

Robustesse des algorithmes géométriques: des outils arithmétiques à l'interface utilisateur dans CGAL

Sylvain Pion

Équipe-Projet GEOMETRICA
INRIA Sophia Antipolis

27 Octobre 2009

Plan

- 1 Introduction
- 2 Algorithmes et primitives
- 3 Robustesse
- 4 Arithmétique
- 5 Implantation dans CGAL
- 6 Conclusion

Plan

- 1 Introduction
- 2 Algorithmes et primitives
- 3 Robustesse
- 4 Arithmétique
- 5 Implantation dans CGAL
- 6 Conclusion

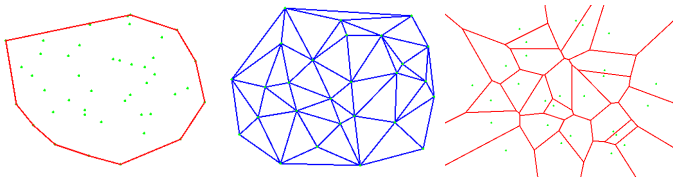
Géométrie Algorithmique

- Domaine de recherche actif depuis 25 ans
- Algorithmes manipulant des objets géométriques en grand nombre
- Emphase sur la complexité asymptotique (modèle Real-RAM)

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

Exemples

- Enveloppes convexes, triangulations, diagrammes de Voronoï



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

CGAL : *Computational Geometry Algorithms Library*

Since 1995 : mise en oeuvre des algorithmes géométriques,
pour un usage industriel et académique.

- Buts : adaptabilité, efficacité, robustesse
- Relecture des spécifications par un Editorial Board
- Langage : C++, programmation générique
- v3.5 : 700.000 lignes de code, 10.000 téléchargements/an
- Open Source : LGPL / QPL, et commercialisé par GeometryFactory depuis 2003

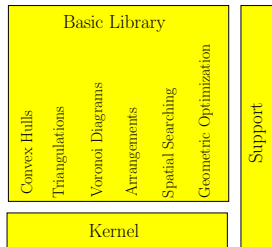
CGAL : *Computational Geometry Algorithms Library*

Since 1995 : mise en oeuvre des algorithmes géométriques,
pour un usage industriel et académique.

- Buts : adaptabilité, efficacité, robustesse
- Relecture des spécifications par un Editorial Board
- Langage : C++, programmation générique
- v3.5 : 700.000 lignes de code, 10.000 téléchargements/an
- Open Source : LGPL / QPL, et commercialisé par GeometryFactory depuis 2003

CGAL : Architecture

Architecture principale : algorithmes géométriques génériques, et noyaux



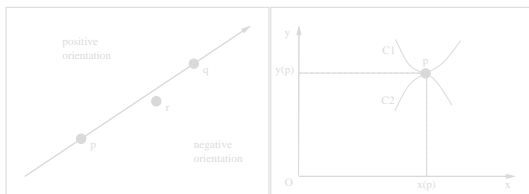
Noyau de primitives géométriques

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...



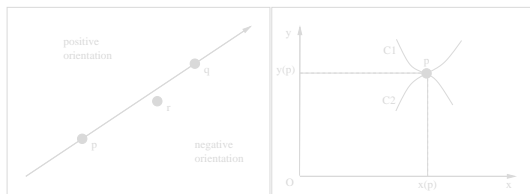
Noyau de primitives géométriques

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...



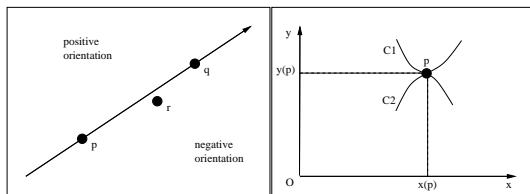
Noyau de primitives géométriques

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...

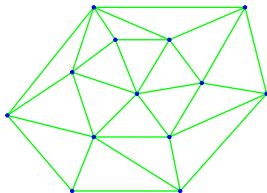


Plan

- 1 Introduction
- 2 Algorithmes et primitives**
- 3 Robustesse
- 4 Arithmétique
- 5 Implantation dans CGAL
- 6 Conclusion

Triangulation de Delaunay

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

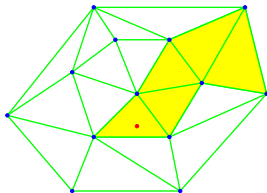


Localisation : prédicat **orientation**(p, q, r), signe de :

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

Triangulation de Delaunay

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

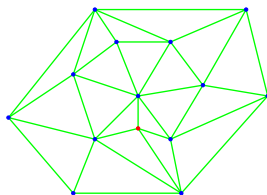
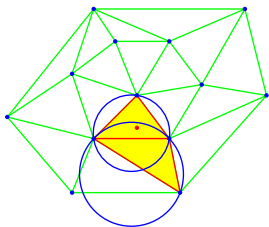


Localisation : prédicat **orientation**(p, q, r), signe de :

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

Triangulation de Delaunay

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

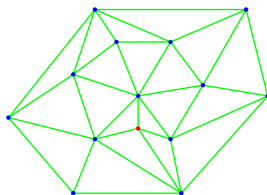
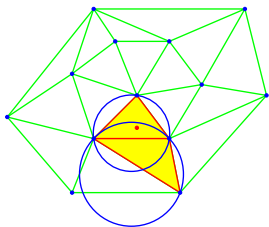


Mise à jour : prédicat `in_circle(p, q, r, s)`, signe de :

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

Triangulation de Delaunay

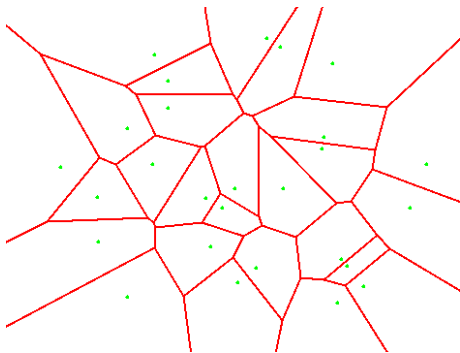
Algorithme incrémental en 2 étapes : **localisation** et **mise à jour**.



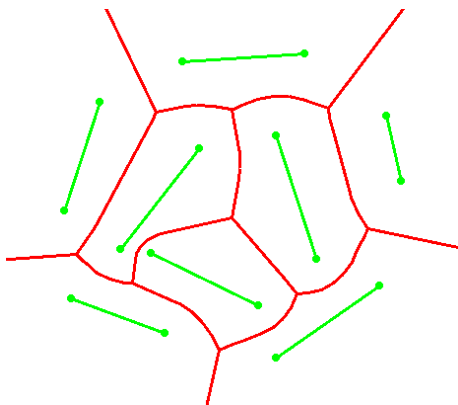
Mise à jour : prédicat **in_circle(p, q, r, s)**, signe de :

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

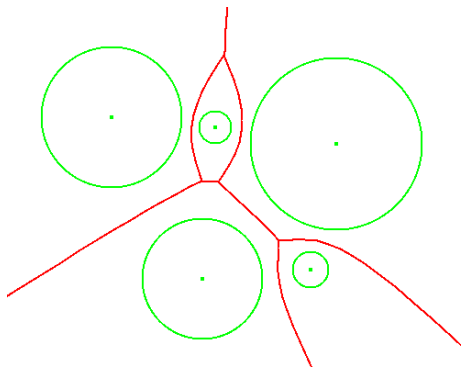
Diagrammes de Voronoï de points



Diagrammes de Voronoï de segments



Diagrammes de Voronoï de cercles

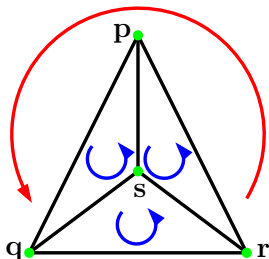


Plan

- 1 Introduction
- 2 Algorithmes et primitives
- 3 Robustesse**
- 4 Arithmétique
- 5 Implantation dans CGAL
- 6 Conclusion

Robustesse

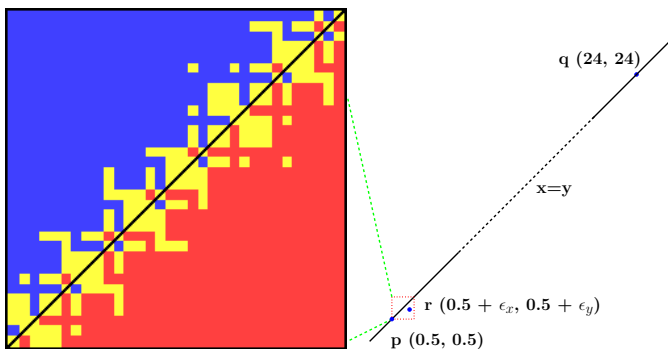
Les algorithmes reposent sur des théorèmes mathématiques, comme :



$$\begin{aligned} \text{orientation}(s, q, r) &> 0 \\ \text{orientation}(p, s, r) &> 0 \\ \text{orientation}(p, q, s) &> 0 \end{aligned} \quad \Rightarrow \quad \text{orientation}(p, q, r) > 0$$

Robustesse

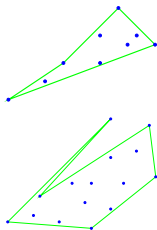
Exemple où la **géométrie flottante** diffère de la géométrie réelle :
orientation de points presque alignés.



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

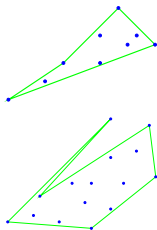
Conséquences possibles sur les algorithmes

- 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.
- Signes de polynômes sur des entrées flottantes :
besoin de $+$, $-$, \times exacts sur des flottants multiprécision.

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.
- Signes de polynômes sur des entrées flottantes :
besoin de $+$, $-$, \times exacts sur des flottants multiprécision.

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.
- Signes de polynômes sur des entrées flottantes :
besoin de $+$, $-$, \times exacts sur des flottants multiprécision.

Plan

- 1 Introduction
- 2 Algorithmes et primitives
- 3 Robustesse
- 4 Arithmétique**
- 5 Implantation dans CGAL
- 6 Conclusion

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]

Prédicats filtrés

Accélération des prédicats exacts par un filtre :

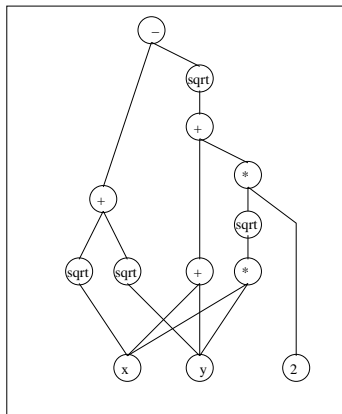
- évaluation flottante avec **certificat**
- arithmétique multiprécision uniquement si besoin

Exemples

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

Arithmétique paresseuse (types de nombres filtrés)

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



Approximation et raffinement à la demande.

[Cas rationnel : Michelucci, Moreau'95]

Bornes de séparation

Définition

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

$$|E| < B(E) \Rightarrow E = 0$$

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 :

$$|E| < B(E) \Rightarrow E = 0$$

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 :

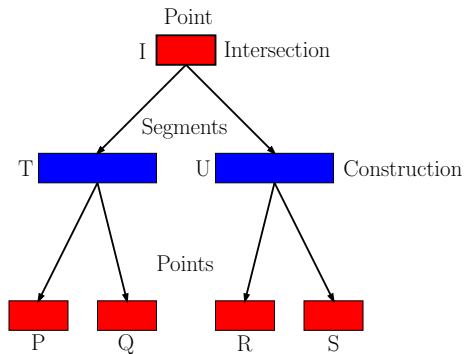
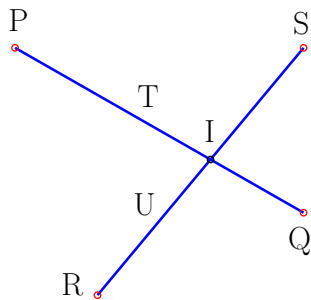
- Burnikel, Funke, Mehlhorn, Schirra, Schmitt "BFMSS" [2002]
- Li, Yap [2001]
- Mignotte "Degree-Measure" [1982]

Constructions géométriques filtrées

Optimisation supplémentaire : stockage en mémoire des objets géométriques

Intérêt : regrouper les calculs, moins de mémoire, généricité

[Fabri,P'06]



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.

Plan

- 1 Introduction
- 2 Algorithmes et primitives
- 3 Robustesse
- 4 Arithmétique
- 5 Implantation dans CGAL**
- 6 Conclusion

Algorithmes, classes de traits, noyaux

Les **algorithmes** sont paramétrés par :

Classes de traits géométriques

- types d'objets : `Point_2`, `Tetrahedron_3`...
- foncteurs prédicats : `Orientation_2`, `Side_of_oriented_sphere_3`...
- foncteurs constructions : `Construct_mid_point_2`, `Construct_circumcenter_3`...

Noyaux : modèles de nombreuses classes de traits.

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

Exemple : `Triangulation_3<Cartesian<double>>`

Algorithmes, classes de traits, noyaux

Les **algorithmes** sont paramétrés par :

Classes de traits géométriques

- types d'objets : `Point_2`, `Tetrahedron_3`...
- foncteurs prédicats : `Orientation_2`, `Side_of_oriented_sphere_3`...
- foncteurs constructions : `Construct_mid_point_2`, `Construct_circumcenter_3`...

Noyaux : modèles de nombreuses classes de traits.

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

Exemple : `Triangulation_3<Cartesian<double>>`

Combinaisons noyau/arithmétique

Paramètres valides pour le noyau `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) : `CGAL : :Filtered_predicate<>`

Noyaux filtrés :

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

Combinaisons noyau/arithmétique

Paramètres valides pour le noyau `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) : `CGAL : :Filtered_predicate<>`

Noyaux filtrés :

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

Combinaisons noyau/arithmétique

Paramètres valides pour le noyau `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) : `CGAL : :Filtered_predicate<>`

Noyaux filtrés :

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

Exemple : un prédicat

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

    CGAL::Orientation
    operator()(Point_2 p, Point_2 q, 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 CGAL::sign( prx*qry - qrx*pry );
    }
};
```

Exemple : utilisation du prédicat

```
typedef CGAL::Cartesian<double> Kernel;  
Kernel::Point_2 p(1,2), q(2,3), r(4,5);  
My_orientation_2<Kernel> orientation;  
CGAL::Orientation o = orientation(p,q,r);
```

Exemple : utilisation de `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);
```

Génération automatique de filtres statiques

Outil : FPG - Filtered Predicate Generator

[Meyer,P'08]

- Source : sous-ensemble du langage C, annoté
- Méthode : filtres semi-statiques utilisant l'homogénéité des expressions
- Formellement prouvé sur quelques cas avec Gappa [Melquiond,P'05]

Encore relativement peu de cas traités faute d'automatisation/intégration.

Exemple : Code source traité par FPGA

```
inline double determinant( double a00, double a01,
                          double a10, double a11 )
{
    return a00*a11 - a10*a01;
}

int
orientation(double px, double py,
            double qx, double qy,
            double rx, double ry)
group px qx rx;
group py qy ry;
{
    return sign(determinant( qx-px, qy-py, rx-px, ry-py ));
}
```

Exemple : Code produit par FPG

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

    // Semi-static filter.
    double maxx = std::fabs(pqx);
    if (maxx < std::fabs(prx)) maxx = std::fabs(prx);
    double maxy = std::fabs(pqy);
    if (maxy < std::fabs(pry)) maxy = std::fabs(pry);
    double eps = 8.8872057372592798e-16 * maxx * maxy;

    if (maxx > maxy) std::swap(maxx, maxy); // Sort them

    // Protect against underflow in the computation of eps.
    if (maxx < 1e-146) /* sqrt(min_double/eps) */ {
        if (maxx == 0)
            return 0;
    }
    // Protect against overflow in the computation of det.
    else if (maxy < 1e153) /* sqrt(max_double [hadamard]/2) */ {
        if (det > eps) return 1;
        if (det < -eps) return -1;
    }
    return UNKNOWN;
}
```

Plan

- 1 Introduction
- 2 Algorithmes et primitives
- 3 Robustesse
- 4 Arithmétique
- 5 Implantation dans CGAL
- 6 Conclusion**

Conclusion et travaux futurs

- Importance/difficulté de la robustesse et de l'efficacité des algo. géométriques
- Besoins multiples au niveau de l'arithmétique
- Intégration des filtres statiques : travail avec les compilateurs et les langages, connections avec les assistants de preuves si possible
- Standardisation et optimisation de l'arithmétique d'intervalles
- Constructions géométriques (composition)

Des questions ?

Conclusion et travaux futurs

- Importance/difficulté de la robustesse et de l'efficacité des algo. géométriques
- Besoins multiples au niveau de l'arithmétique
- Intégration des filtres statiques : travail avec les compilateurs et les langages, connections avec les assistants de preuves si possible
- Standardisation et optimisation de l'arithmétique d'intervalles
- Constructions géométriques (composition)

Des questions ?