Restricted Voronoi Diagrams
for (re)meshing Surfaces and Volumes

Curves and Surfaces 2014

Bruno Lévy
ALICE Géométrie & Lumière
CENTRE INRIA Nancy Grand-Est
OVERVIEW

Part. 1. Introduction

Motivations

The problem

Part. 2. The Predicates

Exact arithmetics with expansions [Shewchuk]

Arithmetic filters [Pion et.al]

Simulation of Simplicity [Edelsbrunner et.al]

Part. 3. Results and Predicate Construction Kit
1
Introduction
Part. 1. Motivations
Meshing and re-meshing
Part. 1. Motivations
Meshing and re-meshing
2

Restricted Voronoi Diagram Algorithms
Part. 2 The algorithm - Input

Pointset $X$

Simplicial complex $S$
Part. 2  The algorithm - Input

Pointset $X$

Simplicial complex $S$
either triangulated surface
or tetrahedralized solid
Part. 2 The algorithm - Input

Pointset $X$

Simplicial complex $S$
either triangulated surface
or tetrahedralized solid

Embedded in $\mathbb{IR}^d$
Part. 2 The algorithm - Output

\[ \text{Vor}(X)_{|S} \quad (\text{Intersection between Vor}(X) \text{ and } S) \]
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
Part. 2  The algorithm

Voronoi cells as iterative convex clipping
Neighbors in increasing distance from $x_i$
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
Bisector of $x_i$, $x_1$
Part. 2  The algorithm

Voronoi cells as iterative convex clipping
Half-space clipping

This side: \( \Pi^-(i, 1) \)

\( x_1 \)
\( x_2 \)
\( x_3 \)
\( x_4 \)
\( x_5 \)
\( x_6 \)
\( x_7 \)
\( x_8 \)
\( x_9 \)
\( x_{10} \)
\( x_{11} \)

The other side: \( \Pi^+(i, 1) \)
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
Half-space clipping

This side: $\Pi^-(i,1)$

The other side: $\Pi^+(i,1)$

Remove $\Pi^-(i,1)$
Part. 2  The algorithm

Voronoi cells as iterative convex clipping
Half-space clipping

Then remove $\Pi^{-}(i,2)$
Part. 2 The algorithm

Voronoi cells as iterative convex clipping

Half-space clipping

... then remove $\Pi^-(i,3)$
Part. 2  The algorithm

Voronoi cells as iterative convex clipping
Half-space clipping

... then remove $\Pi^{-}(i,4)$
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
Half-space clipping

... then remove $\Pi^{-}(i,5)$
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
When should I stop?
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
When should I stop? $R_k$
Part. 2  The algorithm

Voronoi cells as iterative convex clipping
When should I stop?
Part. 2  The algorithm

Voronoi cells as iterative convex clipping

When should I stop? \[ d(x_i, x_k) > 2 R_k \]
Part. 2 The algorithm

Voronoi cells as iterative convex clipping

Observation: \( d(x_i, x_{k+1}) > 2R_k \rightarrow \bigcap \Pi^+(i,k) = \text{Vor}(x_i) \)

[L and Bonneel – Voronoi Parallel Linear Enumeration]
[Dey et.al – Localized co-cone]
Part. 2  The algorithm

Voronoi cells as iterative convex clipping
When should I stop? \( d(x_i, x_k) > 2 R_k \)

“Radius of security” is reached

[L and Bonneel – Voronoi Parallel Linear Enumeration]
[Dey et.al – Localized co-cone]
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
When should I stop? \( d(x_i, x_k) > 2 R_k \)

“Radius of security” is reached

Note: \( R_k \) decreases and \( d(x_i, x_k) \) increases

[L and Bonneel – Voronoi Parallel Linear Enumeration]
[Dey et.al – Localized co-cone]
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
When should I stop? \( d(x_i, x_k) > 2 R_k \)

“Radius of security” is reached
Note: \( R_k \) decreases and \( d(x_i, x_k) \) increases

Advantages:

[L and Bonneel – Voronoi Parallel Linear Enumeration]
[Dey et.al – Localized co-cone]
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
When should I stop? \( d(x_i, x_k) > 2 R_k \)

“Radius of security” is reached
Note: \( R_k \) decreases and \( d(x_i, x_k) \) increases

Advantages:
1. Compute \( \text{Vor}(X) \cap S \) directly (start with f and clip)

[L and Bonneel – Voronoi Parallel Linear Enumeration]
[Dey et.al – Localized co-cone]
Part. 2 The algorithm

Voronoi cells as iterative convex clipping
When should I stop? \[ d(x_i, x_k) > 2 R_k \]

“Radius of security” is reached

Note: \( R_k \) decreases and \( d(x_i, x_k) \) increases

Advantages:
(1) Compute \( \text{Vor}(X) \cap S \) directly (start with f and clip)
(2) Replace Delaunay with ANN! (no d! factor)

[L and Bonneel – Voronoi Parallel Linear Enumeration]
[Dey et.al – Localized co-cone]
Part. 2 The algorithm

Voronoi cells as iterative convex clipping

When should I stop? \[ d(x_i, x_k) > 2R_k \]

“Radius of security” is reached

Note: \( R_k \) decreases and \( d(x_i, x_k) \) increases

Advantages:

1. Compute Vor(X) \( \cap S \) directly (start with f and clip)
2. Replace Delaunay with ANN! (no d! factor)
3. Parallelization is trivial (partition S and // in parts)
   [L and Bonneel – Voronoi Parallel Linear Enumeration]
   [Dey et.al – Localized co-cone]
Part. 2 Difficulties – predicates

Elementary operation: cut a polygon (or polyhedron) with a bisector
Part. 2  Difficulties – predicates

Elementary operation: cut a polygon (or polyhedron) with a bisector

Classify the vertices of the polygon
Part. 2 Difficulties – predicates

Elementary operation: cut a polygon (or polyhedron) with a bisector

Classify the vertices of the polygon

Sign( \( d(p, x_j) - d(p, x_i) \) ) > 0
Elementary operation: cut a polygon (or polyhedron) with a bisector

Classify the vertices of the polygon

\[ \text{Sign}( d(p, x_j) - d(p, x_i)) > 0 \]

\[ \text{Sign}( d(p, x_j) - d(p, x_i)) < 0 \]
Part. 2  Difficulties – predicates

Elementary operation: cut a polygon (or polyhedron) with a bisector

Classify the vertices of the polygon

Compute the intersections

\[ \text{Sign} (d(p, x_j) - d(p, x_i)) > 0 \]

\[ \text{Sign} (d(p, x_j) - d(p, x_i)) < 0 \]
Part. 2  Difficulties – predicates

Elementary operation: cut a polygon (or polyhedron) with a bisector

Classify the vertices of the polygon

Compute the intersections - discard

Elementary operation: cut a polygon (or polyhedron) with a bisector

Classify the vertices of the polygon

Compute the intersections - discard

Sign( d(p, x_j) − d(p, x_i)) > 0

Sign( d(p, x_j) − d(p, x_i)) < 0
Part. 2  Difficulties – predicates

Now clipping with the bisector of \((x_i, x_k)\)
Part. 2 Difficulties – predicates

Now clipping with the bisector of \((x_i, x_k)\)
We need to classify all the points, including these ones!
Part. 2 Difficulties – predicates

Now clipping with the bisector of \((x_i, x_k)\)
We need to classify all the points, including these ones!
(they are intersections between a segment and a bisector)
Part. 2 Difficulties – predicates

Now clipping with the bisector of \((x_i, x_k)\)
We need to classify all the points, including these ones!
(they are intersections between a segment and a bisector)

This generates a new intersection (between a facet and two bisectors)
Part. 2 Difficulties – predicates

Three configurations

1) Side($x_i, x_j, q$) where $q$ is a vertex of $S$
Part. 2 Difficulties – predicates

Three configurations

1) Side1($x_i, x_j, q$)
Part. 2 Difficulties – predicates

Three configurations

1) Side1($x_i, x_j, q$)

2) Side($x_i, x_j, q$) where $q = \Pi(i,k) \cap [p_1, p_2]$
Part. 2 Difficulties – predicates

Three configurations

1) Side1\( (x_i, x_j, q) \)
2) Side2\( (x_i, x_j, x_k, p_1, p_2) \)
Part. 2 Difficulties – predicates

Three configurations

1) Side1(x_i, x_j, q)
2) Side2(x_i, x_j, x_k, p_1, p_2)
3) Side(x_i, x_j, q) where
   \[ q = \Pi(i, k) \cap \Pi(i, l) \cap [p_1, p_2, p_3] \]
Part. 2  Difficulties – predicates

Three configurations

1) Side1(\(x_i,x_j,q\))
2) Side2(\(x_i,x_j,x_k,p_1,p_2\))
3) Side3(\(x_i,x_j,x_k,x_l,p_1,p_2,p_3\))
Part. 2 Difficulties – predicates

Three configurations

1) Side1\((x_i, x_j, q)\)
2) Side2\((x_i, x_j, x_k, p_1, p_2)\)
3) Side3\((x_i, x_j, x_k, x_l, p_1, p_2, p_3)\)

Implementations of exact predicates:
- J. Shewchuk’s code
- CGAL (Pion, Meyer)
Part. 2 Difficulties – predicates

Three configurations

1) Side1($x_i, x_j, q$
2) Side2($x_i, x_j, x_k, p_1, p_2$
3) Side3($x_i, x_j, x_k, x_l, p_1, p_2, p_3$

Implementations of exact predicates:
- J. Shewchuk’s code
- CGAL (Pion, Meyer)

They do not have Side1(), Side2(), Side3() (“exotic predicates”)
Part. 2 Difficulties – predicates

Three configurations

1) Side1\( (x_i, x_j, q) \)
2) Side2\( (x_i, x_j, x_k, p_1, p_2) \)
3) Side3\( (x_i, x_j, x_k, x_l, p_1, p_2, p_3) \)

How to implement Side1(), Side2(), Side3()?
We need an exact “number type” with \(+, -, *, \text{Sign}()\)
Part. 2 Exact Arithmetics

How to implement Side1(), Side2(), Side3()?

We need an exact “number type” with +, -, *, Sign()

Wish list:

- Easy to use
  (no “Guru” needed for each new predicate)
- Reasonably efficient
- Easy to compile/integrate
  (multi_precision.h, multi_precision.cpp and that’s all)
Part. 2 Exact Arithmetics

How to implement Side1(), Side2(), Side3()?

We need an exact “number type” with +,-,*,Sign()

Idea #1: (dense) multi-precision (GMP)

Each number is an array of (32 bits) integers:

\[
\begin{array}{cccc}
\cdots & a_3 & a_2 & a_1 & a_0 \\
\times 2^{3\times32} & \times 2^{2\times32} & \times 2^{1\times32} & \times 2^0
\end{array}
\]

Implement +,-,\,* (reasonably easy)
Sign: look at the leading non-zero component
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3() ?
We need an exact “number type” with +,-,*,Sign()

Idea #1: (dense) multi-precision (GMP)

a “limitation”:

\[ c = a_{10} \times 2^{10\times32} + b_0 \]
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3() ?
We need an exact “number type” with  +,-,*,Sign()

Idea #1: (dense) multi-precision (GMP)

a “limitation”:

\[ c = a_{10} \times 2^{10 \times 32} + b_0 \]

\[ \begin{array}{c}
\text{a10} \\
\hline
0 \\
\hline
\hdots \\
\hline
0 \\
\hline
b0 \\
\end{array} \]

\[ \begin{array}{c}
x 2^{10 \times 32} \\
\hline
\end{array} \]

\[ \begin{array}{c}
x 2^0 \\
\end{array} \]
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3() ?
We need an exact “number type” with +,-,*,Sign()

Idea #2: (sparse) multi-precision

\[ c = a_{10} \times 2^{1032} + b_0 \]

Store the exponents of the components
Part. 2 Exact Arithmetics

How to implement Side1(), Side2(), Side3() ?
We need an exact “number type” with +,-,*,Sign()

Idea #2: (sparse) multi-precision

\[ c = a_{10} \times 2^{10 \times 32} + b_0 \]

Exp.  |  Exp.
---    |  ---
a10    |  10  
b0     |  0   
   Exp. |   Exp.
Part. 2 Exact Arithmetics

How to implement Side1(), Side2(), Side3()?
We need an exact “number type” with +,-,*,Sign()

Idea #2: (sparse) multi-precision

c = a_{10} \times 2^{10\times32} + b_0

<table>
<thead>
<tr>
<th>a_{10}</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mantissa</td>
<td>Exp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b_0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>mantissa</td>
<td>Exp.</td>
</tr>
</tbody>
</table>
Part. 2    Exact Arithmetics

How to implement Side1(), Side2(), Side3() ?
We need an exact “number type” with +,-,*,Sign()

Idea #2: (sparse) multi-precision

\[ c = a_{10} \times 2^{10\times32} + b_0 \]

These are floating point numbers !!!
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3() ?
We need an exact “number type” with +,-,*,Sign()

Idea #3: expansions (Shewchuk)

\[ \cdots \quad x_3 \quad x_2 \quad x_1 \]

Each number is represented by the sum of an array of ‘components’

These are floating point numbers !!!
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3()?
We need an exact “number type” with +,-,*,Sign()

Idea #3: expansions (Shewchuk)

\[ \cdots x_3 x_2 x_1 \]

Each number is represented by the sum of an array of ‘components’
They are sorted in decreasing exponents

These are floating point numbers !!!
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3()?

We need an exact “number type” with \(+, -, *, \text{Sign()}

Idea #3: expansions (Shewchuk)

\[
\ldots \quad x_3 \quad x_2 \quad x_1
\]

Each number is represented by the sum of an array of ‘components’
They are sorted in decreasing exponents
They are ‘non-overlapping’

These are floating point numbers !!!
Part. 2 Exact Arithmetics

How to implement Side1(), Side2(), Side3()?
We need an exact “number type” with \(+, -, *, \text{Sign}()\)

Idea #3: expansions (Shewchuk)

\[ \ldots \quad x_3 \quad x_2 \quad x_1 \]

Each number is represented by the sum of an array of ‘components’
They are sorted in decreasing exponents
They are ‘non-overlapping’
The sign is determined by the first component (highest exponent)

These are floating point numbers !!!
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3()?
We need an exact “number type” with +,-,*,Sign()

Idea #3: expansions (Shewchuk)

Two_sum(double a, double b) → x2 x1
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3()? We need an exact “number type” with +,-,*,Sign()

Idea #3: expansions (Shewchuk)

Two_sum(double a, double b) → $x_2$ $x_1$

Length l + Length m → Length: l+m
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3()?
We need an exact “number type” with +,-,*,Sign()

Idea #3: expansions (Shewchuk)

Two_sum(double a, double b) → x2

+

A double

* → x1

Length: l+m

Length: 2*l

Length l
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3() ?
We need an exact “number type” with +,-,*,Sign()

Idea #3: expansions (Shewchuk)

Expansion * Expansion product implemented by a recursive function ("distillation")
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3()?
We need an exact “number type” with +, -, *, Sign()

Idea #3: expansions (Shewchuk)

Expansion * Expansion product implemented by a recursive function (“distillation”)

Performance? 10 to 40 times slower than standard doubles
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3()?
We need an exact “number type” with +, -, *, Sign()

Idea #3: expansions (Shewchuk)

Expansion * Expansion product implemented by a recursive function (“distillation”)

Performance? 10 to 40 times slower than standard doubles
Use arithmetic filters
  Adaptive precision [Shewchuk]? Too complicated to get right
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3()?
We need an exact “number type” with +, -, *, Sign()

Idea #3: expansions (Shewchuk)

Expansion * Expansion product implemented by a recursive function (“distillation”)

Performance? 10 to 40 times slower than standard doubles
Use arithmetic filters
Adaptive precision [Shewchuk]? Too complicated to get right
Quasi-static filters [Meyer and Pion] – FPG generator (easy to use)
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3() ?
We need an exact “number type” with +,-,*,Sign()

PCK (Predicate Construction Kit)

multi_precision.h / multi_precision.cpp
    A “low-level” expansion class (allocates expansions on stack)
    A “high-level” C++ number type (+,-,*,Sign overloads)

a script that generates the filter with FPG [Meyer and Pion] and the exact precision version with expansions
Part. 2  Exact Arithmetics

How to implement Side1(), Side2(), Side3() ?
We need an exact “number type” with $+, -, \times, \text{Sign()}$

PCK (Predicate Construction Kit)

multi_precision.h / multi_precision.cpp
A “low-level” expansion class (allocates expansions on stack)
A “high-level” C++ number type ($+, -, \times, \text{Sign}$ overloads)

a script that generates the filter with FPG [Meyer and Pion] and the exact precision version with expansions

So we are done ?
Part. 2 Symbolic Perturbation

How to implement Side1(), Side2(), Side3()?

Not yet!!
Part. 2 Symbolic Perturbation

How to implement Side1(), Side2(), Side3()?

Not yet!!
Part. 2  Symbolic Perturbation
Part. 2 Symbolic Perturbation
Part. 2  Symbolic Perturbation
Part. 2 Symbolic Perturbation
Part. 2 Symbolic Perturbation
Part. 2 Symbolic Perturbation
Part. 2 Symbolic Perturbation
Part. 2  Symbolic Perturbation
Part. 2  Symbolic Perturbation
Part. 2  Symbolic Perturbation
Part. 2  Symbolic Perturbation
Part. 2  Symbolic Perturbation
Part. 2 Symbolic Perturbation
Part. 2  Symbolic Perturbation
Part. 2 Symbolic Perturbation
Part. 2 Symbolic Perturbation
Part. 2  Symbolic Perturbation

\[ \Pi(i,j) = \{ p \mid d^2(p, x_i) = d^2(p, x_j) \} \]

[Voronoi]
[Edelsbrunner et.al]
[Devillers et.al]
Part. 2  Symbolic Perturbation

\[ \Pi_w(i,j) = \{ p \mid d^2(p, x_i) - w_i = d^2(p, x_j) - w_j \} \]

[Voronoi]
[Edelsbrunner et.al]
[Devillers et.al]
Part. 2 Symbolic Perturbation

\[ \Pi_w(i,j) = \{ p \mid d^2(p, x_i) - w_i = d^2(p, x_j) - w_j \} \]

The Voronoi diagram is replaced with a power diagram.
Part. 2  Symbolic Perturbation

\[ \Pi_{w}(i,j) = \{ p \mid d^2(p, x_i) - w_i = d^2(p, x_j) - w_j \} \]

The Voronoi diagram is replaced with a power diagram

**Symbolic perturbation – Simulation of Simplicity:**
Define the weight as a function of \( \varepsilon \): \( w_i = \varepsilon^i \)

[Voronoi]
[Edelsbrunner et.al]
[Devillers et.al]
Part. 2 Symbolic Perturbation

\[ \Pi_w^{(i,j)} = \{ p \mid d^2(p, x_i) - w_i = d^2(p, x_j) - w_j \} \]

The Voronoi diagram is replaced with a power diagram.

Symbolic perturbation – Simulation of Simplicity:
Define the weight as a function of \( \varepsilon \): \( w_i = \varepsilon^i \)
The combinatorics is determined by the limit \( \varepsilon \to 0 \).

[Voronoi]
[Edelsbrunner et.al]
[Devillers et.al]
Part. 2  Symbolic Perturbation

How to write side1(), side2(), side3(), side4()?

\[ d^2(q, p_j) - w_j - d^2(q, p_i) + w_i \]

where:

\[ q = \prod_{w(i,k_1)} \cap ... \prod_{w(i,k_d)} \cap [p_1, p_2, p_3 ... p_d] \]
Part. 2 Symbolic Perturbation

How to write side1(), side2(), side3(), side4()?

\[ d^2(q,p_j) - w_j - d^2(q,p_i) + w_i \]

where:

\[ q = \Pi_{w(i,k_1)} \cap \ldots \cap \Pi_{w(i,k_d)} \cap [p_1, p_2, p_3 \ldots p_d] \]

Solve for \( q \) in:

\[
\begin{cases}
q \in \Pi_{w(i,k_1)} \\
\ldots \\
q \in \Pi_{w(i,k_d)} \\
q \in [p_1, p_2, p_3 \ldots p_d]
\end{cases}
\]
Part. 2  Symbolic Perturbation

How to write side1(), side2(), side3(), side4()?

\[ d^2(q,p_j) - w_j - d^2(q,p_i) + w_i \] where:

\[ q = \Pi_{w(i,k_1)} \cap \ldots \cap \Pi_{w(i,k_d)} \cap [p_1, p_2, p_3 \ldots p_d] \]

Solve for \( q \) in:

\[
\begin{align*}
q \in \Pi_{w(i,k_1)} \\
\vdots \\
q \in \Pi_{w(i,k_d)} \\
q \in [p_1, p_2, p_3 \ldots p_d]
\end{align*}
\]

\[ q = (1/d) \cdot Q \]

Keep *numerator* and *denom.* separate (remember, we are not allowed to divide)
Part. 2  Symbolic Perturbation

How to write side1(), side2(), side3(), side4()?

\[
d^2(q, p_j) - w_j - d^2(q, p_i) + w_i
\]

where:

\[
q = \Pi_{w(i,k_1)} \cap \ldots \cap \Pi_{w(i,k_d)} \cap [p_1, p_2, p_3 \ldots p_d]
\]

Solve for \( q \) in:

\[
\begin{align*}
q & \in \Pi_{w(i,k_1)} \\
\ldots \\
q & \in \Pi_{w(i,k_d)} \\
q & \in [p_1, p_2, p_3 \ldots p_d]
\end{align*}
\]

Keep numerator and denom. separate (remember, we are not allowed to divide)

Inject \( q = (1/d) \ Q \) in side1() and multiply by \( d \) to remove the division
Part. 2  Symbolic Perturbation

How to write side1(), side2(), side3(), side4()?

\[ d^2(q,p_j) - w_j - d^2(q,p_i) + w_i \]

where:

\[ q = \prod_{w(i,k_1)} \cap \cdots \cap \prod_{w(i,k_d)} \cap [p_1,p_2,p_3\ldots p_d] \]

Solve for \( q \) in:

\[
\begin{cases}
q \in \prod_{w(i,k_1)} \\
\vdots \\
q \in \prod_{w(i,k_d)} \\
q \in [p_1,p_2,p_3\ldots p_d]
\end{cases}
\]

Keep *numerator* and *denom.* separate (remember, we are not allowed to divide)

Inject \( q=(1/d) Q \) in side1() and multiply by \( d \) to remove the division

Order the terms in \( w_i = \varepsilon^i \)
Part. 2  Symbolic Perturbation

How to write side1(), side2(), side3(), side4()?

\[ d^2(\mathbf{q}, \mathbf{p}_j) - w_j - d^2(\mathbf{q}, \mathbf{p}_i) + w_i \quad \text{where:} \]

\[ \mathbf{q} = \prod_{w(i,k_1)} \cap \ldots \prod_{w(i,k_d)} \cap [p_1, p_2, p_3 \ldots p_d] \]

Solve for \( \mathbf{q} \) in:

\[
\begin{cases}
\mathbf{q} \in \prod_{w(i,k_1)} \\
\cdots \\
\mathbf{q} \in \prod_{w(i,k_d)} \\
\mathbf{q} \in [p_1, p_2, p_3 \ldots p_d]
\end{cases}
\]

Keep numerator and denom. separate (remember, we are not allowed to divide)

Inject \( \mathbf{q} = (1/d) \mathbf{Q} \) in side1() and multiply by \( d \) to remove the division

Order the terms in \( w_i = \varepsilon^i \) the constant one is non-perturbed predicate

if zero, the first non-zero one determines the sign
#include "kernel.pckh"

Sign predicate(side1)(
  point(p0), point(p1), point(q0)  DIM
) {
  scalar r = sq_dist(p0,p1) ;
  r -= 2*dot_at(p1,q0,p0) ;
generic_predicate_result(sign(r)) ;
begin_sos2(p0,p1)
  sos(p0,POSITIVE)
  sos(p1,NEGATIVE)
end_sos
}
#include "kernel.pckh"

Sign predicate(side1)(
    point(p0), point(p1), point(q0) DIM
) {
    scalar r = sq_dist(p0,p1) ;
    \textcolor{blue}{r -= 2*dot_at(p1,q0,p0) ;}
    generic_predicate_result(sign(r)) ;
    begin_sos2(p0,p1)
        sos(p0,POSITIVE)
        sos(p1,NEGATIVE)
    end_sos
}
#include "kernel.pckh"

Sign predicate(side2)( point(p0), point(p1), point(p2), point(q0), point(q1) DIM) {
    scalar l1 = 1*sq_dist(p1,p0) ;
    scalar l2 = 1*sq_dist(p2,p0) ;
    scalar a10 = 2*dot_at(p1,q0,p0);
    scalar a11 = 2*dot_at(p1,q1,p0);
    scalar a20 = 2*dot_at(p2,q0,p0);
    scalar a21 = 2*dot_at(p2,q1,p0);
    scalar Delta = a11 - a10 ;
    scalar DeltaLambda0 = a11 - l1 ;
    scalar DeltaLambda1 = l1 - a10 ;
    scalar r = Delta*l2 - a20*DeltaLambda0 - a21*DeltaLambda1 ;

    Sign Delta_sign = sign(Delta) ;
    Sign r_sign = sign(r) ;
    generic_predicate_result(Delta_sign*r_sign) ;

    begin_sos3(p0,p1,p2)
        sos(p0, Sign(Delta_sign*sign(Delta-a21+a20)))
        sos(p1, Sign(Delta_sign*sign(a21-a20)))
        sos(p2, NEGATIVE)
    end_sos

}
#include "kernel.pckh"

Sign predicate(side2)( point(p0), point(p1), point(p2), point(q0), point(q1) DIM) {
    scalar l1 = 1*sq_dist(p1,p0) ;
    scalar l2 = 1*sq_dist(p2,p0) ;
    scalar a10 = 2*dot_at(p1,q0,p0);
    scalar a11 = 2*dot_at(p1,q1,p0);
    scalar a20 = 2*dot_at(p2,q0,p0);
    scalar a21 = 2*dot_at(p2,q1,p0);
    scalar Delta = a11 - a10 ;
    scalar DeltaLambda0 = a11 - l1 ;
    scalar DeltaLambda1 = l1 - a10 ;
    scalar r = Delta*l2 - a20*DeltaLambda0 - a21*DeltaLambda1 ;

    Sign Delta_sign = sign(Delta) ;
    Sign r_sign = sign(r) ;
    generic_predicate_result(Delta_sign*r_sign) ;

    begin_sos3(p0,p1,p2)
        sos(p0, Sign(Delta_sign*sign(Delta-a21+a20)))
        sos(p1, Sign(Delta_sign*sign(a21-a20)))
        sos(p2, NEGATIVE)
    end_sos
}
#include "kernel.pckh"

Sign predicate(side2)( point(p0), point(p1), point(p2), point(q0), point(q1) DIM) {
    scalar l1 = 1*sq_dist(p1,p0) ;
    scalar l2 = 1*sq_dist(p2,p0) ;
    scalar a10 = 2*dot_at(p1,q0,p0);
    scalar a11 = 2*dot_at(p1,q1,p0);
    scalar a20 = 2*dot_at(p2,q0,p0);
    scalar a21 = 2*dot_at(p2,q1,p0);
    scalar Delta = a11 - a10 ;
    scalar DeltaLambda0 = a11 - l1 ;
    scalar DeltaLambda1 = l1 - a10 ;
    scalar r = Delta*l2 - a20*DeltaLambda0 - a21*DeltaLambda1 ;
    Sign Delta_sign = sign(Delta) ;
    Sign r_sign = sign(r) ;
    generic_predicate_result(Delta_sign*r_sign) ;
    begin_sos3(p0,p1,p2)
        sos(p0, Sign(Delta_sign*sign(Delta-a21+a20)))
        sos(p1, Sign(Delta_sign*sign(a21-a20)))
        sos(p2, NEGATIVE)
    end_sos
}
#include "kernel.pckh"

Sign predicate(side2)( point(p0), point(p1), point(p2), point(q0), point(q1), DIM) {  
    scalar l1 = 1*sq_dist(p1,p0) ;  
    scalar l2 = 1*sq_dist(p2,p0) ;  
    scalar a10 = 2*dot_at(p1,q0,p0);  
    scalar a11 = 2*dot_at(p1,q1,p0);  
    scalar a20 = 2*dot_at(p2,q0,p0);  
    scalar a21 = 2*dot_at(p2,q1,p0);  
    scalar Delta = a11 - a10 ;  
    scalar DeltaLambda0 = a11 - l1 ;  
    scalar DeltaLambda1 = l1 - a10 ;  
    scalar r = Delta*l2 - a20*DeltaLambda0 - a21*DeltaLambda1 ;  
    Sign Delta_sign = sign(Delta) ;  
    Sign r_sign = sign(r) ;  
    generic_predicate_result(Delta_sign*r_sign) ;  
    begin_sos3(p0,p1,p2)  
        sos(p0, Sign(Delta_sign*sign(Delta-a21+a20)))  
        sos(p1, Sign(Delta_sign*sign(a21-a20)))  
        sos(p2, NEGATIVE)  
    end_sos
}

Barycentric coords. of q

Solely depend on dot products between input points

Source PCK
#include "kernel.pckh"

Sign predicate(side2)( point(p0), point(p1), point(p2), point(q0), point(q1) DIM) {
    scalar l1 = 1*sq_dist(p1,p0) ;
    scalar l2 = 1*sq_dist(p2,p0) ;
    scalar a10 = 2*dot_at(p1,q0,p0);
    scalar a11 = 2*dot_at(p1,q1,p0);
    scalar a20 = 2*dot_at(p2,q0,p0);
    scalar a21 = 2*dot_at(p2,q1,p0);
    scalar Delta = a11 - a10 ;
    scalar DeltaLambda0 = a11 - l1 ;
    scalar DeltaLambda1 = l1 - a10 ;
    scalar r = Delta*l2 - a20*DeltaLambda0 - a21*DeltaLambda1 ;

    Sign Delta_sign = sign(Delta) ;
    Sign r_sign = sign(r) ;
    generic_predicate_result(Delta_sign*r_sign) ;

    begin_sos3(p0,p1,p2)
    	sos(p0, Sign(Delta_sign*sign(Delta-a21+a20)))
    	sos(p1, Sign(Delta_sign*sign(a21-a20)))
    	sos(p2, NEGATIVE)
    end_sos
}

Result when in generic position

Source PCK
#include "kernel.pckh"

Sign predicate(side2)( point(p0), point(p1), point(p2), point(q0), point(q1) DIM) {
  scalar l1 = 1*sq_dist(p1,p0) ;
  scalar l2 = 1*sq_dist(p2,p0) ;
  scalar a10 = 2*dot_at(p1,q0,p0);
  scalar a11 = 2*dot_at(p1,q1,p0);
  scalar a20 = 2*dot_at(p2,q0,p0);
  scalar a21 = 2*dot_at(p2,q1,p0);
  scalar Delta = a11 - a10 ;
  scalar DeltaLambda0 = a11 - l1 ;
  scalar DeltaLambda1 = l1 - a10 ;
  scalar r = Delta*l2 - a20*DeltaLambda0 - a21*DeltaLambda1 ;

  Sign Delta_sign = sign(Delta) ;
  Sign r_sign = sign(r) ;
  generic_predicate_result(Delta_sign*r_sign) ;

begin_sos3(p0,p1,p2)
  sos(p0, Sign(Delta_sign*sign(Delta-a21+a20)))
  sos(p1, Sign(Delta_sign*sign(a21-a20)))
  sos(p2, NEGATIVE)
end_sos
}
Part. 2  Symbolic Perturbation – side2

```c
Sign side2_exact_SOS(
    const double* p0, const double* p1, const double* p2,
    const double* q0, const double* q1,
    coord_index_t dim
) {
    const expansion& l1 = expansion_sq_dist(p1, p0, dim);
    const expansion& l2 = expansion_sq_dist(p2, p0, dim);

    const expansion& a10 = expansion_dot_at(p1, q0, p0, dim).scale_fast(2.0);
    const expansion& a11 = expansion_dot_at(p1, q1, p0, dim).scale_fast(2.0);
    const expansion& a20 = expansion_dot_at(p2, q0, p0, dim).scale_fast(2.0);
    const expansion& a21 = expansion_dot_at(p2, q1, p0, dim).scale_fast(2.0);

    const expansion& Delta = expansion_diff(a11, a10);
    Sign Delta_sign = Delta.sign();

    vor_assert(Delta_sign != ZERO);
    const expansion& DeltaLambda0 = expansion_diff(a11, l1);
    const expansion& DeltaLambda1 = expansion_diff(l1, a10);

    const expansion& r0 = expansion_product(Delta, l2);
    const expansion& r1 = expansion_product(a20, DeltaLambda0).negate();
    const expansion& r2 = expansion_product(a21, DeltaLambda1).negate();
    const expansion& r = expansion_sum3(r0, r1, r2);

    Sign r_sign = r.sign();
    ..........
}
```

Exact version with expansions
if(r_sign == ZERO) {
    const double* p_sort[3];
    p_sort[0] = p0;
    p_sort[1] = p1;
    p_sort[2] = p2;
    std::sort(p_sort, p_sort + 3);
    for(index_t i = 0; i < 3; ++i) {
        if(p_sort[i] == p0) {
            const expansion& z1 = expansion_diff(Delta, a21);
            const expansion& z = expansion_sum(z1, a20);
            Sign z_sign = z.sign();
            len_side2_SOS = vor_max(len_side2_SOS, z.length());
            if(z_sign != ZERO) {
                return Sign(Delta_sign * z_sign);
            }
        }
        if(p_sort[i] == p1) {
            const expansion& z = expansion_diff(a21, a20);
            Sign z_sign = z.sign();
            len_side2_SOS = vor_max(len_side2_SOS, z.length());
            if(z_sign != ZERO) {
                return Sign(Delta_sign * z_sign);
            }
        }
        if(p_sort[i] == p2) {
            return NEGATIVE;
        }
    }
    vor_assert_not_reached;
}
return Sign(Delta_sign * r_sign);

Exact version with expansions
#include "kernel.pckh"

Sign predicate(side3)(
    point(p0), point(p1), point(p2), point(p3),
    point(q0), point(q1), point(q2)  DIM
) {

    scalar l1 = 1*sq_dist(p1,p0);
    scalar l2 = 1*sq_dist(p2,p0);
    scalar l3 = 1*sq_dist(p3,p0);

    scalar a10 = 2*dot_at(p1,q0,p0);
    scalar a11 = 2*dot_at(p1,q1,p0);
    scalar a12 = 2*dot_at(p1,q2,p0);
    scalar a20 = 2*dot_at(p2,q0,p0);
    scalar a21 = 2*dot_at(p2,q1,p0);
    scalar a22 = 2*dot_at(p2,q2,p0);

    scalar a30 = 2*dot_at(p3,q0,p0);
    scalar a31 = 2*dot_at(p3,q1,p0);
    scalar a32 = 2*dot_at(p3,q2,p0);

    scalar b00 = a11*a22-a12*a21;
    scalar b01 = a21-a22;
    scalar b02 = a12-a11;
    scalar b10 = a12*a20-a10*a22;
    scalar b11 = a22-a20;
    scalar b12 = a10-a12;
    scalar b20 = a10*a21-a11*a20;
    scalar b21 = a20-a21;
    scalar b22 = a11-a10;

    scalar Delta = b00+b10+b20;

    scalar DeltaLambda0 =
        b01*l1+b02*l2+b00;
    scalar DeltaLambda1 =
        b11*l1+b12*l2+b10;
    scalar DeltaLambda2 =
        b21*l1+b22*l2+b20;

    scalar r = Delta*l3 -
        (a30 * DeltaLambda0 +
         a31 * DeltaLambda1 +
         a32 * DeltaLambda2);

    scalar Delta_sign = sign(Delta) ;
    scalar r_sign = sign(r) ;

    generic_predicate_result( 
        Delta_sign*r_sign 
    ) ;

    begin_sos4(p0,p1,p2,p3)
        sos(p0, Sign(Delta_sign*sign( 
            Delta-(b01+b02)*a30+
            (b11+b12)*a31+
            (b21+b22)*a32) 
        ))
        sos(p1, Sign(Delta_sign* 
            sign((a30*b01)+
            (a31*b11)+
            (a32*b21))) 
        )
        sos(p2, Sign(Delta_sign*sign( 
            (a30*b02)+
            (a31*b12)+
            (a32*b22))) 
        )
        sos(p3, NEGATIVE)
    end_sos
}

Source PCK
Part. 2  Symbolic Perturbation – side4

q is the intersection of three bisectors and a tetrahedron embedded in nD

Note: There is a special case in 3d (no need for the tetrahedron) = insphere3d()

The code of the general nD version (excerpt):

```plaintext
scalar r = Delta*l4 - (a40*DeltaLambda0 + a41*DeltaLambda1 + a42*DeltaLambda2 + a43*DeltaLambda3);

Sign Delta_sign = sign(Delta);
generic_predicate_result(Delta_sign*sign(r));
begin_sos5(p0,p1,p2,p3,p4)
sos(p0, Sign( Delta_sign*sign(Delta - (b01+b02+b03)*a30 + (b11+b12+b13)*a31 + (b21+b22+b23)*a32 + (b31+b32+b33)*a33 )));
sos(p1, Sign( Delta_sign*sign(a30*b01+a31*b11+a32*b21+a33*b31)));
sos(p2, Sign( Delta_sign*sign(a30*b02+a31*b12+a32*b22+a33*b32)));
sos(p3, Sign( Delta_sign*sign(a30*b03+a31*b13+a32*b23+a33*b33)));
sos(p4, NEGATIVE)
end_sos
```

Source PCK
3
Results and Applications
Part 3. Results and Applications – Crash test 1/2

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part 3. Results and Applications – Crash test 1/2
Part 3. Results and Applications – Crash test 1/2
Part 3. Results and Applications – Crash test 1/2
Part 3. Results and Applications – Crash test 2/2
Part 3. Results and Applications – Crash test 2/2
Part 3. Results and Applications - RVD
Part 3. Results and Applications - RVD
Part 3. Results and Applications - RVD
Part 3. Results and Applications - RVD

```
-Cylinder: bash - Konsol

o-[RVD]  nb_v:732633 nb_f:720581 nb_b:0 tri:0 has_w:0 dim:3

/ ======[Result]===== /

o-[I/O]  Saving file out.obj...
o-[Topology]  M1: Xi=-40 nbB=0 nbConn=1
              M2: Xi=-40 nbB=0 nbConn=1
              match.
o-[Total time]  Elapsed time: 8.05 s
Everything OK, Returning status 0

/ ======[System Statistics]===== /

o-[orient2d]  Tot:0 Exact:0
              Exact: 0%
              Len: 0
o-[orient3d]  Tot:596243 Exact:8055
              Exact: 1.35096%
              Len: 3
o-[orient4d]  Tot:0 Exact:0 SOS:0
              Exact: 0%  SOS: 0%
              Num len: 0 Denom len: 0 SOS len: 0
o-[side1]    Tot:23007541 Exact:638 SOS:610
              Exact: 0.002773%  SOS: 0.0026513%
              Len: 2
o-[side2]    Tot:27854873 Exact:76501 SOS:75019
              Exact: 0.274641%  SOS: 0.269321%
              Num len: 15 Denom len: 6 SOS len: 6
o-[side3]    Tot:5440345 Exact:27341 SOS:25858
              Exact: 0.50256%  SOS: 0.475301%
              Num len: 29 Denom len: 15 SOS len: 12
o-[side4/insph.]  Tot:4466223 Exact:230637 SOS:220590
              Exact: 5.34314%  SOS: 5.16314%
              Num len: 16 Denom len: 6 SOS len: 6
o-[Process]  Total elapsed time: 8.06s
Maximum used memory: 808517632 (771M 64K )
```
If we eliminate the zero components during computations, the length of the expansions remain reasonable (see observation in Shewchuk’s paper)
Part 3. Results and Applications – MultiP needed?
Part 3. Results and Applications – MultiP needed ?

Not always !

```
o-[side1]   Tot:911810496 Exact:0 SOS:0
             Exact: 0%  SOS: 0%
             Len: 0

o-[side2]   Tot:638938211 Exact:0 SOS:0
             Exact: 0%  SOS: 0%
             Num len: 0 Denom len: 0 SOS len: 0

o-[side3]   Tot:43078577  Exact:0 SOS:0
             Exact: 0%  SOS: 0%
             Num len: 0 Denom len: 0 SOS len: 0
```
Part 3. Results and Applications – Hex-Dominant

[Ray and Sokolov, Periodic Global Parameterization in 3d]
Part 3. Results and Applications – 3D Anisotropic VD
Part 3. Results and Applications – Optimal transport

Uses a restricted power diagram [Aurenhammer et.al, Merigot et.al]
Part 3. Results and Applications – Optimal transport

Uses a restricted power diagram [Aurenhammer et.al, Merigot et.al]
Part 3. Results and Applications – Optimal transport

Uses a restricted power diagram [Aurenhammer et.al, Merigot et.al]
Part 3. Results and Applications – Optimal transport

Uses a restricted power diagram [Aurenhammer et.al, Merigot et.al]
Part 3. Results and Applications – Optimal transport

Uses a restricted power diagram [Aurenhammer et.al, Merigot et.al]
Acknowledgements

European Research Council
GOODSHAPE ERC-StG-205693
VORPALINE ERC-PoC-334829
ANR MORPHO, ANR BECASIM

J. Shewchuk for discussions about expansion arithmetics.
E. Maitre, Q. Merigot, B. Thibert for discussions about OT.
The PCK (Predicate Construction Kit)

Features:
- Expansion number type – low level API (allocation on stack, efficient)
- High level API with operators (easy to use, less efficient)
- Script to generate FPG filter and exact version with SOS
- Standard predicates (orient2d, 3d, insphere3d, orient4d)
- More exotic predicates for RVD (side1, side2, side3, side 4 in dim 3,4,6,7)
- 3D Delaunay triangulation
- 3D weighted Delaunay triangulation
- Only 6 source files
  - multi_precision.(h,cpp)
  - predicates.(h,cpp)
  - delaunay3d.(h,cpp)
- Fully documented
- No dependency (compiles and runs everywhere*, I got a version on my phone :-)

BSD license (do what you want with it, including business)
Available now (not on website yet, but you can send me an e-mail Bruno.Levy@inria.fr for a sneak preview)

*In the IEEE754 world