

DCGI

DEPARTMENT OF COMPUTER GRAPHICS AND INTERACTION

COMPUTATIONAL GEOMETRY INTRODUCTION

PETR FELKEL

FEL CTU PRAGUE

felkel@fel.cvut.cz

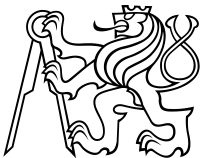
<https://cw.felk.cvut.cz/doku.php/courses/a4m39vg/start>

Based on [Berg] and [Kolingerova]

Version from 24.10.2025

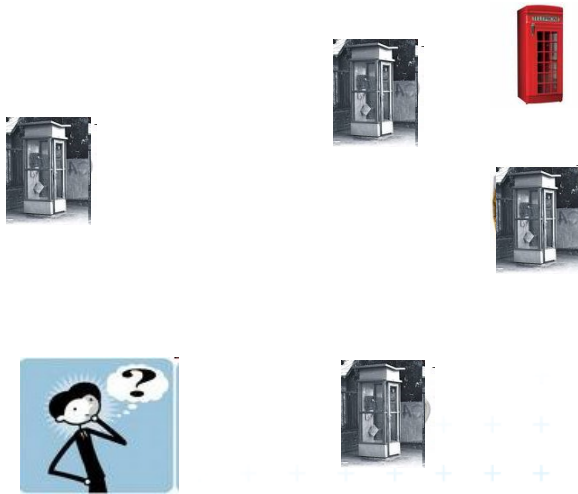
Computational Geometry

1. What is Computational Geometry (CG)?
2. Why to study CG and how?
3. Typical application domains
4. Typical tasks
5. Complexity of algorithms
6. Programming techniques (paradigms) of CG
7. Robustness Issues
8. CGAL – CG algorithm library intro
9. References and resources
10. Course summary



1. What is Computational Geometry?

- CG Solves geometric problems that require clever geometric algorithms
- Ex 1: Where is the nearest phone, metro, pub,...?

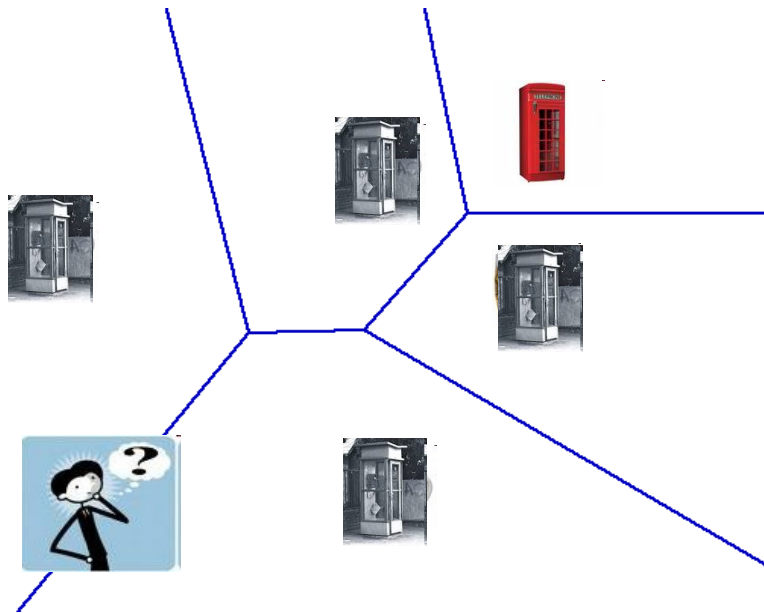


[Berg]

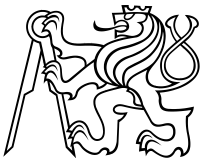


1. What is Computational Geometry?

- CG Solves geometric problems that require clever geometric algorithms
- Ex 1: Where is the nearest phone, metro, pub,...?

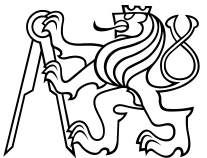
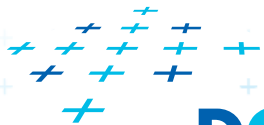
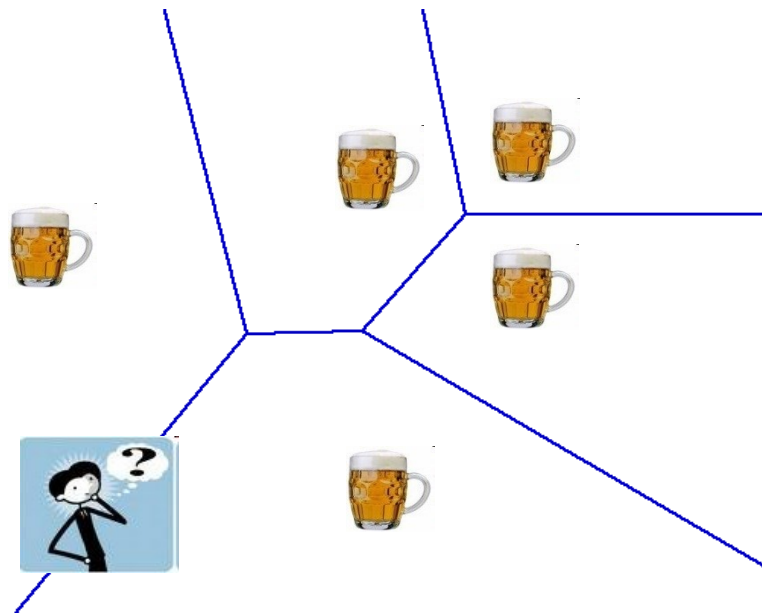


[Berg]



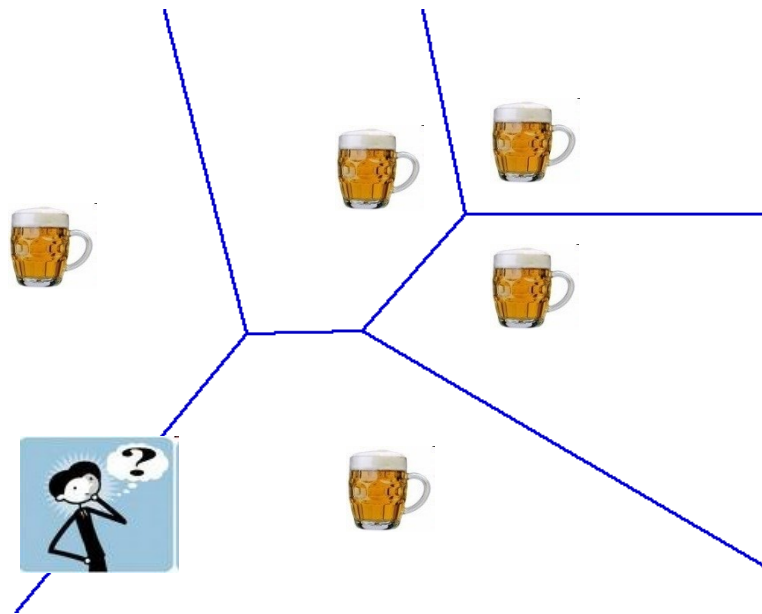
1. What is Computational Geometry?

- CG Solves geometric problems that require clever geometric algorithms
- Ex 1: Where is the nearest phone, metro, pub,...?

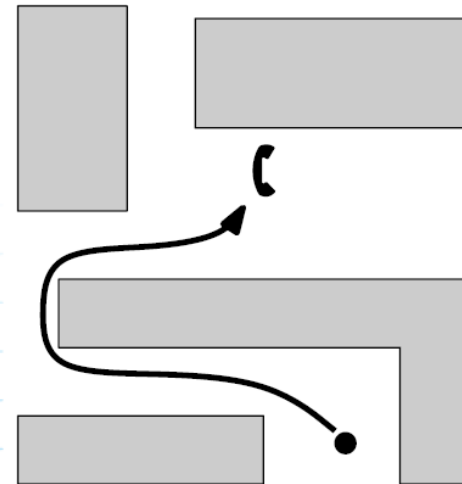


1. What is Computational Geometry?

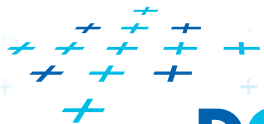
- CG Solves geometric problems that require clever geometric algorithms
- Ex 1: Where is the nearest phone, metro, pub, ...?



Ex 2: How to get there?



[Berg]

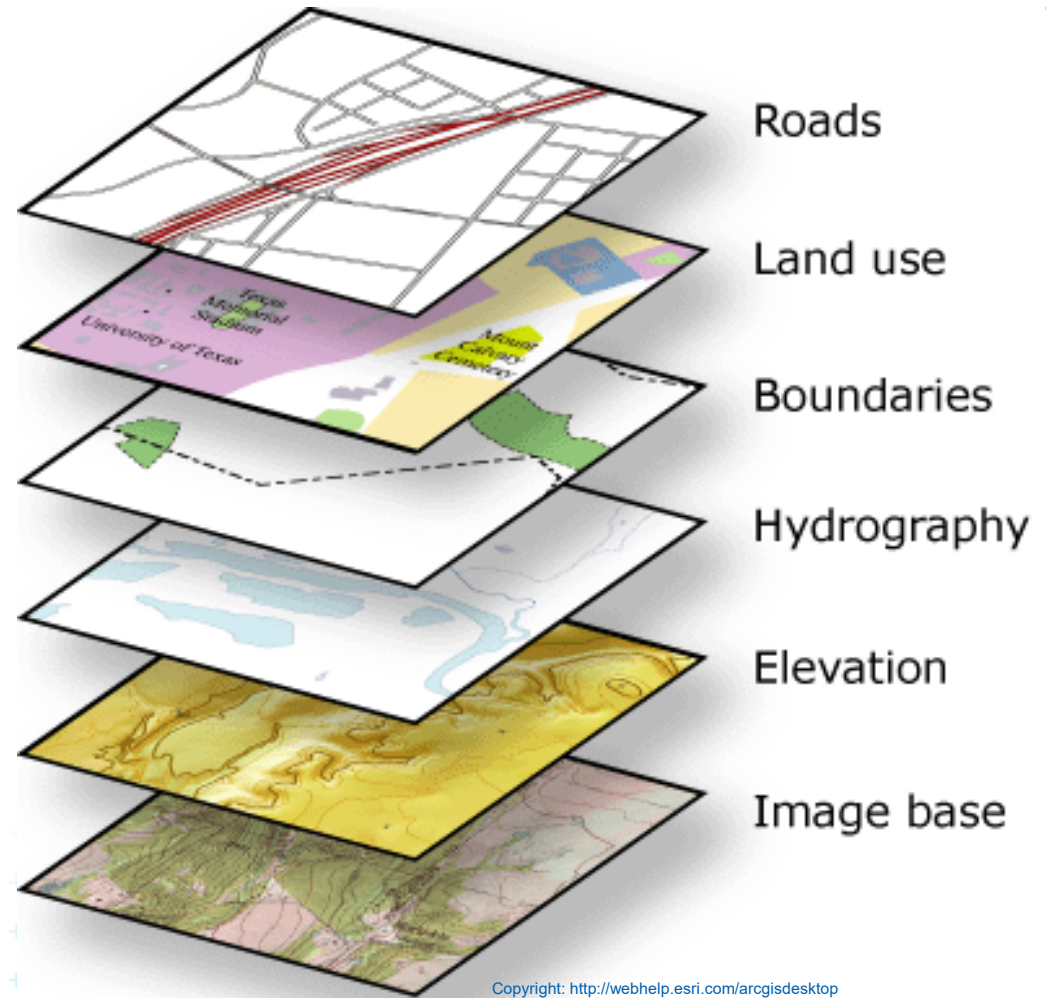


DCGI



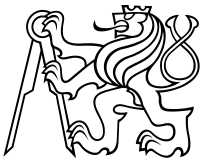
1.1 What is Computational Geometry? (...)

■ Ex 3: Map overlay



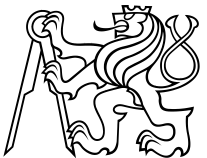
1.2 What is Computational Geometry? (...)

- Good solutions need both:
 - Understanding of the geometric properties of the problem
 - Proper applications of algorithmic techniques (paradigms) and data structures



1.3 What is Computational Geometry? (...)

- Computational geometry – in 1975
= systematic study of algorithms and data structures for geometric objects (points, lines, line segments, n-gons,...)
with focus on exact algorithms that are **asymptotically fast**
 - “Born” in 1975 (Shamos), boom of papers in 90s
(first papers sooner: 1850 Dirichlet, 1908 Voronoi,...)
 - Many problems can be formulated geometrically
(e.g., range queries in databases)



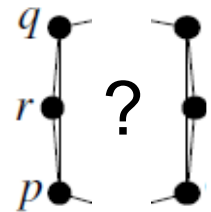
1.4 What is Computational Geometry? (...)

■ Problems:

- Degenerate cases (points on line, with same x, \dots)
 - Ignore them first, include later
- Robustness - correct algorithm but not robust
 - Limited numerical precision of real arithmetic
 - Inconsistent ϵ tests ($a=b, b=c$, but $a \neq c$)

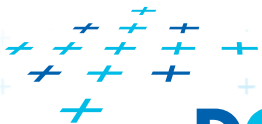


~~exact~~



■ Nowadays:

- focus on **practical implementations**
 - not just on asymptotically fastest algorithms
 - robust
- **nearly correct result** is better than nonsense or crash



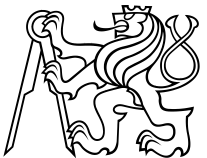
2. Why to study computational geometry?

- Graphics- and Vision-engineer should know it
(„Data structures and algorithms in n^{th} -Dimension“)
 - DSA, PRP, PAL
- Set of ready-to-use tools
- Cool ideas
- You will know new approaches to choose from



2.1 How to teach computational geometry?

- Typical “mathematician” method:
 - definition-theorem-proof
- Our “practical” approach:
 - practical algorithms and their complexity
 - practical programming using a geometric library
- Is it OK for you?



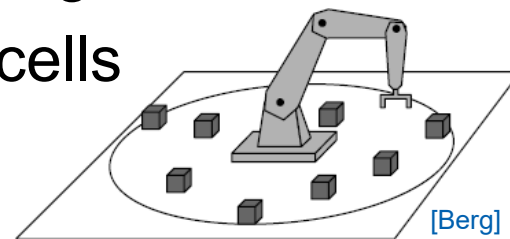
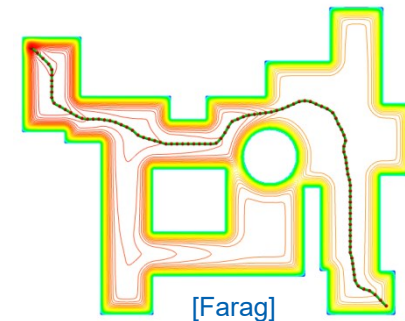
3. Typical application domains

■ Computer graphics

- Collisions of objects
- Mouse localization
- Selection of objects in a region
- Visibility in 3D (hidden surface removal)
- Computation of shadows

■ Robotics

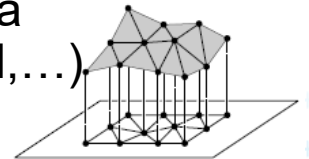
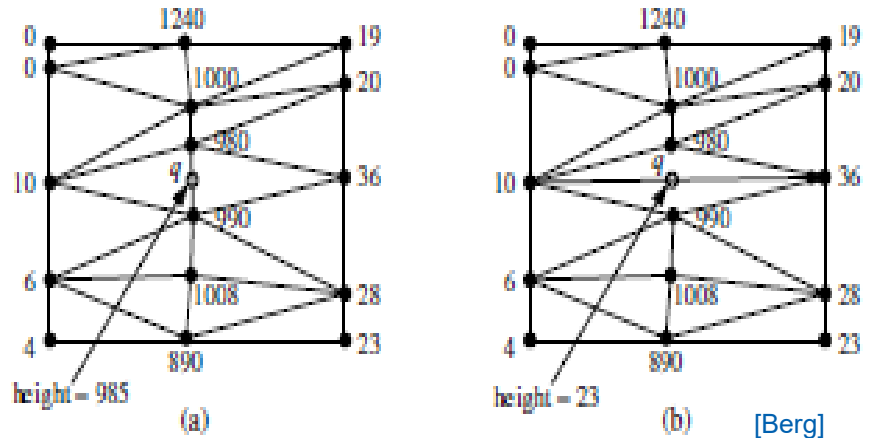
- Motion planning (find path - environment with obstacles)
- Task planning (motion + planning order of subtasks)
- Design of robots and working cells



3.1 Typical application domains (...)

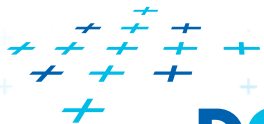
■ GIS

- How to store huge data and search them quickly
- Interpolation of heights
- Overlap of different data
 - Extract information about regions or relations between data (pipes under the construction site, plants x average rainfall,...)
 - Detect bridges on crossings of roads and rivers...



■ CAD/CAM

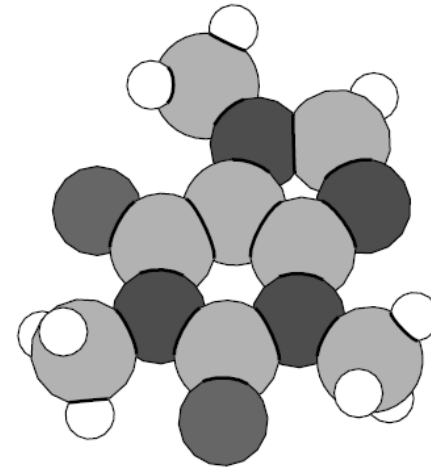
- Intersections and unions of objects
- Visualization and tests without the need to build a prototype
- Manufacturability



3.2 Typical application domains (...)

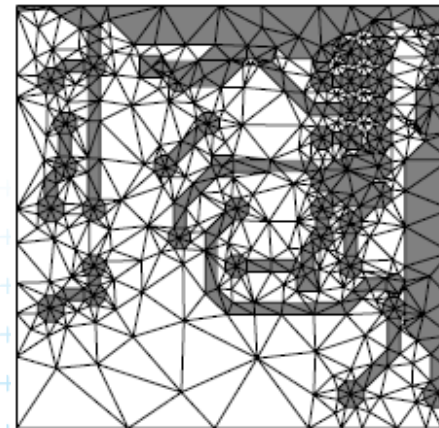
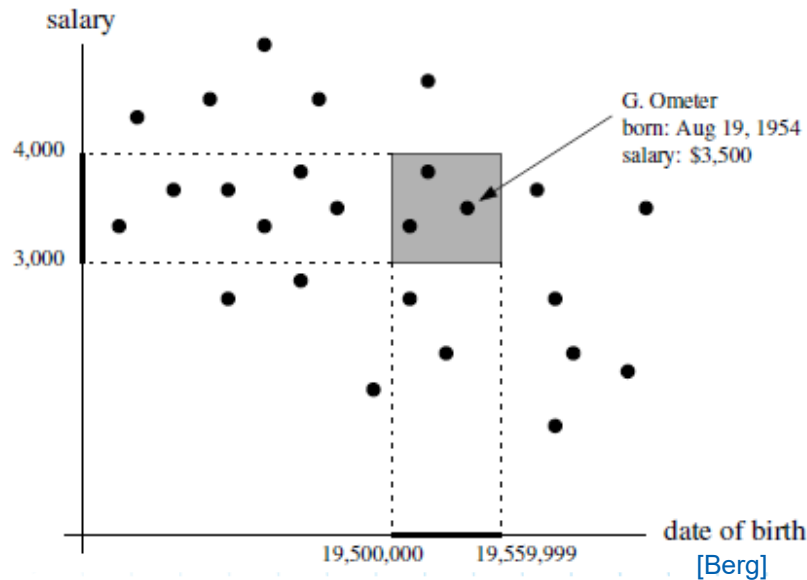
■ Other domains

- Molecular modeling
- DB search
- IC design

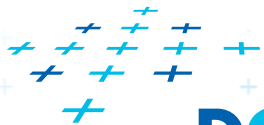


[Berg]

caffeine



[Berg]



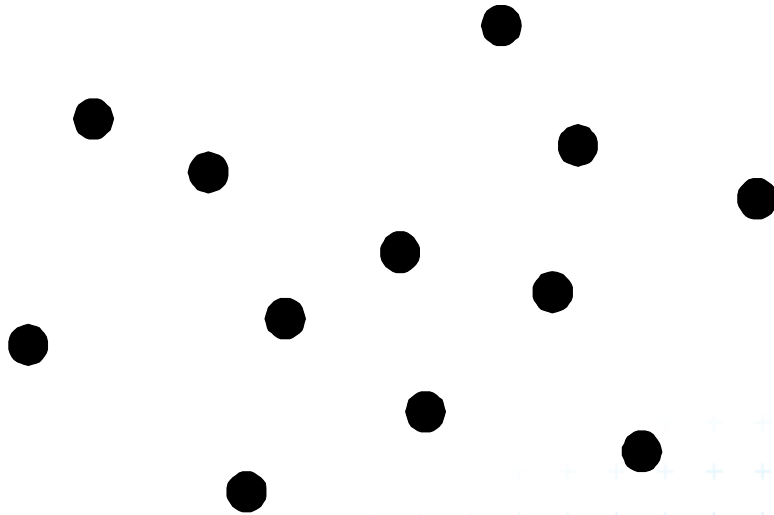
DCGI



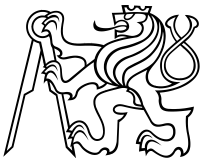
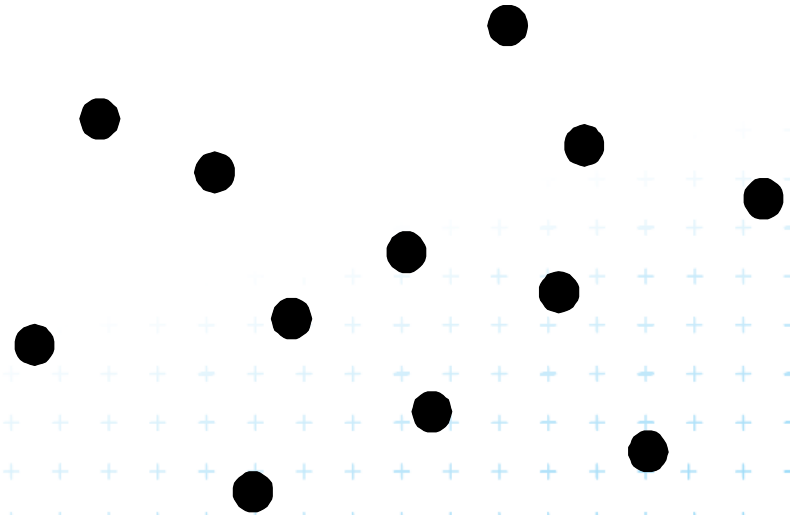
4. Typical tasks in CG

- Geometric searching - fast location of :

The nearest neighbor



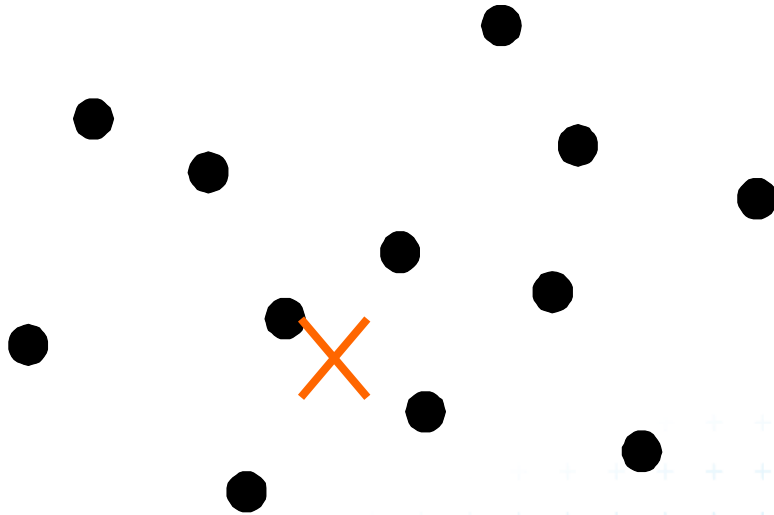
Points in given range (range query)



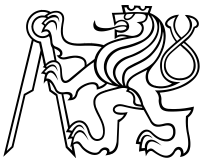
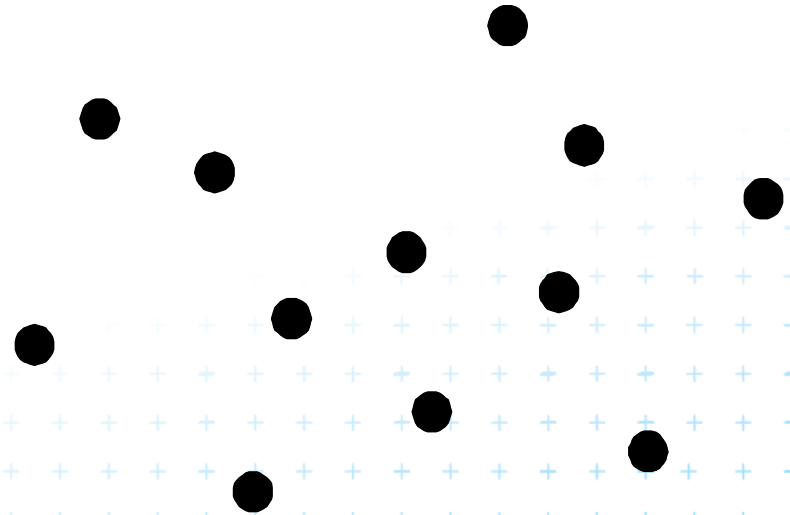
4. Typical tasks in CG

- Geometric searching - fast location of :

The nearest neighbor



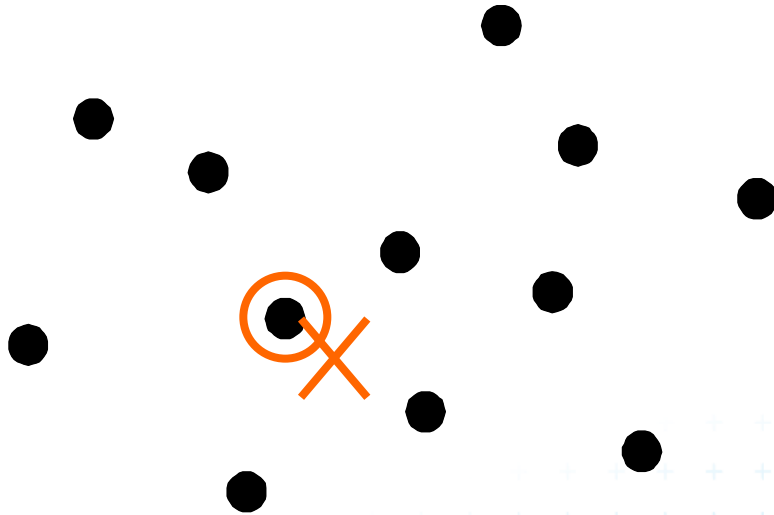
Points in given range (range query)



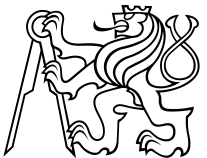
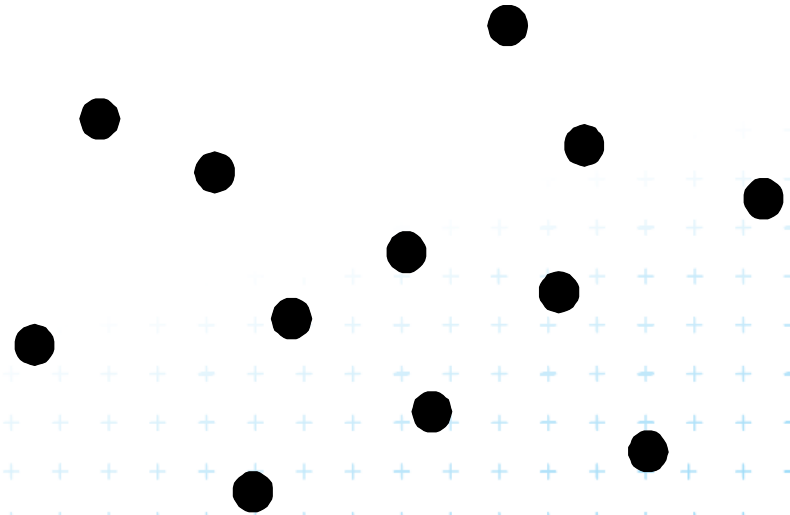
4. Typical tasks in CG

- Geometric searching - fast location of :

The nearest neighbor



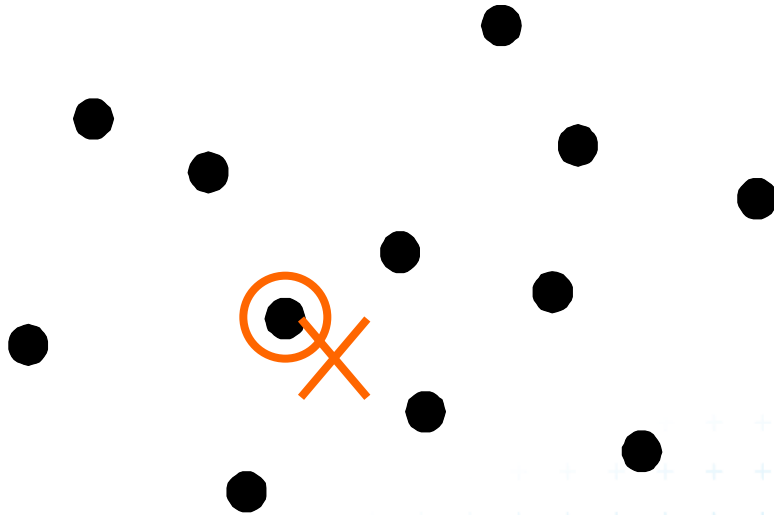
Points in given range (range query)



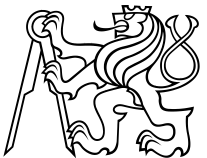
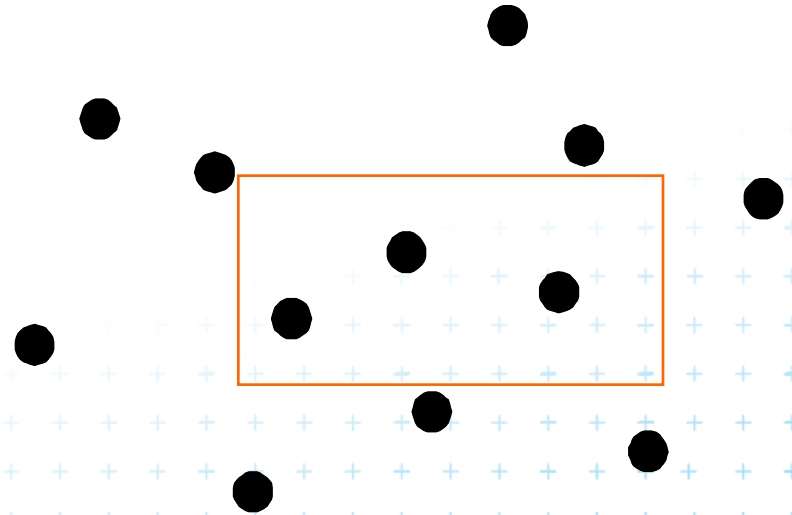
4. Typical tasks in CG

- Geometric searching - fast location of :

The nearest neighbor



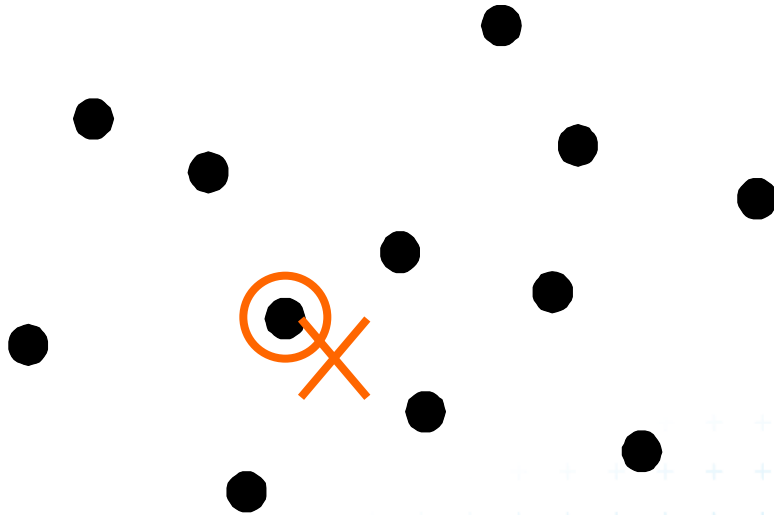
Points in given range (range query)



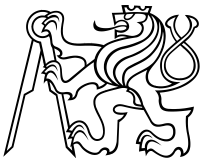
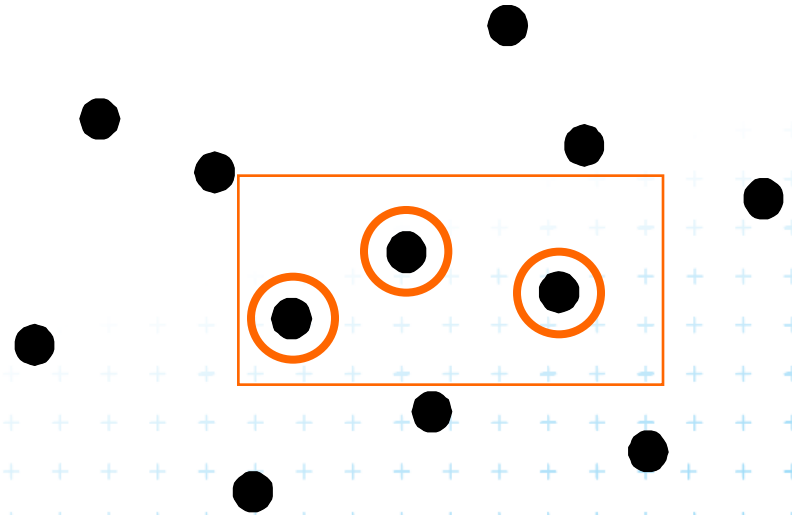
4. Typical tasks in CG

- Geometric searching - fast location of :

The nearest neighbor



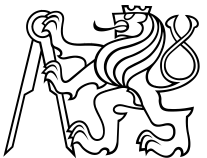
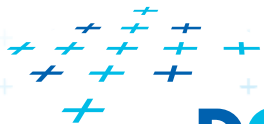
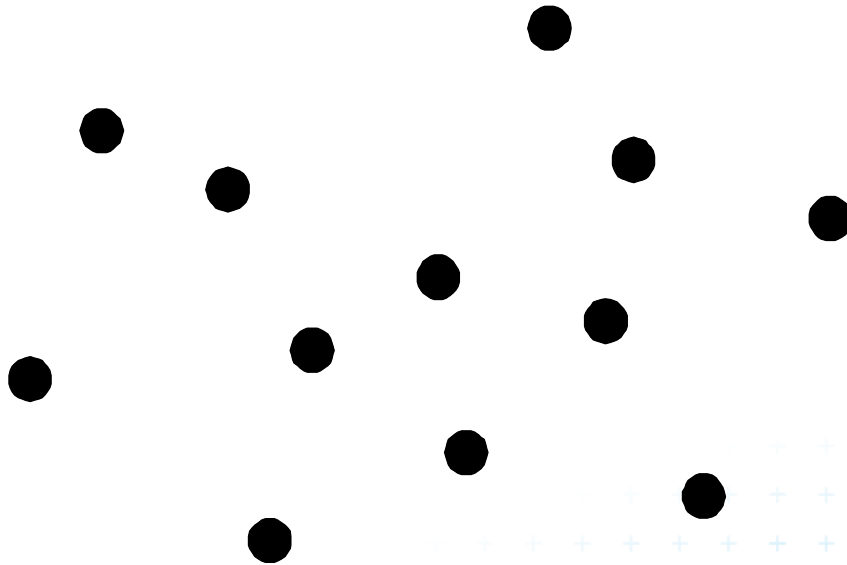
Points in given range (range query)



4.1 Typical tasks in CG

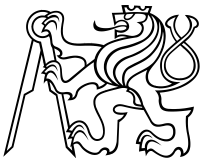
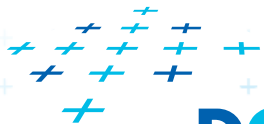
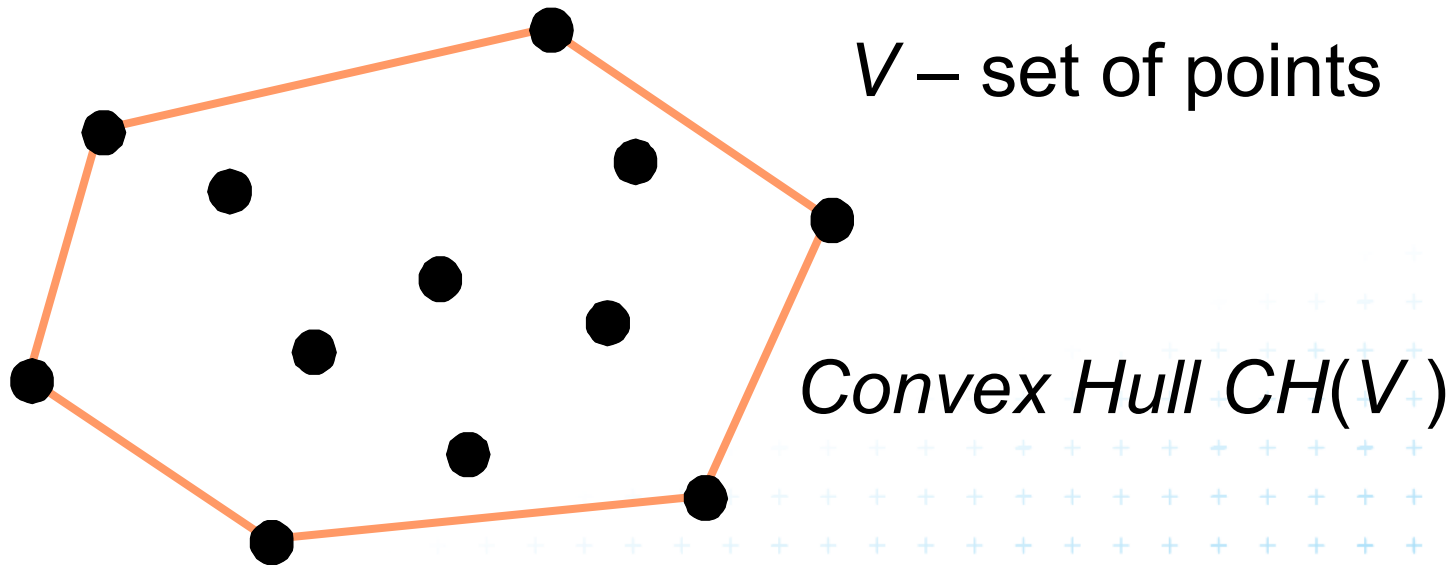
- Convex hull
= smallest enclosing convex polygon in E^2 or
n-gon in E^3 containing all the points

V – set of points



4.1 Typical tasks in CG

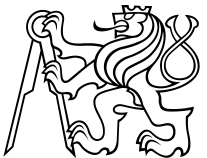
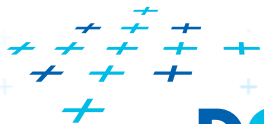
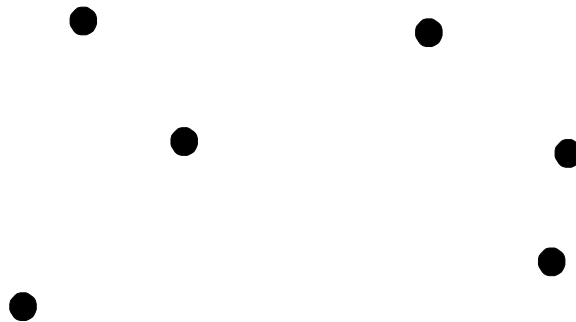
- Convex hull
= smallest enclosing convex polygon in E^2 or
n-gon in E^3 containing all the points



4.2 Typical tasks in CG

■ Voronoi diagrams

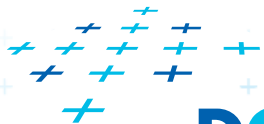
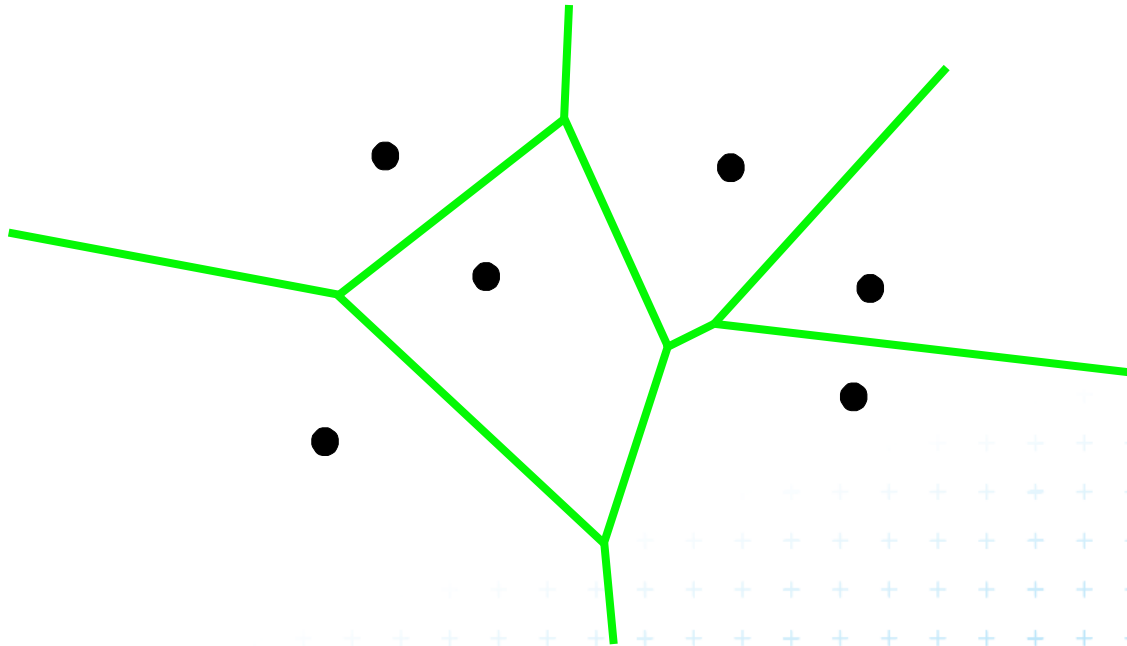
- Space (plane) partitioning into regions whose points are nearest to the given primitive (most usually a point)



4.2 Typical tasks in CG

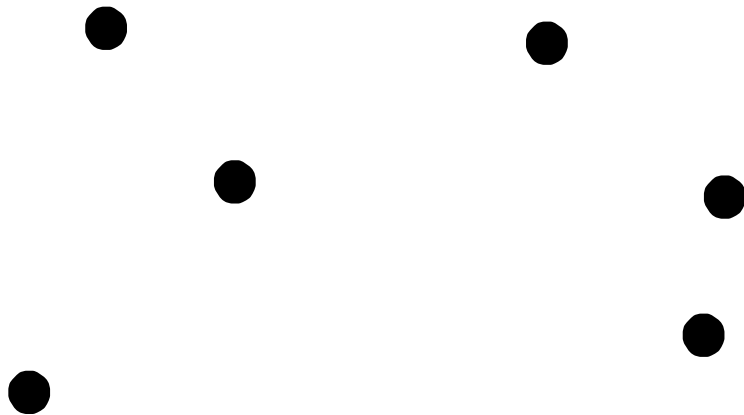
■ Voronoi diagrams

- Space (plane) partitioning into regions whose points are nearest to the given primitive (most usually a point)

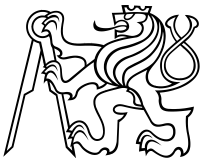


4.3 Typical tasks in CG

- Planar triangulations and space tetrahedronization of given point set

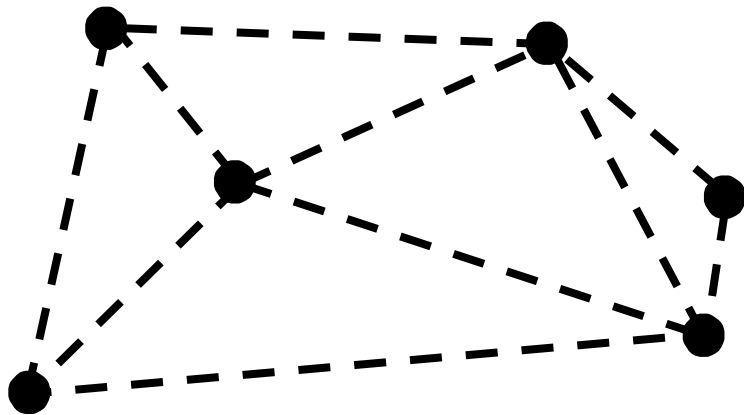


[Maur]

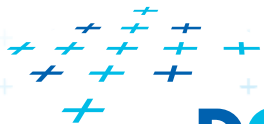


4.3 Typical tasks in CG

- Planar triangulations and space tetrahedronization of given point set



[Maur]

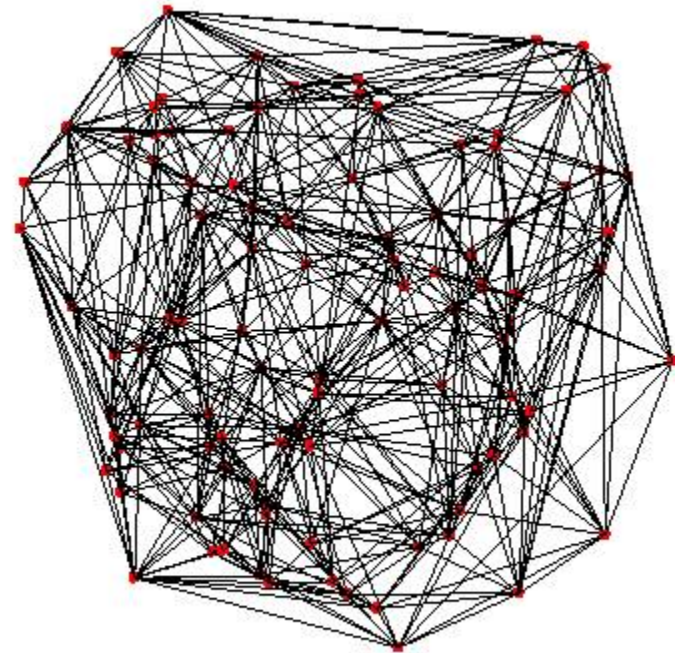
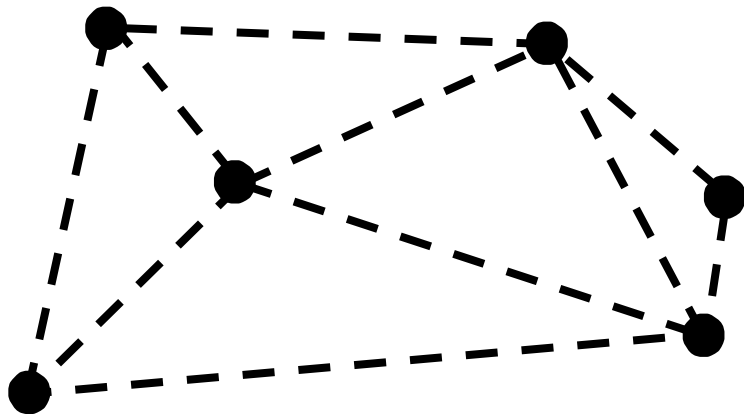


DCGI

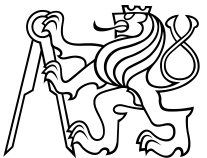


4.3 Typical tasks in CG

- Planar triangulations and space tetrahedronization of given point set



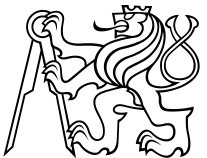
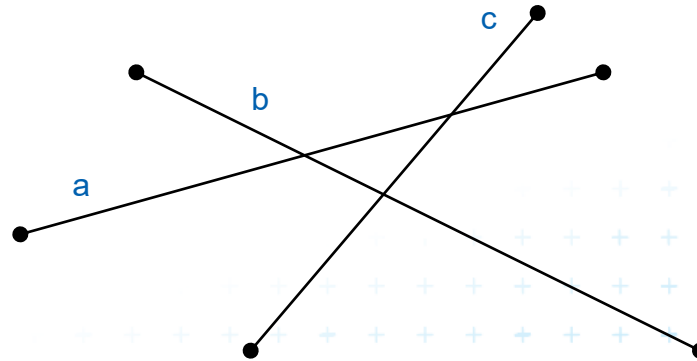
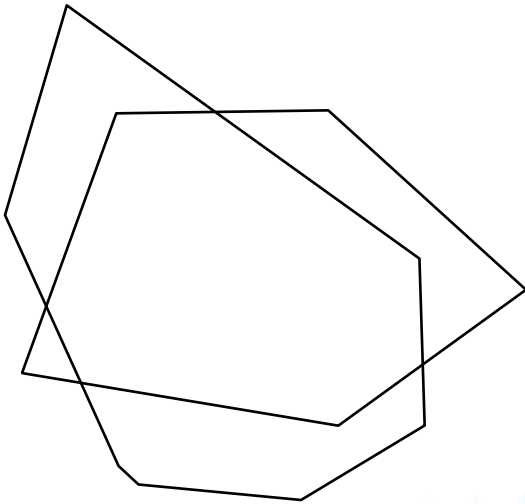
[Maur]



4.4 Typical tasks in CG

■ Intersection of objects

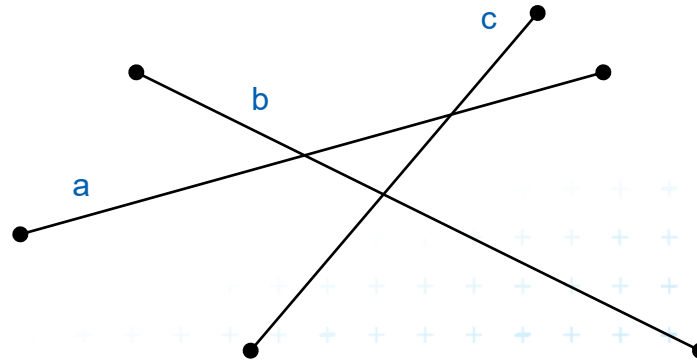
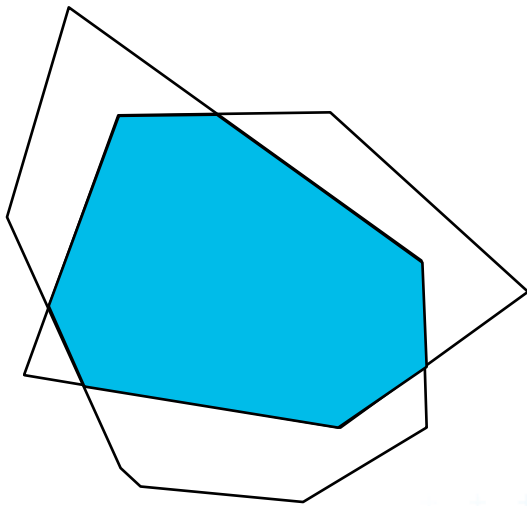
- Detection of common parts of objects
- Usually linear (line segments, polygons, n-gons,...)



4.4 Typical tasks in CG

■ Intersection of objects

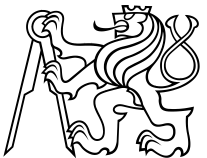
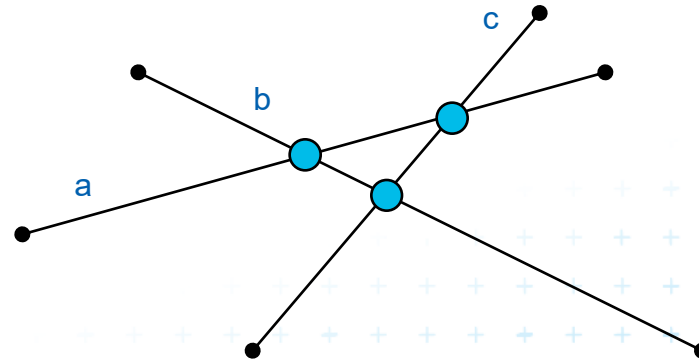
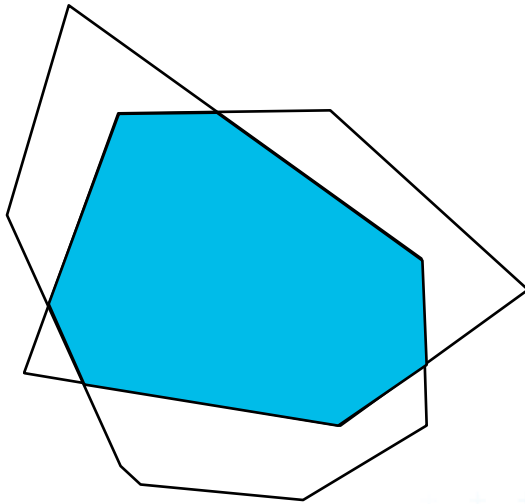
- Detection of common parts of objects
- Usually linear (line segments, polygons, n-gons,...)



4.4 Typical tasks in CG

■ Intersection of objects

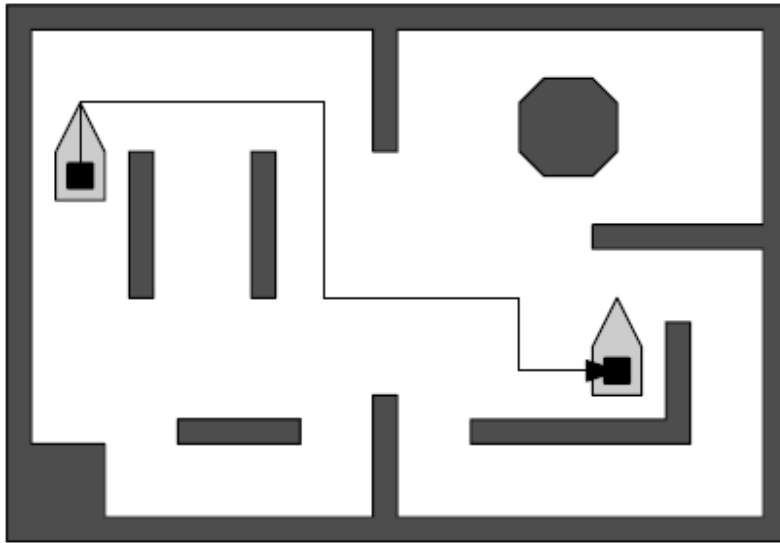
- Detection of common parts of objects
- Usually linear (line segments, polygons, n-gons,...)



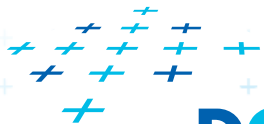
4.5 Typical tasks in CG

■ Motion planning

- Search for the shortest path between two points in the environment with obstacles



[Berg]

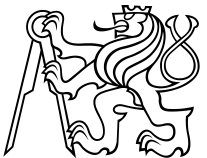


DCGI



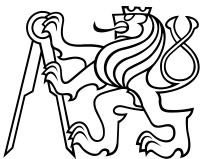
5. Complexity of algorithms and data struc.

- We need a measure for comparison of algorithms
 - Independent on computer HW and prog. language
 - Dependent on the **problem size n**
 - Describing the behavior of the algorithm for **different data**
- Running time, preprocessing time, memory size
 - Asymptotical analysis – functions $O(g(n))$, $\Omega(g(n))$, $\Theta(g(n))$
 - Measurement on real data
- Differentiate:
 - complexity of the algorithm (particular sort) and
 - complexity of the problem (sorting)
 - given by number of edges, vertices, faces, ... = problem size
 - less or equal to the complexity of the best algorithm



5.1 Complexity of algorithms

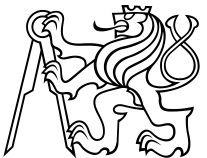
- Worst case behavior
 - Running time for the “worst” data
- Expected behavior (average)
 - expectation of the running time for problems of particular **size** and **probability distribution** of input data
 - Valid only if the probability distribution is the same as expected during the analysis
 - Typically much smaller than the worst case behavior
 - Ex.: Quick sort $O(n^2)$ worst and $O(n \log n)$ expected for standard data distribution



5.2 Complexity of algorithms

■ Amortized analysis

- Average over all operations
single operation may be expensive, but in average over all is small
- guaranteed regardless of probability distribution
which is taken into account in expected (average) behavior
(valid for any distribution)
- Ex.: Number of all pop operations over n steps is $O(n)$
⇒ average for single step is $O(1)$
Graham scan (lecture 4), computation of UHT (lecture 11),
...



6. Programming techniques (paradigms) of CG

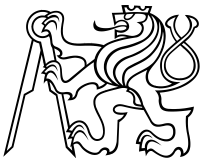
3 phases of a geometric algorithm development

1. Design an algorithm while ignoring all degeneracies
2. Adjust the algorithm to be correct for degenerate cases
 - Degenerate input exists
 - Integrate special cases in general case
 - It is better than lot of case-switches (typical for beginners)
 - e.g.:
 - lexicographic order for points on vertical lines
 - or Symbolic perturbation schemes
3. Implement alg. 2 (use sw library)



6.1 Sorting

- A preprocessing step
- Simplifies the following processing steps
- Sort according to:
 - coordinates x , or y , ... ,
 - lexicographically to $[y, x]$,
 - angles around point
- $O(n \log n)$ time and $O(n)$ space



6.2 Divide and Conquer (divide et impera)

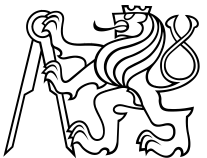
- Split the problem until it is solvable, merge results

```
DivideAndConquer(S)
```

1. **If** known solution **then** return it
2. **else**
3. Split input S to k distinct subsets S_i
4. Foreach i call `DivideAndConquer(S_i)`
5. Merge the results and return the solution

- Prerequisite

- The input data set must be separable
- Solutions of subsets are independent
- The result can be obtained by merging of sub-results



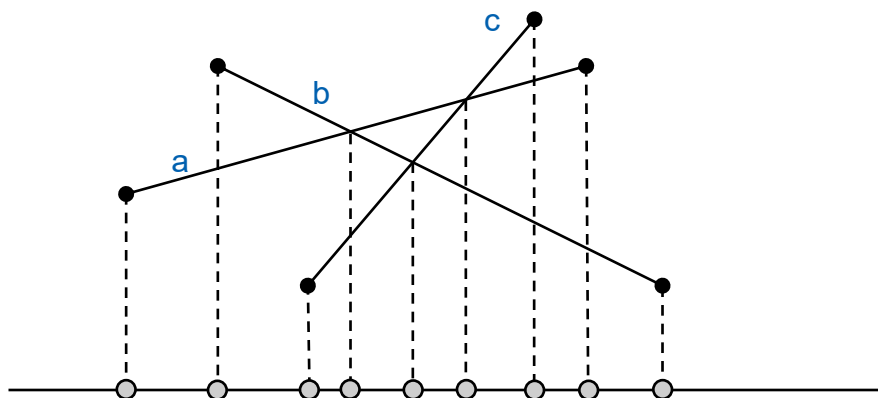
6.3 Sweep algorithm...

- Split the space by a hyperplane (2D: sweep line)
 - “Left” subspace – solved
 - “Right” subspace – to-do (solution unknown yet)
- Stop in event points and update the status
- Data structures:
 - **Event points** – points, where to stop the sweep line and update the status, sorted
 - **Status** – state of the algorithm in the current position of the sweep line
- Prerequisite:
 - Left subspace does not influence the right subspace



6.3 ... Sweep-line algorithm

Intersection of line segments



Event types for segments:

- start
- end
- intersection

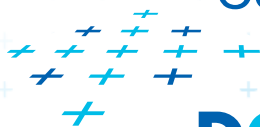
Event points – ordered in event queue

Status: {a}, {a,b}, {c,a,b}, {c,b,a}, ... – edges intersecting the sweep line

axb axc axb
 bxc

Principle:

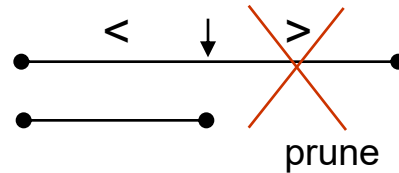
- Before intersection, the segments become neighbors (\Rightarrow we do not miss an intersection)
- Store segments in order they cross the sweep line
- Compute intersection of neighbors only



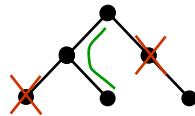
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

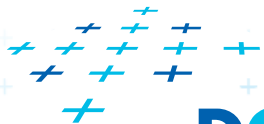
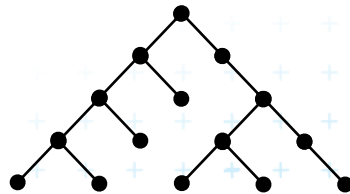
- Binary search



- Search trees



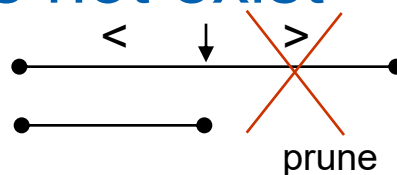
- Back-tracking (stop if solution worse than current optimum)



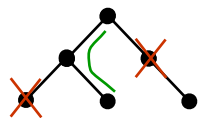
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

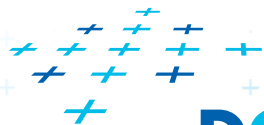
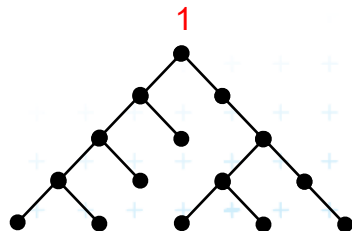
- Binary search



- Search trees



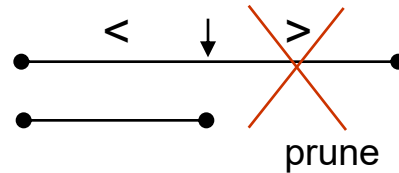
- Back-tracking (stop if solution worse than current optimum)



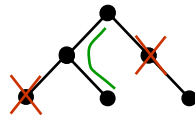
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

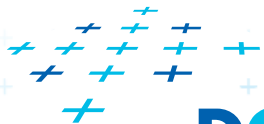
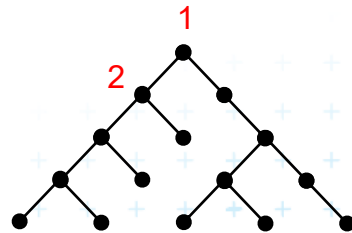
- Binary search



- Search trees



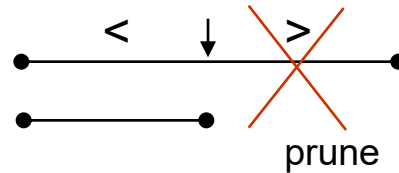
- Back-tracking (stop if solution worse than current optimum)



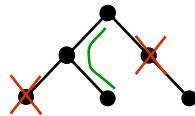
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

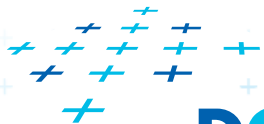
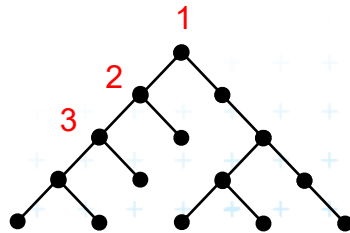
- Binary search



- Search trees



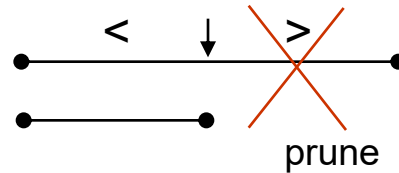
- Back-tracking (stop if solution worse than current optimum)



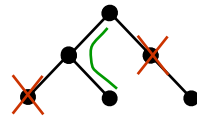
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

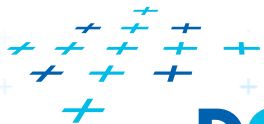
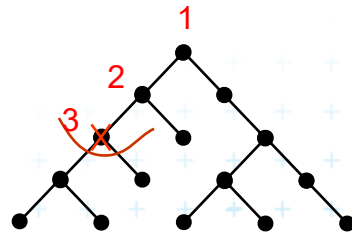
- Binary search



- Search trees



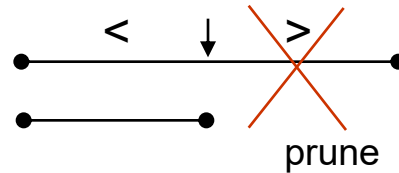
- Back-tracking (stop if solution worse than current optimum)



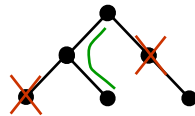
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

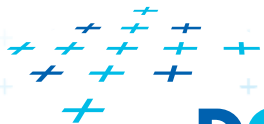
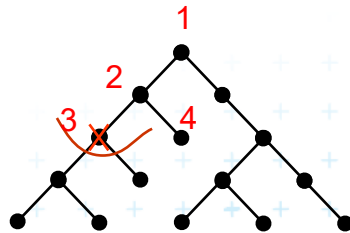
- Binary search



- Search trees



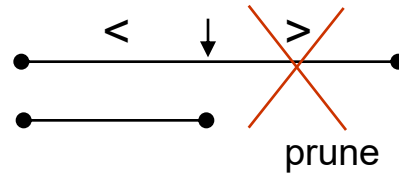
- Back-tracking (stop if solution worse than current optimum)



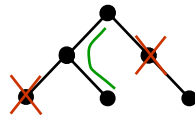
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

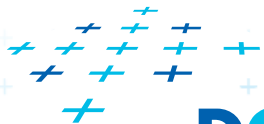
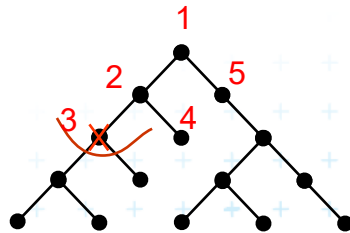
- Binary search



- Search trees



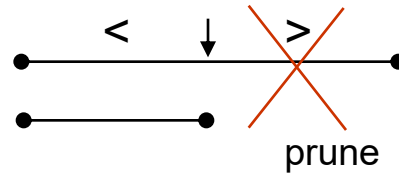
- Back-tracking (stop if solution worse than current optimum)



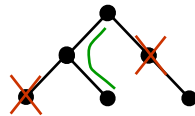
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

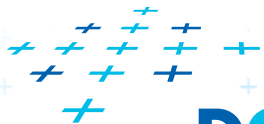
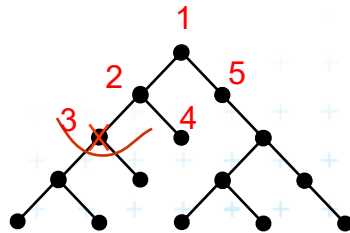
- Binary search



- Search trees



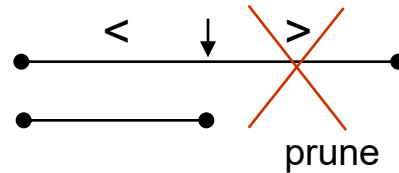
- Back-tracking (stop if solution worse than current optimum)



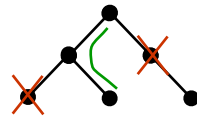
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

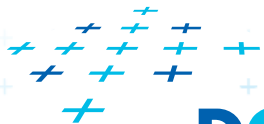
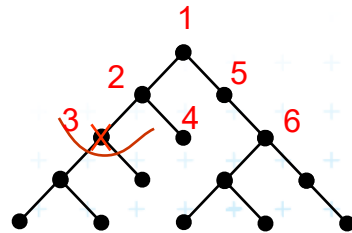
- Binary search



- Search trees



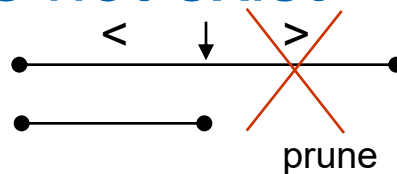
- Back-tracking (stop if solution worse than current optimum)



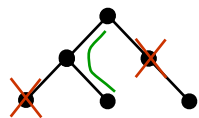
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

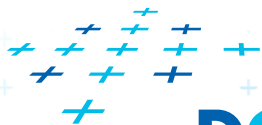
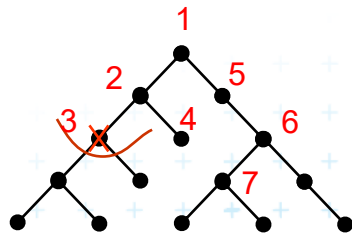
- Binary search



- Search trees



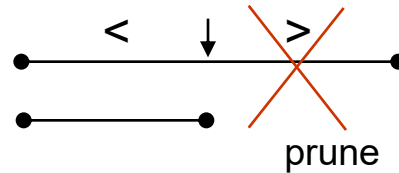
- Back-tracking (stop if solution worse than current optimum)



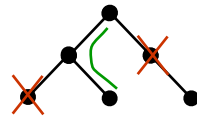
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

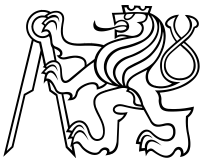
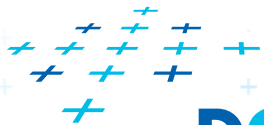
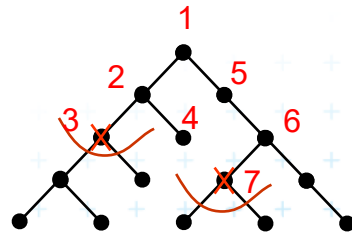
- Binary search



- Search trees



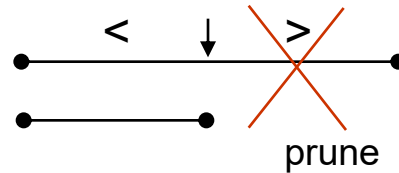
- Back-tracking (stop if solution worse than current optimum)



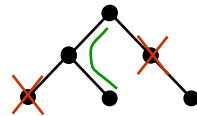
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

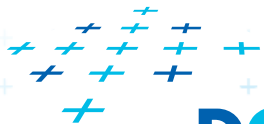
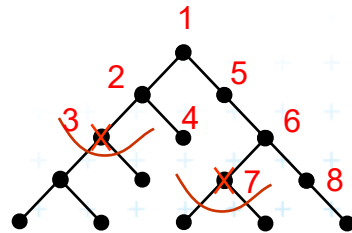
- Binary search



- Search trees



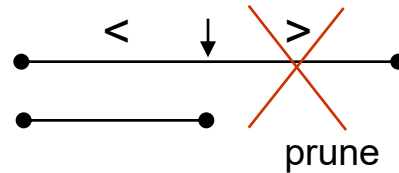
- Back-tracking (stop if solution worse than current optimum)



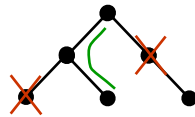
6.4 Prune and search

- Eliminate parts of the state space, where the solution clearly does not exist

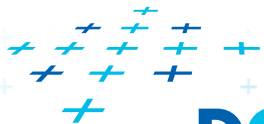
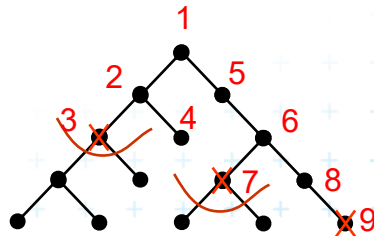
- Binary search



- Search trees

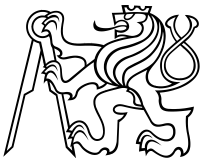
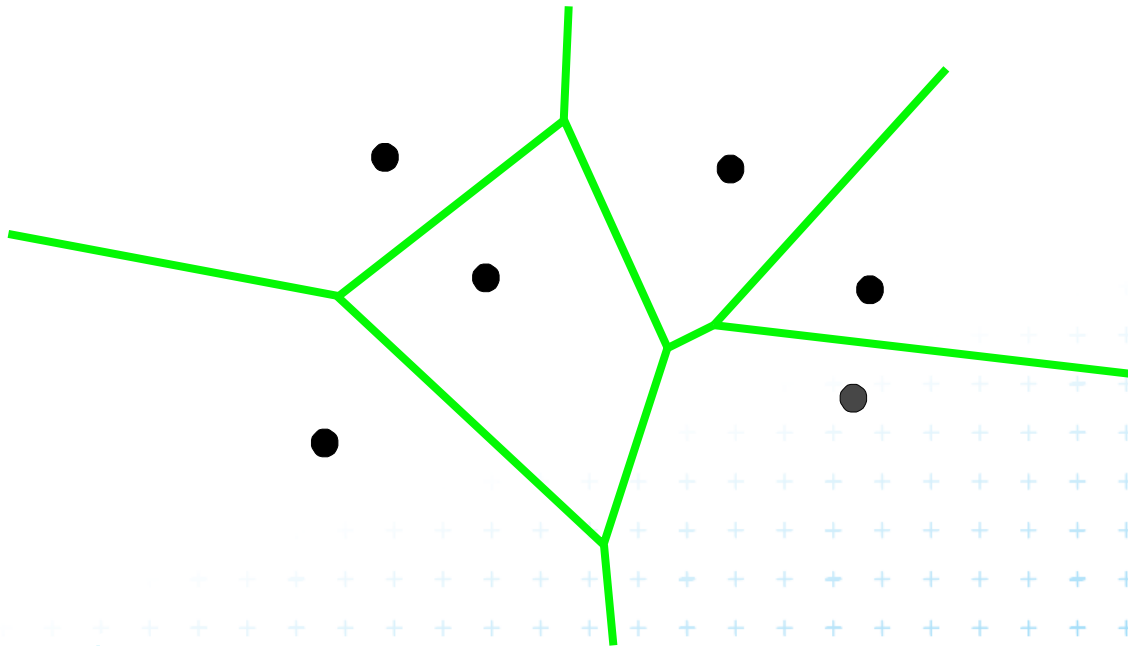


- Back-tracking (stop if solution worse than current optimum)



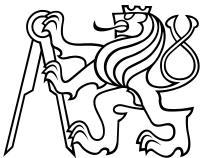
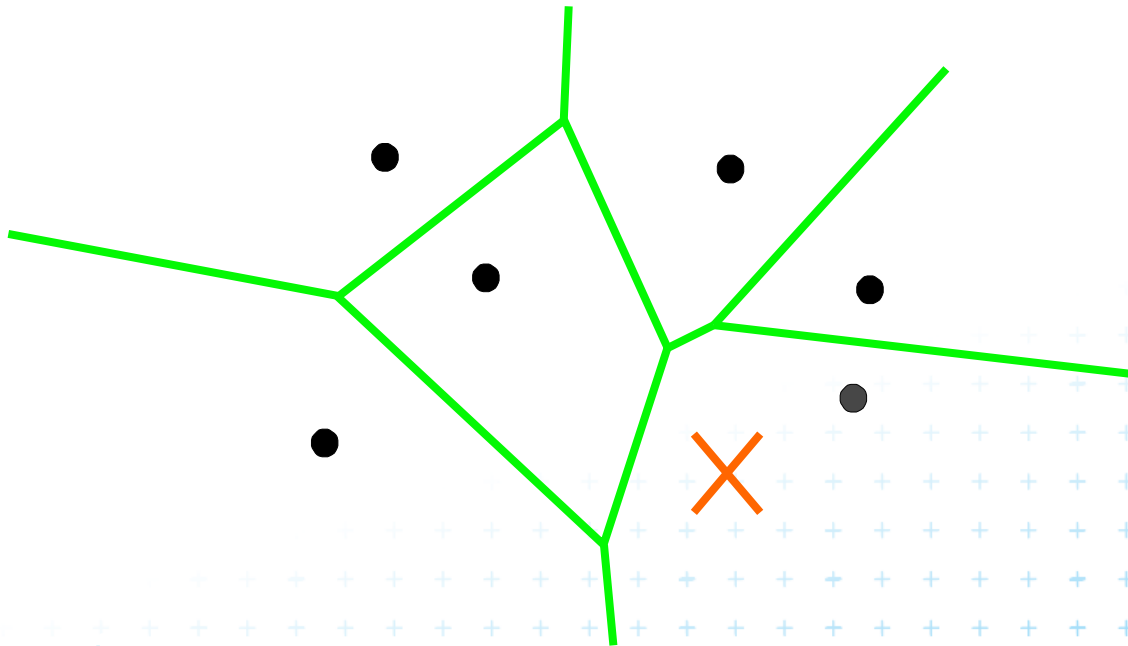
6.5 Locus approach

- Subdivide the search space into regions of constant answer
- Use point location to determine the region
 - Nearest neighbor search example



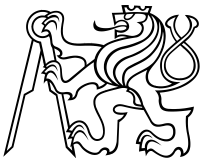
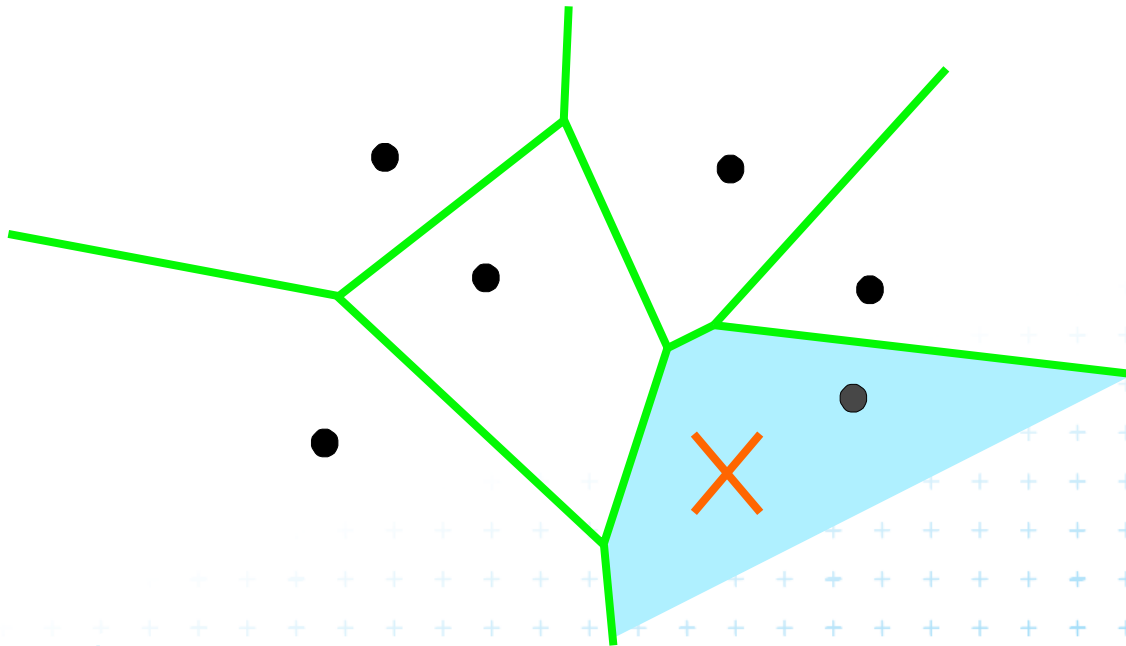
6.5 Locus approach

- Subdivide the search space into regions of constant answer
- Use point location to determine the region
 - Nearest neighbor search example



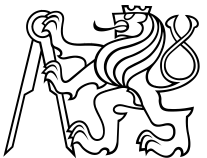
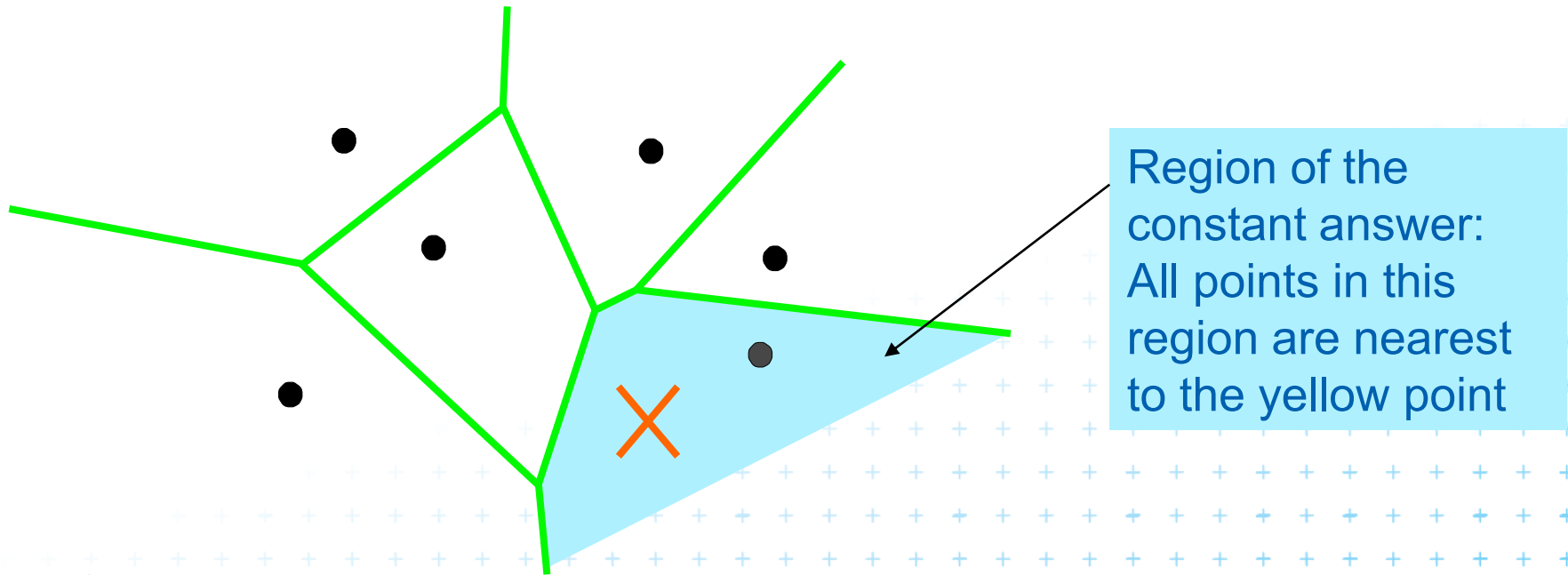
6.5 Locus approach

- Subdivide the search space into regions of constant answer
- Use point location to determine the region
 - Nearest neighbor search example



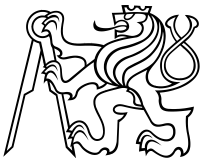
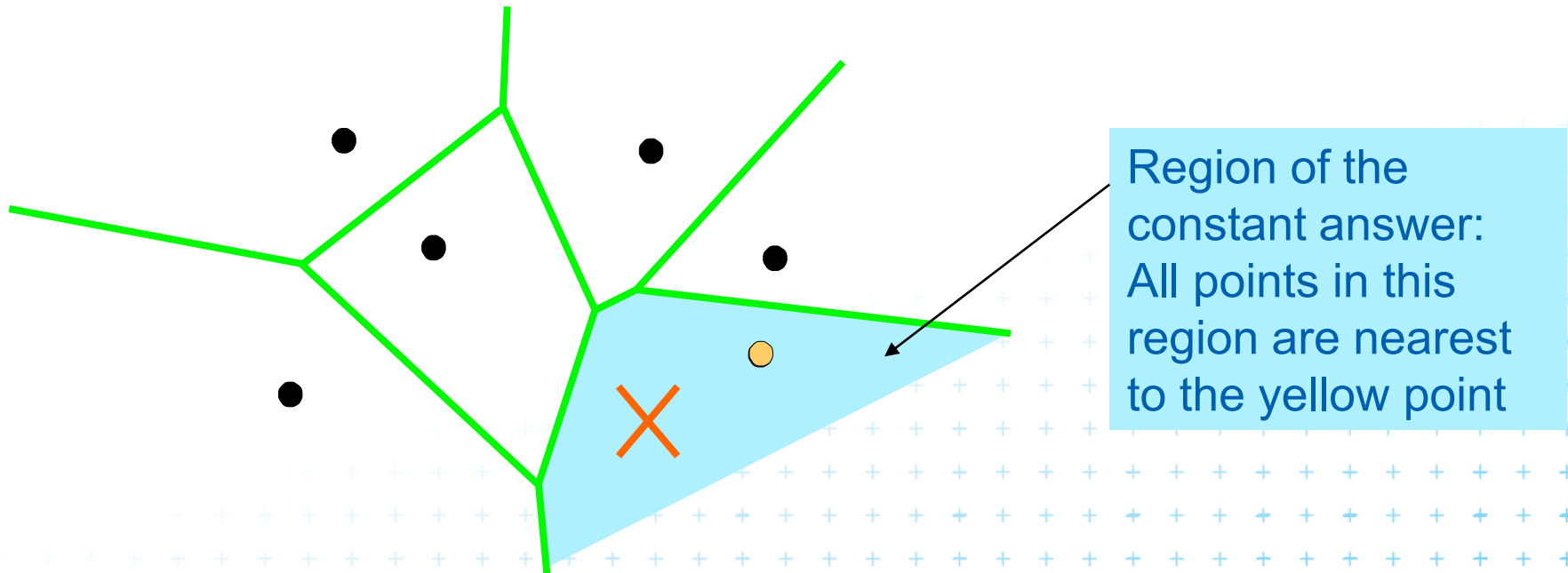
6.5 Locus approach

- Subdivide the search space into regions of constant answer
- Use point location to determine the region
 - Nearest neighbor search example



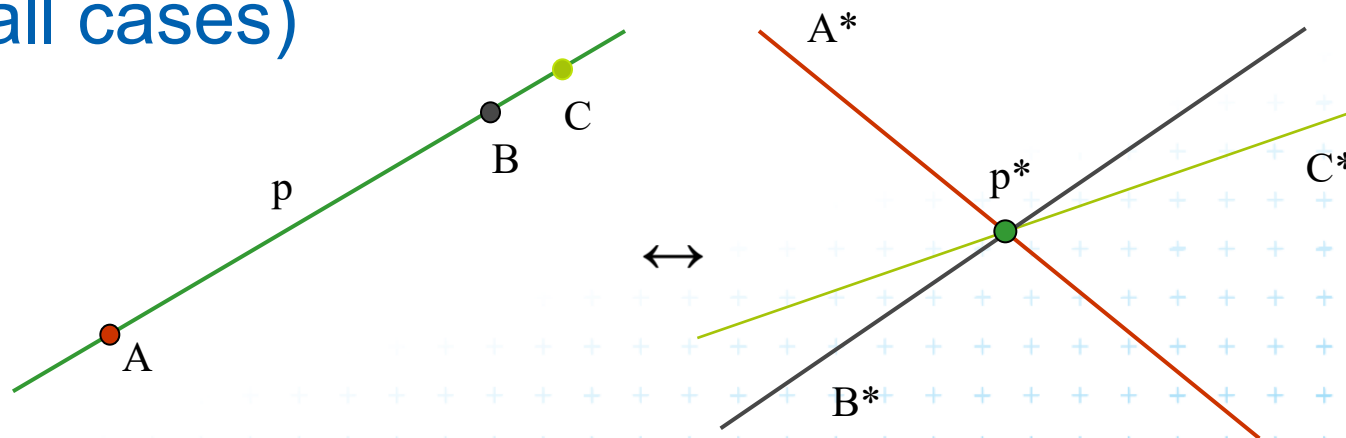
6.5 Locus approach

- Subdivide the search space into regions of constant answer
- Use point location to determine the region
 - Nearest neighbor search example



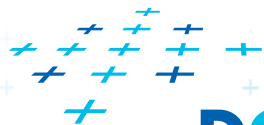
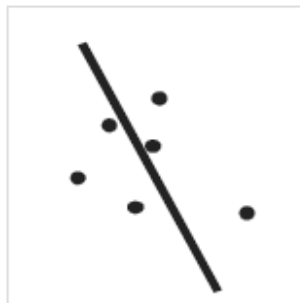
6.6 Dualisation

- Use geometry transform to change the problem into another that can be solved more easily
- Points \leftrightarrow hyper planes
 - Preservation of incidence ($A \in p \wedge p^* \in A^*$)
- Ex. 2D: determine if 3 points lie on a common line (all cases)



6.7 Combinatorial analysis

- = The branch of mathematics which studies the number of different ways of arranging things
- It limits the size of search space
 - Ex. How many subdivisions of a point set can be done by a single line?



6.8 New trends in Computational geometry

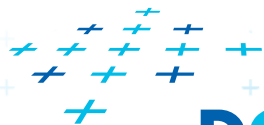
- From 2D to 3D and more from mid 80s, from linear to curved objects
- Focus on line segments, triangles in E^3 and hyper planes in E^d
- Strong influence of combinatorial geometry
- Randomized algorithms
- Space effective algorithms (in place, in situ, data stream algs.)
- Robust algorithms and handling of singularities
- Practical implementation in libraries (CGAL, ...)

■ Approximate algorithms



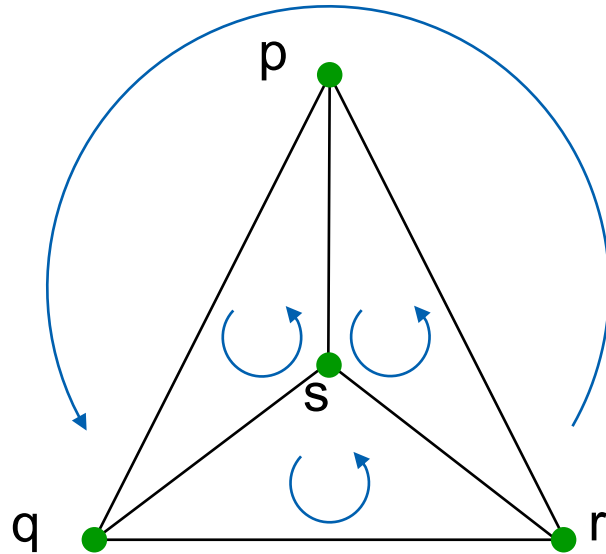
7. Robustness issues

- Geometry in theory is exact
- Geometry with floating-point arithmetic is not exact
 - Limited numerical precision of real arithmetic
 - Numbers are rounded to nearest possible representation
 - Inconsistent *epsilon* tests ($a = b, b = c$, but $a \neq c$)
- Naïve use of floating point arithmetic causes geometric algorithm to
 - Produce slightly or completely wrong output
 - Crash after invariant violation
 - Infinite loop

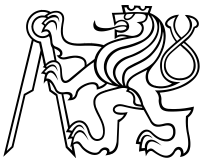


Geometry in theory is exact

- $\text{ccw}(s,q,r) \ \& \ \text{ccw}(p,s,r) \ \& \ \text{ccw}(p,q,s) \Rightarrow \text{ccw}(p,q,r)$

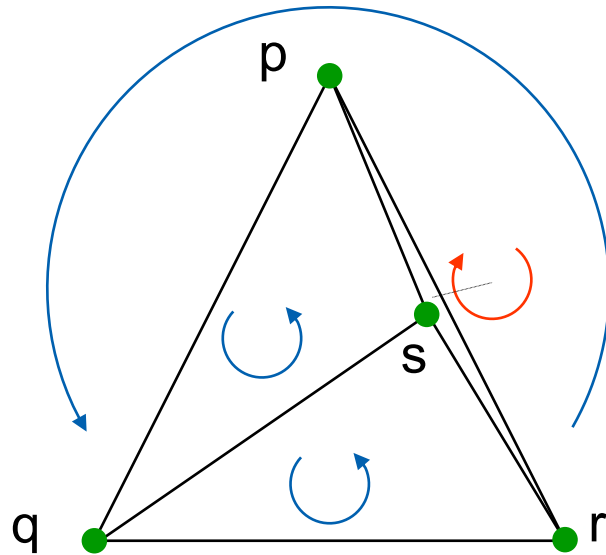


- Correctness proofs of algorithms rely on such theorems



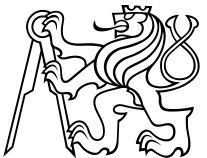
Geometry with float. arithmetic is not exact

- $ccw(s,q,r) \ \& \ !ccw(p,s,r) \ \& \ ccw(p,q,s) \not\Rightarrow ccw(p,q,r)$



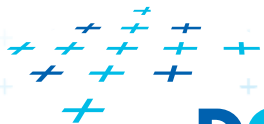
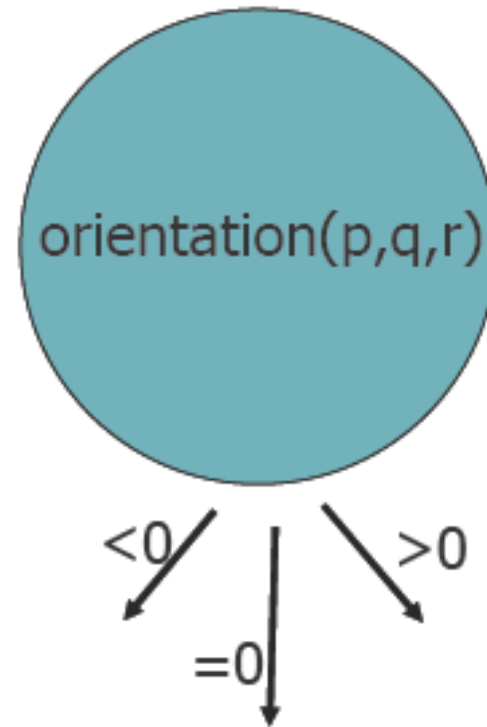
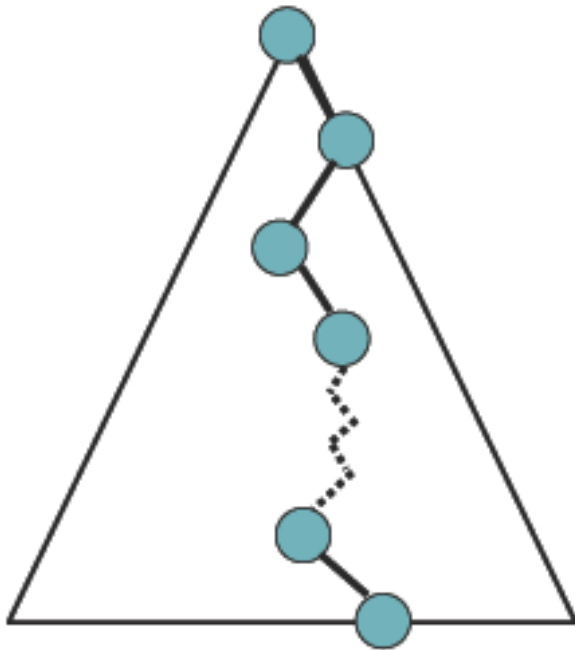
wrong result of the orientation predicate

- Correctness proofs of algorithms rely on such theorems \Rightarrow such algorithms fail



Exact Geometric Computing [Yap]

Make sure that the control flow in the implementation corresponds to the control flow with exact real arithmetic



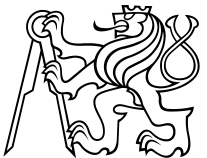
Floating-point arithmetic is not exact

a) Limited precision of storage

- quantization of mantissa

b) Limited precision of computations

- Losing lower bits during addition (aligning to the common exponent)
- Rounding of results after multiplications

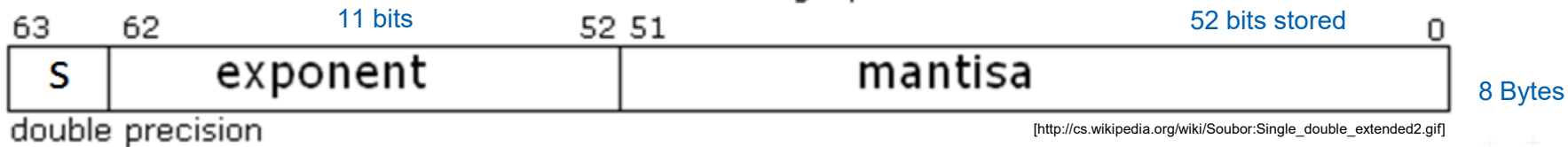
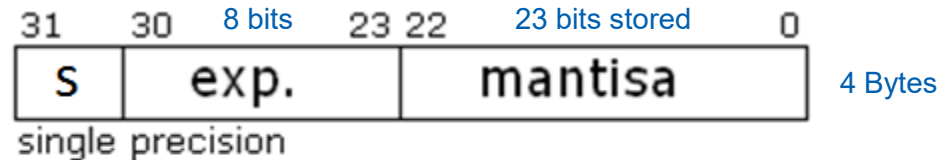


Floating-point arithmetic is not exact

a) Limited numerical precision of real numbers storage

- Numbers represented as normal (with 1 as first digit)

$$\pm \text{mantissa} 2^{\text{exponent}}$$



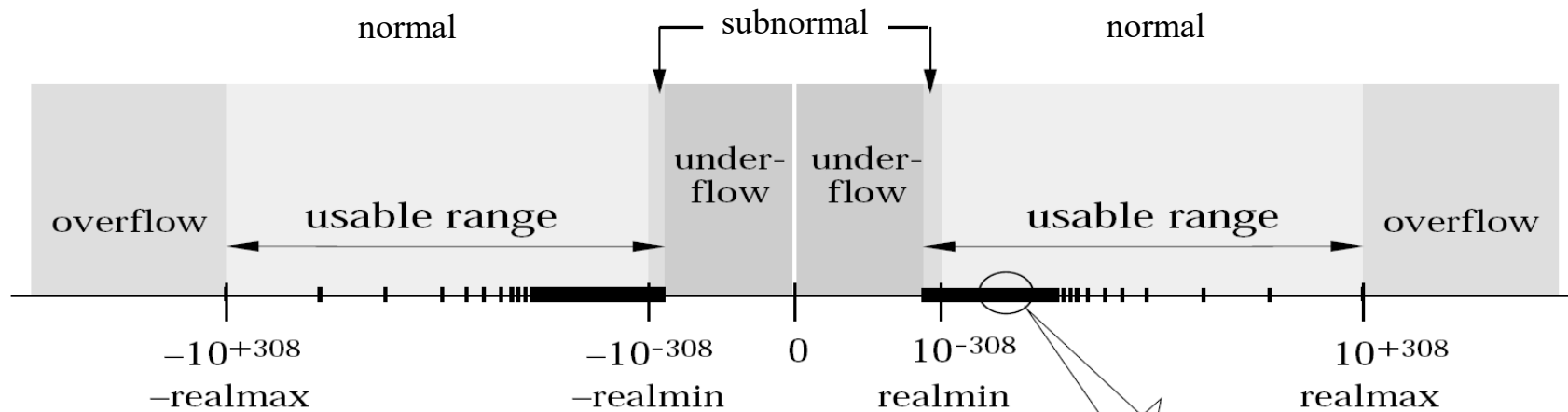
[http://cs.wikipedia.org/wiki/Soubor:Single_double_extended2.gif]

- The normal mantissa m is a 24-bit (53-bit) value whose most significant bit (MSB) is always 1 and is, therefore, not stored.
- Stored numbers are rounded to 24/53 bits mantissa – lower bits are lost



Floating-point arithmetic is not exact

Floating Point Number Line



normal – mantissa starts with 1

subnormal – mantissa starts with 0

each increment of exponent – doubles the gap between values



Floating-point arithmetic is not exact

“powers of 2” are stored exactly: 1, (357, 0.5,...) – up to the size of mantissa

0x3F80000 = 00111111 10000000 00000000 00000000



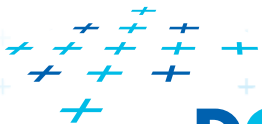
Others NOT: 0.6 stored as 6.0000002384185791015625E-1

0x3F19999A = 00111111 00011001 10011001 10011010



1.40129846432481707092372958329E-45 subnormal

0x00000001 = 00000000 00000000 00000000 00000001



Floating-point special values

+0



-0



subnormal



Any bit 1 in mantissa

+Infinity



-Infinity



NaN



Any bit 1 in mantissa



Floating-point arithmetic is not exact

b) Smaller numbers are shifted right during additions and subtractions to align the digits of the same order

Ex.: $12 - p$ for $p \sim 0.5$

Invisible leading bit – not stored

float

$$12_{10} = 1100_2 = 0 \overset{2^3}{\boxed{10000010}} \overset{1}{\boxed{100000000000000000000000}}_2$$

Normalized mantissa 23 bit

$$p = 0.5_{10} = 0 \overset{2^{-1}}{\boxed{01111110}} \overset{1}{\boxed{000000000000000000000000}}_2$$

$$p = 0.5000008_{10} = 0 \overset{2^{-1}}{\boxed{01111110}} \overset{1}{\boxed{0000000000000000000000001101}}_2$$

Mantissa of p is shifted 4 bits right to align with 12

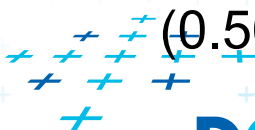
(to have the same exponent 2^3) Sub-normal mantissa

$$p = 0.5000008_{10} = 0 \overset{2^3}{\boxed{10000010}} \overset{1}{\boxed{000100000000000000000000}}_2 \overset{1101}{}$$

→ four least significant bits (LSB) are lost

– The result is 11.5 instead of 11.4999992

(0.500000774860382080078125 and 11.499999904632568359375)



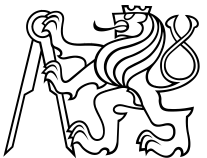
Floating-point arithmetic is not exact

b) Smaller numbers are shifted right during additions and subtractions to align the digits of the same order

Example for float:

- $12 - p$ for $p \sim 0.5$ (near 0.5, such as $0.5+2^{(-23)}$, or 0.5000008)
 - Mantissa of p is shifted 4 bits right to align with 12
→ four least significant bits (LSB) are lost
- $24 - p$ for $p \sim 0.5$ (near 0.5, such as $0.5+2^{(-23)}$, or 0.5000008)
 - Mantissa of p is shifted 5 bits right to align with 24 → 5 LSB are lost

Try it on <http://www.h-schmidt.net/FloatConverter/IEEE754.html> or <http://babbage.cs.qc.cuny.edu/IEEE-754/index.xhtml>



Orientation predicate - definition

$$\text{orientation}(p, q, r) = \text{sign} \left(\det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix} \right) =$$
$$= \text{sign} \left((p_x - r_x)(q_y - r_y) - (p_y - r_y)(q_x - r_x) \right),$$

where point $p = (p_x, p_y), \dots$

$=$ sign of the third coordinate of $= (\vec{u} \times \vec{v})$,

Three points

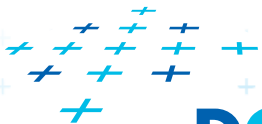
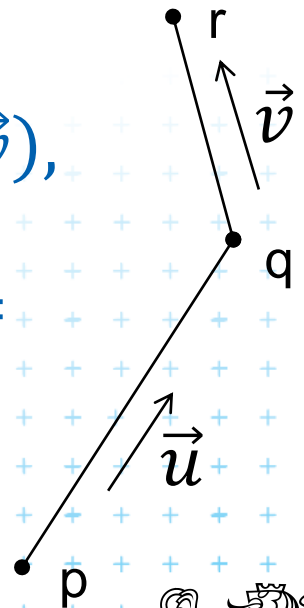
- lie on common line
- form a left turn
- form a right turn

$\text{orientation}(p, q, r) =$

$= 0$

$= +1$ (positive)

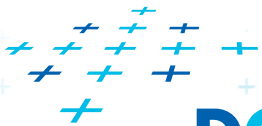
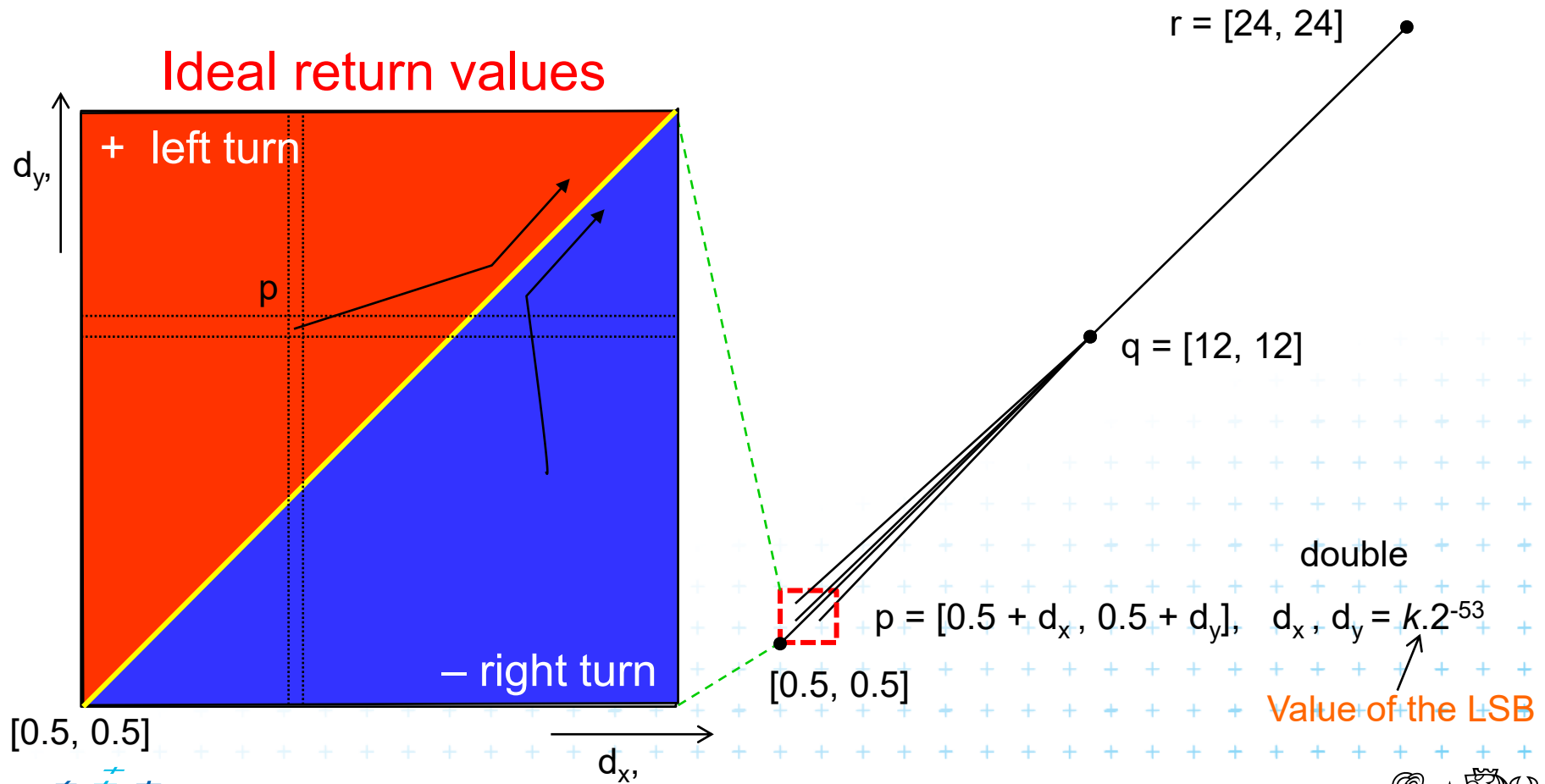
$= -1$ (negative)



Experiment with orientation predicate

Pivot r

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



DCGI

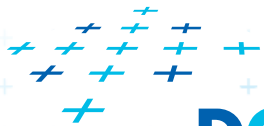
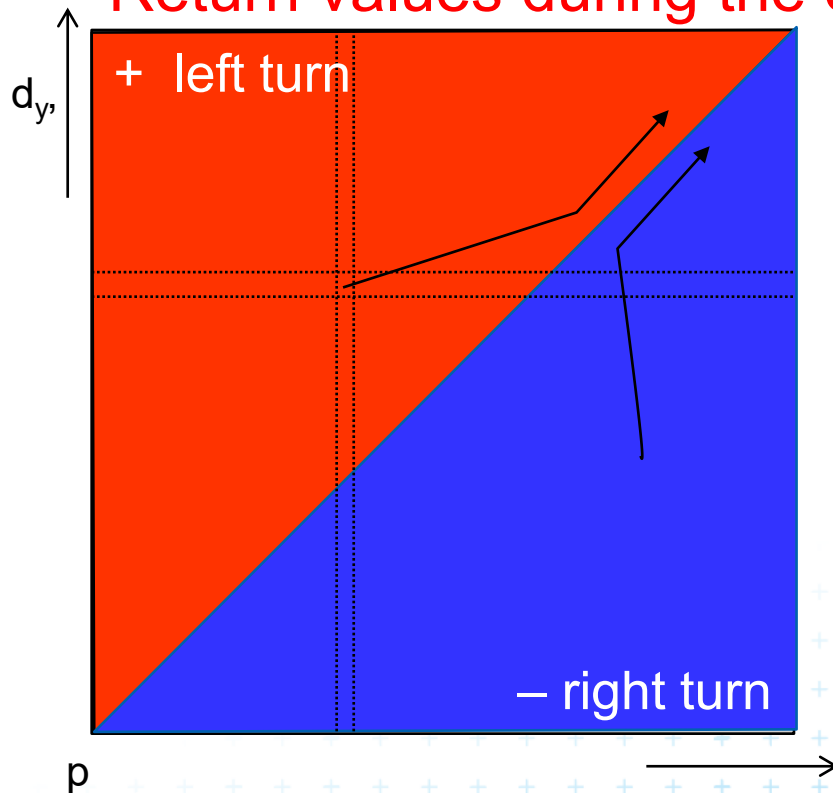


Real results of orientation predicate

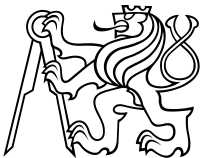
Pivot r

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

Return values during the experiment for exponent > -52



DCGI

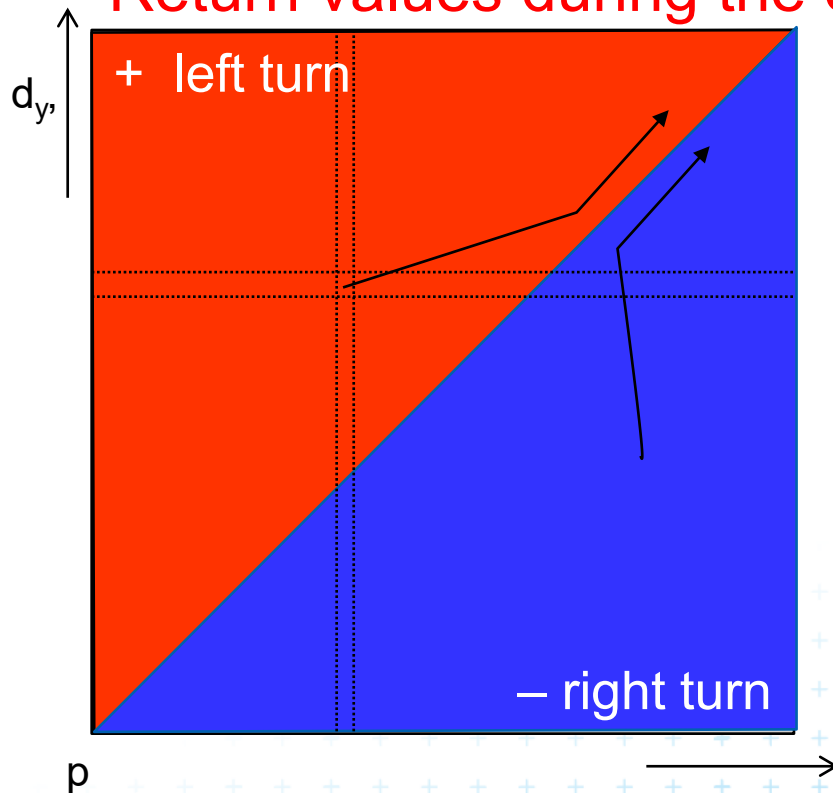


Real results of orientation predicate

Pivot r

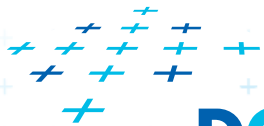
- $\text{orientation}(p,q,r) = \text{sign}((p_x - r_x)(q_y - r_y) - (p_y - r_y)(q_x - r_x))$

Return values during the experiment for exponent > -52



Where is the yellow line?

Err



DCGI

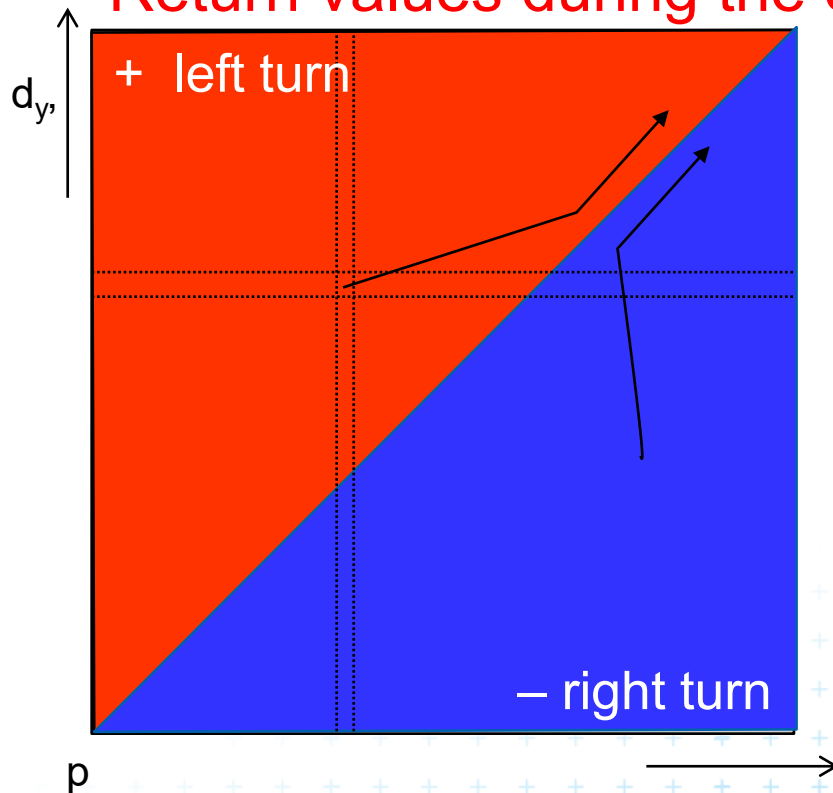


Real results of orientation predicate

Pivot r

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

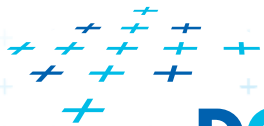
Return values during the experiment for exponent > -52



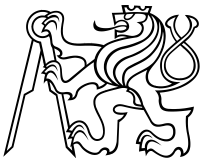
Where is the yellow line?

Input pts rounded – quantization

Err



DCGI

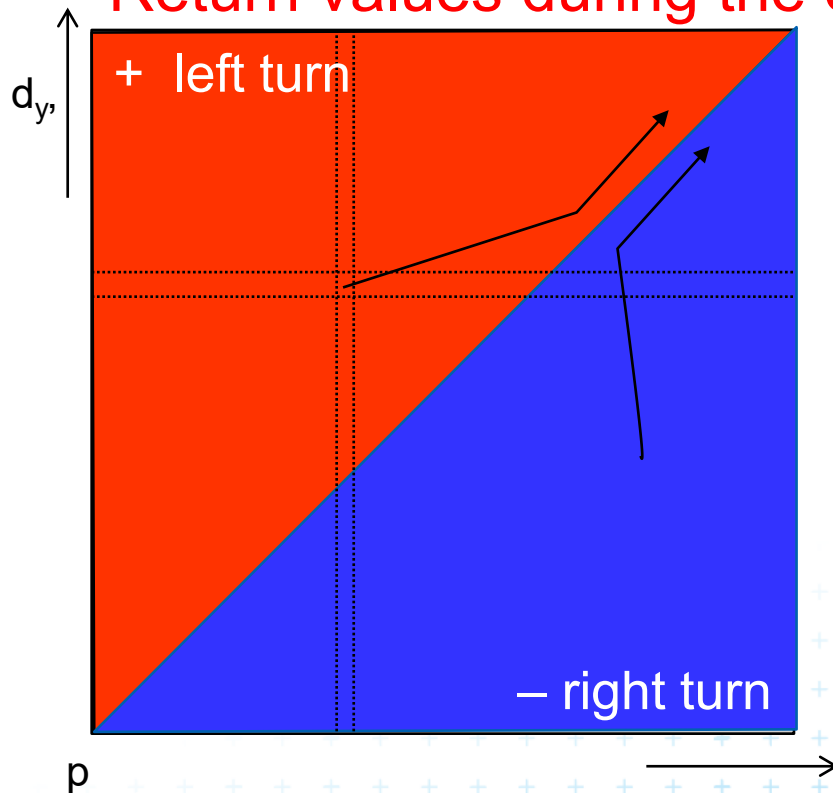


Real results of orientation predicate

Pivot r

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

Return values during the experiment for exponent > -52



Where is the yellow line?

Input pts rounded – quantization

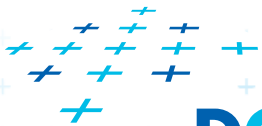
Robust predicate returns

slightly non-zero values - OK

$\text{orientation}(p, q, r) \neq 0$

The input points do not lie on common line due to rounding before storage

Err



DCGI

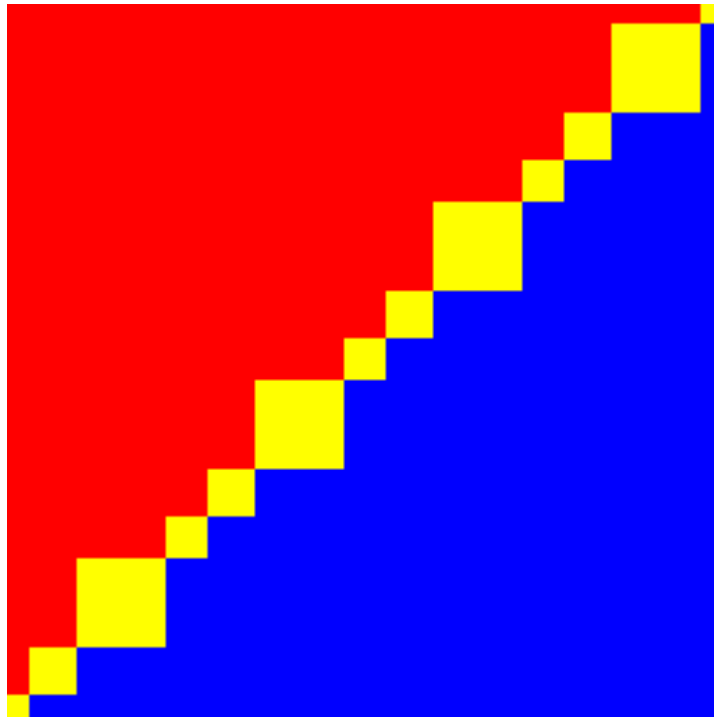


Real results of orientation predicate

Pivot r

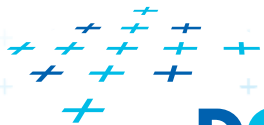
- $\text{orientation}(p,q,r) = \text{sign}((p_x - r_x)(q_y - r_y) - (p_y - r_y)(q_x - r_x))$

Return values during the experiment for exponent -52

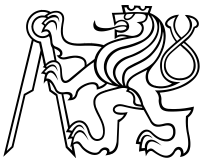


Pivot r ₂₄

Pivot p _{0.5}



DCGI

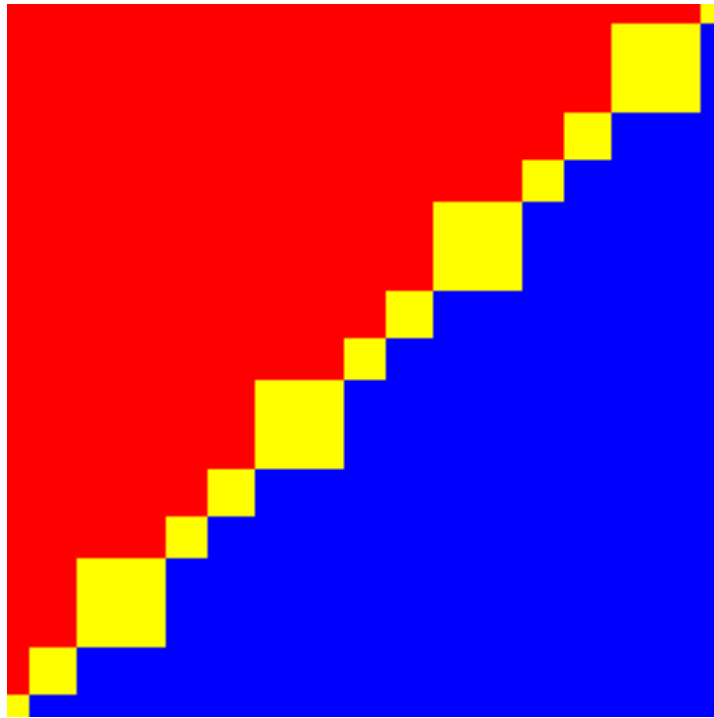


Real results of orientation predicate

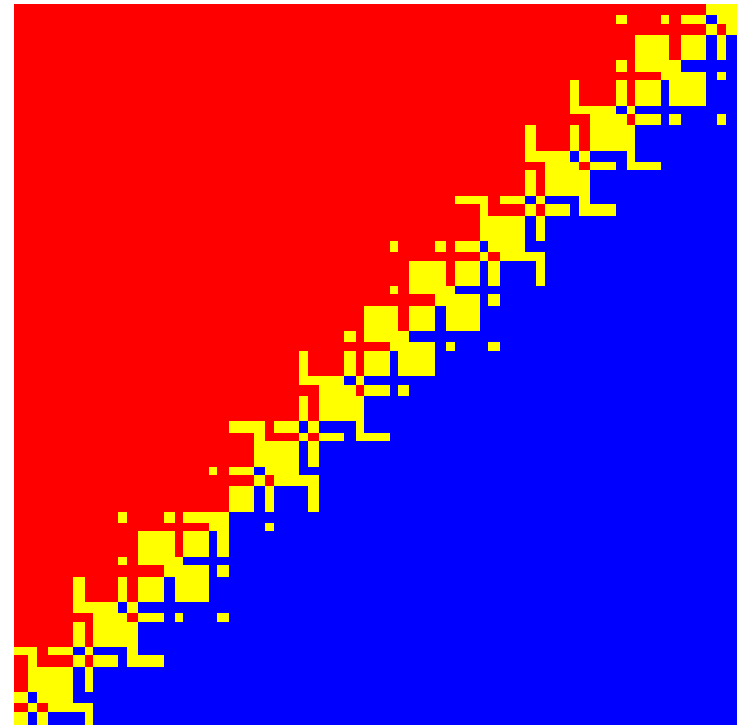
Pivot r

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

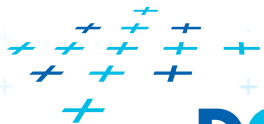
Return values during the experiment for exponent -52



Pivot r ₂₄



Pivot p _{0.5}



DCGI

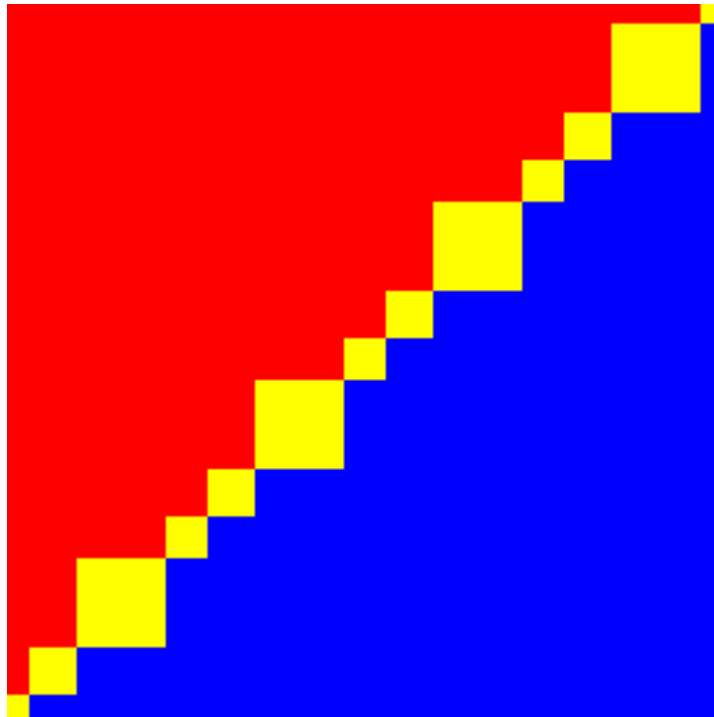


Real results of orientation predicate

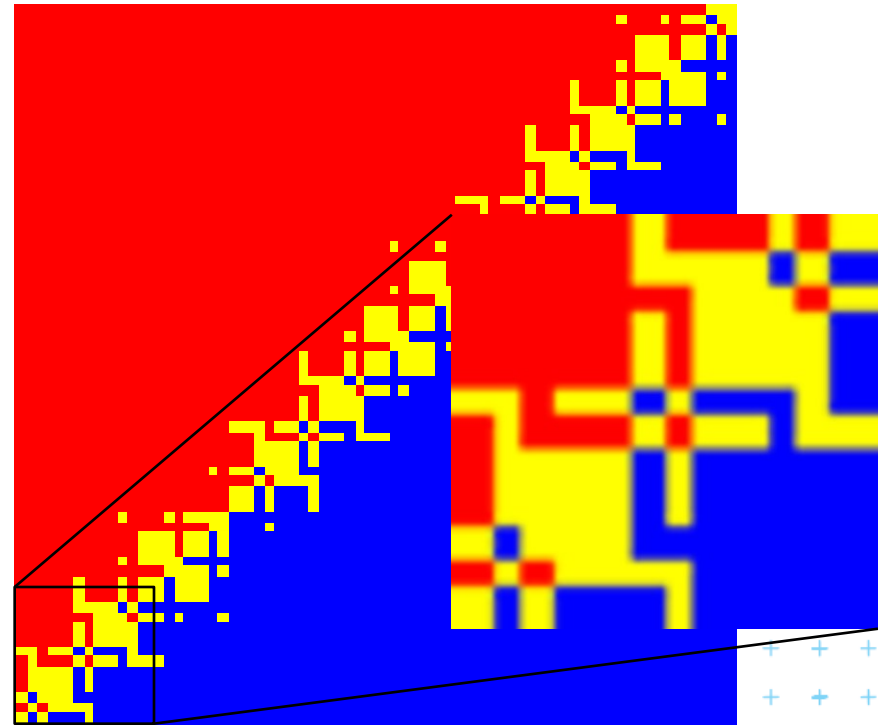
Pivot r

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

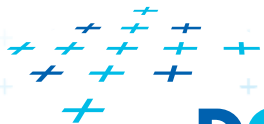
Return values during the experiment for exponent -52



Pivot r ₂₄



Pivot p _{0.5}

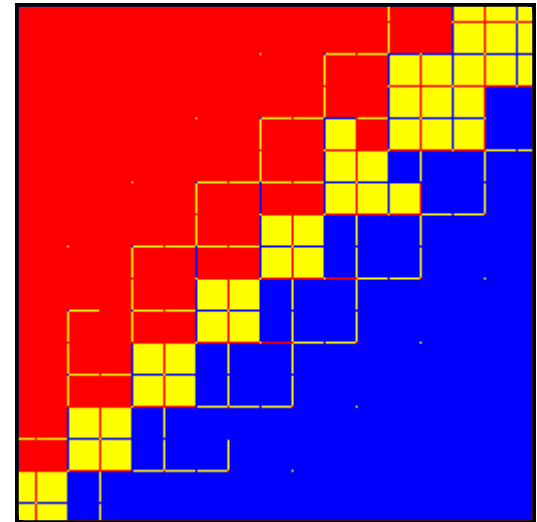
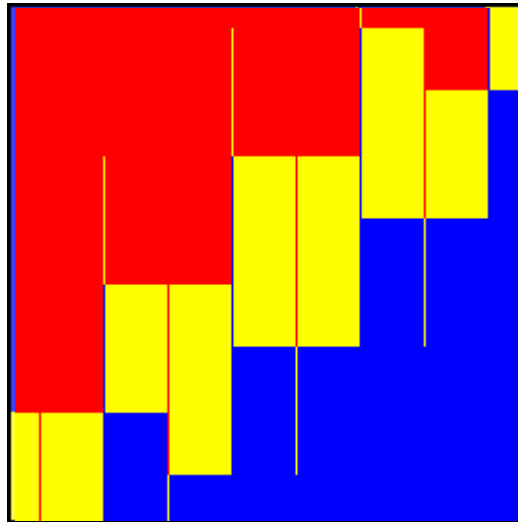
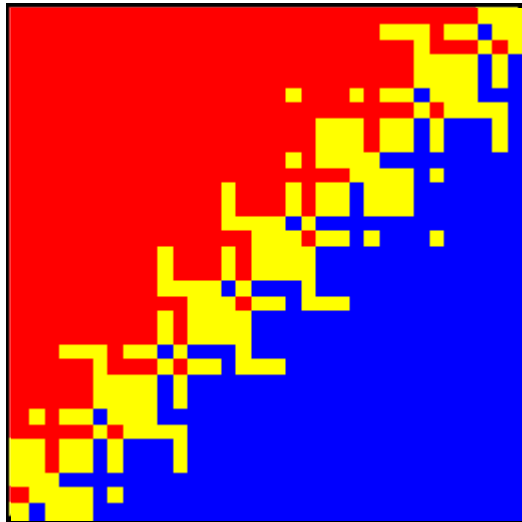


DCGI



Floating point orientation predicate double exp=-53

Pivot p



$$p: \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

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

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

(a)

$$\begin{pmatrix} 0.50000000000002531 \\ 0.5000000000000171 \end{pmatrix}$$

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

$$\begin{pmatrix} 24.000000000000005 \\ 24.00000000000000517765 \end{pmatrix}$$

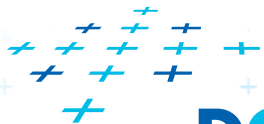
(b)

$$\begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

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

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

(c)



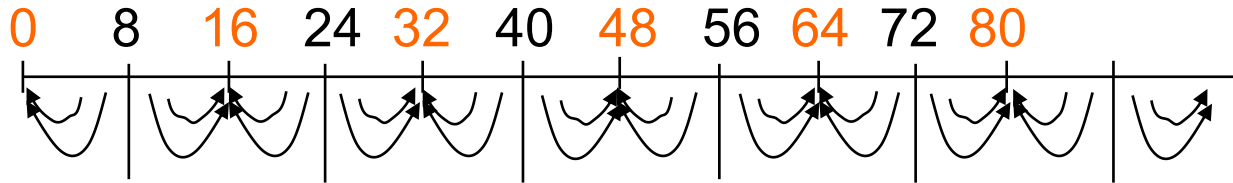
DCGI

[Kettner] with correct colors

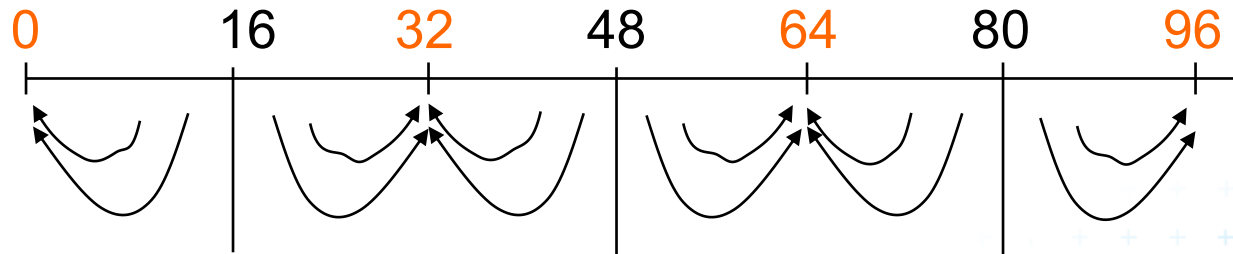


Errors from shift ~ 0.5 right in subtraction

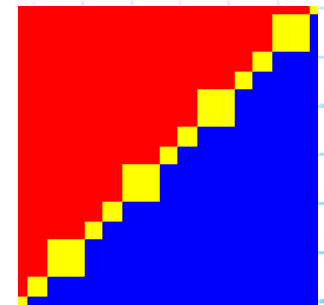
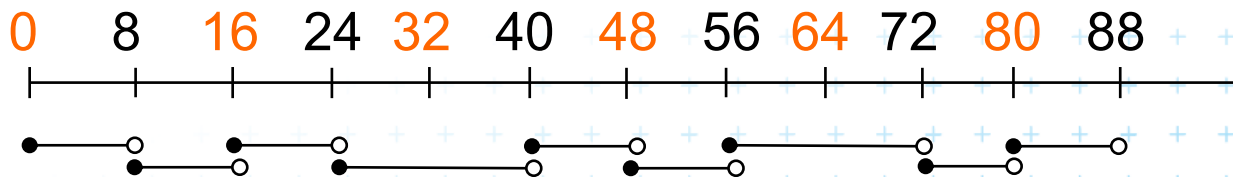
- 4 bits shift $\Rightarrow 2^4$ values rounded to the same value



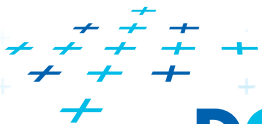
- 5 bits shift $\Rightarrow 2^5$ values rounded to the same value



- Combined intervals of size 8, 16, 24,...



These intervals match the size of rectangular areas of the same value



Orientation predicate – pivot selection

$$\text{orientation}(p, q, r) = \text{sign} \left(\det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix} \right) =$$

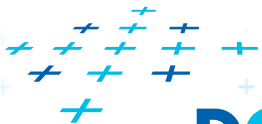
The formula depends on the selection of the **pivot**,
pivot point = row to be subtracted from other rows

$$p: = \text{sign} \left((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x) \right)$$

$$q: = \text{sign} \left((r_x - q_x)(p_y - q_y) - (r_y - q_y)(p_x - q_x) \right)$$

$$r: = \text{sign} \left((p_x - r_x)(q_y - r_y) - (p_y - r_y)(q_x - r_x) \right)$$

$$p_x = 0.5, \quad q_x = 12, \quad r_x = 24$$



Orientation predicate – pivot selection

$$\text{orientation}(p, q, r) = \text{sign} \left(\det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix} \right) =$$

The formula depends on the selection of the **pivot**,
pivot point = row to be subtracted from other rows

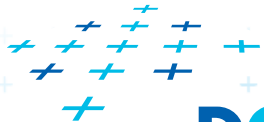
$$p: = \text{sign} \left((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x) \right)$$

$$q: = \text{sign} \left((r_x - q_x)(p_y - q_y) - (r_y - q_y)(p_x - q_x) \right)$$

$$r: = \text{sign} \left((p_x - r_x)(q_y - r_y) - (p_y - r_y)(q_x - r_x) \right)$$

Which pivot is the worst?

$$p_x = 0.5, \quad q_x = 12, \quad r_x = 24$$



Orientation predicate – pivot selection

$$\text{orientation}(p, q, r) = \text{sign} \left(\det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix} \right) =$$

The formula depends on the selection of the **pivot**,
pivot point = row to be subtracted from other rows

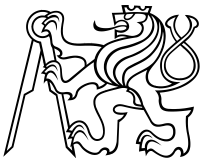
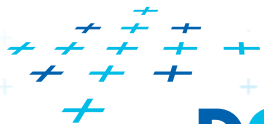
$$p: = \text{sign} \left(\overset{4 \text{ bits lost}}{(q_x - p_x)}(r_y - p_y) - \overset{4 \text{ bits lost}}{(q_y - p_y)}(r_x - p_x) \right)$$

$$q: = \text{sign} \left((r_x - q_x) \overset{4 \text{ bits lost}}{(p_y - q_y)} - (r_y - q_y) \overset{4 \text{ bits lost}}{(p_x - q_x)} \right)$$

$$r: = \text{sign} \left((p_x - r_x)(q_y - r_y) - (p_y - r_y)(q_x - r_x) \right)$$

Which pivot is the worst?

$$p_x = 0.5, \quad q_x = 12, \quad r_x = 24$$



Orientation predicate – pivot selection

$$\text{orientation}(p, q, r) = \text{sign} \left(\det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix} \right) =$$

The formula depends on the selection of the **pivot**,
pivot point = row to be subtracted from other rows

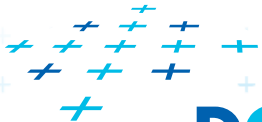
$$p: = \text{sign} \left(\overset{4 \text{ bits lost}}{(q_x - p_x)} \overset{5 \text{ bits lost}}{(r_y - p_y)} - \overset{4 \text{ bits lost}}{(q_y - p_y)} \overset{5 \text{ bits lost}}{(r_x - p_x)} \right)$$

$$q: = \text{sign} \left((r_x - q_x) \overset{4 \text{ bits lost}}{(p_y - q_y)} - (r_y - q_y) \overset{4 \text{ bits lost}}{(p_x - q_x)} \right)$$

$$r: = \text{sign} \left(\overset{5 \text{ bits lost}}{(p_x - r_x)} (q_y - r_y) - \overset{5 \text{ bits lost}}{(p_y - r_y)} (q_x - r_x) \right)$$

Which pivot is the worst?

$$p_x = 0.5, \quad q_x = 12, \quad r_x = 24$$



Orientation predicate – pivot selection

$$\text{orientation}(p, q, r) = \text{sign} \left(\det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix} \right) =$$

The formula depends on the selection of the **pivot**,
pivot point = row to be subtracted from other rows

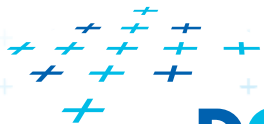
$$p: = \text{sign} \left(\overset{4 \text{ bits lost}}{(q_x - p_x)} \overset{5 \text{ bits lost}}{(r_y - p_y)} - \overset{4 \text{ bits lost}}{(q_y - p_y)} \overset{5 \text{ bits lost}}{(r_x - p_x)} \right)$$

$$q: = \text{sign} \left((r_x - q_x) \overset{4 \text{ bits lost}}{(p_y - q_y)} - (r_y - q_y) \overset{4 \text{ bits lost}}{(p_x - q_x)} \right)$$

$$r: = \text{sign} \left(\overset{5 \text{ bits lost}}{(p_x - r_x)} (q_y - r_y) - \overset{5 \text{ bits lost}}{(p_y - r_y)} (q_x - r_x) \right)$$

Which pivot is the worst?

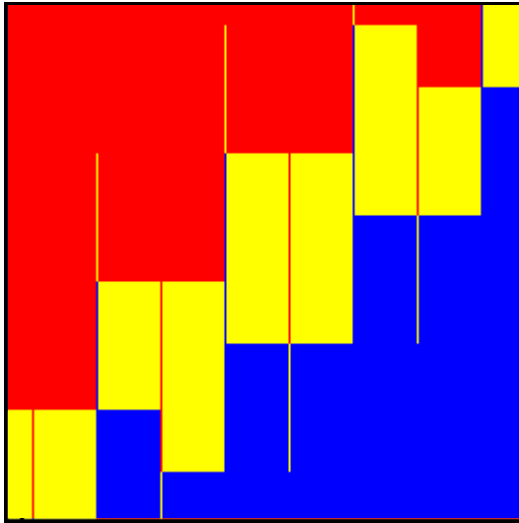
$$p_x = 0.5, \quad q_x = 12, \quad r_x = 24$$



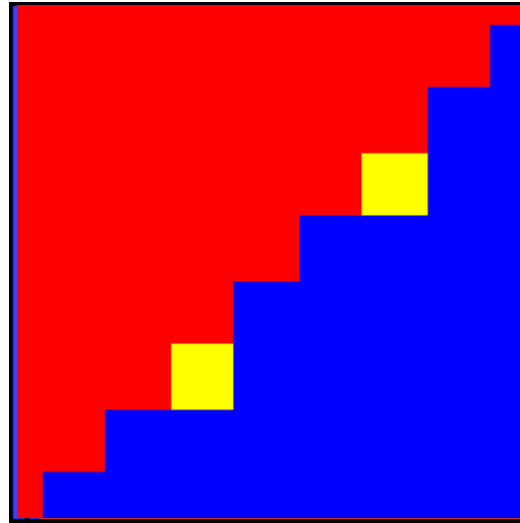
Little improvement - selection of the pivot

(b) double exp=-53

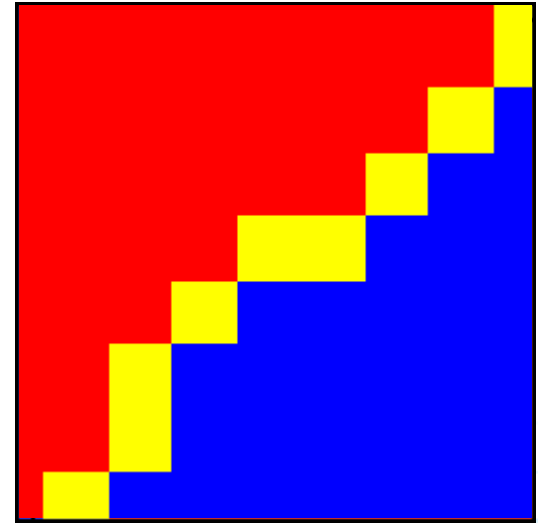
- Pivot – subtracted from the rows in the matrix



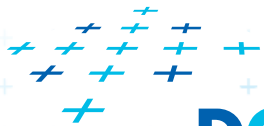
Pivot p _{0.5}



Pivot q ₁₂



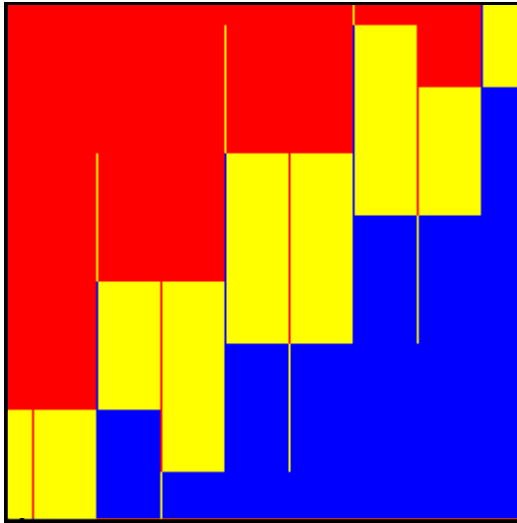
Pivot r ₂₄



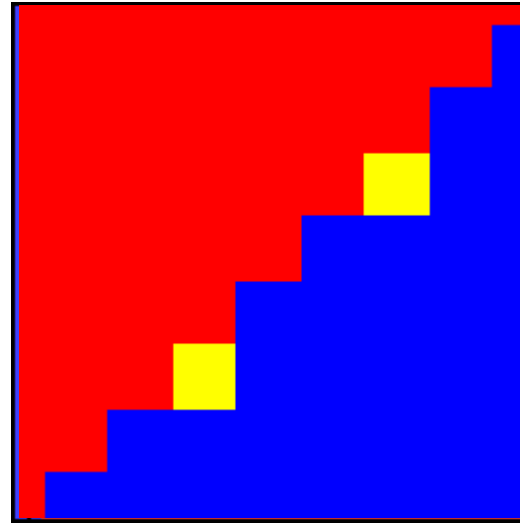
Little improvement - selection of the pivot

(b) double exp=-53

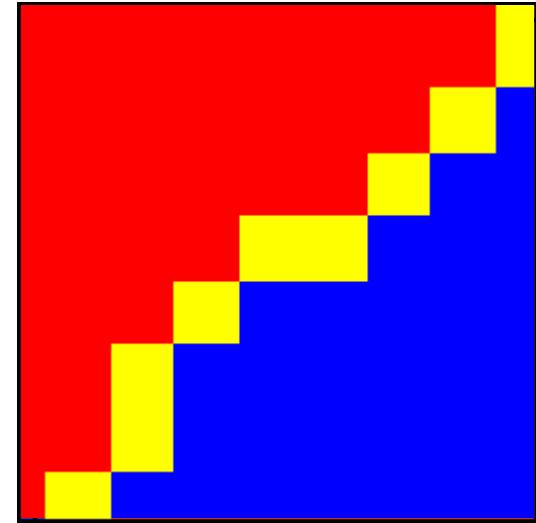
- Pivot – subtracted from the rows in the matrix



Pivot $p_{0.5}$



Pivot q_{12}



Pivot r_{24}

=> Pivot q (point with **middle** x or y coord.) is the best
But it is typically not used – pivot search is too complicated in comparison to the predicate itself

[Kettner]



DCGI



Epsilon tweaking



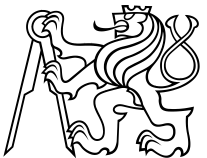
Epsilon tweaking

- Use tolerance $\varepsilon = 0.00005$ to 0.0001 for float



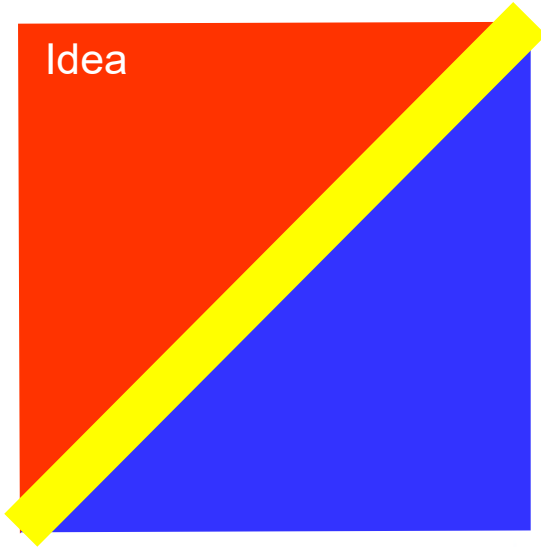
Epsilon tweaking

- Use tolerance $\varepsilon = 0.00005$ to 0.0001 for float
- Points are declared collinear if `float_orient` returns a value $\leq \varepsilon$ $0.5 + 2^{-23}$, the smallest repr. value $0.500\ 000\ 06$

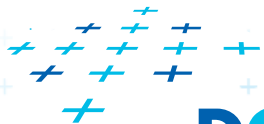


Epsilon tweaking

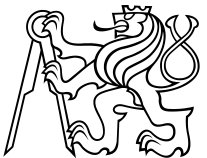
- Use tolerance $\varepsilon = 0.00005$ to 0.0001 for float
- Points are declared collinear if `float_orient` returns a value $\leq \varepsilon$ $0.5+2^{(-23)}$, the smallest repr. value $0.500\ 000\ 06$



Idea: boundary for ε

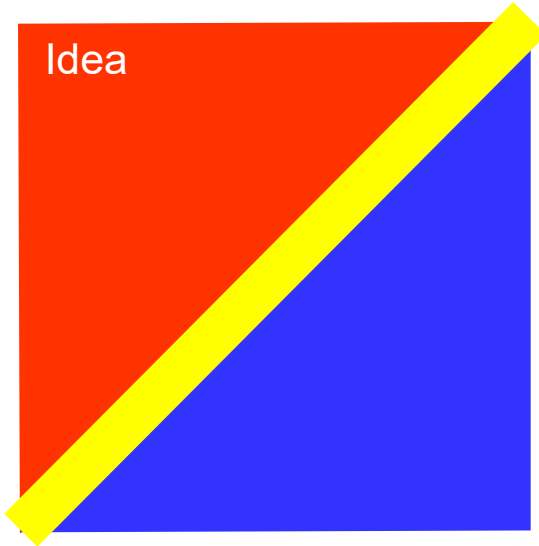


DCGI

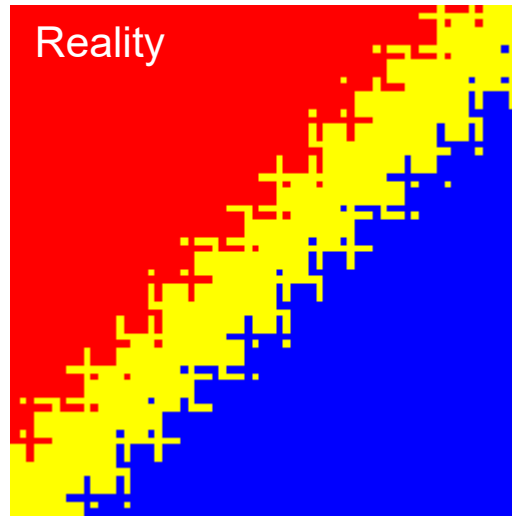


Epsilon tweaking

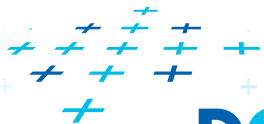
- Use tolerance $\varepsilon = 0.00005$ to 0.0001 for float
- Points are declared collinear if `float_orient` returns a value $\leq \varepsilon$ $0.5+2^{(-23)}$, the smallest repr. value $0.500\ 000\ 06$



Idea: boundary for ε

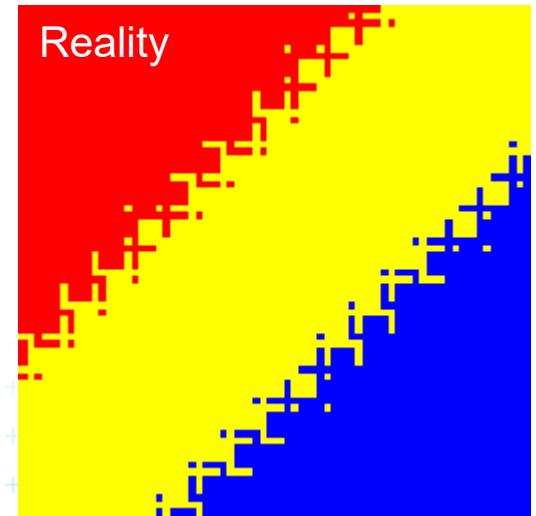
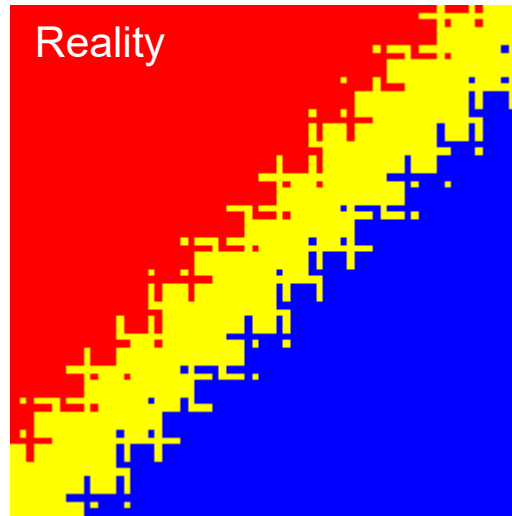
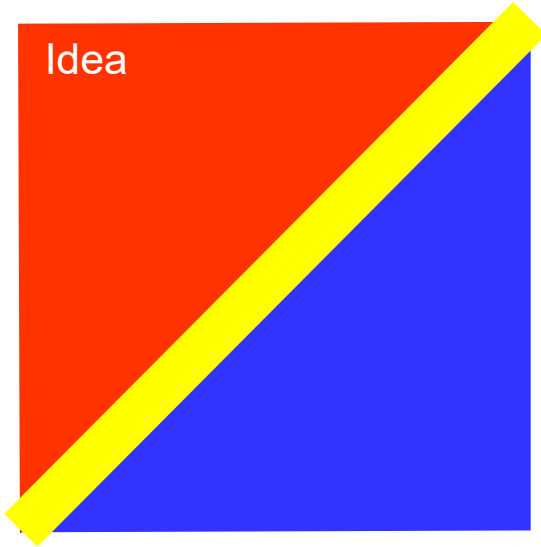


Boundary for $\varepsilon = 0.00005$



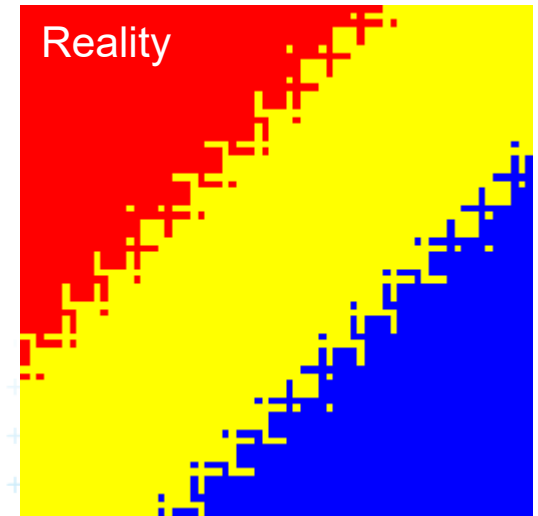
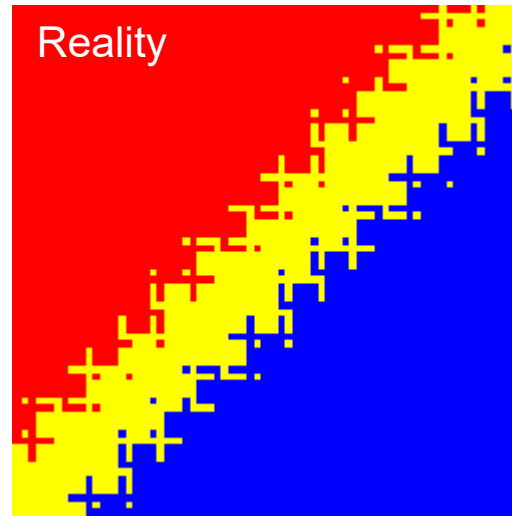
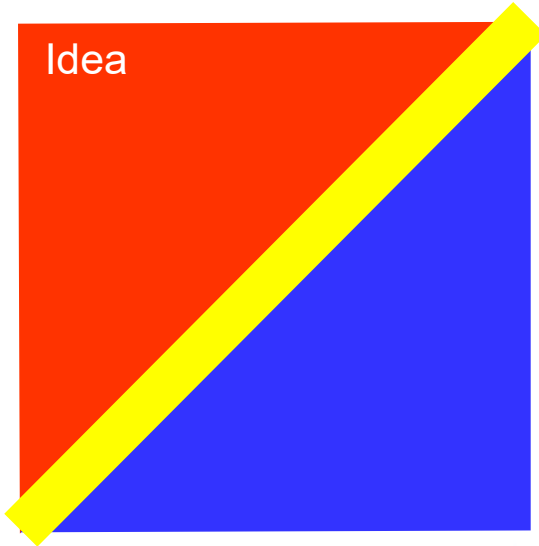
Epsilon tweaking

- Use tolerance $\varepsilon = 0.00005$ to 0.0001 for float
- Points are declared collinear if `float_orient` returns a value $\leq \varepsilon$ $0.5 + 2^{-23}$, the smallest repr. value $0.500\ 000\ 06$

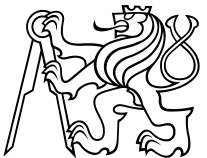


Epsilon tweaking

- Use tolerance $\varepsilon = 0.00005$ to 0.0001 for float
- Points are declared collinear if `float_orient` returns a value $\leq \varepsilon$ $0.5 + 2^{-23}$, the smallest repr. value $0.500\ 000\ 06$

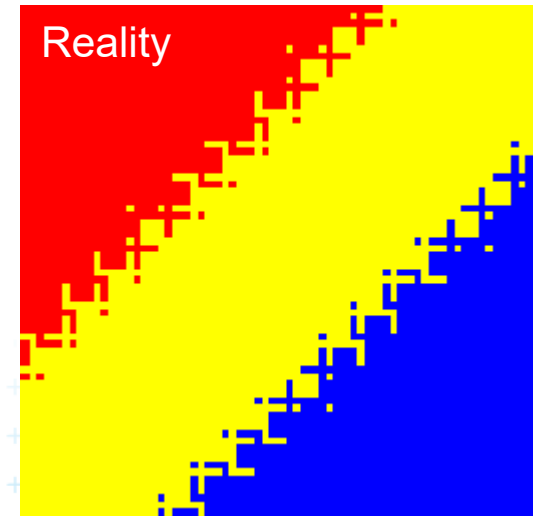
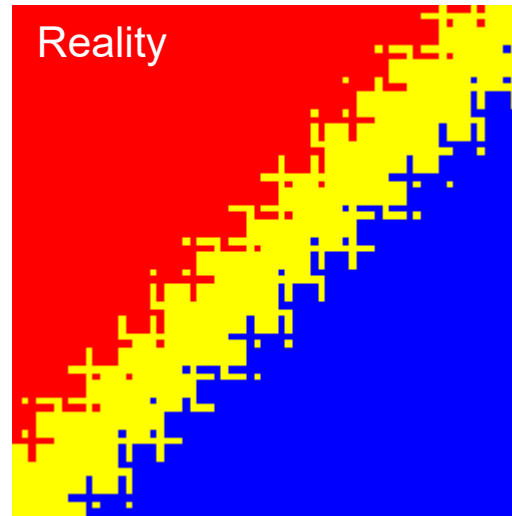
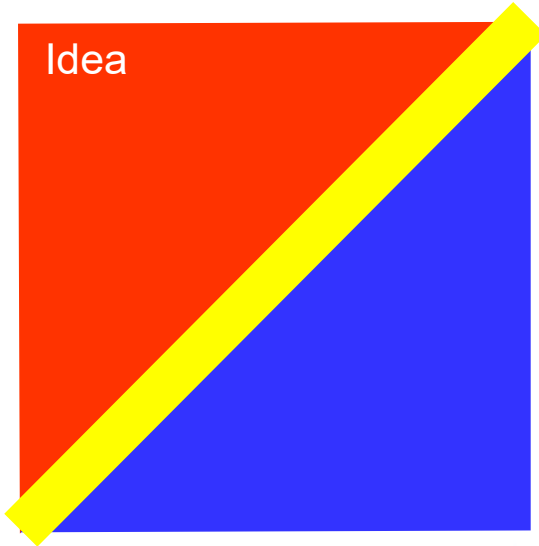


- Boundary is fractured as before, but wider

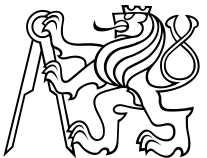


Epsilon tweaking – is the wrong approach

- Use tolerance $\varepsilon = 0.00005$ to 0.0001 for float
- Points are declared collinear if `float_orient` returns a value $\leq \varepsilon$ $0.5+2^{(-23)}$, the smallest repr. value $0.500\ 000\ 06$

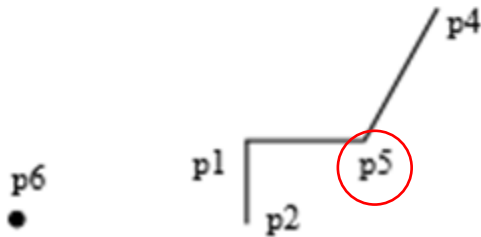
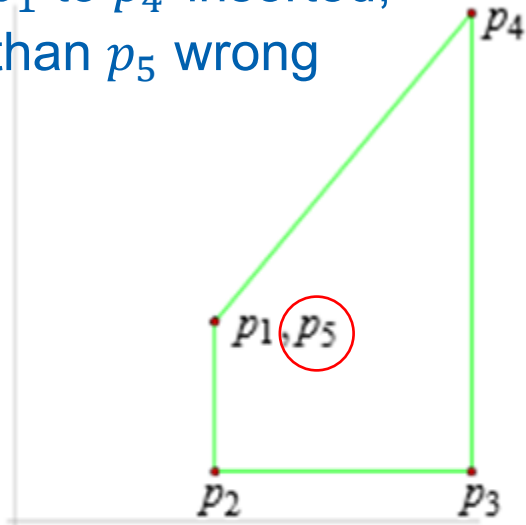


- Boundary is fractured as before, but wider

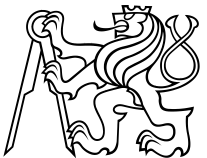
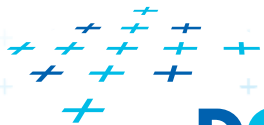


Consequences in convex hull algorithm

p_1 to p_4 inserted,
than p_5 wrong

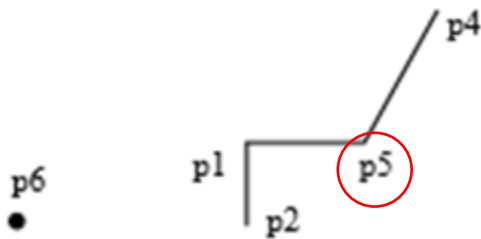
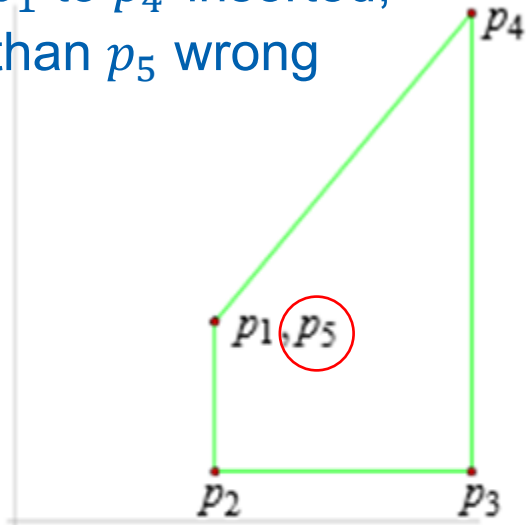


p_5 erroneously inserted



Consequences in convex hull algorithm

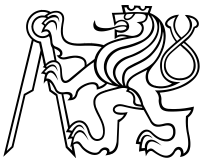
p_1 to p_4 inserted,
than p_5 wrong



[Kettner04]

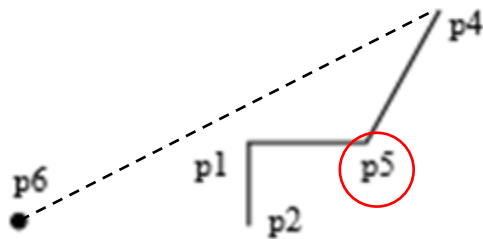
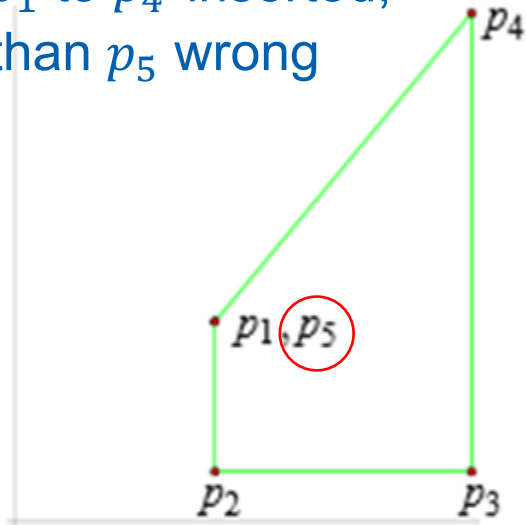
p_5 erroneously inserted

Inserting p_6 \Rightarrow



Consequences in convex hull algorithm

p_1 to p_4 inserted,
than p_5 wrong



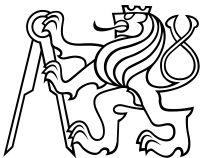
[Kettner04]

p_5 erroneously inserted

Inserting p_6 =>

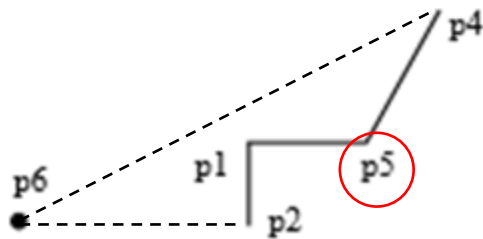
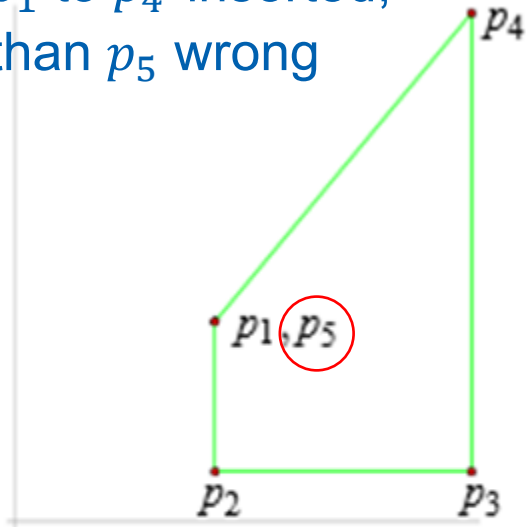


DCGI



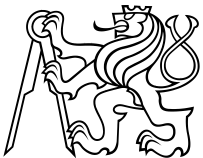
Consequences in convex hull algorithm

p_1 to p_4 inserted,
than p_5 wrong



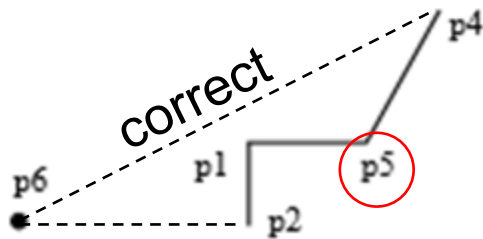
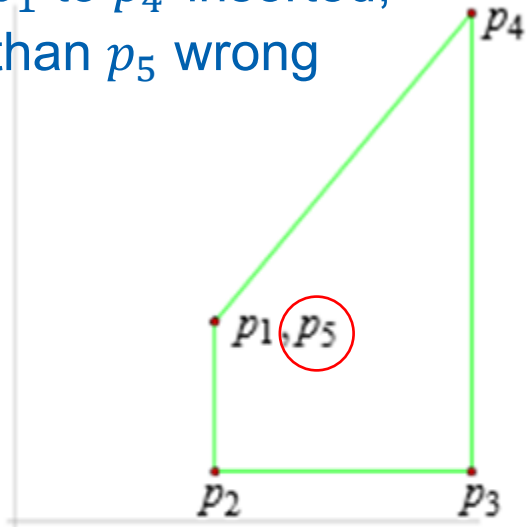
p_5 erroneously inserted

Inserting p_6 \Rightarrow



Consequences in convex hull algorithm

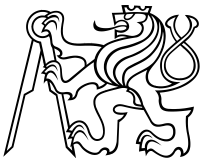
p_1 to p_4 inserted,
than p_5 wrong



[Kettner04]

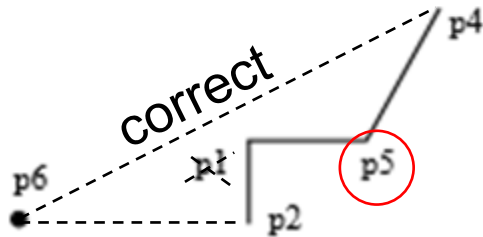
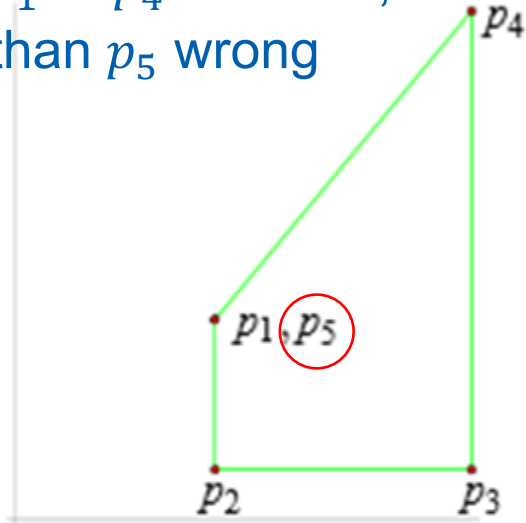
p_5 erroneously inserted

Inserting p_6 \Rightarrow



Consequences in convex hull algorithm

p_1 to p_4 inserted,
than p_5 wrong



[Kettner04]

p_5 erroneously inserted

Inserting p_6 \Rightarrow

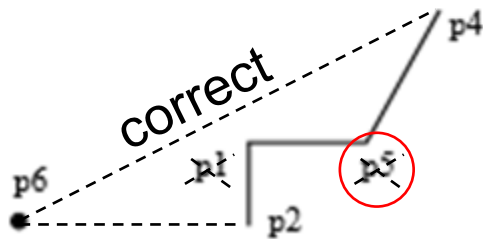
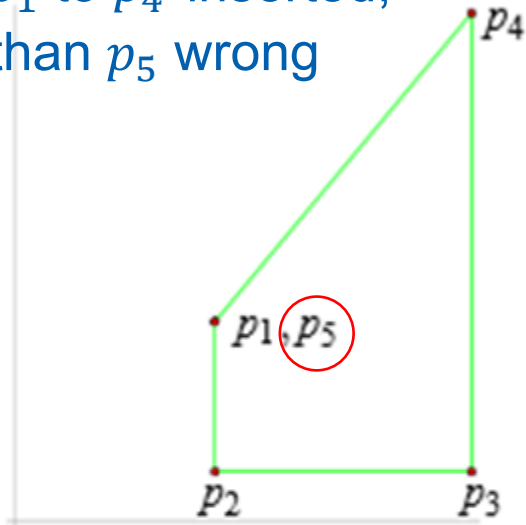


DCGI



Consequences in convex hull algorithm

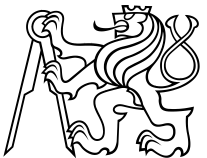
p_1 to p_4 inserted,
than p_5 wrong



[Kettner04]

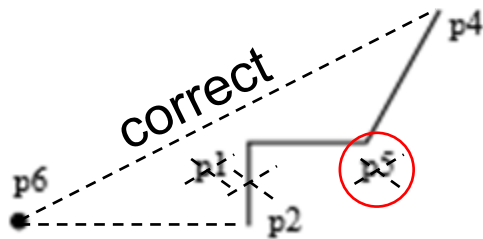
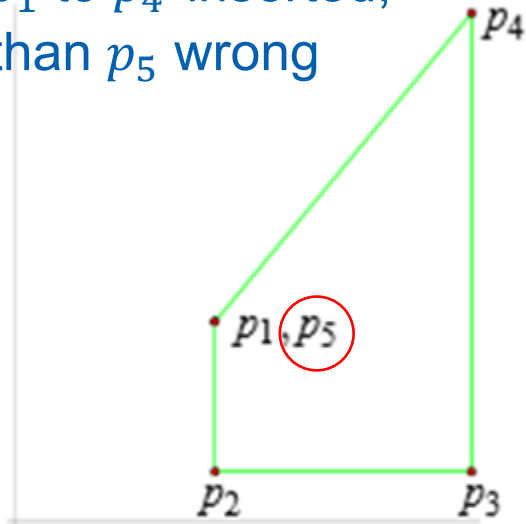
p_5 erroneously inserted

Inserting p_6 \Rightarrow



Consequences in convex hull algorithm

p_1 to p_4 inserted,
than p_5 wrong



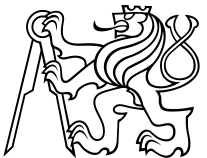
[Kettner04]

p_5 erroneously inserted

Inserting p_6 \Rightarrow

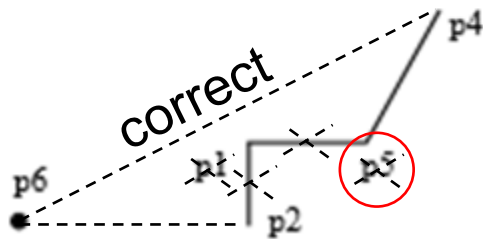
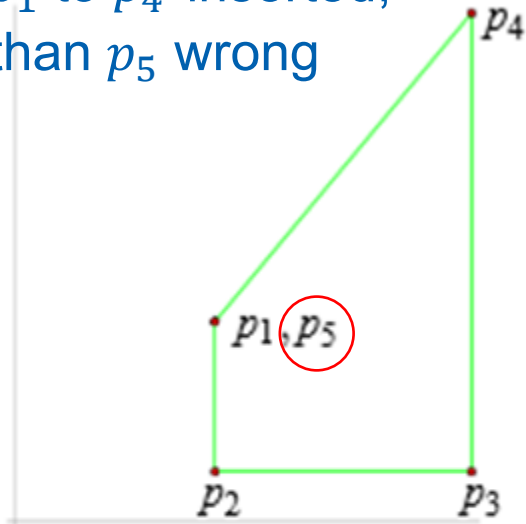


DCGI



Consequences in convex hull algorithm

p_1 to p_4 inserted,
than p_5 wrong



[Kettner04]

p_5 erroneously inserted

Inserting p_6 \Rightarrow

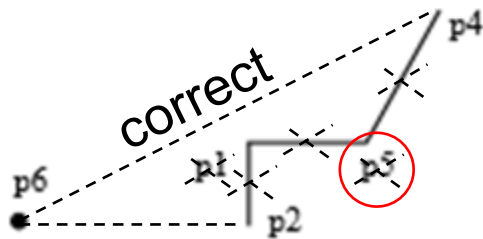
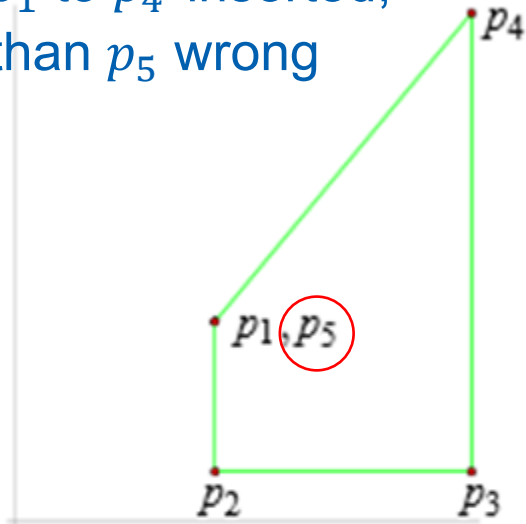


DCGI



Consequences in convex hull algorithm

p_1 to p_4 inserted,
than p_5 wrong



[Kettner04]

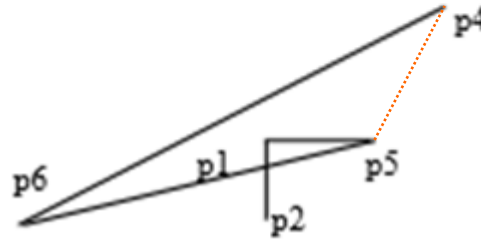
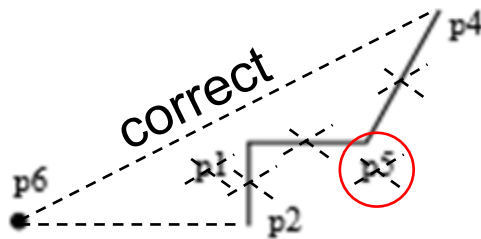
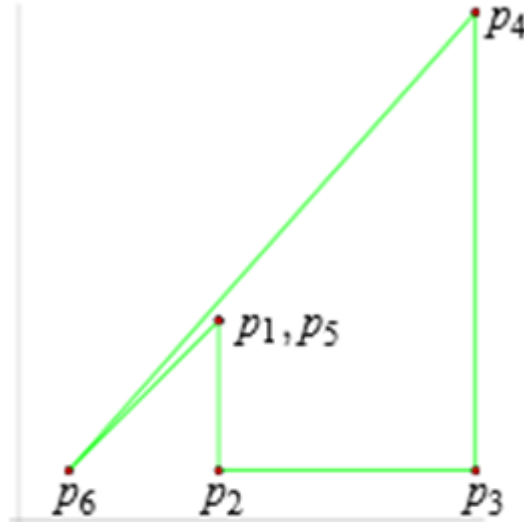
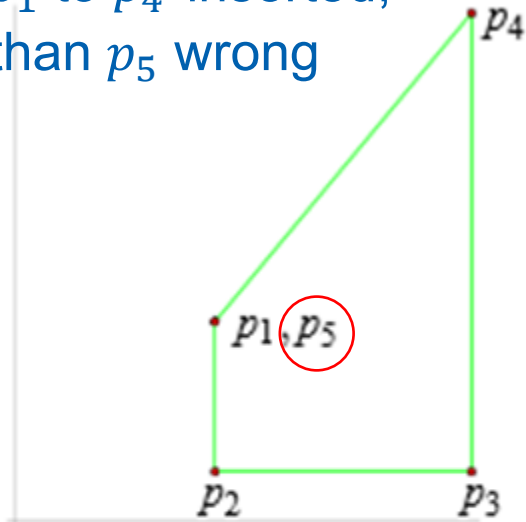
p_5 erroneously inserted

Inserting p_6 \Rightarrow



Consequences in convex hull algorithm

p_1 to p_4 inserted,
 than p_5 wrong



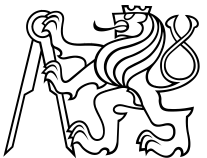
[Kettner04]

p_5 erroneously inserted

Inserting p_6 \Rightarrow

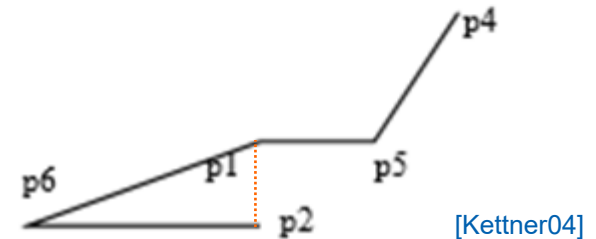
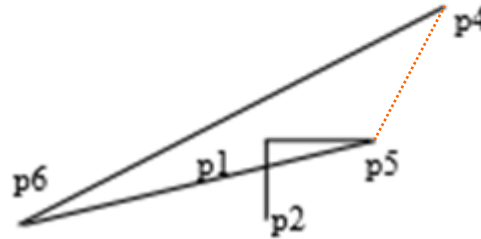
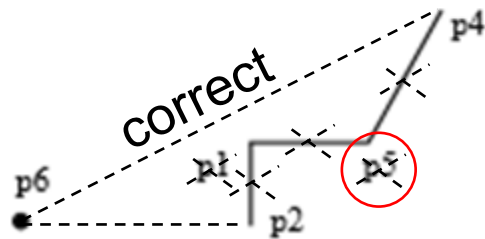
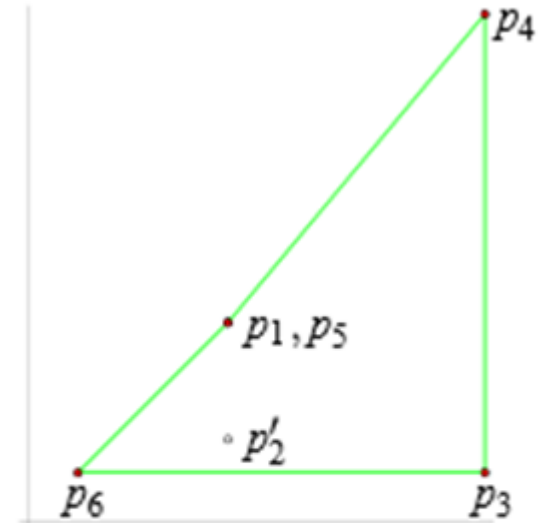
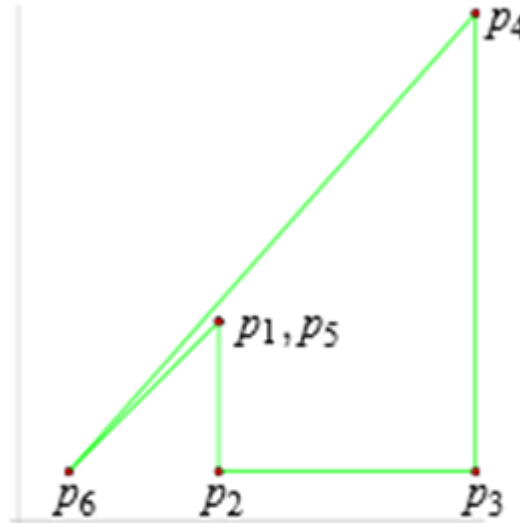
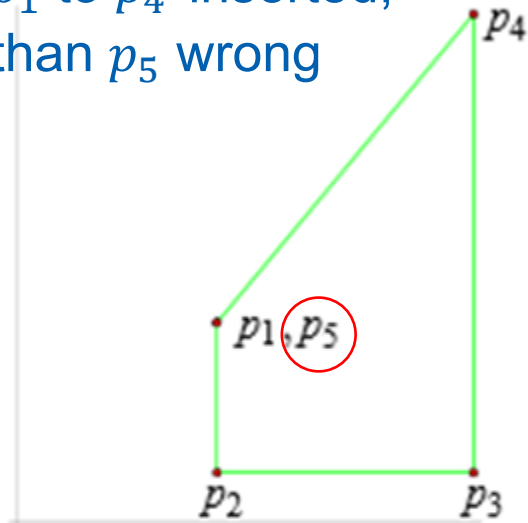
a) p_6 sees $p_4 p_5$ first

\Rightarrow forms $p_4 p_6 p_5$



Consequences in convex hull algorithm

p_1 to p_4 inserted,
than p_5 wrong



[Kettner04]

p_5 erroneously inserted

Inserting p_6 \Rightarrow

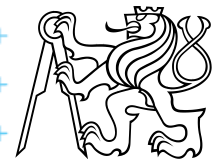
a) p_6 sees p_4p_5 first
 \Rightarrow forms $p_4p_6p_5$

b) p_6 sees p_1p_2 first
 \Rightarrow forms $p_1p_6p_2$



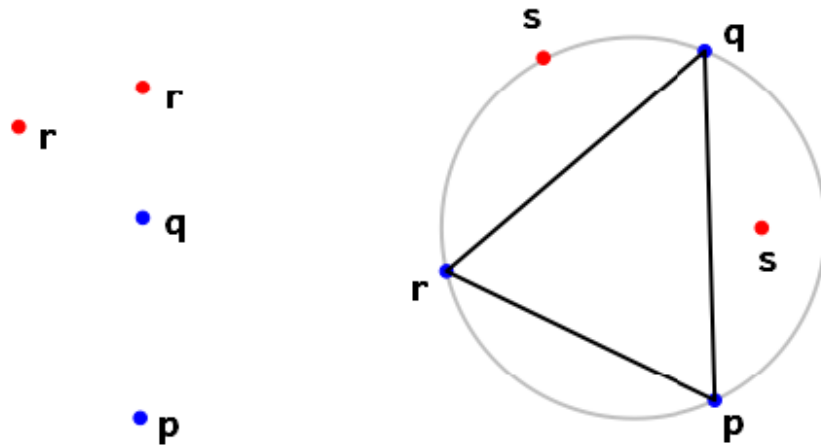
Solution

1. Use predicates, that always return the correct result -> Shewchuk, YAP, LEDA or CGAL
2. Change the algorithm to cope with floating point predicates but still return something *meaningful* (hard to define)
3. Perturb the input so that the floating point implementation gives the correct result on it



Exact geometric computing

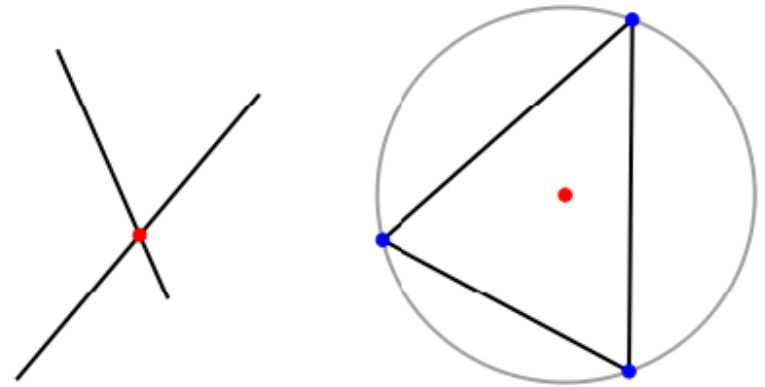
Predicates



orientation

in_circle

Constructions

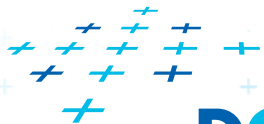


intersection

circumcenter

Use the quantized input points
Predicate may be computed precisely
⇒ Shewchuk

Create new points
Error increases each multiplication
⇒ double the precision

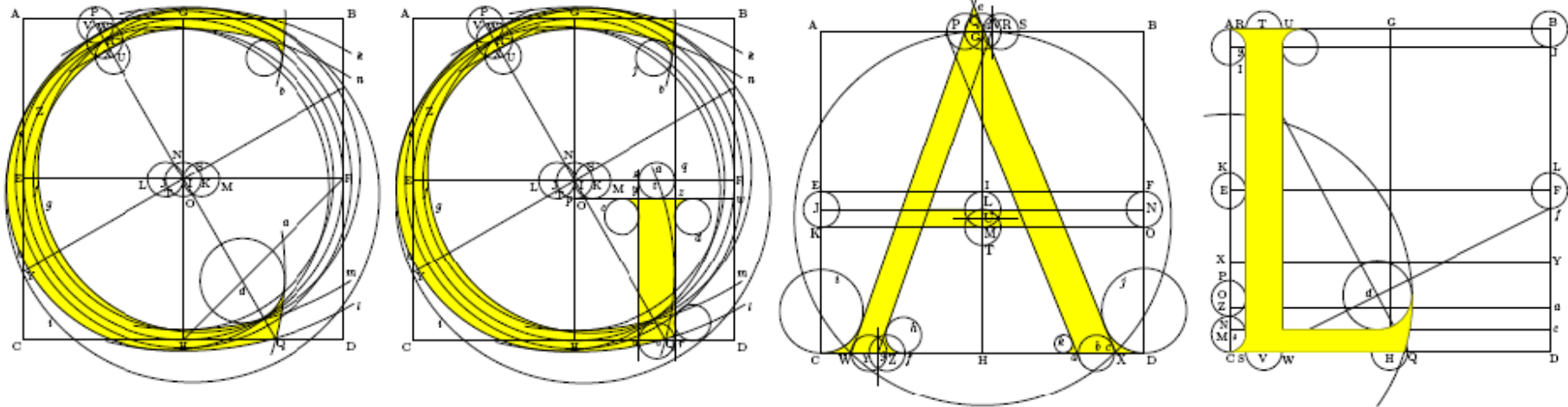


Common solution

- Ignore the errors in classics
 - good for for interactive apps (sound, video)
- Precise algorithms
 - for critical applications
 - need more memory and time
- Combine classic & precise alg.
 - adaptive approach
 - detect places that need precise alg.
 - Before the computations start
 - During computations - after error increase recompute precise

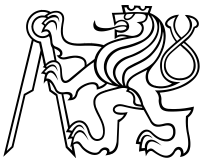


8. CGAL

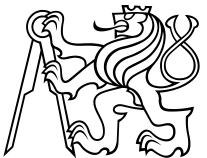


Computational Geometry Algorithms Library

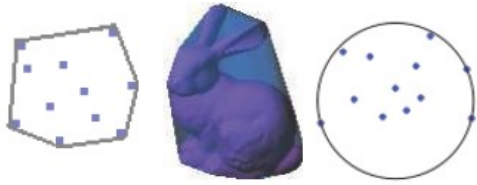
Slides from [siggraph2008-CGAL-course]



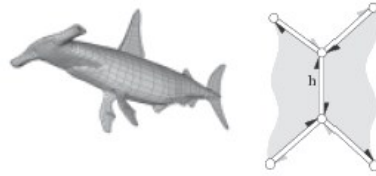
- Large library of geometric algorithms
 - Robust code, huge amount of algorithms
 - Users can concentrate on their own domain
- Open source project
 - Institutional members
(Inria, MPI, Tel-Aviv U, Utrecht U, Groningen U, ETHZ, Geometry Factory, FU Berlin, Forth, U Athens)
 - 500,000 lines of C++ code
 - 10,000 downloads/year (+ Linux distributions)
 - 20 active developers
 - 12 months release cycle



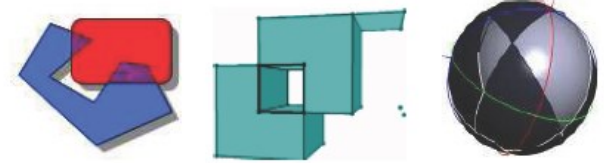
CGAL algorithms and data structures



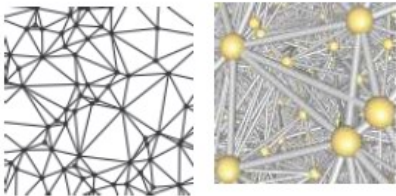
Bounding Volumes



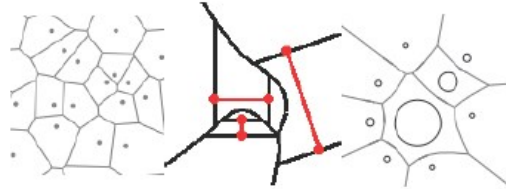
Polyhedral Surface



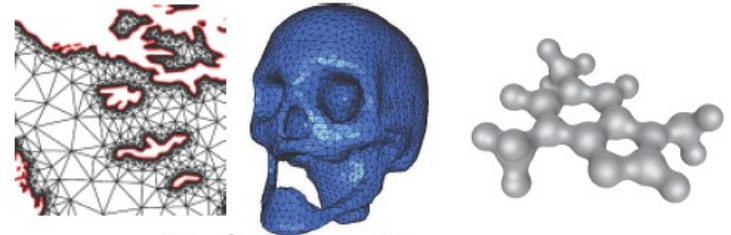
BooleanOperations



Triangulations



Voronoi Diagrams



Mesh Generation



Subdivision



Simplification



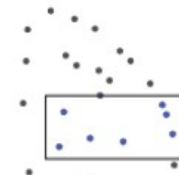
Parametrisation



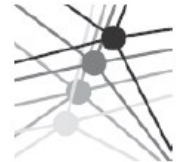
Streamlines



Ridge
Detection



Neighbor
Search



Kinetic
Datastructures



Lower Envelope



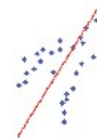
Arrangement



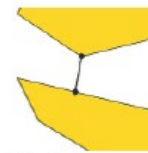
Intersection
Detection



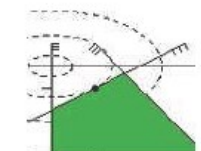
Minkowski
Sum



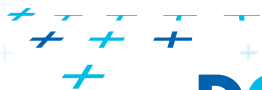
PCA



Polytope
distance



QP Solver



DCGI



CGAL Geometric Kernel (see [Hert] for details)

■ Encapsulates

- the representation of geometric objects
- and the geometric operations and predicates on these objects

■ CGAL provides kernels for

- Points, Predicates, and Exactness
- Number Types
- Cartesian Representation
- Homogeneous Representation



Points, predicates, and Exactness

```
#include "tutorial.h"
#include <CGAL/Point_2.h>
#include <CGAL/predicates_on_points_2.h>
#include <iostream>

int main() {
    Point p( 0.1, 0.2);
    Point q( 1.3, 1.7);
    Point r( 2.2, 6.8);
    switch ( CGAL::orientation( p, q, r)) {
        case CGAL::LEFTTURN:    std::cout << "Left turn.\n"; break;
        case CGAL::RIGHTTURN:   std::cout << "Right turn.\n"; break;
        case CGAL::COLLINEAR:   std::cout << "Collinear.\n"; break;
    }
    return 0;
}
```



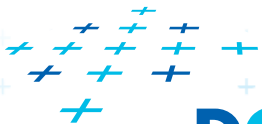
Number Types

Precision
x
slow-down

- Builtin: `double`, `float`, `int`, `long`, ...
- CGAL: `Filtered_exact`, `Interval_nt`, ...
- LEDA: `leda_integer`, `leda_rational`, `leda_real`, ...
- Gmpz: `CGAL::Gmpz`
- others are easy to integrate

Coordinate Representations

- Cartesian $p = (x, y)$: `CGAL::Cartesian<Field_type>`
- Homogeneous $p = (\frac{x}{w}, \frac{y}{w})$: `CGAL::Homogeneous<Ring_type>`

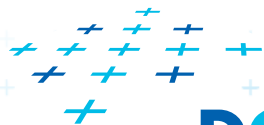


Cartesian with double

```
#include <CGAL/Cartesian.h>
```

```
#include <CGAL/Point_2.h>
```

```
int main() {  
    CGAL::Point_2< CGAL::Cartesian<double> > p( 0.1, 0.2);  
    ...  
}
```



Cartesian with double

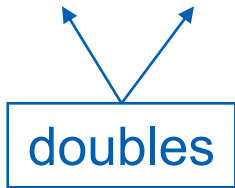
```
#include <CGAL/Cartesian.h>
```

```
#include <CGAL/Point_2.h>
```

```
typedef CGAL::Cartesian<double> Rep;
```

```
typedef CGAL::Point_2<Rep> Point;
```

```
int main() {  
    Point p( 0.1, 0.2);  
    ... ;  
}
```



Cartesian with `Filtered_exact` and `leda_real`

```
#include <CGAL/Cartesian.h>
#include <CGAL/Arithmetic_filter.h>
#include <CGAL/leda_real.h>
#include <CGAL/Point_2.h>
```

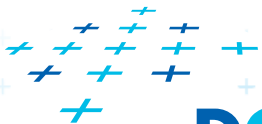
```
typedef CGAL::Filtered_exact<double, leda_real> NT;
typedef CGAL::Cartesian<NT> Rep;
typedef CGAL::Point_2<Rep> Point;
```

Number type

```
int main() {
    Point p( 0.1, 0.2);
    ...
}
```

Filtered exact doubles

A two-line declaration changes the precision of all computations



Exact orientation test – homogeneous rep.

```
#include <CGAL/Homogeneous.h>
#include <CGAL/Point_2.h>
#include <CGAL/predicates_on_points_2.h>
#include <iostream>

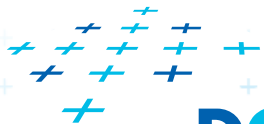
typedef CGAL::Homogeneous<long>
typedef CGAL::Point_2<Rep>

int main() {
    Point p( 1,  2, 10);
    Point q( 13, 17, 10);
    Point r( 22, 68, 10);
    switch ( CGAL::orientation( p, q, r)) {
        case CGAL::LEFTTURN:    std::cout << "Left turn.\n"; break;
        case CGAL::RIGHTTURN:   std::cout << "Right turn.\n"; break;
        case CGAL::COLLINEAR:   std::cout << "Collinear.\n"; break;
    }
}
```

Rep;
Point;

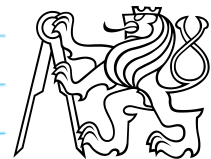
A single-line declaration changes the precision of all computations

Homogeneous points



9 References – for the lectures

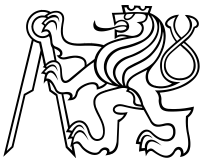
- Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars: **Computational Geometry: Algorithms and Applications, Springer-Verlag, 3rd rev. ed. 2008. 386 pages, 370 fig. ISBN: 978-3-540-77973-5**
<http://www.cs.uu.nl/geobook/>
- **David Mount: Computational Geometry Lecture Notes for Spring 2020, University of Maryland**
<http://www.cs.umd.edu/class/spring2020/cmsc754/Lects/cmsc754-spring2020-lects.pdf>
- **Franko P. Preparata, Michael Ian Shamos: Computational Geometry. An Introduction. Berlin, Springer-Verlag, 1985**
- **Joseph O'Rourke: .: Computational Geometry in C, Cambridge University Press, 1993, ISBN 0-521- 44592-2**
<http://maven.smith.edu/~orourke/books/compgeom.html>
- **Ivana Kolingerová: Aplikovaná výpočetní geometrie, Přednášky, MFF UK 2008**
- **Kettner et al. Classroom Examples of Robustness Problems in Geometric Computations, CGTA 2006,**
http://www.mpi-inf.mpg.de/~kettner/pub/nonrobust_cgta_06.pdf



9.1 References – CGAL

CGAL

- www.cgal.org
- Kettner, L.: Tutorial I: Programming with CGAL
- Alliez, Fabri, Fogel: Computational Geometry Algorithms Library, SIGGRAPH 2008
- Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Michael Seel. **An adaptable and extensible geometry kernel.** *Computational Geometry: Theory and Applications*, 38:16-36, 2007.
[doi:[10.1016/j.comgeo.2006.11.004](https://doi.org/10.1016/j.comgeo.2006.11.004)]



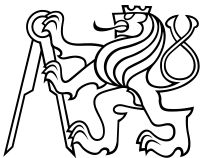
9.2 Useful geometric tools

- OpenSCAD - *The Programmers Solid 3D CAD Modeler*,
<http://www.openscad.org/>
- J.R. Shewchuk - *Adaptive Precision Floating-Point Arithmetic and Fast Robust Predicates*, Effective implementation of Orientation and InCircle predicates <http://www.cs.cmu.edu/~quake/robust.html>
- *OpenMESH* - A generic and efficient polygon mesh data structure,
<https://www.openmesh.org/>
- *VCG Library* - The Visualization and Computer Graphics Library,
<http://vcg.isti.cnr.it/vcglib/>
- *MeshLab* - A processing system for 3D triangular meshes -
<https://sourceforge.net/projects/meshlab/?source=navbar>
- Petr Lobaz: Geometrické výpočty a čísla s plovoucí desetinnou čárkou,
Technická zpráva No. DCSE/TR-2014-04, University of West Bohemia,
2014, <https://otik.zcu.cz/bitstream/11025/21539/1/Lobaz.pdf>



9.3 Collections of geometry resources

- D. Eppstein, *Geometry in Action*,
<http://www.ics.uci.edu/~eppstein/geom.html>.
- Jeff Erickson, *Computational Geometry Pages*,
<https://jeffe.cs.illinois.edu/compgeom/software.html>



10. Computational geom. course summary

- Gives an overview of geometric algorithms
- Explains their complexity and limitations
- Different algorithms for different data
- We focus on
 - discrete algorithms and precise numbers and predicates
 - principles more than on precise mathematical proofs

