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**

Numerical robustness issues

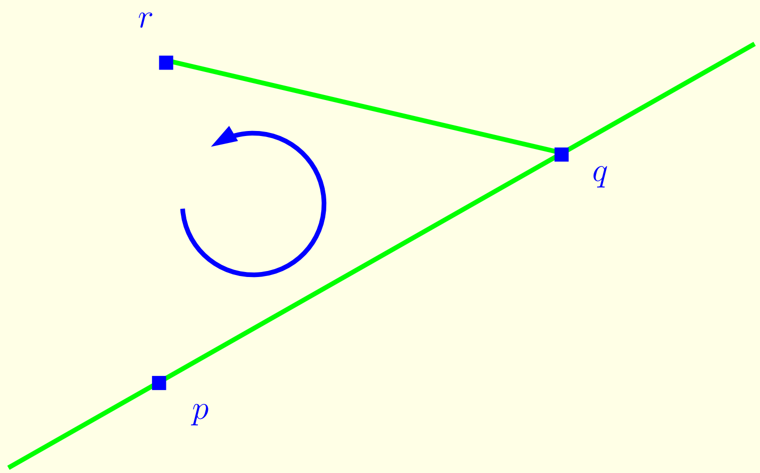
```
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = sqrt( NT(2) );

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



$$orientation(p,q,r) = \mathbf{sign}\left(\det\begin{bmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{bmatrix}\right)$$

$$= \mathbf{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

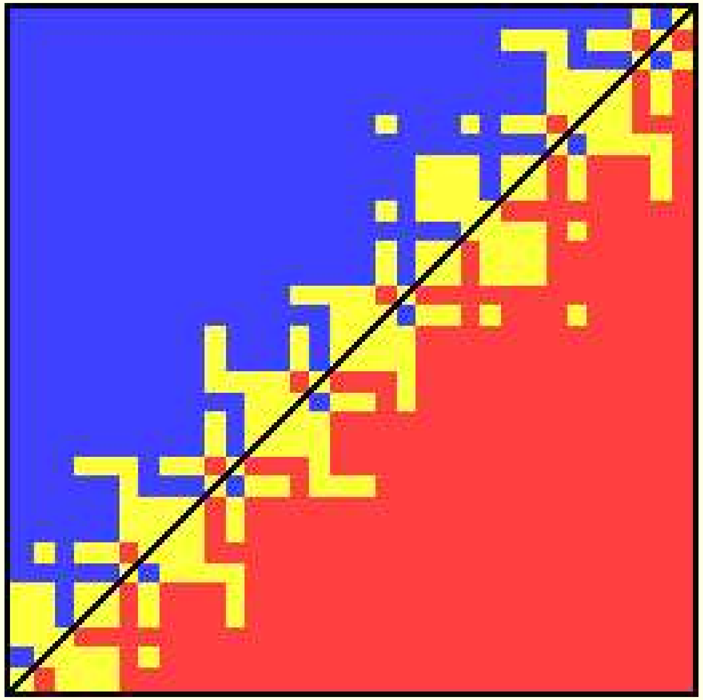
$p = (0.5 + x.u,\ 0.5 + y.u)$
$0 \leq\ x, y\ < 256,\ \ u = 2^{-53}$
$q = (12, 12)$
$r = (24, 24)$

$orientation(p, q, r)$

evaluated with `double`

256 x 256 pixel image

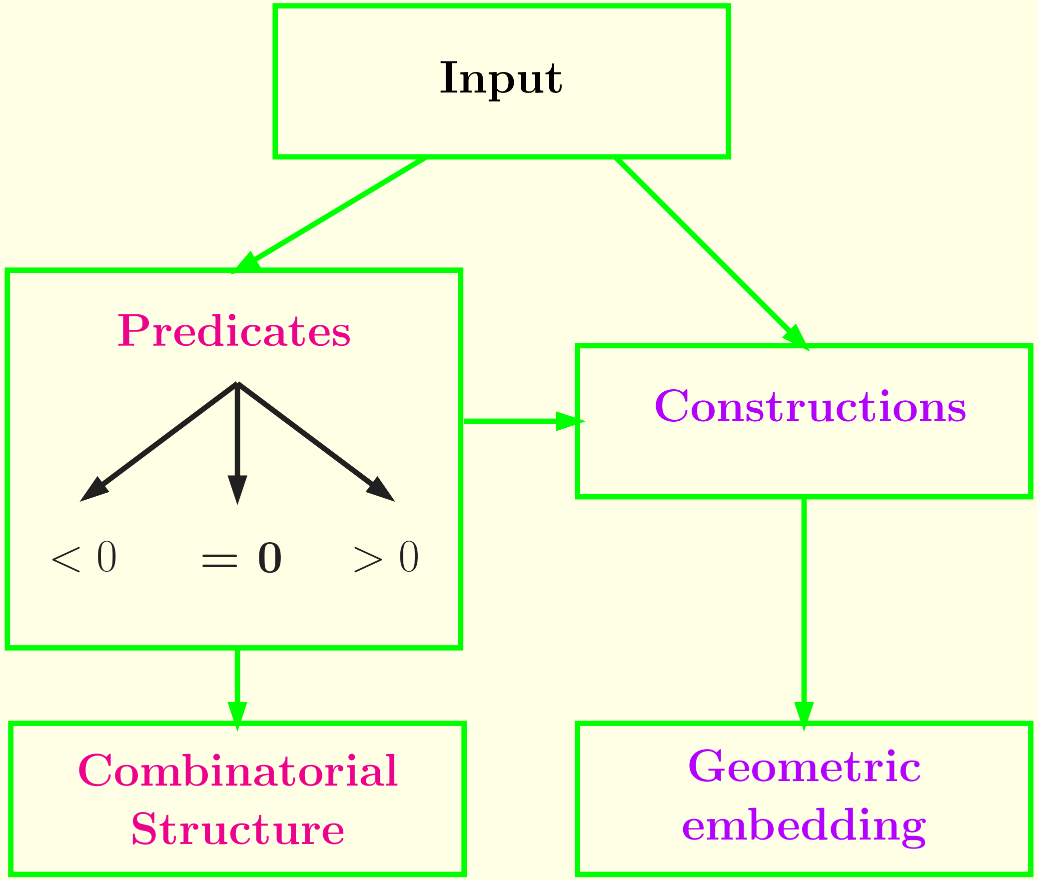
> 0, $= 0$, < 0



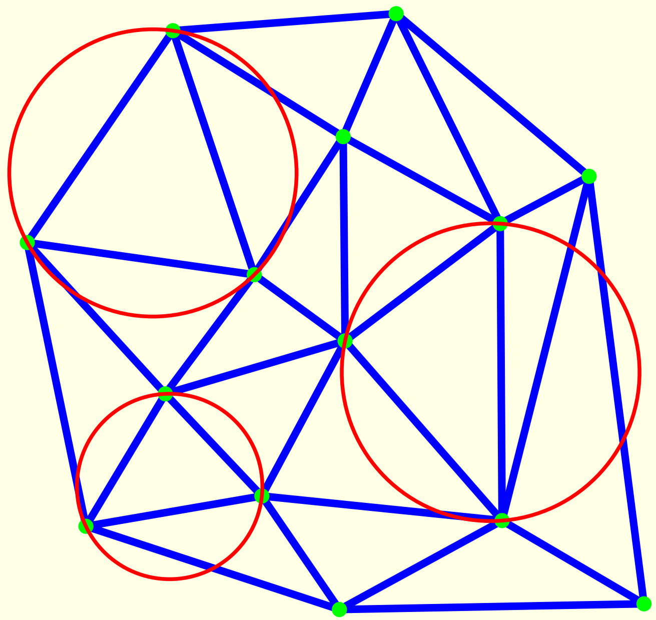
$\longrightarrow$ **inconsistencies** in predicate evaluations

`[Kettner, Mehlhorn, Pion, Schirra, Yap, ESA'04]`

Predicates and Constructions

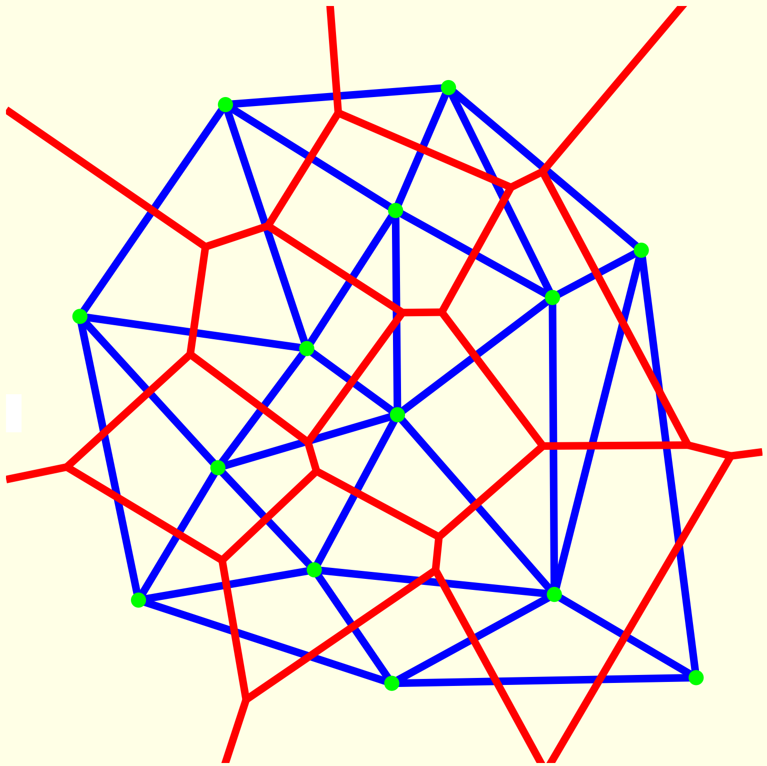


Delaunay triangulation



only **predicates** are used
orientation, in_sphere

Voronoi diagram



constructions are needed
circumcenter

Arithmetic filters

Numerical Robustness in CGAL

imprecise numerical evaluations

$\longrightarrow$ non-robustness

combinatorial result

Exact Geometric Computation

$\neq$

exact arithmetics

Optimize easy cases

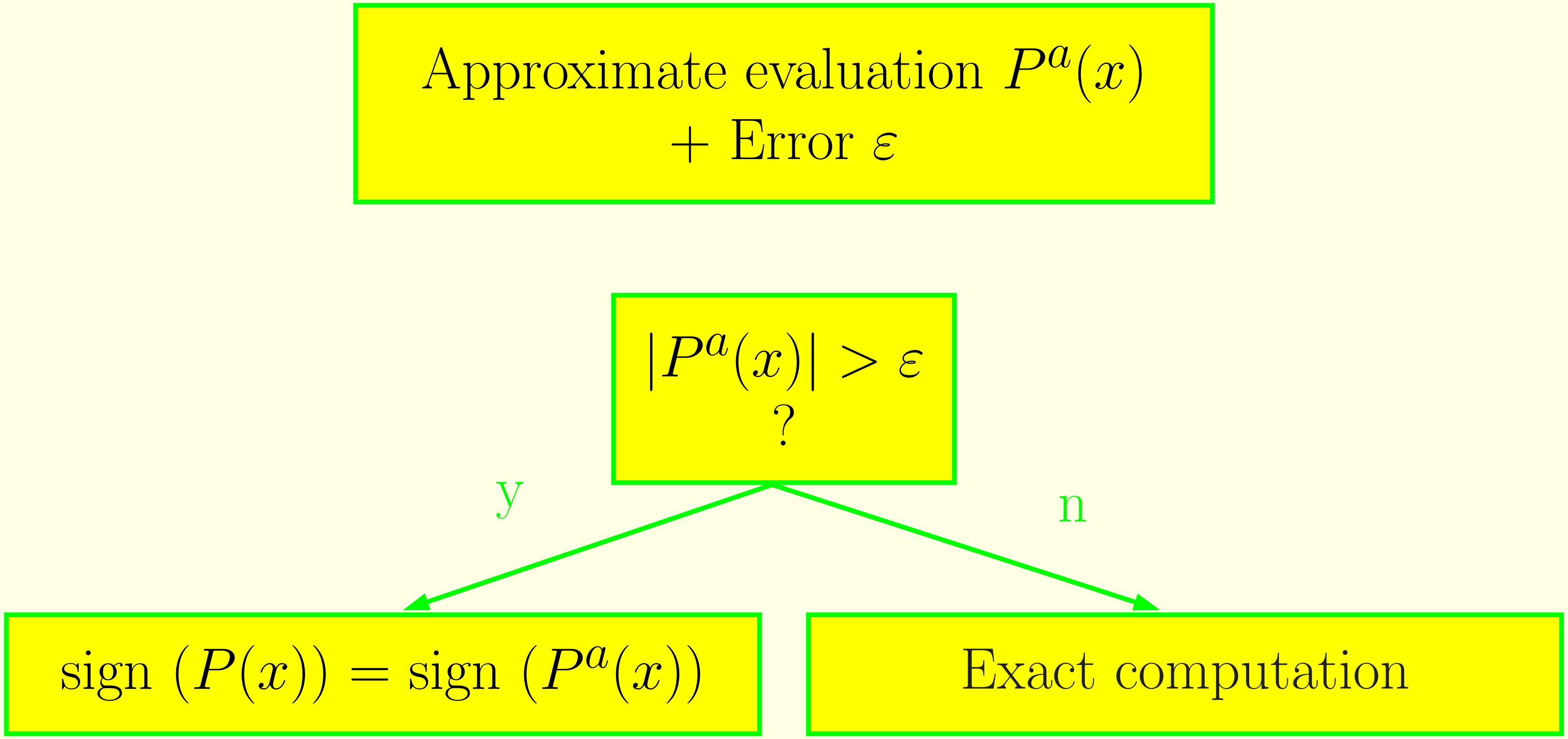
Most expected cases: easy, to be optimized first

Control rounding errors of floating point computation
$\Rightarrow$ exact computation, expensive, not often used

In good cases, **exact geometric computation**
but cost $\simeq$ cost of floating point computation.

Filtering Predicates

sign $(P(x))$ **?**



Dynamic filters: interval arithmetic

Floating point operation replaced by

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

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

$$X + Y \longrightarrow [\underline{x}\underline{+}\underline{y}; \overline{x}\overline{+}\overline{y}]$$

$$X - Y \longrightarrow [\underline{x}\underline{-}\overline{y}; \overline{x}\overline{-}\underline{y}]$$

Optimization:

$$X + Y \longrightarrow [-((-\underline{x})\overline{-}\underline{y}); \overline{x}\overline{+}\overline{y}]$$

(fewer changes of rounding modes)

Operations on intervals

Multiplication :

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

In practice: comparisions for different cases before performing multiplications.

Division : similar

Handling of division by 0.

Comparisons

Inclusion property

if

$$[\underline{x};\overline{x}] \cap [\underline{y};\overline{y}] = \emptyset$$

then

we can decide whether $X < Y$ or $X > Y$

else

we cannot decide.

$\Longrightarrow$ **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

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

(Unit in the Last Place).

Remark 1 : $\texttt{ulp}(x)$ is a power of 2 (or ∞).
Remark 2 : In normal cases : $\texttt{ulp}(x) \simeq x.2^{-53}$

x real, $\mathtt{x}$ value computed in double,
$\mathtt{e_x}$ and $\mathtt{b_x}$ doubles such that

$$\begin{cases} \mathtt{e_x} \geq |x - \mathtt{x}| \\ \mathtt{b_x} \geq |\mathtt{x}| \end{cases}$$

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

$$\begin{cases} \mathtt{b_x} = |\mathtt{x}| \\ \mathtt{e_x} = \frac{1}{2}\mathtt{ulp}(\mathtt{x}) \end{cases}$$

For $+, -, \times, \div, \sqrt{}$, rounding error on result $\mathtt{r}$ smaller than
- $\frac{1}{2}\mathtt{ulp}(\mathtt{r})$ for rounding to nearest mode
- $\mathtt{ulp}(\mathtt{r})$ otherwise.

Addition and substraction

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

$$\begin{cases} \mathtt{b_z} = \mathtt{b_x} + \mathtt{b_y} \\ \mathtt{e_z} = \mathtt{e_x} \overline{+} \mathtt{e_y} \overline{+} \frac{1}{2}\mathtt{ulp(z)} \end{cases}$$

Indeed:

$$\begin{aligned} |z - \mathtt{z}| &= |\underbrace{(z - (x+y))}_{=0} + \underbrace{((x+y) - (\mathtt{x}+\mathtt{y}))}_{\leq \mathtt{e_x}+\mathtt{e_y}} + \underbrace{((\mathtt{x}+\mathtt{y}) - \mathtt{z})}_{\leq \frac{1}{2}\mathtt{ulp(z)}}| \\ &\leq \mathtt{e_x} \overline{+} \mathtt{e_y} \overline{+} \frac{1}{2}\mathtt{ulp(z)} \end{aligned}$$

Multiplication

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

$$\begin{cases} \mathtt{b_z} = \mathtt{b_x} \times \mathtt{b_y} \\ \mathtt{e_z} = \mathtt{e_x} \overline{\times} \mathtt{e_y} \ \overline{+} \ \mathtt{e_y} \overline{\times} |\mathtt{x}| \ \overline{+} \ \mathtt{e_x} \overline{\times} |\mathtt{y}| \ \overline{+} \ \frac{1}{2}\mathtt{ulp}(\mathtt{z}) \end{cases}$$

Indeed:

$$\begin{aligned} |z - \mathtt{z}| &= |\underbrace{(z - (x \times y))}_{=0} + \underbrace{((x \times y) - (\mathtt{x} \times \mathtt{y}))}_{=(\mathtt{x}-x)(\mathtt{y}-y)-(\mathtt{x}-x)\times\mathtt{y}-(\mathtt{y}-y)\times\mathtt{x}} + \underbrace{((\mathtt{x} \times \mathtt{y}) - \mathtt{z})}_{\leq\frac{1}{2}\mathtt{ulp}(\mathtt{z})}| \\ &\leq \mathtt{e_x} \overline{\times} \mathtt{e_y} \ \overline{+} \ \mathtt{e_x} \overline{\times} \mathtt{y} \ \overline{+} \ \mathtt{e_y} \overline{\times} \mathtt{x} \ \overline{+} \ \frac{1}{2}\mathtt{ulp}(\mathtt{z}) \end{aligned}$$

Application: *orientation* predicate

Approximate non guaranteed version

```
int 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;

  if (det > 0)  return  1;
  if (det < 0)  return -1;
  return 0;
}
```

Application: *orientation* predicate

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

```
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
}
```

Variants - Ex : compute the bound at running time

```
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
}
```

Probability of filter failures

Theoretical study: [Devillers-Preparata-99]

Input data **uniformly distributed** in a unit square/cube

static filtering

`orientation 2D`	10^{-15}
`orientation 3D`	5.10^{-14}
`in_circle 2D`	10^{-11}
`in_sphere 3D`	7.10^{-10}

More degenerate cases

	Dynamic	Semi-static
Random	0	870
$\varepsilon = 2^{-5}$	0	1942
$\varepsilon = 2^{-10}$	0	662
$\varepsilon = 2^{-15}$	0	8833
$\varepsilon = 2^{-20}$	0	132153
$\varepsilon = 2^{-25}$	10	192011
$\varepsilon = 2^{-30}$	19536	308522
Grid	49756	299505

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 CGAL

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$< \cdot >$, 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 $<$ **K** $>$ kernel wrapper [Pion]

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

Static + Dynamic filtering in CGAL 3.1

$\longrightarrow$ more generic generator also available for user's predicates
CGAL::Filtered_predicate

Filtering Constructions

Number type **CGAL::Lazy_exact_nt** $<$ **Exact_NT** $>$ `[Pion]`

Delays exact evaluation with **Exact_NT**:

- stores a **DAG** of the expression
- computes first an approximation with **Interval_nt**
- allows to control the relative precision of `to_double`

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 >>

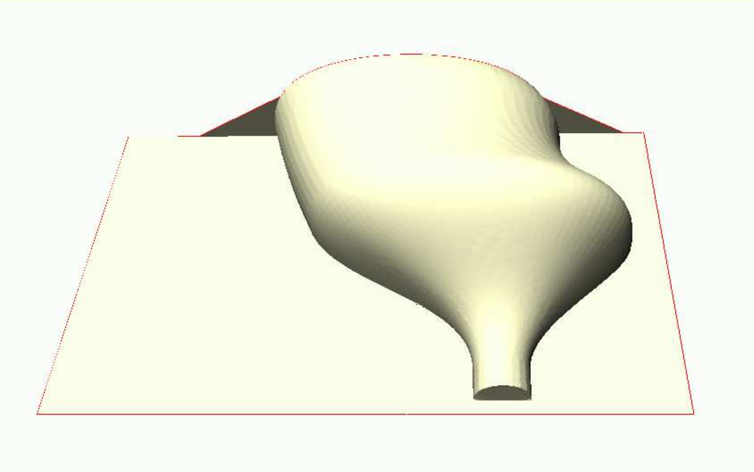
Efficiency

3D Delaunay triangulation
CGAL-3.1-I-124

Pentium-M 1.7 GHz, 1GB
g++ 3.3.2, -O2 -DNDEBUG

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	**loop !**
exact and filtered	< 8 sec

Work in progress

- **Automatric generation of code** from a generic version
- filtering of **constructions**
- **Rounding** of constructions
- **Curved objects** (algebraic methods)