DCGI

DEPARTMENT OF COMPUTER GRAPHICS AND INTERACTION

COMPUTATIONAL GEOMETRY INTRODUCTION

PETR FELKEL

FEL CTU PRAGUE

felkel@fel.cvut.cz

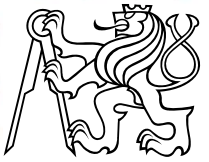
http://service.felk.cvut.cz/courses/X39VGE **A4M39VG**

Based on [Berg] and [Kolingerova]

Version from 29.9.2011

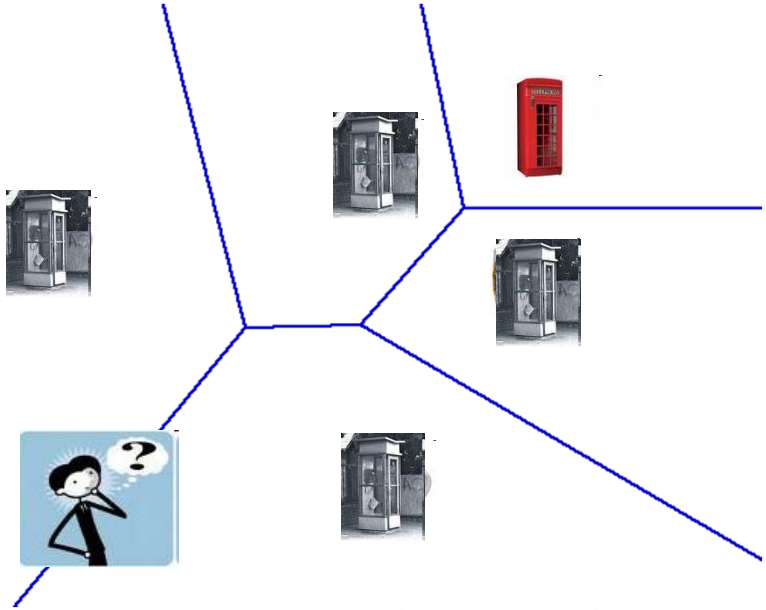
Computational Geometry

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

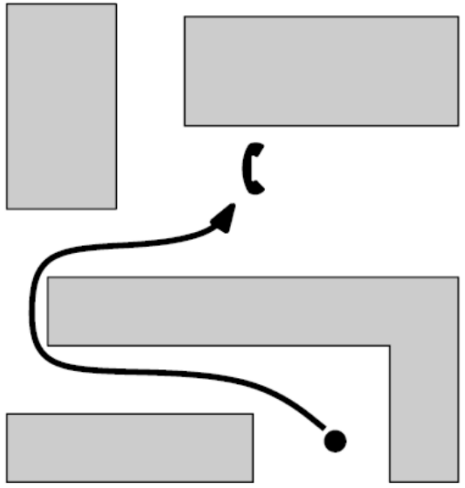


1. What is Computational Geometry?

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



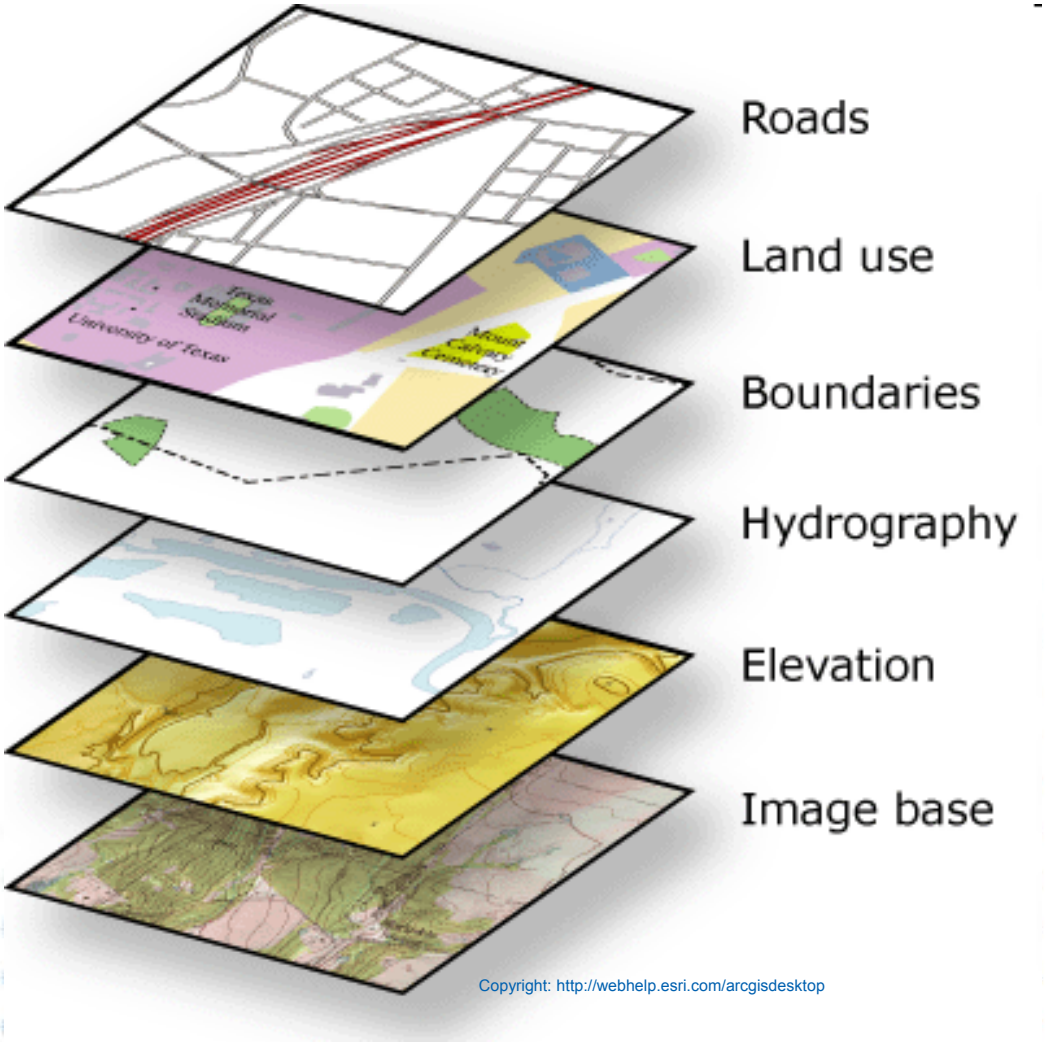
Ex 2: How to get there?



[Berg]

1.1 What is Computational Geometry? (…)

- Ex 3: Map overlay



Copyright: http://webhelp.esri.com/arcgisdesktop

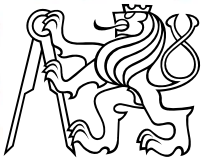
1.2 What is Computational Geometry? (…)

- 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? (…)

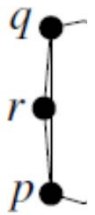
- Computational geometry
  = systematic study of algorithms and data structures for geometric objects (points, lines, line segments, n-gons,…) with focus on exact algorithms that are asymptotically fast

  – "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? (…)

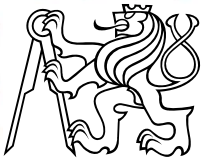
- Problems:
  - Degenerate cases (points on line, with same *x*,…)
    - ignore first, include later
  - Robustness - correct algorithm but not robust
    - Limited numerical precision of real arithmetic
    - Inconsistent *eps* tests (a=b, b=c, but a≠c)



- Nowadays:
  - focus on practical implementations, not just on asymptotically fastest algorithms
  - nearly correct result is better than nonsense or crash

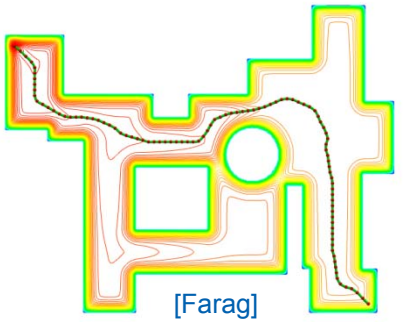
2. Why to study computational geometry?

- Graphics- and Vision- Engineer should know it („DSA in n^{th}-Dimension“)
- Set of ready to use tools
- You will know much more to choose from
- Typical teaching method: statement-proof
- FEL:
  - practical algorithms and their complexity
  - Practical programing using a geometric library
- Is it OK for you?

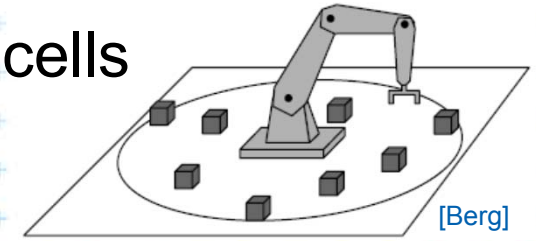


3. Typical application domains

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

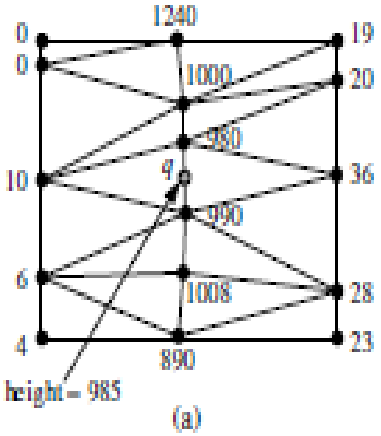

[Farag]

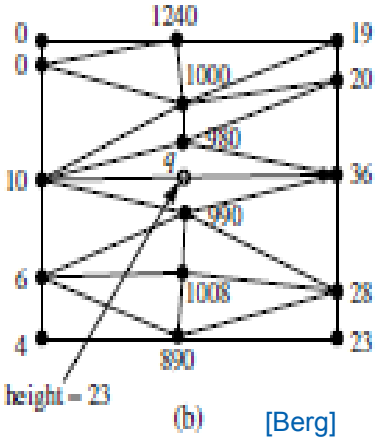
- Robotics
  - Motion planning (find path - environment with obstacles)
  - Task planning (motion + planning order of subtasks)
  - Design of robots and working cells


[Berg]

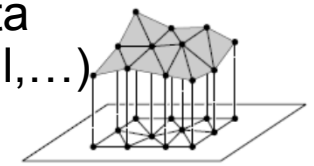
3.1 Typical application domains (…)

- 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…





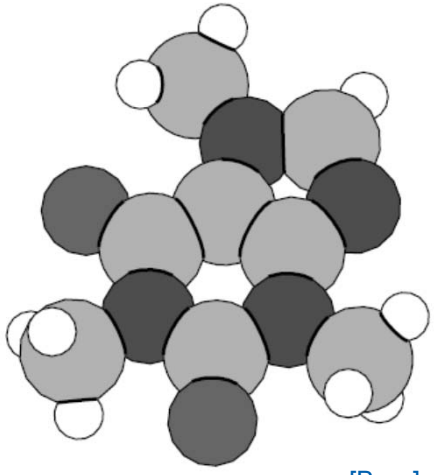
[Berg]



- CAD/CAM
  - Intersections and unions of objects
  - Visualization and tests without need to build a prototype
  - Manufacturability

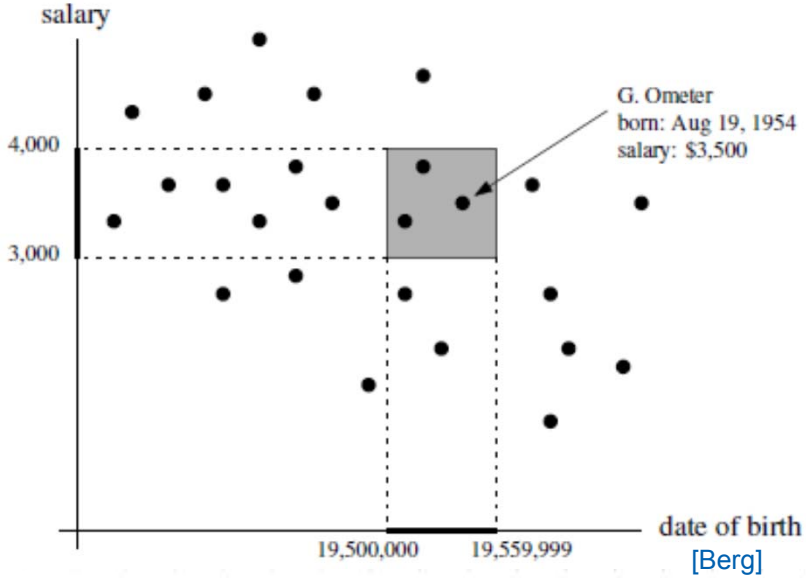
3.2 Typical application domains (…)

- Other domains
  - Molecular modeling
  - DB search
  - IC design

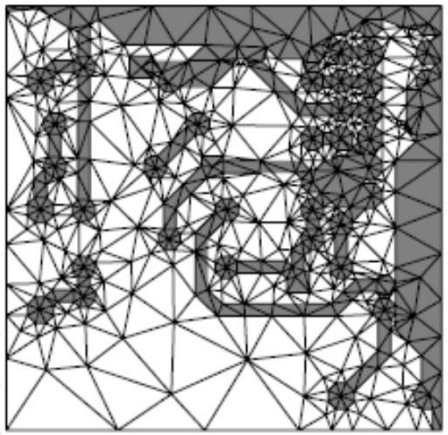


[Berg]

caffeine



[Berg]

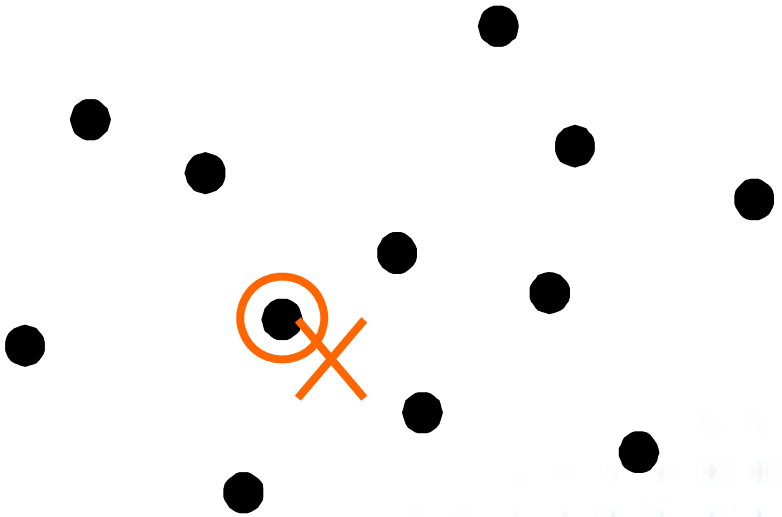


[Berg]

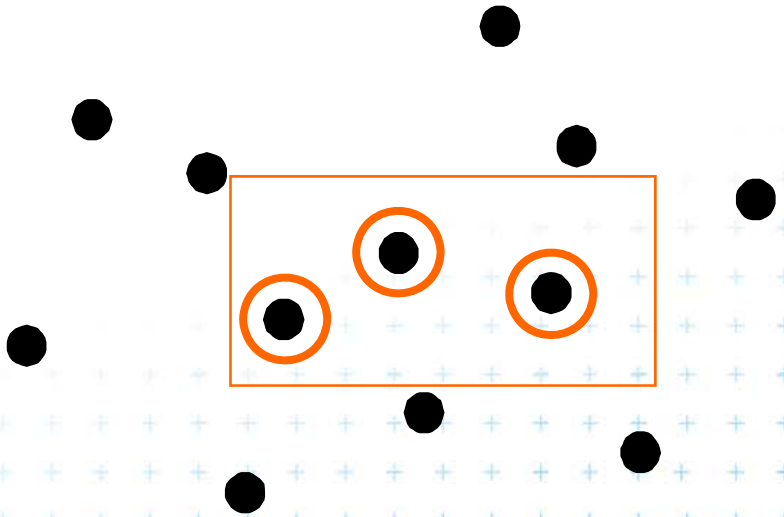
4. Typical tasks in CG

- Geometric searching - fast location of :

The nearest neighbor

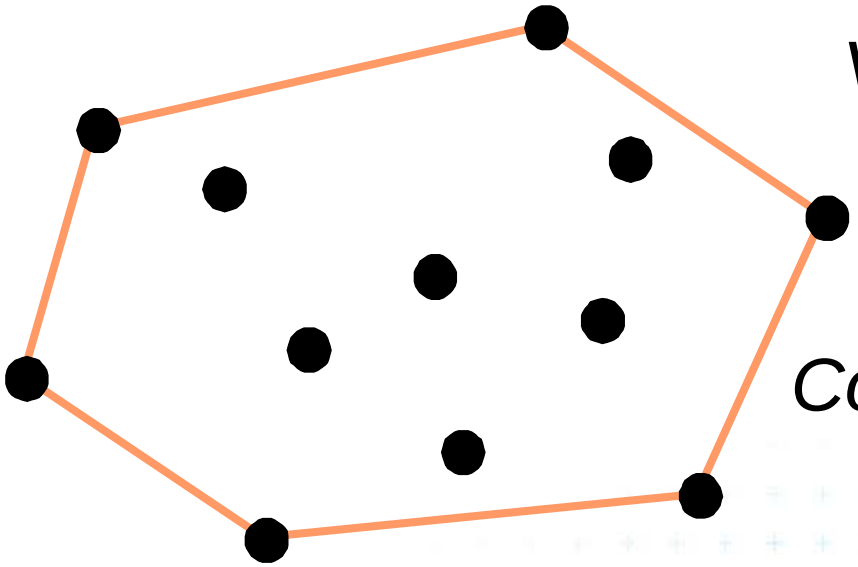


Points in given range (*range query*)



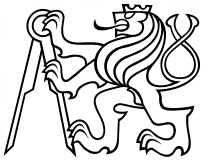
4.1 Typical tasks in CG

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



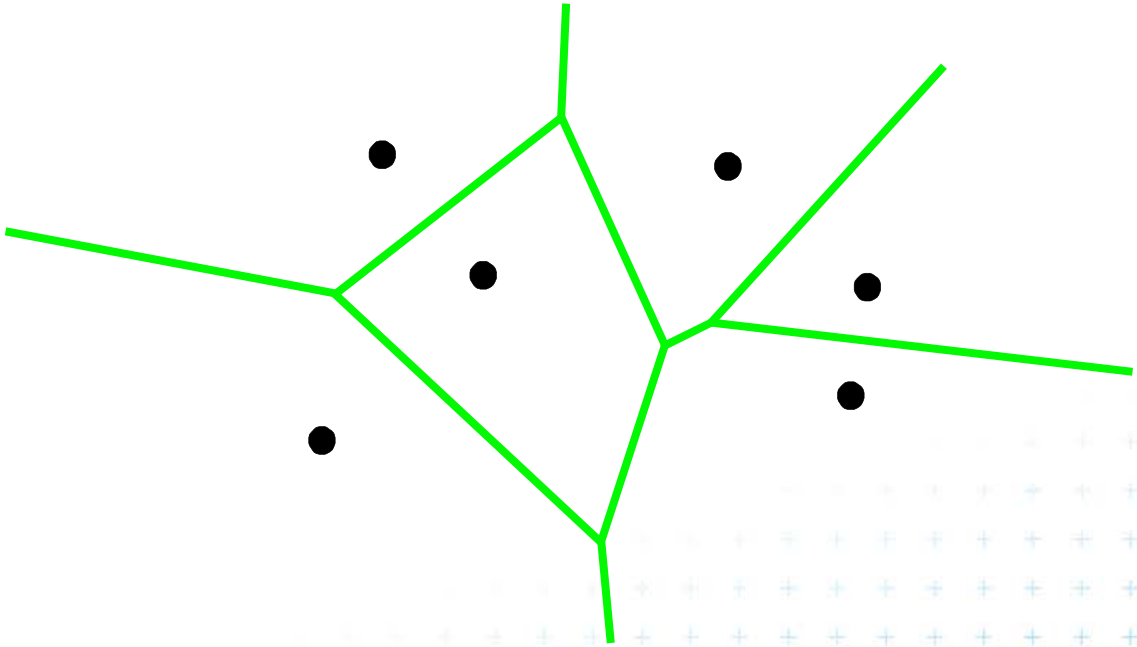
V – set of points

Convex Hull CH(V)



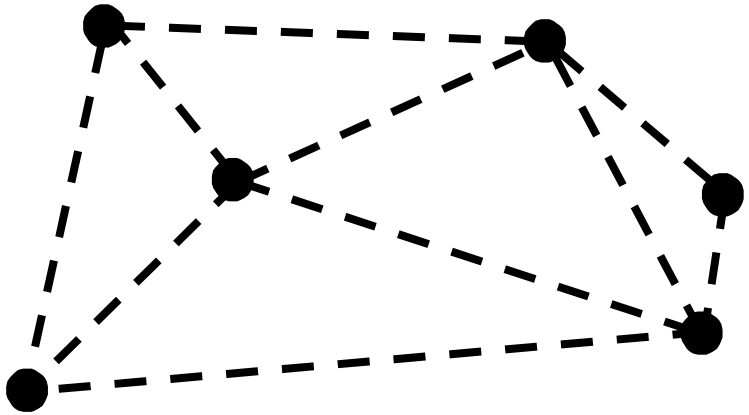
4.2 Typical tasks in CG

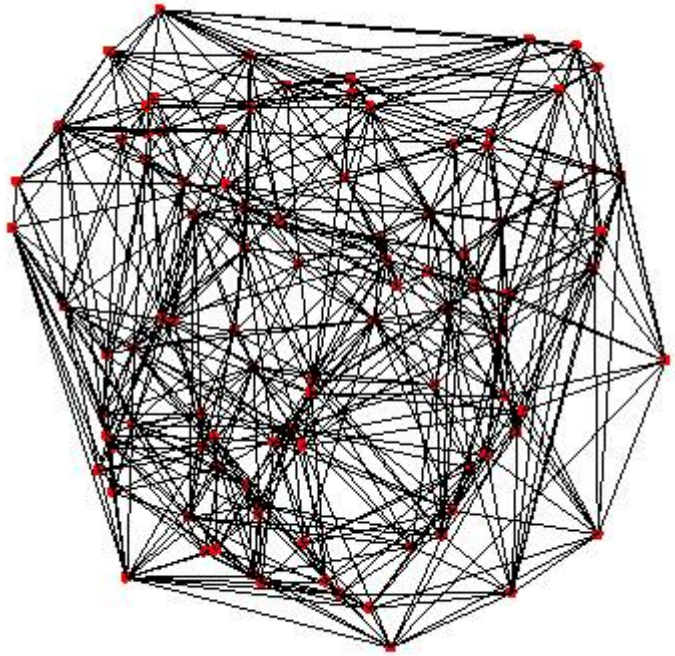
- Voronoi diagrams
  - Space (plane) partitioning into regions whose points are nearest to the given primitive (most usually a point)



4.3 Typical tasks in CG

- Planar triangulations and space tetrahedronization of given point set

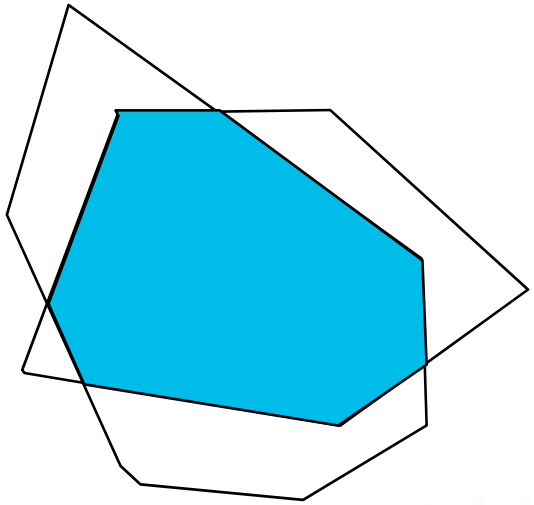


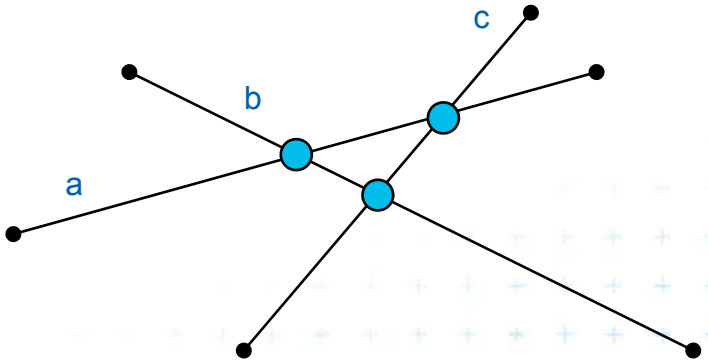


[Maur]

4.4 Typical tasks in CG

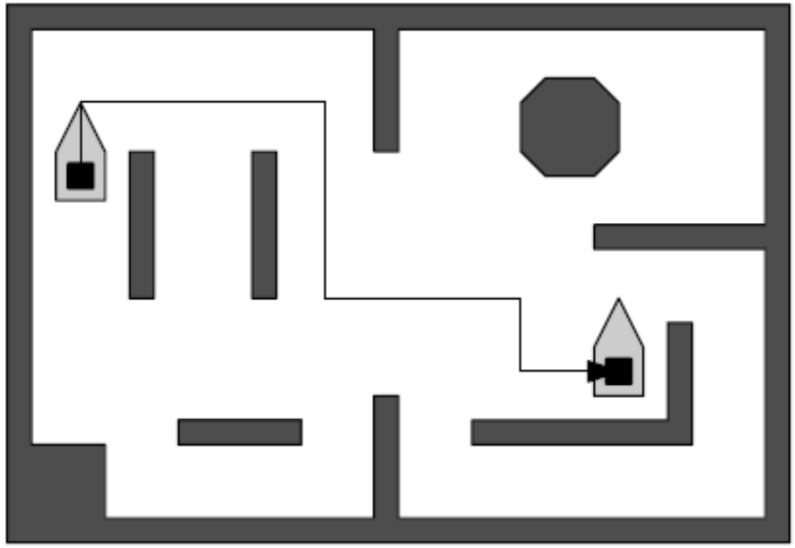
- Intersection of objects
  - Detection of common parts of objects
  - Usually linear (line segments, polygons, n-gons,…)





4.5 Typical tasks in CG

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



[Berg]

5. Complexity of algorithms

- We need a measure for comparison of algorithms
  - Independent on computer HW and prog. language
  - Dependent on problem size *n*
  - Describing the behavior of algorithm for different data
- Running time, preprocessing time, memory size
  - Asymptotical analysis – **O(g(*n*))**, $\mathtt{Z}(g(n))$, $\mathtt{T}(g(n))$
  - Measurement on real data
- Differentiate:
  - complexity of the algorithm and
  - complexity of the problem
    - given by number of edges, vertices, faces,…
    - 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

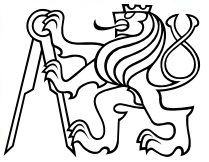
6. Programming techniques (paradigms) of CG

- 3 phases of a geometric algorithm development

1. Ignore all degeneracies and design an algorithm

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, y,…, or 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

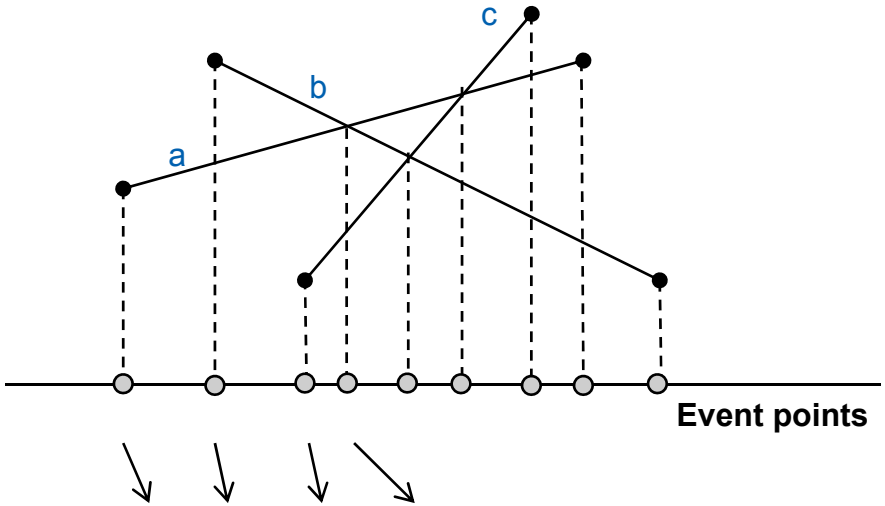
```
DivideAndConquer(S)
1. If known solution then return it
2. else
3.    Split input S to k distinct subsets Si
4.    Foreach i call DivideAndConquer(Si)
5.    Merge the results and return the solution
```

- 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

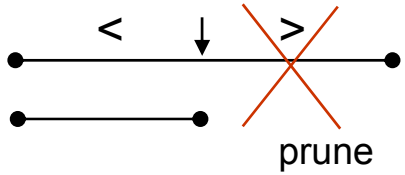
- Split the space by a hyperplane (2D: sweep line)
  - "Left" subspace – solution known
  - "Right" subspace – solution unknown
- Stop in event points and update the status
- Data structures:
  - **Event points** – points, where to stop the sweep line and update the status, sorted
  - **Status** – state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

6.3b Sweep-line algorithm

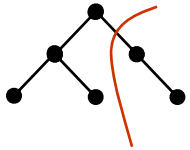


6.4 Prune and search

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



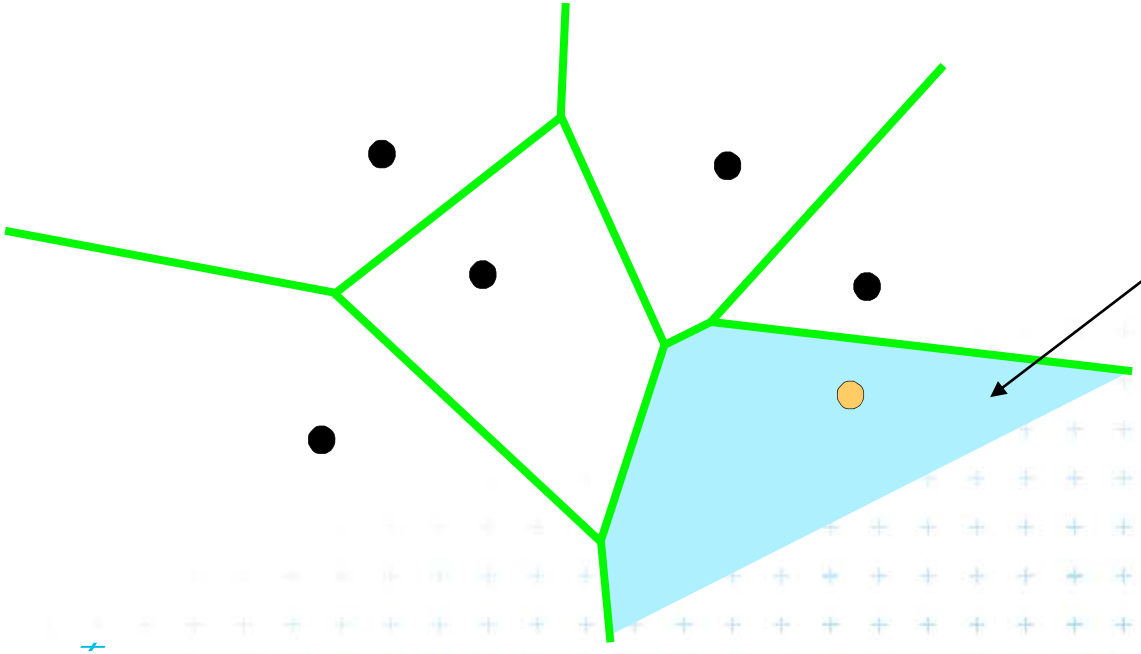
  - Search trees



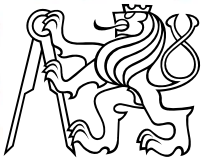
  - Back-tracking (stop if solution worse than current optimum)

6.5 Locus approach

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

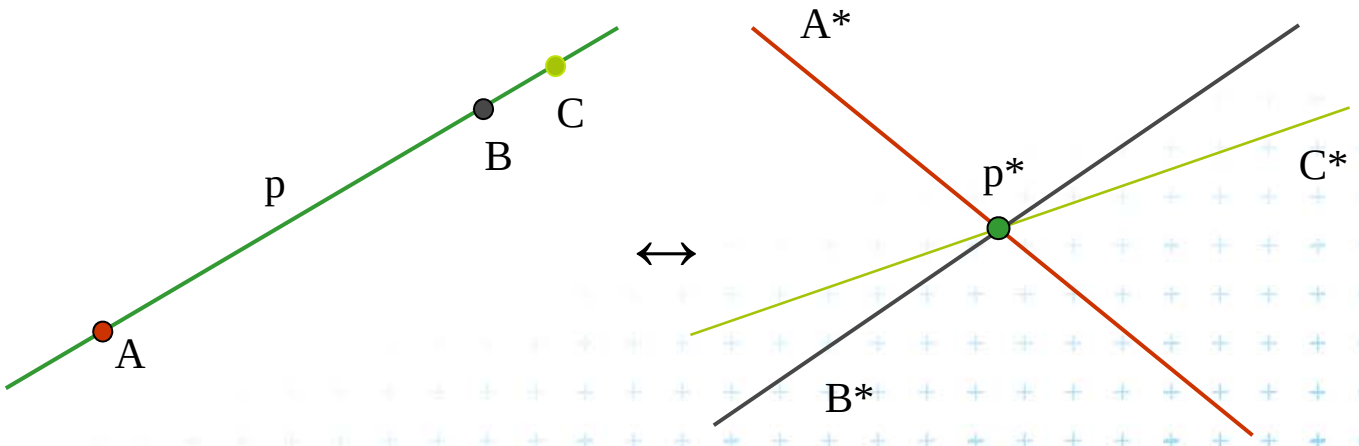


Region of the constant answer: All points in this region are nearest to the yellow point



6.6 Dualisation

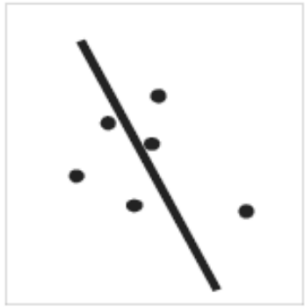
- Use geometry transform to change the problem into another that can be solved more easily
- Points ↔ hyper planes
  - Preservation of incidence (A μ p ² p*μ A*)
- Ex. 2D: determine if 3 points lie on a common line



6.7 Combinatorial analysis

= The branch of mathematics which studies the number of different ways of arranging things

- Ex. How many subdivisions of a point set can be done by one line?

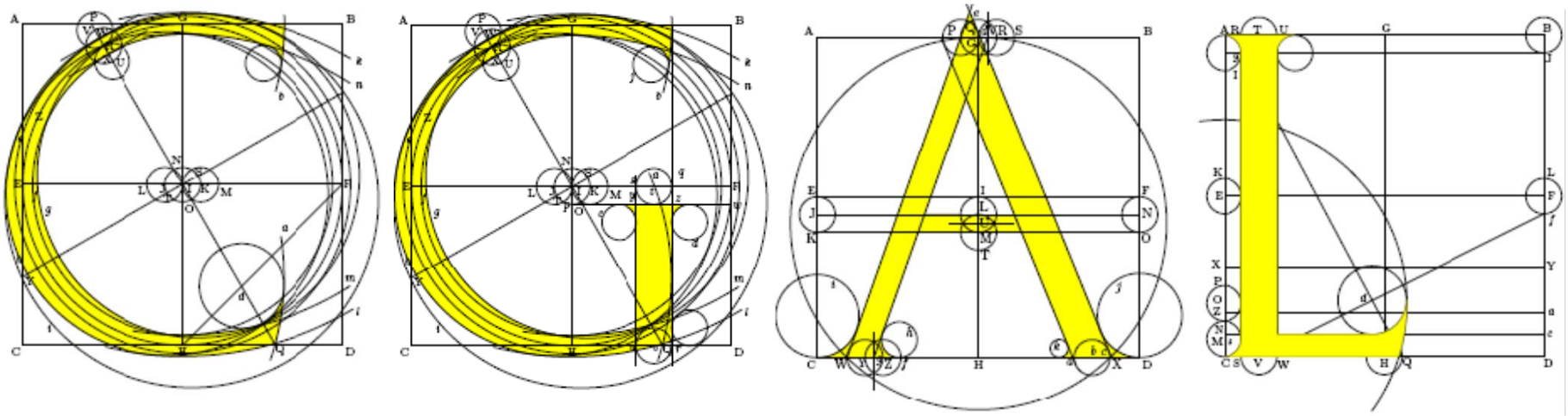


6.8 New trends in Computational geometry

- 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 (LEDA, CGAL, ...

DCGI

7. CGAL



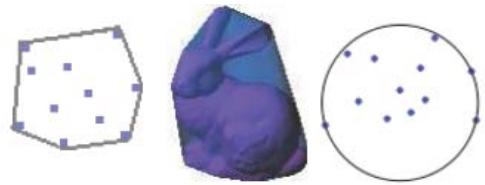
Computational Geometry Algorithms Library

Slides from [siggraph2008-CGAL-course]

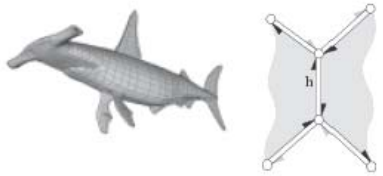
CGAL

- Large library of geometric algorithms
  - Robust code
  - 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

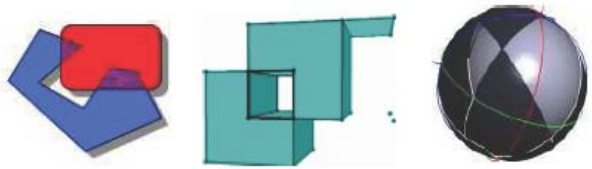
CGAL algorithms and data structures



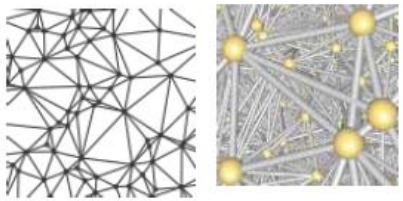
Bounding Volumes



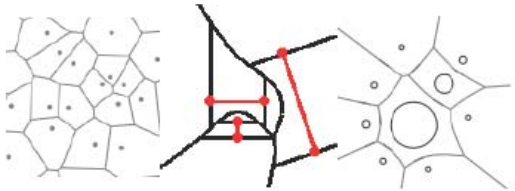
Polyhedral Surface



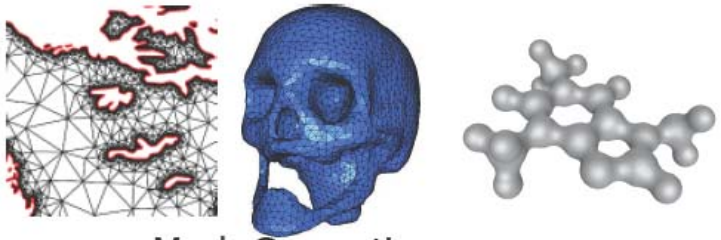
BooleanOperations



Triangulations



Voronoi Diagrams



Mesh Generation

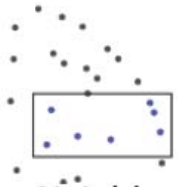
Subdivision

Simplification

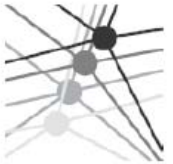
Parametrisation

Streamlines

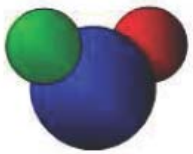
Ridge Detection



Neighbor Search

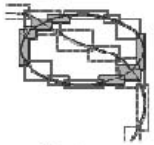


Kinetic Datastructures



Lower Envelope

Arrangement

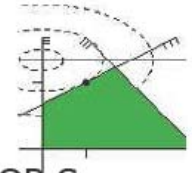


Intersection Detection

Minkowski Sum

PCA

Polytope distance

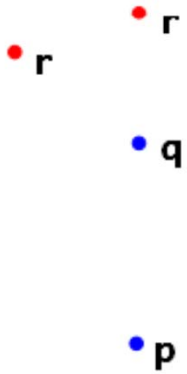


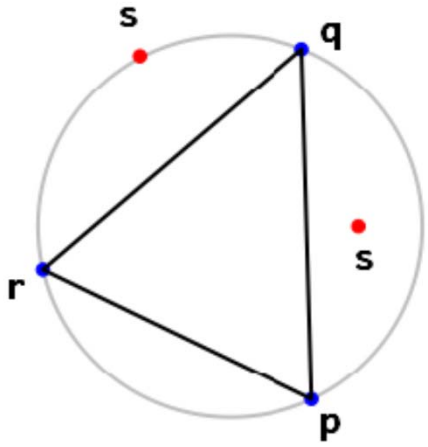
QP Sover

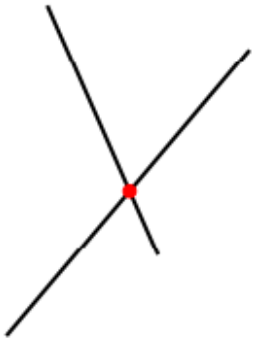
Exact geometric computing

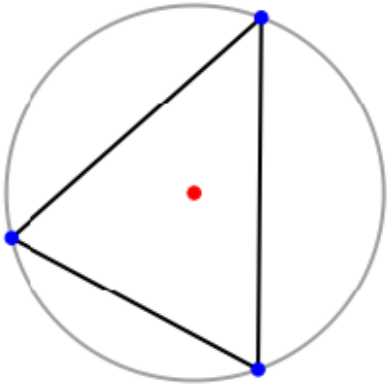
Predicates

Constructions









orientation

in_circle

intersection

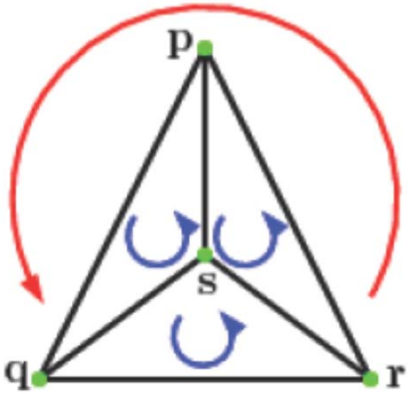
circumcenter

Robustness Issues

- Geometry in theory is exact
- Geometry with floating-point arithmetic is not exact
  - Limited numerical precision of real arithmetic
  - Inconsistent *eps* tests (a=b, b=c, but aû c)
- Naïve use of floating point arithmetic causes geometric algorithm to
  - Produce (slightly) wrong output
  - Crash after invariant violation
  - Infinite loop

Geometry in Theory

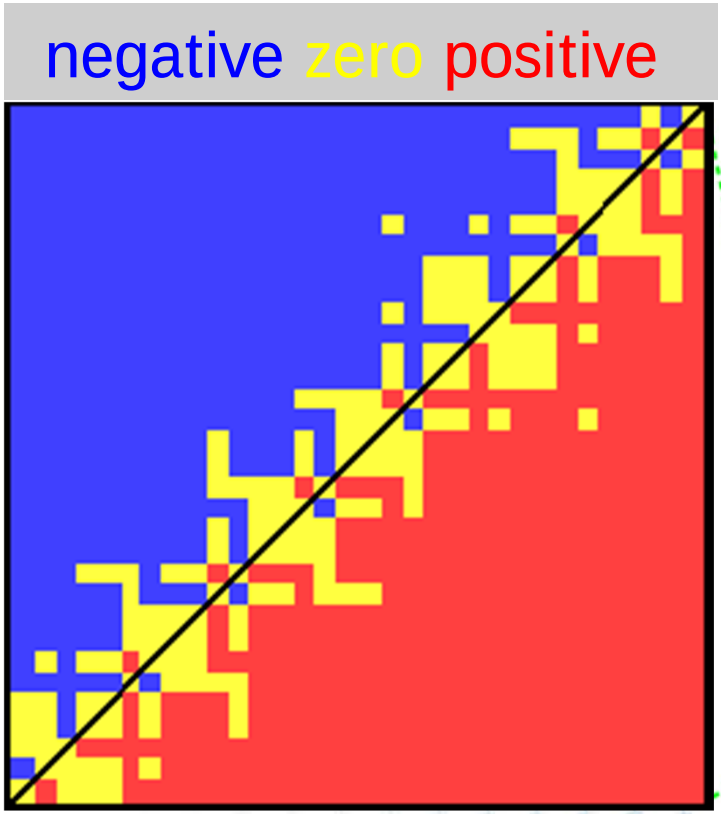
- ccw(s,q,r) & ccw(p,s,r) & ccw(p,q,s) => ccw(p,q,r)

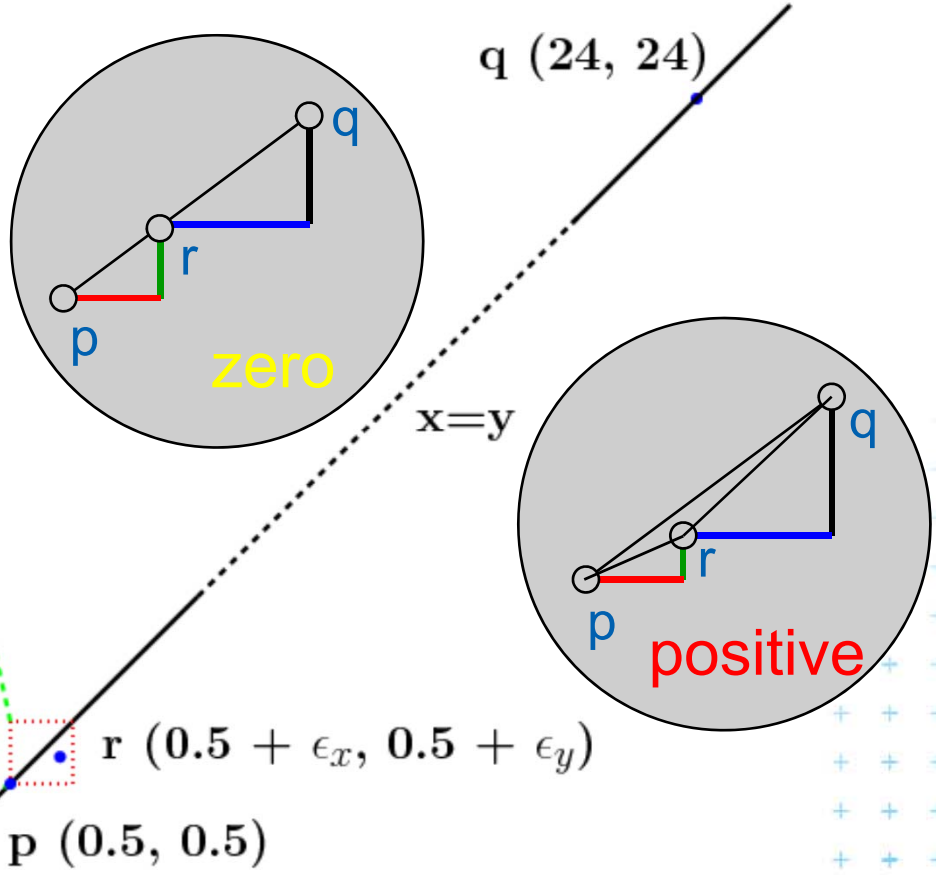


- Correctness proofs of algorithms rely on such theorems

The Trouble with Double – point to line relation

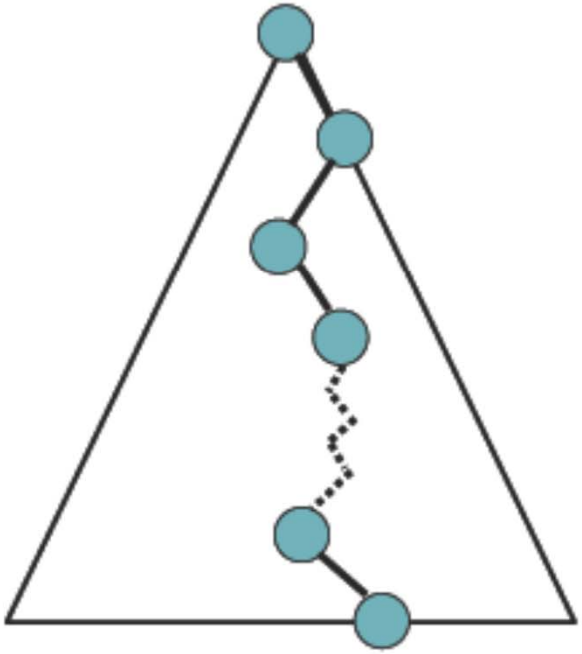
- orientation(p,q,r) = sign($(p_x-r_x)(q_y-r_y)-(p_y-r_y)(q_x-r_x)$)

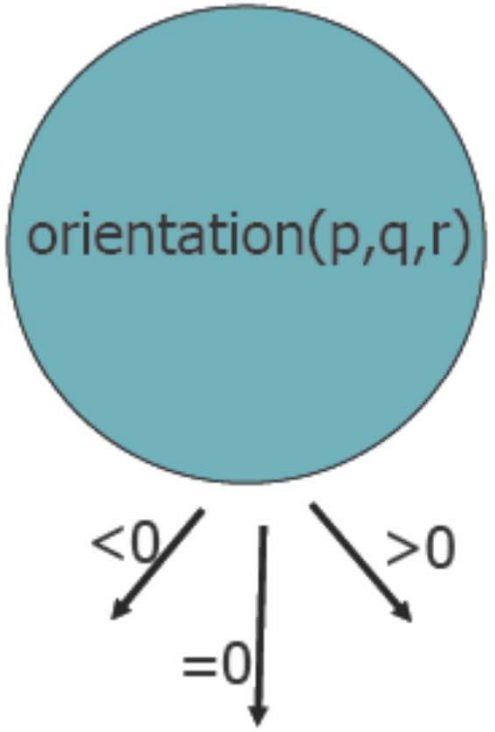




Exact Geometric Computing [Yap]

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

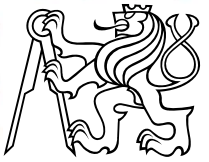




CGAL Geometric Kernel (see [Hert] for details)

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

- 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( 1.0, 0.0);
    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

Precission
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;

int main() {
    Point  p( 0.1, 0.2);

    ...
}
```

One single-line declaration changes the precision of all computations

Exact orientation test

```
#include <CGAL/Homogeneous.h>
#include <CGAL/Point_2.h>
#include <CGAL/predicates_on_points_2.h>
#include <iostream>
typedef CGAL::Homogeneous<long>          Rep;
typedef CGAL::Point_2<Rep>               Point;
int main() {
    Point p( 10,  0, 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;
    }
}
```

8 References

- 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/
- [Mount] Mount, D.: *Computational Geometry Lecture Notes for Spring 2007* http://www.cs.umd.edu/class/spring2007/cmsc754/Lects/comp-geom-lects.pdf
- Franko P. Preperata, 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

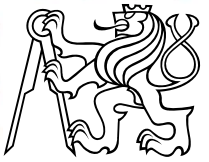
8.1 References

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]

8.2 Collections of geometry resources

- N. Amenta, *Directory of Computational Geometry Software*, http://www.geom.umn.edu/software/cglist/.
- D. Eppstein, *Geometry in Action*, http://www.ics.uci.edu/~eppstein/geom.html.
- Jeff Erickson, *Computational Geometry Pages*, http://compgeom.cs.uiuc.edu/~jeffe/compgeom/



9. 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
  - principles more than on precise mathematical proofs

- Practical experiences by using CGAL library
  - Huge amount of algorithms „on-the-shelf“
  - Control over robustness and precision of computations