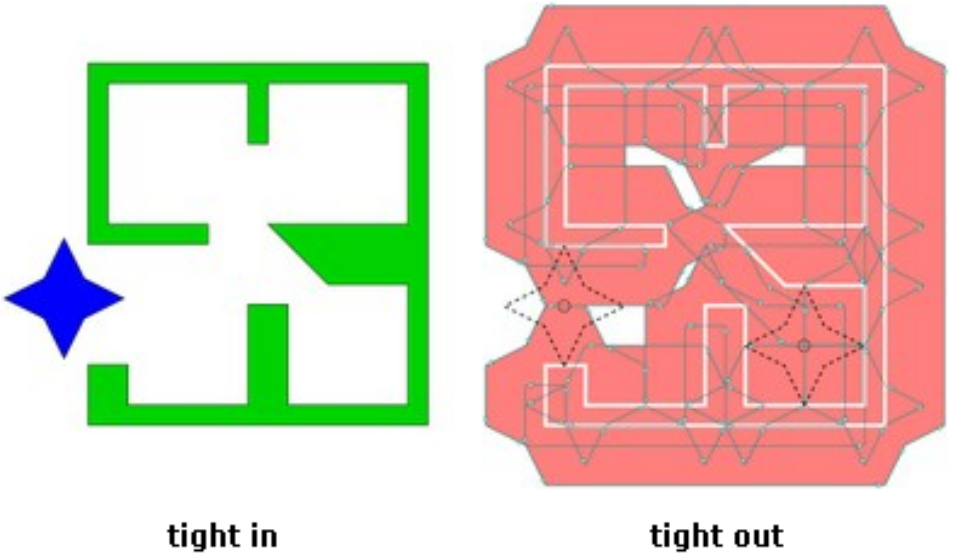
Geometric Computing

Computational Geometry - Winter 2013/2014

Eric Berberich

Some Applications of Geometric Algorithms

- Robot motion planning (polygonal input, no rotation)

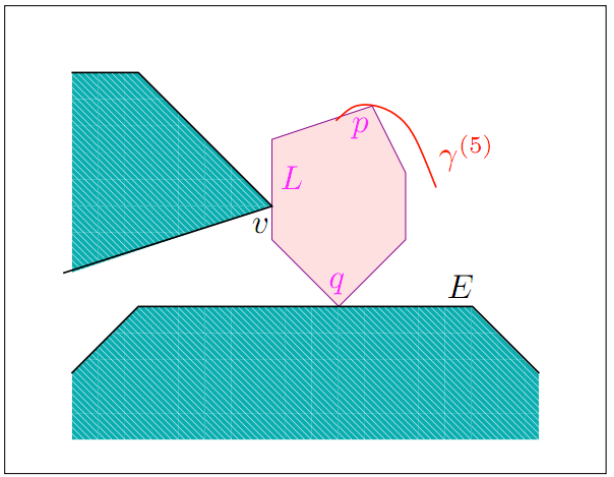


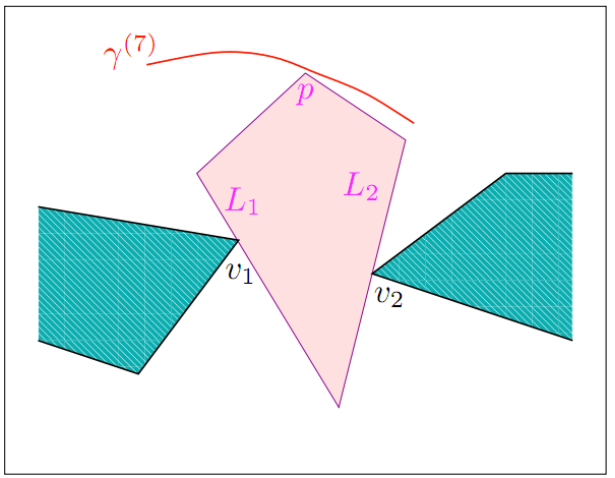
[Wein 2006]

- Robot motion planning (with rotation $\Rightarrow$ degree 4 curves)
- Voronoi diagram on a sphere
- Voronoi cells of lines in 3D
- Offset of a curve (e.g., ellipse $\Rightarrow$ degree 8 curve)
- Arrangement of Bézier Curves
- Intersection of solids (in Computer Aided Design)

Some Applications of Geometric Algorithms

- Robot motion planning (polygonal input, no rotation)
- Robot motion planning (with rotation $\Rightarrow$ degree 4 curves)



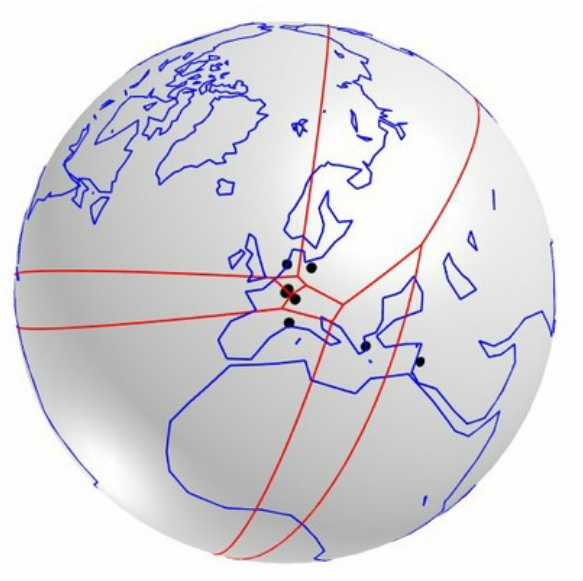


[Wein 2005]

- Voronoi diagram on a sphere
- Voronoi cells of lines in 3D
- Offset of a curve (e.g., ellipse $\Rightarrow$ degree 8 curve)
- Arrangement of Bézier Curves
- Intersection of solids (in Computer Aided Design)

Some Applications of Geometric Algorithms

- Robot motion planning (polygonal input, no rotation)
- Robot motion planning (with rotation $\Rightarrow$ degree 4 curves)
- Voronoi diagram on a sphere

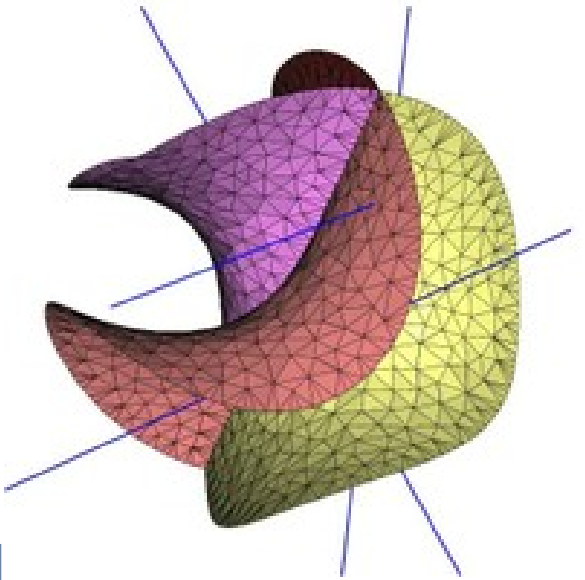


[Fogel et al. 2008]

- Voronoi cells of lines in 3D
- Offset of a curve (e.g., ellipse $\Rightarrow$ degree 8 curve)
- Arrangement of Bézier Curves
- Intersection of solids (in Computer Aided Design)

Some Applications of Geometric Algorithms

- Robot motion planning (polygonal input, no rotation)
- Robot motion planning (with rotation $\Rightarrow$ degree 4 curves)
- Voronoi diagram on a sphere
- Voronoi cells of lines in 3D

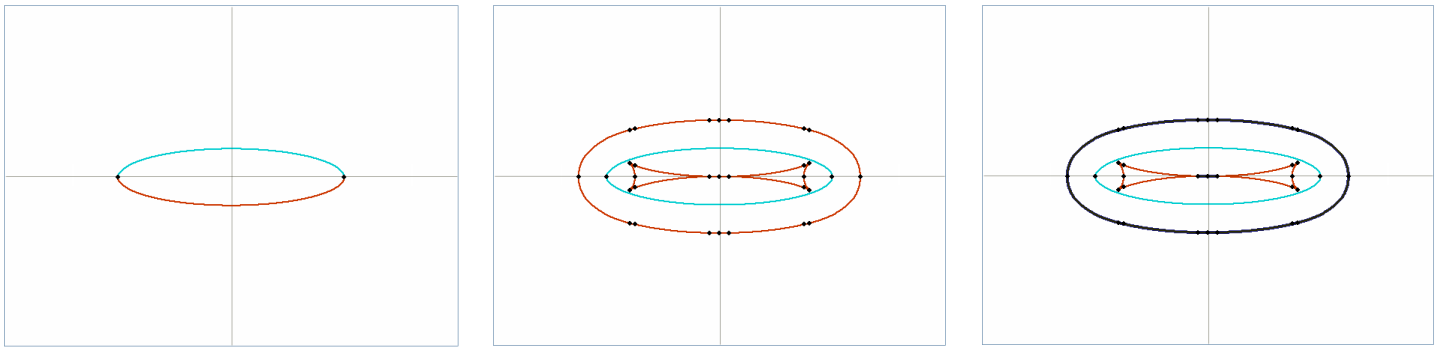


[Hemmer et al. 2010]

- Offset of a curve (e.g., ellipse $\Rightarrow$ degree 8 curve)
- Arrangement of Bézier Curves
- Intersection of solids (in Computer Aided Design)

Some Applications of Geometric Algorithms

- Robot motion planning (polygonal input, no rotation)
- Robot motion planning (with rotation $\Rightarrow$ degree 4 curves)
- Voronoi diagram on a sphere
- Voronoi cells of lines in 3D
- Offset of a curve (e.g., ellipse $\Rightarrow$ degree 8 curve)



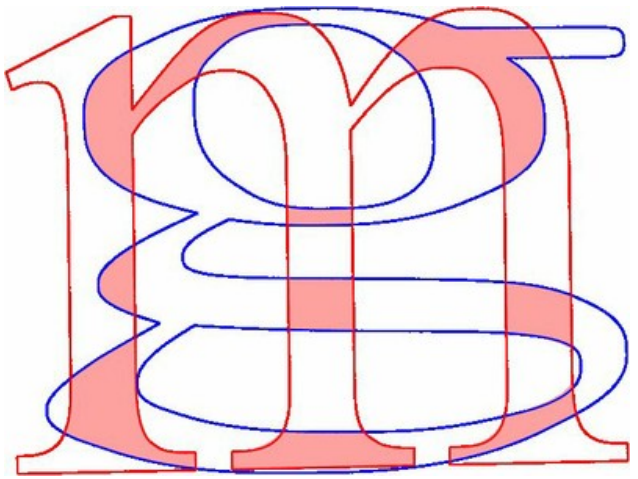
ellipse: $8x^2 + 127y^2 - 8 = 0$

offset: $67649929216x^8 - 38623220817920x^2y^4 - 8339109519360y^2x^4 + \cdots = 0$

- Arrangement of Bézier Curves
- Intersection of solids (in Computer Aided Design)

Some Applications of Geometric Algorithms

- Robot motion planning (polygonal input, no rotation)
- Robot motion planning (with rotation $\Rightarrow$ degree 4 curves)
- Voronoi diagram on a sphere
- Voronoi cells of lines in 3D
- Offset of a curve (e.g., ellipse $\Rightarrow$ degree 8 curve)
- Arrangement of Bézier Curves

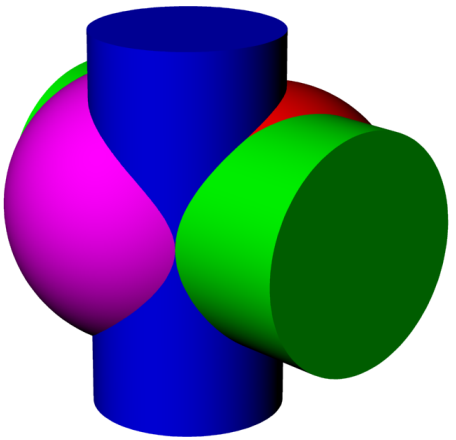


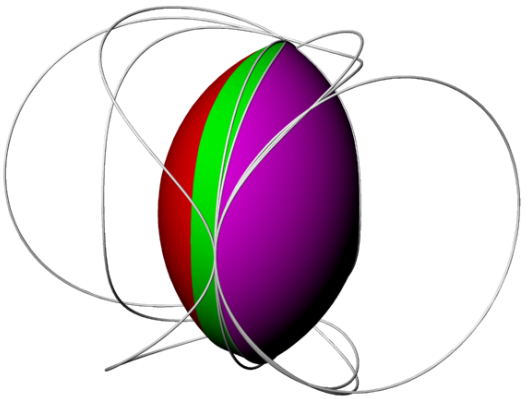
[Hanniel, Wein 2007]

- Intersection of solids (in Computer Aided Design)

Some Applications of Geometric Algorithms

- Robot motion planning (polygonal input, no rotation)
- Robot motion planning (with rotation $\Rightarrow$ degree 4 curves)
- Voronoi diagram on a sphere
- Voronoi cells of lines in 3D
- Offset of a curve (e.g., ellipse $\Rightarrow$ degree 8 curve)
- Arrangement of Bézier Curves
- Intersection of solids (in Computer Aided Design)





[Kettner et al. 2005]

Theoretical Description vs. Real Implementation

Real RAM model

The theoretical formulation of geometric algorithms is based on the **assumption that computations are carried out with exact real arithmetic**, i.e. correct evaluation of *predicates*

- "do circles C_1, C_2 and C_3 intersect in one common point?", or
- "does the point P lie on the left/right of the line L?"

The assumption of a Real RAM seems to be innocent, **but**

- computers only work with floating point arithmetic which is either bounded (hardware) or unbounded (software)
- using approximate computation induces errors which eventually lead to severe problems, that is, crashes, non-terminating algorithms, wrong results

Task: Convex Hull in 2D

Given: $S := \{p_1, \ldots, p_n\}$, with $p_i \in \mathbb{R}^2$

Convex Hull

Task: Convex Hull in 2D

Given: $S := \{p_1, \ldots, p_n\}$, with $p_i \in \mathbb{R}^2$

Convex Hull

- CH(S) *is the smallest convex set containing all points*
- $p \in S$ is *extreme* if there is a closed halfplane containing S such that p in the only point in S that lies on the boundary of the halfpace

Task: Convex Hull in 2D

Given: $S := \{p_1, \ldots, p_n\}$, with $p_i \in \mathbb{R}^2$

Convex Hull

- CH(S) *is the smallest convex set containing all points*
- $p \in S$ is *extreme* if there is a closed halfplane containing S such that p in the only point in S that lies on the boundary of the halfpace

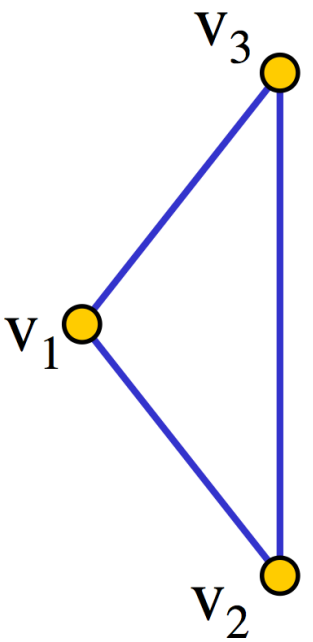
We maintain CH(S) as cyclically ordered (counter-clockwise) list L of its extreme points: e.g. $L = (v_1, v_4, v_5, v_7)$

First Algorithm

Convex Hull

- initialize L to a counter-clockwise triangle[a] (a, b, c) with $a, b, c \in S$ and remove them from S
- for each $r \in S$ do
  - if there is an edge e visible from r, determine the sequence $((v_i, v_{i+1}), (v_{i+1}, v_{i+2}), ...(v_{j-1}, v_j))$ of edges that are *weakly visible* from r (can be empty)
  - replace the subsequence $(v_{i+1}, ...v_{j-1})$ in L by r ("adding tangents") and remove r from S

[a] we assume that a, b, c are not collinear

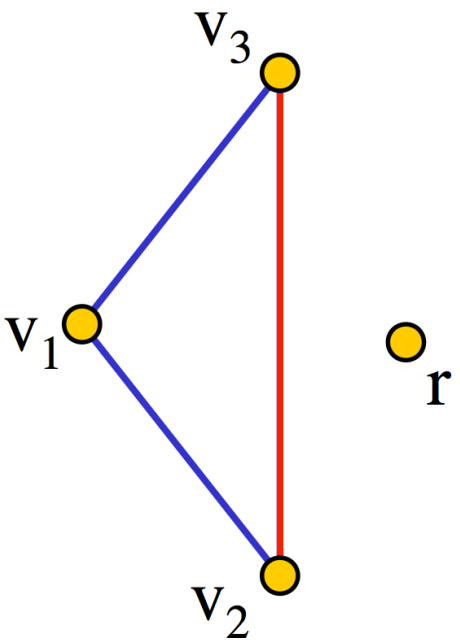


First Algorithm

Convex Hull

- initialize L to a counter-clockwise triangle[a] (a, b, c) with $a, b, c \in S$ and remove them from S
- for each $r \in S$ do
  - if there is an edge e visible from r, determine the sequence $((v_i, v_{i+1}), (v_{i+1}, v_{i+2}), ...(v_{j-1}, v_j))$ of edges that are *weakly visible* from r (can be empty)
  - replace the subsequence $(v_{i+1}, ...v_{j-1})$ in L by r ("adding tangents") and remove r from S

[a] we assume that a, b, c are not collinear

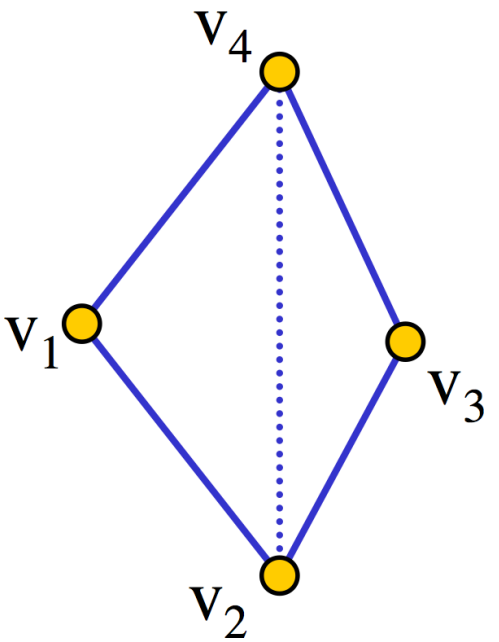


First Algorithm

Convex Hull

- initialize L to a counter-clockwise triangle[a] (a, b, c) with $a, b, c \in S$ and remove them from S
- for each $r \in S$ do
  - if there is an edge e visible from r, determine the sequence $((v_i, v_{i+1}), (v_{i+1}, v_{i+2}), ...(v_{j-1}, v_j))$ of edges that are *weakly visible* from r (can be empty)
  - replace the subsequence $(v_{i+1}, ...v_{j-1})$ in L by r ("adding tangents") and remove r from S

[a] we assume that a, b, c are not collinear

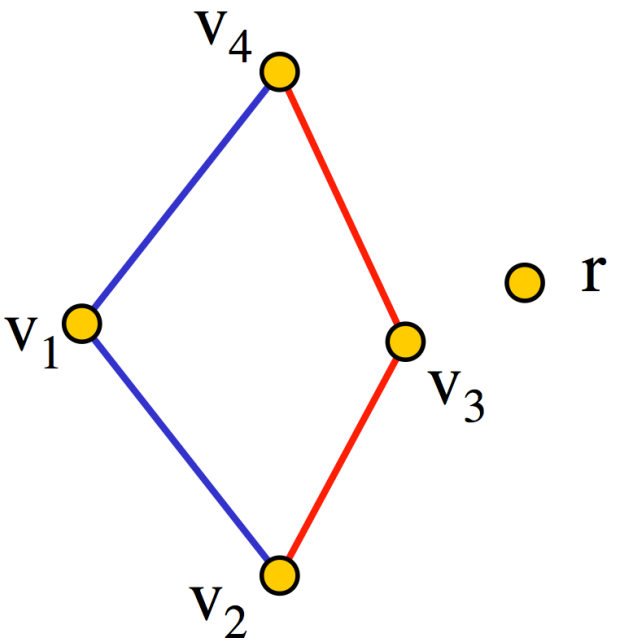


First Algorithm

Convex Hull

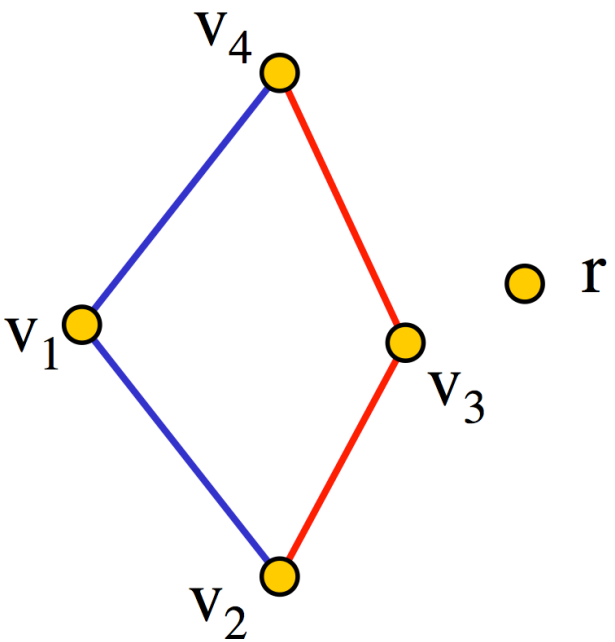
- initialize L to a counter-clockwise triangle[a] (a, b, c) with $a, b, c \in S$ and remove them from S
- for each $r \in S$ do
  - if there is an edge e visible from r, determine the sequence $((v_i, v_{i+1}), (v_{i+1}, v_{i+2}), ...(v_{j-1}, v_j))$ of edges that are *weakly visible* from r (can be empty)
  - replace the subsequence $(v_{i+1}, ...v_{j-1})$ in L by r ("adding tangents") and remove r from S

[a] we assume that a, b, c are not collinear



Correctness Properties

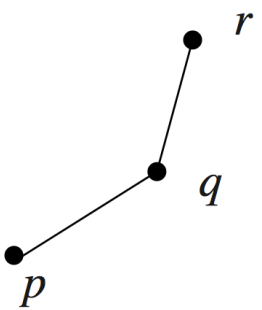
- A point r is outside of CH(S) iff there is an i such that the edge (v_i, v_{i+1}) is *visible* from r ( ).
- If r is outside of CH(S), then the set of edges that are *weakly visible* ( ) from r forms a non-empty consecutive subchain; so does the set of edges that a not weakly visible from r (i.e. list is partitioned into two non-empty contiguous sets!)

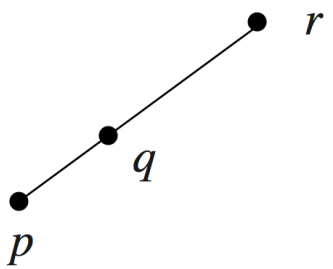


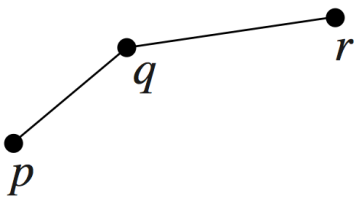
Correctness Properties and Orientation

- A point r is outside of CH(S) iff there is an i such that the edge (v_i, v_{i+1}) is *visible* from r (orientation$(v_i, v_{i+1}, r) > 0$).
- If r is outside of CH(S), then the set of edges that are *weakly visible* (orientation is non-negative) from r forms a non-empty consecutive subchain; so does the set of edges that a not weakly visible from r (i.e. list is partitioned into two non-empty contiguous sets!)

$$\text{orientation}(p,q,r) = \begin{cases} -1 & \text{if } p \neq q \text{ and } r \text{ lies to the left of } l(p,q) \\ 0 & \text{if } p = q \text{ or } p \neq q \text{ and } r \text{ lies on } l(p,q) \\ +1 & \text{if } p \neq q \text{ and } r \text{ lies to the right of } l(p,q) \end{cases}$$







Analysis

$O(n^2)$

for each of n points processed

- each edge of current hull is checked for visibility $O(n)$
- remove zero or more points from the current hull $O(n)$

We can make it down to $O(n \log n)$:

- at most $O(n)$ update steps
- sorting points lexicographically $O(n \log n)$
  - every new point is outside the CH
  - one of the edges incident to the lexicographically largest vertex is visible from new point

there are other convex hull algorithms, our focus is different:

Analysis

$O(n^2)$

for each of n points processed

- each edge of current hull is checked for visibility $O(n)$
- remove zero or more points from the current hull $O(n)$

We can make it down to $O(n \log n)$:

- at most $O(n)$ update steps
- sorting points lexicographically $O(n \log n)$
  - every new point is outside the CH
  - one of the edges incident to the lexicographically largest vertex is visible from new point

there are other convex hull algorithms, our focus is different:

Analysis

$O(n^2)$

for each of n points processed

- each edge of current hull is checked for visibility $O(n)$
- remove zero or more points from the current hull $O(n)$

We can make it down to $O(n \log n)$:

- at most $O(n)$ update steps
- sorting points lexicographically $O(n \log n)$
  - every new point is outside the CH
  - one of the edges incident to the lexicographically largest vertex is visible from new point

there are other convex hull algorithms, our focus is different:

Orientation Predicate

$$
\begin{aligned}
\mathit{orientation}(p,q,r) &= \operatorname{sign}\begin{vmatrix} 1 & 1 & 1 \\ p_x & q_x & r_x \\ p_y & q_y & r_y \end{vmatrix} \\
&= \operatorname{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))
\end{aligned}
$$

- $(q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)$ is equal to twice the signed area of triangle pqr
  - *negative* if p, q, r are oriented counterclockwise around the triangle
  - *positive* if p, q, r are oriented clockwise around the triangle
- can be shown by cross-products of the vectors $\vec{pq}$ and $\vec{pr}$
- orientation follows from signed area of triangle

Exchanging two points flips the sign, ie. the orientation

Are we ready to implement?

Ideally: Real RAM

Machine where each arithmetic operation is exact and takes time $O(1)$

Idea: *Implementation of a Real RAM = RAM +* `double`

Not a good idea! We see instances where correctness properties are violated:

- a point r outside sees no edge
- a point r outside sees all edges
- a point r inside sees an edge
- a point r outside sees a non-consecutive set of edges

Geometry of Float-Orientation

$$orientation(p,q,r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

$0 \leq, I, J < 256$ and $u = 2^{-53}$

$$p = \begin{pmatrix} 0.5 + I \cdot u \\ 0.5 + J \cdot u \end{pmatrix}$$

$$q = \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$r = \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$

What do you expect?

- creating 256x256 pixel image
- pixel (x, y) shows $orientation(p(x, y), q, r)$:
  negative (left-turn), collinear, positive (right-turn)

Geometry of Float-Orientation

$$orientation(p,q,r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

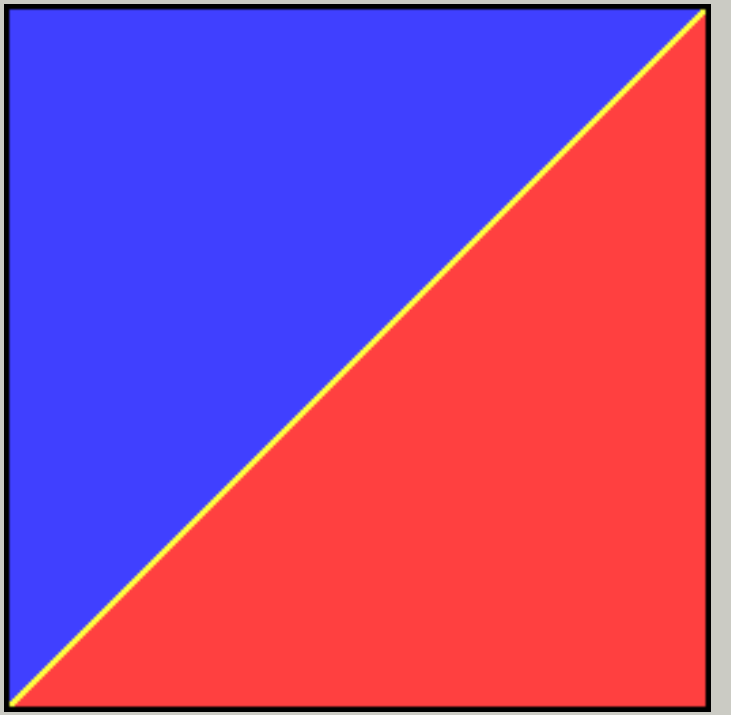
$0 \leq, I, J < 256$ and $u = 2^{-53}$

$$p = \begin{pmatrix} 0.5 + I \cdot u \\ 0.5 + J \cdot u \end{pmatrix}$$

$$q = \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$r = \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$

Maybe this?



- creating 256x256 pixel image
- pixel (x, y) shows $orientation(p(x, y), q, r)$:
  negative (left-turn), collinear, positive (right-turn)

Geometry of Float-Orientation

$$orientation(p,q,r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

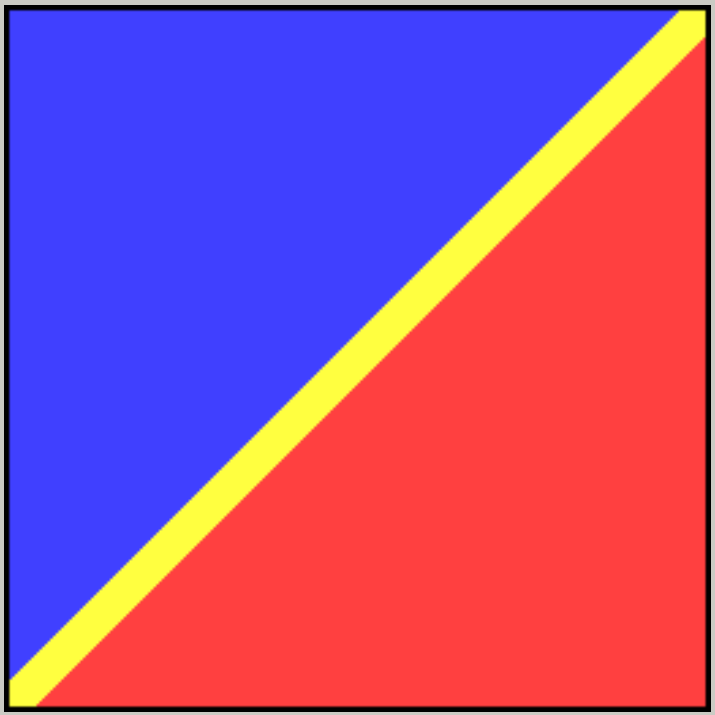
$0 \leq, I, J < 256$ and $u = 2^{-53}$

$$p = \begin{pmatrix} 0.5 + I \cdot u \\ 0.5 + J \cdot u \end{pmatrix}$$

$$q = \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$r = \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$

Looks weird!?



- creating 256x256 pixel image
- pixel (x, y) shows $orientation(p(x, y), q, r)$:
  negative (left-turn), collinear, positive (right-turn)

Geometry of Float-Orientation

$$orientation(p,q,r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

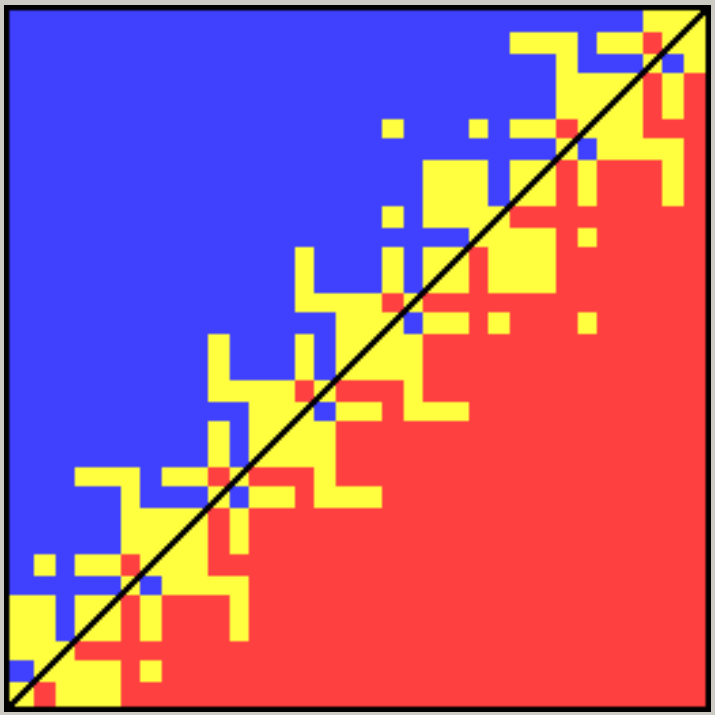
$0 \leq, I, J < 256$ and $u = 2^{-53}$

$$p = \begin{pmatrix} 0.5 + I \cdot u \\ 0.5 + J \cdot u \end{pmatrix}$$

$$q = \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$r = \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$

Truth is "artistic"



- creating 256x256 pixel image
- pixel (x, y) shows $orientation(p(x, y), q, r)$:
  negative (left-turn), collinear, positive (right-turn)

Geometry of Float-Orientation

$$orientation(p,q,r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

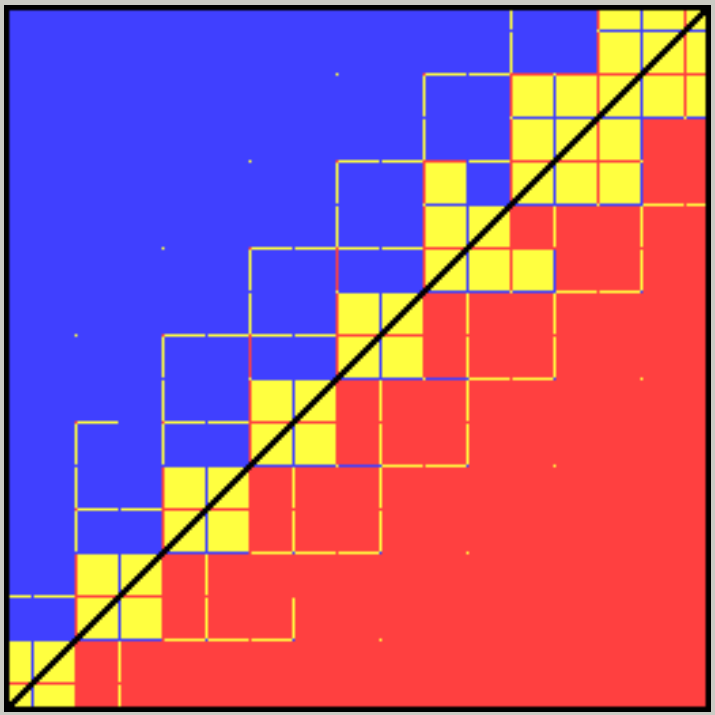
$0 \leq, I, J < 256$ and $u = 2^{-53}$

$$p = \begin{pmatrix} 0.5 + I \cdot u \\ 0.5 + J \cdot u \end{pmatrix}$$

$$q = \begin{pmatrix} 8.8000000000000007 \\ 8.8000000000000007 \end{pmatrix}$$

$$r = \begin{pmatrix} 12.1 \\ 12.1 \end{pmatrix}$$

Truth is "artistic"



- creating 256x256 pixel image
- pixel (x, y) shows $orientation(p(x, y), q, r)$:
  negative (left-turn), collinear, positive (right-turn)

Geometry of Float-Orientation

$$orientation(p,q,r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

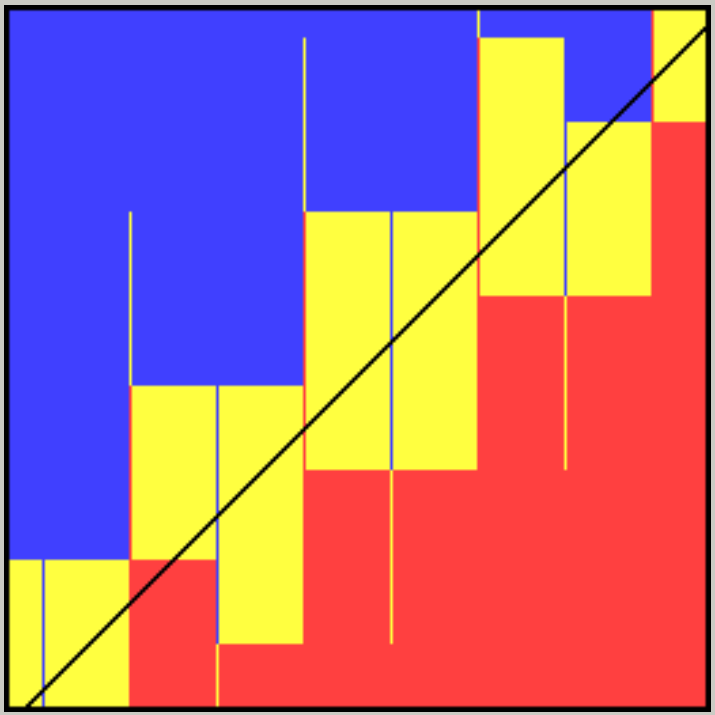
$0 \leq, I, J < 256$ and $u = 2^{-53}$

$$p = \begin{pmatrix} 0.50000000000002531 + I \cdot u \\ 0.50000000000001710 + J \cdot u \end{pmatrix}$$

$$q = \begin{pmatrix} 17.300000000000001 \\ 17.300000000000001 \end{pmatrix}$$

$$r = \begin{pmatrix} 24.000000000000500000 \\ 24.000000000000517765 \end{pmatrix}$$

Truth is “artistic”



- creating 256x256 pixel image
- pixel (x, y) shows $orientation(p(x, y), q, r)$:
  negative (left-turn), collinear, positive (right-turn)

Floating-point `double`

$\pm s 2^e$

- *significant*[a] $s = 1.s_1 s_2 \ldots s_{52}, s_i \in 0, 1$
- *exponent* $-1023 < e < 1024$

[a] Denormalized numbers do not play a role here

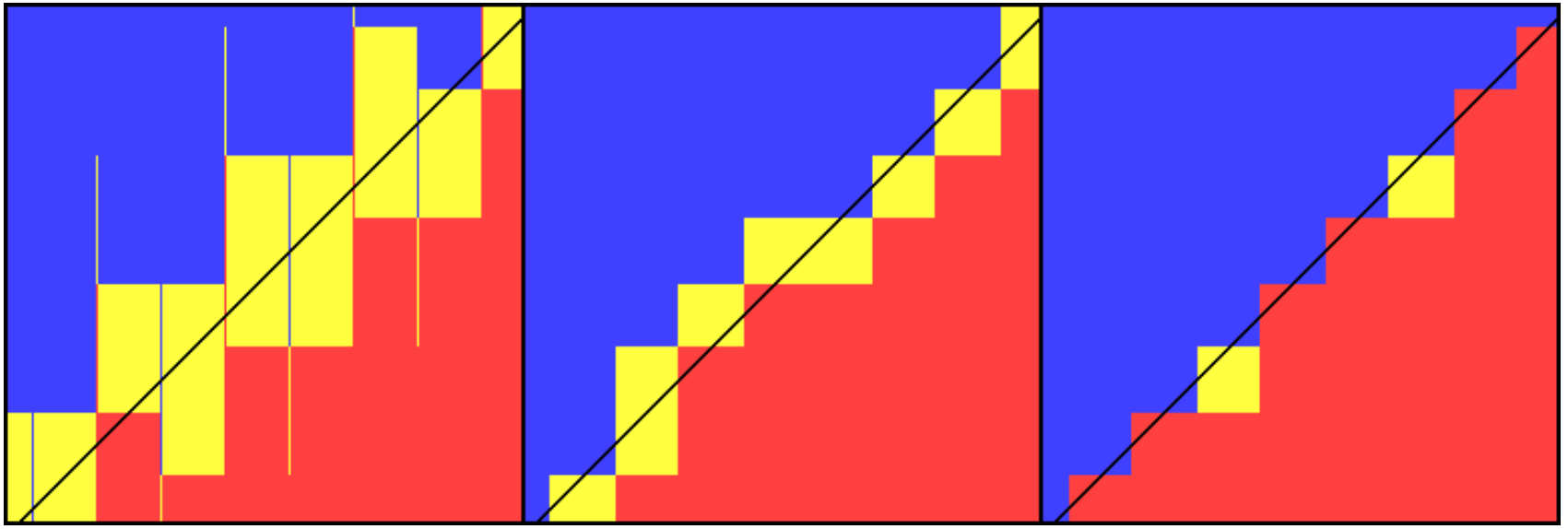
Take away: Floating point arithmetic is not exact, **it rounds**!

Two digits precision:

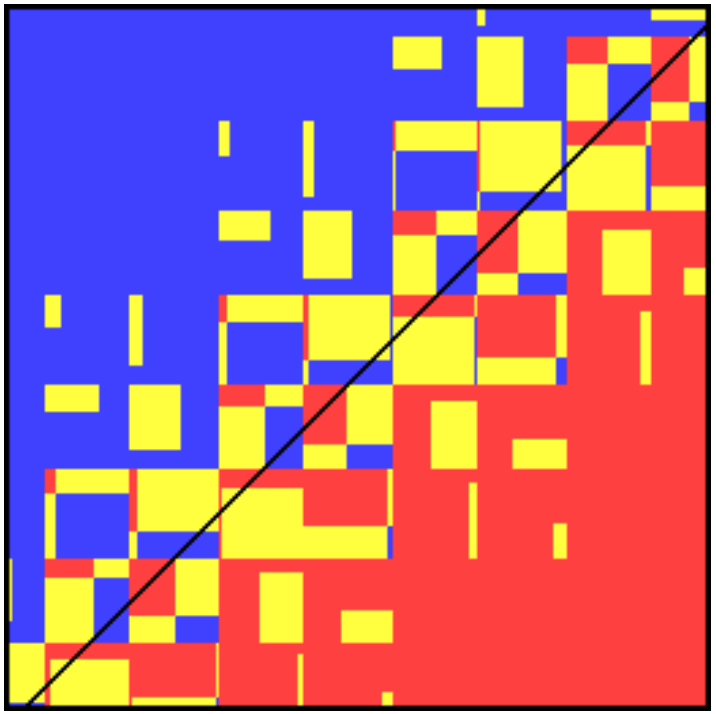
$$0.36 \cdot 0.11 = 0.40 \text{ (real: } 0.396)$$

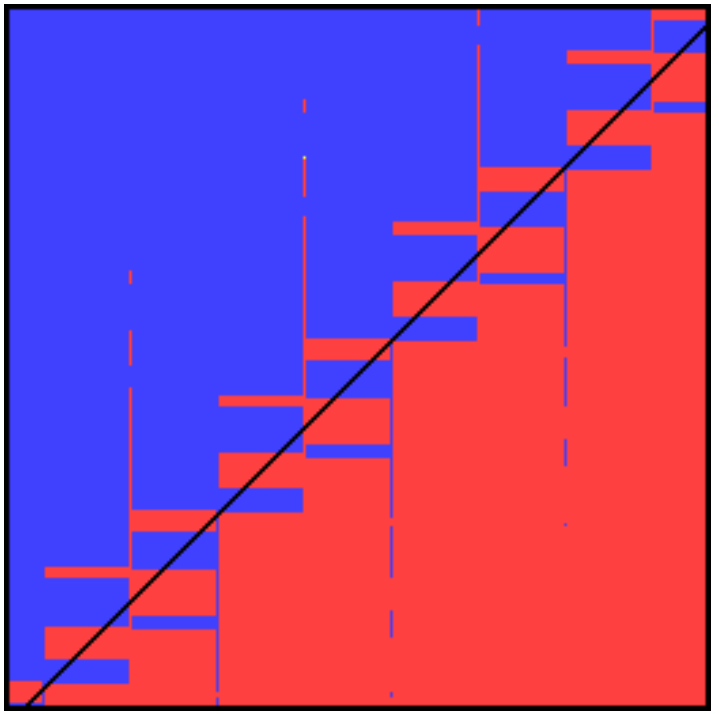
Changing order?

Changing order? Still wrong!



Extended Double Arithmetic



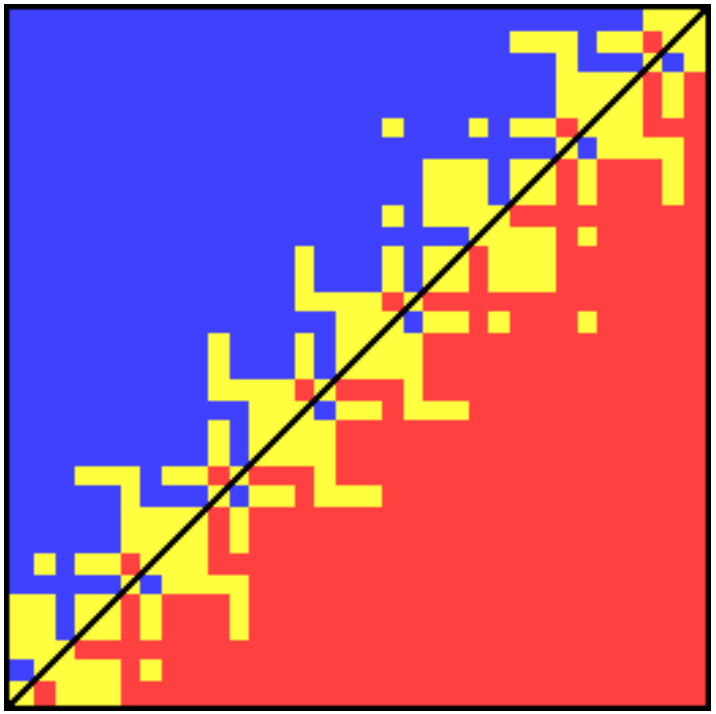


- evaluate $(q_x - p_x)(r_y - p_y)$ and $(q_y - p_y)(r_x - p_x)$ in extended double
- convert to double and compare

- all orientation in extended double

But why block structure?

$$orientation(p,q,r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

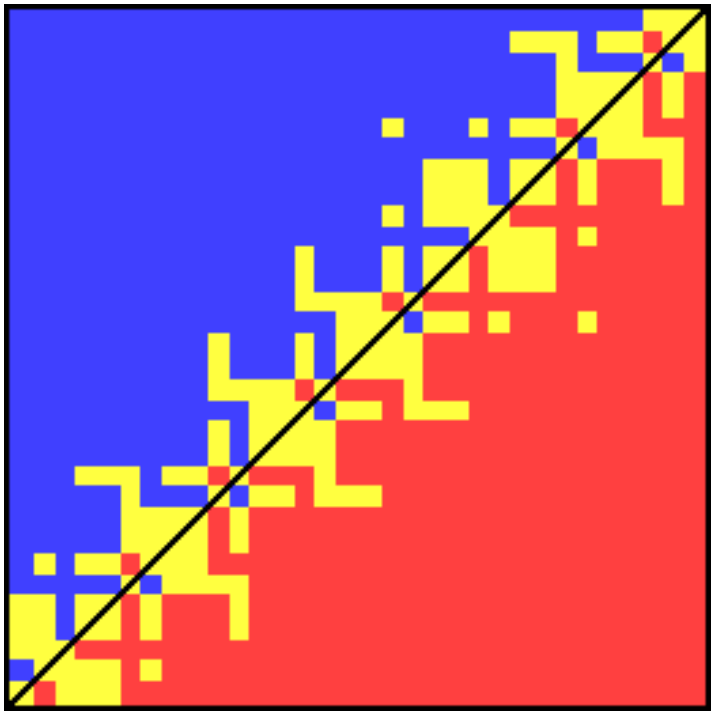


- round-off errors occur in substraction and multiplication
- for $q_x = 12$ and $p_x = 0.5 + lu \approx 0.5$ consider $q_x - p_x$
- 12 has 4 binary digits, so we lose the last 4 bits of p_x
- difference $q_x - p_x$ stays ***constant* for 2^4 values of l**
- rounding to nearest creates a block

- similar for other substractions and multiplications

But why block structure?

$$orientation(p,q,r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$



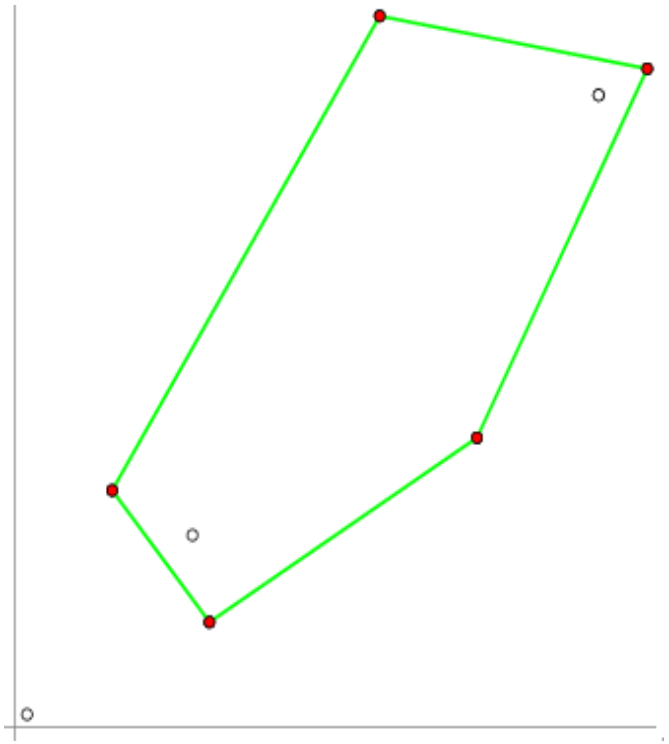
- round-off errors occur in substraction and multiplication
- for $q_x = 12$ and $p_x = 0.5 + lu \approx 0.5$ consider $q_x - p_x$
- 12 has 4 binary digits, so we lose the last 4 bits of p_x
- difference $q_x - p_x$ stays ***constant* for 2^4 values of l**
- rounding to nearest creates a block

- similar for other substractions and multiplications

Global Consequences

- A point outside sees no edge of the current hull.

p_1=(7.3000000000000194, 7.3000000000000167)
p_2=(24.000000000000068, 24.000000000000071)
p_3=(24.00000000000005, 24.000000000000053)
p_4=(0.50000000000001621, 0.50000000000001243)
p_5=(8, 4) p_6=(4, 9) p_7=(15, 27)
p_8=(26, 25) p_9=(19, 11)



float_orient(p_1,p_2,p_3) > 0
float_orient(p_1,p_2,p_4) < 0
float_orient(p_2,p_3,p_4) < 0
float_orient(p_3,p_1,p_4) < 0

Global Consequences

- A point outside sees all edges of the current hull.

p_1=(200, 49.200000000000003)
p_2=(100, 49.600000000000001)
p_3=(-233.33333333333334, 50.93333333333333)
p_4=(166.66666666666669, 49.333333333333336)

float_orient(p_1,p_2,p_3) > 0
float_orient(p_1,p_2,p_4) > 0
float_orient(p_2,p_3,p_4) > 0
float_orient(p_3,p_1,p_4) > 0

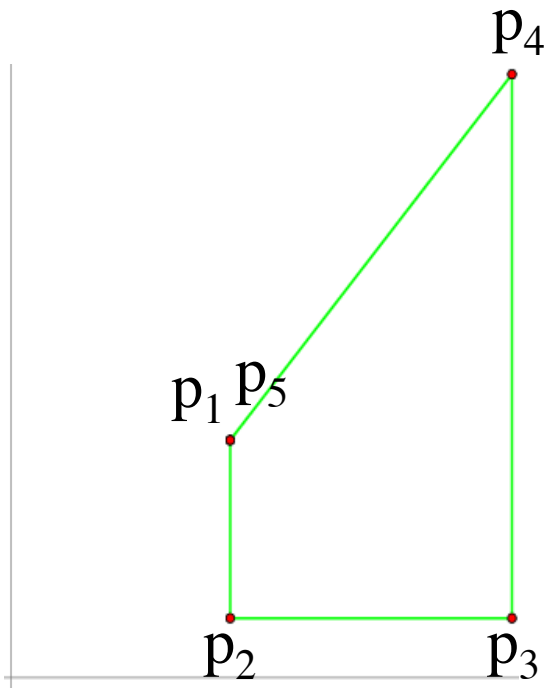
Depending on the implementation:

- **Algorithm does not terminate!**
- **Algorithm crashes!**

Global Consequences

- A point inside sees an edge of the current hull.

p_1=(24.000000000000005, 24.000000000000053)
p_2=(24.0, 6.0)
p_3=(54.85, 6.0)
p_4= (54.850000000000357, 61.000000000000121)
p_5=(24.000000000000068, 24.000000000000071)

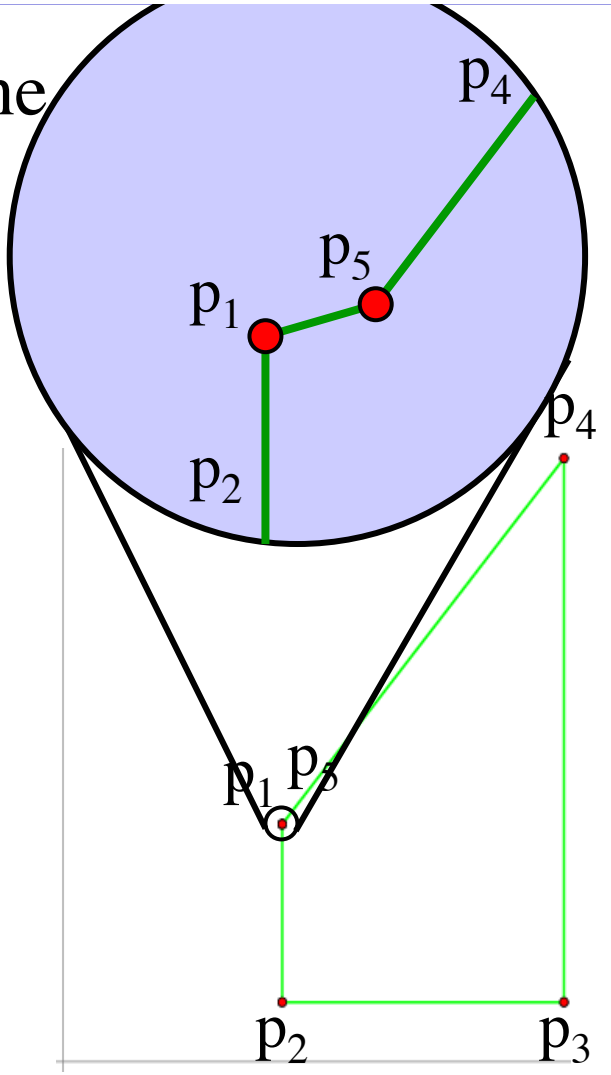


(p_1,p_2,p_3,p_4) form a convex quadrilateral
p5 is truly inside this quadrilateral, but
float_orient(p_4,p_1,p_5) > 0

Global Consequences

- A point inside sees an edge of the

p_1=(24.00000000000005, 24.000000000000053)
p_2=(24.0, 6.0)
p_3=(54.85, 6.0)
p_4= (54.850000000000357, 61.000000000000121)
p_5=(24.000000000000068, 24.000000000000071)



(p_1,p_2,p_3,p_4) form a convex quadrilateral
p5 is truly inside this quadrilateral, but
float_orient(p_4,p_1,p_5) > 0

Global Consequences

- A point outside sees a non-contigu

p_1=(24.000000000000005, 24.000000000000053)

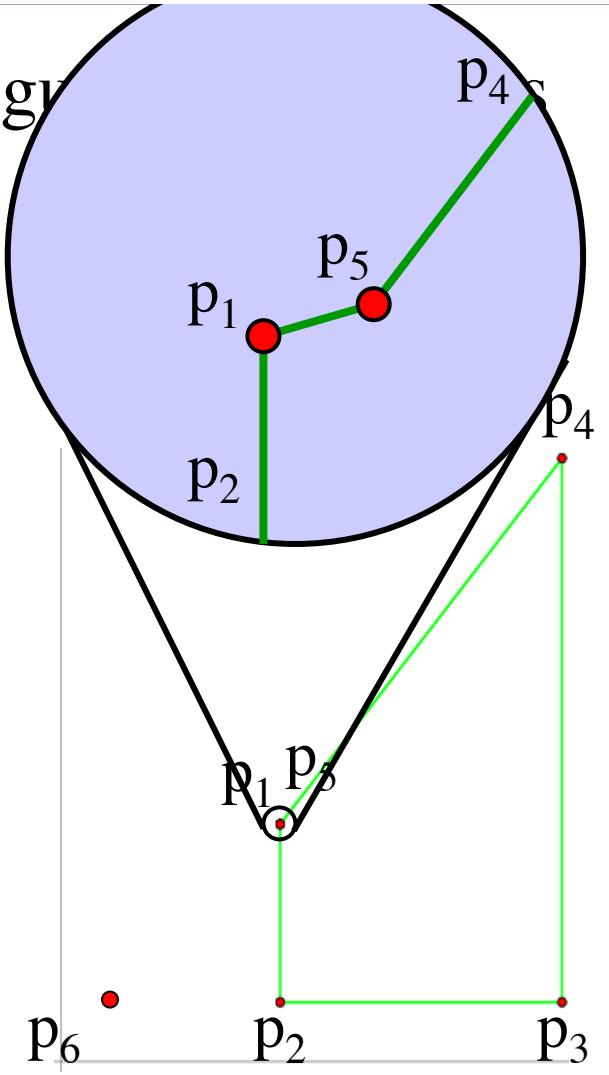
p_2=(24.0, 6.0)

p_3=(54.85, 6.0)

p_4= (54.850000000000357, 61.000000000000121)

p_5=(24.000000000000068, 24.000000000000071)

p_6=(6, 6)

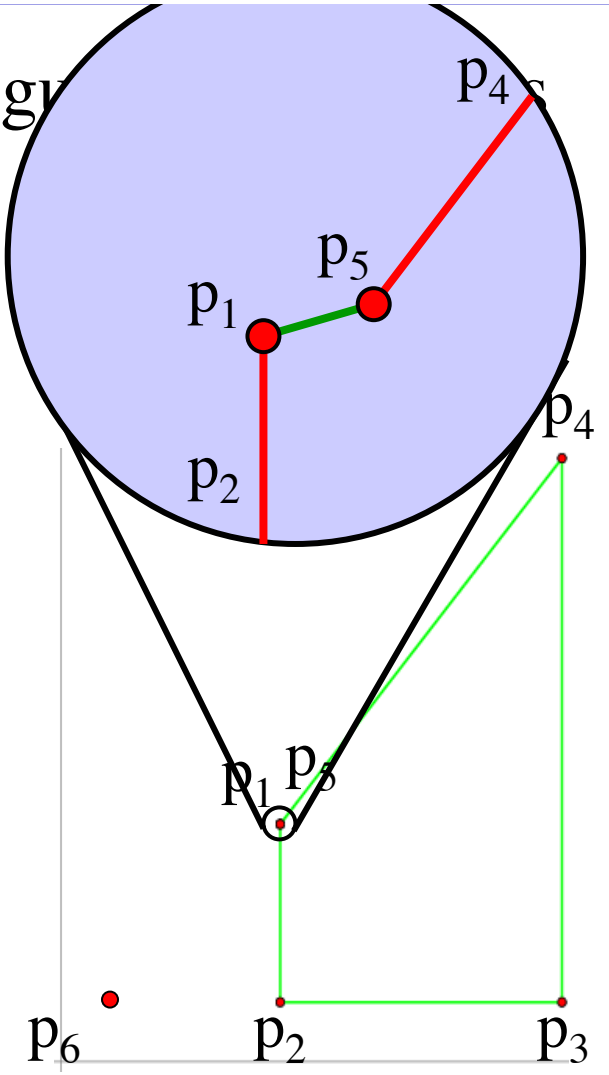


Global Consequences

- A point outside sees a non-contigu

p_1=(24.00000000000005, 24.000000000000053)
p_2=(24.0, 6.0)
p_3=(54.85, 6.0)
p_4= (54.850000000000357, 61.000000000000121)
p_5=(24.000000000000068, 24.000000000000071)
p_6=(6, 6)

float_orient(p_4,p_5,p_6) > 0
float_orient(p_5,p_1,p_6) < 0
float_orient(p_1,p_2,p_6) > 0



Global Consequences

- A point outside sees a non-contigu

p_1=(24.00000000000005, 24.000000000000053)

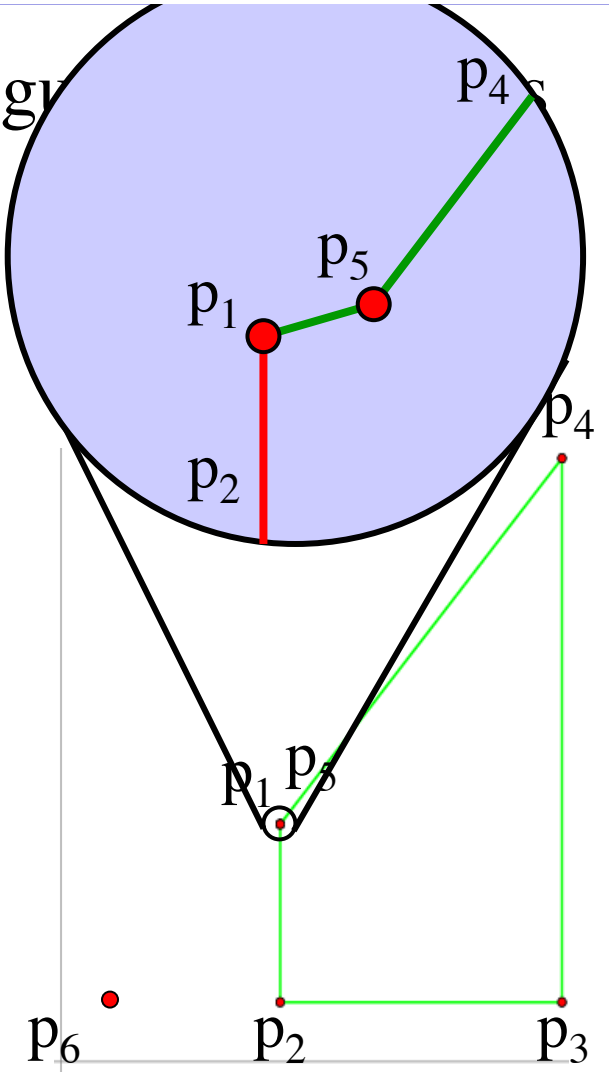
p_2=(24.0, **10.0**)

p_3=(54.85, 6.0)

p_4= (54.850000000000357, 61.000000000000121)

p_5=(24.000000000000068, 24.000000000000071)

p_6=(6, 6)



Global Consequences

- A point outside sees a non-contigu

p_1=(24.000000000000005, 24.000000000000053)

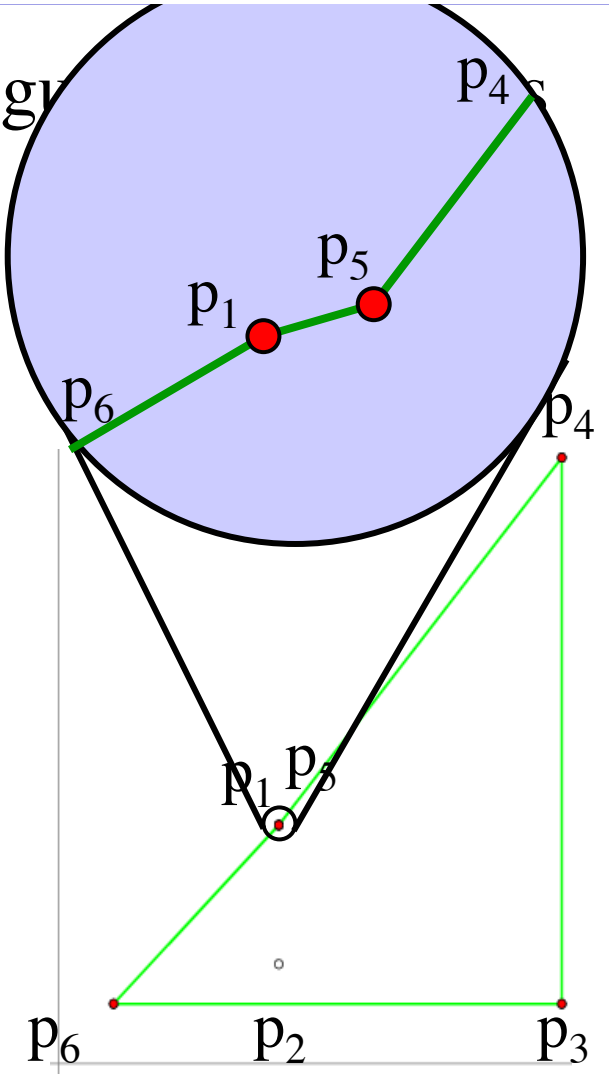
p_2=(24.0, **10.0**)

p_3=(54.85, 6.0)

p_4= (54.850000000000357, 61.000000000000121)

p_5=(24.000000000000068, 24.000000000000071)

p_6=(6, 6)



Global Consequences

- A point outside sees a non-contigu

p_1=(24.00000000000005, 24.000000000000053)

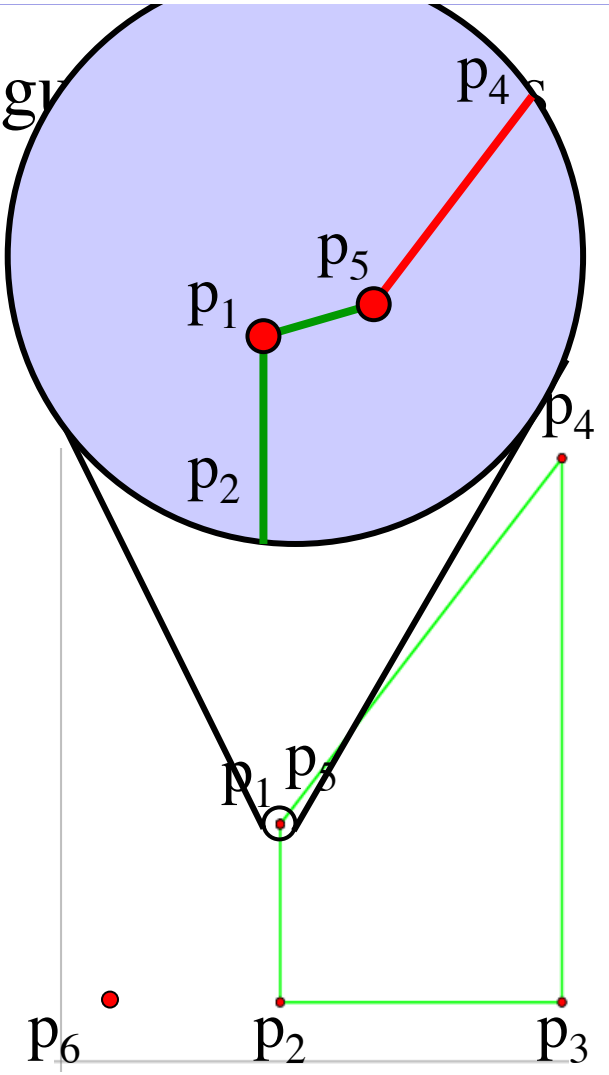
p_2=(24.0, **6.0**)

p_3=(54.85, 6.0)

p_4= (54.850000000000357, 61.000000000000121)

p_5=(24.000000000000068, 24.000000000000071)

p_6=(6, 6)



Global Consequences

- A point outside sees a non-contiguous set of edges

p_1=(24.000000000000005, 24.000000000000053)

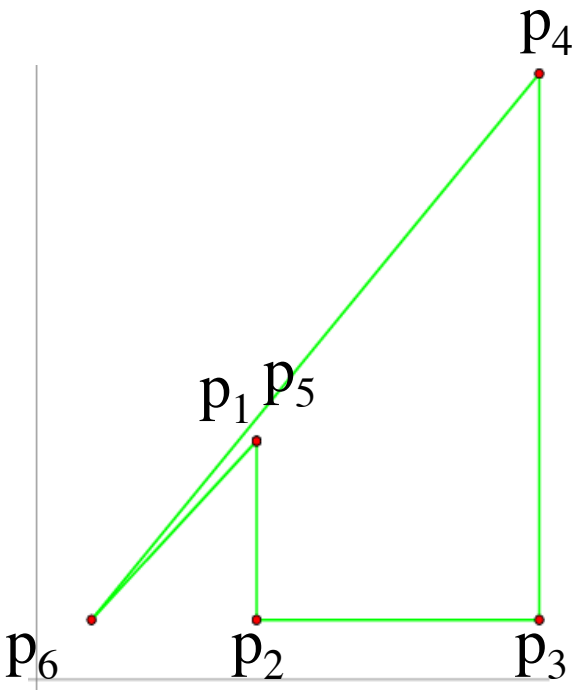
p_2=(24.0, **6.0**)

p_3=(54.85, 6.0)

p_4= (54.850000000000357, 61.000000000000121)

p_5=(24.000000000000068, 24.000000000000071)

p_6=(6, 6)



Global Consequences

- A point outside sees a non-contigu

p_1=(24.000000000000005, 24.000000000000053)

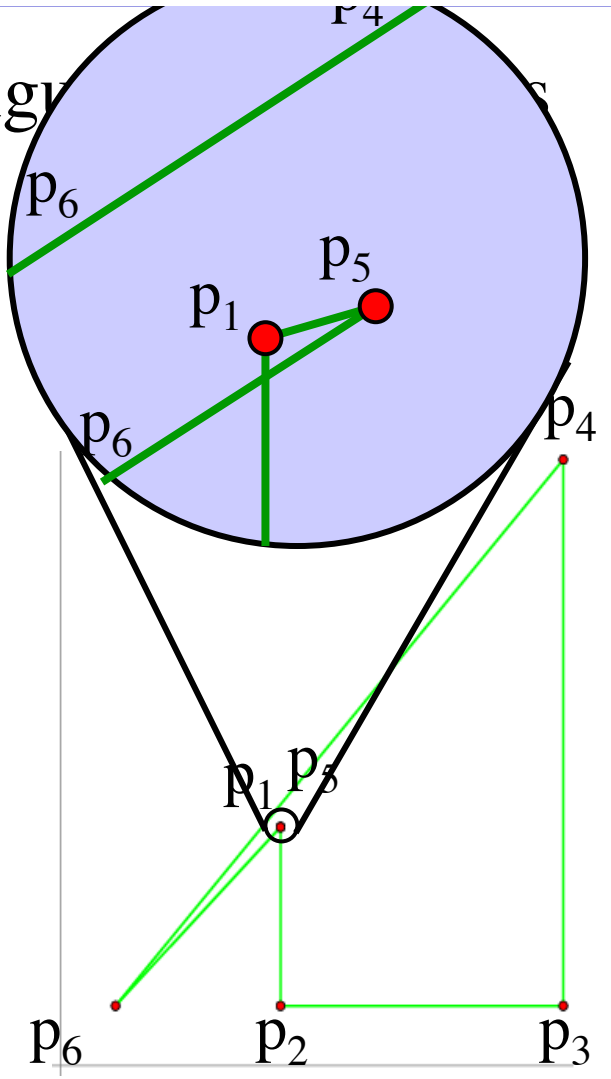
p_2=(24.0, **6.0**)

p_3=(54.85, 6.0)

p_4= (54.850000000000357, 61.000000000000121)

p_5=(24.000000000000068, 24.000000000000071)

p_6=(6, 6)



Non solutions

Idea 1: “start roundish or with extreme”

Counter-arguments:

- example C was roundish, modifiable to start with extremes
- nearly collinear points can still occur

Idea 2: *Epsilon-Tweaking*: let $0 := [-\epsilon, +\epsilon]$

(rounding to zero) makes it even worse, as more points outside the current line, do not see any edge (enforced collinearity)

Idea 3: Data is unrealistic

- practical experience
- degeneracies (collinearities) are on purpose in data sets (CAD, architecture)
- using floating-point will always lead to rounding errors
- larger data sets increase probability of collinearity

Non solutions

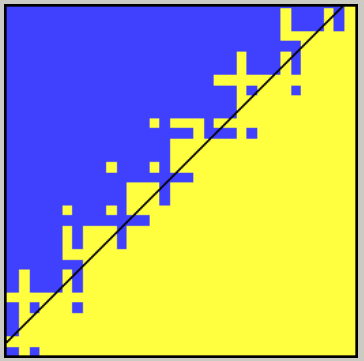
Idea 1: “start roundish or with extreme”

Counter-arguments:

- example C was roundish, modifiable to start with extremes
- nearly collinear points can still occur

Idea 2: *Epsilon-Tweaking*: let $0 := [-\epsilon, +\epsilon]$

(rounding to zero) makes it even worse, as more points outside the current line, do not see any edge (enforced collinearity)



$$\epsilon = 10^{-10}$$

$$p_{left} = \begin{pmatrix} 0.5 \\ 0.50000000000833222 \end{pmatrix}$$

$$q = \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$r = \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$

Idea 3: Data is unrealistic

- practical experience
- degeneracies (collinearities) are on purpose in data sets

Non solutions

Idea 1: "start roundish or with extreme"

Counter-arguments:

- example C was roundish, modifiable to start with extremes
- nearly collinear points can still occur

Idea 2: *Epsilon-Tweaking*: let $0 := [-\epsilon, +\epsilon]$

(rounding to zero) makes it even worse, as more points outside the current line, do not see any edge (enforced collinearity)

Idea 3: Data is unrealistic

- practical experience
- degeneracies (collinearities) are on purpose in data sets (CAD, architecture)
- using floating-point will always lead to rounding errors
- larger data sets increase probability of collinearity

But numerical analysis also uses floats!?

Right. But research there is aware of floating-point pitfalls:

- many numerical algorithms are self-correcting (errors will be remedied later)
- in contrast to geometric algorithms

Why all that mess?

Take Home Message I

The combinatorial (convex hull/geometric) algorithm is conducted by evaluations of non-continuous functions.

Obvious *discontinuity* in:

$$\text{orientation}(p, q, r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

Why no output that is "close" to correct hull?

- points far away may be classified as "inside"
- misclassified point may create a slightly non-convex polygon, which gets amplified

Why all that mess?

Take Home Message I

The combinatorial (convex hull/geometric) algorithm is conducted by evaluations of non-continuous functions.

Obvious *discontinuity* in:

$$\text{orientation}(p, q, r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

Why no output that is "close" to correct hull?

- points far away may be classified as "inside"
- misclassified point may create a slightly non-convex polygon, which gets amplified

The Mathematician's Solution

Modify your algorithm/implementation in way such that

- all (also degenerate) cases can be handled, and
- all predicates are evaluated exactly.

The latter can be "achieved" since

- we only produce *algebraic numbers* α over $\mathbb{Q}$, that is, solutions of polynomial systems with rational coefficients,
- one can compute with such numbers in the same way as with rational numbers,
- in particular, we can decide whether $\alpha = 0$, > 0 or < 0.

Take Home Message II

This is a purely theoretical answer, and no algorithm which naively follows this approach will ever be efficient.

We want *reliable* and efficient geometric computing

- solve problem for all input
- solve problem approximately for all input
- either of these, but coordinates are integers bounded by *M*
- program never crashes and always produces some polygon
- or we guarantee nothing

We want *reliable* and efficient geometric computing

- solve problem for all input
- solve problem approximately for all input
- either of these, but coordinates are integers bounded by M
- ~~program never crashes and always produces some polygon~~
- ~~or we guarantee nothing~~

We want *reliable* and efficient geometric computing

- solve problem for all input
  exact geometric computation paradigm
- solve problem approximately for all input

- either of these, but coordinates are integers bounded by *M*

- ~~program never crashes and always produces some polygon~~
- ~~or we guarantee nothing~~

We want *reliable* and efficient geometric computing

- solve problem for all input
  exact geometric computation paradigm
- solve problem approximately for all input
  allow perturbations of input
- either of these, but coordinates are integers bounded by *M*

- ~~program never crashes and always produces some polygon~~
- ~~or we guarantee nothing~~

We want *reliable* and efficient geometric computing

- solve problem for all input
  exact geometric computation paradigm
- solve problem approximately for all input
  allow perturbations of input
- either of these, but coordinates are integers bounded by *M*
  change algorithm to cope with floats
- ~~program never crashes and always produces some polygon~~
- ~~or we guarantee nothing~~

We want *reliable* and efficient geometric computing

- solve problem for all input
  exact geometric computation paradigm
- solve problem approximately for all input
  allow perturbations of input
- either of these, but coordinates are integers bounded by *M*
  change algorithm to cope with floats
- ~~program never crashes and always produces some polygon~~
- ~~or we guarantee nothing~~

Challenge: Reliable and efficiency!

- theoretically good algorithms
- implementations can compete with unreliable variants

Further Reading

Classroom Examples of Robustness Problems in Geometric Computations, Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap
`www.mpi-inf.mpg.de/departments/d1/ClassroomExamples/`

What Every Computer Scientist Should Know About Floating-Point Arithmetic, David Goldberg
`docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html`

Adaptive Precision Floating-Point Arithmetic and Fast Robust Predicates for Computational Geometry, Jonathan Richard Shewchuk
`http://www.cs.cmu.edu/~quake/robust.html`

Geometric Predicates

Geometric Predicate

sign of a polynomial evaluated in the input's coordinates

- non-continuous
- evaluation with fixed-precision arithmetic leads to errors

Bound on Coordinates - Static Approach

- bound on each input coordinate $|x_i| < X_i$
- analyse expressions of predicates
- choose a precision P such that rounding never occur

Discussion

- usually needs floats with arbitrary precision
- `double` is insufficient
- still “constant” time as P is fixed!
- but slower than machine built-in

We will see semi-static and dynamic approaches

Bound on Coordinates - Static Approach

- bound on each input coordinate $|x_i| < X_i$
- analyse expressions of predicates
- choose a precision P such that rounding never occur

Discussion

- usually needs floats with arbitrary precision
- `double` is insufficient
- still “constant” time as P is fixed!
- but slower than machine built-in

We will see semi-static and dynamic approaches

Controlled Perturbation

Applicable for many cases:

- input comes from imprecise measurements or computations
- perturbation (in limits) allowed
- degeneracies can (will) be eliminated
- geometric objects keep their traits (no approximation)

Idea

randomly perturb input and hope such that degenerate situations for predicates do not occur

Two General Ways

Given input points $p_1, \ldots, p_n$, and algorithm A. Let $P > 0$ be A's working precision and $\delta > 0$ the maximal perturbation:

fixed P, search δ

Find $\delta > 0$ and $\tilde{p}_1, \ldots, \tilde{p}_n$ with $||p_i - \tilde{p}_i|| \leq \delta$ such that A applied on $\tilde{p}_1, \ldots, \tilde{p}_n$ yields correct result for $\tilde{p}_1, \ldots, \tilde{p}_n$ with precision P.

Find close ("δ") instance solvable with runtime guarantee.

fixed δ, search P

Find $P > 0$ and $\tilde{p}_1, \ldots, \tilde{p}_n$ with $||p_i - \tilde{p}_i|| \leq \delta$ such that A applied on $\tilde{p}_1, \ldots, \tilde{p}_n$ yields correct result for $\tilde{p}_1, \ldots, \tilde{p}_n$ with precision P.

Find precision to achieve closeness guarantee.

Guarded predicates

Guarded Geometric Predicate

returns $+1, 0, -1$ only if this answer is *certified*,
otherwise it returns ∇, i.e. the "guard fires"

- guarded predicates should be competitive
- firing guard triggers start over of algorithm
- firing is a "rare" event, $P(\text{"a guard fires"}) < \frac{1}{2}$

Scheme of CP

For input: $p_1, \ldots, p_n$,

fixed P, search δ

Find $\delta > 0$ and $\tilde{p}_1, \ldots, \tilde{p}_n$ with $||p_i - \tilde{p}_i|| \leq \delta$ such that A applied on $\tilde{p}_1, \ldots, \tilde{p}_n$ yields correct result for $\tilde{p}_1, \ldots, \tilde{p}_n$ with precision P.

$\tilde{A}$ is A with guarded predicates.

1. Init $\delta \leftarrow 2^{-P}$
2. Choose $\tilde{p}_1, \ldots, \tilde{p}_n$ at random with precision P and with $||p_i - \tilde{p}_i|| < \delta$
3. Run $\tilde{A}$ on $\tilde{p}_1, \ldots, \tilde{p}_n$
4. If any predicate returns ∇ during execution, $\delta \leftarrow 2\delta$ and goto (2)
5. return δ and $\tilde{p}_1, \ldots, \tilde{p}_n$ and the result of $\tilde{A}$ on that input

Scheme of CP

For input: $p_1, \ldots, p_n$,

fixed δ, search P

Find $P > 0$ and $\tilde{p}_1, \ldots, \tilde{p}_n$ with $||p_i - \tilde{p}_i|| \leq \delta$ such that A applied on $\tilde{p}_1, \ldots, \tilde{p}_n$ yields correct result for $\tilde{p}_1, \ldots, \tilde{p}_n$ with precision P.

$\tilde{A}$ is A with guarded predicates.

1. Init $P \leftarrow 4$
2. Choose $\tilde{p}_1, \ldots, \tilde{p}_n$ at random with precision P and with $||p_i - \tilde{p}_i|| < \delta$
3. Run $\tilde{A}$ on $\tilde{p}_1, \ldots, \tilde{p}_n$
4. If any predicate returns ∇ during execution, $P \leftarrow 2P$ and goto (2)
5. return P and $\tilde{p}_1, \ldots, \tilde{p}_n$ and the result of $\tilde{A}$ on that input

Perturbing?

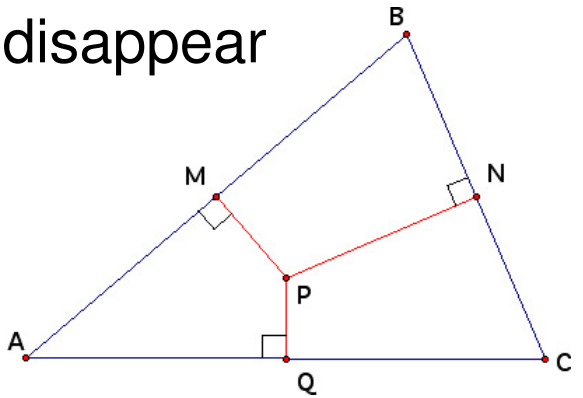
- moving objects
- scaled objects
- stretching objects
- shearing
- rotating

Applicable?

- input can be perturbed (numerical)
- output can be of different structure (combinatorial)

- polygonal chains with vertex coordinates
- polyhedron given by incidence lattice and equation of faces

Comment: Symbolic degeneracies do not disappear



Controlled perturbation successful?

It can be shown that CP terminates, as for large enough δ it holds: $P(\text{“any guard fires”}) < \frac{1}{2}$

Challenge: “What is meant by ’large enough’?”

Must be analyzed for the concrete case.

We first give examples how to guard predicates:

Predicate

Start with a mathematical expression

$$\text{orientation}(p, q, r) := \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))$$

Forward Error Analysis

- compute an explicit bound on the numerical error (caused by fixed precision P)
- consider input $a_1, \ldots, a_n$ and
- polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$
- $E := f(a_1, \ldots, a_n)$
- let $\oplus$, $\ominus$, $\otimes$ respective operations performed with precision P

Recursive definition - Semi-static approach

Define values $\tilde{E}$, $\tilde{E}_{\text{sup}}$ and ind_E recursively:

E	$\tilde{E}$	$\tilde{E}_{\text{sup}}$	ind_E
a_i	a_i	$\lvert a_i \rvert$	0
$A + B$	$A \oplus B$	$\tilde{A}_{\text{sup}} \oplus \tilde{B}_{\text{sup}}$	$1 + \max(\text{ind}_A, \text{ind}_B)$
$A - B$	$A \ominus B$	$\tilde{A}_{\text{sup}} \oplus \tilde{B}_{\text{sup}}$	$1 + \max(\text{ind}_A, \text{ind}_B)$
$A \times B$	$A \otimes B$	$\tilde{A}_{\text{sup}} \otimes \tilde{B}_{\text{sup}}$	$1 + \text{ind}_A + \text{ind}_B$

- $\tilde{E}$ is the result of evaluating $f(a_1, \ldots, a_n)$ with precision P!
- $\tilde{E}_{\text{sup}}$ and ind_E yield an upper bound:

$$|E - \tilde{E}| \leq B := 2^{-P} \times \tilde{E}_{\text{sup}} \times \text{ind}_E$$

Proof idea: low-level numerical analysis arguments and induction on the expression tree (see [BFS2011])

Bound can be extended to division, square- and k-th root (not needed today)

Recursive definition - Semi-static approach

Define values $\tilde{E}$, $\tilde{E}_{\text{sup}}$ and ind_E recursively:

E	$\tilde{E}$	$\tilde{E}_{\text{sup}}$	ind_E
a_i	a_i	$\lvert a_i \rvert$	0
$A + B$	$A \oplus B$	$\tilde{A}_{\text{sup}} \oplus \tilde{B}_{\text{sup}}$	$1 + \max(\text{ind}_A, \text{ind}_B)$
$A - B$	$A \ominus B$	$\tilde{A}_{\text{sup}} \oplus \tilde{B}_{\text{sup}}$	$1 + \max(\text{ind}_A, \text{ind}_B)$
$A \times B$	$A \otimes B$	$\tilde{A}_{\text{sup}} \otimes \tilde{B}_{\text{sup}}$	$1 + \text{ind}_A + \text{ind}_B$

- $\tilde{E}$ is the result of evaluating $f(a_1, \ldots, a_n)$ with precision P!
- $\tilde{E}_{\text{sup}}$ and ind_E yield an upper bound:

$$|E - \tilde{E}| \leq B := 2^{-P} \times \tilde{E}_{\text{sup}} \times \text{ind}_E$$

Proof idea: low-level numerical analysis arguments and induction on the expression tree (see [BFS2011])

Bound can be extended to division, square- and k-th root (not needed today)

Implications

Proposition

If $\tilde{E} > B$, then $E > 0$. If $\tilde{E} < -B$, then $E < 0$.

Let's exercise the orientation predicate (draw tree!):

$$E := (q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)$$

for points $p = (1, 1)$, $q = (2, -1)$ $r = (4, 3)$ and $P = 52$

It holds $\tilde{E} = 8$, $\tilde{E}_{\text{sup}} = 22$ and $\text{ind}_E = 4$.

Hence, $B := 2^{-52} \times 22 \times 4 = 11 \times 2^{-49}$.

As $\tilde{E} > B$, it holds that $E > 0$.

How about precision 4 or 3?

Implications

Proposition

If $\tilde{E} > B$, then $E > 0$. If $\tilde{E} < -B$, then $E < 0$.

Let's exercise the orientation predicate (draw tree!):

$$E := (q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)$$

for points $p = (1, 1)$, $q = (2, -1)$ $r = (4, 3)$ and $P = 52$

It holds $\tilde{E} = 8$, $\tilde{E}_{\text{sup}} = 22$ and $\text{ind}_E = 4$.

Hence, $B := 2^{-52} \times 22 \times 4 = 11 \times 2^{-49}$.

As $\tilde{E} > B$, it holds that $E > 0$.

How about precision 4 or 3?

Implications

Proposition

If $\tilde{E} > B$, then $E > 0$. If $\tilde{E} < -B$, then $E < 0$.

Let's exercise the orientation predicate (draw tree!):

$$E := (q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)$$

for points $p = (1, 1)$, $q = (2, -1)$ $r = (4, 3)$ and $P = 52$

It holds $\tilde{E} = 8$, $\tilde{E}_{\text{sup}} = 22$ and $\text{ind}_E = 4$.

Hence, $B := 2^{-52} \times 22 \times 4 = 11 \times 2^{-49}$.

As $\tilde{E} > B$, it holds that $E > 0$.

How about precision 4 or 3?

Guarding predicates with FEA

- given $f \in \mathbb{Z}[x_1, \ldots, x_n]$ and input $a_1, \ldots, a_n$ and
- goal: evaluate $E := f(a_1, \ldots, a_n)$
- first: evaluate $\tilde{E}$, $\tilde{E}_{\text{sup}}$ and ind_E
- set $B := 2^{-P} \times \tilde{E}_{\text{sup}} \times \text{ind}_E$
- if $\tilde{E} > B$ return $+1$,
  if $\tilde{E} < -B$, return -1,
  otherwise return ∇

i.e. if $-B \leq \tilde{E} \leq B$ the sign of E cannot be certified.

Overhead? - Exercise!

Interval Arithmetic - Dynamic Approach

Let $a \in \mathbb{R}$, for precision P we define

$\overleftarrow{a} := \max\{b | b \leq a, b \text{ is of precision } P\}$

$\overrightarrow{a} := \min\{b | b \geq a, b \text{ is of precision } P\}$

For $[a, b]$ and $[c, d]$ with a, b, c, d of precision P, we define

$[a, b] \boxplus [c, d] := [\overleftarrow{a + c}, \overrightarrow{b + d}]$

$[a, b] \boxminus [c, d] := [\overleftarrow{a - d}, \overrightarrow{b - c}]$

$[a, b] \boxtimes [c, d] := [\overleftarrow{\min\{ac, ad, bc, bd\}}, \overrightarrow{\max\{ac, ad, bc, bd\}}]$

Interval Arithmetic - Properties

Again, we denote with $\tilde{E}$ the evaluation of expression E where each operation $\cdot$ gets replaced by $\boxdot$ and each a_i by $[\overleftarrow{a_i}, \overrightarrow{a_i}]$

Definitions are designed to hold this property:

Proposition

Let $e \in \mathbb{R}$ be the value of E and $[\tilde{e}_{\text{min}}, \tilde{e}_{\text{max}}]$ be the value of $\tilde{E}$. Then, $e \in [\tilde{e}_{\text{min}}, \tilde{e}_{\text{max}}]$.

Proof idea: induction on expression tree

Example of IA, $P = 4$ (3 bits mantissa, plus sign bit)

$$E = (12 - 30) \times (17 + 14) - 51 = -609$$

Note $30 = (1.111)_2 \cdot 2^3$, but $51 = (1.10011)_2 \cdot 2^5$, so

(similar for 12 and 14)

- $\overleftarrow{51} = (1.100)_2 \cdot 2^5 = 32 + 16 = 48$ and
- $\overrightarrow{51} = (1.101)_2 \cdot 2^5 = 32 + 16 + 4 = 52$

Combining:

$$\begin{aligned}
\tilde{E} &= ([12,12] \ominus [30,30]) \otimes ([16,18] \oplus [14,14]) \ominus [48,52] \\
&= ([-18,-18] \otimes [30,32]) \ominus [48,52] \\
&= [\overleftarrow{-576}, \overrightarrow{-540}] \ominus [48,52] \\
&= [-576,-512] \ominus [48,52] \\
&= [\overleftarrow{-628}, \overrightarrow{-560}] \\
&= [-640,-512]
\end{aligned}$$

Example of IA, $P = 4$ (3 bits mantissa, plus sign bit)

$$E = (12 - 30) \times (17 + 14) - 51 = -609$$

Note $30 = (1.111)_2 \cdot 2^3$, but $51 = (1.10011)_2 \cdot 2^5$, so

(similar for 12 and 14)

- $\overleftarrow{51} = (1.100)_2 \cdot 2^5 = 32 + 16 = 48$ and
- $\overrightarrow{51} = (1.101)_2 \cdot 2^5 = 32 + 16 + 4 = 52$

Combining:

$$\begin{aligned}
\tilde{E} &= ([12,12] \ominus [30,30]) \otimes ([16,18] \oplus [14,14]) \ominus [48,52] \\
&= ([-18,-18] \otimes [30,32]) \ominus [48,52] \\
&= [\overleftarrow{-576}, \overrightarrow{-540}] \ominus [48,52] \\
&= [-576,-512] \ominus [48,52] \\
&= [\overleftarrow{-628}, \overrightarrow{-560}] \\
&= [-640,-512]
\end{aligned}$$

Guarding predicates with IA

- given $f \in \mathbb{Z}[x_1, \ldots, x_n]$ and input $a_1, \ldots, a_n$ and
- goal: evaluate $E := f(a_1, \ldots, a_n)$
- first: evaluate $\tilde{E} = [\tilde{e}_{\text{min}}, \tilde{e}_{\text{max}}]$ with IA at precision P
- if $0 \notin [\tilde{e}_{\text{min}}, \tilde{e}_{\text{max}}]$ return $\text{sign}(\tilde{e}_{\text{max}})$ $(= \text{sign}(\tilde{e}_{\text{min}}))$, otherwise return ∇

Overhead? - Exercise!

Disadvantage:

- intervals tend to be too large (esp. for large degrees)
- it often holds $0 \in [\tilde{e}_{\text{min}}, \tilde{e}_{\text{max}}]$
- makes guard less effective
- However, for $P \to \infty$, $w(\tilde{E}) \to 0$

Guarding predicates with IA

- given $f \in \mathbb{Z}[x_1, \ldots, x_n]$ and input $a_1, \ldots, a_n$ and
- goal: evaluate $E := f(a_1, \ldots, a_n)$
- first: evaluate $\tilde{E} = [\tilde{e}_{\min}, \tilde{e}_{\max}]$ with IA at precision P
- if $0 \notin [\tilde{e}_{\min}, \tilde{e}_{\max}]$ return $\text{sign}(\tilde{e}_{\max})$ $(= \text{sign}(\tilde{e}_{\min}))$, otherwise return ∇

Overhead? - Exercise!

Disadvantage:

- intervals tend to be too large (esp. for large degrees)
- it often holds $0 \in [\tilde{e}_{\min}, \tilde{e}_{\max}]$
- makes guard less effective
- However, for $P \to \infty$, $w(\tilde{E}) \to 0$

“Large enough?”

- degeneracy of a d-dimensional predicate: $d + 1$ values lie in the same hyper-plane.
- approximate arithmetic suffices to determine sign, if simplex spanned by perturbed input becomes sufficiently large.

Examples

- Delaunay triangulations [Funke et al.]
- arrangements of circles [Halperin, Leiserowitz]

Mehlhorn, Sagraloff, Osbild

possible for predicates that are signs of polynomials

Region of Uncertainty

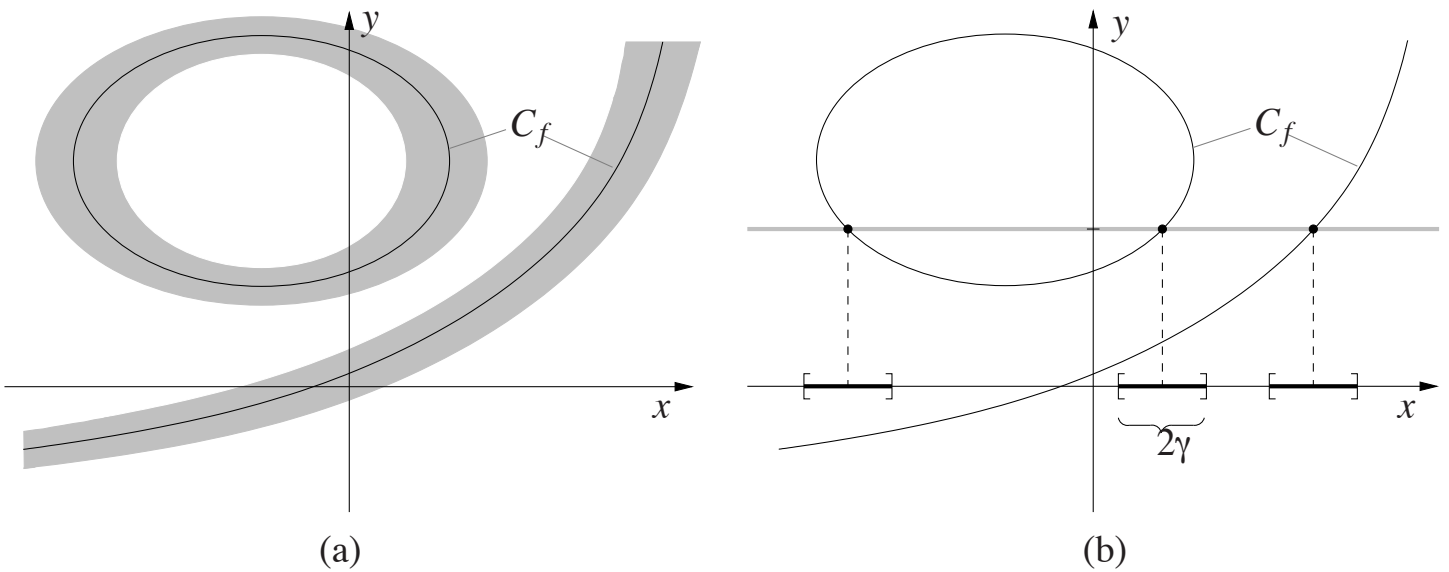


Figure 1: In (a), the critical set of f is indicated as a curve and the region of uncertainty is shown shaded. The region of uncertainty is located around the critical set. The horizontal axis corresponds to the last coordinate and the vertical axis corresponds to all other coordinates. In (b), a fixed value of y is indicated as a grey horizontal line. The x-axis shows the projection of the intersection of the line with the region of uncertainty.

What about constructions?

Geometric algorithms also consists of *constructions*, such as the intersection of object.

Arrangement of circles:

- needs to construct intersections of circles
- exact construction not possible, as this requires much higher precision
- store *inexact intersection point*: at most some distance ω away from exact intersection point
- must be observed in “forbidden area” considerations

Disadvantages

CP nicely sails around degenerate (or near-degenerate) instances

- if exact output is requested, CP is not an option

Thus, we next discuss

- simulation of simplicity
- exact computations

References


Dan Halperin and Christian R. Shelton.
A perturbation scheme for spherical arrangements with application to molecular modeling.
Computational Geometry, 10(4):273 – 287, 1998.


Christop Burnikel, Stefan Funke, and Michael Seel.
Exact geometric computation using cascading.
International Journal of Computational Geometry & Applications, 11(03):245–266, 2001.


Dan Halperin and Eran Leiserowitz.
Controlled perturbation for arrangements of circles.
In *COMPUT. GEOM. THEORY APPL*, pages 264–273. ACM Press, 2003.


Stefan Funke, Christian Klein, Kurt Mehlhorn, and Susanne Schmitt.
Controlled perturbation for delaunay triangulations.
In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pages 1047–1056, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.


Kurt Mehlhorn, Ralf Osbild, and Michael Sagraloff.
A general approach to the analysis of controlled perturbation algorithms.
Comput. Geom. Theory Appl., 44(9):507–528, November 2011.

Dealing with Degeneracies

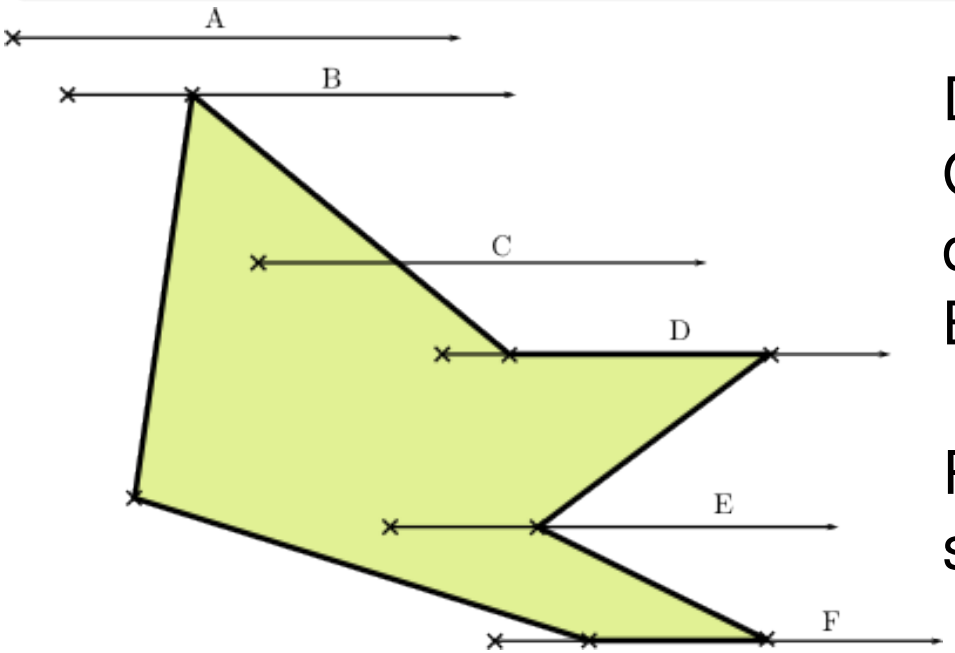
- real-world input data comes with degeneracies
- we want to deal with it
- **we do not want to implement all special cases**

Example: Point-in-polygon test

Given polygon P and point p

- shoot a horizontal ray from p to infinity
- count the intersections m of the ray with P
- if m is odd, p lies inside P, otherwise outside

[Jordan curve theorem]



Difficult to implement:
One has to deal with degenerate cases:
B(0), D(1), E(1), F(0)

Remark: PiP is a simple problem in 2D

Approach

Simulation of Simplicity

- given: program that works for input in general position
- technique to automatically convert the program
- converted works robustly, correctly, and efficient for degenerate cases

[Edelsbrunner, Mücke 1990]

Simulation of Simplicity

Ideas

SoS is a technique that introduces an

- infinitesimal
- virtual
- perturbation
- which does not affect the result
- but eliminates the degenerations.

Simulated conceptual perturbation

- perturbation never computed
- “break ties consistently”
- happens in ... predicates!

Modifying predicates

- replace each coordinate (“parameter”) by a polynomial in $\varepsilon > 0$
- polynomials chosen such that with $\varepsilon \to 0$, perturbed input goes to original set
- exact value of ε not important, can be seen as indeterminant

Example: Orientation predicate (in 2D)

The dD version is a $(d+1) \times (d+1)$-dimensional matrix.

$$M_P := \begin{pmatrix} 1 & v_{1,1} & v_{1,2} \\ 1 & v_{2,1} & v_{2,2} \\ 1 & v_{3,1} & v_{3,2} \end{pmatrix}$$

Example: Orientation predicate (in 2D)

The dD version is a $(d+1) \times (d+1)$-dimensional matrix.

$$M_P(\varepsilon) := \begin{pmatrix} 1 & v_{1,1}(\varepsilon) & v_{1,2}(\varepsilon) \\ 1 & v_{2,1}(\varepsilon) & v_{2,2}(\varepsilon) \\ 1 & v_{3,1}(\varepsilon) & v_{3,2}(\varepsilon) \end{pmatrix}$$

Example: Orientation predicate (in 2D)

The dD version is a $(d+1) \times (d+1)$-dimensional matrix.

$$M_P(\varepsilon) := \begin{pmatrix} 1 & v_{1,1} + \varepsilon(1,1) & v_{1,2} + \varepsilon(1,2) \\ 1 & v_{2,1} + \varepsilon(2,1) & v_{2,2} + \varepsilon(2,2) \\ 1 & v_{3,1} + \varepsilon(3,1) & v_{3,2} + \varepsilon(3,2) \end{pmatrix}$$

Example: Orientation predicate (in 2D)

The dD version is a $(d+1) \times (d+1)$-dimensional matrix.

$$M_P(\varepsilon) := \begin{pmatrix} 1 & v_{1,1} + \varepsilon(1,1) & v_{1,2} + \varepsilon(1,2) \\ 1 & v_{2,1} + \varepsilon(2,1) & v_{2,2} + \varepsilon(2,2) \\ 1 & v_{3,1} + \varepsilon(3,1) & v_{3,2} + \varepsilon(3,2) \end{pmatrix}$$

For $\delta \geq d$, we choose $\varepsilon(i,j) = \varepsilon^{2^{i \cdot \delta - j}}$

Example: Orientation predicate (in 2D)

The dD version is a $(d+1) \times (d+1)$-dimensional matrix.

$$M_P(\varepsilon) := \begin{pmatrix} 1 & v_{1,1} + \varepsilon^{2^{1\cdot\delta-1}} & v_{1,2} + \varepsilon^{2^{1\cdot\delta-2}} \\ 1 & v_{2,1} + \varepsilon^{2^{2\cdot\delta-1}} & v_{2,2} + \varepsilon^{2^{2\cdot\delta-2}} \\ 1 & v_{3,1} + \varepsilon^{2^{3\cdot\delta-1}} & v_{3,2} + \varepsilon^{2^{3\cdot\delta-2}} \end{pmatrix}$$

For $\delta \geq d$, we choose $\varepsilon(i,j) = \varepsilon^{2^{i\cdot\delta-j}}$

Example: Orientation predicate (in 2D)

The dD version is a $(d+1) \times (d+1)$-dimensional matrix.

$$M_P(\varepsilon) := \begin{pmatrix} 1 & v_{1,1} + \varepsilon^{2^{1\cdot\delta-1}} & v_{1,2} + \varepsilon^{2^{1\cdot\delta-2}} \\ 1 & v_{2,1} + \varepsilon^{2^{2\cdot\delta-1}} & v_{2,2} + \varepsilon^{2^{2\cdot\delta-2}} \\ 1 & v_{3,1} + \varepsilon^{2^{3\cdot\delta-1}} & v_{3,2} + \varepsilon^{2^{3\cdot\delta-2}} \end{pmatrix}$$

Lemma

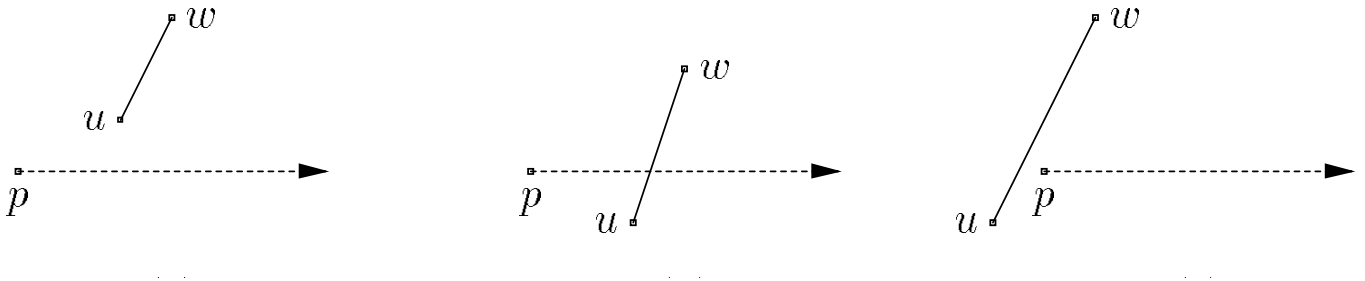
$M_P(\varepsilon)$ is nondegenerate if $\varepsilon > 0$ is sufficiently small.

Proof: Show that for no choice of $d+1$ mutually distinct indices the determinant of $M_P(\varepsilon)$ is zero.

- sort the terms of $\det M(\varepsilon)$ in order of increasing exponents of ε.
- $\det M$ is the first, and $(-1)^{\lceil 2/d \rceil} \cdot \varepsilon^{2^{i_1\cdot\delta-d}+\ldots+2^{i_d\cdot\delta-1}}$ is the last
- each term is of form $b\varepsilon^c$; as ε is arbitrarily small, absolute value of first non-vanishing term is smaller than sum of all remaining terms
- such a term always exists, as
  $\prod_{(i,j)\prec(k,l)} \varepsilon(i,j) = \prod_{(i,j)\prec(k,l)} \varepsilon^{2^{i\cdot\delta-j}} > \varepsilon^{2^{k\cdot\delta-l}} = e(k,l)$
- such a term cannot cancel, in particular not the last one!

Point in Polygon

Algorithm needs orientation test



- left: $p_y \notin [u_y, w_y] \Rightarrow$ no intersection
- middle: (u, w, p) is a left-turn $\Rightarrow$ intersection
- right: (u, w, p) is a right-turn $\Rightarrow$ no intersection

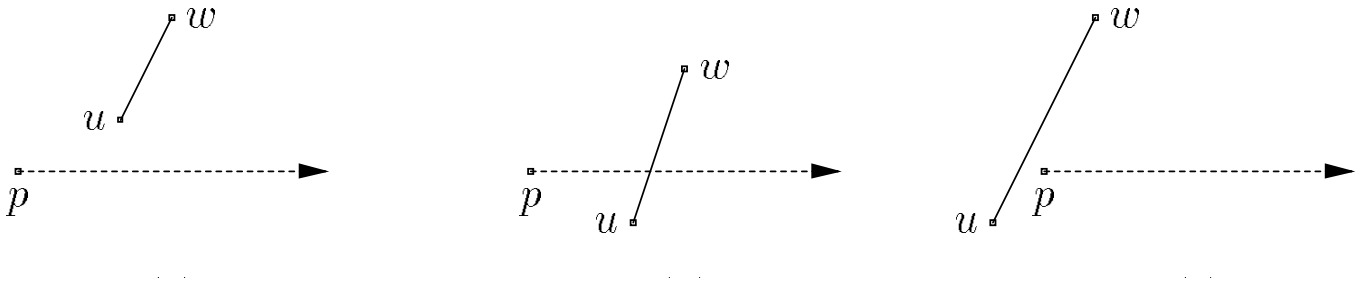
$$\begin{pmatrix} 1 & v_{1,1}(\varepsilon) & v_{1,2}(\varepsilon) \\ 1 & v_{2,1}(\varepsilon) & v_{2,2}(\varepsilon) \\ 1 & v_{3,1}(\varepsilon) & v_{3,2}(\varepsilon) \end{pmatrix}$$

For PiP it is sufficient to perturb only $p =: v_1$.

Missing: $p \in \partial P$

Point in Polygon

Algorithm needs orientation test



- left: $p_y \notin [u_y, w_y] \Rightarrow$ no intersection
- middle: (u, w, p) is a left-turn $\Rightarrow$ intersection
- right: (u, w, p) is a right-turn $\Rightarrow$ no intersection

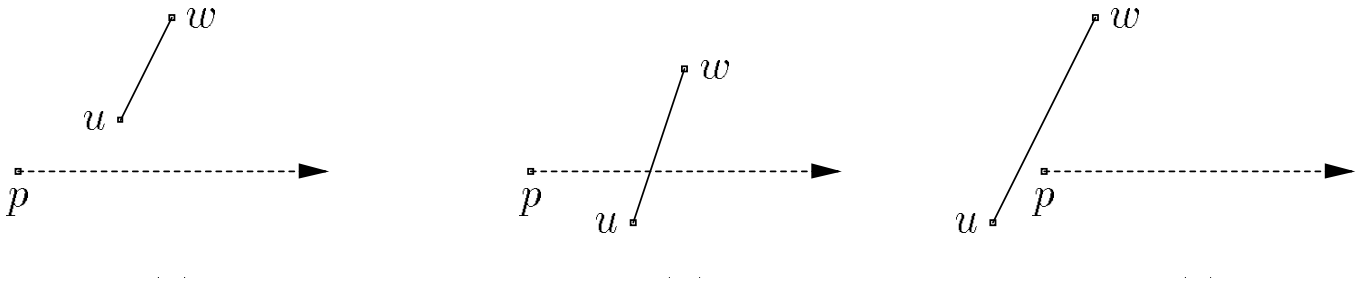
$$\begin{pmatrix} 1 & v_{1,1}(\varepsilon) & v_{1,2}(\varepsilon) \\ 1 & v_{2,1} & v_{2,2} \\ 1 & v_{3,1} & v_{3,2} \end{pmatrix}$$

For PiP it is sufficient to perturb only $p =: v_1$.

Missing: $p \in \partial P$

Point in Polygon

Algorithm needs orientation test



- left: $p_y \notin [u_y, w_y] \Rightarrow$ no intersection
- middle: (u, w, p) is a left-turn $\Rightarrow$ intersection
- right: (u, w, p) is a right-turn $\Rightarrow$ no intersection

$$\begin{pmatrix} 1 & v_{1,1}(\varepsilon) & v_{1,2}(\varepsilon) \\ 1 & v_{2,1} & v_{2,2} \\ 1 & v_{3,1} & v_{3,2} \end{pmatrix}$$

For PiP it is sufficient to perturb only $p =: v_1$.

Missing: $p \in \partial P$

Changes

Instead of evaluating one determinant, SoS . . .

- evaluates subdeterminants
- order is based on “perturbation” (read $\varepsilon(i, j)$)
- tree-like way
- sign is determined by the first non-vanishing term
- subdeterminant in leaves are ensured to be non-zero!

Any predicate

SoS is for cases of topological primitives f

- input consists of a finite number n of parameters
- there is only a finite number of outputs (no new objects)
- can be iterated: $+1, -1$ are *simple* situations, 0 is degenerate
- $f^{-1}(0)$ is the degenerate input, must be zero-dimensional
- embedded in n-dimensional space
- every ball around a degenerate point contains a “safe” spot in a neighboring cell

SoS Discussion

- conceptually simple
- algorithms do not have to get altered
- but “case distinction” takes place in choosing $\varepsilon(i, j)$
- $\Rightarrow$ needs a problem-specific perturbation scheme

References

Herbert Edelsbrunner, Ernst Peter Mücke.
Simulation of Simplicity: A Technique to Cope with Deegerate Cases in Geometric Algorithms.
ACM Transactions on Graphics, 9(1):66–104, 1990.

The Exact Geometric Computation Paradigm

Why being unreliable? Why perturbing input? Why avoiding degeneracies?

How to achieve all goals at the same time?

While staying efficient!

The Exact Geometric Computation Paradigm

Why being unreliable? Why perturbing input? Why avoiding degeneracies?

How to achieve all goals at the same time?

While staying efficient!

The Exact Geometric Computation Paradigm

Why being unreliable? Why perturbing input? Why avoiding degeneracies?

How to achieve all goals at the same time?

While staying efficient!

Correctness

Remember: A geometric algorithm's flow depends on

- output of predicates
- (objects constructed)

Robustness problem is a deviation of theoretical and implemented flow!

Exact Geometric Paradigm

Every geometric primitive returns the mathematically correct result (including degeneracies)!

If followed, there cannot be any deviation!

Weak version

Weak Exact Geometric Paradigm

Every geometric *predicate* returns the mathematically correct result!

Not robust in cases where algorithm exploits implicit assumptions on constructed objects, which are not met by approximate versions.

What can we expect?

Advantages:

- correctness carries over from theory in paper to implementation
- no further analysis (to decrease the degree of predicates)
- no perturbation needed
- handles all degeneracies (if covered in paper!)

Disadvantage: How to achieve it without (too big) penalty?

What can we expect?

Advantages:

- correctness carries over from theory in paper to implementation
- no further analysis (to decrease the degree of predicates)
- no perturbation needed
- handles all degeneracies (if covered in paper!)

Disadvantage: How to achieve it without (too big) penalty?

How to achieve?

requires exact number types

- depending on problem
- bigfloats, integers, rational, one-root-numbers, algebraic numbers (roots of polynomials)
- sign of number/expression?
- comparison of two numbers/expression?
- remember: comparison of two number is expression (substraction)
- **reliable zero test**
- arithmetic? exact or approximate?

Exact number types

Integer $n \in \mathbb{Z}$

- how to store?
- “integral” to all other number types

Bigfloats $x = sm2^e$

- arbitrary mantissa length
- suffices for weak EGC paradigm

Rational $x = \frac{n}{d}$ with $n, d \in \mathbb{Z}$ (similar for one-root numbers)

- two integers
- canonicalize?
- if all constructions stay rational, suffices for EGC paradigm
- we will see an example

Exact number types

Integer $n \in \mathbb{Z}$

- how to store?
- “integral” to all other number types

Bigfloats $x = sm2^e$

- arbitrary mantissa length
- suffices for weak EGC paradigm

Rational $x = \frac{n}{d}$ with $n, d \in \mathbb{Z}$ (similar for one-root numbers)

- two integers
- canonicalize?
- if all constructions stay rational, suffices for EGC paradigm
- we will see an example

Exact number types

Integer $n \in \mathbb{Z}$

- how to store?
- “integral” to all other number types

Bigfloats $x = sm2^e$

- arbitrary mantissa length
- suffices for weak EGC paradigm

Rational $x = \frac{n}{d}$ with $n, d \in \mathbb{Z}$ (similar for one-root numbers)

- two integers
- canonicalize?
- if all constructions stay rational, suffices for EGC paradigm
- we will see an example

Examples

- GMP
- MPFR
- LEDA
- CORE
- NTL

Price?

- severe slow downs
- non-constant computations
- larger memory usage (usually large arrays of machine ints)

End of the story?

Read the EGC paradigm carefully

Every geometric primitive returns the mathematically correct result (including degeneracies)!

Price?

- severe slow downs
- non-constant computations
- larger memory usage (usually large arrays of machine ints)

End of the story?

Read the EGC paradigm carefully

Every geometric primitive returns the mathematically correct result (including degeneracies)!

Price?

- severe slow downs
- non-constant computations
- larger memory usage (usually large arrays of machine ints)

End of the story?

Read the EGC paradigm carefully

Every geometric primitive **returns** the mathematically correct result (including degeneracies)!

Price?

- severe slow downs
- non-constant computations
- larger memory usage (usually large arrays of machine ints)

End of the story?

Read the EGC paradigm carefully

Every geometric primitive **returns** the mathematically correct result (including degeneracies)!

Implications:

- not all computations must be performed exactly
- any technique that produces the correct output is allowed

Filtered Predicates

Filter

The implementation of a predicate is a filter if

- it returns either the correct answer or ∇ (“filter failure”)
- it is faster than the exact predicate

Comparison to Controlled Perturbation:

- guarded predicates (with double precision) are filters
- instead of aborting on ∇, one switches to an exact predicate

Challenge

- design filters that are fast
- and fail only occasionally
- (avoid to call filter if zero-result is expected)

Filtered Predicates

Filter

The implementation of a predicate is a filter if

- it returns either the correct answer or ∇ ("filter failure")
- it is faster than the exact predicate

Comparison to Controlled Perturbation:

- guarded predicates (with double precision) are filters
- instead of aborting on ∇, one switches to an exact predicate

Challenge

- design filters that are fast
- and fail only occasionally
- (avoid to call filter if zero-result is expected)

Filtered Predicates

Filter

The implementation of a predicate is a filter if

- it returns either the correct answer or ∇ ("filter failure")
- it is faster than the exact predicate

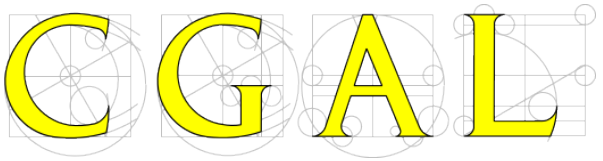
Comparison to Controlled Perturbation:

- guarded predicates (with double precision) are filters
- instead of aborting on ∇, one switches to an exact predicate

Challenge

- design filters that are fast
- and fail only occasionally
- (avoid to call filter if zero-result is expected)

CGAL follows the EGC



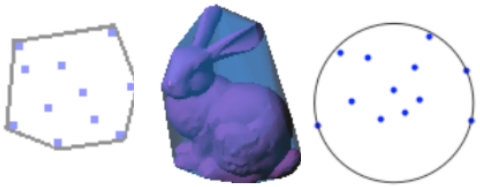
- focus on geometry
- interfaces with de facto standards: STL, Boost, GMP
- ease of integration and customization
- main stream language C++
- huge collection with uniform APIs
- modular and not monolithic
- ε-free

seperates

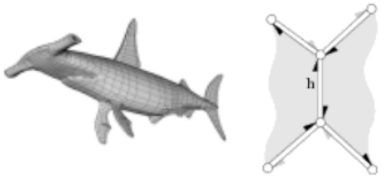
- *combinatorial layer* of algorithms/data structures from
- *numerical layer* (the geometric primitives) in a *kernel*

through generic programming

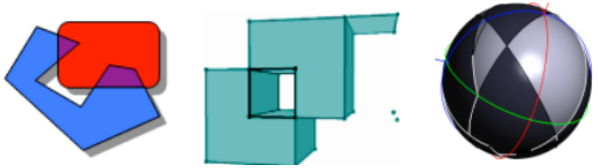
Algorithms and Data structures



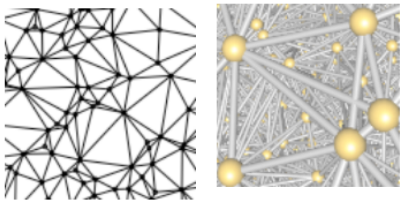
Bounding Volumes



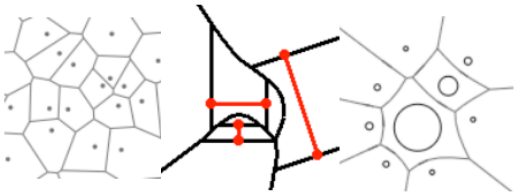
Polyhedral Surface



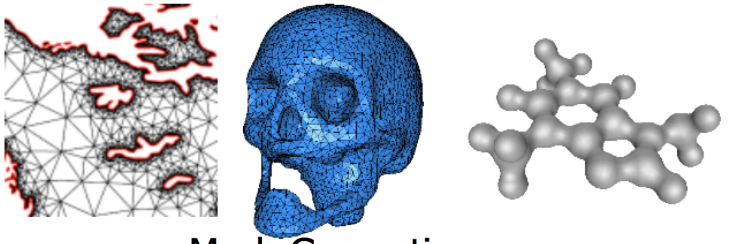
Boolean Operations



Triangulations

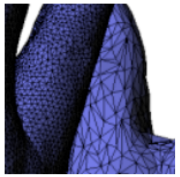


Voronoi Diagrams

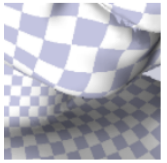


Mesh Generation

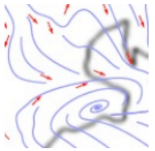
Subdivision



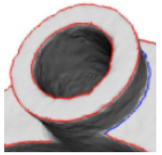
Simplification



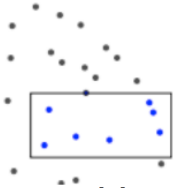
Parameterization



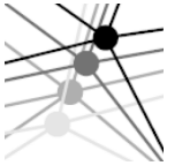
Streamlines



Ridge Detection

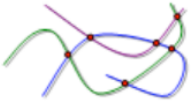


Neighbor Search



Kinetic Data Structures

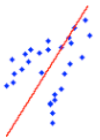
Lower Envelope



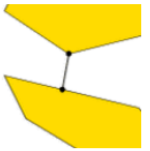
Arrangement

Intersection Detection

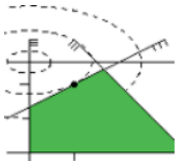
Minkowski Sum



PCA



Polytope Distance

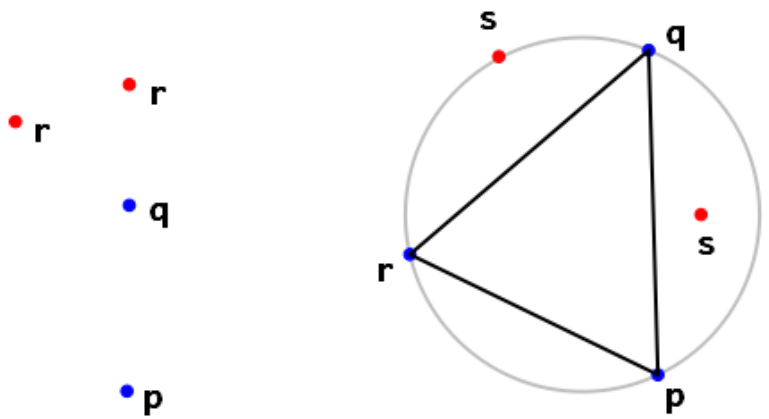


QP Solver

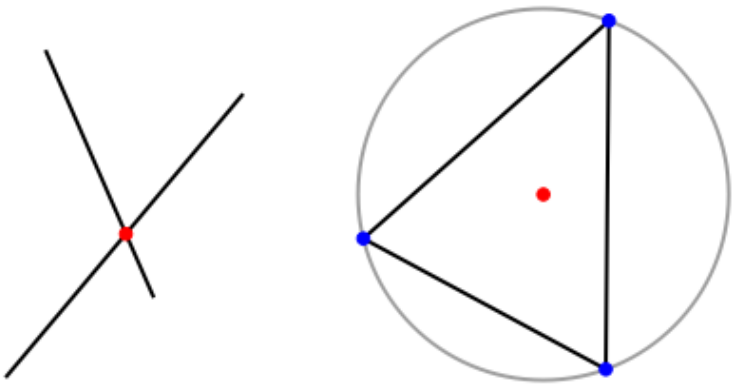
CGAL's kernels

Kernel collects

- *objects* in 2D, 3D and dD:
  points, segments, ray, line, triangle, spheres
- *predicates* (like orientations or in_circle



- *constructions* (like intersection or circumcenter)



Instantiating kernels

- `CGAL::Cartesian< double >` (fast but unreliable)
- `CGAL::Cartesian< gmpq >` (reliable but slow)
- `CGAL::Filtered_kernel< CGAL::Cartesian< gmpq > >`
  (fast and reliable)

- `CGAL::Exact_predicate_exact_constructions` (EGC)
- `CGAL::Exact_predicate_inexact_constructions` (weak EGC)

Instantiating algorithms

```
1  // missing includes!
2  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
3  typedef CGAL::Triangulation_2<K>          Triangulation;
4  typedef Triangulation::Vertex_circulator  Vertex_circulator;
5  typedef Triangulation::Point              Point;
6  int main() {
7    std::ifstream in("data.cin");
8    std::istream_iterator<Point> begin(in);
9    std::istream_iterator<Point> end;
10   Triangulation t;
11   t.insert(begin, end);
12   Vertex_circulator vc =
13     t.incident_vertices(t.infinite_vertex()), done(vc);
14   if (vc != 0) {
15     do { std::cout << vc->point() << std::endl;
16     } while(++vc != done);
17   }
18   return 0;
19 }
```

Instantiating algorithms

```
1  // missing includes!
2  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
3  typedef CGAL::Triangulation_2<K>          Triangulation;
4  typedef Triangulation::Vertex_circulator  Vertex_circulator;
5  typedef Triangulation::Point              Point;
6  int main() {
7    std::ifstream in("data.cin");
8    std::istream_iterator<Point> begin(in);
9    std::istream_iterator<Point> end;
10   Triangulation t;
11   t.insert(begin, end);
12   Vertex_circulator vc =
13     t.incident_vertices(t.infinite_vertex()), done(vc);
14   if (vc != 0) {
15     do { std::cout << vc->point() << std::endl;
16     } while(++vc != done);
17   }
18   return 0;
19 }
```

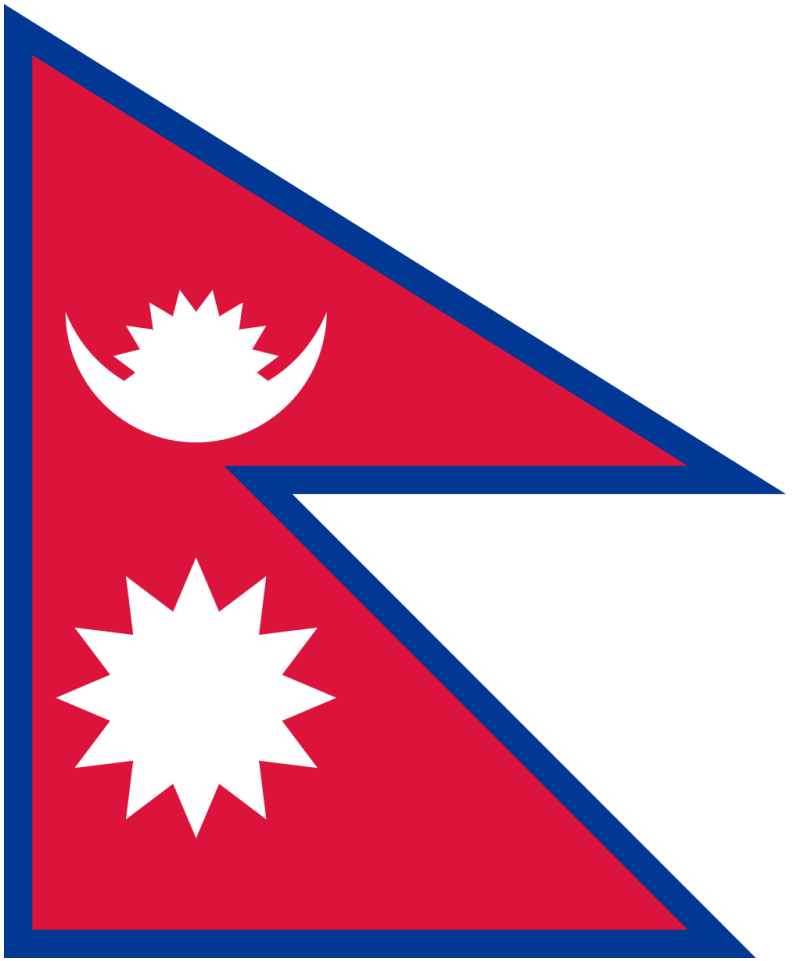
Changing line 2 to `typedef CGAL::Cartesian< double > K;` yields a faster, but unreliable implementation!

Non-linear geometry

- things are getting complicated, when points and segments are left behind ...
- circles, conics, cubics, algebraic curves $p(x, y) = 0$
- their intersections have non-rational coordinates
- more precisely: need roots of polynomials of multiplied degrees

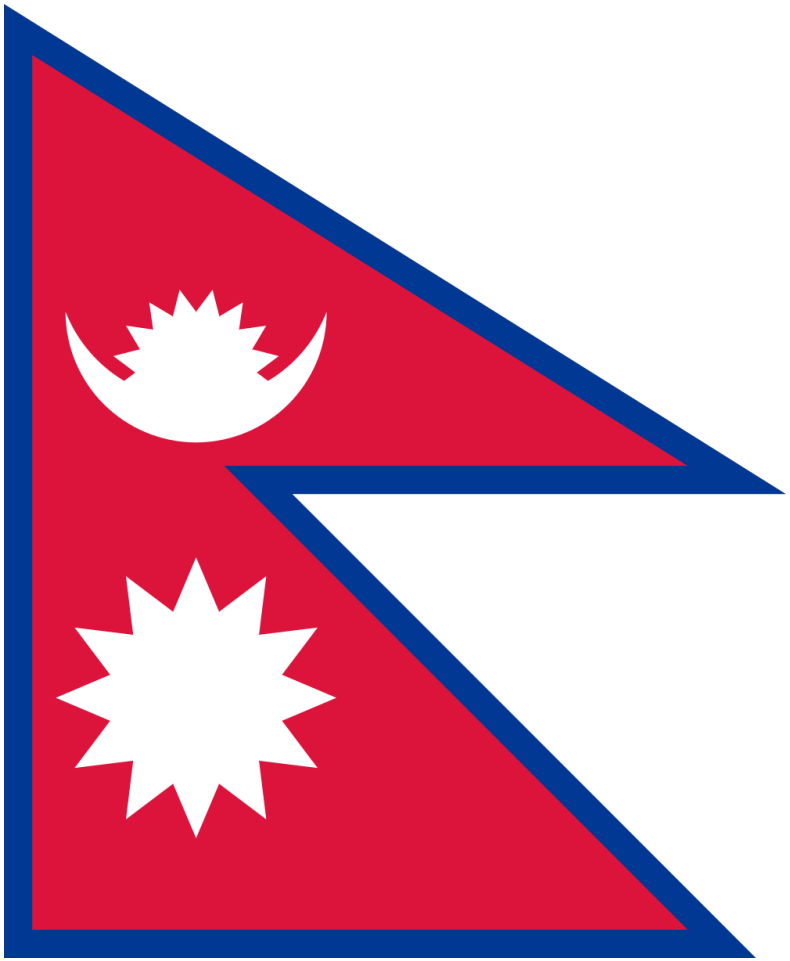
CGAL also has non-linear "*algebraic kernels*"
(in 1D, 2D [and 3D])

The Flag of Nepal



http://0xc.de/flags/nepal/

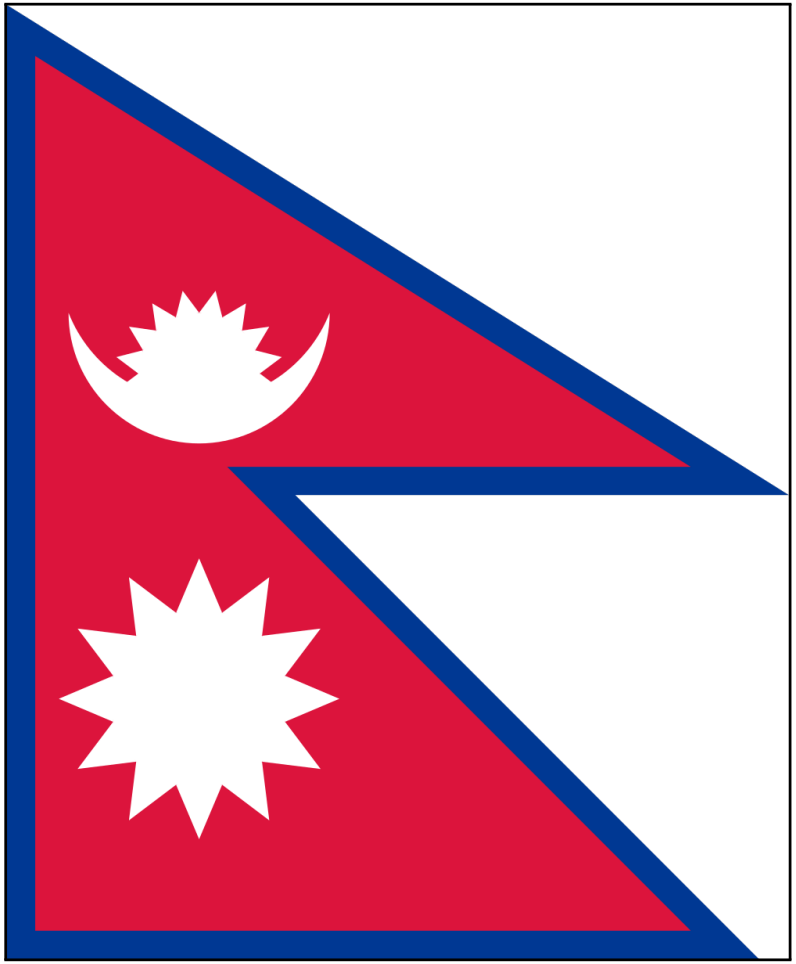
The Flag of Nepal



(A) Method of Making the Shape inside the Border (1) On the lower portion of a crimson cloth draw a line AB of the required length from left to right. (2) From A draw a line AC perpendicular to AB making AC equal to AB plus one third AB. From AC mark off D making line AD equal to line AB. Join BD. (3) From BD mark off E making BE equal to AB. (4) Touching E draw a line FG, starting from the point F on line AC, parallel to AB to the right hand-side. Mark off FG equal to AB. (5) Join CG.
(B) Method of Making the Moon (6) From AB mark off AH making AH equal to one-fourth of line AB and starting from H draw a line HI parallel to line AC touching line CG at point I. (7) Bisect CF at J and draw a line JK parallel to AB touching CG at point K. (8) Let L be the point where lines JK and HI cut one another. (9) Join JG. (10) Let M be the point where line JG and HI cut one another. (11) With centre M and with a distance shortest from M to BD mark off N on the lower portion of line HI. (12) Touching M and starting from O, a point on AC, draw a line from left to right parallel to AB. (13) With centre L and radius LN draw a semi-circle on the lower portion and let P and Q be the points where it touches the line OM respectively. (14) With centre M and radius MQ draw a semi-circle on the lower portion touching P and Q. (15) With centre N and radius NM draw an arc touching PNQ [sic] at R and S. Join RS. Let T be the point where RS and HI cut one another. (16) With Centre T and radius TS draw a semi-circle on the upper portion of PNQ touching it at two points. (17) With centre T and radius TM draw an arc on the upper portion of PNQ touching at two points. (18) Eight equal and similar triangles of the moon

http://0xc.de/flags/nepal/

The Flag of Nepal



$1 : \alpha \approx 1.21901\ldots$

α is an *algebraic number*

`http://0xc.de/flags/nepal/`

(A) Method of Making the Shape inside the Border (1) On the lower portion of a crimson cloth draw a line AB of the required length from left to right. (2) From A draw a line AC perpendicular to AB making AC equal to AB plus one third AB. From AC mark off D making line AD equal to line AB. Join BD. (3) From BD mark off E making BE equal to AB. (4) Touching E draw a line FG, starting from the point F on line AC, parallel to AB to the right hand-side. Mark off FG equal to AB. (5) Join CG.
(B) Method of Making the Moon (6) From AB mark off AH making AH equal to one-fourth of line AB and starting from H draw a line HI parallel to line AC touching line CG at point I. (7) Bisect CF at J and draw a line JK parallel to AB touching CG at point K. (8) Let L be the point where lines JK and HI cut one another. (9) Join JG. (10) Let M be the point where line JG and HI cut one another. (11) With centre M and with a distance shortest from M to BD mark off N on the lower portion of line HI. (12) Touching M and starting from O, a point on AC, draw a line from left to right parallel to AB. (13) With centre L and radius LN draw a semi-circle on the lower portion and let P and Q be the points where it touches the line OM respectively. (14) With centre M and radius MQ draw a semi-circle on the lower portion touching P and Q. (15) With centre N and radius NM draw an arc touching PNQ [sic] at R and S. Join RS. Let T be the point where RS and HI cut one another. (16) With Centre T and radius TS draw a semi-circle on the upper portion of PNQ touching it at two points. (17) With centre T and radius TM draw an arc on the upper portion of PNQ touching at two points. (18) Eight equal and similar triangles of the moon

Exact number types - continued

Algebraic numbers/expressions:

- expression dag/tree storing approximation(s) per node
- *zero bound B*, i.e. if $|\tilde{E}| < B$, then $E = 0$
  or
- square-free polynomial + isolating interval $[\ell, r]$
- interval can be refined with bisection
  (or quicker using “Newton”-like scheme)
- sign needs evaluation of polynomial
- deciding equality needs gcd of two polynomials

Exercise: How to compare two such numbers?

Exact number types - continued

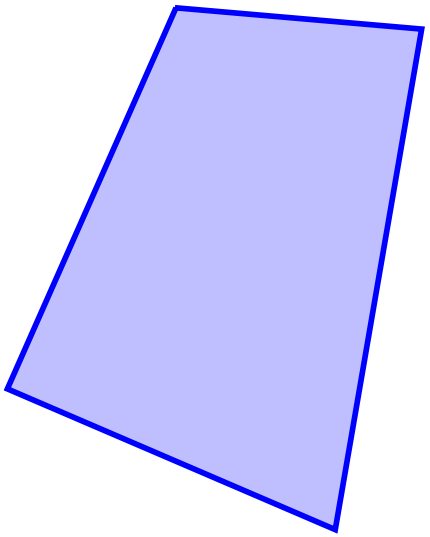
Algebraic numbers/expressions:

- expression dag/tree storing approximation(s) per node
- *zero bound B*, i.e. if $|\tilde{E}| < B$, then $E = 0$
  or
- square-free polynomial + isolating interval $[\ell, r]$
- interval can be refined with bisection
  (or quicker using “Newton”-like scheme)
- sign needs evaluation of polynomial
- deciding equality needs gcd of two polynomials

Exercise: How to compare two such numbers?

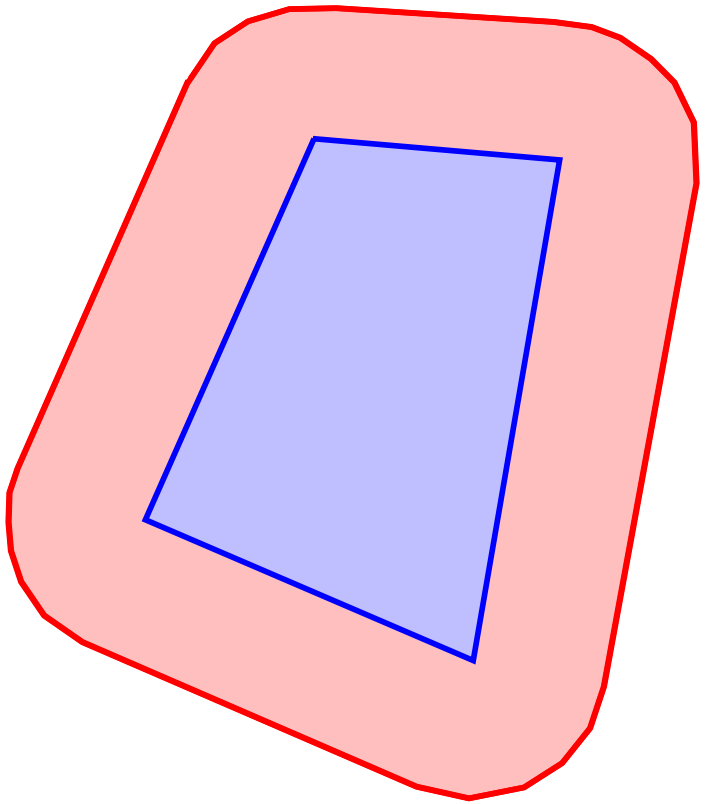
Example: Offsets

We start with a polygon



Example: Offsets

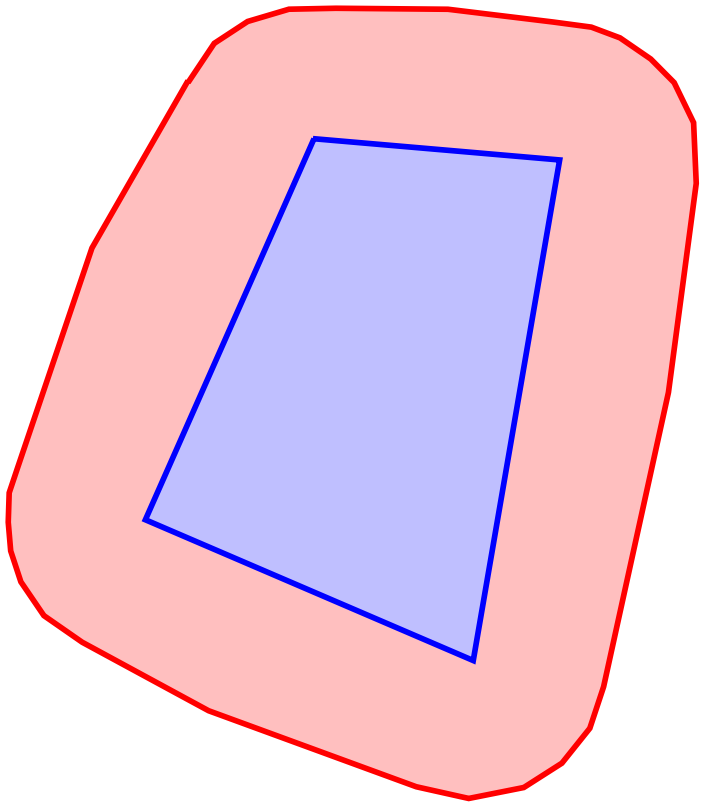
We start with a polygon



- Constructing Offsets:
  - Exact

Example: Offsets

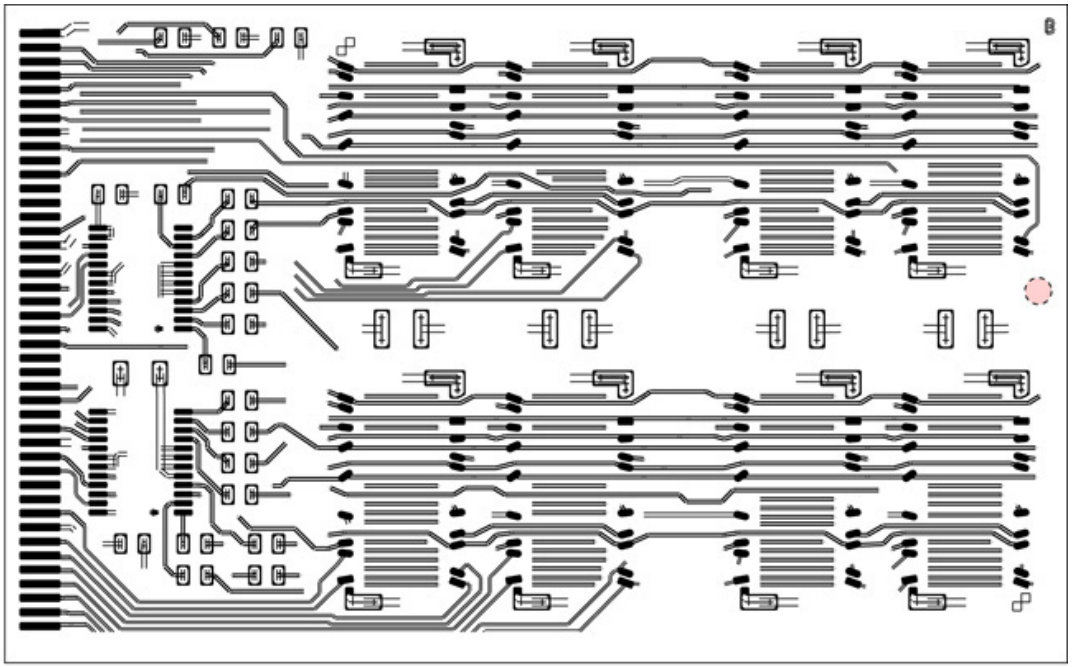
We start with a polygon



- Constructing Offsets:
  - Exact
  - Approximate

Why? Applications

- motion planning
- manufacturing
- computer aided design



Challenge

- algorithm easy and straightforward (we'll sketch two)
  - requires arrangement of curves of degree 2
  - with degeneracies
- getting it implemented is non-trivial
  - numerical and combinatorial issues
  - large vertical body of tools needed

Solution 1: EGC
Solution 2: approximate with rational arithmetic

Definitions: Polygon and Offset

Polygonal region ...

... is a shape with piece-wise linear boundary connected at *vertices*

Minkowski sum of X and Y

$$X \oplus Y := \{x + y \mid x \in X, y \in Y\}$$

Disk

For $r \in \mathbb{R}$, we write $D_r := \{p \in \mathbb{R}^2 \mid d(0, p) \leq r\}$ for the (closed) r-disk centered at the origin.

r-offset of X

$$\text{offset}(X, r) := X \oplus D_r$$

Implementations

Minkowski sum/Offset

- LEDA [Mehlhorn, Näher]
- CGAL (Exact + Approx.) [Wein 2007]

Construction by Convolution

Three steps:

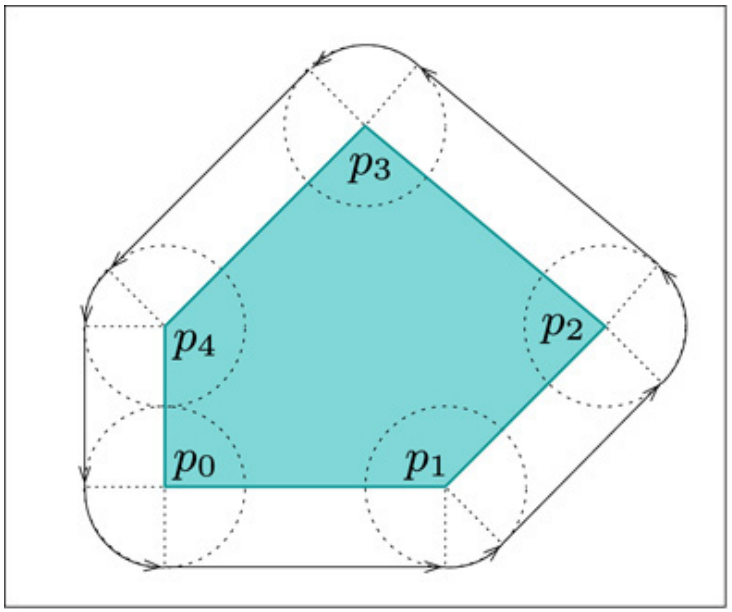
1. compute *convolution* of polygon and disc
   - consists of line segments and circular arcs
2. construct the *arrangement* of convolution
   - planar subdivision which induces 0-, 1-, and 2-dimensional cells (vertices, edges, and faces)
3. *deduce* shape of offset from this arrangement
   - pick the right faces

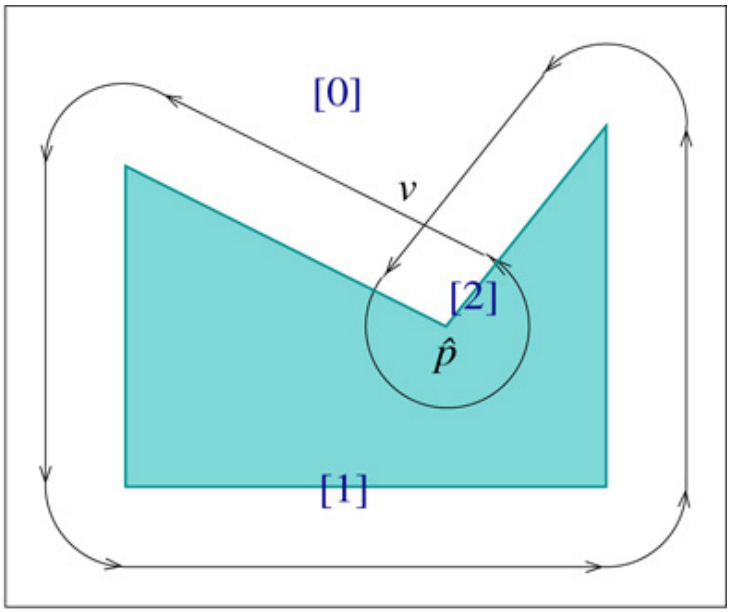
Remark: Steps (2) and (3) are trivial for convex polygons

Step 1: Convolution

Create sequence of curves:

- iterate through oriented edges of P in clockwise order
  - create line segment parallel to current e (on its right side)
  - extend by circular arc centered at target of e





- has complexity $O(n)$

Involved curves

Circlular arc around vertex p_i

- centered at vertex of polygon - has rational coordinates
- radius r is also rational
- Angle: $\pi - \angle(p_{i-1}, p_i, p_{i+1})$ (non-reflex) and $3\pi - \angle(p_{i-1}, p_i, p_{i+1})$ (reflex)
- subset of a circle (a degenerated conic)

Line segment $\overrightarrow{v_1 v_2}$ parallel to $\overrightarrow{p_1 p_2}$, $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$

- let ϕ denote the angle $\overrightarrow{p_1 p_2}$ forms with x-axis
- $\ell = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- $v_j = (x_j + \frac{r}{\ell}(y_2 - y_1), y_j + \frac{r}{\ell}(x_1 - x_2))$ for $j = 1, 2$
- if $\overrightarrow{p_1 p_2}$ is supported by $ax + by + c = 0$ $(a, b, c \in \mathbb{Q})$, then $\overrightarrow{v_1 v_2}$ is supported $ax + by + (c + \ell r) = 0$, usually $\ell \notin \mathbb{Q}$

Curves and Intersections

Supporting lines $ax + by + (c + \ell r) = 0$

- subset of all points lying at distance r from $ax + by + c = 0$
- all these points are solutions to

$$\frac{(ax + by + c)^2}{a^2 + b^2} = r^2$$

degenerate conic of two parallel lines

All curves are conics (curves of degree 2)

Vertices

- intersections of two segments or a segment and a circle have coordinates that are roots of integral degree 4 polynomials
  $\Rightarrow$ algebraic numbers of degree 4

Curves and Intersections

Supporting lines $ax + by + (c + \ell r) = 0$

- subset of all points lying at distance r from $ax + by + c = 0$
- all these points are solutions to

$$\frac{(ax + by + c)^2}{a^2 + b^2} = r^2$$

degenerate conic of two parallel lines

All curves are conics (curves of degree 2)

Vertices

- intersections of two segments or a segment and a circle have coordinates that are roots of integral degree 4 polynomials
  $\Rightarrow$ algebraic numbers of degree 4

Number types

Number types capable to represent such coordinates

- `LEDA::Real`
- `CORE::Expr`

use seperations bounds to determine whether an expression stored is 0

or

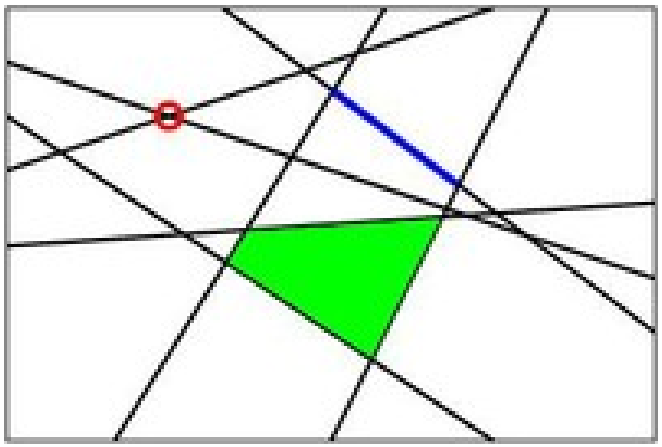
- $x = (q, [\ell, r])$ with $q \in \mathbb{Z}[x]$ and $\ell, r \in \mathbb{Q}$ (or $\in \mathbb{B}$)

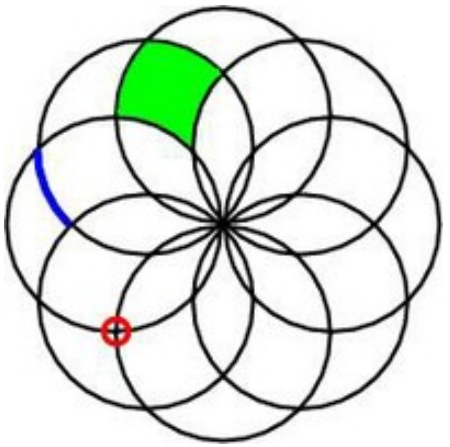
requires $\gcd(q_1, q_2)$ to determine equality of two such numbers

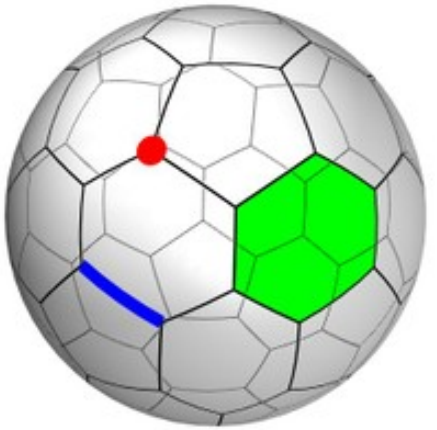
Step 2: Arrangement

Arrangement

Subdivision of plane into cells of dimension 0 (vertices), dimension 1 (edges) and dimension 2 (faces) incuded by a set of curves







Arrangements in CGAL

- constructs, overlays, maintains, modifies, traverses, queries arrangements of curves in the plane (and on surfaces)
- modular due to generic programming
- efficient and robust, if used with exact geometric operations
- implements generic sweep line/zone algorithm, that
  - handle all degeneracies: e.g., vertical curves, multiple curves running through a common point, etc.
  - use *visitor* pattern to decouple *combinatorics* of sweep/zoning from *output*, e.g. reporting intersections or constructing representation of arrangement
- `Arrangement_2<GeoTraits,...>`
  - defines the family of curves that induce the arrangement
  - aggregates
    - geometric types (point, curve)
    - geometric constructions and predicates to control execution path of algorithms

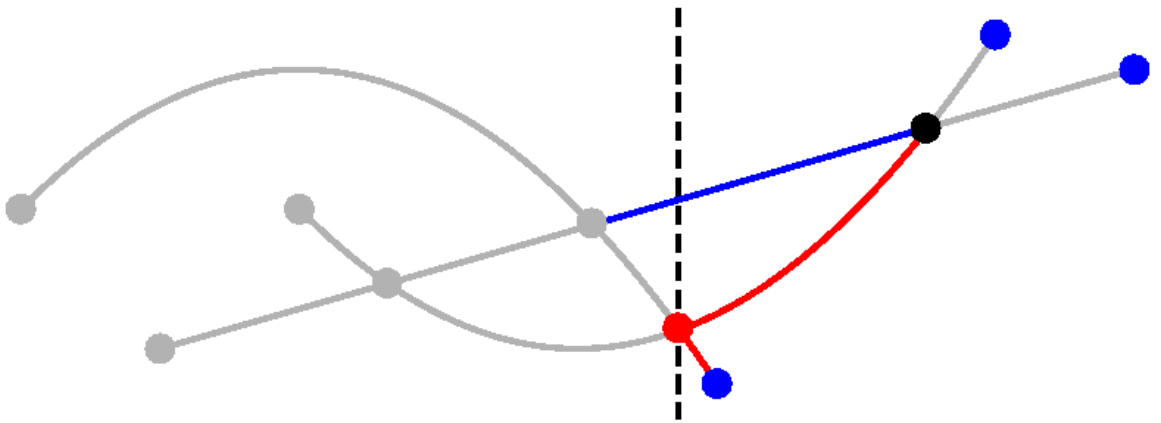
Arrangements in CGAL

- constructs, overlays, maintains, modifies, traverses, queries arrangements of curves in the plane (and on surfaces)
- modular due to generic programming
- efficient and robust, if used with exact geometric operations
- implements generic sweep line/zone algorithm, that
  - handle all degeneracies: e.g., vertical curves, multiple curves running through a common point, etc.
  - use *visitor* pattern to decouple *combinatorics* of sweep/zoning from *output*, e.g. reporting intersections or constructing representation of arrangement
- `Arrangement_2<GeoTraits,...>`
  - defines the family of curves that induce the arrangement
  - aggregates
    - geometric types (point, curve)
    - geometric constructions and predicates to control execution path of algorithms

Sweep line

Algorithm by Bentley-Ottmann sweep (1979)

- preprocessing:
  - obtain *x*-monotone curves
  - create initial xy-ordered *event queue* for curve endpoints
- as long as event queue is not empty
  - update *status structure* wrt to incident curves of event
  - check adjacent curves in status structure for future intersections and insert them into event queue

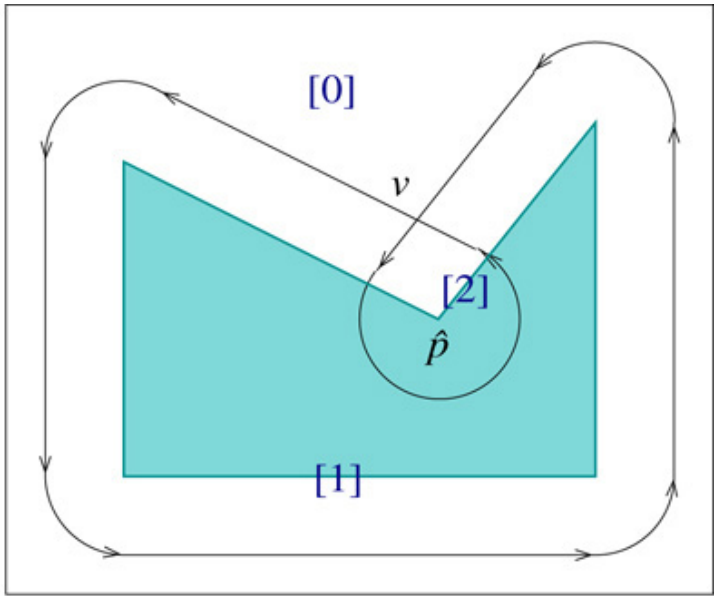


Step 3: Deduce offset

Compute for each face f of arrangement:

winding number $W(f)$ $W(f) \geq 0$

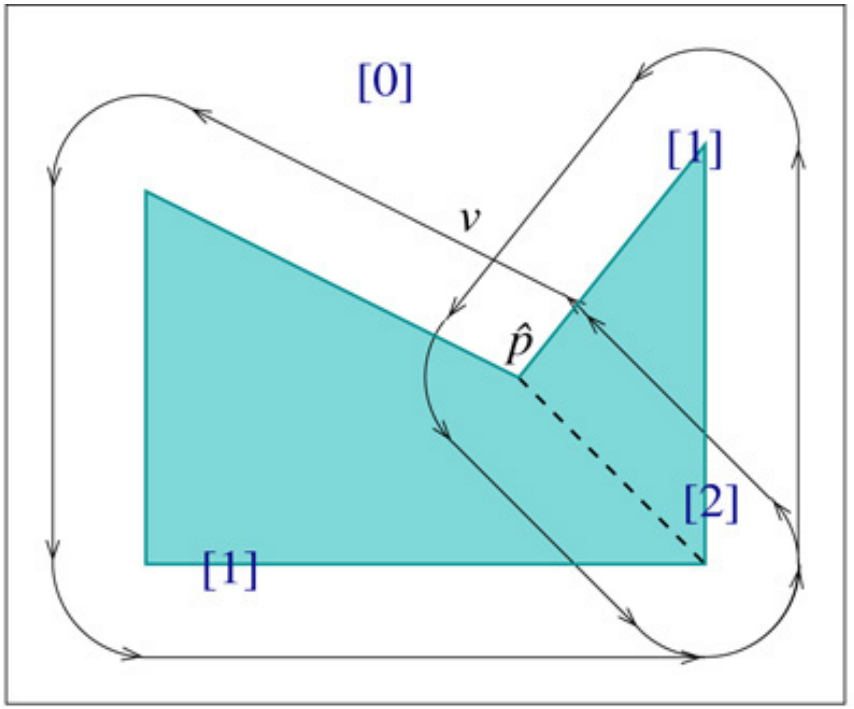
#times the convolution curve winds CCW'ly around f $-$
#times the convolution curve winds CW'ly around f



- can be computed for all faces by BFS over face-adjacency graph - starting with unbounded face
- we finally report on the union of all faces with $W(f) > 0$

Alternative: Construction by Convex Decomposition

1. decompose P into convex sub-polygon $P_1, \ldots, P_k$
2. compute $O_i := \mathit{offset}(P_i, r)$ for all i
3. compute $\bigcup O_i$
4. keep faces with cover number > 0



But still requires arrangements of conics (each O_i is one), and to overlay them

Conics?

All involved curves are conics.

1. we'll see shortly an idea how to make intersections rational!
2. Alternative: curves of arbitrary degree
   - conics have been developped in 2002, no further development
   - algebraic curves are more efficient than conics today ($\approx$5x)
   - offset of offset becomes feasible (problem here: coordinates of intersections)

Approximate Construction

- aim at conservative approximation that is sufficient for many applications
- compute generalized polygon $\tilde{P}$ consisting of line segments and circular arcs
- whose vertices have rational coefficients
- with $\tilde{P} \supseteq P_r = P \oplus D_r$
- and: allow to make approximation error ε arbitrarily small
- that is: output is an arbitrarily tight superset of exact offset

an *interior* approximation is similar

Approximate Construction

- aim at conservative approximation that is sufficient for many applications
- compute generalized polygon $\tilde{P}$ consisting of line segments and circular arcs
- whose vertices have rational coefficients
- with $\tilde{P} \supseteq P_r = P \oplus D_r$
- and: allow to make approximation error ε arbitrarily small
- that is: output is an arbitrarily tight superset of exact offset

an *interior* approximation is similar

Observations

One-root numbers

For $\alpha, \beta, \gamma \in \mathbb{Q}$ and $\gamma > 0$, $\alpha + \beta\sqrt{\gamma}$ is a *one-root number* (ORN)

Traits:

- can be used to represent solutions of $ax^2 + bx + c = 0$, $a, b, c \in \mathbb{Q}$
- Exercise: sign of ORN and comparison of two only needs rational arithmetic

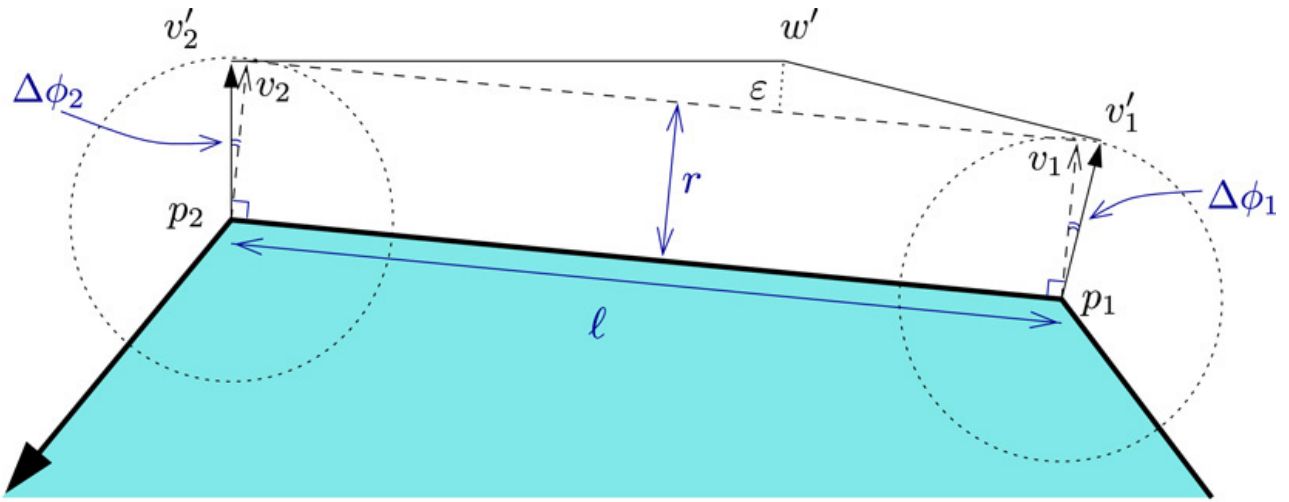
Intersections of rational circles and/or rational lines can be represent with ORN-coordinates.
$\Rightarrow$ all geometric predicates only need rational arithmetic (one order of magnitude faster than conics)

Idea of approximation

For slanted $\overrightarrow{p_1 p_2}$:

- approximate $\overrightarrow{v_1 v_2}$ by $\overrightarrow{v_1' w'}$ and $\overrightarrow{w' v_2'}$
- with v_1', w' and v_2' having rational coordinates
- and $v_i' \in V((x - x_j)^2 + (y - y_j)^2 - r^2)$

R. Wein / Computer-Aided Design 39 (2007) 518–527



Only rational number and one-root numbers involved!

References


Chee-Keng Yap.
Towards exact geometric computation.
Computational Geometry, 7(1-2):3–23, 1997.


Hervé Brönnimann, Christoph Burnikel, and Sylvain Pion.
Interval arithmetic yields efficient dynamic filters for computational geometry.
Discrete Applied Mathematics, 109(1-2):25–47, 2001.


Jean-Daniel Boissonnat and Monique Teillaud.
Effective Computational Geometry for Curves and Surfaces (Mathematics and Visualization).
Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.


Ron Wein.
Exact and approximate construction of offset polygons.
Computer-Aided Design, 39(6):518 – 527, 2007.


CGAL, Computational Geometry Algorithms Library.
http://www.cgal.org.

Algebraic Numbers

Algebraic numbers/expressions:

- expression dag/tree storing approximation(s) per node
- *zero bound* B, i.e. if $|\tilde{E}| < B$, then $E = 0$
  or
- polynomial + isolating interval $[\ell, r]$
- interval can be refined with bisection
  (or quicker using “Newton”-like scheme)
- sign needs evaluation of polynomial
- deciding equality needs gcd of two polynomials

Exercise: How to compare two such numbers?

Algebraic Numbers

Algebraic numbers/expressions:

- expression dag/tree storing approximation(s) per node
- *zero bound* B, i.e. if $|\tilde{E}| < B$, then $E = 0$
  or
- polynomial + isolating interval $[\ell, r]$
- interval can be refined with bisection
  (or quicker using “Newton”-like scheme)
- sign needs evaluation of polynomial
- deciding equality needs gcd of two polynomials

Exercise: How to compare two such numbers?

Polynomials

For a given polynomial $f \in D[x]$ of degree n

- *root* α_0 if $f(\alpha_0) = 0$ ("algebraic number" if $D = \mathbb{Z}$)
- Fundamental Theorem of Algebra:

$$f = \mathrm{lcf}(f) \prod_{i=1}^{k} (x - \alpha_i)^{m_i}$$

  where $\alpha_i \in \bar{Q}$ is a *root* with *multiplicity* $m_i > 0$ ($\sum m_i = n$)
- $f, g \in D[x]$, one can compute $h := \gcd(f, g)$ over Q
- f is *squarefree* iff $\gcd(f, f')$ is a constant (degree 0)
- square-free part of f is given by $\bar{f} = \frac{f}{\gcd(f, f')}$, up to a constant factor

Root Finding

For a polynomial $f(x) \in \mathbb{R}[x]$, compute its real (complex) roots.

Root Finding

For a polynomial $f(x) \in \mathbb{R}[x]$, compute its real (complex) roots.

Problem

No closed-form solution in the general case ($\deg(f) > 4$)
$\Rightarrow$ Determine isolating regions, that is,

- disjoint intervals I_j each containing exactly one real root (*Real Root Isolation*)
- disjoint discs D_j each containing exactly one complex root (*Complex Root Isolation*)

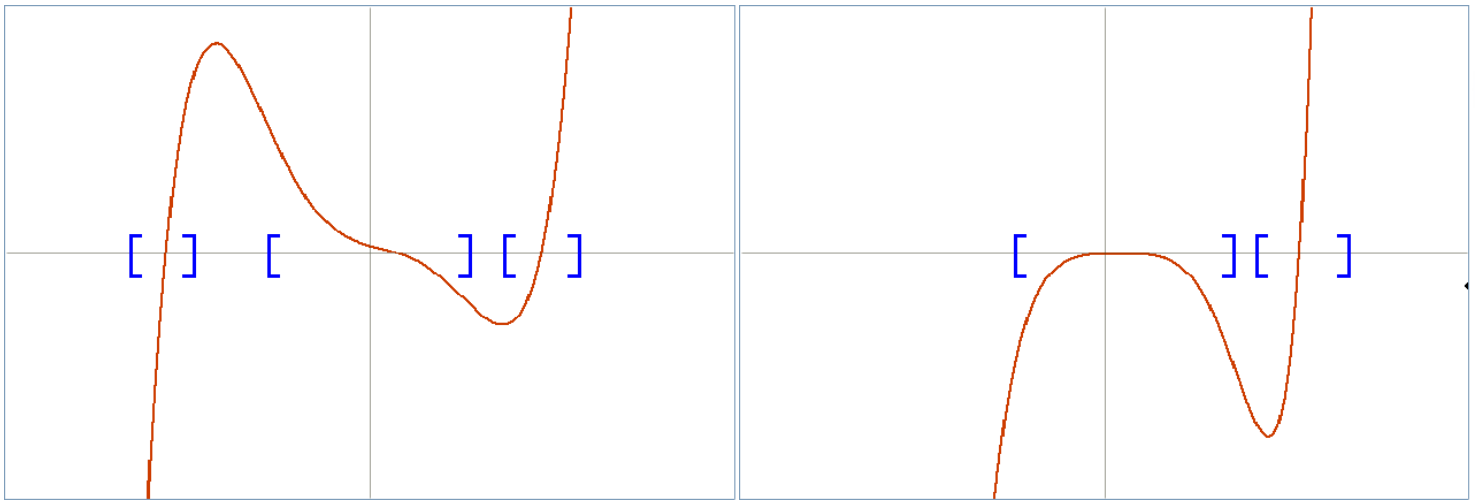
In addition, we want to refine isolating intervals/discs (*Root Refinement*)

(Real) Root Isolation

Given polynomial $p(x) := \sum_{i=0}^{n} a_i x^i \ldots$

Isolation

. . . compute an isolating interval I_k for each element in $V_{\mathbb{R}}(p) := \{\alpha, p(\alpha) = 0\}$ and $V_{\mathbb{R}}(p) \subset \bigcup_k I_k$



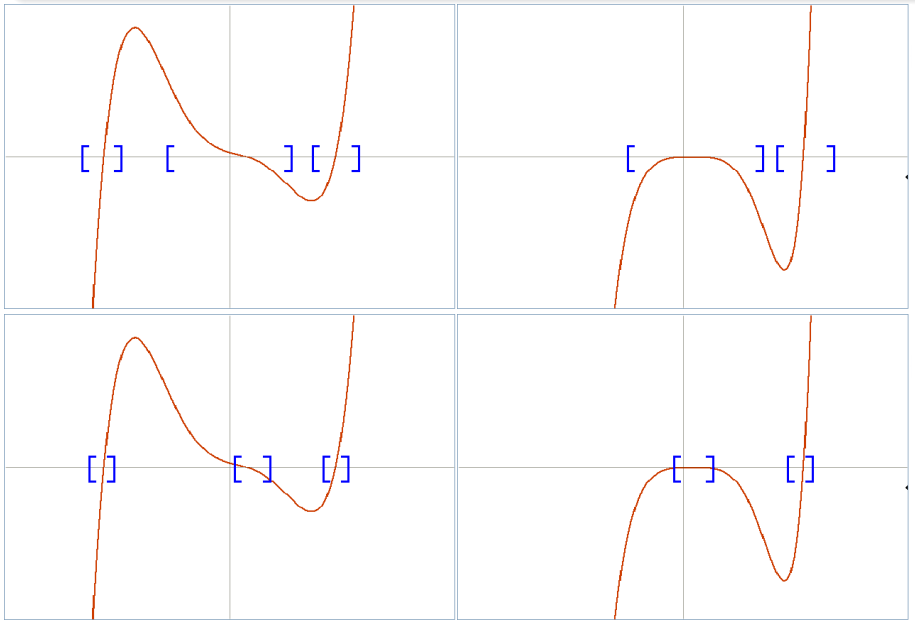
- Roots of higher multiplicity
- Coefficients a_i are not always integers!
- In complex case we consider disks $\subset \mathbb{C}$ than intervals $\subset \mathbb{R}$.

(Real) Root Isolation

Given polynomial $p(x) := \sum_{i=0}^{n} a_i x^i$ ...

Isolation

...compute an isolating interval I_k for each element in $V_{\mathbb{R}}(p) := \{\alpha, p(\alpha) = 0\}$ and $V_{\mathbb{R}}(p) \subset \bigcup_k I_k$



Refinement

...refine intervals arbitrary close to roots.

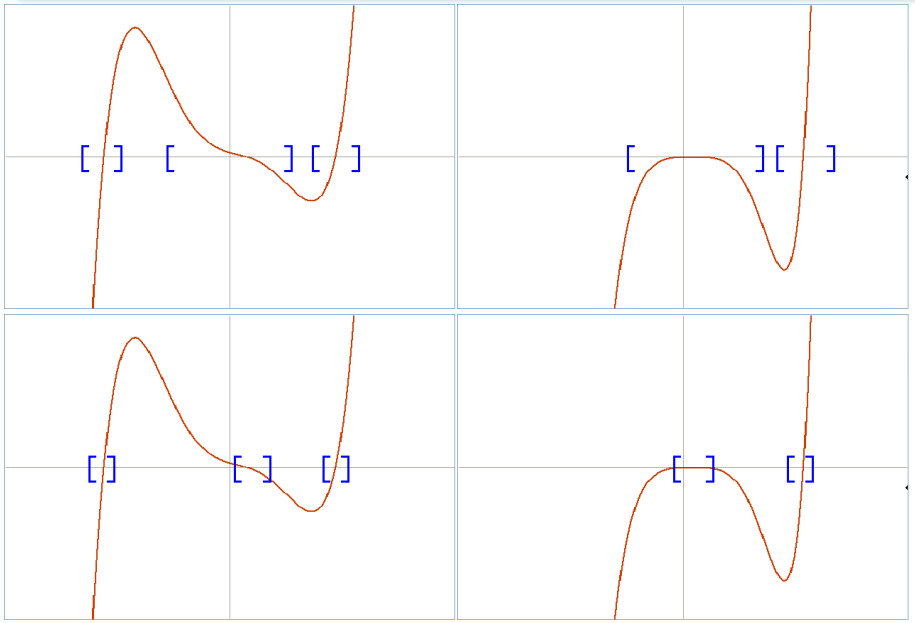
- Roots of higher multiplicity
- Coefficients a_i are not always integers!
- In complex case we consider disks $\subset \mathbb{C}$ than intervals $\subset \mathbb{R}$.

(Real) Root Isolation

Given polynomial $p(x) := \sum_{i=0}^{n} a_i x^i \ldots$

Isolation

. . . compute an isolating interval I_k for each element in $V_{\mathbb{R}}(p) := \{\alpha, p(\alpha) = 0\}$ and $V_{\mathbb{R}}(p) \subset \bigcup_k I_k$



Refinement

. . . refine intervals arbitrary close to roots.

- Roots of higher multiplicity
- Coefficients a_i are not always integers!
- In complex case we consider disks $\subset \mathbb{C}$ than intervals $\subset \mathbb{R}$.

Real Root Isolation - EVAL

EVAL(f, I_0), f squarefree

- $Q \leftarrow \{I_0\}$
- While Q is non-empty
  1. Remove an $I = [a, b]$ from Q, with $\text{mid}(I) = m$ and $\text{width}(I) = w$
  2. If $C_0(I)$ holds, discard I
  3. Else If $C_1(I)$ holds then output I if $S_1(I)$ holds and discard I otherwise
  4. Else If $f(m) = 0$, output $[m, m]$
  5. Else Split $I = [\ell, r]$ at m and put $[\ell, m]$ and $[m, r]$ into Q

Real Root Isolation - EVAL

EVAL(f, I_0), f squarefree

- $Q \leftarrow \{I_0\}$
- While Q is non-empty
  1. Remove an $I = [a, b]$ from Q, with $\text{mid}(I) = m$ and $\text{width}(I) = w$
  2. If $C_0(I)$ holds, discard I
  3. Else If $C_1(I)$ holds then output I if $S_1(I)$ holds and discard I otherwise
  4. Else If $f(m) = 0$, output $[m, m]$
  5. Else Split $I = [\ell, r]$ at m and put $[\ell, m]$ and $[m, r]$ into Q

Real Root Isolation - EVAL

EVAL(f, I_0), f squarefree

- $Q \leftarrow \{I_0\}$
- While Q is non-empty
  1. Remove an $I = [a, b]$ from Q, with $\text{mid}(I) = m$ and $\text{width}(I) = w$
  2. If $C_0(I)$ holds, discard I
  3. Else If $C_1(I)$ holds then output I if $S_1(I)$ holds and discard I otherwise
  4. Else If $f(m) = 0$, output $[m, m]$
  5. Else Split $I = [\ell, r]$ at m and put $[\ell, m]$ and $[m, r]$ into Q

- $C_0(I) \equiv |f(m)| - \sum_{k \geq 1} \frac{|f^{(k)}(m)|}{k!} (\frac{w}{2})^k > 0$
- $C_1(I) \equiv |f'(m)| - \sum_{k \geq 1} \frac{|f^{(k+1)}(m)|}{k!} (\frac{w}{2})^k > 0$
- $S_1(I) \equiv f(a)f(b) < 0$

Descartes' Rule of Signs for Intervals

For an interval $I = (a, b)$ and $n := \deg f$, let

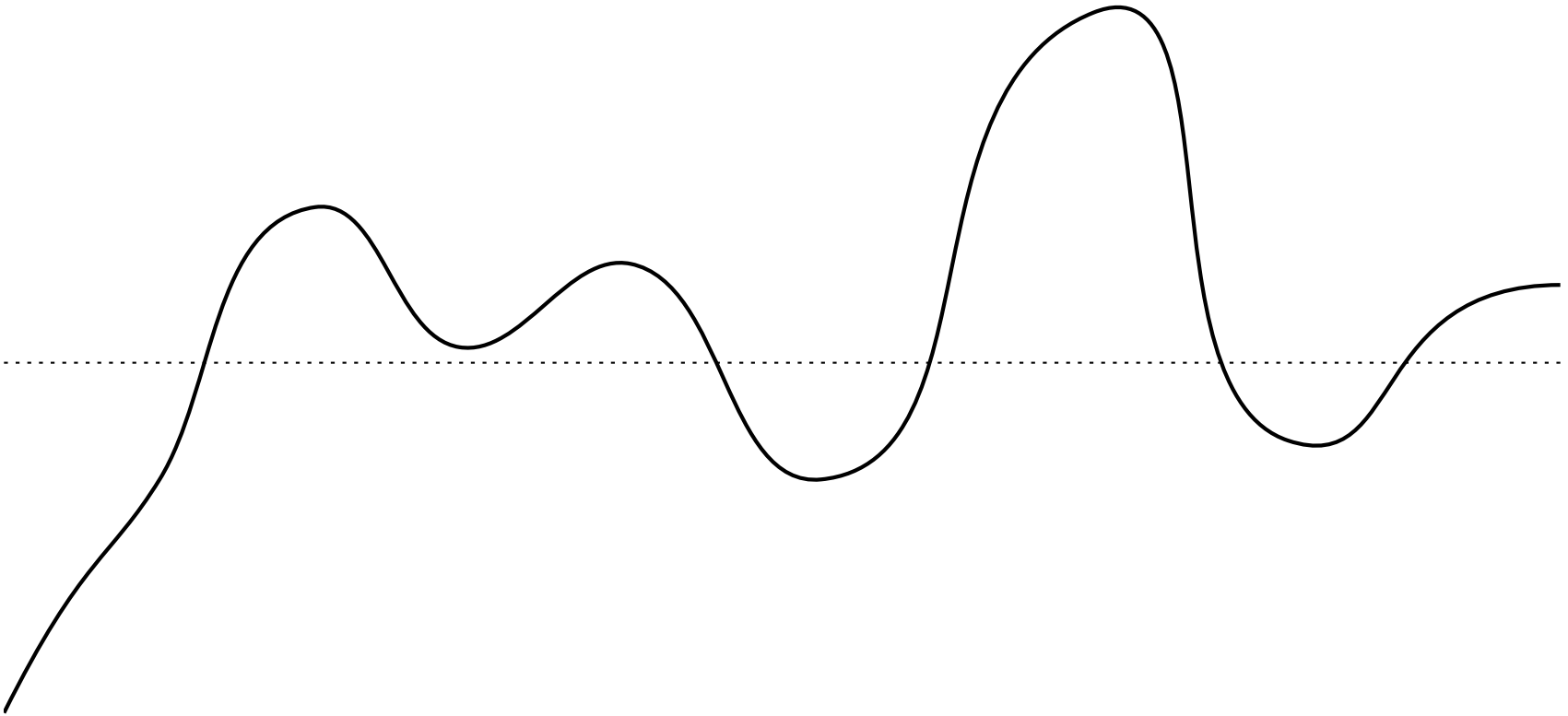
$$f_I(x) = (x+1)^n \cdot f\left(\frac{ax+b}{x+1}\right)$$

and $v := \mathrm{var}(f, I)$ the number of sign variations in the coefficient sequence of f_I. Then, for the number m of real roots in I,

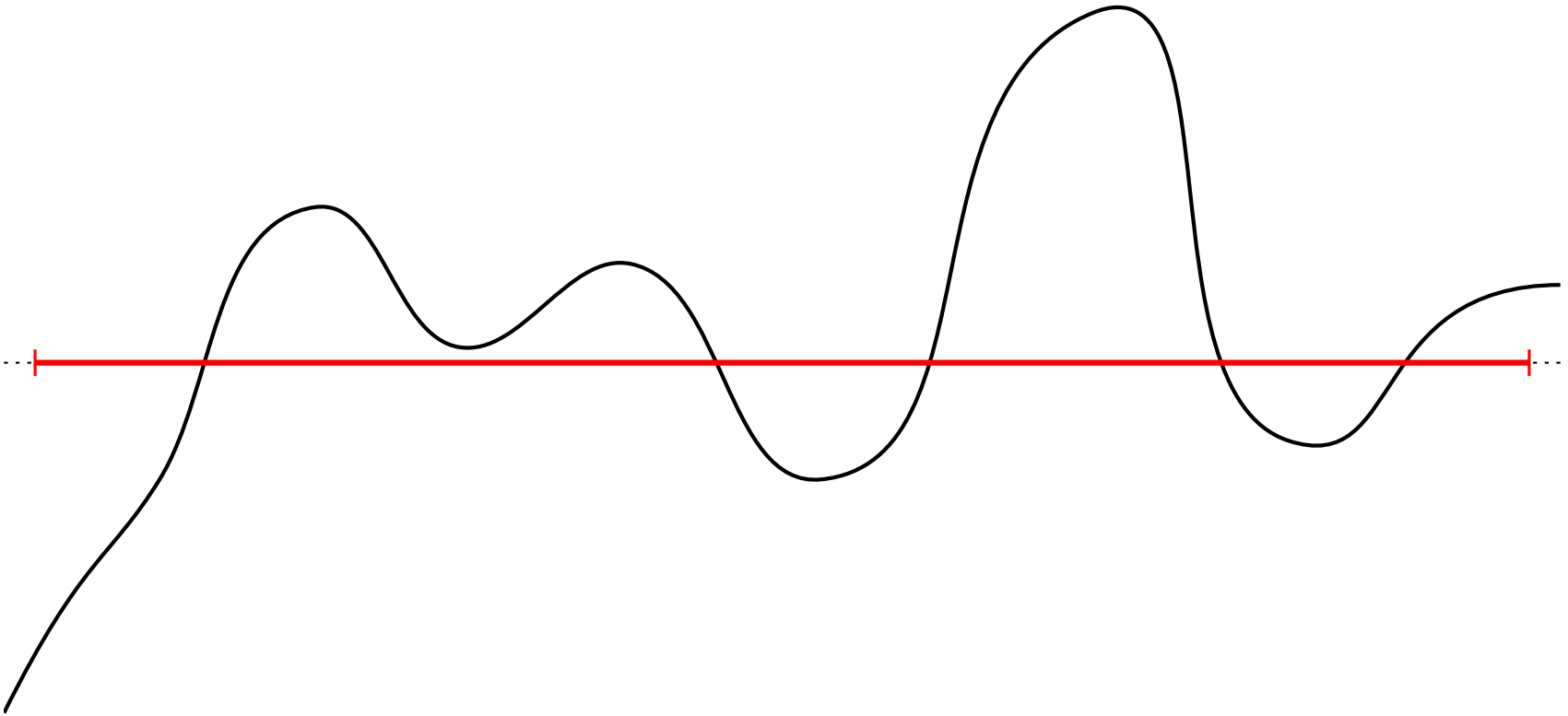
- $m \leq v$, and $m \equiv v \bmod 2$.
- In particular, $v \leq 1$ implies $m = v$.

Example: $f = x^3 - 2x^2 - x + 1$ and $I = (1/2, 4)$.
Then, $f_I = (1/8)x^3 - (15/2)x^2 - (43/2)x + 29$ and $v = 2$.
$\Rightarrow$ f has 0 or 2 real roots in I.

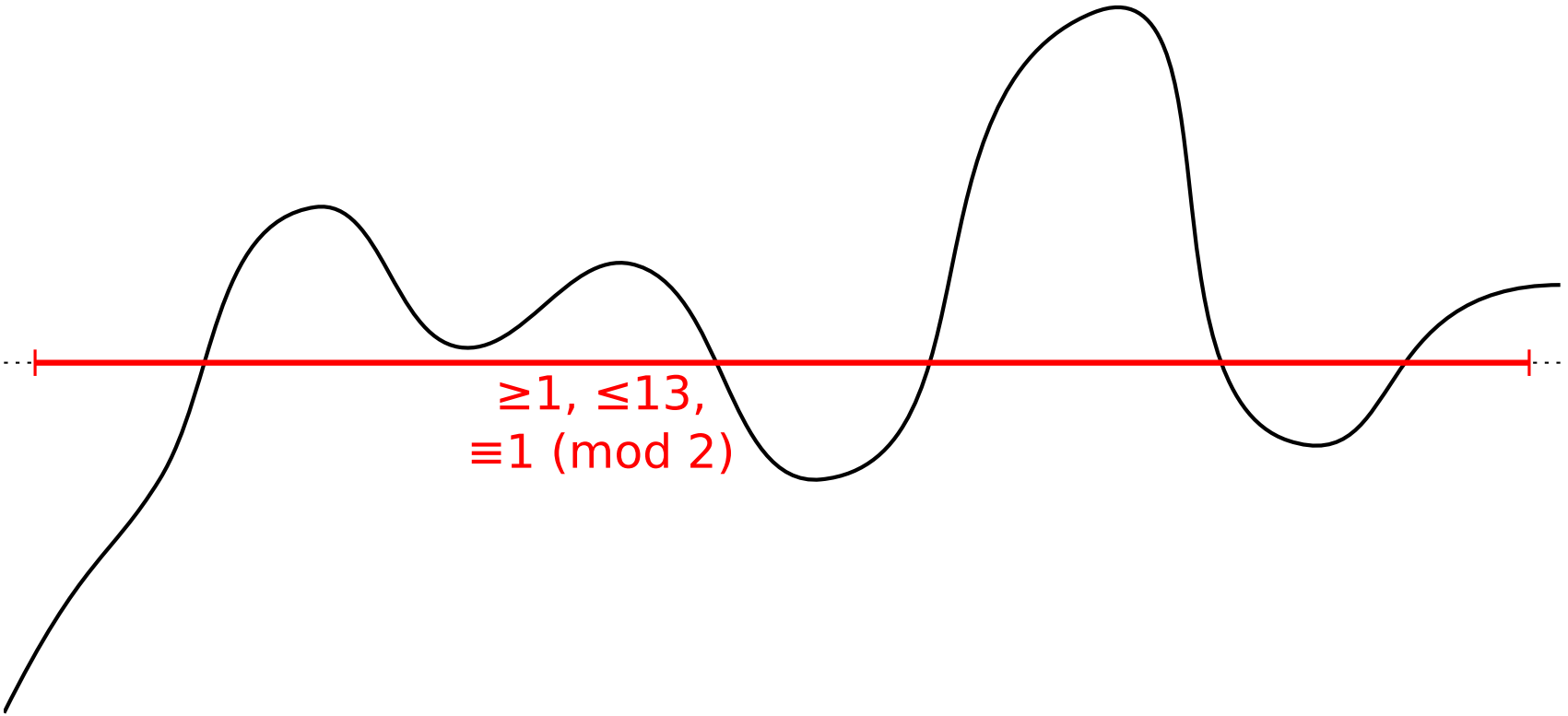
Real Root Isolation - The Descartes Method



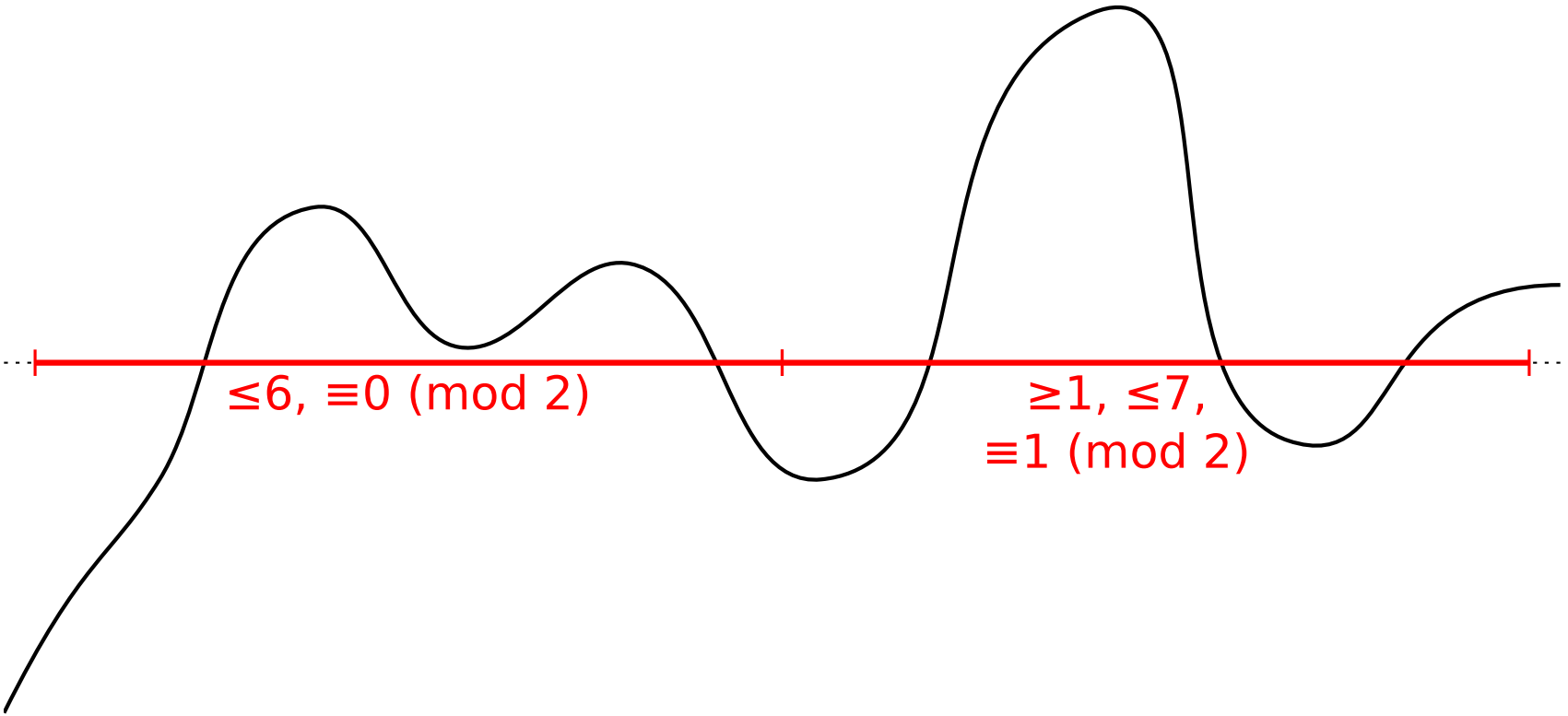
Real Root Isolation - The Descartes Method



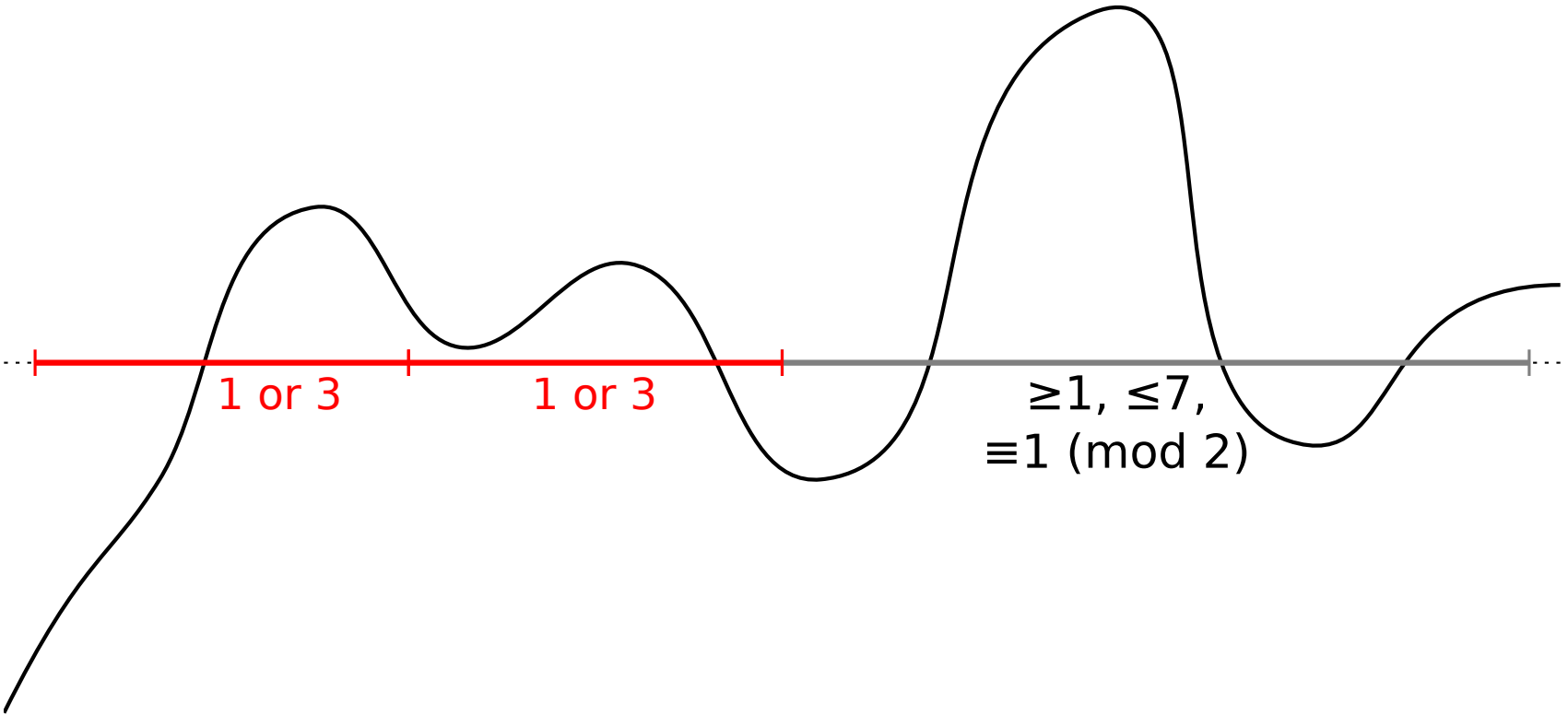
Real Root Isolation - The Descartes Method



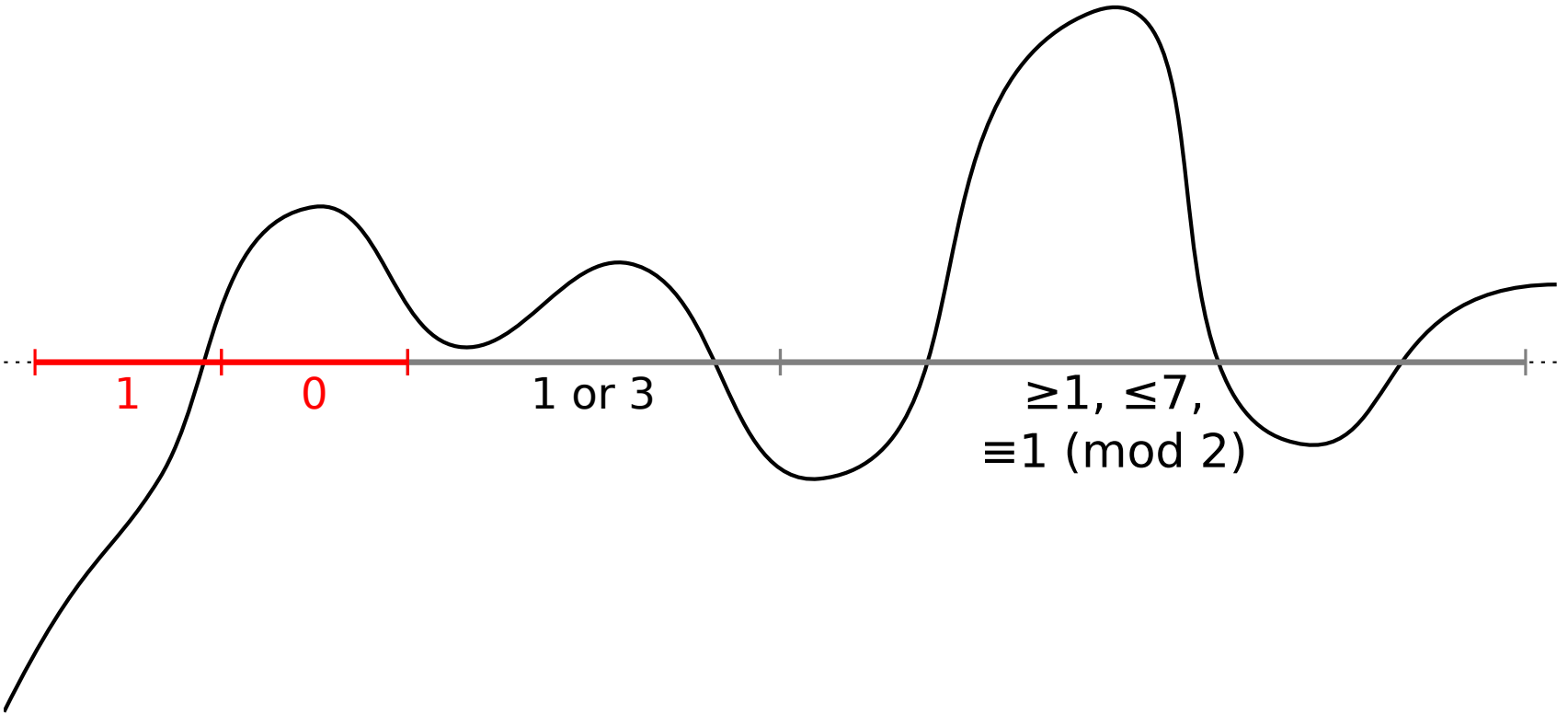
Real Root Isolation - The Descartes Method



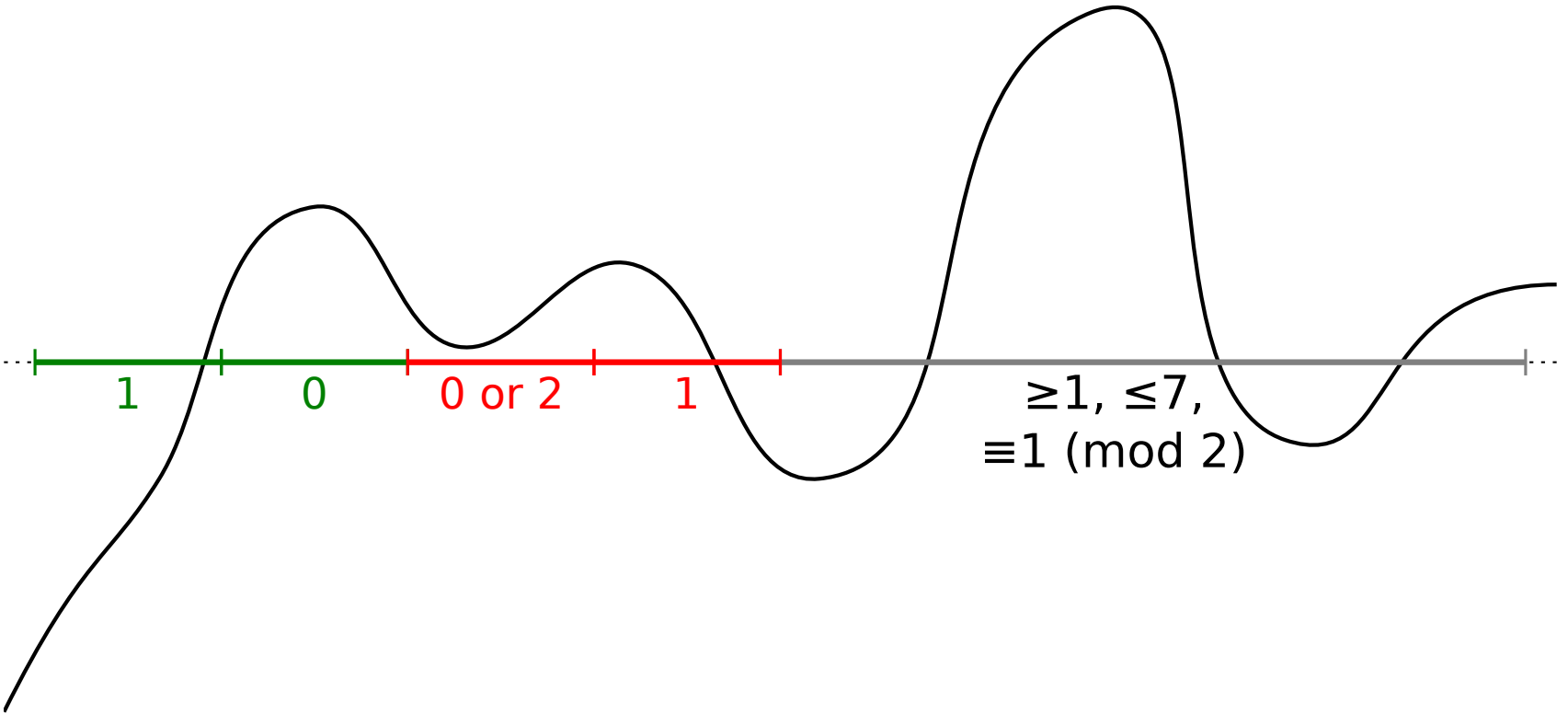
Real Root Isolation - The Descartes Method



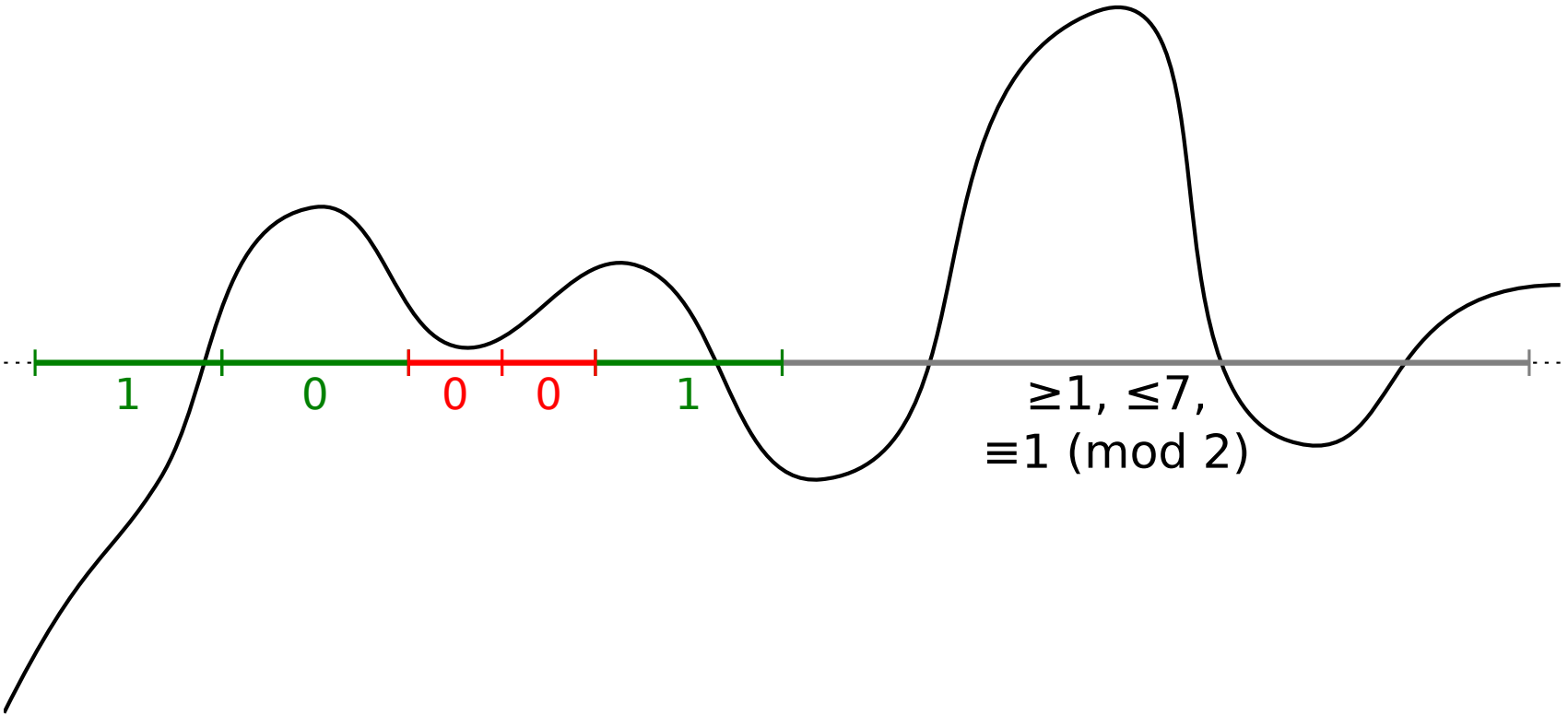
Real Root Isolation - The Descartes Method



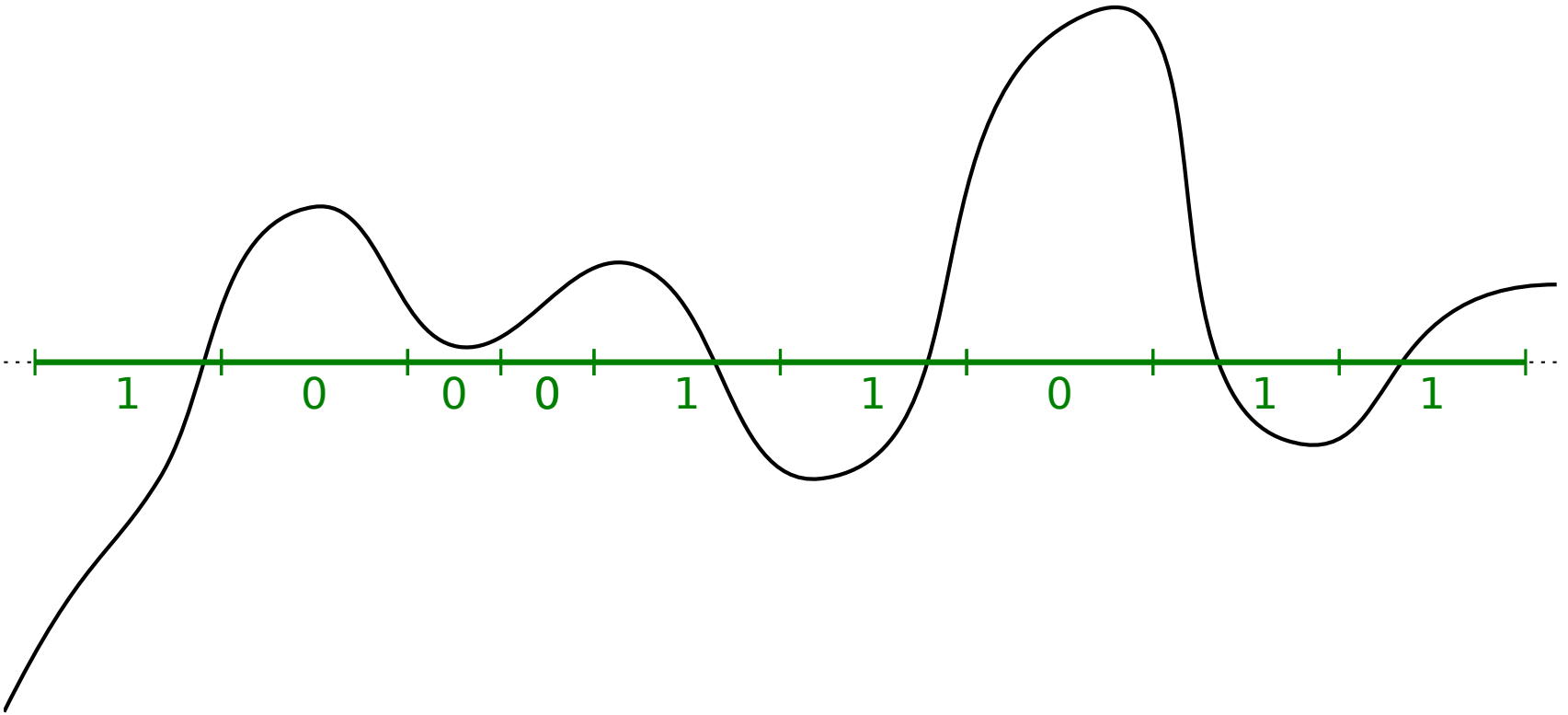
Real Root Isolation - The Descartes Method



Real Root Isolation - The Descartes Method



Real Root Isolation - The Descartes Method



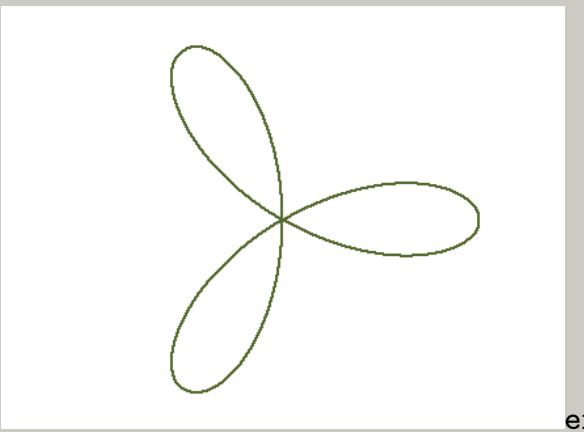
Algebraic curve

Algebraic curve

Defined implicitly by bivariate polynomial $f \in \mathbb{Q}[x, y]$, given by zeroset $V_{\mathbb{R}}(f) := \{(x, y) \in \mathbb{R}^2 \mid f(x, y) = 0\}$.

Example

$f := x^4 + (-1)x^3 + (2y^2)x^2 + (3y^2)x + (y^4)$



exacus.

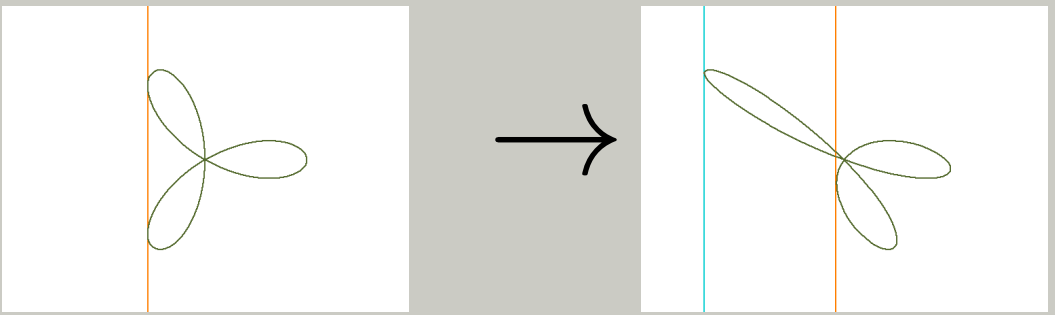
mpi-inf.mpg.de

Algebraic curve

Algebraic curve

Defined implicitly by bivariate polynomial $f \in \mathbb{Q}[x, y]$, given by zeroset $V_{\mathbb{R}}(f) := \{(x, y) \in \mathbb{R}^2 \mid f(x, y) = 0\}$.

Shearing



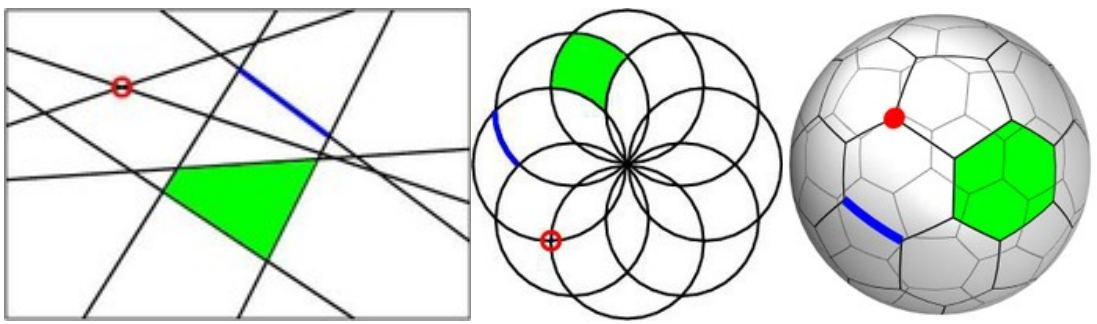
- $f(x, y) \to f(x + sy, y), s \in \mathbb{Z}$
- sparse f becomes dense
- loose geometry

Computing Arrangements

Many geometric algorithms are based on the computation of

Arrangements

Subdivision of plane (space) into cells of dimension 0 (vertices), dimension 1 (edges) and dimension 2 (faces) incuded by a set of curves (surfaces)

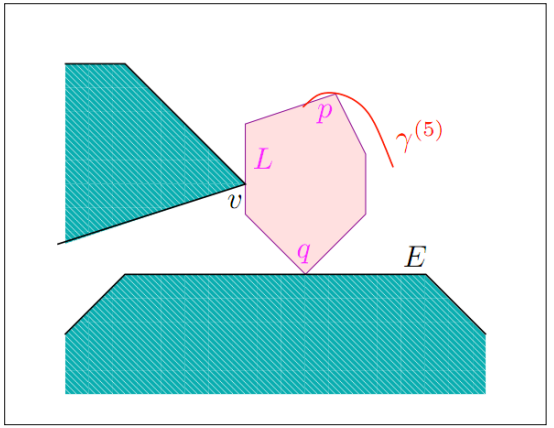


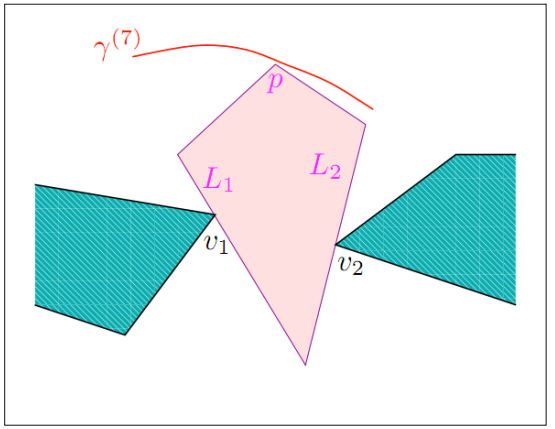
Input: lines, planes, circles, spheres, rational functions, Bézier curves/patches, or general (semi-)algebraic curves/surfaces

Applications of arrangements of algebraic curves

- Robot motion planning (polygonal input $\Rightarrow$ degree 4 curves)

[Wein 2005]

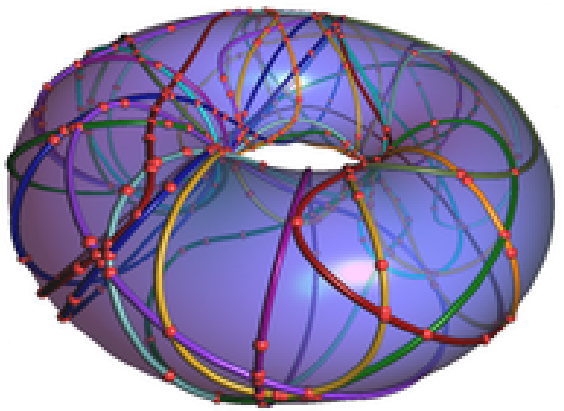




- Arrangements on a surface (via parameterization)
- Analysis of algebraic surfaces (via projection)
- Voronoi cells of lines in 3D (via projection)

Applications of arrangements of algebraic curves

- Robot motion planning (polygonal input $\Rightarrow$ degree 4 curves)
- Arrangements on a surface (via parameterization)



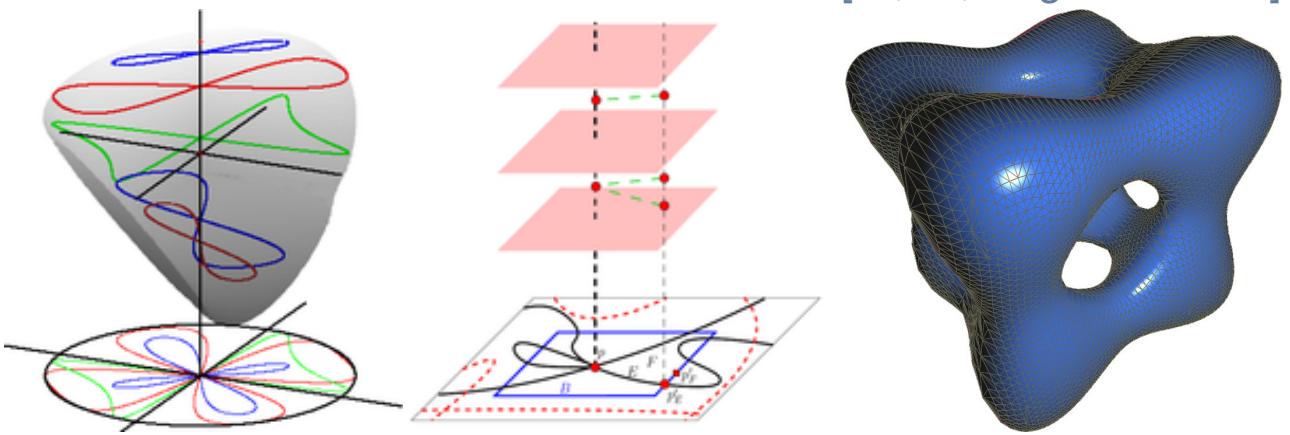
[B., K. 2010]

- Analysis of algebraic surfaces (via projection)
- Voronoi cells of lines in 3D (via projection)

Applications of arrangements of algebraic curves

- Robot motion planning (polygonal input $\Rightarrow$ degree 4 curves)
- Arrangements on a surface (via parameterization)
- Analysis of algebraic surfaces (via projection)

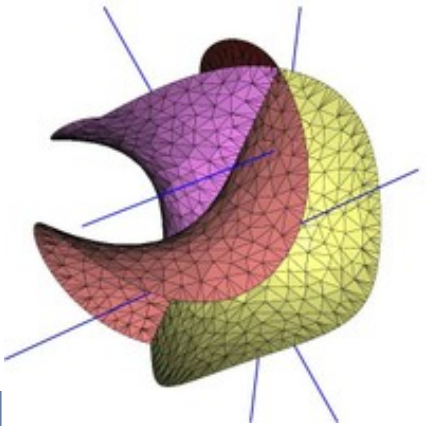
[B., K., Sagraloff 2010]



- Voronoi cells of lines in 3D (via projection)

Applications of arrangements of algebraic curves

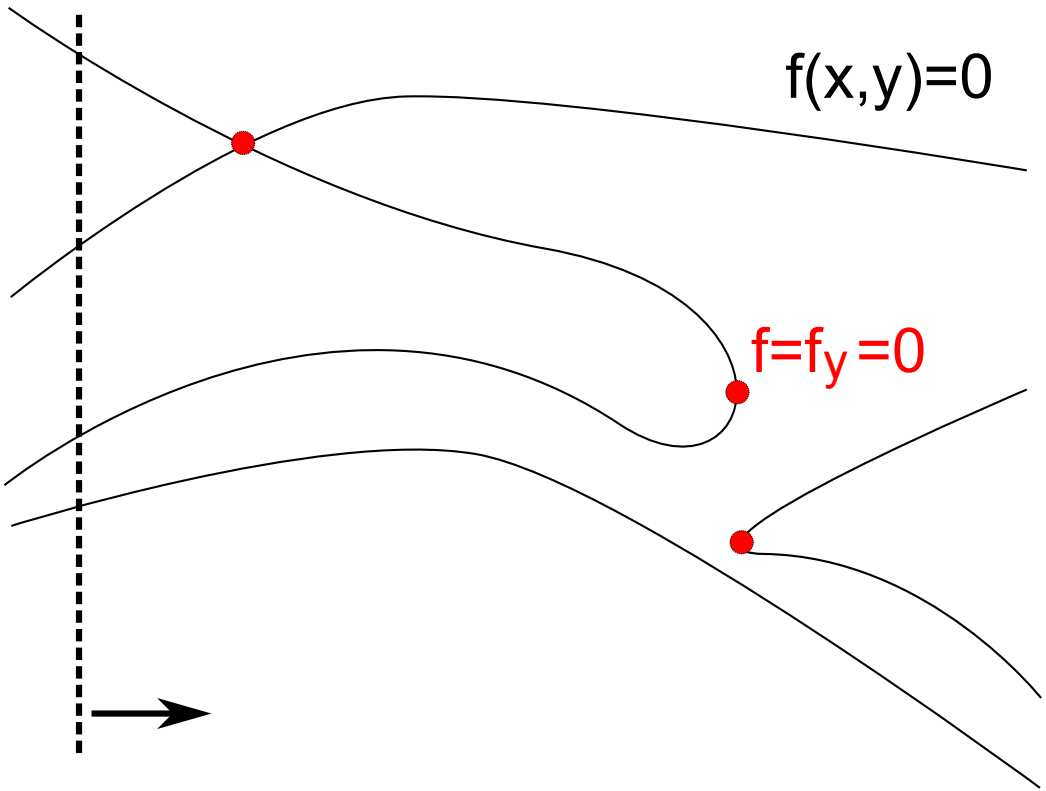
- Robot motion planning (polygonal input $\Rightarrow$ degree 4 curves)
- Arrangements on a surface (via parameterization)
- Analysis of algebraic surfaces (via projection)
- Voronoi cells of lines in 3D (via projection)



[Hemmer, Setter, Halperin 2010]

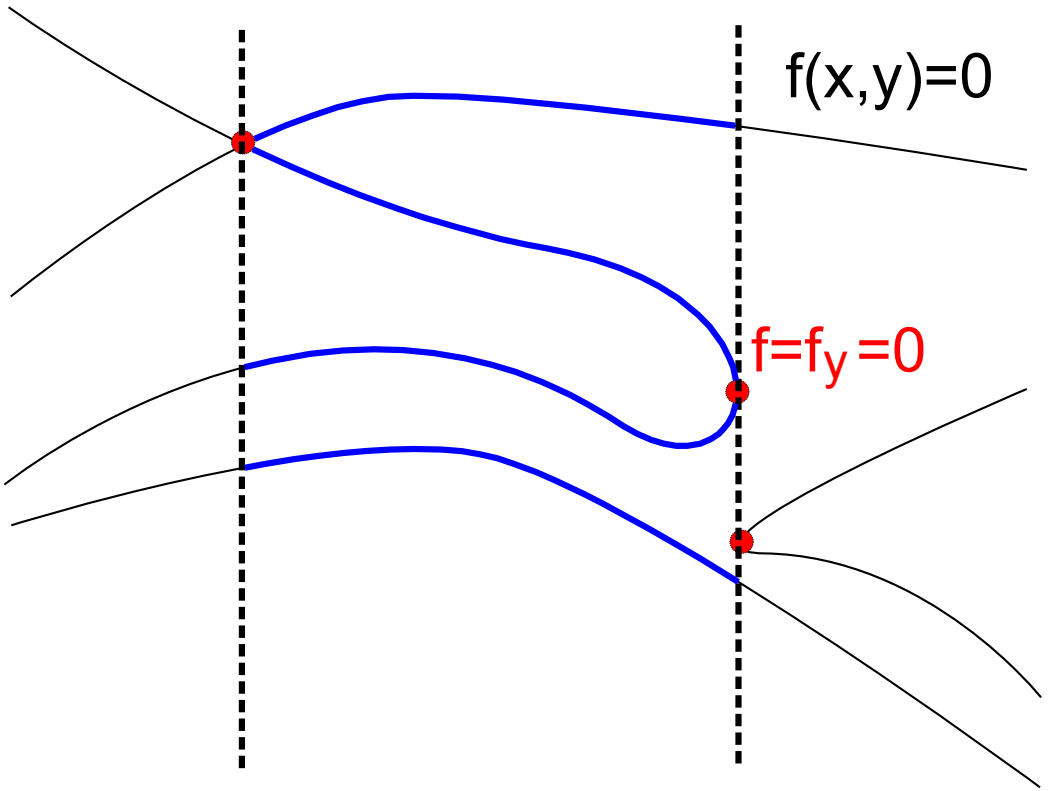
Computing the Topology of an Algebraic Plane Curve

- Sweep with a vertical line along the x-axes, and
- stop whenever an x-critical point (α, β) is reached ($f(\alpha, \beta) = f_y(\alpha, \beta) = 0$).
- the latter step is equivalent to computing the projections of the x-critical points



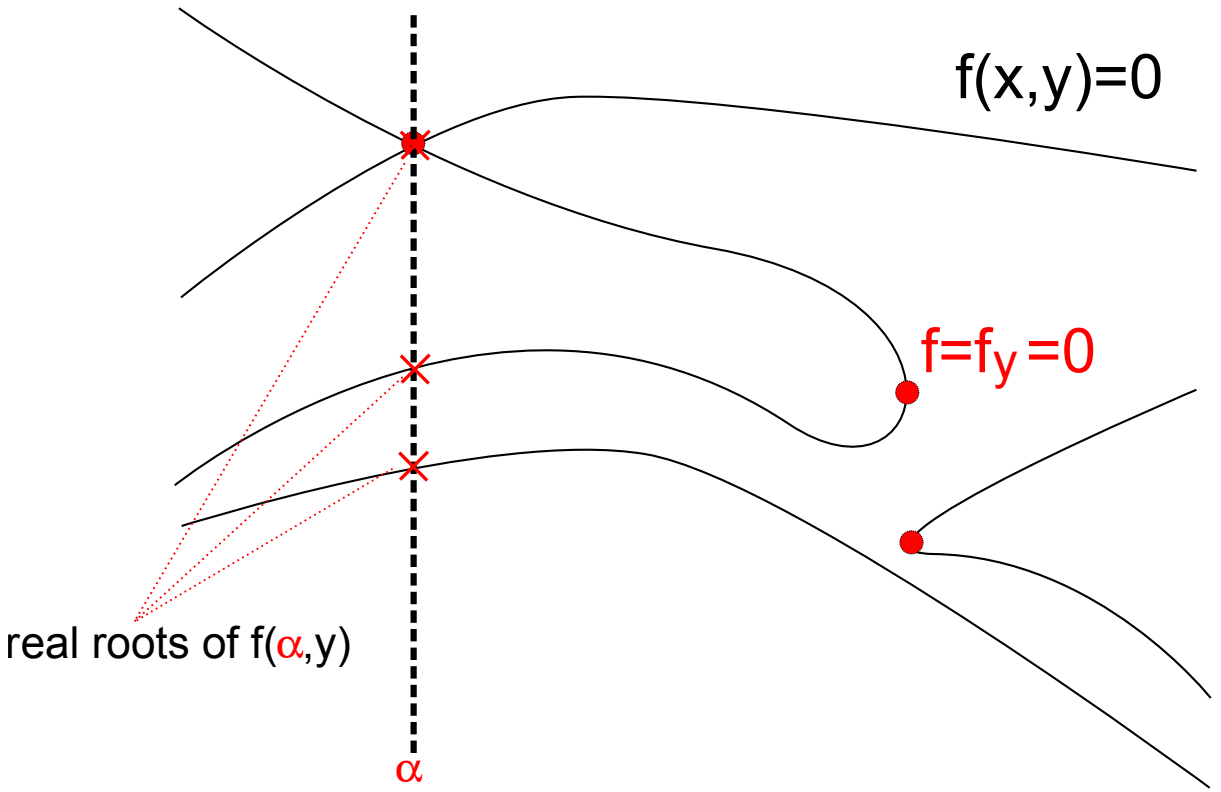
Computing the Topology of an Algebraic Plane Curve

Between two consecutive *x*-critical points, the curve decomposes into disjoint function graphs



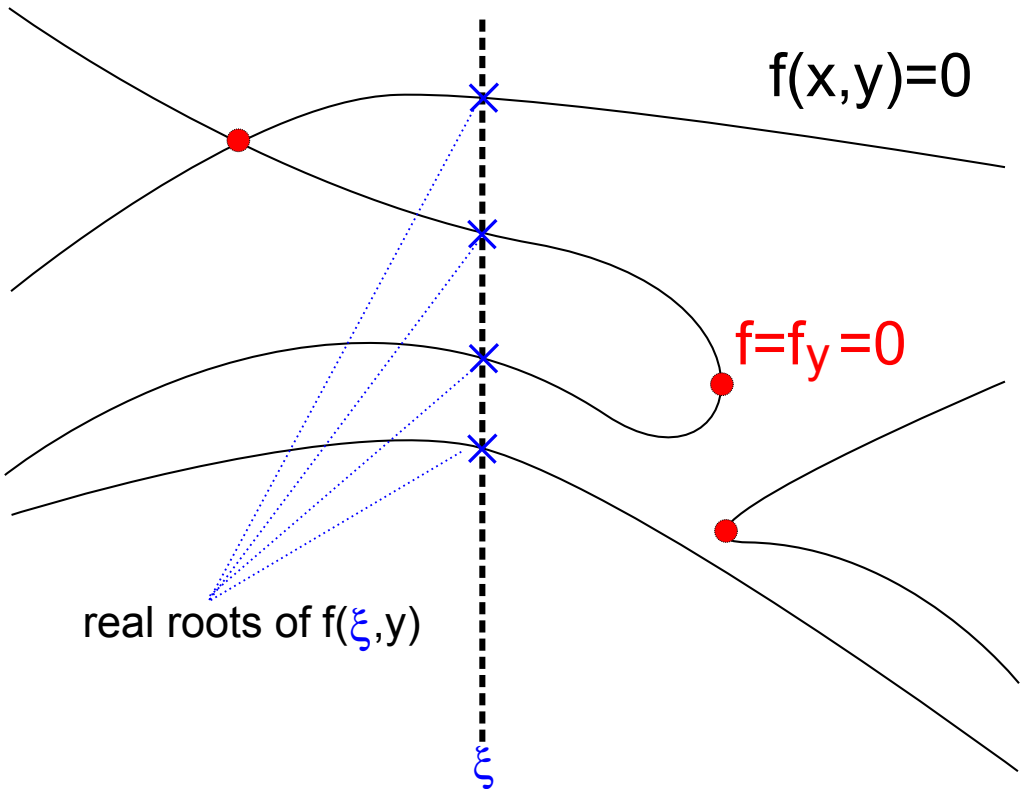
Computing the Topology of an Algebraic Plane Curve

Compute the fiber at x-critical values α, that is, the real roots of $f(\alpha, y)$, and



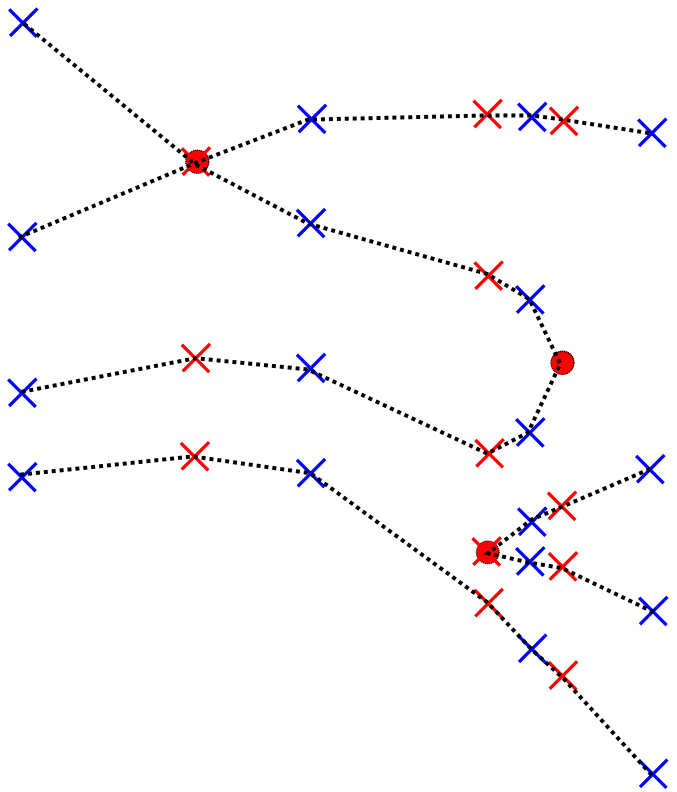
Computing the Topology of an Algebraic Plane Curve

compute the fibers for points in between.



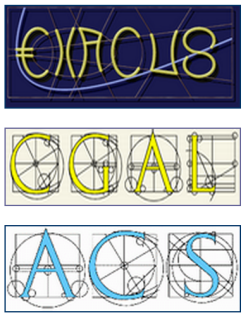
Computing the Topology of an Algebraic Plane Curve

Connect corresponding points in the fibers.



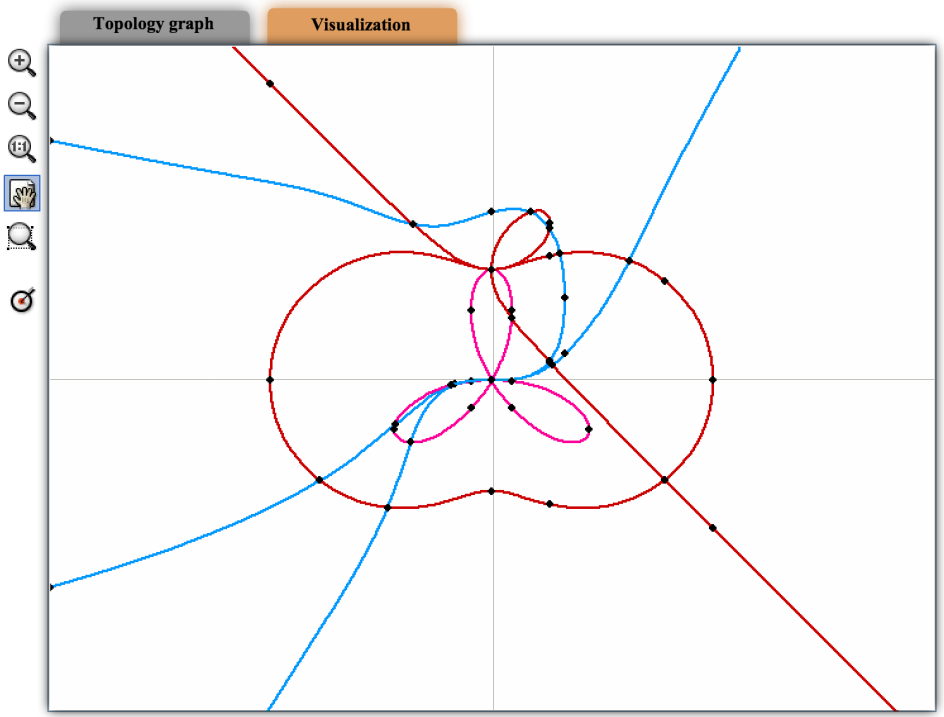
Webdemo

Visualizing arrangements of implicit algebraic curves



max planck institut informatik

Home
Gallery of algebraic curves
Gallery of algebraic surfaces (NEW!)
XAlci web-demo
XTri web-demo (NEW!)
Send feedback
Using the program



Faces: 26 Vertices total: 38
Edges: 63 Isolated vertices: 0

Arcs | Vertices | Faces

1. (-oo; +oo) - (-2; 2.67024); arcno: 0
2. (-oo; -oo) - (-1.55567; -0.905755); arcno: 1
3. (-2; 0) - (-1.55567; -0.905755); arcno: 0
4. (-oo; -oo) - (-0.941676; -1.14856); arcno: 0
5. (-1.55567; -0.905755) - (-0.941676; -1.14856); arcno: 0
6. (-0.880086; -0.444803) - (-0.872285; -0.396958); arcno: 1
7. (-1.55567; -0.905755) - (-0.872285; -0.396958); arcno: 1
8. (-0.941676; -1.14856) - (-0.732556; -0.562431); arcno: 0
9. (-0.880086; -0.444803) - (-0.732556; -0.562431); arcno: 0
10. (-oo; +oo) - (-0.70949; 1.40779); arcno: 2
11. (-2; 2.67024) - (-0.70949; 1.40779); arcno: 2
12. (-0.872285; -0.396958) - (-0.362887; -0.0455271); arcno:
13. (-0.872285; -0.396958) - (-0.362887; -0.0455271); arcno:
14. (-0.732556; -0.562431) - (-0.331494; -0.0377487); arcno:
15. (-0.362887; -0.0455271) - (-0.331494; -0.0377487); arcnc
16. (-0.732556; -0.562431) - (-0.184504; -0.253156); arcno: 0
17. (-0.331494; -0.0377487) - (-0.184504; -0.0114495); arcnc

Rasterize: (•) complete ( ) complete one-color ( ) selected arcs

☑ show coordinate axes Save plot

Type in polynomials delimited by comma: Coordinates: (-0.1595903563; 0.4488636364)

y^4+(-1)*y^3+(2*x^2)*y^2+(3*x^2)*y+(x^4),y^7+(-3)*y^6+(2*x^2+(-1)*x+2)*y^5+(x^3+(-6)*x^2+x+2)*y^4+(x^4+(-2)*x^3+2*x^2+x+(-3))*y^3+(2*x^5+(-3)*x^4+x^3+10*x^2+(-1)*x+1)*y^2+((-1)*x^5+3*x^4+4*x^3+(-12)*x^2)*y+(x^7+(-3)*x^5+(-1)*x^4+(-4)*x^3+4*x^2),(-1)*y^7+(2*x)*y^5+((-8)*x^3+(-8)*x^2)*y^3+(8*x^5+8)*y^2+((-16)*x^3)*y+(8*x^6)

Analyse curves
Show topology graph
Visualize curve arcs

Xalci on Flash. Version 2.1.0 Built 13.03.2012

Polynomials can be entered in both rational and floating-point formats, coefficients of arbitrary length are accepted:
for instance, **1e-3(x+2y)-5/1234(1.45y^2+x^2y)-0.0001** is a valid input.

To see more examples of algebraic curves and arrangements of such, visit our Curve gallery. Web-application design and server software by Pavel Emeliyanenko.

Why is this difficult?

Consider again the offset of the ellipse $8x^2 + 127y^2 - 8 = 0$:

$$f(x,y) = 67649929216x^8 - 38623220817920x^2y^4 - 8339109519360y^2x^4 + \cdots = 0$$

The projections of the x-critical points of f are the real roots of the (resultant) polynomial $R(x) = \mathrm{res}(f, f_y, y)$:

$$R(x) = \det \underbrace{\begin{bmatrix} 17048839192576 & 0 & 25673940910080 + 36245563637760x^2 & \cdots \\ 0 & 17048839192576 & 0 & \cdots \\ \vdots & & & \vdots \\ 136390713540608 & 0 & 154043645460480 + 217473381826560x^2 & \cdots \\ 0 & 136390713540608 & 0 & \cdots \\ \vdots & & & \vdots \end{bmatrix}}_{\text{Töplitz-like matrix of size } 15\times 15 \text{ with polynomial entries in } x}$$

$$= \underbrace{839\ldots161}_{\text{Bit-length}\approx 500} \cdot x^{48} - 178\ldots956 \cdot x^{46} + \cdots$$

$\Rightarrow$ Computing R is costly; R is complicated, and thus we need efficient techniques to further proceed with R (root finding, etc.).

Why is this difficult?

Computing the fibers at a projection α of an x-critical point is even more difficult:

- $f_\alpha(y) = f(\alpha, y) \in \mathbb{R}[y]$ is a polynomial with complicated algebraic coefficients (polynomials in α)
- $f_\alpha(y)$ has a multiple root, and
- α has a high algebraic degree

$\Rightarrow$ Any approach which exclusively performs exact computation with the coefficients of $f_\alpha(y)$ will fail in practice.

Computing Resultants (and GCDs) of Polynomials

Resultant $R(x)$ of two bivariate polynomials can be computed via a signed-remainder sequence (similar to Euclid's algorithm):

- coefficients in the intermediate results become very large
- serial approach
- bottleneck in all previous implementations

New implementation on graphics hardware:

- modular approach working over prime fields $\mathbb{Z}/p_i$
- for each prime p_i,
  - consider interpolation points $\alpha_{ij} \in \mathbb{Z}/p_i$
  - evaluate $R(\alpha_{ij}) \in \mathbb{Z}/p_i$ as the determinant of the Sylvestermatrix $S(x)$, where $x = \alpha_{ij}$ (Schur's algorithm exploiting the special structure of S)
  - interpolate $R(x)$ over $\mathbb{Z}/p_i$
- lift to a solution $R(x) \in \mathbb{Z}[x]$ via Chinese Remaindering

Computing Resultants (and GCDs) of Polynomials

Game Changer

Resultants and GCDs can now be computed up to 500$\times$ faster!

Computing Resultants (and GCDs) of Polynomials

Game Changer

Resultants and GCDs can now be computed up to 500× faster!

Take Home Message III

Modular algorithms are well-suited for symbolic computations

- coefficient blowup can be avoided
- parallelization

All sophisticated computer algebra systems are currently switching to highly parallelized modular computation.

Results for Arrangements of Algebraic Curves

- Computing arrangements of planar algebraic curves of arbitrary degree
  - complete (all special cases included) and exact
  - curves of degree up to 50 are handled in a reasonable time on a normal workstation
  - curves of lower degree (such as the offset of the ellipse) are handled in milliseconds

Arrangement computation in CGAL ...

... is reduced to CAD-like analyses ...

- of single curves, to decompose them into x-monotone arcs
- of pairs of curves, to provide geometric operations such as intersections and vertical alignment of points and curves

Contribution

While ignoring upstream (combinatorial) layers we show

- **curve analysis** and **curve-pair analysis**
- **only two symbolic operations**
- **new symbolic-numerical lifting** avoids shearing
- **efficient**

Arrangement computation in CGAL ...

... is reduced to CAD-like analyses ...

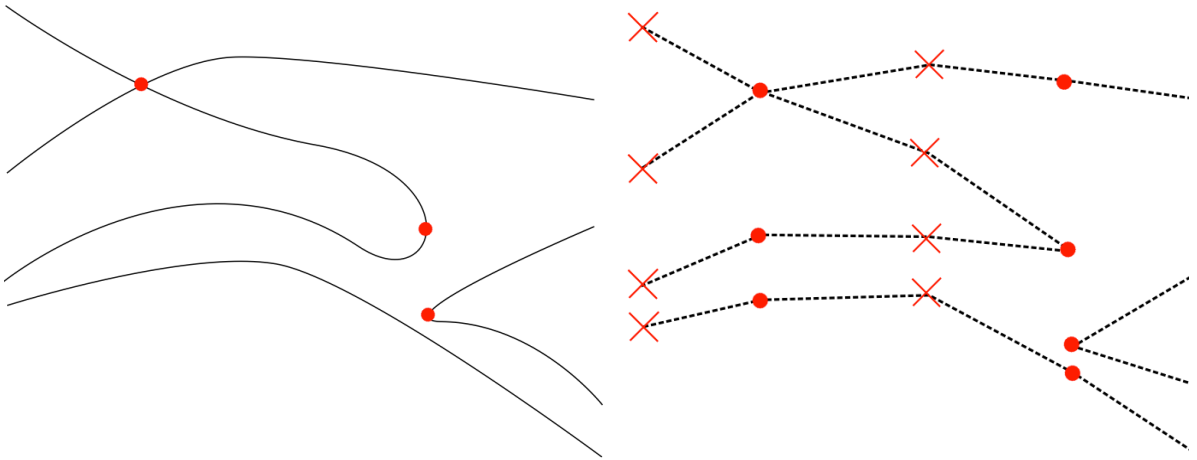
- of single curves, to decompose them into x-monotone arcs
- of pairs of curves, to provide geometric operations such as intersections and vertical alignment of points and curves

Contribution

While ignoring upstream (combinatorial) layers we show

- **curve analysis** and **curve-pair analysis**
- **only two symbolic operations**
- **new symbolic-numerical lifting** avoids shearing
- **efficient**

Curve analysis: Isotopic approximation

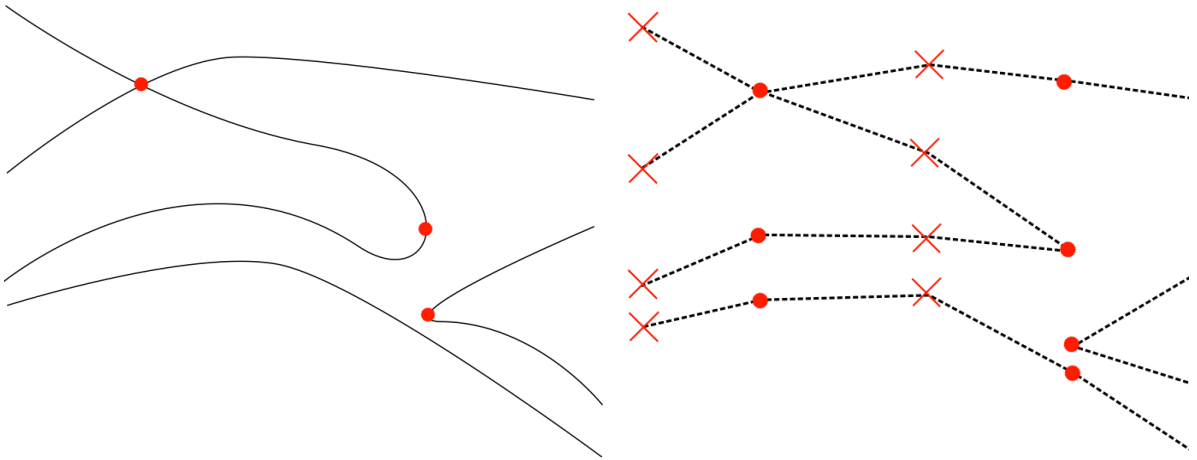


- Approximate curve with a planar (segment)-graph whose
- Vertices are located on the curve
- Identify x-extreme points, i.e., (x, y) with

$$f(x, y) = f_y(x, y) = 0$$

- Identify (vertical) asymptotes

Curve analysis: Isotopic approximation



- Approximate curve with a planar (segment)-graph whose
- Vertices are located on the curve
- Identify x-extreme points, i.e., (x, y) with

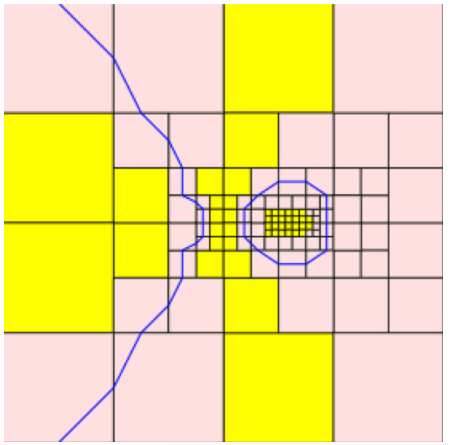
$$f(x, y) = f_y(x, y) = 0$$

- Identify (vertical) asymptotes

Subdivision

Subdivide initial box until a subbox is either

- *discarded by exclusion predicate*: e.g., $0 \notin \square f([a,b],[c,d])$
- *"certified" by inclusion predicate*, based on $\square f_x$ and $\square f_y$ (plus separation bound)
  - often: ε-close
  - optimal: box to contain an isotopic approximation (hard to achieve)



[Burr,Choi,Yap 2010]

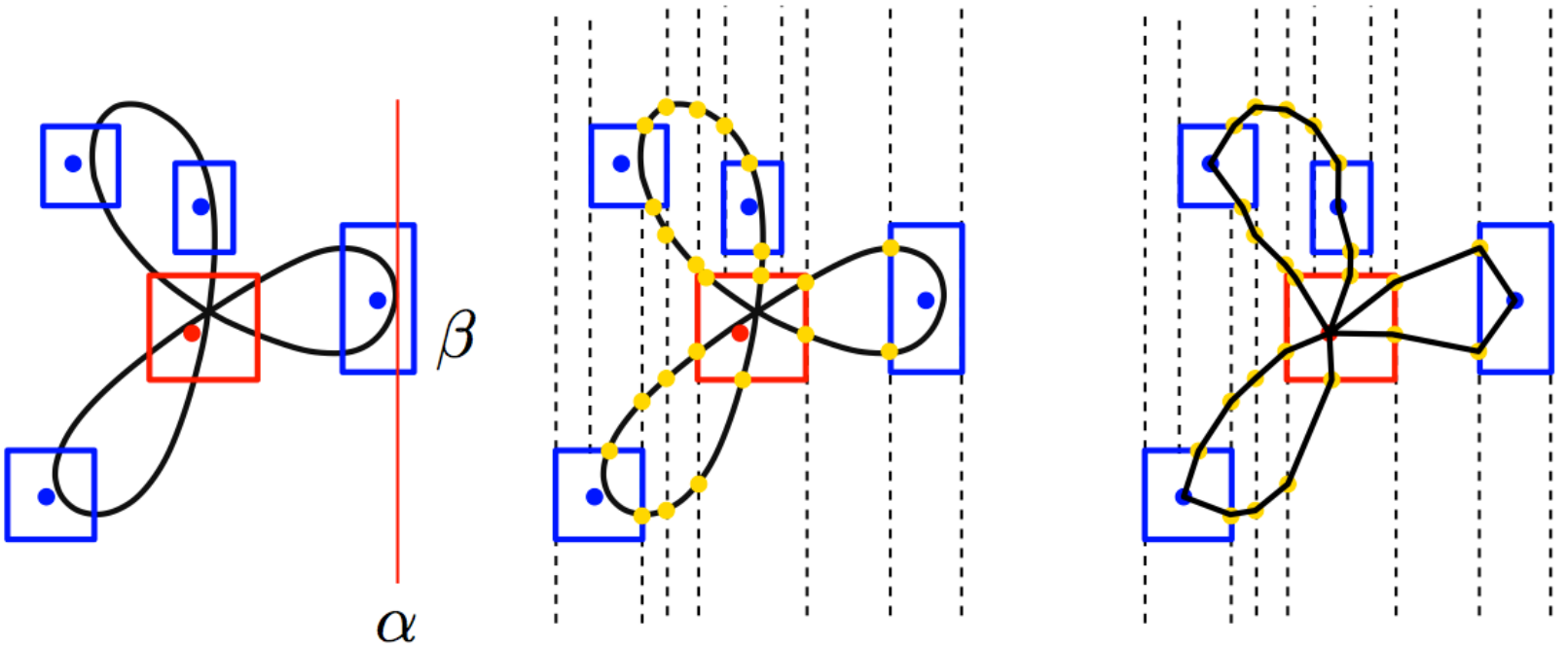
Discussion

(+) Fast, especially with numerical predicates

(o) Localizable, though subject to miss components

(-) Practically not certifying. i.e., completeness requires non-practical bounds

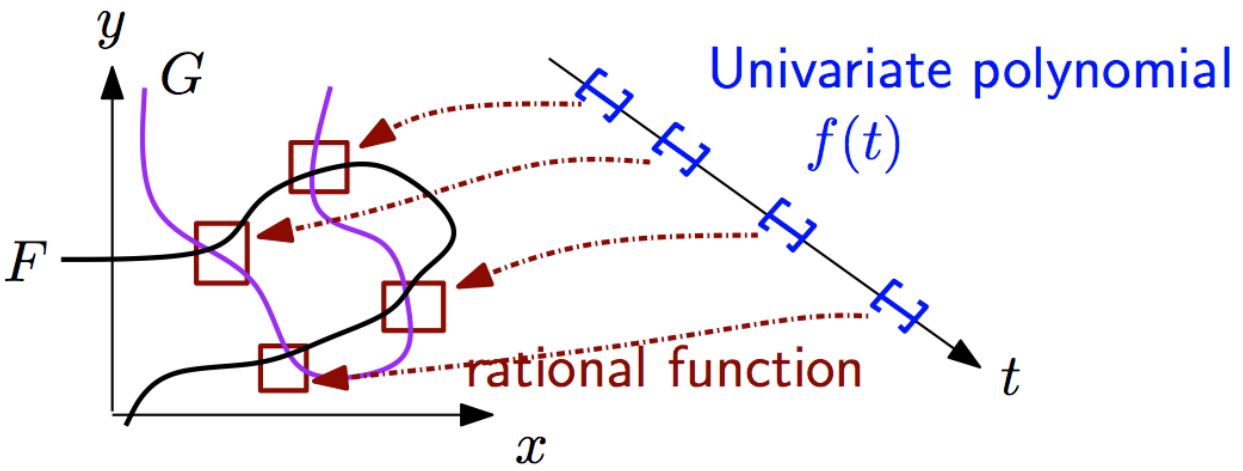
Isotop



[Talk by Lazard], [Bouzidi, Lazard, Pouget, Rouillier 2011]

- Isolate critical points (α, β) with boxes and compute $k := \text{mult}(\beta, f(\alpha, y))$
- Rectangular partition; topology in rectangles often “simple”; esp. when avoiding curves $\frac{\partial^k f}{\partial y^k}$ in rectangles
- Connect boxes

Rational Univariate Representation

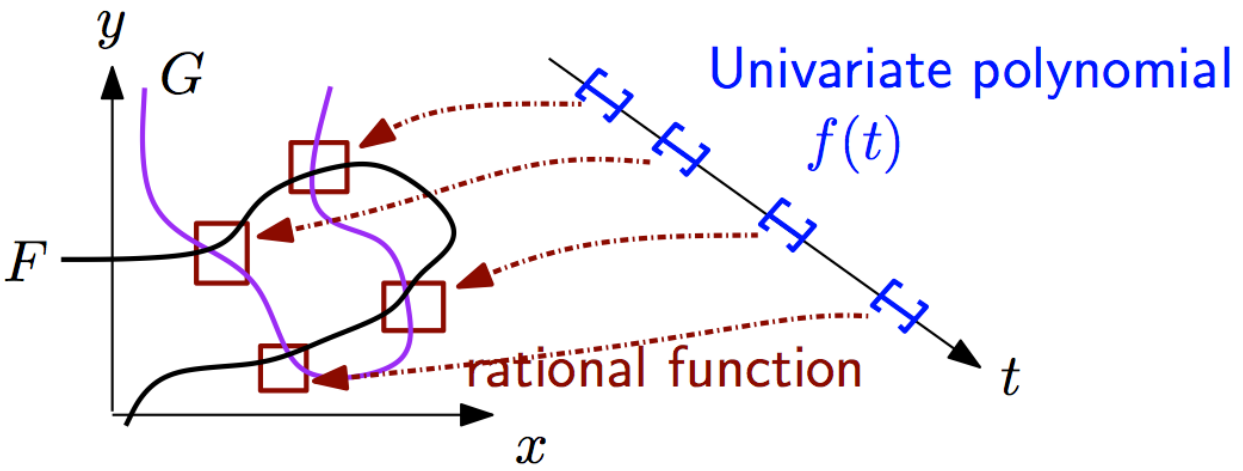


[Talk by Lazard]

- Certification via RUR based on Gröbner basis
- Computationally “equivalent” to subresultants
- BUT: New version relies on modular arithmetic
- Not loosing any solution, but spurios exists
- Thus: Final validation needed

Isotop beat CGAL’s approach (next) in many examples.

Rational Univariate Representation

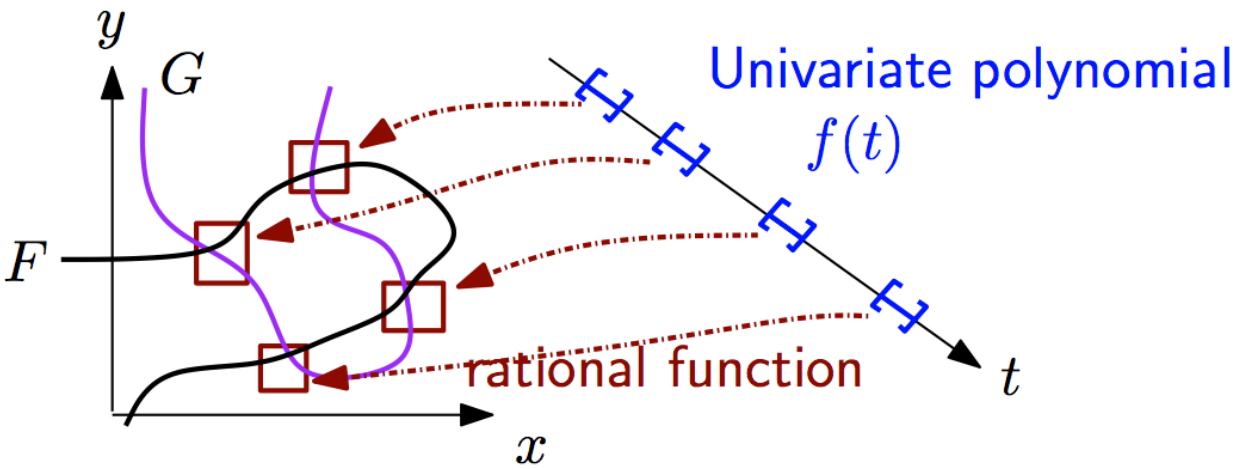


[Talk by Lazard]

- Certification via RUR based on Gröbner basis
- Computationally “equivalent” to subresultants
- BUT: New version relies on modular arithmetic
- Not loosing any solution, but spurios exists
- Thus: Final validation needed

Isotop beat CGAL’s approach (next) in many examples.

Rational Univariate Representation

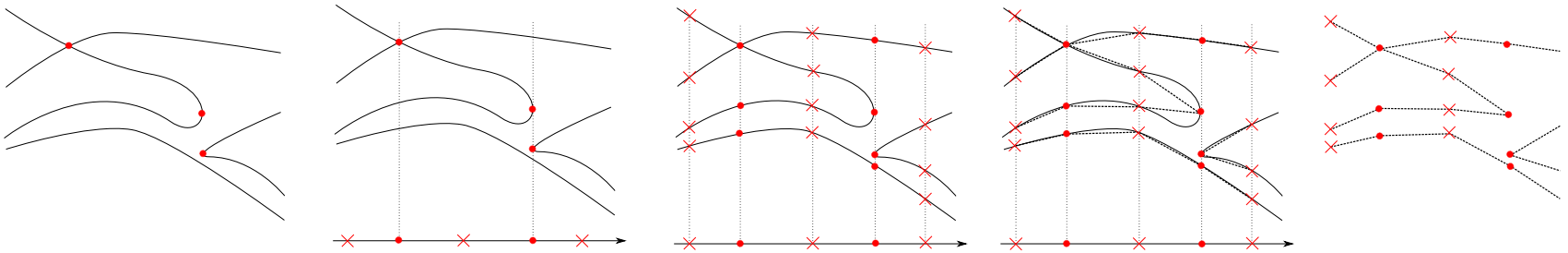


[Talk by Lazard]

- Certification via RUR based on Gröbner basis
- Computationally “equivalent” to subresultants
- BUT: New version relies on modular arithmetic
- Not loosing any solution, but spurios exists
- Thus: Final validation needed

Isotop beat CGAL’s approach (next) in many examples.

Curve analysis via projection – the CAD way

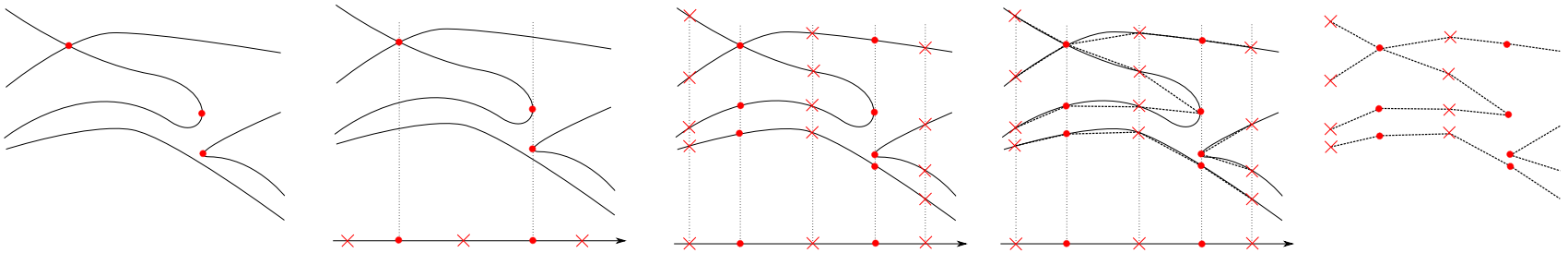


- **Projection**. Isolating roots of resultant
- **Lifting**. Roots of *fiber-polynomials* $f(\alpha, y)$ and $f(q_I, y)$
  - $f(\alpha, y)$ has *non-rational coefficients* and *multiple roots*
- **Connect** arcs to given vertices (skipped today)

Crux in **Lifting**. CGAL's solution: [Eigenwillig, K., Wolpert 2007]

- non-rational coefficients handled by "interval-Descartes"
- requires to know #distinct (real and complex) roots
- needs full subresultant sequence: expensive to compute
- more than one multiple root: expensive shearing

Curve analysis via projection – the CAD way



- **Projection**. Isolating roots of resultant
- **Lifting**. Roots of *fiber-polynomials* $f(\alpha, y)$ and $f(q_l, y)$
  - $f(\alpha, y)$ has *non-rational coefficients* and *multiple roots*
- **Connect** arcs to given vertices (skipped today)

Crux in **Lifting**. CGAL's solution: [Eigenwillig, K., Wolpert 2007]

- non-rational coefficients handled by "interval-Descartes"
- requires to know #distinct (real and complex) roots
- needs full subresultant sequence: expensive to compute
- more than one multiple root: expensive shearing

Our efficient symbolic-numerical algorithm

- Needs only GCDs and resultants as symbolic operations

Projection:

- GPU-resultants are up to 500x faster [Emeliyanenko 2010, 2011]
- RS for real root isolation for $p \in \mathbb{Z}[x]$ [Rouillier, Zimmermann 2004]

Lifting:

- First try new filter FASTLIFT based on numerical root solver
- If it rarely fails, apply complete LIFT based on BISOLVE [BES 2011]

Connect:

- Straight-forward using interval arithmetic

- No shearing needed; output given in the original coordinate system; enables fast lexicographic comparisons
- Beats CGAL's previous implementation also if none is GPU-supported

Our efficient symbolic-numerical algorithm

- Needs only GCDs and resultants as symbolic operations

Projection:

- GPU-resultants are up to 500x faster [Emeliyanenko 2010, 2011]
- RS for real root isolation for $p \in \mathbb{Z}[x]$ [Rouillier, Zimmermann 2004]

Lifting:

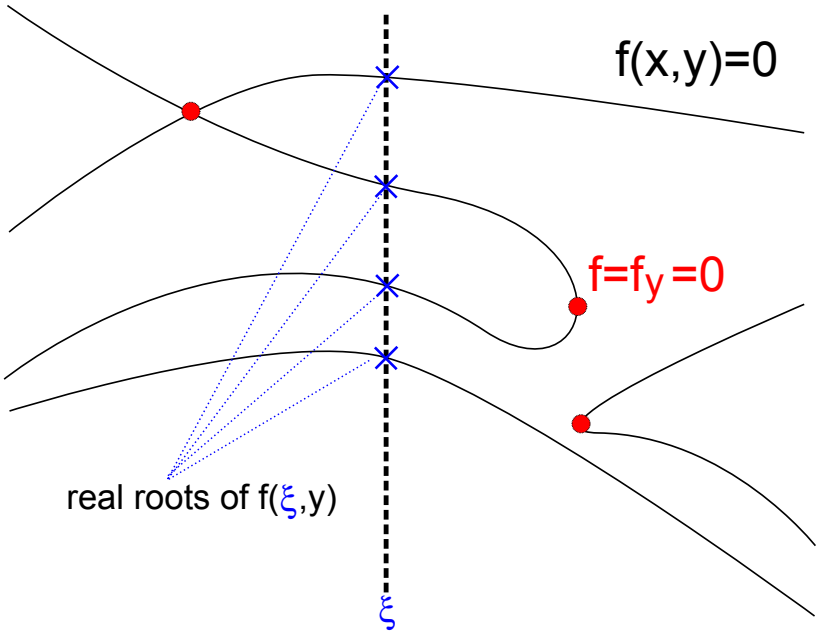
- First try new filter FASTLIFT based on numerical root solver
- If it rarely fails, apply complete LIFT based on BISOLVE [BES 2011]

Connect:

- Straight-forward using interval arithmetic

- No shearing needed; output given in the original coordinate system; enables fast lexicographic comparisons
- Beats CGAL's previous implementation also if none is GPU-supported

FastLift – a hybrid method



For fiber at α we

- Determine upper bound *U* on #distinct roots *m* of $f(\alpha, y)$
- Increase lower bound *L* on #distinct roots by running iterations of numerical and certified root isolator
- Aim: Match both

FastLift – Step 1: upper bound U on #distinct roots

Lemma

[Teissier 1973] For an x-critical point $p = (\alpha, \beta)$ of $C = V(f)$:

$$\text{mult}(f(\alpha, y), \beta) = \text{Int}(f, f_y, p) - \text{Int}(f_x, f_y, p) + 1 \qquad (1)$$

- $\text{mult}(f(\alpha, y), \beta)$: multiplicity of β as root of $f(\alpha, y) \in \mathbb{R}[y]$
- $\text{Int}(f, f_y, p)$, $\text{Int}(f_x, f_y, p)$: pairwise intersection multiplicities of the curves $f = 0$, $f_x = 0$ and $f_y = 0$ at p, respectively.

Let $R_1 := \text{res}(f, f_y; y)$ and $R_2 := \text{res}(f_x, f_y; y)$ with $R_1, R_2 \in \mathbb{Z}[x]$.

FastLift – Step 1: upper bound U on #distinct roots

$$
\begin{aligned}
m &= \#\{\text{distinct complex roots of } f(\alpha, y)\} \\
&= \deg_y f - \deg \gcd(f(\alpha, y), f_y(\alpha, y)) \\
&= \deg_y f - \sum_{\beta \in \mathbb{C}: f(\alpha,\beta)=0} (\text{mult}(f(\alpha, y), \beta) - 1) \\
&= \deg_y f - \sum_{\substack{\beta \in \mathbb{C}: \\ (\alpha,\beta) \text{ is } x\text{-critical}}} (\text{Int}(f, f_y, (\alpha, \beta)) - \text{Int}(f_x, f_y, (\alpha, \beta))) \\
&= \deg_y f - \text{mult}(R_1, \alpha) + \sum_{\substack{\beta \in \mathbb{C}: \\ (\alpha,\beta) \text{ is } x\text{-critical}}} \text{Int}(f_x, f_y, (\alpha, \beta)) \qquad (2) \\
&\leq \deg_y f - \text{mult}(R_1, \alpha) + \sum_{\beta \in \mathbb{C}} \text{Int}(f_x, f_y, (\alpha, \beta)) \qquad (3) \\
&= \deg_y f - \text{mult}(R_1, \alpha) + \text{mult}(R_2, \alpha) =: U \qquad (4)
\end{aligned}
$$

FastLift – Step 1: upper bound U on #distinct roots

$$
\begin{aligned}
m &= \#\{\text{distinct complex roots of } f(\alpha, y)\} \\
&= \deg_y f - \deg \gcd(f(\alpha, y), f_y(\alpha, y)) \\
&= \deg_y f - \sum_{\beta \in \mathbb{C}: f(\alpha,\beta)=0} (\text{mult}(f(\alpha, y), \beta) - 1) \\
&= \deg_y f - \sum_{\substack{\beta \in \mathbb{C}: \\ (\alpha,\beta) \text{ is } x\text{-critical}}} (\text{Int}(f, f_y, (\alpha, \beta)) - \text{Int}(f_x, f_y, (\alpha, \beta))) \\
&= \deg_y f - \text{mult}(R_1, \alpha) + \sum_{\substack{\beta \in \mathbb{C}: \\ (\alpha,\beta) \text{ is } x\text{-critical}}} \text{Int}(f_x, f_y, (\alpha, \beta)) \qquad (2) \\
&\leq \deg_y f - \text{mult}(R_1, \alpha) + \sum_{\beta \in \mathbb{C}} \text{Int}(f_x, f_y, (\alpha, \beta)) \qquad (3) \\
&= \deg_y f - \text{mult}(R_1, \alpha) + \text{mult}(R_2, \alpha) =: U \qquad (4)
\end{aligned}
$$

FastLift – Step 1: upper bound U on #distinct roots

$$
\begin{aligned}
m &= \#\{\text{distinct complex roots of } f(\alpha, y)\} \\
&= \deg_y f - \deg \gcd(f(\alpha, y), f_y(\alpha, y)) \\
&= \deg_y f - \sum_{\beta \in \mathbb{C}: f(\alpha,\beta)=0} (\text{mult}(f(\alpha, y), \beta) - 1) \\
&= \deg_y f - \sum_{\substack{\beta \in \mathbb{C}: \\ (\alpha,\beta) \text{ is } x\text{-critical}}} (\text{Int}(f, f_y, (\alpha, \beta)) - \text{Int}(f_x, f_y, (\alpha, \beta))) \\
&= \deg_y f - \text{mult}(R_1, \alpha) + \sum_{\substack{\beta \in \mathbb{C}: \\ (\alpha,\beta) \text{ is } x\text{-critical}}} \text{Int}(f_x, f_y, (\alpha, \beta)) \qquad (2) \\
&= \deg_y f - \text{mult}(R_1, \alpha) + \sum_{\beta \in \mathbb{C}} \text{Int}(f_x, f_y, (\alpha, \beta)) \qquad (3) \\
&= \deg_y f - \text{mult}(R_1, \alpha) + \text{mult}(R_2, \alpha) =: U \qquad (4)
\end{aligned}
$$

Most likely – as we will see

FastLift – Step 2: run numerical solver

Run modified Aberth-Ehrlich method on $f(\alpha, y)$

In each iteration (with increasing precision)

- Solver reports L discs $D_1, \ldots, D_L \subset \mathbb{C}$
  - Discs contain all (real and complex) roots
  - D_i contains at least one root
  - D_i is not neccessarily isolating for contained roots
- $L \leq m \leq U := \deg f - \mathrm{mult}(\alpha, R_1) + \mathrm{mult}(\alpha, R_2)$
- If $L = U$, we know that all roots are isolated; refine discs that complex conjugates do not intersect real axis

If $m < U$, process does not terminate!

When reaching a certain precision, stop FastLift, start Lift; local for current α

FastLift – Step 2: run numerical solver

Run modified Aberth-Ehrlich method on $f(\alpha, y)$

In each iteration (with increasing precision)

- Solver reports L discs $D_1, \ldots, D_L \subset \mathbb{C}$
  - Discs contain all (real and complex) roots
  - D_i contains at least one root
  - D_i is not neccessarily isolating for contained roots
- $L \leq m \leq U := \deg f - \text{mult}(\alpha, R_1) + \text{mult}(\alpha, R_2)$
- If $L = U$, we know that all roots are isolated;
  refine discs that complex conjugates do not intersect real axis

If $m < U$, process does not terminate!

When reaching a certain precision, stop FastLift, start Lift; local for current α

FastLift – Step 2: run numerical solver

Run modified Aberth-Ehrlich method on $f(\alpha, y)$

In each iteration (with increasing precision)

- Solver reports L discs $D_1, \ldots, D_L \subset \mathbb{C}$
  - Discs contain all (real and complex) roots
  - D_i contains at least one root
  - D_i is not neccessarily isolating for contained roots
- $L \leq m \leq U := \deg f - \text{mult}(\alpha, R_1) + \text{mult}(\alpha, R_2)$
- If $L = U$, we know that all roots are isolated; refine discs that complex conjugates do not intersect real axis

If $m < U$, process does not terminate!

When reaching a certain precision, stop FastLift, start Lift; local for current α

FastLift – Step 2: run numerical solver

Run modified Aberth-Ehrlich method on $f(\alpha, y)$

In each iteration (with increasing precision)

- Solver reports L discs $D_1, \ldots, D_L \subset \mathbb{C}$
  - Discs contain all (real and complex) roots
  - D_i contains at least one root
  - D_i is not neccessarily isolating for contained roots
- $L \leq m \leq U := \deg f - \text{mult}(\alpha, R_1) + \text{mult}(\alpha, R_2)$
- If $L = U$, we know that all roots are isolated; refine discs that complex conjugates do not intersect real axis

If $m < U$, process does not terminate!

When reaching a certain precision, stop FastLift, start Lift; local for current α

FastLift – Failures are rare

At the x-critical α

- (Geo 1): $V(f)$ has no vertical asymptote or, if so, $f(\alpha, y)$ is square-free
- (Geo 2): For all $p = (\alpha, \beta)$ with $p \in V(f_x) \cap V(f_y)$, it holds $p \in V(f)$
- (Num 1): Enough iterations to achieve $L = m$
- (Num 2): Convergence of D_i to roots with increasing precision and number of iterations

FastLift – Failures are rare

At the x-critical α

- (Geo 1): $V(f)$ has no vertical asymptote or, if so, $f(\alpha, y)$ is square-free
- (Geo 2): For all $p = (\alpha, \beta)$ with $p \in V(f_x) \cap V(f_y)$, it holds $p \in V(f)$
- (Num 1): Enough iterations to achieve $L = m$
- (Num 2): Convergence of D_i to roots with increasing precision and number of iterations

FastLift – Failures are rare

At the x-critical α

- (Geo 1): $V(f)$ has no vertical asymptote or, if so, $f(\alpha, y)$ is square-free
- (Geo 2): For all $p = (\alpha, \beta)$ with $p \in V(f_x) \cap V(f_y)$, it holds $p \in V(f)$
  Both are given in almost all cases
- (Num 1): Enough iterations to achieve $L = m$
  Practical problem; dependence on degree and bitsize of input
- (Num 2): Convergence of D_i to roots with increasing precision and number of iterations
  Theoretical proof is lacking; in practice always observed.

FastLift – Failures are rare

At the x-critical α

- (Geo 1): $V(f)$ has no vertical asymptote or, if so, $f(\alpha, y)$ is square-free
- (Geo 2): For all $p = (\alpha, \beta)$ with $p \in V(f_x) \cap V(f_y)$, it holds $p \in V(f)$

- (Num 1): Enough iterations to achieve $L = m$
- (Num 2): Convergence of D_i to roots with increasing precision and number of iterations

- Call Lift only for fibers where FastList fails
- FastLift lifts most fibers extremely fast
- Though it additionally requires $\mathrm{res}(f_x, f_y; y)$ (on GPU)
- For curves in generic position Geo (1) and Geo (2) are given, i.e., FastLift is practically complete.

FastLift – Failures are rare

At the x-critical α

- (Geo 1): $V(f)$ has no vertical asymptote or, if so, $f(\alpha, y)$ is square-free
- (Geo 2): For all $p = (\alpha, \beta)$ with $p \in V(f_x) \cap V(f_y)$, it holds $p \in V(f)$

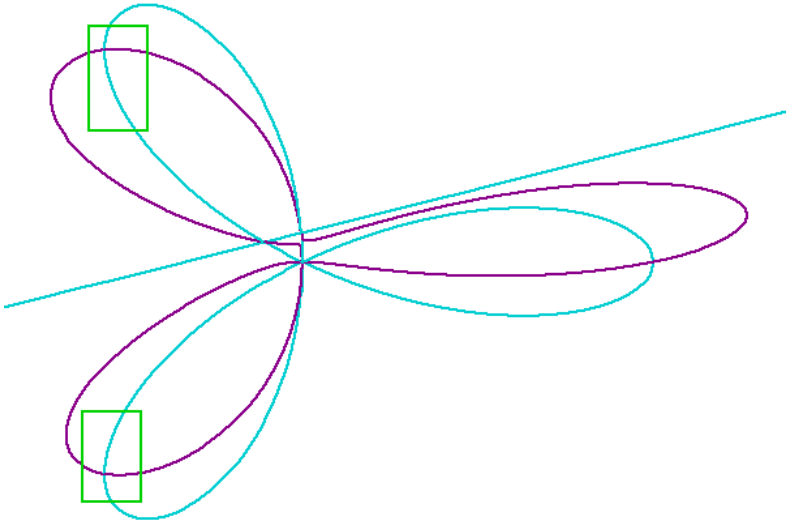
- (Num 1): Enough iterations to achieve $L = m$
- (Num 2): Convergence of D_i to roots with increasing precision and number of iterations

- Call Lift only for fibers where FastList fails
- FastLift lifts most fibers extremely fast
- Though it additionally requires $\text{res}(f_x, f_y; y)$ (on GPU)
- For curves in generic position Geo (1) and Geo (2) are given, i.e., FastLift is practically complete.

LIFT – a complete method for lifting with BISOLVE

BISOLVE [ALENEX 2011]

- Fastly isolating boxes for $g_1(x,y) = g_2(x,y) = 0$
- No shearing
- Needs only (GPUable) GCDs and resultants



- Determine **multiple** roots β_i of $f(\alpha, y)$ with multiplicities k_i by running BISOLVE **locally** on

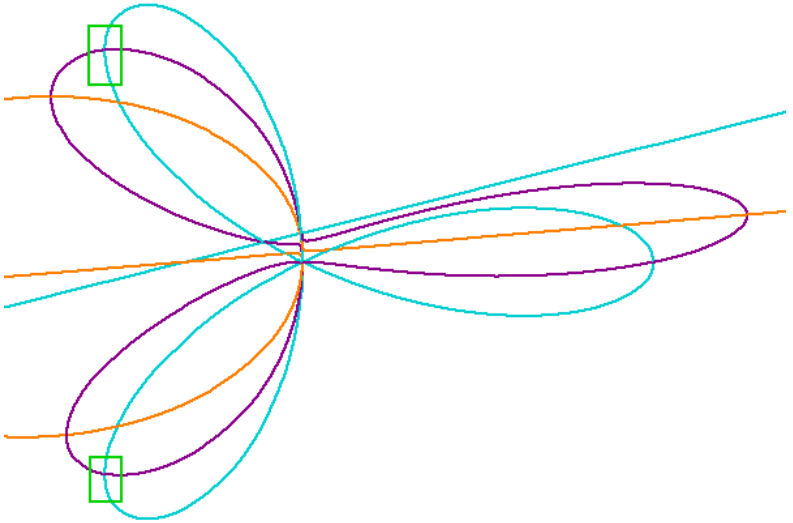
$$\{f = f_y = 0\}|_{x=\alpha}, \{f_y = f_{yy} = 0\}|_{x=\alpha}, \{f_{yy} = f_{yyy} = 0\}|_{x=\alpha}, \ldots$$

- Refine box containing (α, β_i) with IA to avoid $\frac{\partial f}{\partial y^{k_i}}$
- Run "interval Descartes" to isolate **ordinary roots** of $f(\alpha, y)$ — representing "arcs running between critical points"

LIFT – a complete method for lifting with BISOLVE

BISOLVE [ALENEX 2011]

- Fastly isolating boxes for $g_1(x,y) = g_2(x,y) = 0$
- No shearing
- Needs only (GPUable) GCDs and resultants



- Determine **multiple** roots β_i of $f(\alpha, y)$ with multiplicities k_i by running BISOLVE **locally** on

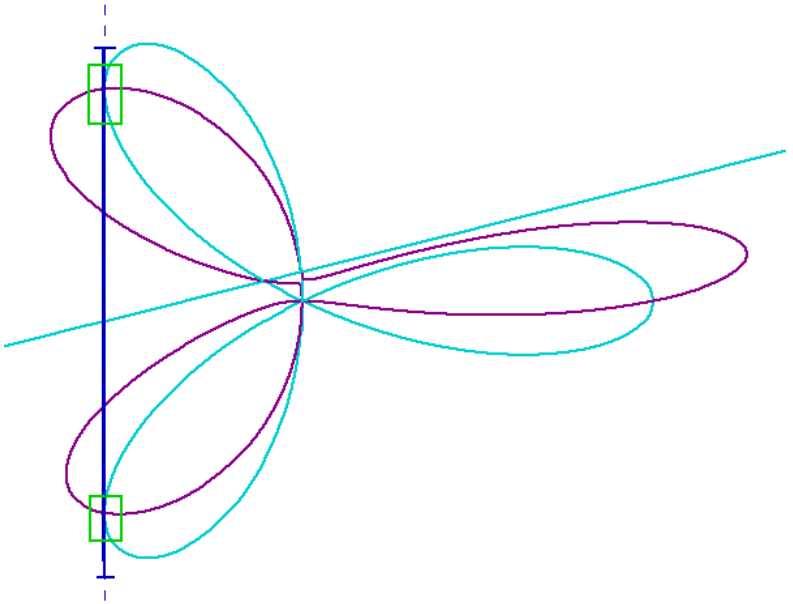
$$\{f = f_y = 0\}|_{x=\alpha}, \{f_y = f_{yy} = 0\}|_{x=\alpha}, \{f_{yy} = f_{yyy} = 0\}|_{x=\alpha}, \ldots$$

- Refine box containing (α, β_i) with IA to avoid $\frac{\partial f}{\partial y^{k_i}}$
- Run "interval Descartes" to isolate **ordinary roots** of $f(\alpha, y)$ — representing "arcs running between critical points"

LIFT – a complete method for lifting with BISOLVE

BISOLVE [ALENEX 2011]

- Fastly isolating boxes for $g_1(x,y) = g_2(x,y) = 0$
- No shearing
- Needs only (GPUable) GCDs and resultants



- Determine **multiple** roots β_i of $f(\alpha, y)$ with multiplicities k_i by running BISOLVE **locally** on

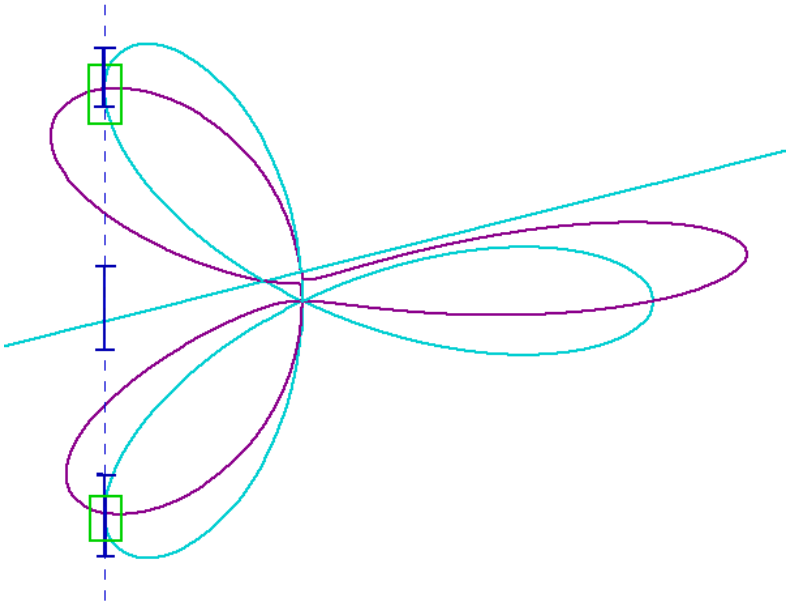
$$\{f = f_y = 0\}|_{x=\alpha}, \{f_y = f_{yy} = 0\}|_{x=\alpha}, \{f_{yy} = f_{yyy} = 0\}|_{x=\alpha}, \ldots$$

- Refine box containing (α, β_i) with IA to avoid $\frac{\partial f}{\partial y^{k_i}}$
- Run "interval Descartes" to isolate **ordinary roots** of $f(\alpha, y)$ — representing "arcs running between critical points"

LIFT – a complete method for lifting with BISOLVE

BISOLVE [ALENEX 2011]

- Fastly isolating boxes for $g_1(x,y) = g_2(x,y) = 0$
- No shearing
- Needs only (GPUable) GCDs and resultants



- Determine **multiple** roots β_i of $f(\alpha, y)$ with multiplicities k_i by running BISOLVE **locally** on

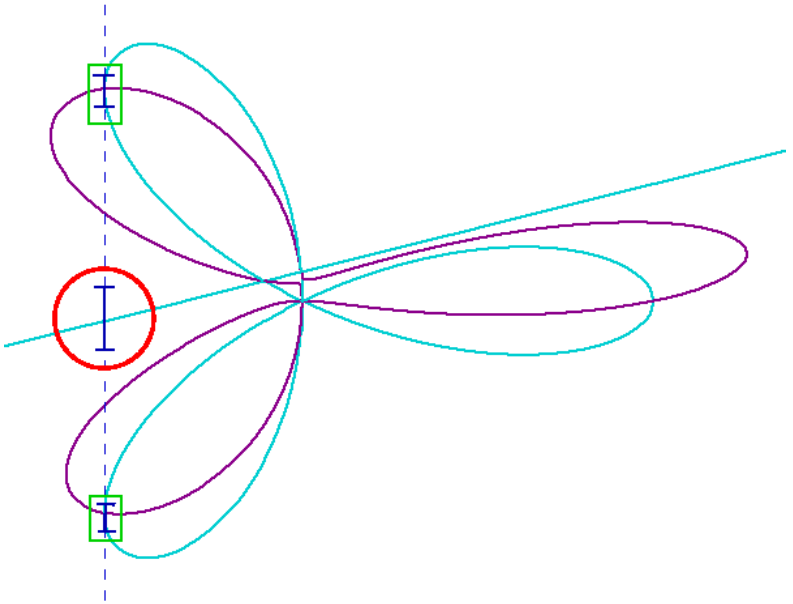
$$\{f = f_y = 0\}|_{x=\alpha}, \{f_y = f_{yy} = 0\}|_{x=\alpha}, \{f_{yy} = f_{yyy} = 0\}|_{x=\alpha}, \ldots$$

- Refine box containing (α, β_i) with IA to avoid $\frac{\partial f}{\partial y^{k_i}}$
- Run "interval Descartes" to isolate **ordinary roots** of $f(\alpha, y)$ — representing "arcs running between critical points"

LIFT – a complete method for lifting with BISOLVE

BISOLVE [ALENEX 2011]

- Fastly isolating boxes for $g_1(x, y) = g_2(x, y) = 0$
- No shearing
- Needs only (GPUable) GCDs and resultants



- Determine **multiple** roots β_i of $f(\alpha, y)$ with multiplicities k_i by running BISOLVE **locally** on

$$\{f = f_y = 0\}|_{x=\alpha}, \{f_y = f_{yy} = 0\}|_{x=\alpha}, \{f_{yy} = f_{yyy} = 0\}|_{x=\alpha}, \ldots$$

- Refine box containing (α, β_i) with IA to avoid $\frac{\partial f}{\partial y^{k_i}}$
- Run "interval Descartes" to isolate **ordinary roots** of $f(\alpha, y)$ — representing "arcs running between critical points"

Experiments curve analysis

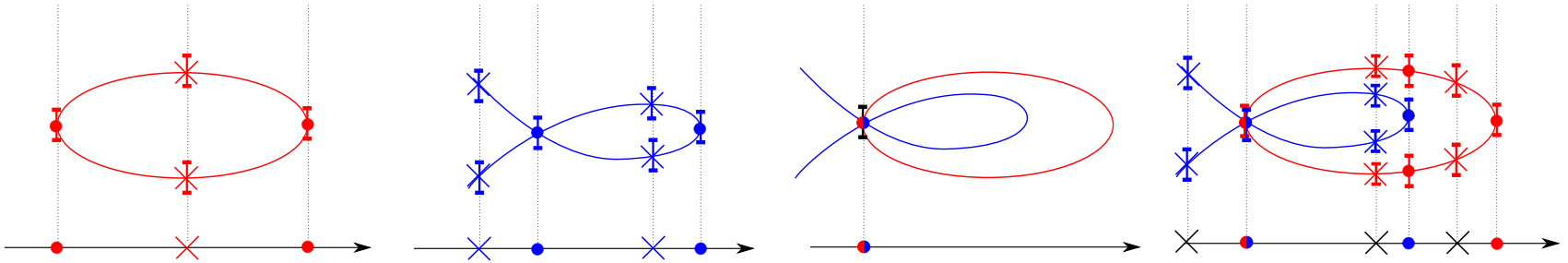
Table : Running times (in sec) for analyses of algebraic curves of various families; **timeout:** algorithm timed out ($>$ 400 sec)

type	BISOLVE	LIFTANA	FASTANA	AK_2*
rnd d, 9, 10	0.83	0.73	**0.24**	0.66
rnd d, 15, 2048	20.48	50.06	**13.31**	16.82
rnd s, 9, 10	0.25	0.33	**0.11**	1.00
rnd s, 15, 2048	15.66	32.15	**8.08**	24.32
trans, 9	14.78	14.91	**2.97**	159.22
project, 4·4	10.01	11.93	**2.24**	38.91
SA_2_4_eps	0.35	2.18	**0.64**	73.00
L6_circles	4.52	92.24	**2.10**	224.19
swinnerton	19.61	16.81	**7.12**	304.09
degree_7_surf	14.57	48.94	**7.39**	timeout
challenge_12b	16.48	52.71	**44.30**	timeout

* uses GPU for resultant

Curve-pair analysis via projection

"For each x, determine the vertical alignment of curves"

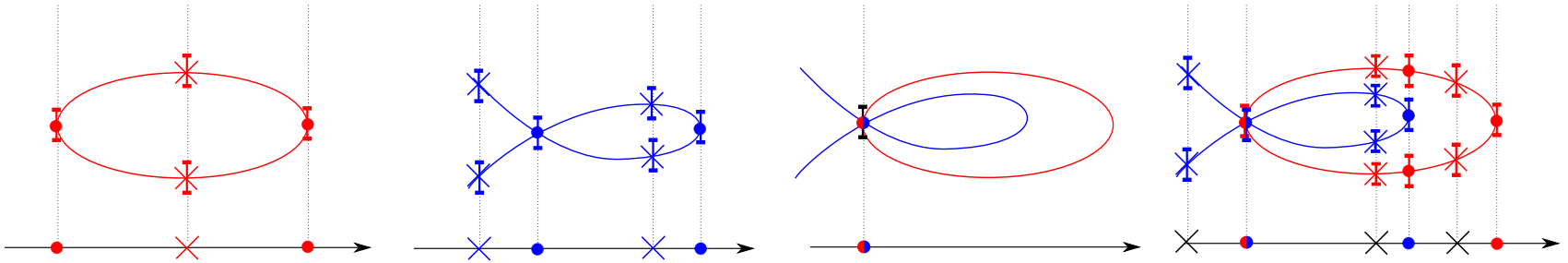


Principal subtasks:

- *Determine* critical points of both curves and their intersections and *sort* them *lexicographically*:
- Achieved by single curve analyses and BISOLVE
- Plus real root isolation of univariate polynomials and comparison of distinct solutions
- Actually: Algorithm is the one shipped with CGAL, except that BISOLVE replaces uses of subresultants and shearing.

Curve-pair analysis via projection

"For each x, determine the vertical alignment of curves"



Principal subtasks:

- *Determine* critical points of both curves and their intersections and *sort* them *lexicographically*:
- Achieved by single curve analyses and BISOLVE
- Plus real root isolation of univariate polynomials and comparison of distinct solutions
- Actually: Algorithm is the one shipped with CGAL, except that BISOLVE replaces uses of subresultants and shearing.

Experiments arrangements

Table : Running times (in sec) for arrangements of algebraic curves;
timeout: algorithm timed out ($>$ 4000 sec)

(P) number of projected surface intersections		
#resultants	FastKernel	Ak_2*
2	**0.21**	0.49
4	**1.03**	1.64
6	**5.14**	7.84
8	**22.69**	35.77
10	**58.37**	91.84

(X) combinations of special curves		
#curves	FastKernel	Ak_2*
2	**9.2**	81.93
4	**248.87**	730.57
6	**689.39**	3030.27
8	**1129.98**	timeout
10	**1201.34**	timeout

* uses GPU for resultant

Achievements

Algorithms for curve- and curve-pair analyses that

- Reduce symbolic operations: GCDs and resultants.
  Perform both on the GPU
- Successfully apply symbolic-numerical filters
  Strategy: “Use certified approximations whenever possible”
- Never shear
- Algorithms are implemented
- Tremendous speed-ups for curve analyses
- Improvements on curve-pair analyses in non-generic case

Not the end ...

- Fine-tune implementation, e.g., better factorizing of elimination polynomial, more caching, numerical filter for CPA
- Comparison with other approaches (e.g., RS3)
- Combine with subdivision methods to make them certified and complete
- Use in surface analysis and enhance its lifting in similar way