td>$7.10^{-10}$</td>
</tr>
</tbody>
</table>
More degenerate cases

<table>
<thead>
<tr>
<th></th>
<th>Dynamic</th>
<th>Semi-static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0</td>
<td>870</td>
</tr>
<tr>
<td>$\varepsilon = 2^{-5}$</td>
<td>0</td>
<td>1942</td>
</tr>
<tr>
<td>$\varepsilon = 2^{-10}$</td>
<td>0</td>
<td>662</td>
</tr>
<tr>
<td>$\varepsilon = 2^{-15}$</td>
<td>0</td>
<td>8833</td>
</tr>
<tr>
<td>$\varepsilon = 2^{-20}$</td>
<td>0</td>
<td>132153</td>
</tr>
<tr>
<td>$\varepsilon = 2^{-25}$</td>
<td>10</td>
<td>192011</td>
</tr>
<tr>
<td>$\varepsilon = 2^{-30}$</td>
<td>19536</td>
<td>308522</td>
</tr>
<tr>
<td>Grid</td>
<td>49756</td>
<td>299505</td>
</tr>
</tbody>
</table>

Number of filter failures for dynamic and static filters during the computation of a Delaunay triangulation on $10^5$ points).

Data on an integer grid with precision of 30 bits, with relative perturbation.
Comparaison : dynamic vs static filters

static filtering

- **fails more often** than more precise interval arithmetic filtering
- **faster**
- **harder to write**: needs analysis of each predicate.

Fastest method: **Cascading filters**
Implementation in GOAL
**Arithmetic tools**

- **Multiprecision integers**
  Exact evaluation of signs / values of polynomial expressions with integer coefficients
  `CGAL::MP_Float, GMP::mpz_t, LEDA::integer, ...`

- **Multiprecision floats**
  idem, with float coefficients \((n2^m, n, m \in \mathbb{Z})\)
  `CGAL::MP_Float, GMP::mpf_t, LEDA::bigfloat, ...`

- **Multiprecision rationals**
  Exact evaluation of signs / values of rational expressions
  `CGAL::Quotient< · >, GMP::mpq_t, LEDA::rational, ...`

- **Algebraic numbers**
  Exact comparison of roots of polynomials
  `LEDA::real, Core::Expr` (work in progress in CGAL)
Dynamic filtering

Number types: CGAL::Interval_nt, MPFR/MPFI, boost::interval

CGAL::Filtered_kernel $\langle K \rangle$ kernel wrapper

Replaces predicates of $K$ by filtered and exact predicates. (exact predicates computed with MP_Float)

Static + Dynamic filtering in CGAL 3.1

$\rightarrow$ more generic generator also available for user’s predicates

CGAL::Filtered_predicate
Number type \texttt{CGAL::Lazy\_exact\_nt} \textless \texttt{Exact\_NT} \textgreater

Delays exact evaluation with \texttt{Exact\_NT}:

- stores a \texttt{DAG} of the expression
- computes first an approximation with \texttt{Interval\_nt}
- allows to control the relative precision of \texttt{to\_double}

\texttt{CGAL::Lazy\_kernel} in CGAL 3.2
Predefined kernels

**Exact_predicates_exact_constructions_kernel**
Filtered_kernel< Cartesian< Lazy_exact_nt< Quotient< MP_Float >>>

**Exact_predicates_exact_constructions_kernel_with_sqrt**
Filtered_kernel< Cartesian< Core::Expr >>

**Exact_predicates_inexact_constructions_kernel**
Filtered_kernel< Cartesian< double >>>

Robustness in
Efficiency

3D Delaunay triangulation
CGAL-3.1-I-124

1,000,000 random points
Simple_CARTESIAN< double > 48.1 sec
Simple_CARTESIAN< MP_Float > 2980.2 sec
Filtered_kernel (dynamic filtering) 232.1 sec
Filtered_kernel (static + dynamic filtering) 58.4 sec

49.787 points (Dassault Systèmes)
double exact and filtered loop ! < 8 sec
• **Automatric generation of code** from a generic version

• filtering of *constructions*

• **Rounding** of constructions

• **Curved objects** (algebraic methods)