Robustesse Numérique en Géométrie Algorithmique, et application pratique dans CGAL

Sylvain Pion

Projet GEOMETRICA
INRIA Sophia Antipolis

1er Juin 2006

Plan

Plan

Géométrie Algorithmique

- Domaine de recherche actif ces 20-30 dernières années
- Algorithmes manipulant des objets géométriques en grand nombre
- Emphase sur la complexité asymptotique (modèle Real-RAM)

Domaines d'applications : CAO, SIG, biologie moléculaire, médical...

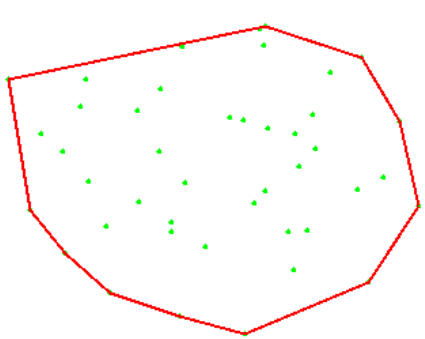
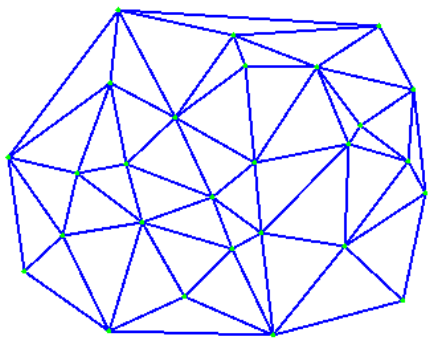
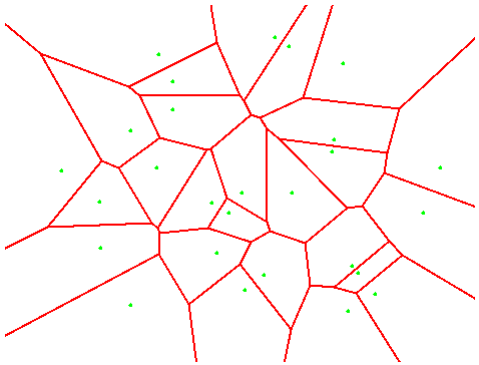
Géométrie Algorithmique

- Domaine de recherche actif ces 20-30 dernières années
- Algorithmes manipulant des objets géométriques en grand nombre
- Emphase sur la complexité asymptotique (modèle Real-RAM)

Domaines d'applications : CAO, SIG, biologie moléculaire, médical...

Exemples

- Enveloppes convexes, triangulations, diagrammes de Voronoï

- Reconstruction de surfaces, maillages
- Opérations booléennes sur polygones, arrangements
- Optimisation géométrique
- ...

CGAL : *Computational Geometry Algorithms Library*

Depuis 1995 : mise en oeuvre des algorithmes de la Géométrie Algorithmique.

- Critères : adaptabilité, efficacité, robustesse
- Review des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.2 : 100 modules, 500.000 lignes, 10.000 téléchargements/an
- Open Source : LGPL et QPL (commercialisé par GeometryFactory depuis 2003)

CGAL : *Computational Geometry Algorithms Library*

Depuis 1995 : mise en oeuvre des algorithmes de la Géométrie Algorithmique.

- Critères : adaptabilité, efficacité, robustesse
- Review des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.2 : 100 modules, 500.000 lignes, 10.000 téléchargements/an
- Open Source : LGPL et QPL (commercialisé par GeometryFactory depuis 2003)

CGAL : *Computational Geometry Algorithms Library*

Depuis 1995 : mise en oeuvre des algorithmes de la Géométrie Algorithmique.

- Critères : adaptabilité, efficacité, robustesse
- Review des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.2 : 100 modules, 500.000 lignes, 10.000 téléchargements/an
- Open Source : LGPL et QPL (commercialisé par GeometryFactory depuis 2003)

CGAL : *Computational Geometry Algorithms Library*

Depuis 1995 : mise en oeuvre des algorithmes de la Géométrie Algorithmique.

- Critères : adaptabilité, efficacité, robustesse
- Review des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.2 : 100 modules, 500.000 lignes, 10.000 téléchargements/an
- Open Source : LGPL et QPL (commercialisé par GeometryFactory depuis 2003)

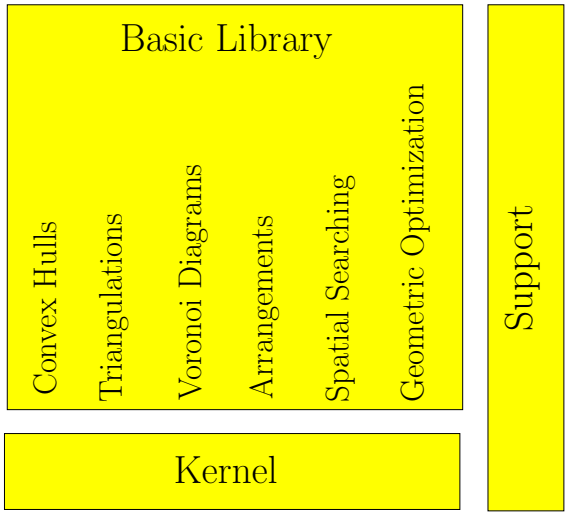
CGAL : *Computational Geometry Algorithms Library*

Depuis 1995 : mise en oeuvre des algorithmes de la Géométrie Algorithmique.

- Critères : adaptabilité, efficacité, robustesse
- Review des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.2 : 100 modules, 500.000 lignes, 10.000 téléchargements/an
- Open Source : LGPL et QPL (commercialisé par GeometryFactory depuis 2003)

CGAL : Architecture

Architecture générale : noyau, basic library, support library



Noyau de primitives géométriques

Les algorithmes se découpent logiquement en :

- une partie combinatoire (construit un graphe)
- une partie numérique (fait appel aux coordonnées)

La seconde fait appel à des primitives regroupées dans le noyau :

- Objets de base : points, segments, droites, cercles, coniques...
- Prédicats : orientations, comparaisons d'abscisses...
- Constructions : calcul d'intersections, de distances...

Noyau de primitives géométriques

Les algorithmes se découpent logiquement en :

- une partie combinatoire (construit un graphe)
- une partie numérique (fait appel aux coordonnées)

La seconde fait appel à des primitives regroupées dans le noyau :

- Objets de base : points, segments, droites, cercles, coniques...
- Prédicats : orientations, comparaisons d'abscisses...
- Constructions : calcul d'intersections, de distances...

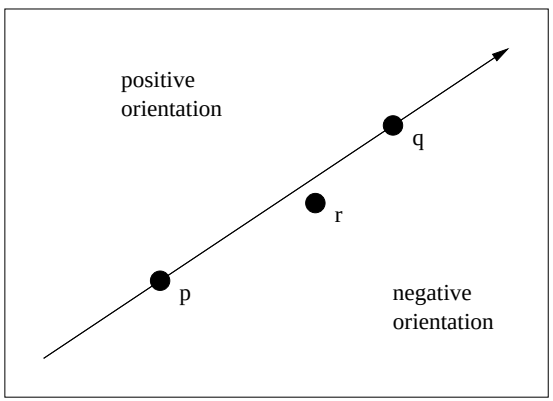
Noyau de primitives géométriques

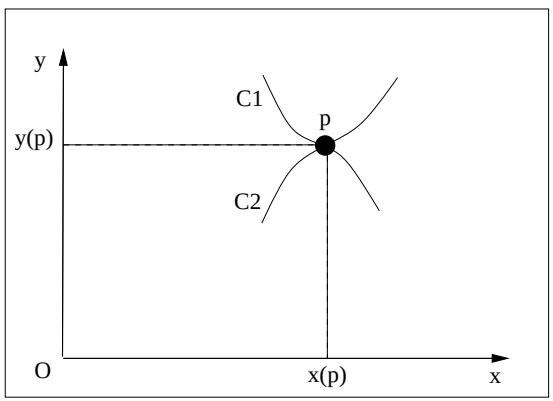
Les algorithmes se découpent logiquement en :

- une partie combinatoire (construit un graphe)
- une partie numérique (fait appel aux coordonnées)

La seconde fait appel à des primitives regroupées dans le noyau :

- Objets de base : points, segments, droites, cercles, coniques...
- Prédicats : orientations, comparaisons d'abscisses...
- Constructions : calcul d'intersections, de distances...

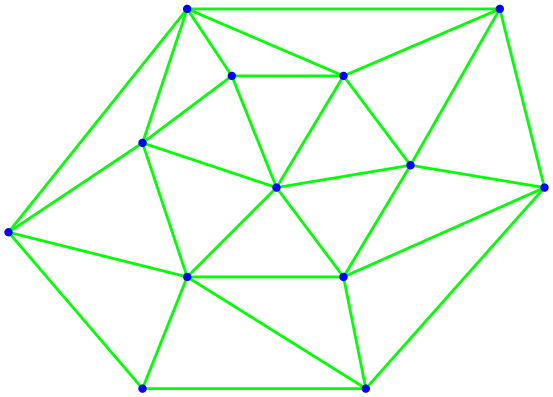




Plan

Triangulation de Delaunay

Algorithme incrémental en 2 étapes : localisation et mise à jour.

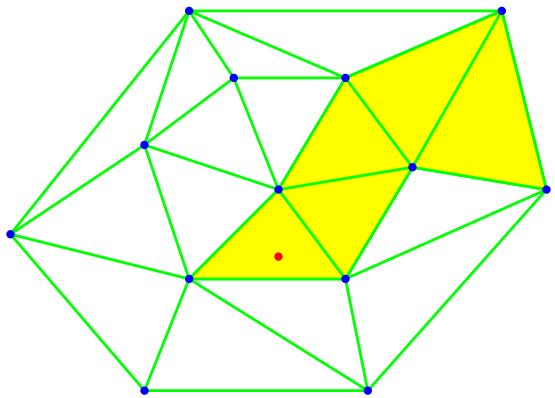


Localisation : prédicat orientation(p, q, r), signe de :

$$\begin{vmatrix} 1 & px & py \\ 1 & qx & qy \\ 1 & rx & ry \end{vmatrix} = \begin{vmatrix} qx - px & qy - py \\ rx - px & ry - py \end{vmatrix}$$

Triangulation de Delaunay

Algorithme incrémental en 2 étapes : localisation et mise à jour.

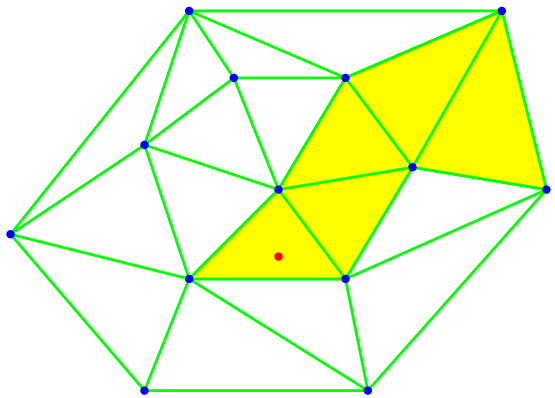


Localisation : prédicat orientation(p, q, r), signe de :

$$\begin{vmatrix} 1 & px & py \\ 1 & qx & qy \\ 1 & rx & ry \end{vmatrix} = \begin{vmatrix} qx - px & qy - py \\ rx - px & ry - py \end{vmatrix}$$

Triangulation de Delaunay

Algorithme incrémental en 2 étapes : localisation et mise à jour.

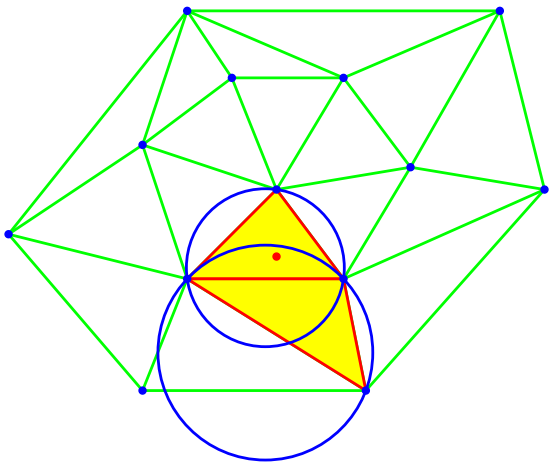


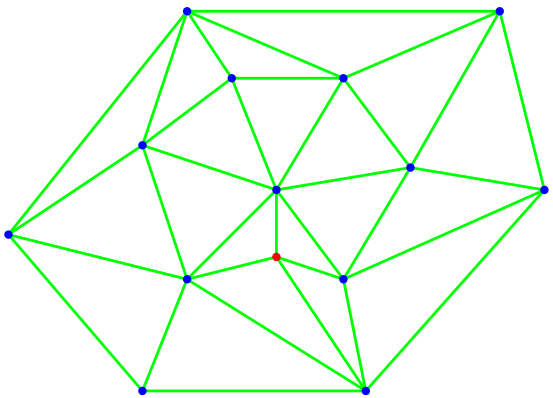
Localisation : prédicat orientation(p, q, r), signe de :

$$\begin{vmatrix} 1 & px & py \\ 1 & qx & qy \\ 1 & rx & ry \end{vmatrix} = \begin{vmatrix} qx - px & qy - py \\ rx - px & ry - py \end{vmatrix}$$

Triangulation de Delaunay

Algorithme incrémental en 2 étapes : localisation et mise à jour.



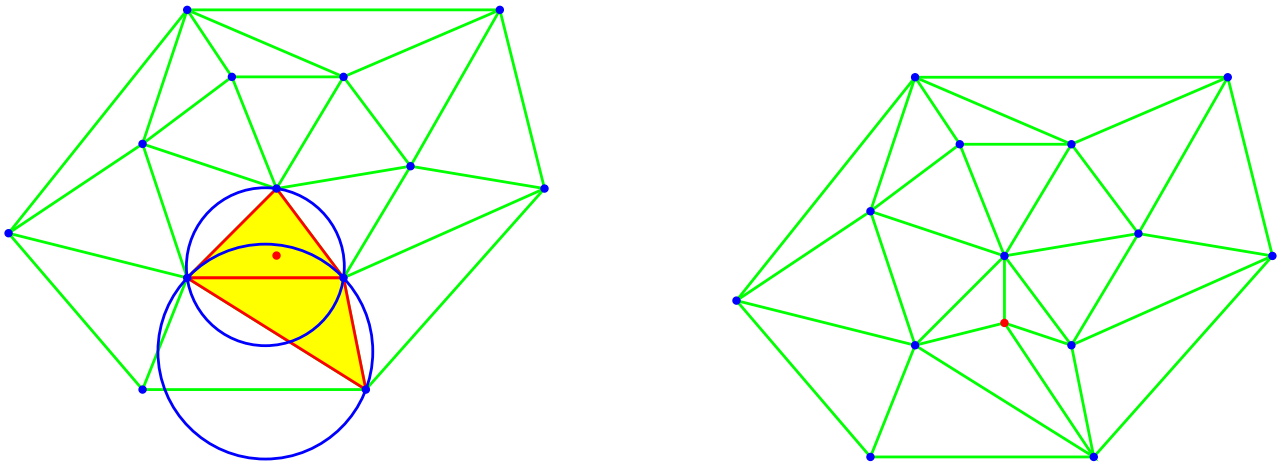


Mise à jour : prédicat in_circle(p, q, r, s), signe de :

$$\begin{vmatrix} 1 & px & py & px^2+py^2 \\ 1 & qx & qy & qx^2+qy^2 \\ 1 & rx & ry & rx^2+ry^2 \\ 1 & sx & sy & sx^2+sy^2 \end{vmatrix}$$

Triangulation de Delaunay

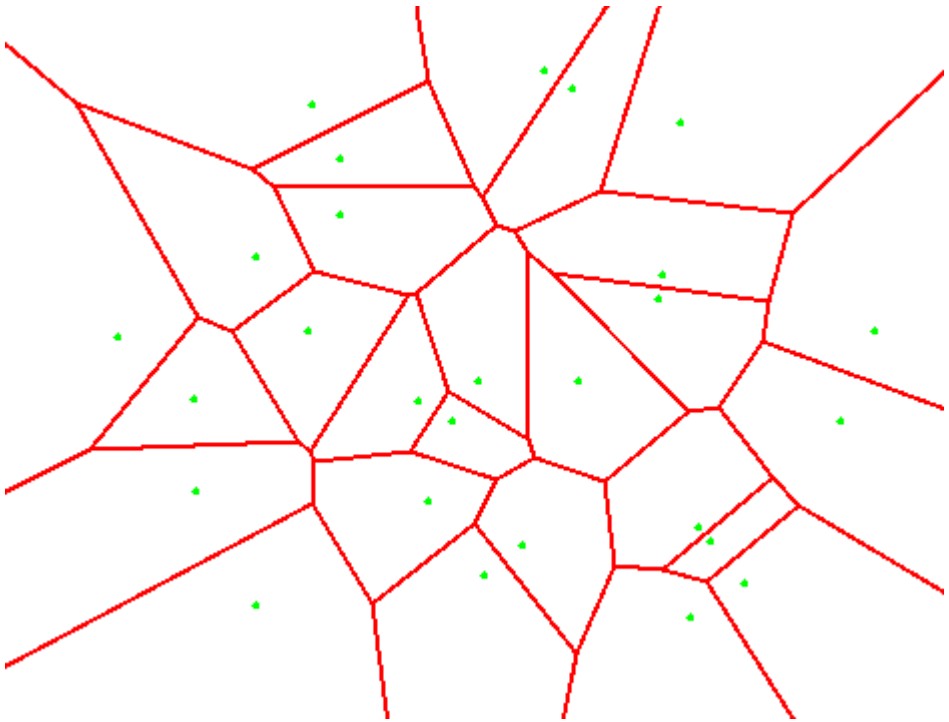
Algorithme incrémental en 2 étapes : localisation et mise à jour.



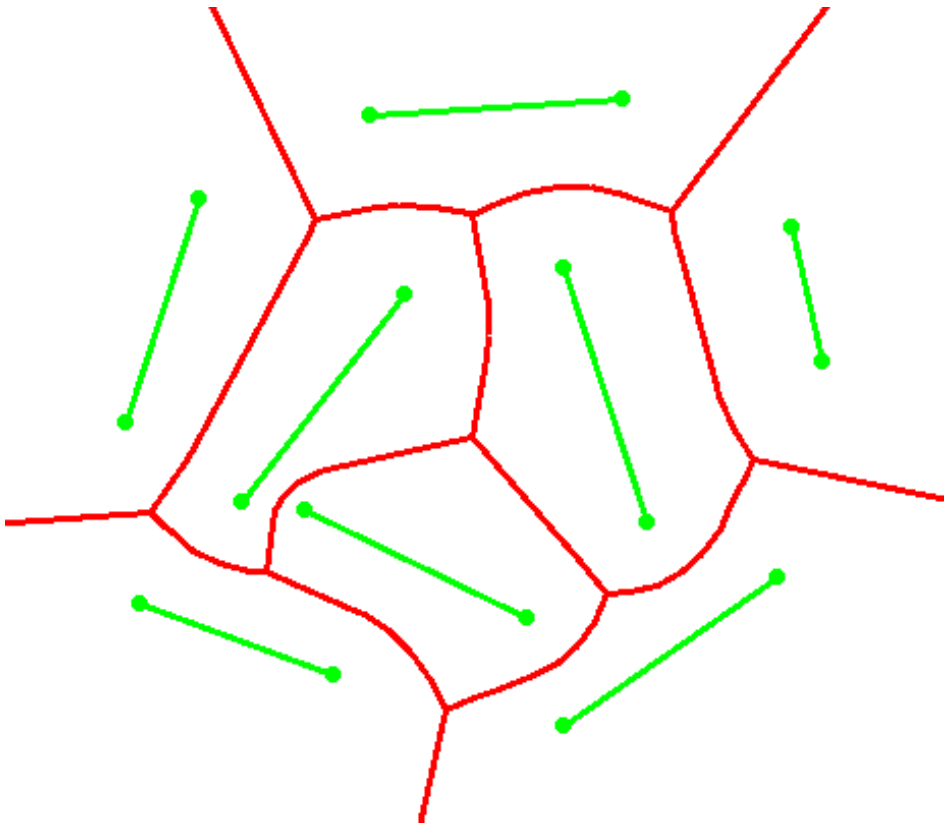
Mise à jour : prédicat in_circle(p, q, r, s), signe de :

$$\begin{vmatrix} 1 & px & py & px^2 + py^2 \\ 1 & qx & qy & qx^2 + qy^2 \\ 1 & rx & ry & rx^2 + ry^2 \\ 1 & sx & sy & sx^2 + sy^2 \end{vmatrix}$$

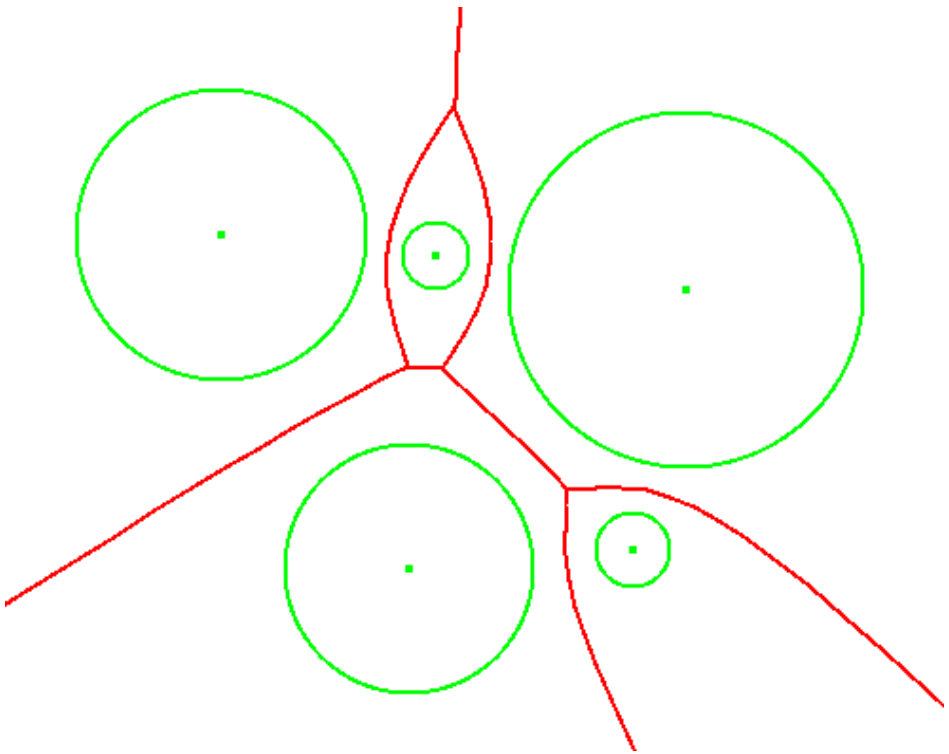
Diagrammes de Voronoï de points



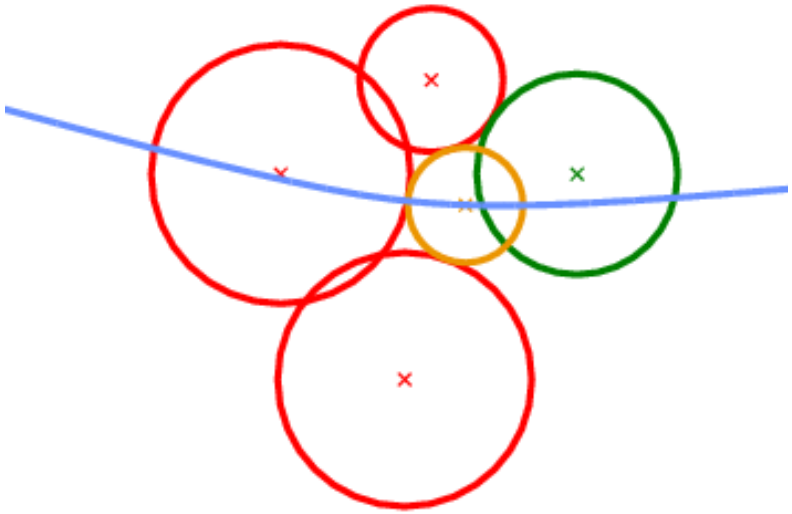
Diagrammes de Voronoï de segments



Diagrammes de Voronoï de cercles



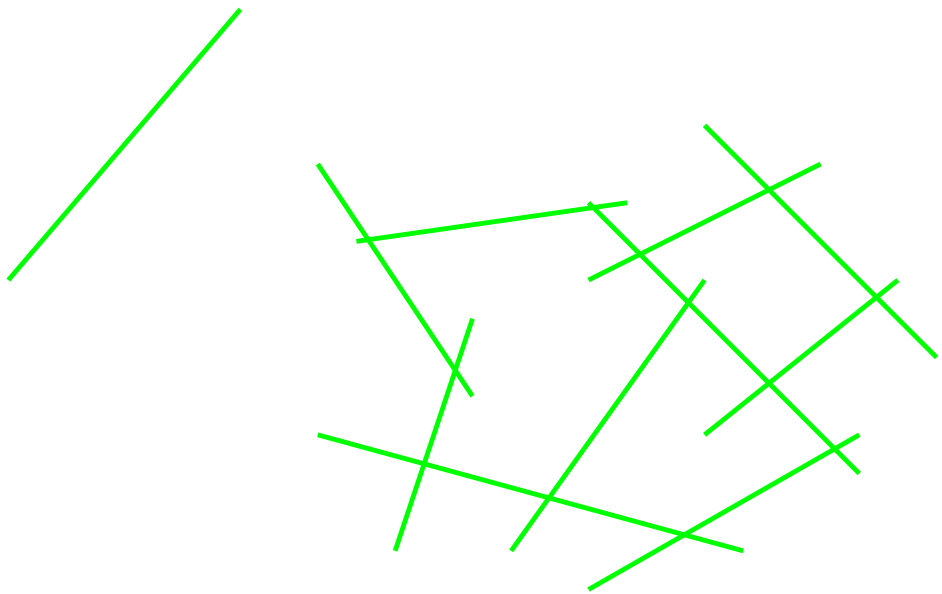
Un des prédicats du diagramme de Voronoï de cercles



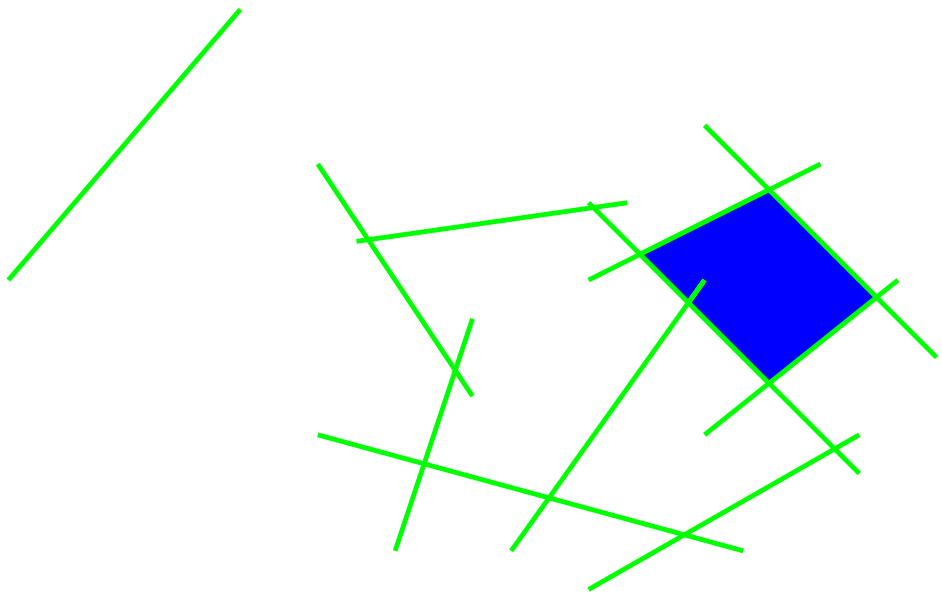
Techniques de comparaisons de racines

[Karavelas, Emiris : SODA'03]

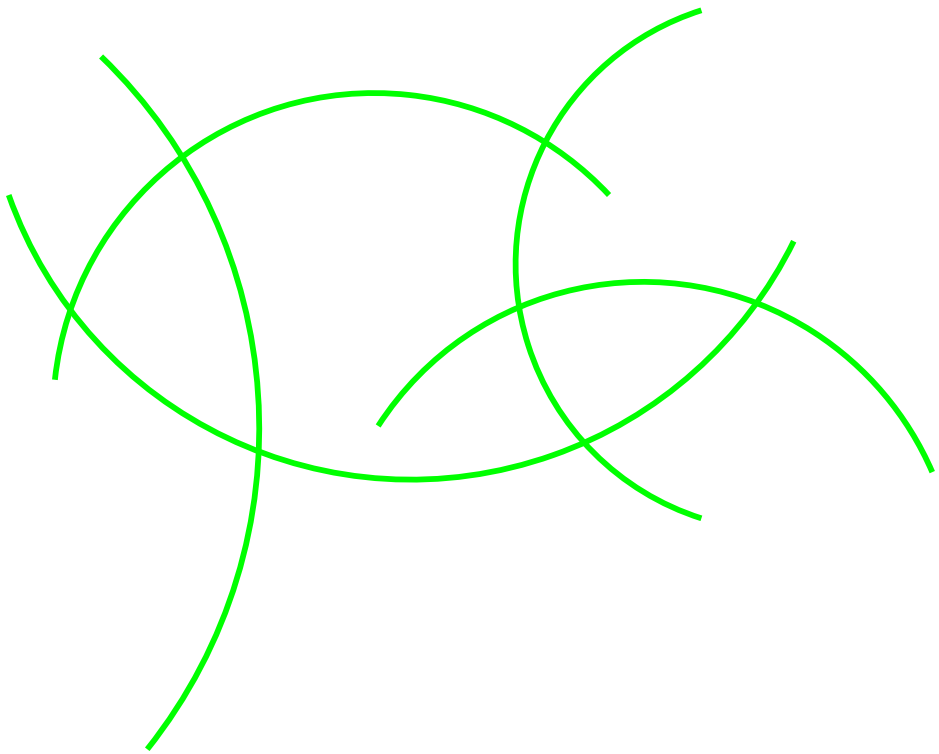
Arrangements de segments de droites



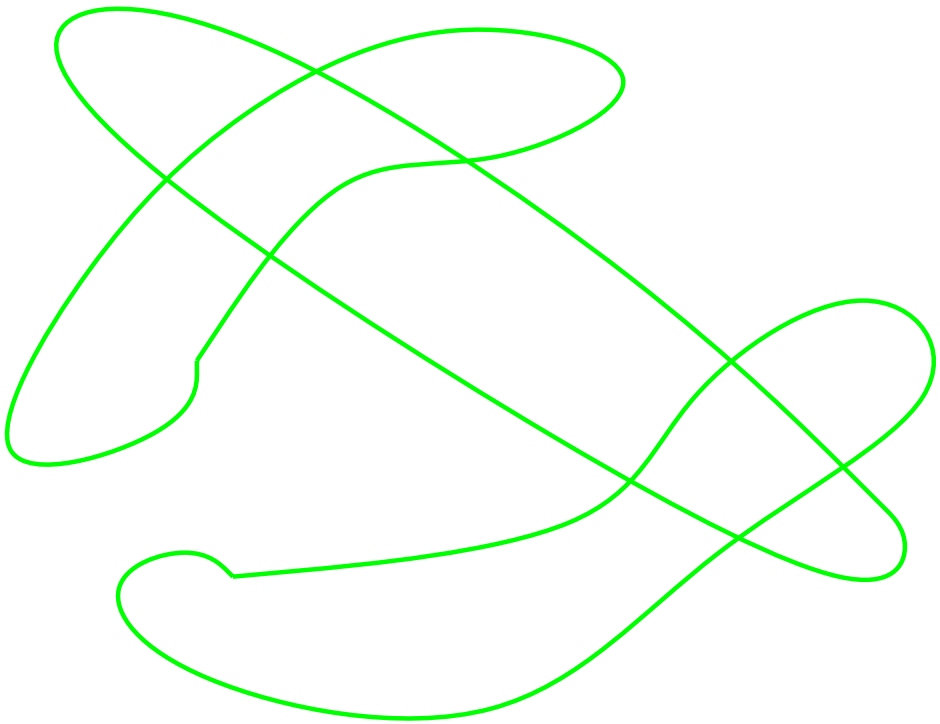
Arrangements de segments de droites



Arrangements d’arcs de cercles

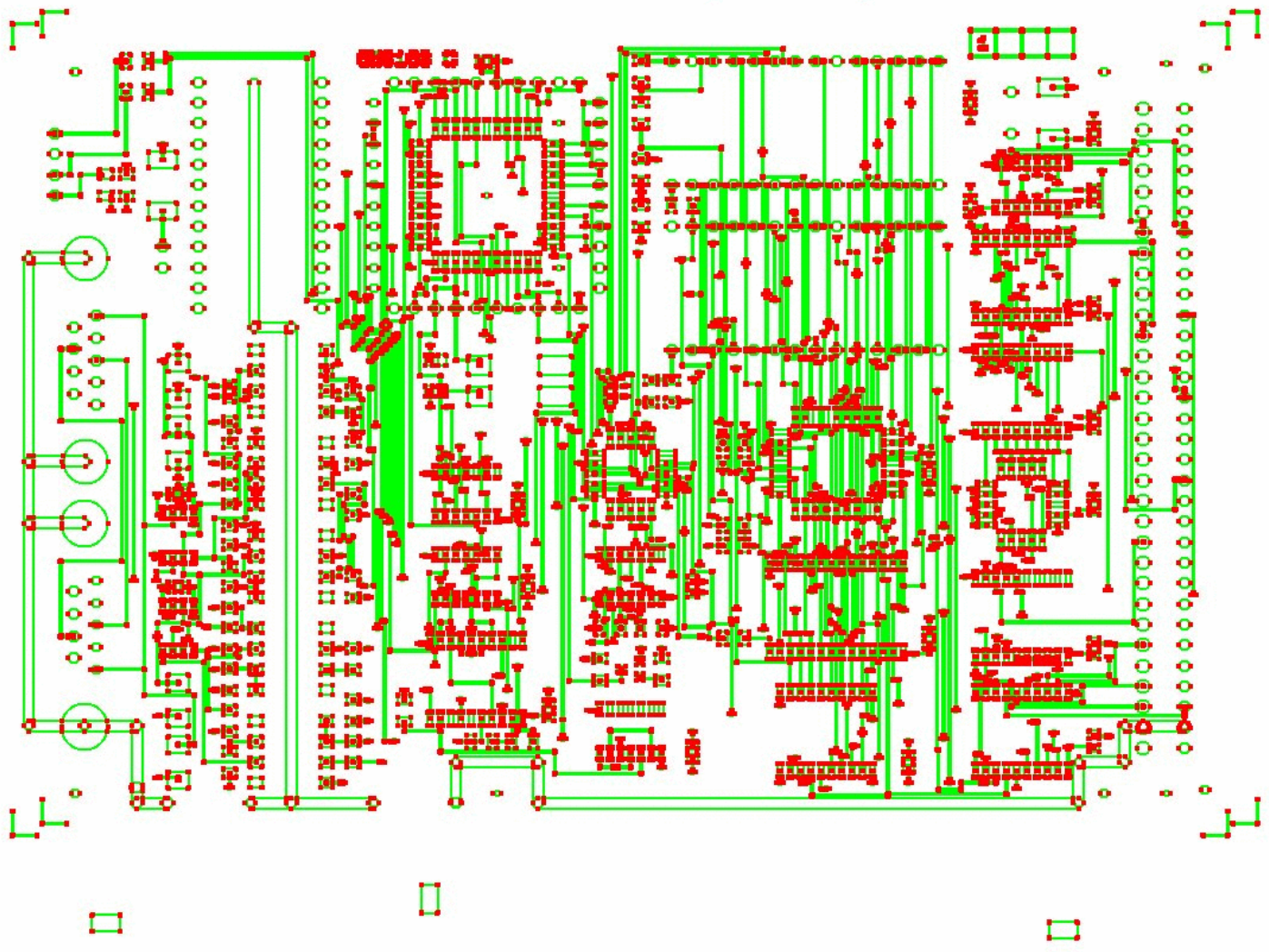


Arrangements de courbes

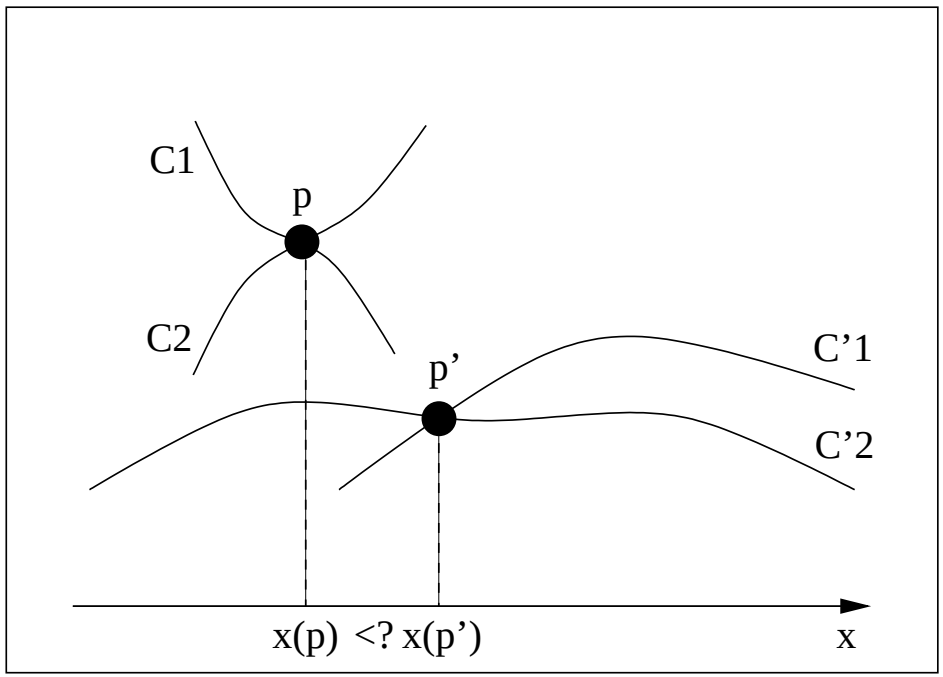


Noyau de calculs certifié pour la CAO
En 3D : arrangements de sphères, de quadriques...

Une application : union de polygones en VLSI



Comparaison d'abscisses d'intersection de courbes

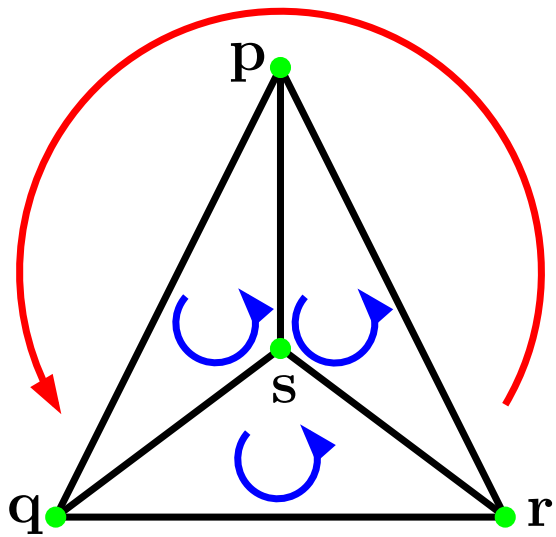


Courbes algébriques, comparaisons de nombres algébriques

Plan

Robustesse

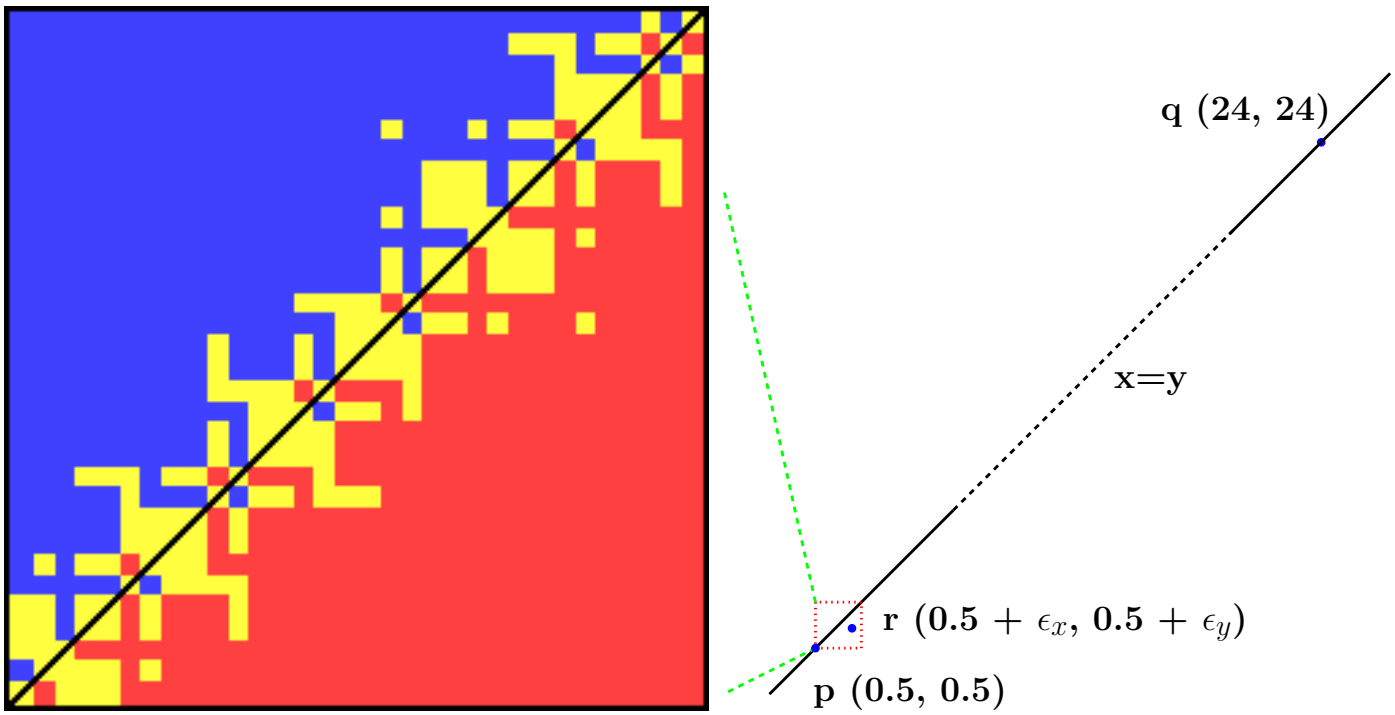
Les algorithmes reposent sur des théorèmes mathématiques, comme :



ccw(s, q, r)
ccw(p, s, r) => ccw(p, q, r)
ccw(p, q, s)

Robustesse

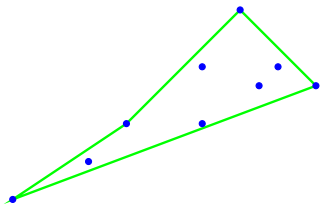
Exemple où la géométrie flottante diffère de la géométrie réelle :
orientation de points presque alignés.



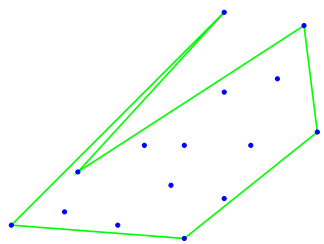
[Kettner, Mehlhorn, Schirra, P., Yap, ESA'04]

Conséquences possibles sur les algorithmes

- Le résultat est légèrement faux



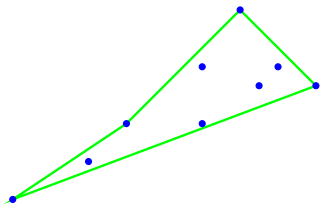
- Le résultat est grossièrement faux

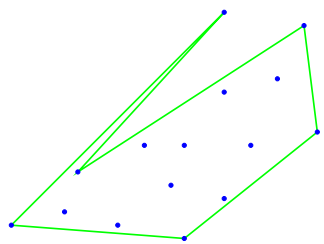


- L'algorithme s'arrête face à un cas supposé impossible
- L'algorithme boucle indéfiniment

Conséquences possibles sur les algorithmes

- Le résultat est légèrement faux





- Le résultat est grossièrement faux
- L'algorithme s'arrête face à un cas supposé impossible
- L'algorithme boucle indéfiniment

Robustesse : solutions

- Traitement au cas par cas : fastidieux, risque d'erreur, pas joli mathématiquement
- Utiliser des prédicats exacts (*Exact Geometric Computing*)

Remarques

- Le calcul flottant échoue sur les cas [presque] dégénérés.
- Ces cas arrivent plus ou moins souvent en pratique.

Robustesse : solutions

- Traitement au cas par cas : fastidieux, risque d'erreur, pas joli mathématiquement
- Utiliser des prédicats exacts (*Exact Geometric Computing*)

Remarques

- Le calcul flottant échoue sur les cas [presque] dégénérés.
- Ces cas arrivent plus ou moins souvent en pratique.

Robustesse : solutions

- Traitement au cas par cas : fastidieux, risque d'erreur, pas joli mathématiquement
- Utiliser des prédicats exacts (*Exact Geometric Computing*)

Remarques

- Le calcul flottant échoue sur les cas [presque] dégénérés.
- Ces cas arrivent plus ou moins souvent en pratique.

Plan

Types de nombres

Les primitives géométriques sont paramétrées par l'arithmétique.

- Entiers multiprécision [GMP, MPFR, LEDA...]
- Rationels multiprécision
- Flottants multiprécision
- Arithmétique d'intervalles (bornes `double` ou MP)

Nombres algébriques :

- Évaluation numérique avec bornes de séparations [CORE, LEDA]
- Polynômes, Sturm, résultants... [CGAL, SYNAPS]

Types de nombres

Les primitives géométriques sont paramétrées par l'arithmétique.

- Entiers multiprécision [GMP, MPFR, LEDA...]
- Rationels multiprécision
- Flottants multiprécision
- Arithmétique d'intervalles (bornes `double` ou MP)

Nombres algébriques :

- Évaluation numérique avec bornes de séparations [CORE, LEDA]
- Polynômes, Sturm, résultants... [CGAL, SYNAPS]

Types de nombres

Les primitives géométriques sont paramétrées par l'arithmétique.

- Entiers multiprécision [GMP, MPFR, LEDA...]
- Rationels multiprécision
- Flottants multiprécision
- Arithmétique d'intervalles (bornes `double` ou MP)

Nombres algébriques :

- Évaluation numérique avec bornes de séparations [CORE, LEDA]
- Polynômes, Sturm, résultants... [CGAL, SYNAPS]

Types de nombres

Les primitives géométriques sont paramétrées par l'arithmétique.

- Entiers multiprécision [GMP, MPFR, LEDA...]
- Rationels multiprécision
- Flottants multiprécision
- Arithmétique d'intervalles (bornes `double` ou MP)

Nombres algébriques :

- Évaluation numérique avec bornes de séparations [CORE, LEDA]
- Polynômes, Sturm, résultants... [CGAL, SYNAPS]

Calcul modulaire

Autre représentation des entiers qui permet d'effectuer des additions et multiplications en temps linéaire.

Théorème des restes chinois : Soit $m_1 \dots m_n$ des entiers premiers entre eux, on considère l'homéomorphisme d'anneaux :

$$(\mathbb{Z}/_{M\mathbb{Z}}, +, \times) \sim \prod_{i=1}^{n} \left(\mathbb{Z}/_{m_i\mathbb{Z}}, +, \times\right)$$

où $M = \prod_{i=1}^{n} m_i$.

On peut effectuer les opérations $(+, -, \times)$ dans $\mathbb{Z}/_{m_i\mathbb{Z}}$ pour tous les m_i, puis récupérer la valeur dans $\mathbb{Z}/_{M\mathbb{Z}}$.

Calcul modulaire

Autre représentation des entiers qui permet d'effectuer des additions et multiplications en temps linéaire.
Théorème des restes chinois : Soit $m_1 \dots m_n$ des entiers premiers entre eux, on considère l'homéomorphisme d'anneaux :

$$\left(\mathbb{Z}/_{M\mathbb{Z}}, +, \times\right) \sim \prod_{i=1}^{n} \left(\mathbb{Z}/_{m_i\mathbb{Z}}, +, \times\right)$$

où $M = \prod_{i=1}^{n} m_i$.
On peut effectuer les opérations $(+, -, \times)$ dans $\mathbb{Z}/_{m_i\mathbb{Z}}$ pour tous les m_i, puis récupérer la valeur dans $\mathbb{Z}/_{M\mathbb{Z}}$.

Calcul modulaire : prérequis

- Une borne sur le résultat doit être connue avant de commencer les calculs
- Pas adaptatif : la valeur (et le signe) du résultat dépend de tous les moduli
- Pas d'algorithmes de division où de racine carrée
- Les algorithmes de conversion des valeurs sont en $O(n \log n)$ (théorique) à $O(n^2)$ (pratique)

Calcul modulaire : avantages

- parallelisable
- $(+, -, \times)$ en temps linéaire,
  environ 8 fois plus lent que du calcul flottant,
  multiplié par le degré du polynôme
- Tests d'égalité très rapide

Calcul modulaire : exemple

$$m_i = 134217689 = 2^{27} - 39$$
$$a, b, c, d \in \mathbb{Z}/_{m_i\mathbb{Z}} \equiv [-m_i/2; m_i/2]$$
$$|a \times d - b \times c| < 2^{53}$$

```
double  MOD_det2x2 (double a, b, c, d)
{
  double det = a*d-b*c;
  return det - M_i * round (det * M_i_inv);
}
```

Finalement :

Flottant :	$2\times$, 1 +
Modulaire :	$4\times$, 4 +
Decoupage :	$2\times$, 4 +, overflow. . .

Calcul modulaire : exemple

$$m_i = 134217689 = 2^{27} - 39$$
$$a, b, c, d \in \mathbb{Z}/_{m_i\mathbb{Z}} \equiv [-m_i/2; m_i/2]$$
$$|a \times d - b \times c| < 2^{53}$$

```
double  MOD_det2x2 (double a, b, c, d)
{
  double det = a*d-b*c;
  return det - M_i * round (det * M_i_inv);
}
```

Finalement :

Flottant :	$2\times$, 1 +
Modulaire :	$4\times$, 4 +
Decoupage :	$2\times$, 4 +, overflow. . .

Calcul modulaire : exemple

$$m_i = 134217689 = 2^{27} - 39$$
$$a, b, c, d \in \mathbb{Z}/_{m_i\mathbb{Z}} \equiv [-m_i/2; m_i/2]$$
$$|a \times d - b \times c| < 2^{53}$$

```
double  MOD_det2x2 (double a, b, c, d)
{
  double det = a*d-b*c;
  return det - M_i * round (det * M_i_inv);
}
```

Finalement :

Flottant :	$2 \times$, 1 +
Modulaire :	$4 \times$, 4 +
Decoupage :	$2 \times$, 4 +, overflow. . .

Calcul modulaire : calcul du signe (Lagrange)

$$M = \prod_{i=1}^{n} m_i, \quad |x| < \frac{M}{2}, \quad \forall i \; x \equiv x_i [m_i]$$

$$x \equiv \left(\sum_{i=1}^{n} \left(x_i w_i^{(n)} [m_i] \right) \frac{M}{m_i} \right) [M], \quad w_i^{(n)} \equiv \left(\frac{M}{m_i} [m_i] \right)^{-1}$$

$$\frac{x}{M} = \mathrm{Frac}\left(\frac{x}{M} \right) = \mathrm{Frac}\left(\sum_{i=1}^{n} \frac{x_i w_i^{(n)} [m_i]}{m_i} \right)$$

Erreur : $\epsilon = n 2^{-54}$

Si la valeur calculée est plus grande que l'erreur, on obtient le signe, sinon :

$$|x| < 2\epsilon M < \frac{M}{2m_n}$$

Calcul modulaire : calcul du signe (Lagrange)

$$M = \prod_{i=1}^{n} m_i, \quad |x| < \frac{M}{2}, \quad \forall i \; x \equiv x_i [m_i]$$

$$x \equiv \left(\sum_{i=1}^{n} \left(x_i w_i^{(n)} [m_i] \right) \frac{M}{m_i} \right) [M], \quad w_i^{(n)} \equiv \left(\frac{M}{m_i} [m_i] \right)^{-1}$$

$$\frac{x}{M} = \mathrm{Frac}\left(\frac{x}{M}\right) = \mathrm{Frac}\left(\sum_{i=1}^{n} \frac{x_i w_i^{(n)} [m_i]}{m_i} \right)$$

Erreur : $\epsilon = n 2^{-54}$

Si la valeur calculée est plus grande que l'erreur, on obtient le signe, sinon :

$$|x| < 2\epsilon M < \frac{M}{2m_n}$$

Calcul modulaire : calcul du signe (Lagrange)

$$M = \prod_{i=1}^{n} m_i, \quad |x| < \frac{M}{2}, \quad \forall i \; x \equiv x_i [m_i]$$

$$x \equiv \left(\sum_{i=1}^{n} \left(x_i w_i^{(n)} [m_i] \right) \frac{M}{m_i} \right) [M], \quad w_i^{(n)} \equiv \left(\frac{M}{m_i} [m_i] \right)^{-1}$$

$$\frac{x}{M} = \text{Frac}\left(\frac{x}{M}\right) = \text{Frac}\left(\sum_{i=1}^{n} \frac{x_i w_i^{(n)} [m_i]}{m_i} \right)$$

Erreur : $\epsilon = n 2^{-54}$

Si la valeur calculée est plus grande que l'erreur, on obtient le signe, sinon :

$$|x| < 2\epsilon M < \frac{M}{2m_n}$$

Calcul modulaire : calcul du signe (Lagrange)

$$M = \prod_{i=1}^{n} m_i, \quad |x| < \frac{M}{2}, \quad \forall i \; x \equiv x_i [m_i]$$

$$x \equiv \left(\sum_{i=1}^{n} \left(x_i w_i^{(n)} [m_i] \right) \frac{M}{m_i} \right) [M], \quad w_i^{(n)} \equiv \left(\frac{M}{m_i} [m_i] \right)^{-1}$$

$$\frac{x}{M} = \mathrm{Frac}\left(\frac{x}{M}\right) = \mathrm{Frac}\left(\sum_{i=1}^{n} \frac{x_i w_i^{(n)} [m_i]}{m_i} \right)$$

Erreur : $\epsilon = n 2^{-54}$

Si la valeur calculée est plus grande que l'erreur, on obtient le signe, sinon :

$$|x| < 2\epsilon M < \frac{M}{2m_n}$$

Calcul modulaire : calcul du signe (Lagrange)

$$M = \prod_{i=1}^{n} m_i, \quad |x| < \frac{M}{2}, \quad \forall i \; x \equiv x_i [m_i]$$

$$x \equiv \left(\sum_{i=1}^{n} \left(x_i w_i^{(n)} [m_i] \right) \frac{M}{m_i} \right) [M], \quad w_i^{(n)} \equiv \left(\frac{M}{m_i} [m_i] \right)^{-1}$$

$$\frac{x}{M} = \mathrm{Frac}\left(\frac{x}{M} \right) = \mathrm{Frac}\left(\sum_{i=1}^{n} \frac{x_i w_i^{(n)} [m_i]}{m_i} \right)$$

Erreur : $\epsilon = n 2^{-54}$

Si la valeur calculée est plus grande que l'erreur, on obtient le signe, sinon :

$$|x| < 2\epsilon M < \frac{M}{2 m_n}$$

Calcul modulaire : signes de déterminants

d	grands	moyens	0	0 proba
2	7.5	8.0	8.8	4.8
3	15.5	18.1	18.9	7.3
4	32.5	36.3	38.9	12.5
5	109	116	119	26.8
6	184	192	196	40.5
10	1009	1033	1038	134
14	3320	3360	3380	324
20	16900	17000	17100	870

- entrées sur $b = 53$ bits
- compléxité théorique $O(bd^4 \log d)$
- temps en microsecondes sur UltraSparc @ 200MHz

Calcul modulaire : comparaisons

d	Flottant	MOD	GMP
2	0.2	7.6	6
3	0.4	16	33
4	0.8	36	136
5	2.2	75	435
6	6.3	186	1064
10	25.8	1021	10810
20	190	17000	431000

- Flottant : calcul inexact
- MOD : calcul modulaire exact
- GMP : calcul exact classique

Prédicats filtrés

Accélération des prédicats exacts par un filtre :

- évaluation flottante avec certificat
- arithmétique multiprécision uniquement si besoin

Exemples

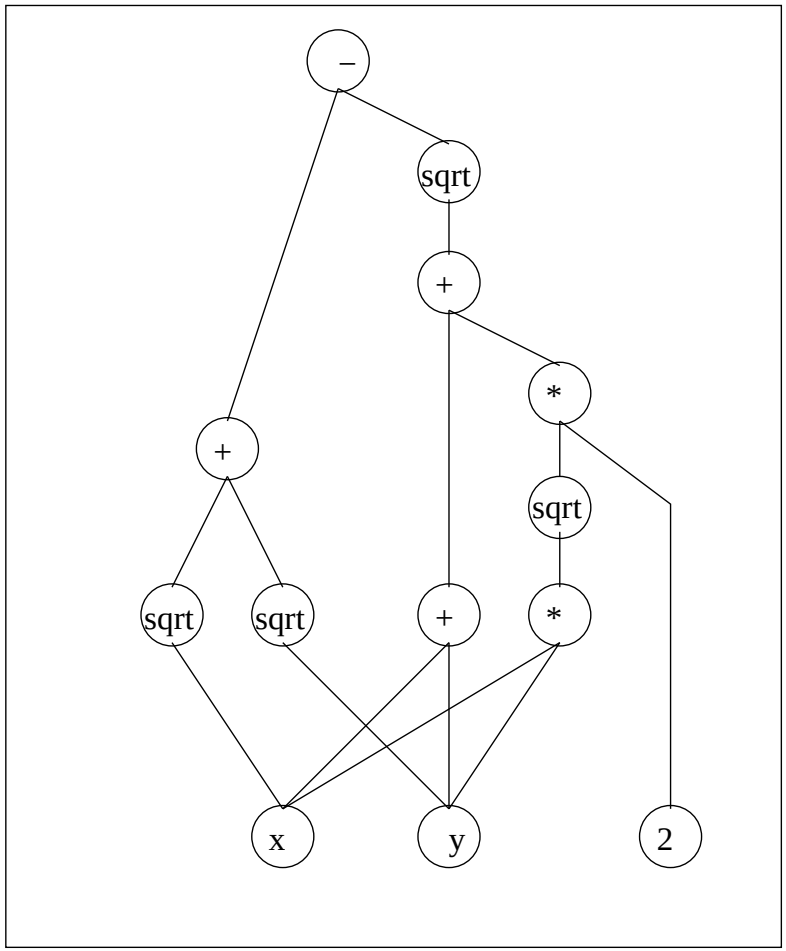
- arithmétique d'intervalles (filtres dynamiques), [Burnikel, Funke, Seel – Brönnimann, Burnikel, P.'98]
- ou analyse de code (filtres statiques) [Fortune'93... Melquiond, P.'05]

Aspects programmation :

- génération automatique de prédicats filtrés
- composition/cascade des méthodes

Type de nombres filtrés

DAG des opérations en mémoire. Ex : $\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}}$



Approximation et rafinement de précision itératif, à la demande.

Bornes de séparation

Définition

Une borne de séparation d'une expression E est une valeur $B(E)$ telle que :

$$|E| < B(E) => E = 0$$

Complexité : Au pire (valeur nulle), il faudra itérer jusqu'à une précision inférieure à $B(E)$ pour calculer le signe de E.

On sait calculer des bornes de séparations pour les expressions algébriques.
Il existe 3 types de bornes de séparations faciles à calculer non-comparables :

- Burnikel, Funke, Mehlhorn, Schirra, Schmitt "BFMSS" [2002]
- Li, Yap [2001]
- Mignotte "Degree-Measure" [1982]

Bornes de séparation

Définition

Une borne de séparation d'une expression E est une valeur $B(E)$ telle que :

$$|E| < B(E) => E = 0$$

Complexité : Au pire (valeur nulle), il faudra itérer jusqu'à une précision inférieure à $B(E)$ pour calculer le signe de E.

On sait calculer des bornes de séparations pour les expressions algébriques.
Il existe 3 types de bornes de séparations faciles à calculer non-comparables :

- Burnikel, Funke, Mehlhorn, Schirra, Schmitt "BFMSS" [2002]
- Li, Yap [2001]
- Mignotte "Degree-Measure" [1982]

Bornes de séparation

Définition

Une borne de séparation d'une expression E est une valeur $B(E)$ telle que :

$$|E| < B(E) => E = 0$$

Complexité : Au pire (valeur nulle), il faudra itérer jusqu'à une précision inférieure à $B(E)$ pour calculer le signe de E.

On sait calculer des bornes de séparations pour les expressions algébriques.
Il existe 3 types de bornes de séparations faciles à calculer non-comparables :

- Burnikel, Funke, Mehlhorn, Schirra, Schmitt "BFMSS" [2002]
- Li, Yap [2001]
- Mignotte "Degree-Measure" [1982]

Borne BFMSS

On note $E = \frac{U}{L}$. U, L sont des entiers algébriques, on maintient des bornes u, ℓ sur leurs conjugués avec les règles suivantes :

E	$u(E)$	$\ell(E)$
entier n	$\lvert n \rvert$	1
$E' \pm E''$	$u'\ell'' + \ell' u''$	$\ell'\ell''$
$E' \times E''$	$u'u''$	$\ell'\ell''$
$E' \div E''$	$u'\ell''$	$\ell' u''$
$\sqrt[p]{E'}$	$\min(\sqrt[p]{u'\ell'^{p-1}}, u')$	$\min(\ell', \sqrt[p]{u'^{p-1}\ell'})$

On obtient une borne de séparation sur E par :

$$\frac{1}{u(E)^{D(E)-1}\ell(E)}$$

Exemple

$$\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}} > 0?$$

Avec $x = \frac{a}{b}$ et $y = \frac{c}{d}$, a, b, c, d entiers $< 2^L$.

Bornes de séparation :

- BFMSS : $2^{-(96L+60)}$
- Li-Yap : $2^{-(28L+60)}$

Extensions

- diamond operator [Schmitt'03].
- version binaire [P., Yap'03].

Exemple

$$\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}} > 0?$$

Avec $x = \frac{a}{b}$ et $y = \frac{c}{d}$, a, b, c, d entiers $< 2^L$.

Bornes de séparation :

- BFMSS : $2^{-(96L+60)}$
- Li-Yap : $2^{-(28L+60)}$

Extensions

- diamond operator [Schmitt'03].
- version binaire [P., Yap'03].

Prédicats filtrés : comparaisons

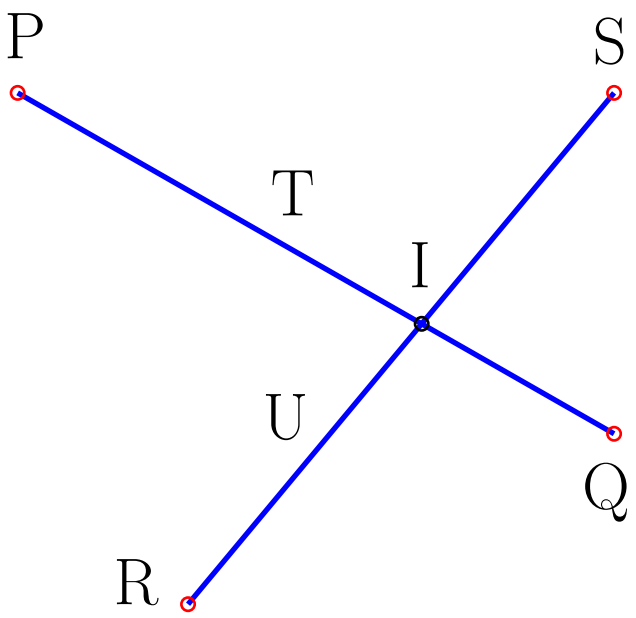
Temps de calcul d'une triangulation de Delaunay 3D.

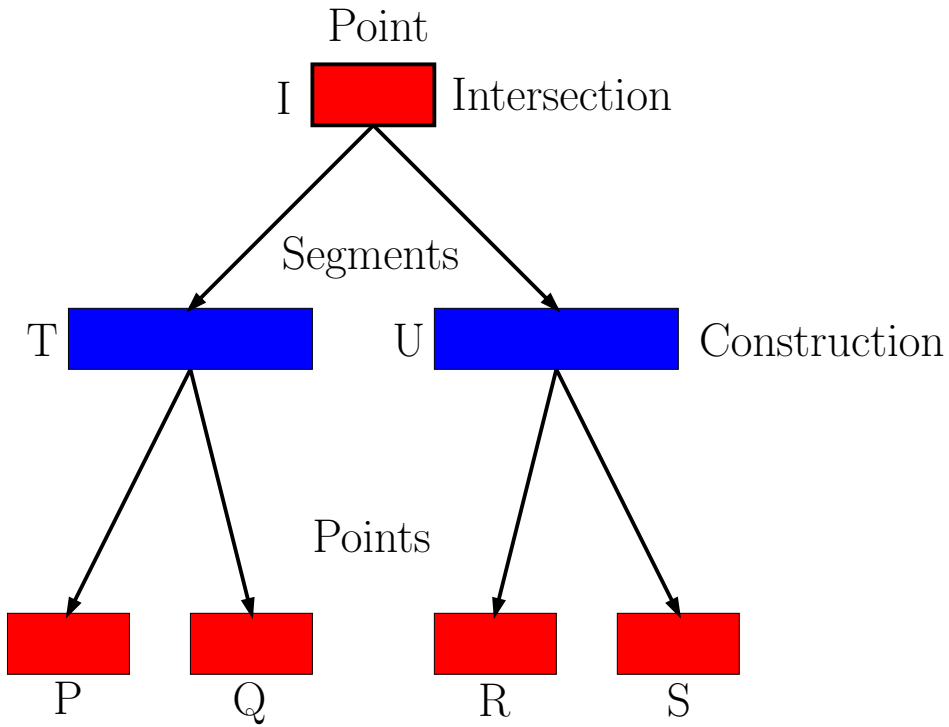
	R5	E	M	B	D
double	40.6	41.0	43.7	50.3	loops
MPF	3,063	2,777	3,195	3,472	214
Interval + MPF	137.2	133.6	144.6	165.1	15.8
semi static + Interval + MPF	51.8	61.0	59.1	93.1	8.9
almost static + semi static + Interval + MPF	44.4	55.0	52.0	87.2	8.0
Shewchuk's predicates	57.9	57.5	62.8	71.7	7.2
CORE Expr	570	3520	1355	9600	173
LEDA real	682	640	742	850	125
`Lazy_exact_nt<MPF>`	705	631	726	820	67

Critère important : taux d'échec des filtres.
Interface utilisateur dans CGAL : choix de noyau différent.

Constructions géométriques filtrées

Difficulté supplémentaire : stockage en mémoire des objets géométriques
Intérêt : regroupement des calculs, et moins de mémoire





Plan

Algorithmes et classes de traits

Les algorithmes sont paramétrés (templates) par des classes de traits géométriques, qui fournissent les :

- types des objets manipulés par l'algorithme : Point_2, Tetrahedron_3...
- prédicats que l'algorithme applique aux objets : Orientation_2, Side_of_oriented_sphere_3...
- constructions : Construct_mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

Ces deux derniers sont fournis sous forme de function objects.
Les besoins des algorithmes sont décrits sous forme de concepts pour leur paramètre template.

Algorithmes et classes de traits

Les algorithmes sont paramétrés (templates) par des classes de traits géométriques, qui fournissent les :

- types des objets manipulés par l'algorithme : Point_2, Tetrahedron_3...
- prédicats que l'algorithme applique aux objets : Orientation_2, Side_of_oriented_sphere_3...
- constructions : Construct_mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

Ces deux derniers sont fournis sous forme de function objects.
Les besoins des algorithmes sont décrits sous forme de concepts pour leur paramètre template.

Algorithmes et classes de traits

Les algorithmes sont paramétrés (templates) par des classes de traits géométriques, qui fournissent les :

- types des objets manipulés par l'algorithme : Point_2, Tetrahedron_3...
- prédicats que l'algorithme applique aux objets : Orientation_2, Side_of_oriented_sphere_3...
- constructions : Construct_mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

Ces deux derniers sont fournis sous forme de function objects.

Les besoins des algorithmes sont décrits sous forme de concepts pour leur paramètre template.

Algorithmes et classes de traits

Les algorithmes sont paramétrés (templates) par des classes de traits géométriques, qui fournissent les :

- types des objets manipulés par l'algorithme : Point_2, Tetrahedron_3...
- prédicats que l'algorithme applique aux objets : Orientation_2, Side_of_oriented_sphere_3...
- constructions : Construct_mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

Ces deux derniers sont fournis sous forme de function objects.
Les besoins des algorithmes sont décrits sous forme de concepts pour leur paramètre template.

Noyaux

Le noyau peut être utilisé comme modèle pour le paramètre de classe de traits géométrique dans de nombreux algorithmes.

Noyaux classiques, paramétrés par des types de nombres :

Cartesian<FT>

Homogeneous<RT>

Ex : Triangulation_3<Cartesian<double> >

Cartesian<double> est un modèle du concept TriangulationTraits_3.

La fonctionalité du noyau est aussi disponible via des fonctions globales :

`CGAL : :orientation(p, q, r)`..

Noyaux

Le noyau peut être utilisé comme modèle pour le paramètre de classe de traits géométrique dans de nombreux algorithmes.
Noyaux classiques, paramétrés par des types de nombres :
Cartesian<FT>
Homogeneous<RT>
Ex : Triangulation_3<Cartesian<double> >
Cartesian<double> est un modèle du concept TriangulationTraits_3.
La fonctionalité du noyau est aussi disponible via des fonctions globales :
`CGAL : :orientation(p, q, r)`..

Noyaux

Le noyau peut être utilisé comme modèle pour le paramètre de classe de traits géométrique dans de nombreux algorithmes.
Noyaux classiques, paramétrés par des types de nombres :
Cartesian<FT>
Homogeneous<RT>
Ex : Triangulation_3<Cartesian<double> >
Cartesian<double> est un modèle du concept TriangulationTraits_3.
La fonctionalité du noyau est aussi disponible via des fonctions globales :
`CGAL : :orientation(p, q, r)`..

Noyaux

Le noyau peut être utilisé comme modèle pour le paramètre de classe de traits géométrique dans de nombreux algorithmes.
Noyaux classiques, paramétrés par des types de nombres :
Cartesian<FT>
Homogeneous<RT>
Ex : Triangulation_3<Cartesian<double> >
Cartesian<double> est un modèle du concept TriangulationTraits_3.
La fonctionalité du noyau est aussi disponible via des fonctions globales :
`CGAL : :orientation(p, q, r)`..

Noyaux

Le noyau peut être utilisé comme modèle pour le paramètre de classe de traits géométrique dans de nombreux algorithmes.
Noyaux classiques, paramétrés par des types de nombres :
Cartesian<FT>
Homogeneous<RT>
Ex : Triangulation_3<Cartesian<double> >
Cartesian<double> est un modèle du concept TriangulationTraits_3.
La fonctionalité du noyau est aussi disponible via des fonctions globales :
`CGAL : :orientation(p, q, r)`..

Types de nombres

Paramètres valides pour les noyaux Cartesian...

FP :

double, float

Multi-précision :

Gmpz, Gmpq, CGAL : :MP_Float, leda : :integer...

Types de nombres incluant du filtrage :

leda : :real, CORE : :Expr, CGAL : :Lazy_exact_nt<>

Types de nombres

Paramètres valides pour les noyaux Cartesian...
FP :
double, float
Multi-précision :
Gmpz, Gmpq, CGAL : :MP_Float, leda : :integer...
Types de nombres incluant du filtrage :
leda : :real, CORE : :Expr, CGAL : :Lazy_exact_nt<>

Types de nombres

Paramètres valides pour les noyaux Cartesian...
FP :
double, float
Multi-précision :
Gmpz, Gmpq, CGAL : :MP_Float, leda : :integer...
Types de nombres incluant du filtrage :
leda : :real, CORE : :Expr, CGAL : :Lazy_exact_nt<>

Types de nombres

Paramètres valides pour les noyaux Cartesian...
FP :
double, float
Multi-précision :
Gmpz, Gmpq, CGAL : :MP_Float, leda : :integer...
Types de nombres incluant du filtrage :
leda : :real, CORE : :Expr, CGAL : :Lazy_exact_nt<>

Outils internes

Arithmétique d'intervalles : CGAL : :Interval_nt, boost : :interval
Générateur de prédicats filtrés (dynamique) utilisant des exceptions C++ : CGAL : :Filtered_predicate<>

Outils internes

Arithmétique d'intervalles : CGAL : :Interval_nt, boost : :interval
Générateur de prédicats filtrés (dynamique) utilisant des exceptions C++ :
CGAL : :Filtered_predicate<>

Noyaux filtrés

CGAL : :Filtered_kernel$< K >$ fournit des prédicats avec des filtres statiques, et tous les autres dynamiques.

Noyaux recommandés :

CGAL : :Exact_predicates_exact_constructions_kernel

CGAL : :Exact_predicates_inexact_constructions_kernel

Noyaux filtrés

CGAL : :Filtered_kernel$< K >$ fournit des prédicats avec des filtres statiques, et tous les autres dynamiques.
Noyaux recommandés :
CGAL : :Exact_predicates_exact_constructions_kernel
CGAL : :Exact_predicates_inexact_constructions_kernel

Exemple 1

```
template < typename K >
struct My_orientation_2
{
  typedef typename K::RT        RT;
  typedef typename K::Point_2   Point_2;

  CGAL::Orientation
  operator()(const Point_2 &p, const Point_2 &q,
             const Point_2 &r) const
  {
    RT prx = p.x() - r.x();   RT pry = p.y() - r.y();
    RT qrx = q.x() - r.x();   RT qry = q.y() - r.y();
    return static_cast<CGAL::Orientation> (
             CGAL::sign( prx*qry - qrx*pqy ) );
  }
};
```

Exemple 2

```
// On l'utilise
typedef CGAL::Cartesian<double>  Kernel;
Kernel::Point_2 p(1, 2), q(2, 3), r(4, 5);
My_orientation_2<Kernel> orientation;
CGAL::Orientation ori = orientation(p, q, r);
```

Utiliser `Filtered_predicate`

```
typedef CGAL::Simple_cartesian<double> K;
typedef CGAL::Simple_cartesian<CGAL::Interval_nt_advanced> FK;
typedef CGAL::Simple_cartesian<CGAL::MP_Float> EK;
typedef CGAL::Cartesian_converter<K, EK> C2E;
typedef CGAL::Cartesian_converter<K, FK> C2F;

typedef CGAL::Filtered_predicate<My_orientation_2<EK>,
                                 My_orientation_2<FK>,
                                 C2E, C2F>  Orientation_2;

...
  K::Point_2 p(1,2), q(2,3), r(3,4);
  Orientation_2 orientation;
  orientation(p, q, r);
  return 0;
```

Plan

Travaux en cours

- Traitement efficace d'objets courbes de petits degrés
- Amélioration du traitement des constructions géométriques
- Arrondis géométriques
- ...

Des questions ?

Travaux en cours

- Traitement efficace d'objets courbes de petits degrés
- Amélioration du traitement des constructions géométriques
- Arrondis géométriques
- ...

Des questions ?