Bases d'arithmétique pour le calcul géométrique

Sylvain Pion

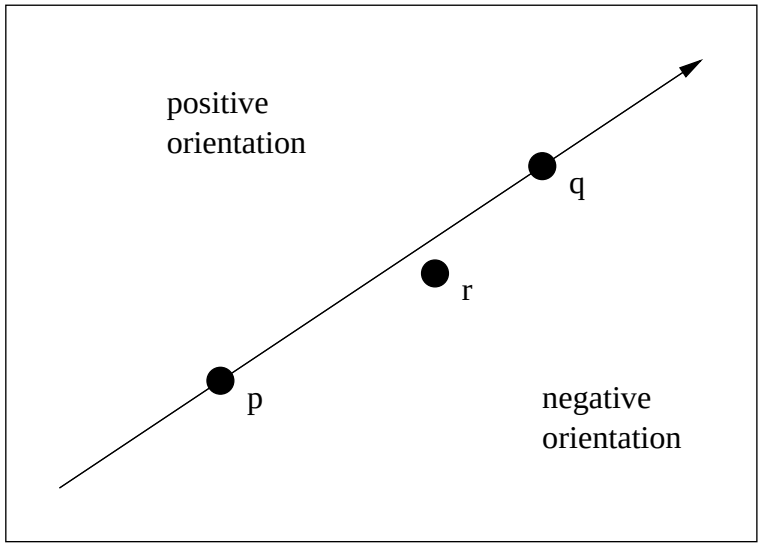
INRIA Sophia Antipolis

Plan

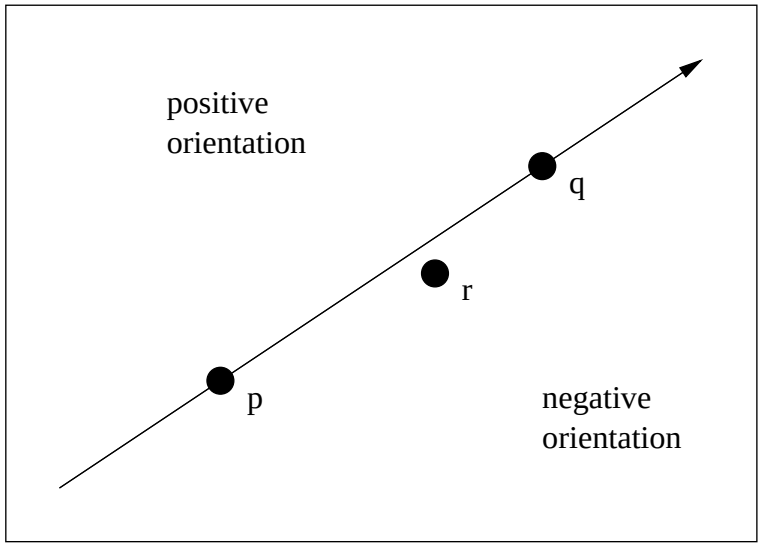
- Liens entre la géométrie et l'arithmétique
- Calcul flottant
- Calcul exact
- Calcul modulaire
- Bornes de séparations
- Filtres arithmétiques
- Implantation dans CGAL

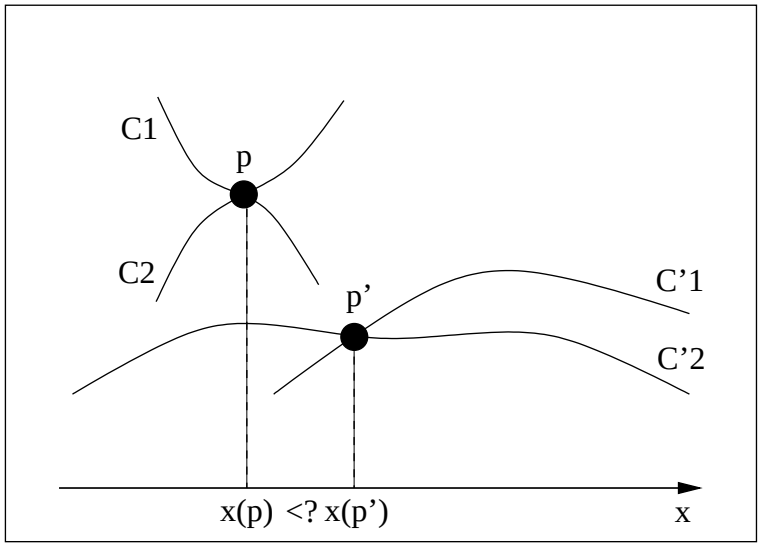
Introduction

Exemples de prédicats géométriques

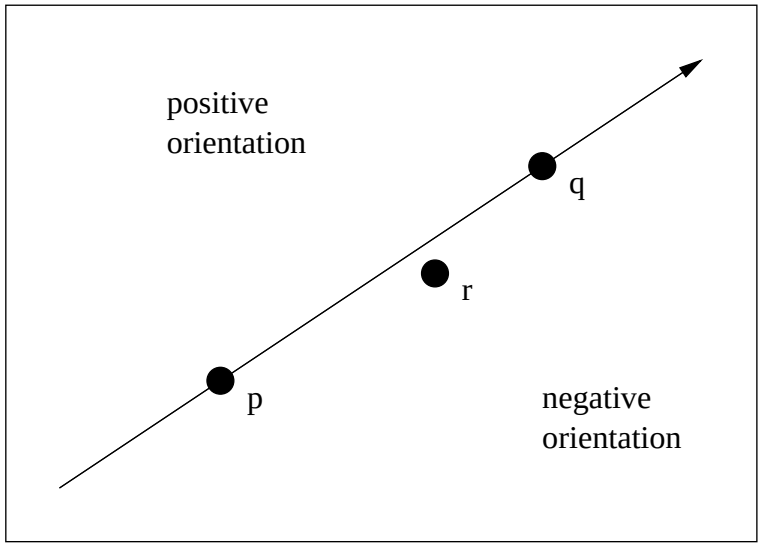


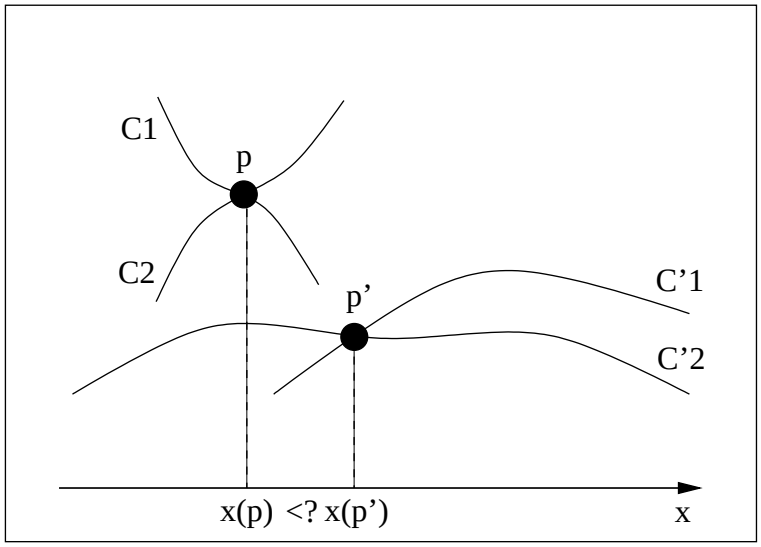
Exemples de prédicats géométriques





Exemples de prédicats géométriques

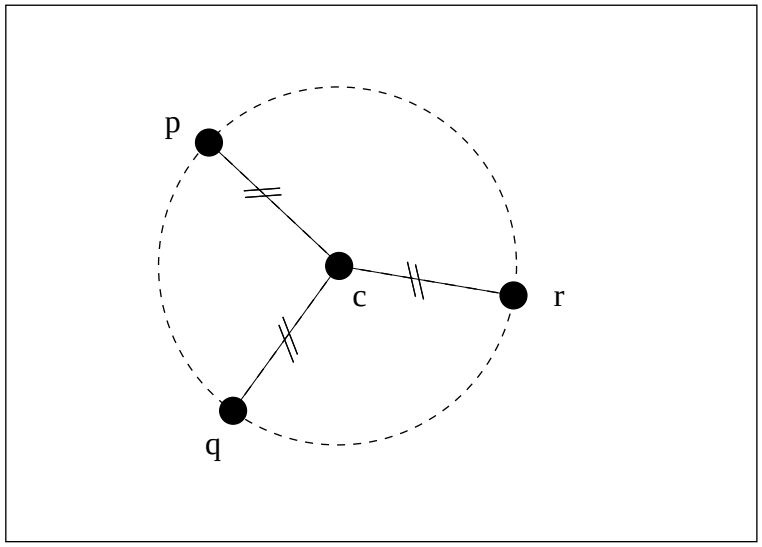




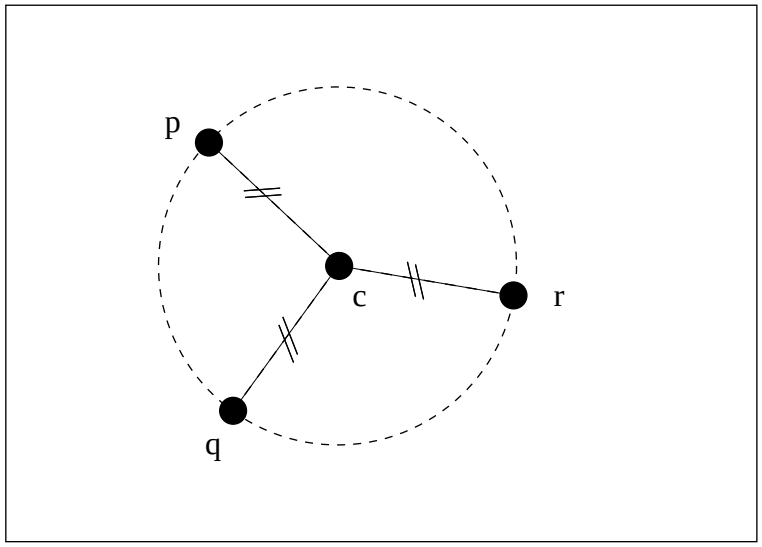
orientation$(p, q, r) =$
sign$((x(p) - x(r)) \times (y(q) - y(r)) - (x(q) - x(r)) \times (y(p) - y(r)))$

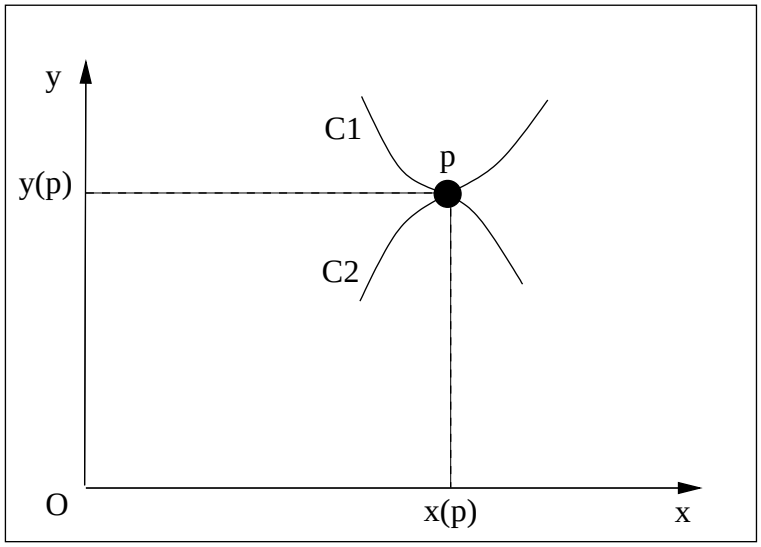
Prédicat de degré 2.

Exemples de constructions géométriques



Exemples de constructions géométriques





De la géométrie à l'arithmétique

Algorithme géométrique

De la géométrie à l'arithmétique

Algorithme géométrique

$\Rightarrow$ Opérations géométriques (prédicats & constructions)

De la géométrie à l'arithmétique

Algorithme géométrique

$\Rightarrow$ Opérations géométriques (prédicats & constructions)

$\Rightarrow$ Opérations algébriques sur les coordonnées/coefficients

De la géométrie à l'arithmétique

Algorithme géométrique

$\Rightarrow$ Opérations géométriques (prédicats & constructions)

$\Rightarrow$ Opérations algébriques sur les coordonnées/coefficients

$\Rightarrow$ Opérations arithmétiques $(+, -, \times, \div, \sqrt{}\ldots)$

Arithmétique ⇒ Géométrie

Coût de l'arithmétique ⇒ Complexité en temps des algorithmes géométriques

Arithmétique ⇒ Géométrie

Coût de l'arithmétique ⇒ Complexité en temps des algorithmes géométriques

Arithmétique approchée ⇒ problèmes de robustesse des algorithmes géométriques

Le modèle Real-RAM

Modèle de calculateur réel à accès aléatoire (RAM = Random access machine).

Modèle théorique du comportement de l'arithmétique réelle sur ordinateur.

Le modèle Real-RAM

Modèle de calculateur réel à accès aléatoire (RAM = Random access machine).

Modèle théorique du comportement de l'arithmétique réelle sur ordinateur.

- Toutes les opérations arithmétiques sur les réels coûtent un temps O(1) (et sont exactes).

Le modèle Real-RAM

Modèle de calculateur réel à accès aléatoire (RAM = Random access machine).

Modèle théorique du comportement de l'arithmétique réelle sur ordinateur.

- Toutes les opérations arithmétiques sur les réels coûtent un temps O(1) (et sont exactes).
- Toutes les variables réelles occupent O(1) espace mémoire.

Le modèle Real-RAM

Modèle de calculateur réel à accès aléatoire (RAM = Random access machine).

Modèle théorique du comportement de l'arithmétique réelle sur ordinateur.

- Toutes les opérations arithmétiques sur les réels coûtent un temps O(1) (et sont exactes).
- Toutes les variables réelles occupent O(1) espace mémoire.

Les analyses de complexité des algorithmes géométriques sont faites traditionnellement dans ce modèle.

Relation avec la réalité des ordinateurs ?

Deux approches :

- Calcul flottant, approché.

Relation avec la réalité des ordinateurs ?

Deux approches :

- Calcul flottant, approché.
- Calcul exact, plus lent.

Relation avec la réalité des ordinateurs ?

Deux approches :

- Calcul flottant, approché.
- Calcul exact, plus lent.

Pour la géométrie : quelle approche est-elle la meilleure en pratique ?

Relation avec la réalité des ordinateurs ?

Deux approches :

- Calcul flottant, approché.
- Calcul exact, plus lent.

Pour la géométrie : quelle approche est-elle la meilleure en pratique ?

Quel est le véritable coût de l'approche exacte ?

Calcul flottant

Standard IEEE 754

Standardisation des opérations flottantes de base des ordinateurs (1985).

Représentation machine de $(-1)^s \times 1.m \times 2^e$ (pour la double précision, 64 bits):

s	exposant	mantisse
1	11	52

Standard IEEE 754

Standardisation des opérations flottantes de base des ordinateurs (1985).

Représentation machine de $(-1)^s \times 1.m \times 2^e$ (pour la double précision, 64 bits):

s	exposant	mantisse
1	11	52

- 5 opérations : $+, -, \times, \div, \sqrt{}$

Standard IEEE 754

Standardisation des opérations flottantes de base des ordinateurs (1985).

Représentation machine de $(-1)^s \times 1.m \times 2^e$ (pour la double précision, 64 bits):

s	exposant	mantisse
1	11	52

- 5 opérations : $+, -, \times, \div, \sqrt{}$
- 4 modes d'arrondis : au plus proche (nombre représentable), vers 0, vers $+\infty$, vers $-\infty$.

Standard IEEE 754

Standardisation des opérations flottantes de base des ordinateurs (1985).

Représentation machine de $(-1)^s \times 1.m \times 2^e$ (pour la double précision, 64 bits):

s	exposant	mantisse
1	11	52

- 5 opérations : $+, -, \times, \div, \sqrt{}$
- 4 modes d'arrondis : au plus proche (nombre représentable), vers 0, vers $+\infty$, vers $-\infty$.
- Valeurs spéciales : $+\infty$, $-\infty$, dénormaux, NaNs.

Standard IEEE 754

Standardisation des opérations flottantes de base des ordinateurs (1985).

Représentation machine de $(-1)^s \times 1.m \times 2^e$ (pour la double précision, 64 bits):

s	exposant	mantisse
1	11	52

- 5 opérations : $+, -, \times, \div, \sqrt{}$
- 4 modes d'arrondis : au plus proche (nombre représentable), vers 0, vers $+\infty$, vers $-\infty$.
- Valeurs spéciales : $+\infty$, $-\infty$, dénormaux, NaNs.
- Relativement bien suivie par l'industrie (langages, compilateurs, processeurs).

Standard IEEE 754

Standardisation des opérations flottantes de base des ordinateurs (1985).

Représentation machine de $(-1)^s \times 1.m \times 2^e$ (pour la double précision, 64 bits):

s	exposant	mantisse
1	11	52

- 5 opérations : $+, -, \times, \div, \sqrt{}$
- 4 modes d'arrondis : au plus proche (nombre représentable), vers 0, vers $+\infty$, vers $-\infty$.
- Valeurs spéciales : $+\infty$, $-\infty$, dénormaux, NaNs.
- Relativement bien suivie par l'industrie (langages, compilateurs, processeurs).

Voir : `http://stevehollasch.com/cgindex/coding/ieeefloat.html`

Erreurs d'arrondis

Définition : x étant un flottant positif, et y le plus petit flottant plus grand que x, on définit $\texttt{ulp}(x) = y - x$ (Unit in the Last Place).

Erreurs d'arrondis

Définition : x étant un flottant positif, et y le plus petit flottant plus grand que x, on définit $\texttt{ulp}(x) = y - x$ (Unit in the Last Place).

Rq 1 : $\texttt{ulp}(x)$ est une puissance de 2 (ou ∞).
Rq 2 : Dans les cas normaux : $\texttt{ulp}(x) \simeq x2^{-53}$

Erreurs d'arrondis

Définition : x étant un flottant positif, et y le plus petit flottant plus grand que x, on définit $\texttt{ulp}(x) = y - x$ (Unit in the Last Place).

Rq 1 : $\texttt{ulp}(x)$ est une puissance de 2 (ou ∞).
Rq 2 : Dans les cas normaux : $\texttt{ulp}(x) \simeq x2^{-53}$

Propriété : Pour toutes les opérations $+, -, \times, \div, \sqrt{}$,
la différence entre la valeur calculée r et la valeur exacte,
autrement dit, l'erreur d'arrondi, est majorée par :
$\texttt{ulp}(r)/2$ pour le mode d'arrondi au plus proche, et
$\texttt{ulp}(r)$ sinon.

Erreurs d'arrondis

Définition : x étant un flottant positif, et y le plus petit flottant plus grand que x, on définit $\texttt{ulp}(x) = y - x$ (Unit in the Last Place).

Rq 1 : $\texttt{ulp}(x)$ est une puissance de 2 (ou ∞).
Rq 2 : Dans les cas normaux : $\texttt{ulp}(x) \simeq x2^{-53}$

Propriété : Pour toutes les opérations $+, -, \times, \div, \sqrt{}$,
la différence entre la valeur calculée r et la valeur exacte,
autrement dit, l'erreur d'arrondi, est majorée par :
$\texttt{ulp}(r)/2$ pour le mode d'arrondi au plus proche, et
$\texttt{ulp}(r)$ sinon.

Attention : Ceci n'est valable que pour des opérations prises une à une.

Quelques propriétés du calcul flottant

Il n'y a pas d'underflow pour $+, -$:
$a - b = 0 \Longleftrightarrow a = b$

Quelques propriétés du calcul flottant

Il n'y a pas d'underflow pour $+, -$:
$a - b = 0 \Longleftrightarrow a = b$

Détection des NaNs:
$a = a \Longleftrightarrow a$ n'est pas un NaN

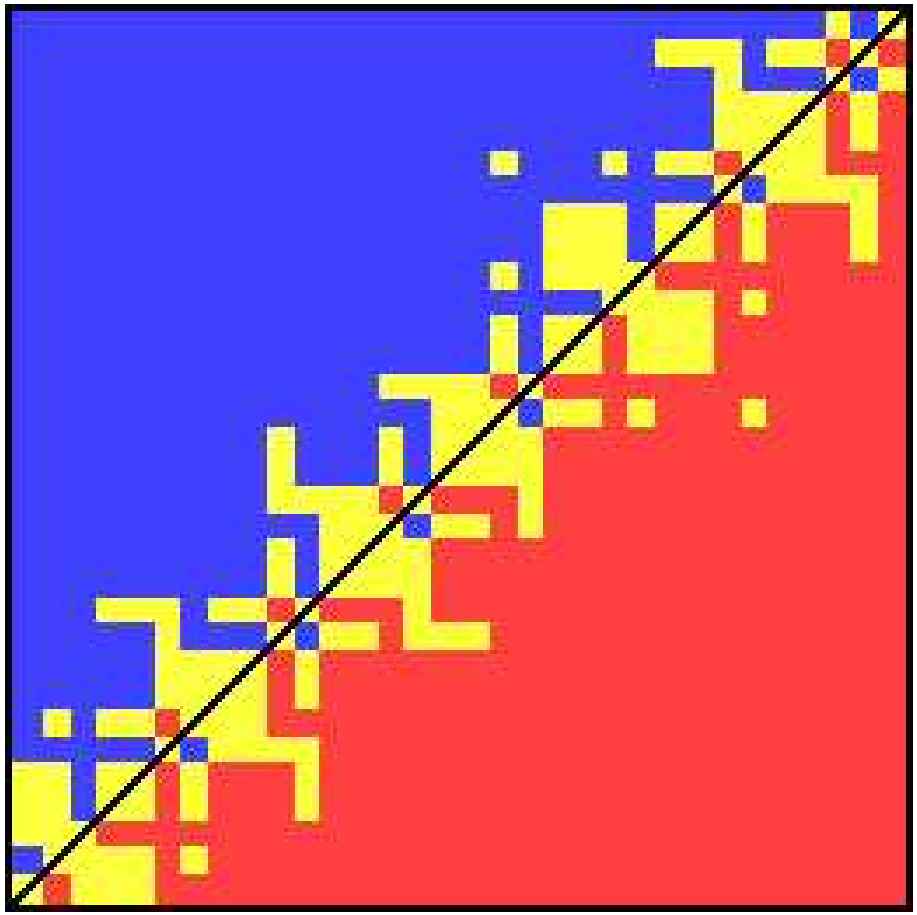
Quelques propriétés du calcul flottant

Il n'y a pas d'underflow pour $+, -$:
$a - b = 0 \Longleftrightarrow a = b$

Détection des NaNs:
$a = a \Longleftrightarrow a$ n'est pas un NaN

Monotonicité pour un mode d'arrondi donné:
$a + b <= c + d$ calculé $\Longleftrightarrow a + b <= c + d$ exact
(idem pour les autres opérations)

Géométrie du prédicat `orientation` approché



[Kettner-Mehlhorn-Schirra-P-Yap 04]

Calcul multi-précision

Précision multiple

Calcul exact sur des entiers ($\mathbb{Z}$) :

- O($n = \log N$) mémoire
- $+, -$: O(n) temps.
- $\times, \div$:

O(n^2)	si n petit
O($n^{\approx 1.5}$)	si n moyen (Karatsuba)
O($n \log n \log\log n$)	si n grand (Schönhage Strassen)

Précision multiple

Calcul exact sur des entiers ($\mathbb{Z}$) :

- O($n = \log N$) mémoire
- $+, -$: O(n) temps.
- $\times, \div$:

O(n^2)	si n petit
O($n^{\approx 1.5}$)	si n moyen (Karatsuba)
O($n \log n \log\log n$)	si n grand (Schönhage Strassen)

Évaluation exacte de polynômes sur des entrées entières de taille O(n) : $\geq$ O(nd)

Précision multiple

Calcul exact sur des entiers ($\mathbb{Z}$) :

- O($n = \log N$) mémoire
- $+, -$: O(n) temps.
- $\times, \div$:

O(n^2)	si n petit
O($n^{\approx 1.5}$)	si n moyen (Karatsuba)
O($n \log n \log\log n$)	si n grand (Schönhage Strassen)

Évaluation exacte de polynômes sur des entrées entières de taille O(n) : $\geq$ O(nd)

Bibliothèques : GMP, LEDA, CGAL, BigNum...

Multiplication de Karatsuba

On coupe les opérandes x et y en deux parties de tailles égales (bits de poids forts et faibles) :

forts	faibles
x_1	x_0

Multiplication de Karatsuba

On coupe les opérandes x et y en deux parties de tailles égales (bits de poids forts et faibles) :

forts	faibles
x_1	x_0

Soit b la puissance de 2 telle que $x = x_1 b + x_0$ et $y = y_1 b + y_0$. On voit que :

$$xy = (b^2 + b)x_1 y_1 - b(x_1 - x_0)(y_1 - y_0) + (b + 1)x_0 y_0$$

Donc, on utilise 3 multiplications de nombres de taille $n/2$ (au lieu de 4).

Multiplication de Karatsuba

On coupe les opérandes x et y en deux parties de tailles égales (bits de poids forts et faibles) :

forts	faibles
x_1	x_0

Soit b la puissance de 2 telle que $x = x_1 b + x_0$ et $y = y_1 b + y_0$. On voit que :

$$xy = (b^2 + b)x_1 y_1 - b(x_1 - x_0)(y_1 - y_0) + (b + 1)x_0 y_0$$

Donc, on utilise 3 multiplications de nombres de taille $n/2$ (au lieu de 4).

Complexité asymptotique : $O(n^{log(3)/log(2)=1.585})$

Multiplication de Karatsuba

On coupe les opérandes x et y en deux parties de tailles égales (bits de poids forts et faibles) :

forts	faibles
x_1	x_0

Soit b la puissance de 2 telle que $x = x_1 b + x_0$ et $y = y_1 b + y_0$. On voit que :

$$xy = (b^2 + b)x_1 y_1 - b(x_1 - x_0)(y_1 - y_0) + (b + 1)x_0 y_0$$

Donc, on utilise 3 multiplications de nombres de taille $n/2$ (au lieu de 4).

Complexité asymptotique : $O(n^{log(3)/log(2)=1.585})$

Pour en savoir plus : `http://www.swox.com/gmp/manual/Algorithms.html`

Calcul modulaire

Calcul modulaire

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

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

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

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);
}
```

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)

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

Calcul modulaire : calcul du signe (Lagrange)

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

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

Calcul modulaire : calcul du signe (Lagrange)

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

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

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

Calcul modulaire : calcul du signe (Lagrange)

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

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

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

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

Calcul modulaire : calcul du signe (Lagrange)

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

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

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

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 : 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

Nombres rationnels

Juste une paire d'entiers exacts : numérateur / dénominateur.

Attention : même l'addition double le nombre de bits !

On peut utiliser la normalisation (pas gratuit...) pour réduire la taille size :

- Soit on est chanceux (probabilité faible).
- Soit on a râté une simplification algébrique.
- Autres cas ?

Sinon : croissance exponentielle avec la profondeur des opérations.

Nombres à virgule flottante multi-précision

$m2^e$, où m et e sont des entiers multi-précision.

Nombres à virgule flottante multi-précision

$m2^e$, où m et e sont des entiers multi-précision.

On peut adjoindre une précision p à x telle que :

$m2^e - 2^p \leq x \leq m2^e + 2^p$

Nombres à virgule flottante multi-précision

$m2^e$, où m et e sont des entiers multi-précision.

On peut adjoindre une précision p à x telle que :

$m2^e - 2^p \leq x \leq m2^e + 2^p$

p peut être fixée à chaque opération, ou globalement.

p peut être propagée.

Nombres à virgule flottante multi-précision

$m2^e$, où m et e sont des entiers multi-précision.

On peut adjoindre une précision p à x telle que :

$m2^e - 2^p \leq x \leq m2^e + 2^p$

p peut être fixée à chaque opération, ou globalement.

p peut être propagée.

Bibliothèques : MPFR, CGAL::MP_Float.

Propagation d'erreurs

Soit (x, p_x) un nombre flottant multi-précision et une précision associée correspondant à un réel X. Idem pour (y, p_y).

Alors on peut obtenir une approximation de $X + Y$ par $(x + y, p_{x+y})$, où:

$$\begin{aligned} |(X - x) + (Y - y)| &<= |X - x| + |Y - y| \\ |(X - x) + (Y - y)| &<= 2^{p_x} + 2^{p_y} \\ |(X + Y) - (x + y)| &<= 2^{p_{x+y}} \end{aligned}$$

$$\Longrightarrow p_{x+y} = 1 + max(p_x, p_y)$$

Propagation d'erreurs

Soit (x, p_x) un nombre flottant multi-précision et une précision associée correspondant à un réel X. Idem pour (y, p_y).

Alors on peut obtenir une approximation de $X + Y$ par $(x + y, p_{x+y})$, où:

$$\begin{aligned} |(X - x) + (Y - y)| &<= |X - x| + |Y - y| \\ |(X - x) + (Y - y)| &<= 2^{p_x} + 2^{p_y} \\ |(X + Y) - (x + y)| &<= 2^{p_{x+y}} \end{aligned}$$

$$\Longrightarrow p_{x+y} = 1 + max(p_x, p_y)$$

Ceci est valable si $x + y$ n'est pas arrondi. Sinon, il faut en tenir compte.

Bornes de séparation

Calcul adaptatif et Bornes de séparations

Calcul entier exact : on calcule tous les bits de l'expression.

Si on ne veut que le signe (ou une approximation), on ne veut que les bits de poids forts.

Ex: calcul du signe : 1 bit de précision relative suffit.

Comment calculer juste les bits dont on a besoin, et pas beaucoup plus ?

Calcul adaptatif et Bornes de séparations

Calcul entier exact : on calcule tous les bits de l'expression.

Si on ne veut que le signe (ou une approximation), on ne veut que les bits de poids forts.

Ex: calcul du signe : 1 bit de précision relative suffit.

Comment calculer juste les bits dont on a besoin, et pas beaucoup plus ?

Évaluation paresseuse/adaptative :

Entrées de l'expression : entiers

Composée des opérations (DAG) : $+, -, \times, /, \sqrt[p]{}, \diamond$

Calcul adaptatif et Bornes de séparations

Calcul entier exact : on calcule tous les bits de l'expression.

Si on ne veut que le signe (ou une approximation), on ne veut que les bits de poids forts.

Ex: calcul du signe : 1 bit de précision relative suffit.

Comment calculer juste les bits dont on a besoin, et pas beaucoup plus ?

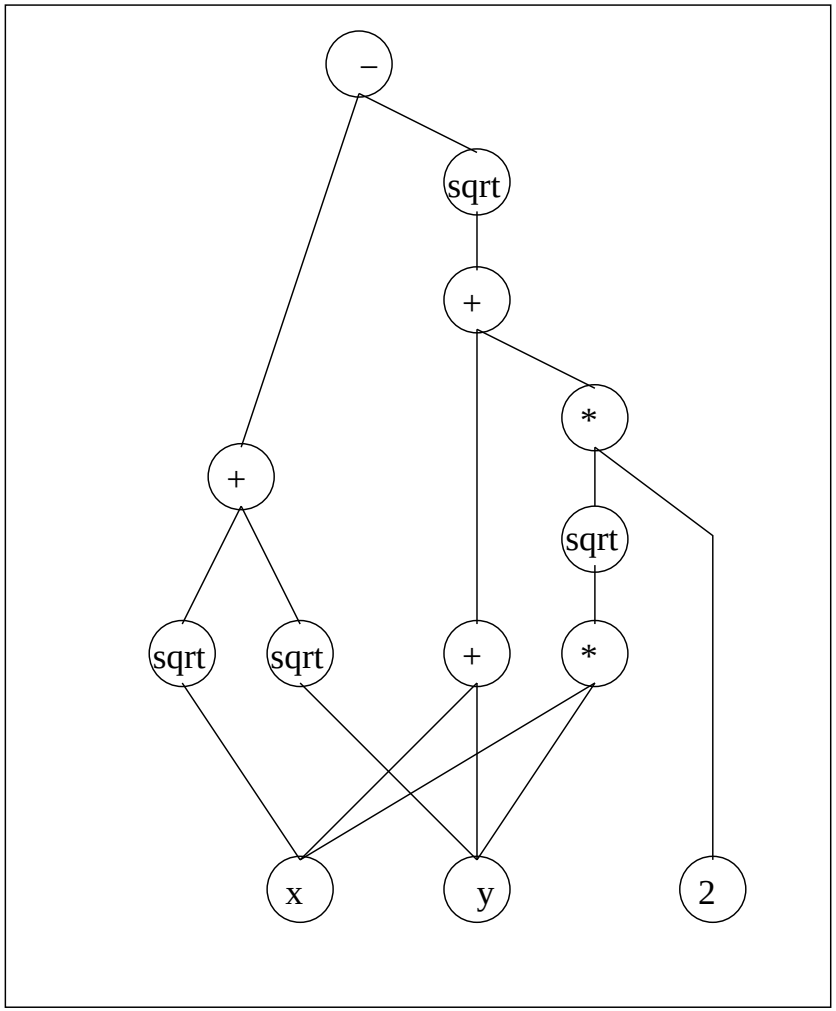
Évaluation paresseuse/adaptative :

Entrées de l'expression : entiers

Composée des opérations (DAG) : $+, -, \times, /, \sqrt[p]{}, \diamond$

Recherche en cours : nombres algébriques définis par leurs polynômes.

Exemple : $\texttt{signe}(\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}})$



Principe

On évalue l'expression numériquement à une certaine précision :

$$|E - E_p| < \epsilon_p$$

Si la précision est insuffisante pour déterminer le signe de manière certaine, on recommence en évaluant plus précisemment.

$$|E_p| > \epsilon_p ?$$

Principe

On évalue l'expression numériquement à une certaine précision :

$$|E - E_p| < \epsilon_p$$

Si la précision est insuffisante pour déterminer le signe de manière certaine, on recommence en évaluant plus précisemment.

$$|E_p| > \epsilon_p ?$$

Problème : si la valeur vaut exactement zéro, cette itération boucle indéfiniment.

Quand s'arrêter ?

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

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

Quand s'arrêter ?

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.

Quand s'arrêter ?

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
$E' \pm E''$	$u'\ell'' + \ell' u''$	$\ell'\ell''$
$E' \times E''$	$u'u''$	$\ell'\ell''$
$E' \div E''$	$u'\ell''$	$\ell' u''$
$\sqrt[p]{E'}$	$\min(\sqrt[p]{u'\ell'^{p-1}}, u')$	$\min(\ell', \sqrt[p]{u'^{p-1}\ell'})$
$\diamond(j, F_n, F_{n-1}, \ldots, F_0)$	...	...

On obtient une borne de séparation sur E par :

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

Exemple

$$\texttt{signe}(\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}})?$$

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 : $2^{-(28L+60)}$

Version binaire

On remarque que si $\frac{p}{q} \neq \frac{p'}{q'}$, alors :

$$\left|\frac{p}{q} - \frac{p'}{q'}\right| \geq \frac{1}{qq'}$$

Version binaire

On remarque que si $\frac{p}{q} \neq \frac{p'}{q'}$, alors :

$$\left|\frac{p}{q} - \frac{p'}{q'}\right| \geq \frac{1}{qq'}$$

Mais si, de plus, $q = 2^n$ et $q' = 2^{n'}$, alors :

$$\left|\frac{p}{q} - \frac{p'}{q'}\right| \geq \frac{1}{\max(q, q')}$$

Version binaire

On remarque que si $\frac{p}{q} \neq \frac{p'}{q'}$, alors :

$$\left|\frac{p}{q} - \frac{p'}{q'}\right| \geq \frac{1}{qq'}$$

Mais si, de plus, $q = 2^n$ et $q' = 2^{n'}$, alors :

$$\left|\frac{p}{q} - \frac{p'}{q'}\right| \geq \frac{1}{\max(q, q')}$$

De même pour des puissances de k quelconques.

En pratique, les entrées sont souvent des nombres :

- flottants $(k = 2)$
- décimaux $(k = 2, 5)$

BFMSS[k]

Idée générale :

extraire une "plus grande" puissance de k possible de $E = E' k^{v(E)}$.

On obtiendra une borne de séparation :

$$B(E) = B(E') k^{v(E)}$$

BFMSS[k]

Idée générale :

extraire une "plus grande" puissance de k possible de $E = E' k^{v(E)}$.

On obtiendra une borne de séparation :

$$B(E) = B(E') k^{v(E)}$$

Pour BFMSS, on sépare en deux : $v = v^+ - v^-$ (avec $v^+ \geq 0, v^- \geq 0$).

$$U = k^{v^+} U'$$
$$L = k^{v^-} L'$$

BFMSS[k=2], règles

On applique les règles suivantes :

E	$u_2 = u_2(E)$	$\ell_2 = \ell_2(E)$
binary rational $n2^m$	$\lvert n\rvert$	1
$E' \pm E''$	$2^{v'^+ + v''^- - v^+} u'_2 \ell''_2$ $+2^{v'^- + v''^+ - v^+} \ell'_2 u''_2$	$\ell'_2 \ell''_2$
$E' \times E''$	$u'_2 u''_2$	$\ell'_2 \ell''_2$
$E' \div E''$	$u'_2 \ell''_2$	$\ell'_2 u''_2$
$\sqrt[p]{E'}$, $2^{v'} u'_2 \geq \ell'_2$	$\sqrt[p]{2^{\widetilde{v} - pv^+} u'_2 \ell'^{p-1}_2}$	ℓ'_2
$\sqrt[p]{E'}$, $2^{v'} u'_2 < \ell'_2$	u'_2	$\sqrt[p]{2^{\widetilde{v} - pv^-} u'^{p-1}_2 \ell'_2}$

E	$v^+ = v^+(E)$	$v^- = v^-(E)$
binary rational $n2^m$	$\max(0, m)$	$\max(0, -m)$
$E' \pm E''$	$\min(v'^+ + v''^-,$ $v'^- + v''^+)$	$v'^- + v''^-$
$E' \times E''$	$v'^+ + v''^+$	$v'^- + v''^-$
$E' \div E''$	$v'^+ + v''^-$	$v'^- + v''^+$
$\sqrt[p]{E'}$, $2^{v'} u'_2 \geq \ell'_2$	$\lfloor \widetilde{v}/p \rfloor$ where $\widetilde{v} = v'^+ + (p-1)v'^-$	v'^-
$\sqrt[p]{E'}$, $2^{v'} u'_2 < \ell'_2$	v'^+	$\lfloor \widetilde{v}/p \rfloor$ where $\widetilde{v} = (p-1)v'^+ + v'^-$

Comparaison

$$\frac{B_2(E)}{B(E)} = 2^{v^+ D(E)}$$

Exemple : $\texttt{signe}(\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}})$?

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

Bornes de séparation :

- BFMSS : $2^{-(96L+60)}$
- Li-Yap : $2^{-(28L+60)}$
- BFMSS[2] : $2^{-(7{,}5L+cste)}$

Déterminant

Matrice aléatoire avec entrées $x2^{-100}$, $|x| < 2^{100}$.

Déterminant calculé avec la méthode de programmation dynamique.

Bornes de séparations obtenues :

n	$(n!)nL$	BFMSS bitbound	nL	BFMSS[2] bitbound
2	400	164	200	101
3	1800	657	300	169
4	9600	2267	400	248
5	60,000	10,468	500	326

Implantation

Bibliothèques : CORE et LEDA.

CORE implante :

- $+, -, \times, /, \sqrt{}$
- Fournit la meilleure borne parmi BFMSS[2], Li-Yap, Degree-Measure.
- Utilise GMP
- Utilise des filtres dynamiques pour éliminer les cas faciles.

Recherche actuelle : $\diamond$ operator

Rajouter l'opération : extraction de racine de polynôme

- Uspensky/Descartes/Sturm pour isoler les racines

- Itérations de Newton pour obtenir plus de précision

Recherche actuelle : $\diamond$ operator

Rajouter l'opération : extraction de racine de polynôme

- Uspensky/Descartes/Sturm pour isoler les racines

- Itérations de Newton pour obtenir plus de précision

Prototypes commencent à marcher. À venir : problèmes d'efficacité

- Quel pourrait être une bon filtre ? Analyse par intervalles ?

Remarque : coût de la représentation symbolique

Construire et stocker le DAG dans chaque prédicat a un coût.

Il est évité si on considère l'analyse d'un prédicat dans son ensemble.

L'information est alors "stockée dans le code".

L'autre extrême : les filtres

Optimiser les cas faciles

Bornes de séparation : traitent les cas les pires.

Cas attendu le plus souvent : cas "facile", à optimiser.

Optimiser les cas faciles

Bornes de séparation : traitent les cas les pires.

Cas attendu le plus souvent : cas "facile", à optimiser.

Contrôler les erreurs d'arrondis du calcul flottant $\Rightarrow$ on ne fait plus que rarement du calcul exact coûteux

Dans les "bons cas", on obtient une solution géométriquement exacte pour le coût du calcul flottant.

Filtres dynamiques : arithmétique d'intervalles

Idée : on remplace chaque opération flottante par une opération sur un intervalle de flottants $[\underline{x}; \overline{x}]$ qui encode l'erreur d'arrondi.

Propriété d'inclusion : à chaque opération, l'intervalle contient la valeur exacte X.

Filtres dynamiques : arithmétique d'intervalles

Idée : on remplace chaque opération flottante par une opération sur un intervalle de flottants $[\underline{x}; \overline{x}]$ qui encode l'erreur d'arrondi.

Propriété d'inclusion : à chaque opération, l'intervalle contient la valeur exacte X.

Opérations : on utilise les modes d'arrondis IEEE 754 :

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

Filtres dynamiques : arithmétique d'intervalles

Idée : on remplace chaque opération flottante par une opération sur un intervalle de flottants $[\underline{x}; \overline{x}]$ qui encode l'erreur d'arrondi.

Propriété d'inclusion : à chaque opération, l'intervalle contient la valeur exacte X.

Opérations : on utilise les modes d'arrondis IEEE 754 :

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

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

Filtres dynamiques : arithmétique d'intervalles

Idée : on remplace chaque opération flottante par une opération sur un intervalle de flottants $[\underline{x}; \overline{x}]$ qui encode l'erreur d'arrondi.

Propriété d'inclusion : à chaque opération, l'intervalle contient la valeur exacte X.

Opérations : on utilise les modes d'arrondis IEEE 754 :

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

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

Optimisation :

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

Moins de changements de modes d'arrondis.

Multiplication et division d'intervalles

Multiplication :

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

En pratique, on utilise des tests pour les différents cas avant de faire les multiplications.

Multiplication et division d'intervalles

Multiplication :

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

En pratique, on utilise des tests pour les différents cas avant de faire les multiplications.

Division : similaire.

Traitement de la division par zéro.

Comparaisons

Grâce à la propriété d'inclusion, si

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

alors on sait décider si $X < Y$ ou $X > Y$.

Comparaisons

Grâce à la propriété d'inclusion, si

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

alors on sait décider si $X < Y$ ou $X > Y$.

Sinon, on ne peut pas décider de la comparaison.

$\Longrightarrow$ Échec du filtre

Filtres statiques

Analyse statique de la propagation d'erreur d'arrondi sur l'evaluation d'un polynôme, en supposant des bornes sur les entrées.

Filtres statiques

Analyse statique de la propagation d'erreur d'arrondi sur l'evaluation d'un polynôme, en supposant des bornes sur les entrées.

Notations : x est une variable réelle, $\mathtt{x}$ sa valeur calculée avec des doubles, $\mathtt{e_x}$ et $\mathtt{b_x}$ sont des doubles tels que :

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

Filtres statiques

Analyse statique de la propagation d'erreur d'arrondi sur l'evaluation d'un polynôme, en supposant des bornes sur les entrées.

Notations : x est une variable réelle, $\mathtt{x}$ sa valeur calculée avec des doubles, $\mathtt{e_x}$ et $\mathtt{b_x}$ sont des doubles tels que :

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

Initialement, on peut obtenir par un arrondi au plus proche (si les valeurs ne tiennent pas exactement dans un `double`) :

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

Addition et soustraction

La propagation d'erreur sur une addition $z = x + y$ est la suivante :

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

En effet :

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

Multiplication

La propagation d'erreur sur une multiplication $z = x \times y$ est la suivante :

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

En effet :

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

Application : prédicat `orientation`

Code flottant approché :

```
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 : prédicat `orientation`

Code avec filtre statique (pour des entrées bornées par 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
}
```

Variantes : Ex : calcul de la borne à l'exécution

```
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
}
```

Probabilités d'échec des filtres

Étude théorique : [Devillers-Preparata-99]

Entrées uniformément distribuées dans un carré/cube unitaire :

`orientation 2D`	10^{-15}
`orientation 3D`	5.10^{-14}
`cocircularité 2D`	10^{-11}
`cosphéricité 3D`	7.10^{-10}

Probabilités d'échec des filtres

Étude théorique : [Devillers-Preparata-99]

Entrées uniformément distribuées dans un carré/cube unitaire :

`orientation 2D`	10^{-15}
`orientation 3D`	5.10^{-14}
`cocircularité 2D`	10^{-11}
`cosphéricité 3D`	7.10^{-10}

... pour des données distribuées de manière homogène.

Sur des cas plus dégénérés

	Dynamique	Semi-statique
Aléatoire	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
Grille	49756	299505

Nombres d'échecs des filtres dynamiques et statiques pendant le calcul de Delaunay (10^5 points). Entrées sur une grille entière sur 30 bits, avec perturbation relative.

Comparison : filtres dynamique vs statique

Peut échouer plus souvent que l'arithmétique d'intervalles (moins précis), mais plus rapide.

Filtres statiques plus difficiles à faire : nécessite une analyse de chaque prédicat.

Comparison : filtres dynamique vs statique

Peut échouer plus souvent que l'arithmétique d'intervalles (moins précis), mais plus rapide.

Filtres statiques plus difficiles à faire : nécessite une analyse de chaque prédicat.

Schéma le plus rapide : cascader plusieurs méthodes.

Filtres : remarques

Fragile : essayer d'éviter des mauvais cas dans les algorithmes !

- Éviter les calculs cascadés (se baser sur les données initiales)

- Éviter de tester les cas dégénérés que vous connaissez (créés par l'algorithme).

- Éviter les constructions, car il y a de meilleures solutions pour les prédicats.

Travaux actuels

- Génération automatique de code, à partir de la version générique, pour les meilleures méthodes.
- Filtrage des constructions géométriques.
- Rounding des constructions.

Implantation dans CGAL

Algorithmes et classes de traits

Les algorithmes sont paramétrés (templates) par des traits classes,
qui leur fournissent:

- les types d'objets que l'algorithme manipule : Point_2, Tetrahedron_3...

Algorithmes et classes de traits

Les algorithmes sont paramétrés (templates) par des traits classes,
qui leur fournissent:

- les types d'objets que l'algorithme manipule : Point_2, Tetrahedron_3...
- les prédicats que l'algorithme applique dessus : Orientation_2, Side_of_oriented_sphere_3...

Algorithmes et classes de traits

Les algorithmes sont paramétrés (templates) par des traits classes, qui leur fournissent:

- les types d'objets que l'algorithme manipule : Point_2, Tetrahedron_3...
- les prédicats que l'algorithme applique dessus : Orientation_2, Side_of_oriented_sphere_3...
- les constructions : Mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

Les deux derniers sont fournis sous forme de functors (classes avec un function operator `operator()`), ce qui permet de passer un état.

Algorithmes et classes de traits

Les algorithmes sont paramétrés (templates) par des traits classes, qui leur fournissent:

- les types d'objets que l'algorithme manipule : Point_2, Tetrahedron_3...
- les prédicats que l'algorithme applique dessus : Orientation_2, Side_of_oriented_sphere_3...
- les constructions : Mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

Les deux derniers sont fournis sous forme de functors (classes avec un function operator `operator()`), ce qui permet de passer un état.

On décrit les besoins d'un algorithm envers son paramètre de traits class comme un concept, dont une instance particulière possible est un modèle.

Noyaux

Le noyau rassemble beaucoup de types d'objets, de prédicats et de constructions, et peut servir de paramètre de traits classes directement à de nombreux algorithmes. Certains algorithmes doivent définir leur propre traits class.

Noyaux

Le noyau rassemble beaucoup de types d'objets, de prédicats et de constructions, et peut servir de paramètre de traits classes directement à de nombreux algorithmes. Certains algorithmes doivent définir leur propre traits class.

Noyaux classiques, paramétrés par des types de nombres :
Cartesian<FT>
Homogeneous<RT>

Noyaux

Le noyau rassemble beaucoup de types d'objets, de prédicats et de constructions, et peut servir de paramètre de traits classes directement à de nombreux algorithmes. Certains algorithmes doivent définir leur propre traits class.

Noyaux classiques, paramétrés par des types de nombres :
Cartesian<FT>
Homogeneous<RT>

Ex : Triangulation_3<Cartesian<double> >

Noyaux

Le noyau rassemble beaucoup de types d'objets, de prédicats et de constructions, et peut servir de paramètre de traits classes directement à de nombreux algorithmes. Certains algorithmes doivent définir leur propre traits class.

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.

Noyaux

Le noyau rassemble beaucoup de types d'objets, de prédicats et de constructions, et peut servir de paramètre de traits classes directement à de nombreux algorithmes. Certains algorithmes doivent définir leur propre traits class.

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 également accessible directement via des fonctions globales : `CGAL::orientation(p, q, r)`..

Types de nombres

Paramètres valides pour les noyaux Cartesian...

Types de nombres

Paramètres valides pour les noyaux Cartesian...

Flottants :
double, float

Types de nombres

Paramètres valides pour les noyaux Cartesian...

Flottants :
double, float

Multi-précision :
Gmpz, Gmpq, CGAL::MP_Float, leda::integer...

Types de nombres

Paramètres valides pour les noyaux Cartesian...

Flottants :
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

Outils internes

Arithmétique d'intervalles: CGAL::Interval_nt, boost::interval

Générateur de prédicat filtré (dynamique) utilisant les exceptions C++ : CGAL::Filtered_predicate<>

Noyaux filtrés

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

Noyaux filtrés

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

Noyaux recommandés :
CGAL::Exact_predicates_exact_constructions_kernel
CGAL::Exact_predicates_inexact_constructions_kernel