Geometric predicates for curves

Sylvain Pion
Monique Teillaud

Introduction - examples

Algebraic expression of the predicates

- A first approach
- Polynomial approach

Number types

Filtering techniques

Some experimental results

Conclusion

Introduction

Implementing geometric algorithms is difficult

Decisions depend on geometric questions

predicates

Evaluation of continuous functions

rounding errors

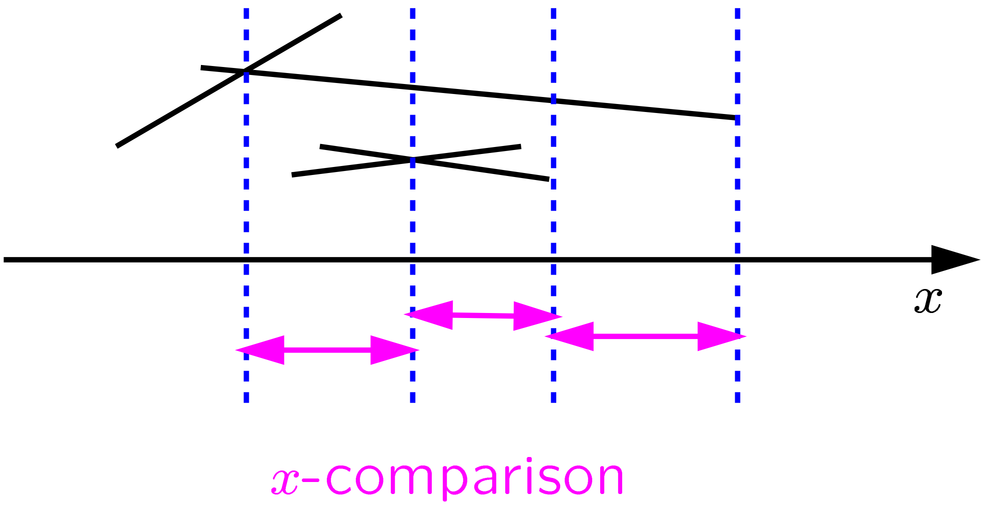
Combinatorial algorithms

Erroneous numerical computations

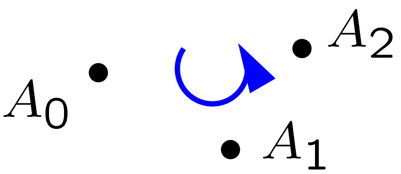
$\longrightarrow$ crash

Example : plane sweep

Intersections of line segments

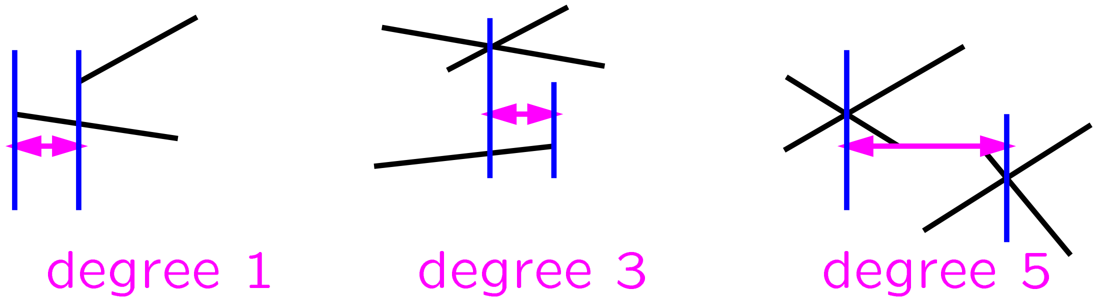


tests boil down to some orientation tests

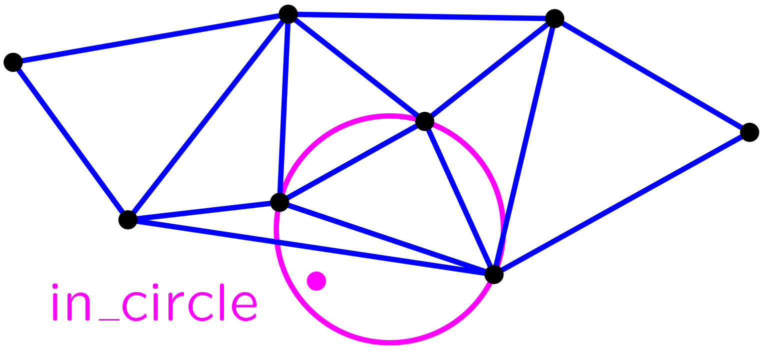


$$\text{orient}(A_0, A_1, A_2) = sign \begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{vmatrix}$$

data = coordinates of endpoints



Example : Delaunay triangulation



in_circle$(A_0, A_1, A_2, A_3) =$

$$sign \begin{vmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ x_0^2 + y_0^2 & x_1^2 + y_1^2 & x_2^2 + y_2^2 & x_3^2 + y_3^2 \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

Two directions

Work on the algorithms

Algorithms supporting inconsistencies

Algorithms using low degree predicates

Work on the predicates

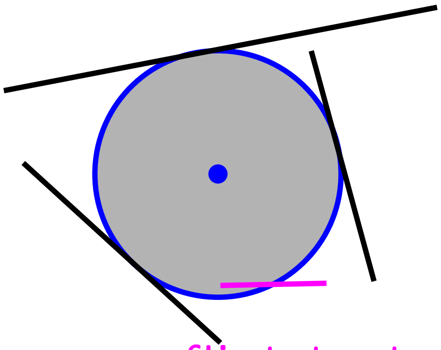
Exact predicates

Efficient predicates

Algebraic Expression

A first approach

Voronoi diagram of line segments



[Burnikel 96]

- Compute the center and squared radius
- Compute the squared distance from the center to the segment
- Compare

Implementation

summer internship, Christophe Delage

to be integrated into CGAL

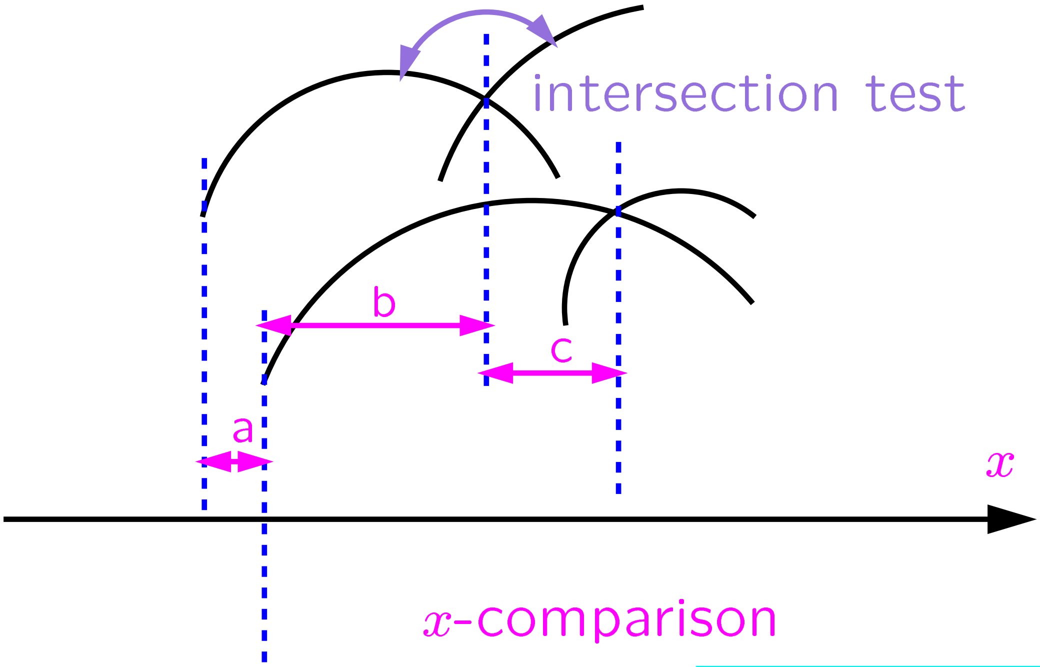
$\surd \longrightarrow$ exactness...

Algebraic Expression

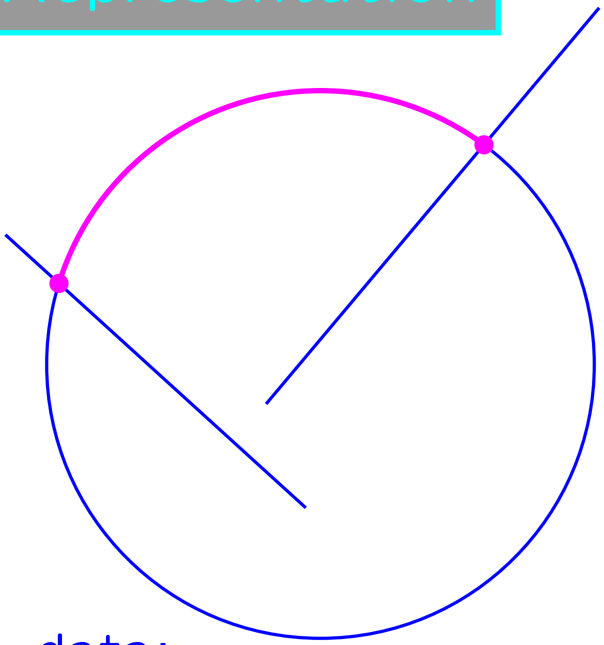
Polynomial approach

Arrangement of circle arcs

post-doc FIABLE, Alexandra Fronville [DFMT]



Representation



circle

center (α, β)

squared radius r

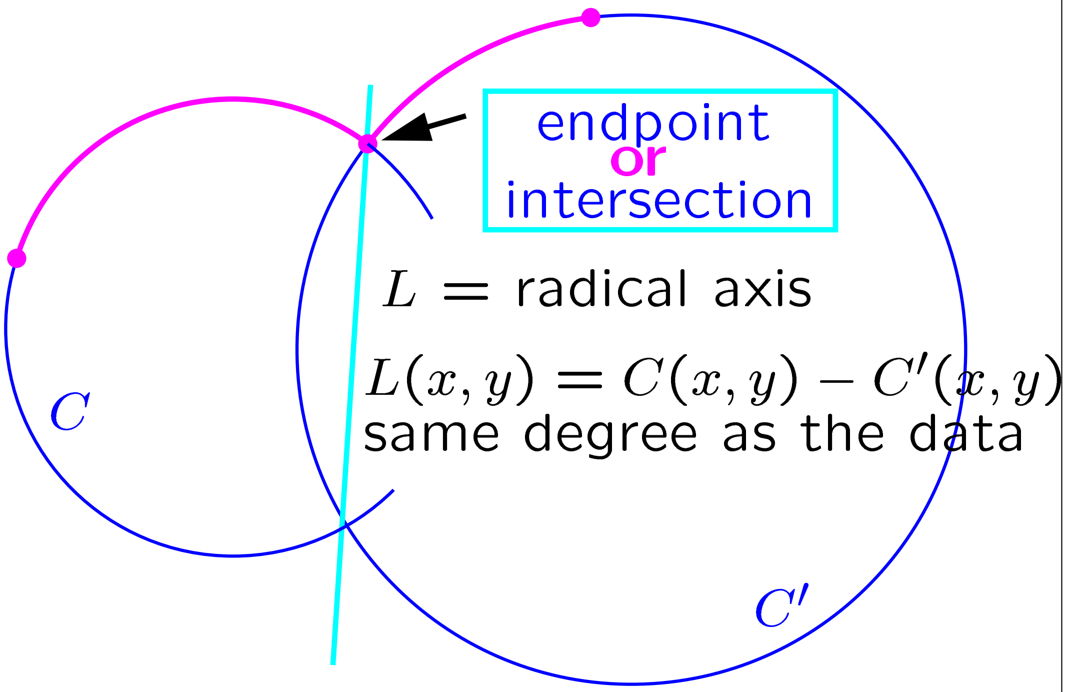
line

$px + qy + s = 0$

data:

α, β, p, q degree 1

r, s degree 2



L = radical axis

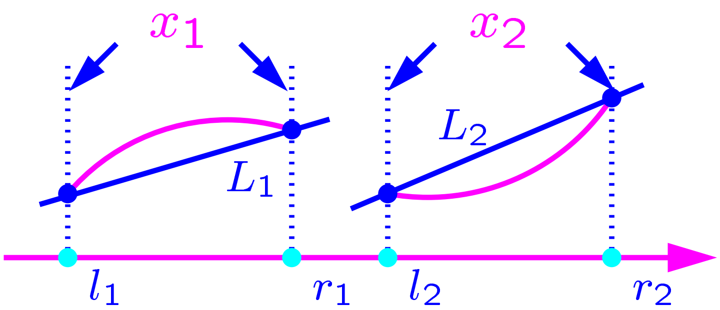
$L(x,y) = C(x,y) - C'(x,y)$
same degree as the data

$\longrightarrow$ Same representation for

output arcs
input arcs

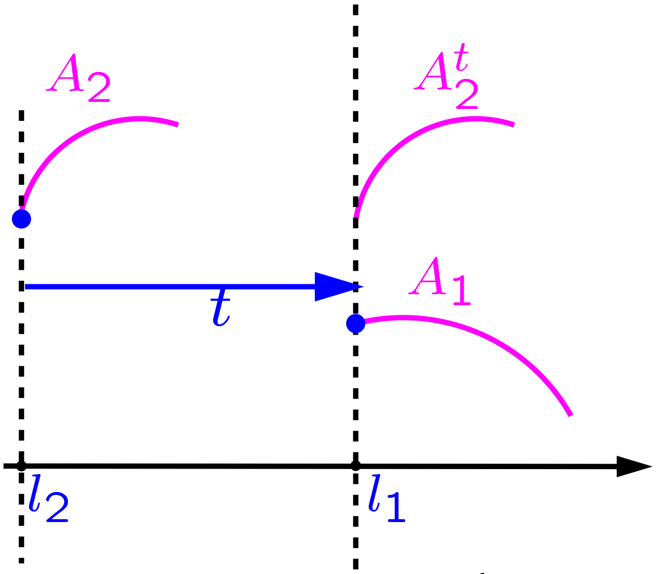
x-comparison

$x_i = L_i \cap C_i = l_i$ or r_i



First approach

- Compute x_1 and x_2
- Compare



translate A_2 by t

so that abscissae coincide

Sign of t such that $\begin{cases} C_1(x+t,y)=0 \\ L_1(x+t,y)=0 \\ C_2(x,z)=0 \\ L_2(x,z)=0 \end{cases}$ has a solution x,y,z

Bezoutian resultant

$$P(t) = P_0 t^4 + P_1 t^3 + P_2 t^2 + P_3 t + P_4$$

P_i polynomials in

$$\alpha_1, \beta_1, r_1, p_1, q_1, s_1$$
$$\alpha_2, \beta_2, r_2, p_2, q_2, s_2$$

degrees 8 to 12

659 monomials. . .

$\longrightarrow$ sign of a root t ?

Use of invariants

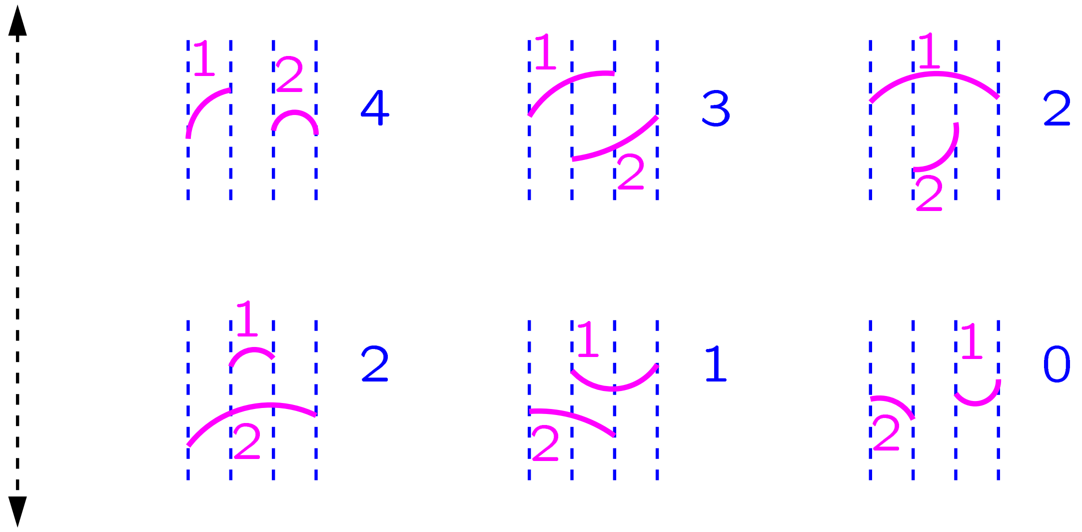
$\longrightarrow$ much simpler expressions

for the coefficients $P_0, \ldots, P_4$

$\longrightarrow$ sign of a root t ?

Geometric Predicates

Number of positive roots (translations) of P



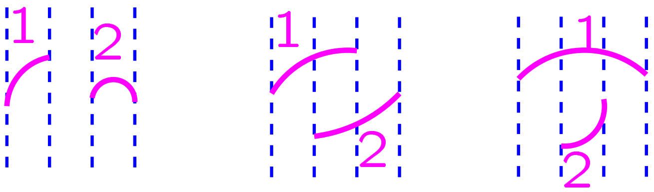
Configuration

Strategy of evaluation

- Determination of the configuration

  not always necessary
  to compare 2 abscissae

$l_1 < l_2$



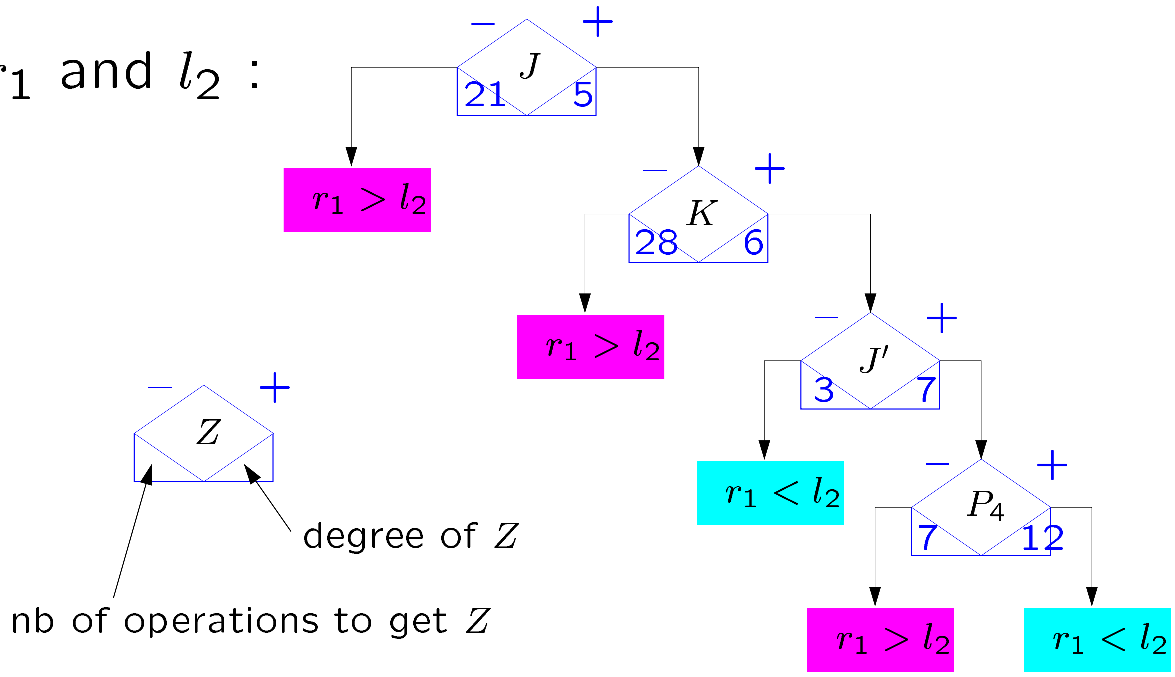
- Descartes rule

  \# positive roots
  $=$
  \# sign changes in $P_0, \ldots, P_4$

  $\longrightarrow$ sign of the P_i's ?

signs of polynomials of degrees 5,6,. . . , at most 12

r_1 and l_2 :



Implementation

summer internship, Vaibhav Gupta

to be integrated into CGAL

Number Types

C++, generic programming

```
template < class T >
bool f ( T a, T b, T c )
{
    T d = a+b*c;
    if ( d > 0 ) return true;
    else return false;
}
```

double

fast, non-exact

multiprecision integers (rationals)

slow, exact on +, -, x (/)
ex: GMP mpz (mpq)
LEDA integer (rational)

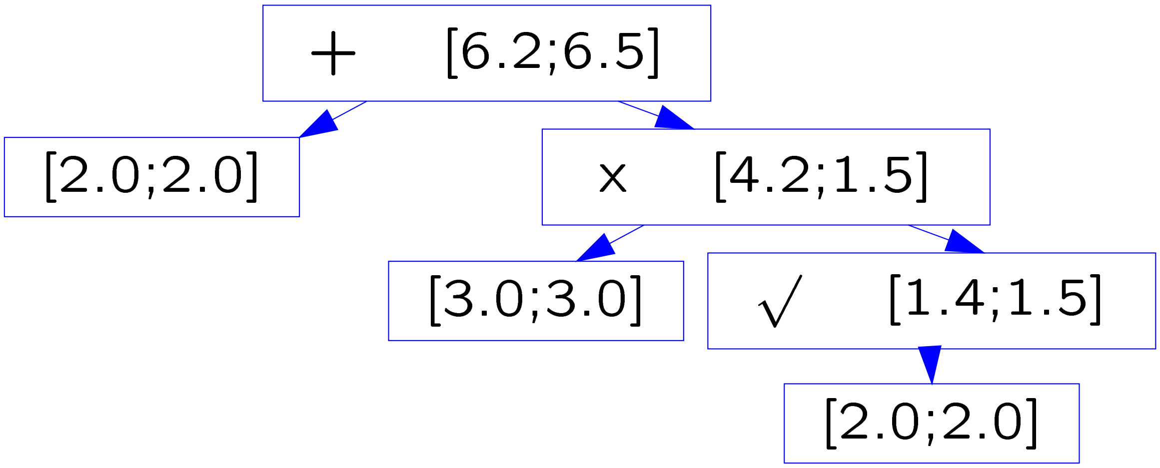
number types supporting roots

faster than multiprecision
separation bounds, filters
ex: LEDA real, Core

Lazy exact evaluation

Expression = Directed Acyclic Graph

$2 + 3.\sqrt{2} =$



sign computation : separation bounds

Arithmetic Filtering

Multi-precision number types are slow
Exact computation needed in 1% cases

$\longrightarrow$ Do a fast approximate computation
with certificate

if sign is certified
$\longrightarrow$ done
if not
$\longrightarrow$ use a more precise
and more expensive scheme

Static Filters

Bound on the input $\Longrightarrow$ error bound

Static filter : constant global bounds

Semi-static filter :

tighter bound computed at each call

Tools to compute the error bounds

Do not support division

Dynamic Filter

Control round-off errors

of each operation, at run time

Interval arithmetic

Real variable $x \in [\underline{x}; \overline{x}]$

Inclusion property

$x + y \in [\underline{x \underline{+} y}; \overline{x \overline{+} y}]$

Sign :

$$\begin{aligned} \underline{x} > 0 &\implies x > 0 \\ \overline{x} < 0 &\implies x < 0 \\ \underline{x} = \overline{x} = 0 &\implies x = 0 \end{aligned}$$

otherwise : unknown

Filtered predicate

```
bool dynamic_filter_f ( double a, b, c )
{
    try {
        return f<Interval_nt>(a,b,c);
    }
    catch (...) {
        return f<MP_Float>(a,b,c);
    }
}
```

First

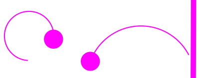
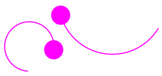
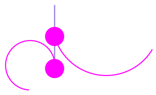
Experimental Results

Summary of arithmetic tools

	-,+,x	/	$\sqrt{}$
Filters			
static	•		•
semi-static	•		•
dynamic (interval)	•	•	•
Number types			
double	×	×	×
GMPZ, CGAL::MP_Float	•		
Quotient<GMPZ>	•	•	
LEDA real, Core	•	•	•

Geometric Predicates

circle arcs

μs		first Interval + L_real	polynomial static +semi-static +Interval +GMP	polynomial static +semi-static +Interv. first +GMP
left right		2.48	0.36	0.36
		67	24.3	6.8
		2170	129	128

with LEDA 4.2

Conclusion

Ongoing work

More implementations in CGAL

Other curves (conic arcs,. . .)

Voronoi diagrams of circles

Future work

Other predicates

- Visibility graphs
- Surfaces
- . . .