Robustesse, efficacité et généricité dans la bibliothèque CGAL

Sylvain Pion

Équipe-Projet GEOMETRICA
INRIA Sophia Antipolis - Méditerranée

30 Janvier 2008

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, géologie, biologie moléculaire, médical...

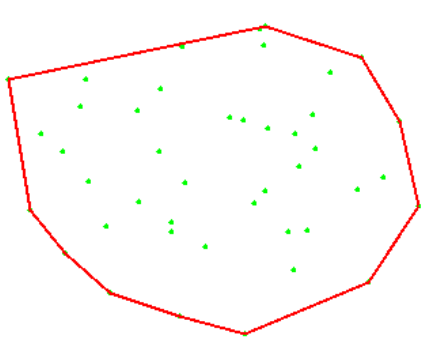
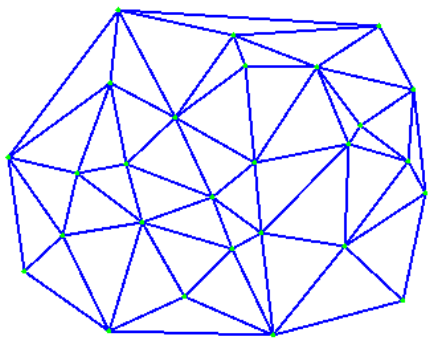
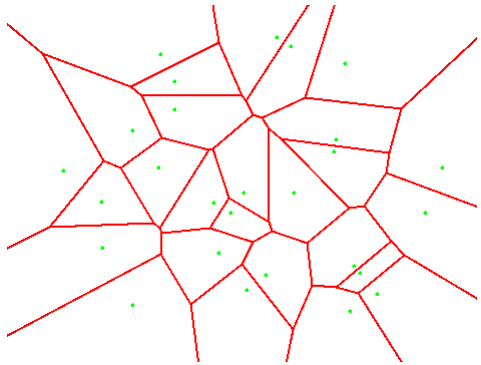
Géométrie Algorithmique

- 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, géologie, 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 géométriques,
pour un public industriel et académique.

- Critères : adaptabilité, efficacité, robustesse
- Vérification de la qualité des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.3 : 100 modules, 600.000 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 géométriques,
pour un public industriel et académique.

- Critères : adaptabilité, efficacité, robustesse
- Vérification de la qualité des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.3 : 100 modules, 600.000 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 géométriques,
pour un public industriel et académique.

- Critères : adaptabilité, efficacité, robustesse
- Vérification de la qualité des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.3 : 100 modules, 600.000 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 géométriques,
pour un public industriel et académique.

- Critères : adaptabilité, efficacité, robustesse
- Vérification de la qualité des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.3 : 100 modules, 600.000 lignes, 10.000 téléchargements/an
- Open Source : LGPL et QPL (commercialisé par GeometryFactory depuis 2003)

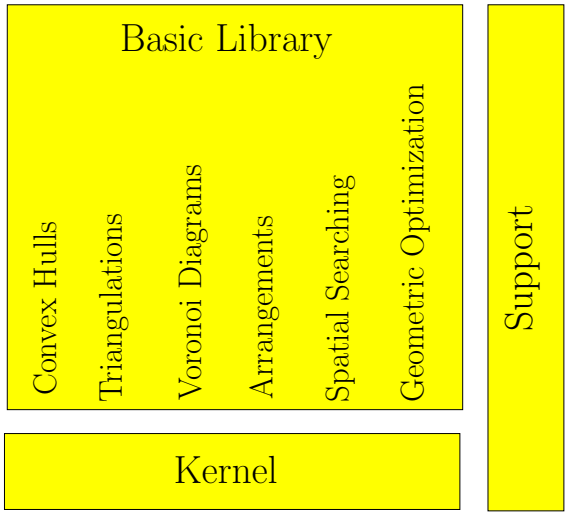
CGAL : *Computational Geometry Algorithms Library*

Depuis 1995 : mise en oeuvre des algorithmes géométriques,
pour un public industriel et académique.

- Critères : adaptabilité, efficacité, robustesse
- Vérification de la qualité des contributions par un Editorial Board
- Langage choisi : C++ (programmation générique)
- v3.3 : 100 modules, 600.000 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...

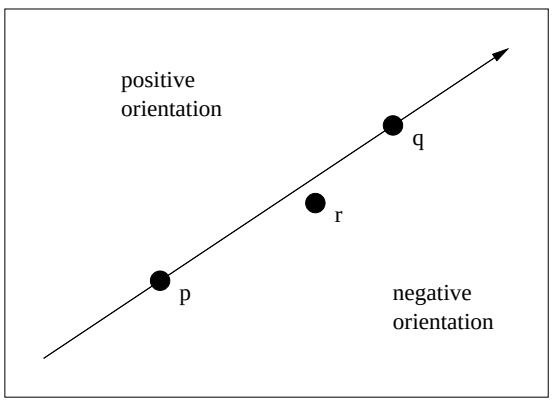
Noyau de primitives géométriques

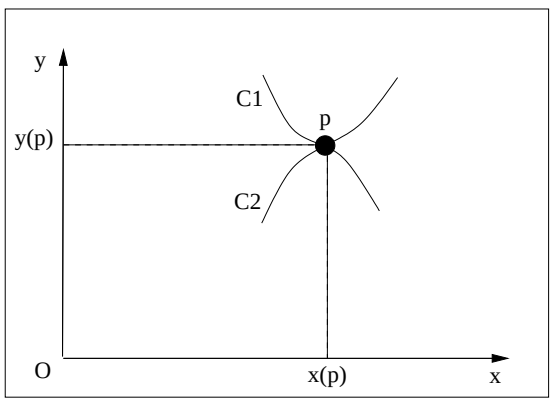
Les algorithmes se découpent logiquement en :

- 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...

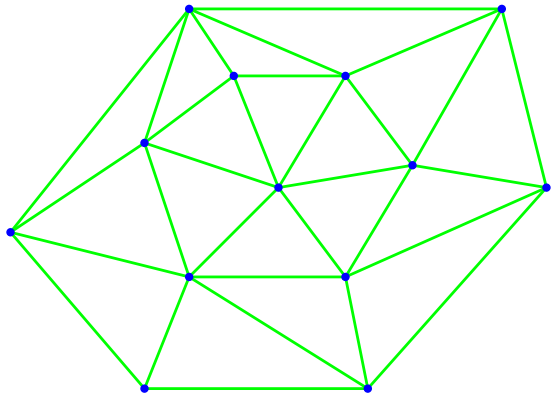




Plan

Triangulation de Delaunay

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

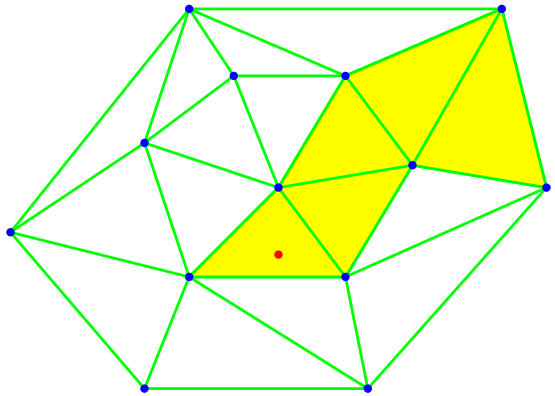


Localisation : prédicat orientation(p, q, r), signe de :

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

Triangulation de Delaunay

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



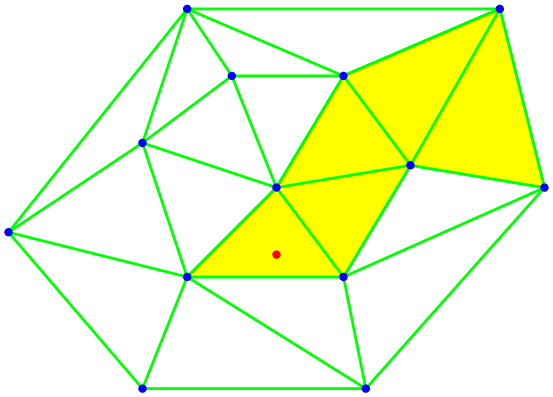
Localisation : prédicat orientation(p, q, r), signe de :

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

INRIA

Triangulation de Delaunay

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

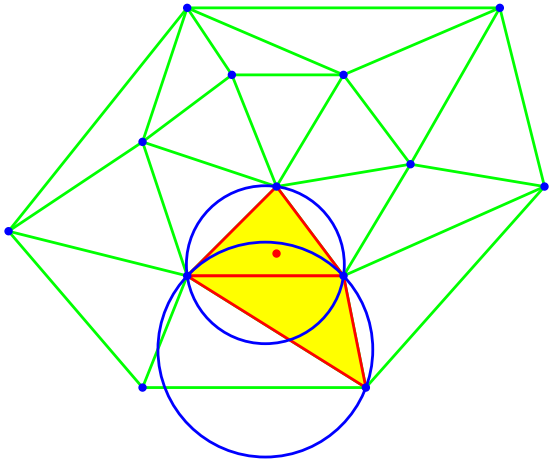


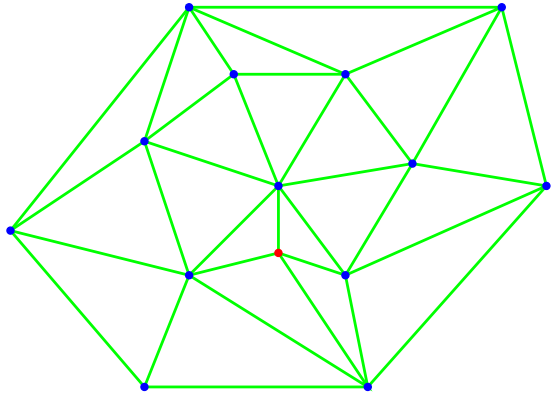
Localisation : prédicat orientation(p, q, r), signe de :

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

Triangulation de Delaunay

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





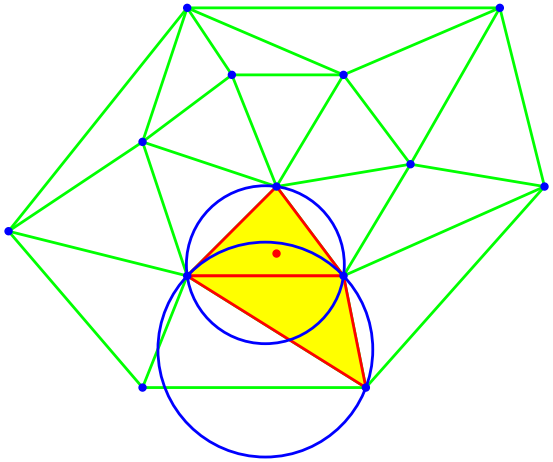
Mise à jour : prédicat in_circle(p, q, r, s), signe de :

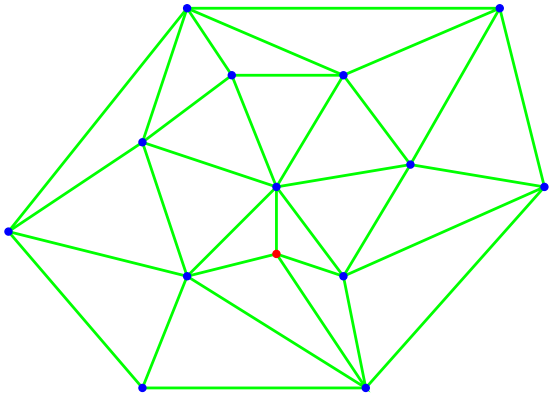
$$\begin{vmatrix} 1 & px & py & px^2+py^2 \\ 1 & qx & qy & qx^2+qy^2 \\ 1 & rx & ry & rx^2+ry^2 \\ 1 & sx & sy & sx^2+sy^2 \end{vmatrix}$$

INRIA

Triangulation de Delaunay

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



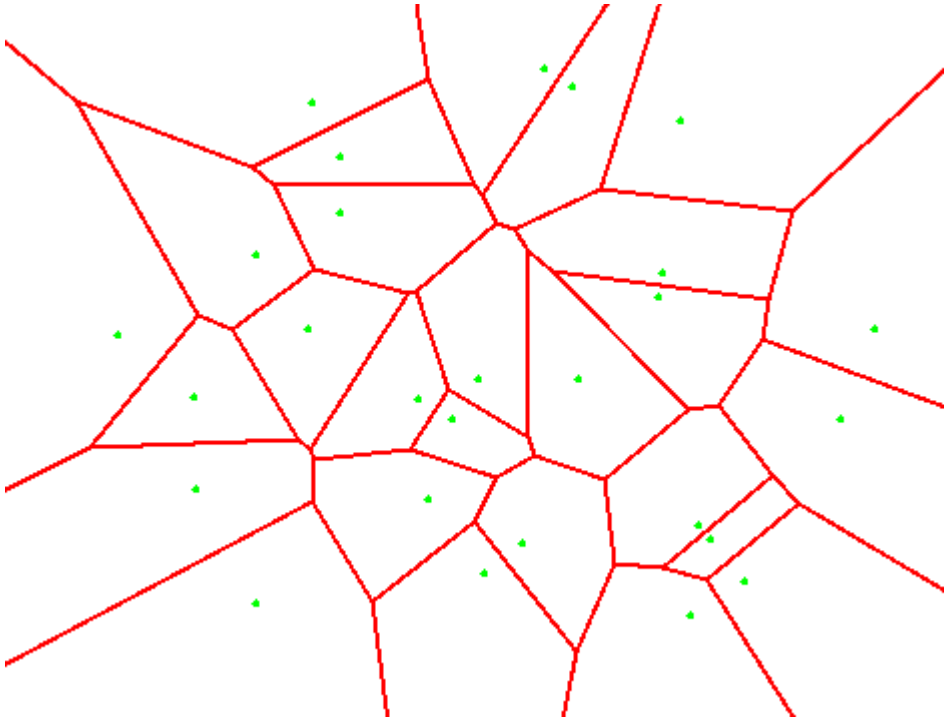


Mise à jour : prédicat in_circle(p, q, r, s), signe de :

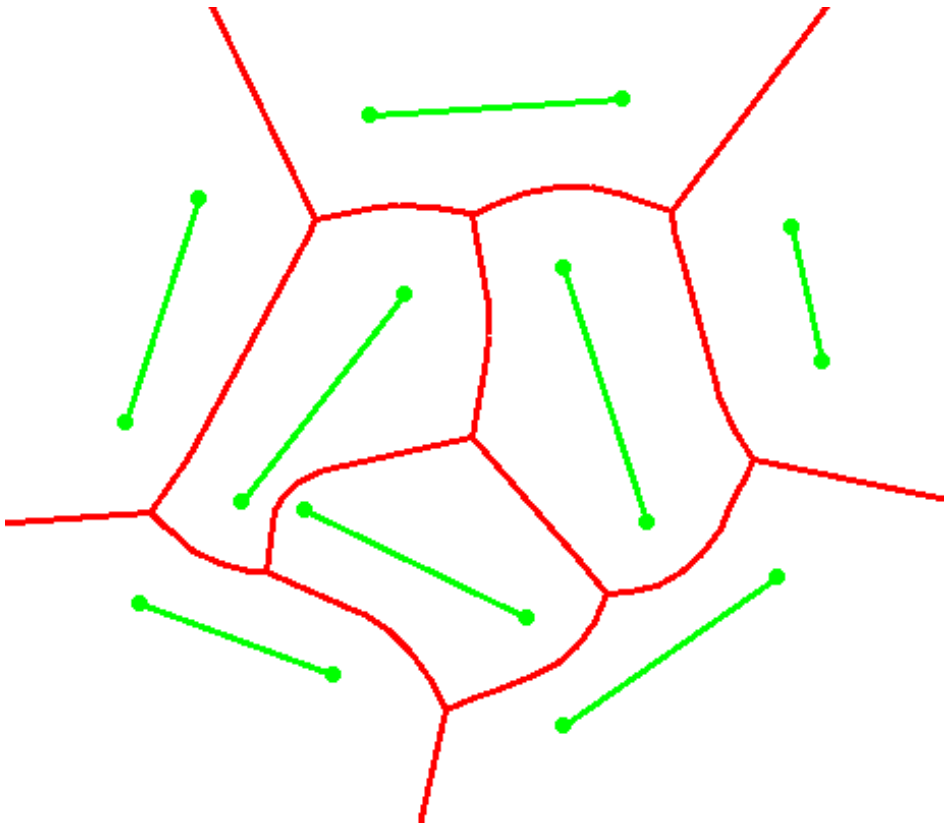
$$\begin{vmatrix} 1 & px & py & px^2 + py^2 \\ 1 & qx & qy & qx^2 + qy^2 \\ 1 & rx & ry & rx^2 + ry^2 \\ 1 & sx & sy & sx^2 + sy^2 \end{vmatrix}$$

INRIA

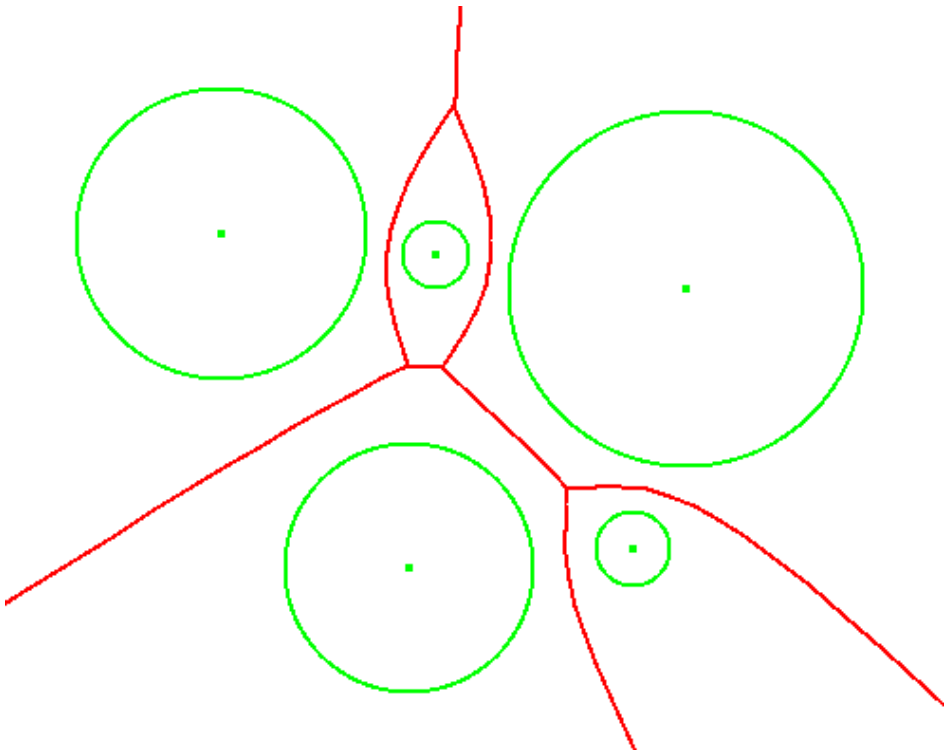
Diagrammes de Voronoï de points



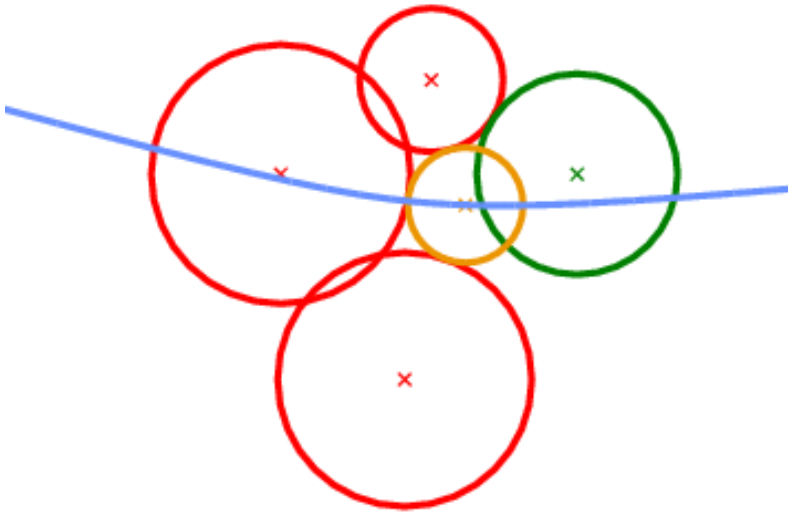
Diagrammes de Voronoï de segments



Diagrammes de Voronoï de cercles



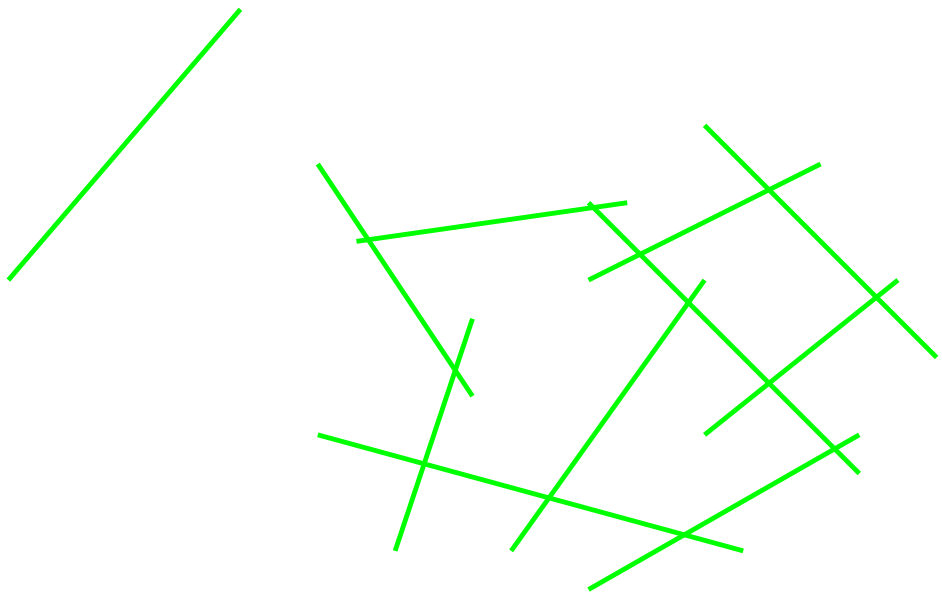
Un des prédicats du diagramme de Voronoï de cercles



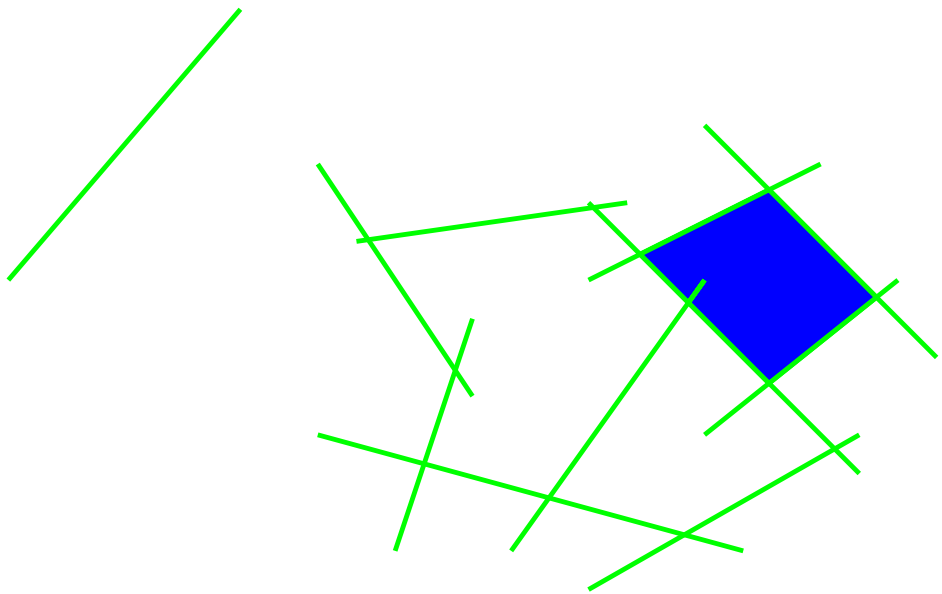
Techniques de comparaisons de racines

[Karavelas, Emiris : SODA'03]

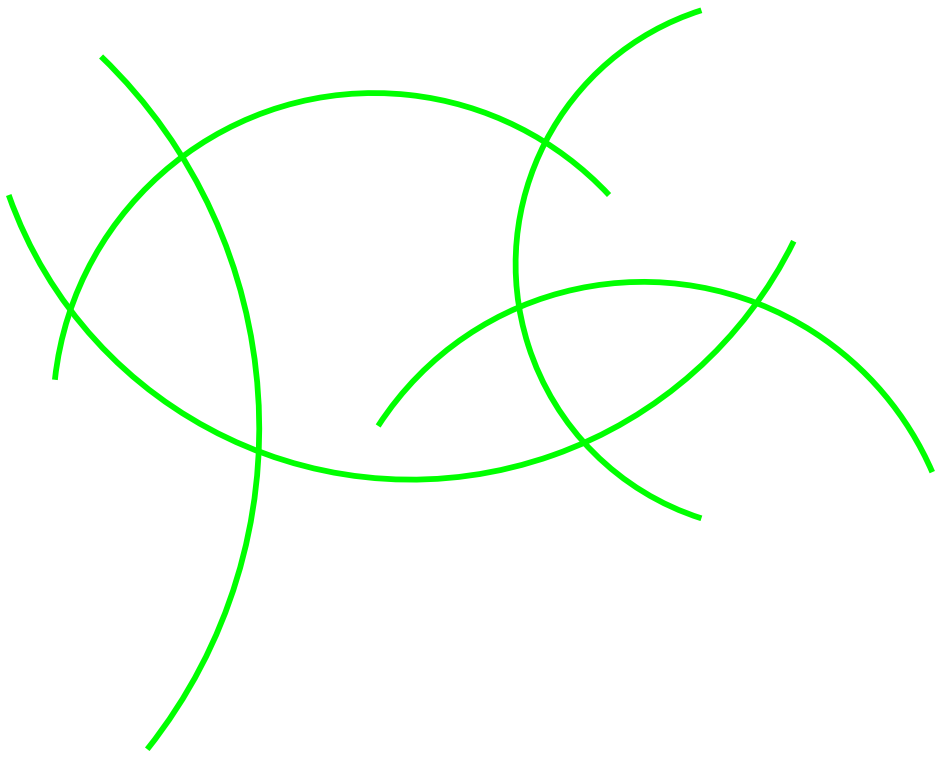
Arrangements de segments de droites



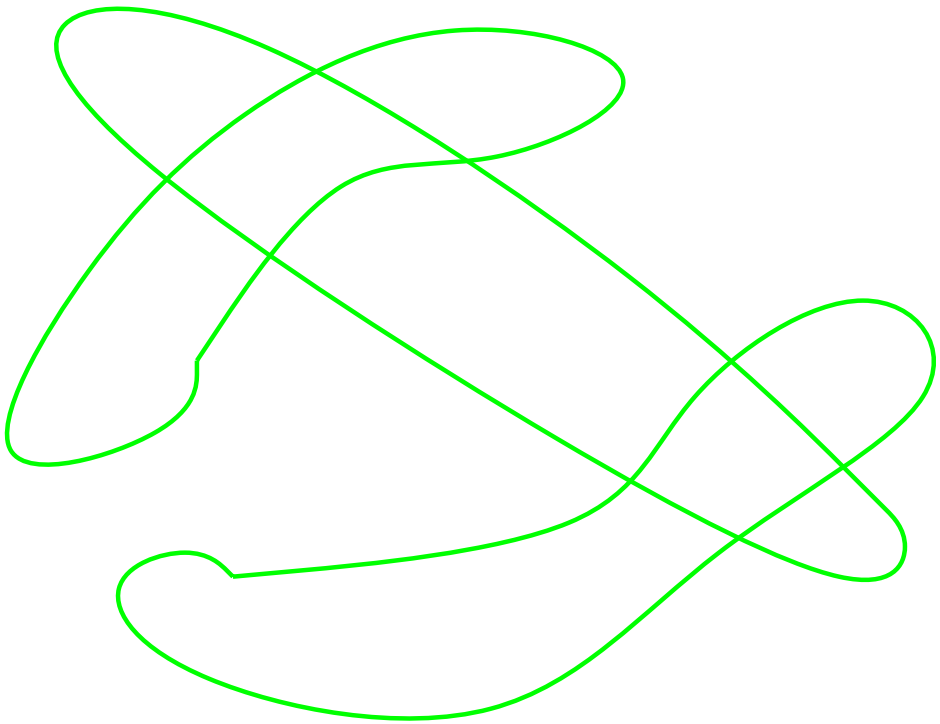
Arrangements de segments de droites



Arrangements d'arcs de cercles

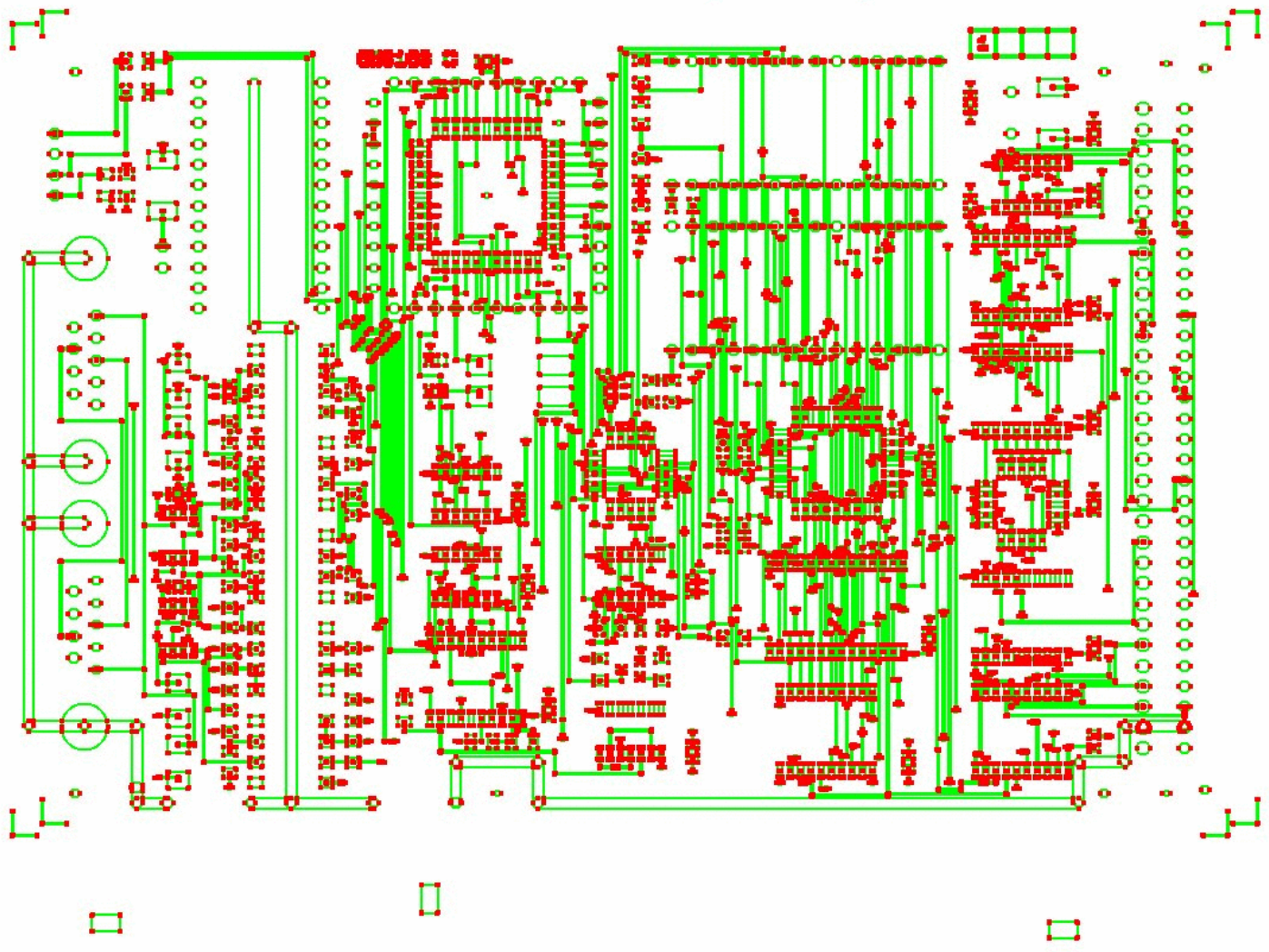


Arrangements de courbes

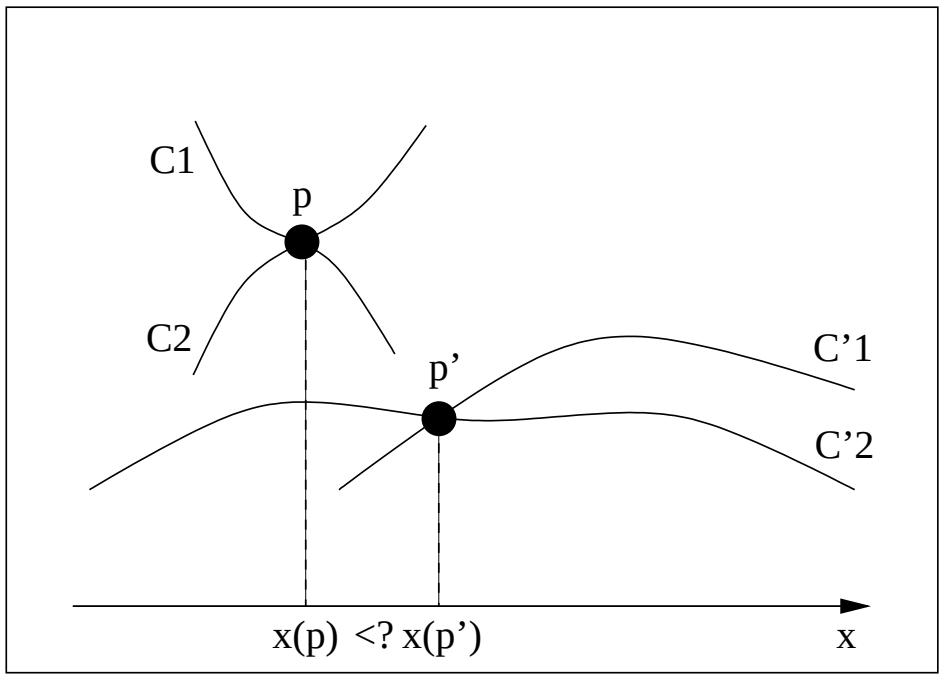


Noyau de calculs certifié pour la CAO
En 3D : arrangements de sphères, de quadriques...

Une application : union de polygones en VLSI



Comparaison d'abscisses d'intersection de courbes

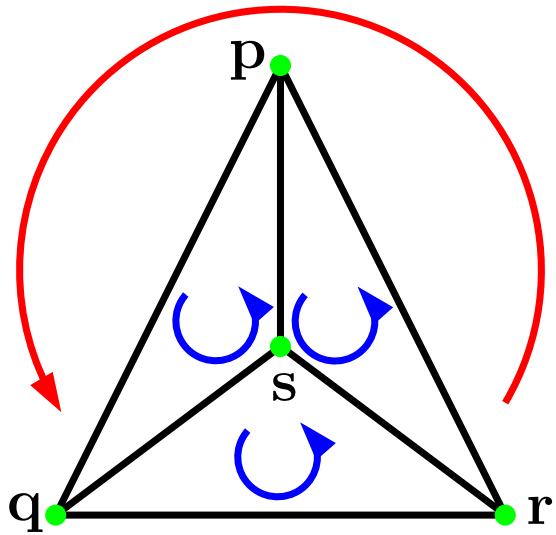


Courbes algébriques, comparaisons de nombres algébriques

Plan

Robustesse

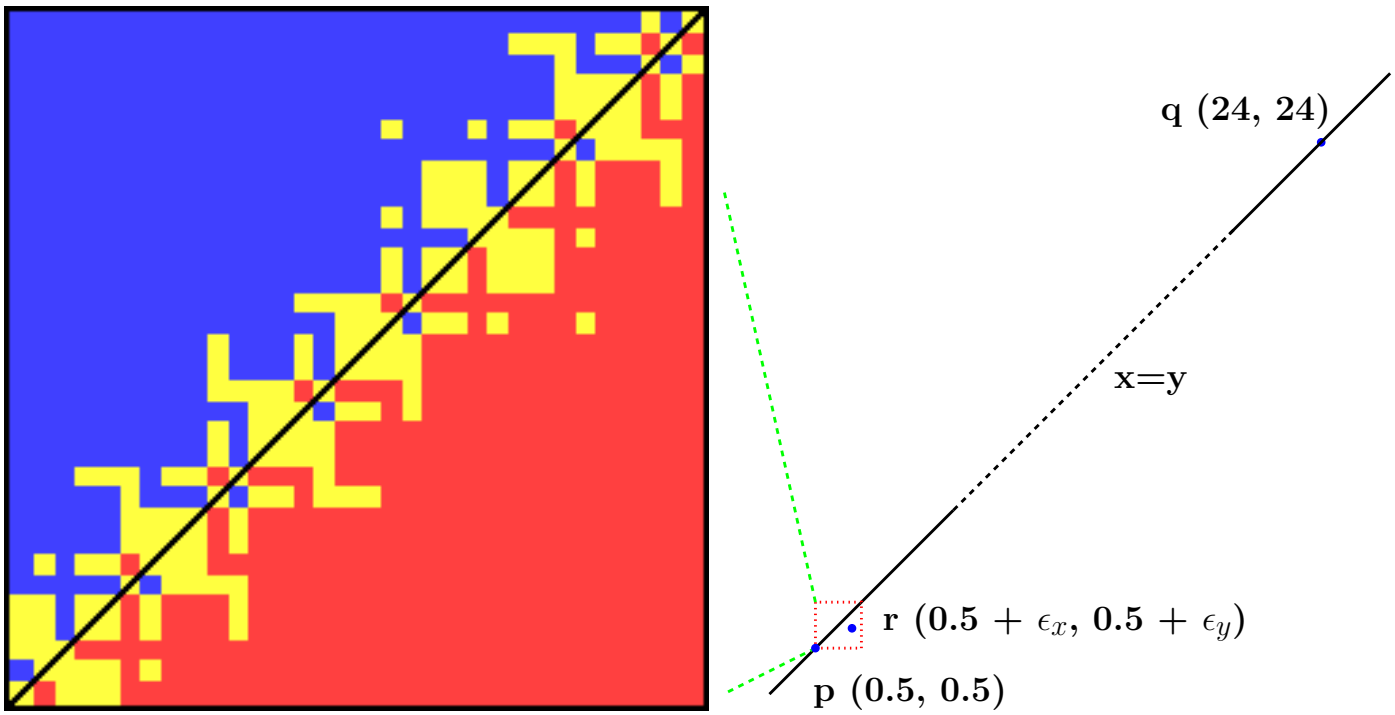
Les algorithmes reposent sur des théorèmes mathématiques, comme :



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

Robustesse

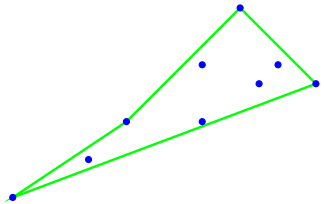
Exemple où la géométrie flottante diffère de la géométrie réelle :
orientation de points presque alignés.



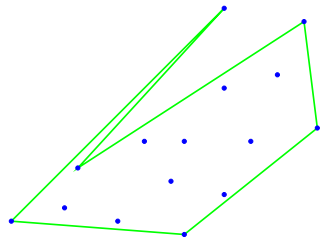
[Kettner, Mehlhorn, Schirra, P., Yap, ESA'04]

Conséquences possibles sur les algorithmes

- Le résultat est légèrement faux



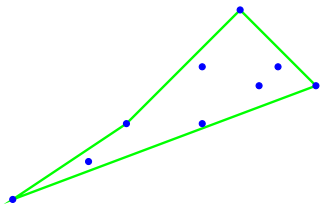
- Le résultat est grossièrement faux

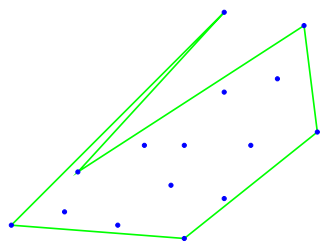


- L'algorithme s'arrête face à un cas supposé impossible
- L'algorithme boucle indéfiniment

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]

Prédicats filtrés

Accélération des prédicats exacts par un filtre :

- évaluation flottante avec certificat
- arithmétique multiprécision uniquement si besoin

Exemples

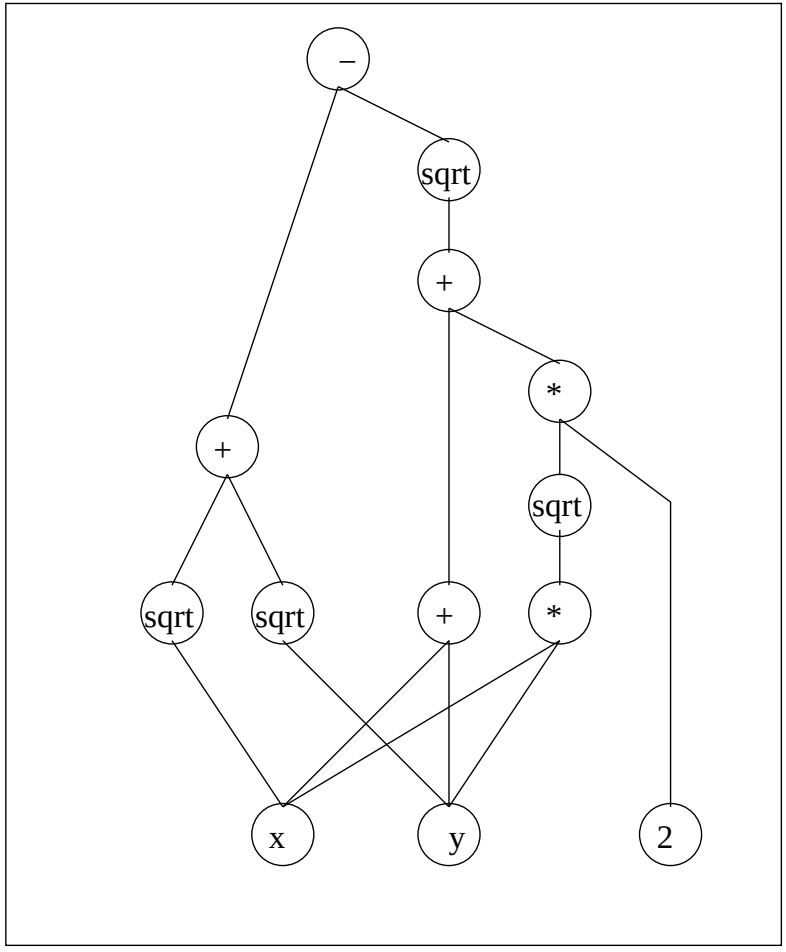
- arithmétique d'intervalles (filtres dynamiques), [Burnikel, Funke, Seel – Brönnimann, Burnikel, P.'98]
- ou analyse de code (filtres statiques) [Fortune'93... Melquiond, P.'05]

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 :

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

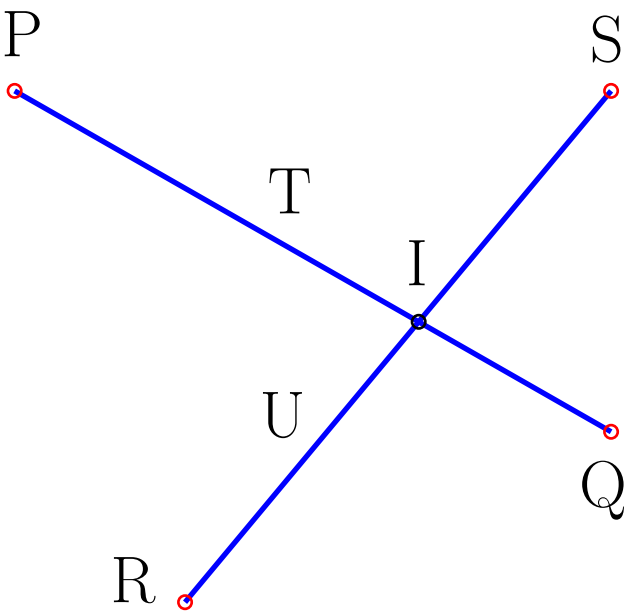
Complexité : Au pire (valeur nulle), il faudra itérer jusqu'à une précision inférieure à $B(E)$ pour calculer le signe de E.

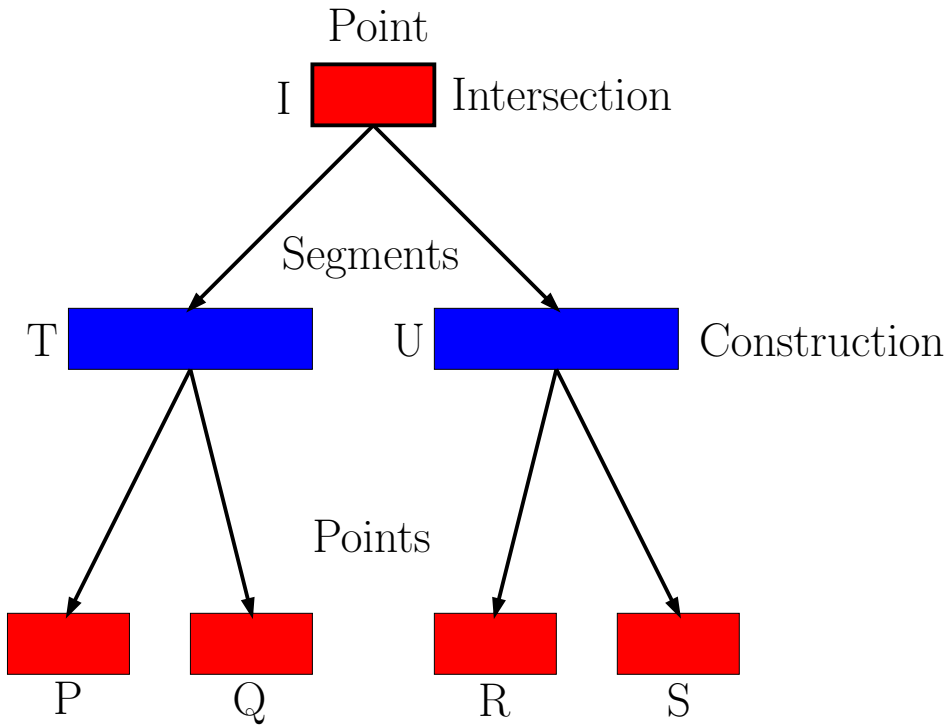
On sait calculer des bornes de séparations pour les expressions algébriques.
Il existe 3 types de bornes de séparations faciles à calculer non-comparables :

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

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 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;
```

Implémentation `Filtered_predicate`

```
template <class EP, class AP, class C2E, class C2A>
struct Filtered_predicate
{
  EP  ep;
  AP  ap;
  C2E c2e;
  C2A c2a;

  typedef typename AP::result_type  result_type;

  template <typename... Args>
  result_type operator()(const Args&... args) const
  {
    try {
      result_type res = ap(c2a(args)...);
      if (! is_indeterminate(res))
        return res;
    }
    catch (Interval_nt_advanced::unsafe_comparison) {}
    return ep(c2e(args)...);
  }
};
```

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.

INRIA

Plan

Langages et interfaces

Langages interprêtés :

- Scilab : clone de Matlab, utilisé par les ingénieurs
  structure de donnée centrale : matrice
- Python : similitudes avec C++, polymorphisme dynamique

Langages compilés :

- Java : similaire à C++, facilités d'interfaçage

Interface graphiques :

- Qt, OpenGL...
- Ipe

Langages et interfaces

Langages interprêtés :

- Scilab : clone de Matlab, utilisé par les ingénieurs
  structure de donnée centrale : matrice
- Python : similitudes avec C++, polymorphisme dynamique

Langages compilés :

- Java : similaire à C++, facilités d'interfaçage

Interface graphiques :

- Qt, OpenGL...
- Ipe

Langages et interfaces

Langages interprêtés :

- Scilab : clone de Matlab, utilisé par les ingénieurs
  structure de donnée centrale : matrice
- Python : similitudes avec C++, polymorphisme dynamique

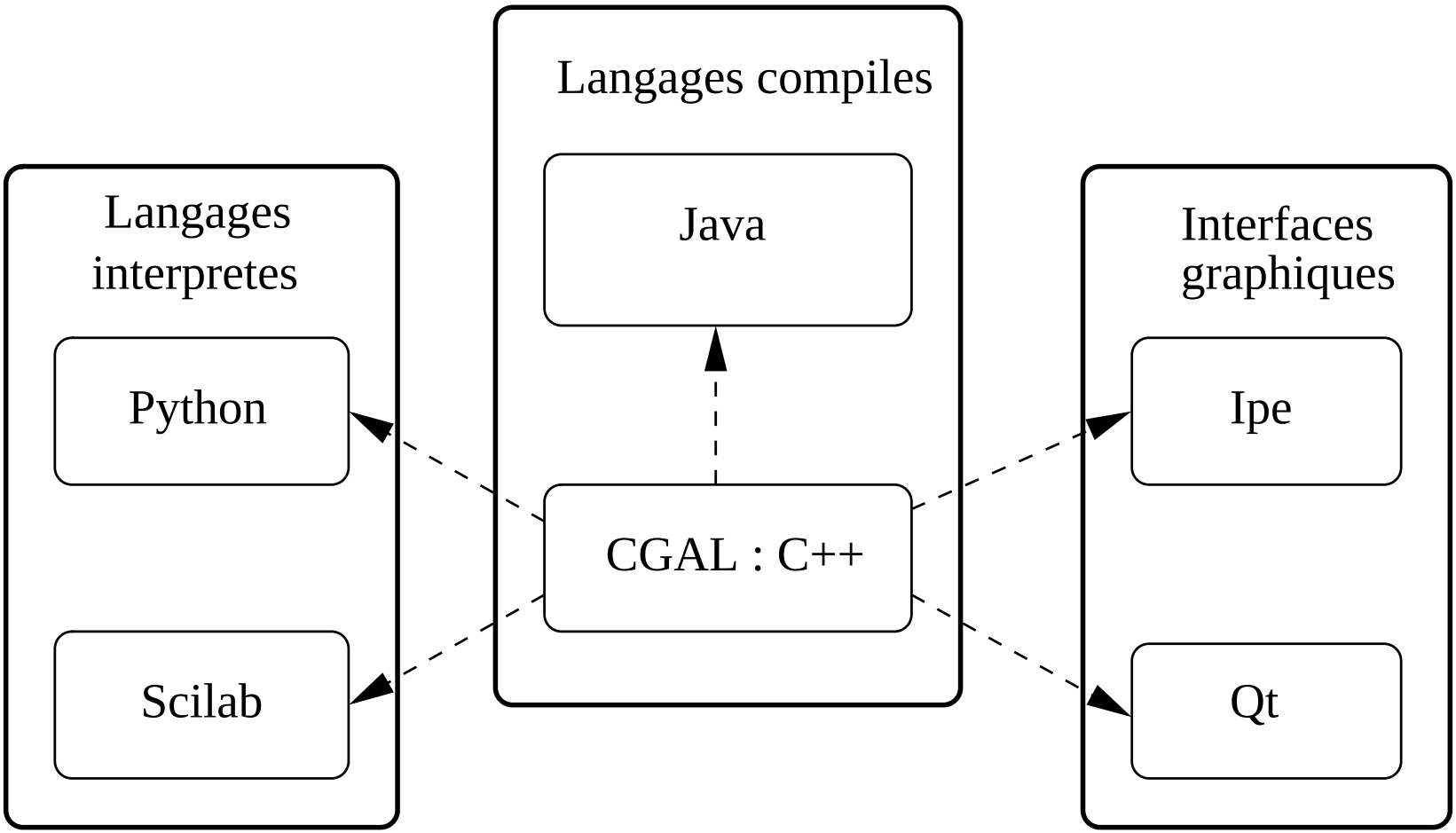
Langages compilés :

- Java : similaire à C++, facilités d'interfaçage

Interface graphiques :

- Qt, OpenGL...
- Ipe

Langages et interfaces



Plan

Conclusion

- Les logiciels sont fondamentaux pour supporter la recherche théorique
- La robustesse et le passage à l'échelle sont difficiles
- La programmation générique simplifie cette tâche.

- Disponibilité : `http://www.cgal.org/`

Des questions ?