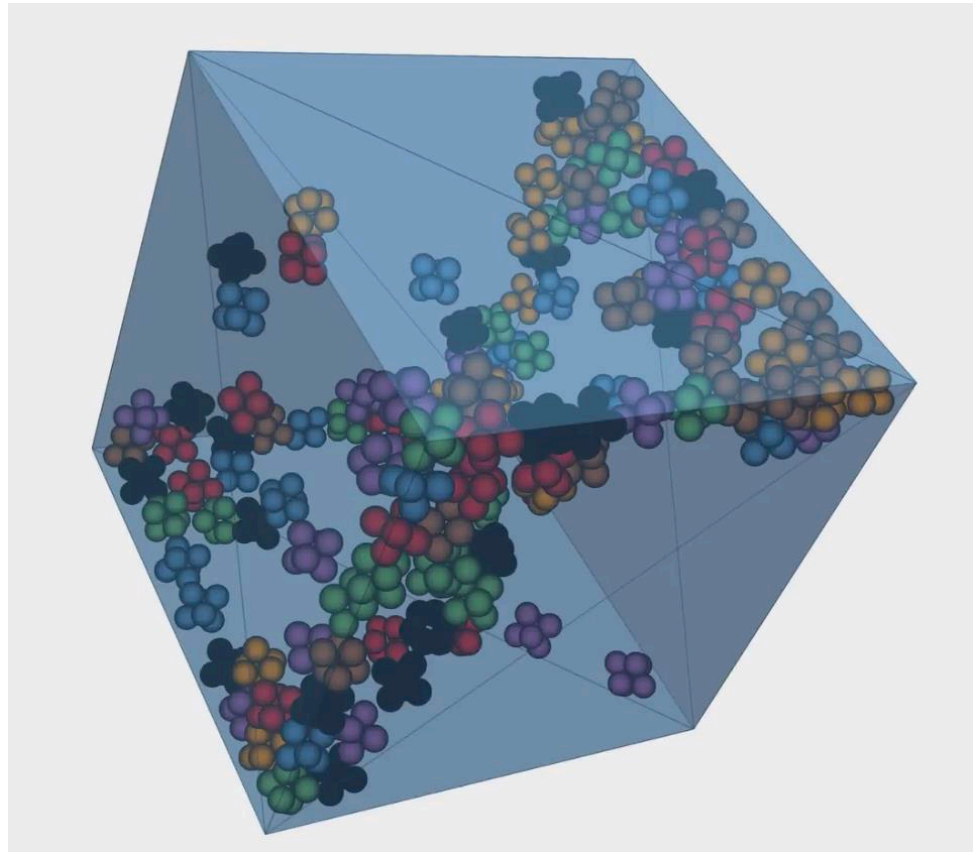
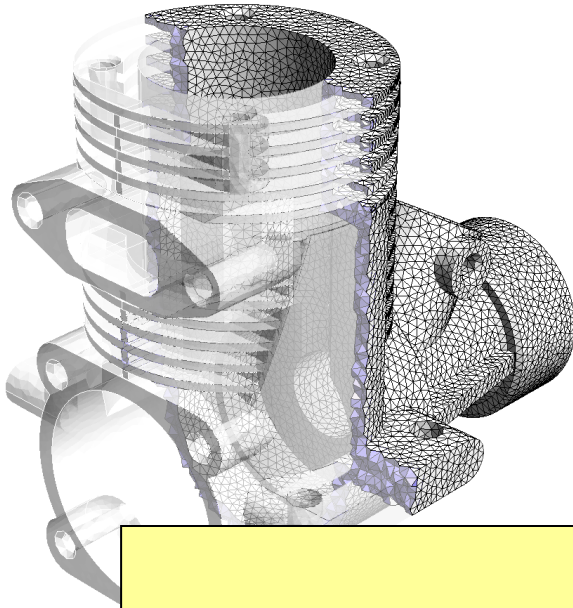


Why numerical geometry ?



Numerical Geometry



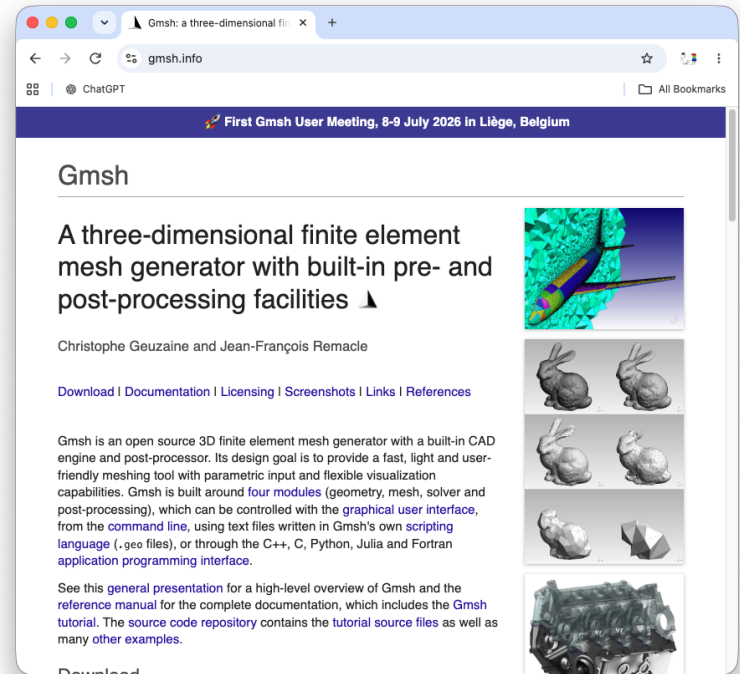
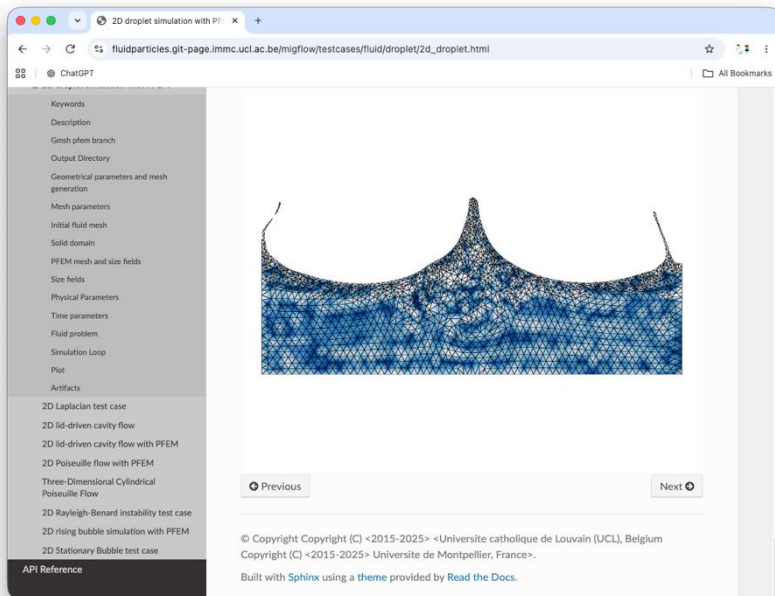
- 2026-2027 : Implementation of PFEM technique
- Python only !

- Klee problem
- Robust predicates
- Delaunay triangulations
- Alpha-shape algorithms
- Use of my friend Claude
- And a lot of fun

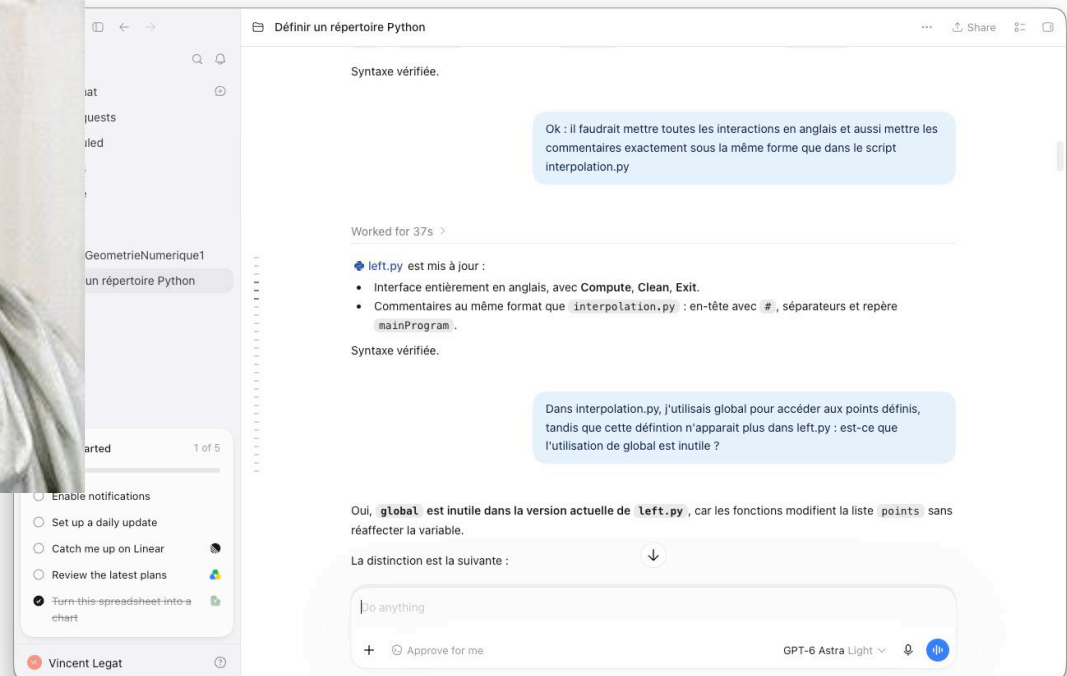
Jean-François Remacle
Vincent Legat

Félix Ruyffelaere
Nathan Tihon
Okay Kerem
Illy Perl
Miguel De Le Court

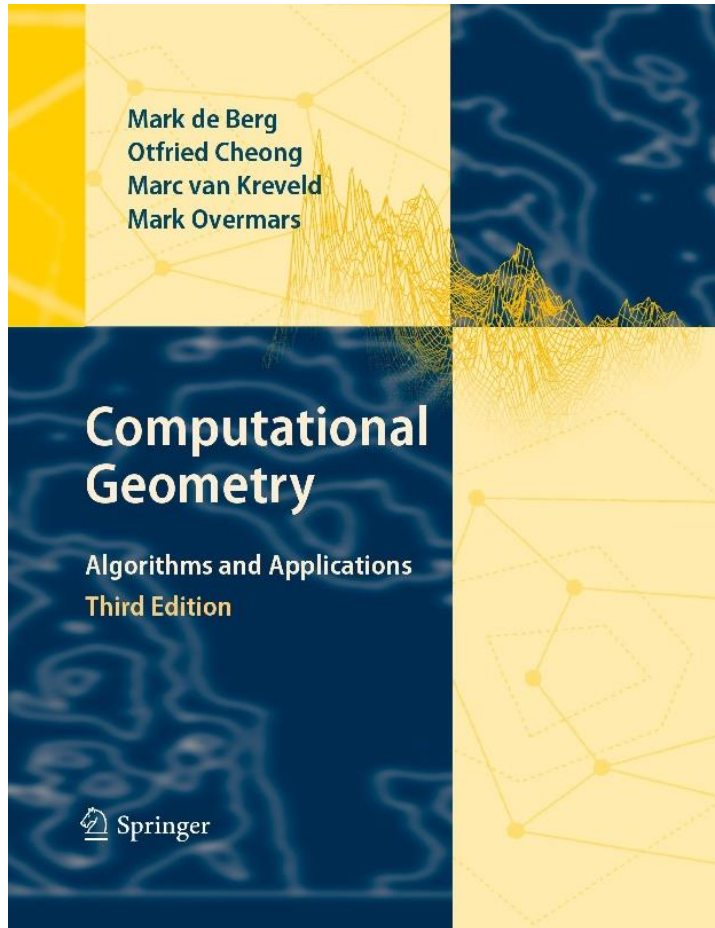
The tools



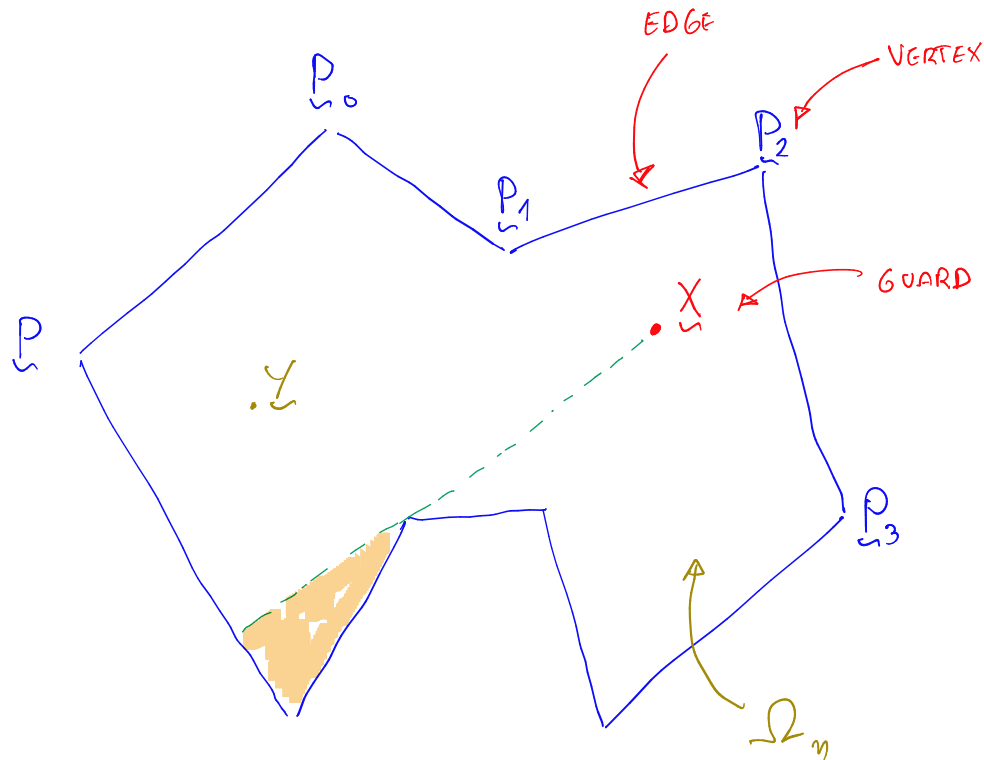
The tools



The textbook



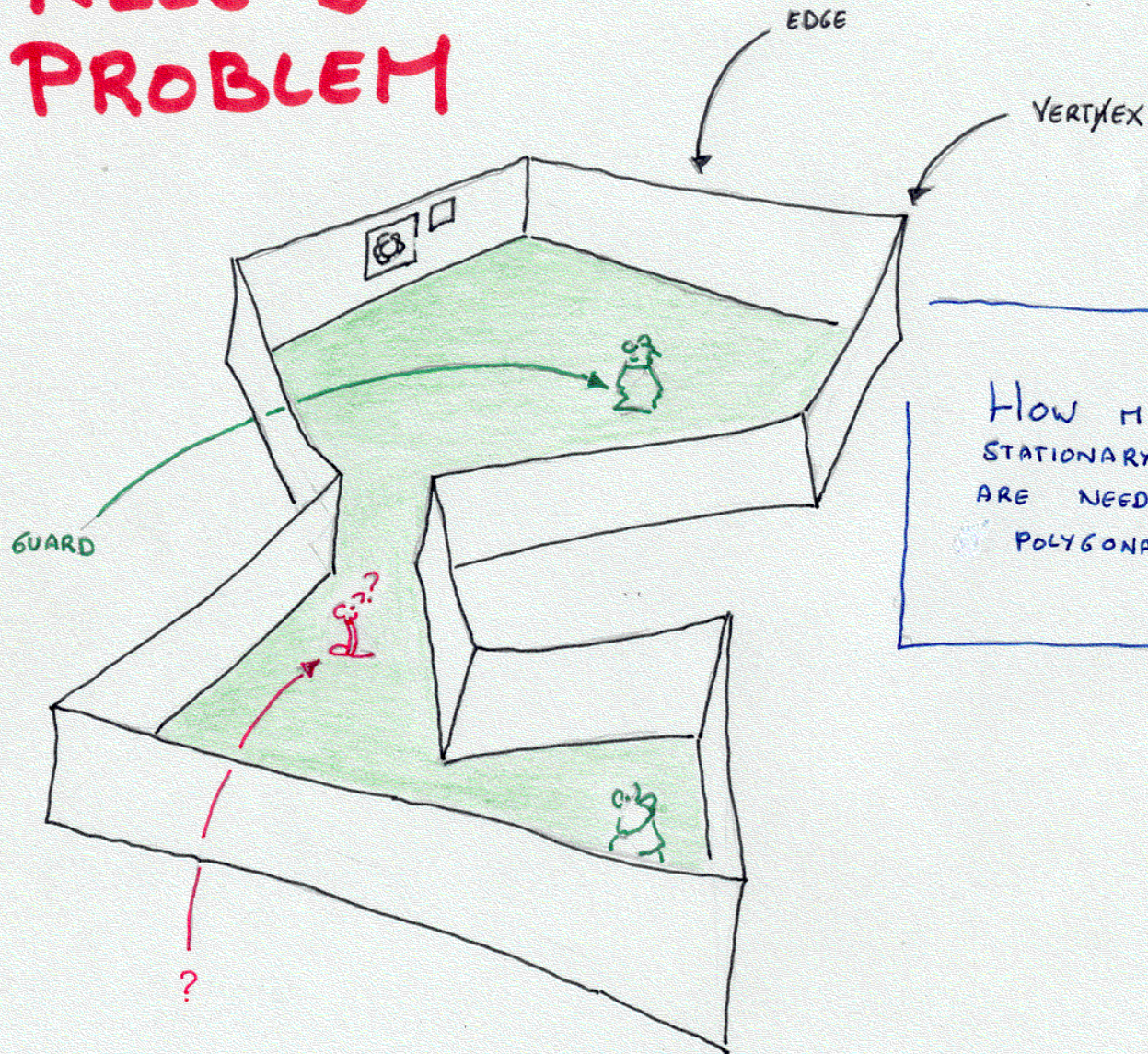
Klee's problem



$\{X, Y, Z\}$ COVERS Ω_n
SET OF GUARDS

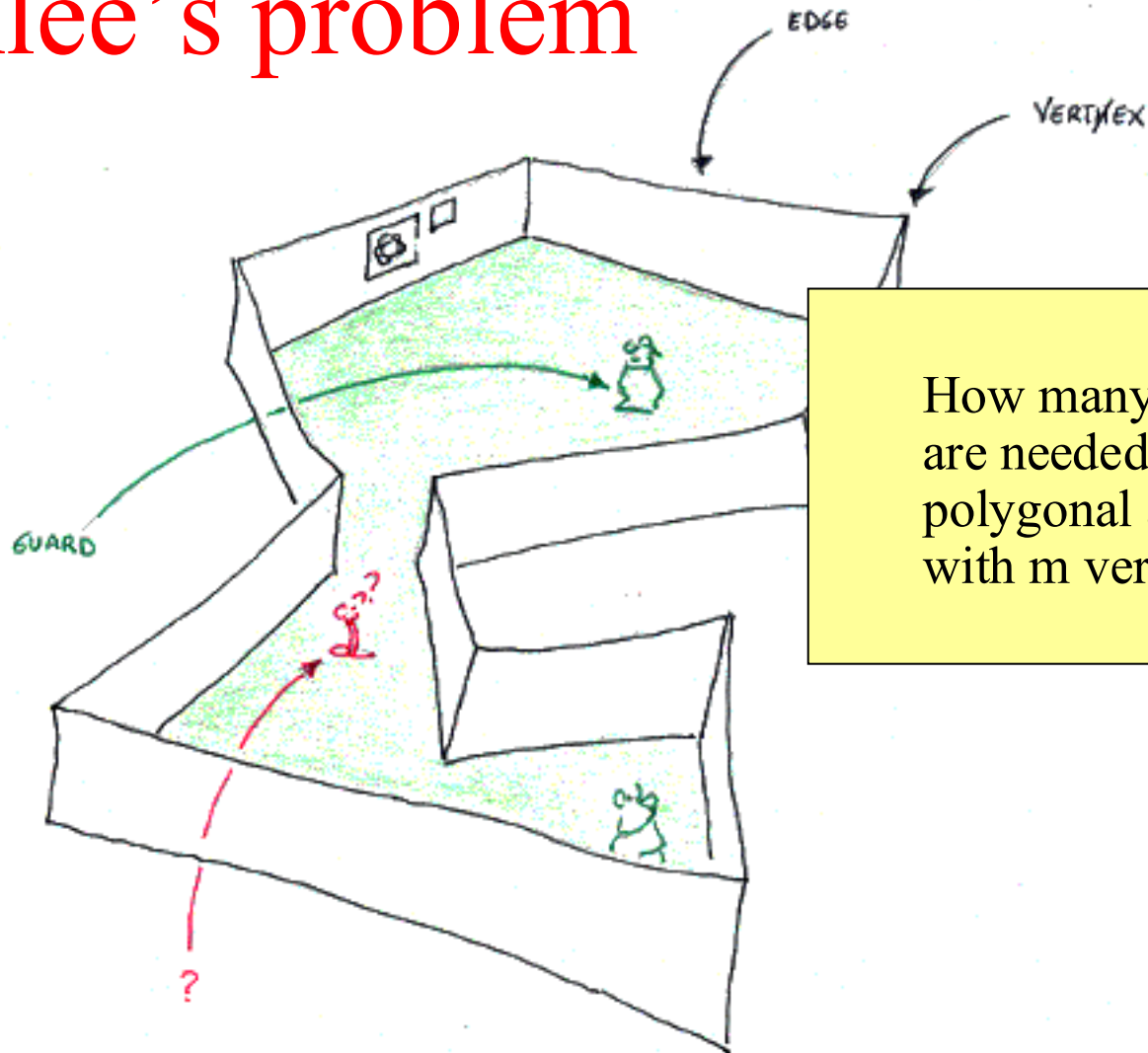
How MANY
STATIONARY GUARDS
ARE NEEDED
TO GUARD
ANY POLYGONAL ROOM
WITH n VERTICES ?

KLEE'S PROBLEM



How many
stationary guards
are needed to guard any
polygonal art gallery room
with n vertices?

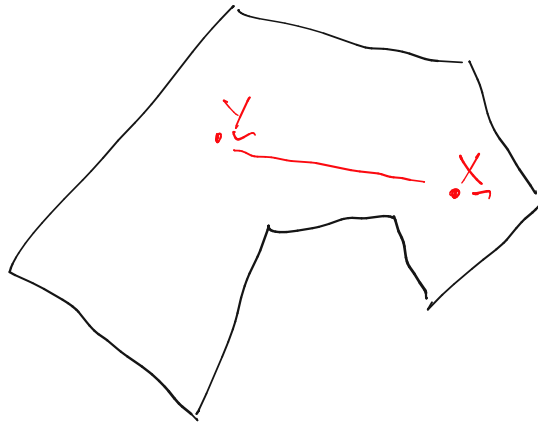
Klee's problem



How many stationary guards are needed to guard any polygonal art gallery room with n vertices ?

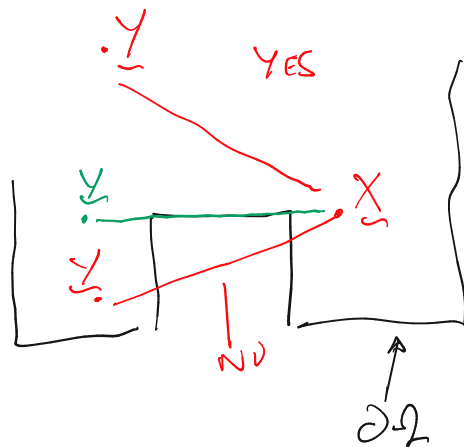
Visibility...

$XY \subset \Omega$
VISIBILITY



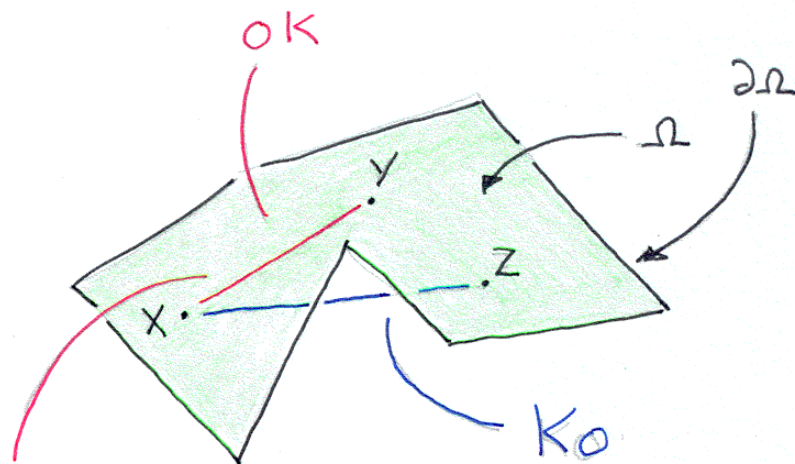
CLEAR
VISIBILITY

$$(XY \cap \partial\Omega) \subseteq \{X, Y\}$$

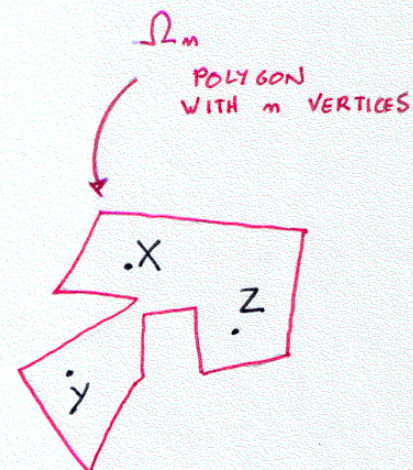


...and clear visibility !

VISIBILITY



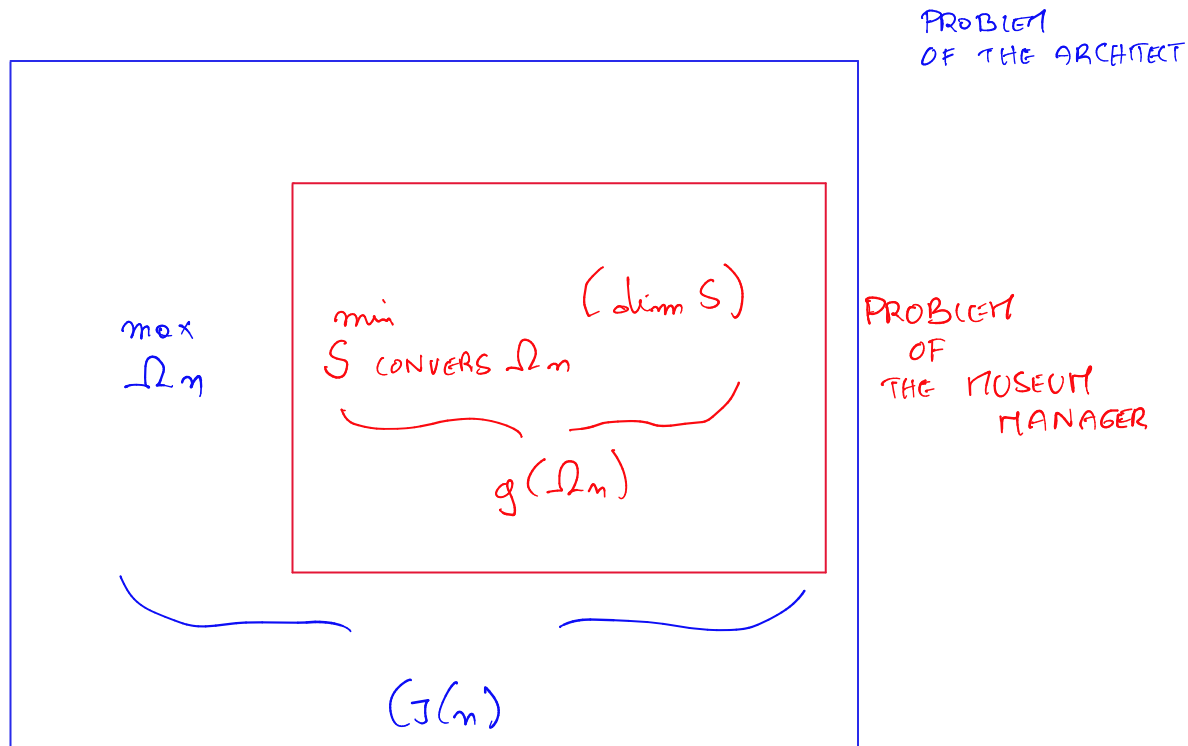
$xy \subseteq \Omega$
 $(xy \cap \partial\Omega) \subseteq \{x, y\}$
 VISIBILITY
 CLEAR VISIBILITY



$\{x, y, z\}$ COVERS Ω_m
 SET OF GUARDS

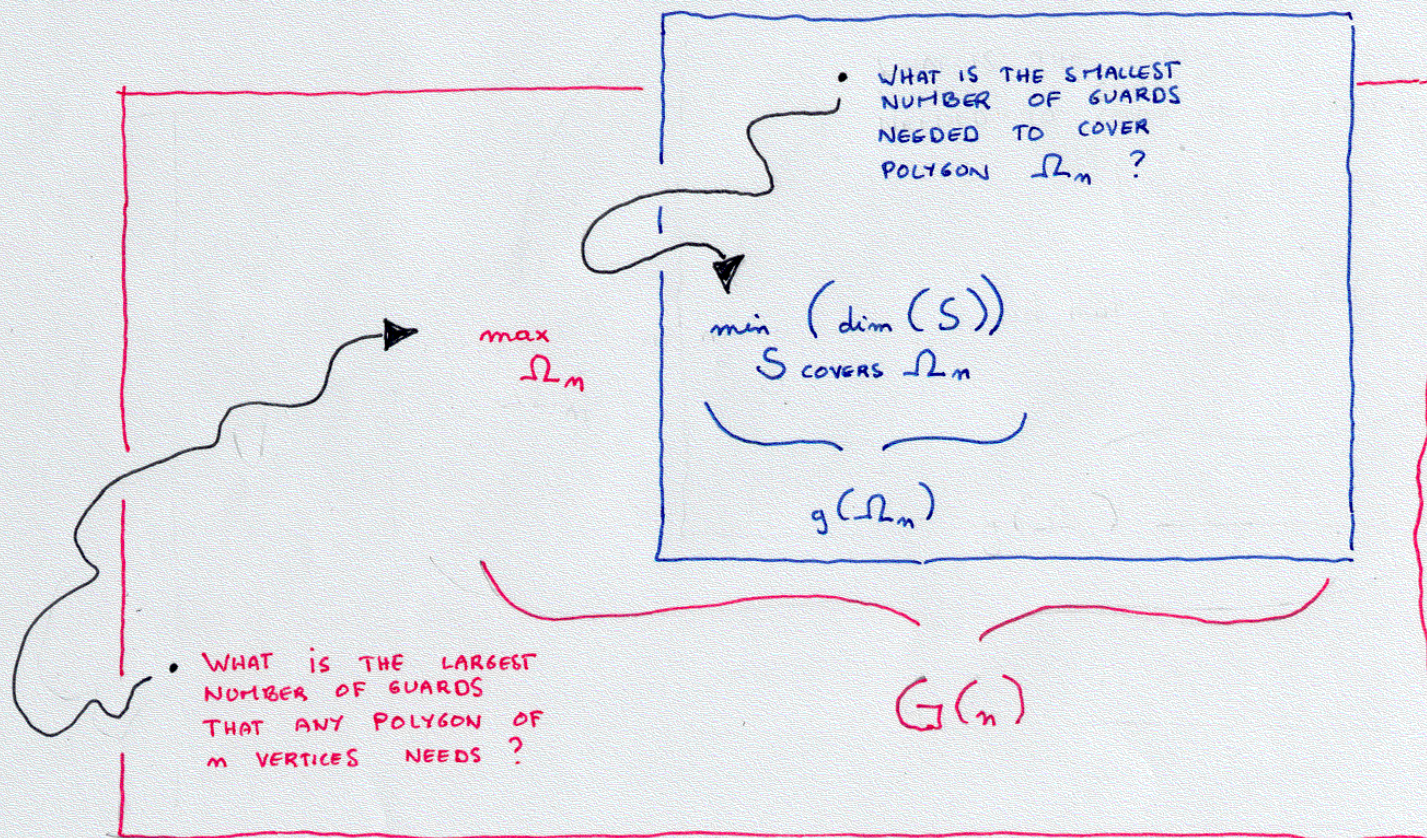
IF

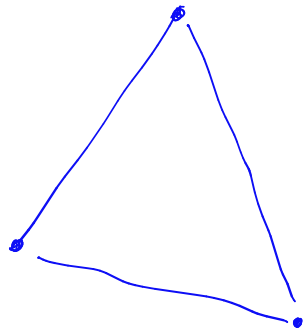
EVERY POINT OF Ω IS
 VISIBLE TO A GUARD AT LEAST



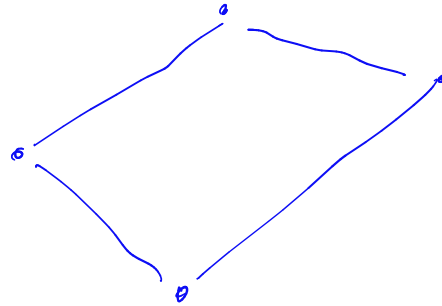
A so exciting
problem for
mathematicians !

MAX-MIN FORMULATION

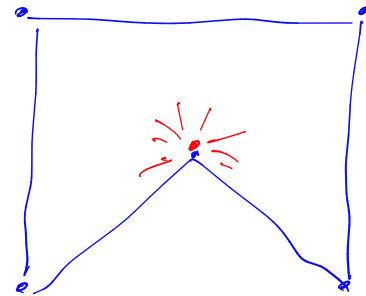




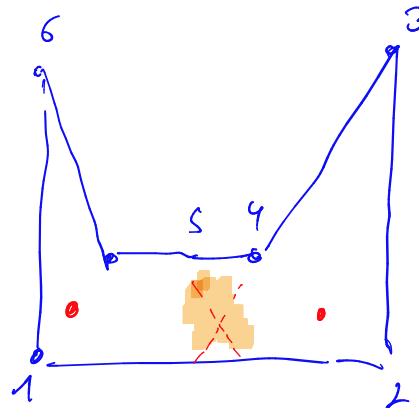
$$G(3) = 1$$



$$G(4) = 1$$



$$G(5) = 1$$



$$G(6) = 1$$

The fool's method !

EMPIRICAL EXPLORATION

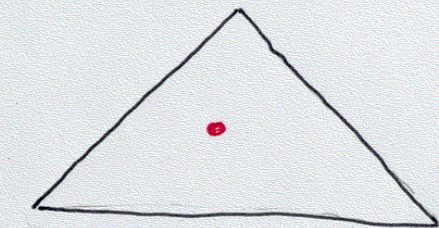
$$1 \leq G(n) \leq n$$



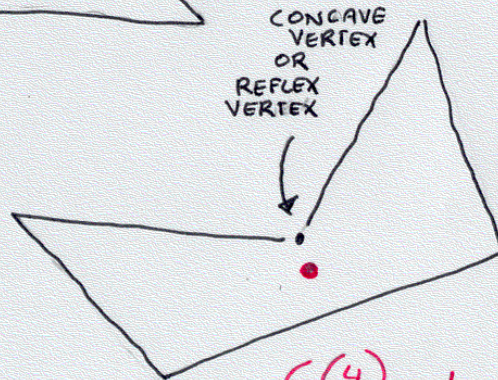
INTUITIVELY, WE PUT A GUARD AT EACH VERTEX
THIS IS NOT EVEN SO CLEAR
AT LEAST, IT DEMANDS A PROOF
IT TURNS OUT TO BE TRUE HOWEVER IN 2D

(NOT IN 3D !)

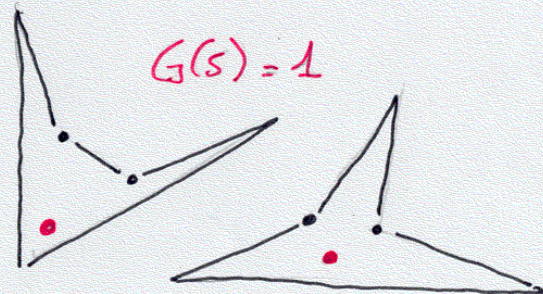
DO NOT TRUST
INTUITION ...



$$G(3) = 1$$



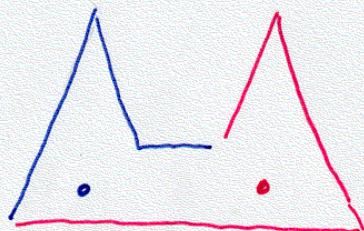
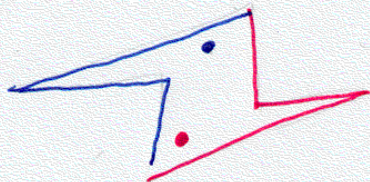
$$G(4) = 1$$



$$G(5) = 1$$

BUT

$$G(6) = 2 \dots !$$



CHVÁTAL'S COMB SHAPE

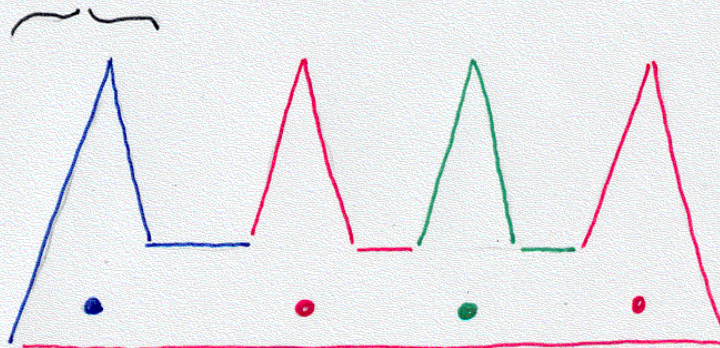
EACH PRONG
REQUIRES ITS OWN GUARD!

(1975)

4 PRONGS

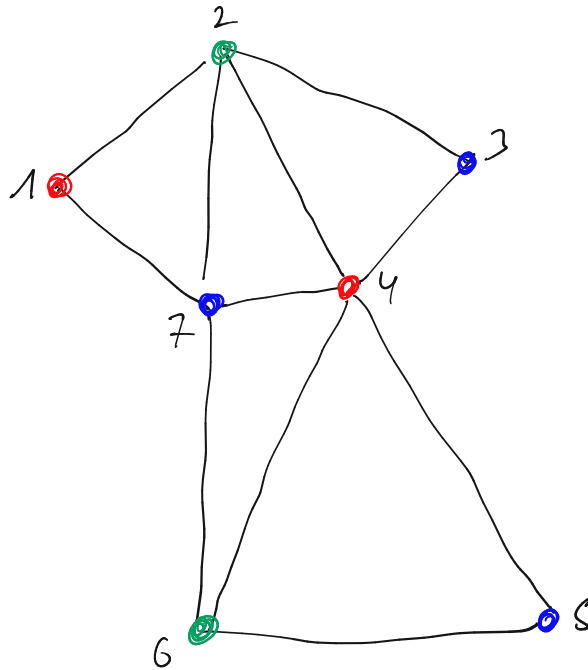
12 EDGES

PRONG



$$G(n) = \lfloor n/3 \rfloor$$

A proof!



STEP 1

TRIANGULATION
WITH NON-CROSSING
DIAGONALS

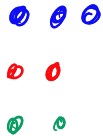
STEP 2

3-COLORING
OF THE TRIANGULATION

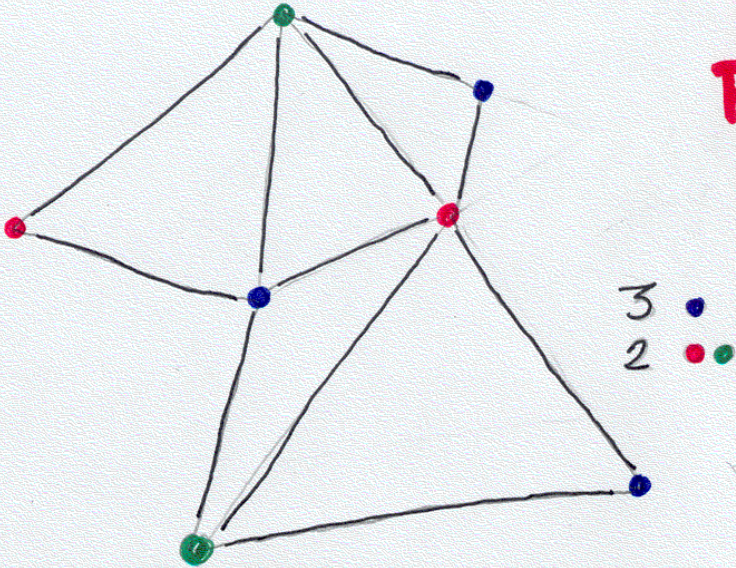
STEP 3

PIGEONS - BOXES

7 VERTICES = BIRDS
3 COLOR = BOXES



FISK'S PROOF

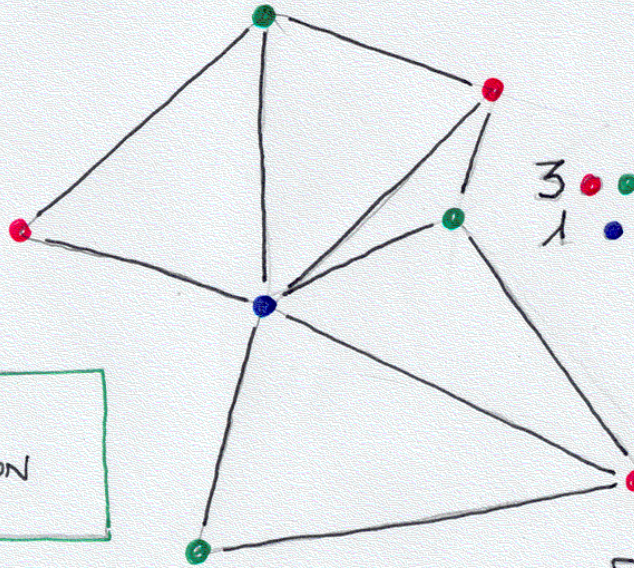


STEP 1

TRIANGULATION
WITH NON-CROSSING
DIAGONALS

STEP 2

THREE-COLORING
OF THE TRIANGULATION



STEP 4

PIGEON-HOLE PRINCIPLE

7 VERTICES 3 COLORS

AT LEAST, ONE HOLE
MUST CONTAIN NO MORE

$$\lfloor 7/3 \rfloor = 2$$

STEP 3

PLACING GUARDS
AT ALL THE VERTICES
ASSIGNED ONE COLOR
GUARANTEES VISIBILITY

TRIANGULATION THEORY

LEMMA

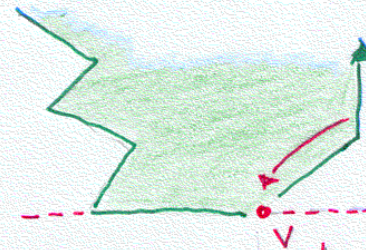
EVERY POLYGON
MUST HAVE AT LEAST
ONE STRICTLY CONVEX VERTEX

LEMMA

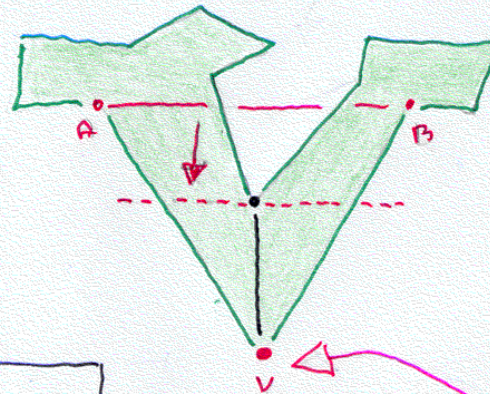
EVERY POLYGON
OF $n \geq 4$ VERTICES
HAS A DIAGONAL

THEOREM

EVERY POLYGON
MAY BE PARTITIONED
INTO TRIANGLES BY
THE ADDITION OF DIAGONALS



THE
RIGHTMOST
LOWEST VERTEX
IS STRICTLY CONVEX

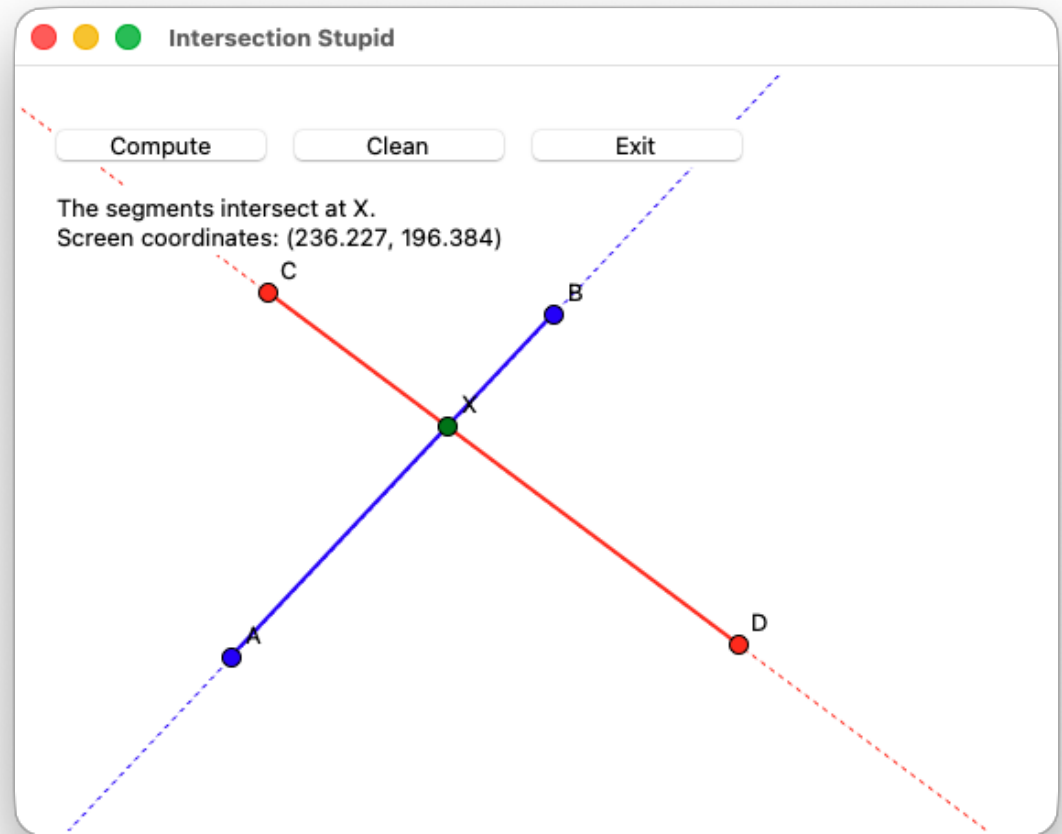


PROOF BY INDUCTION

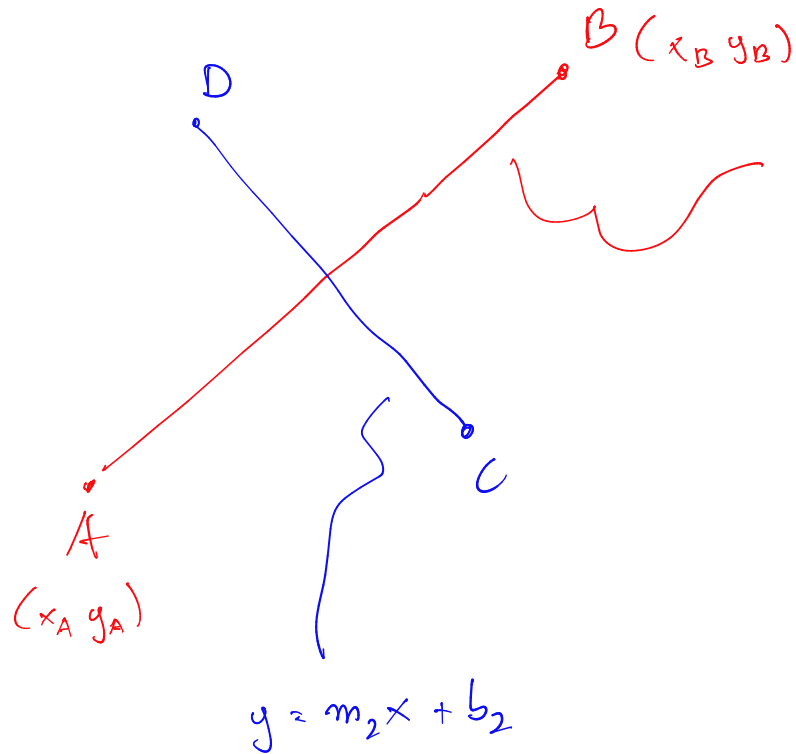
$n=3$ TRUE

$n \geq 4$ USES THE LEMMA TO
SPILT INTO 2 POLYGONS !

Intersecting two segments



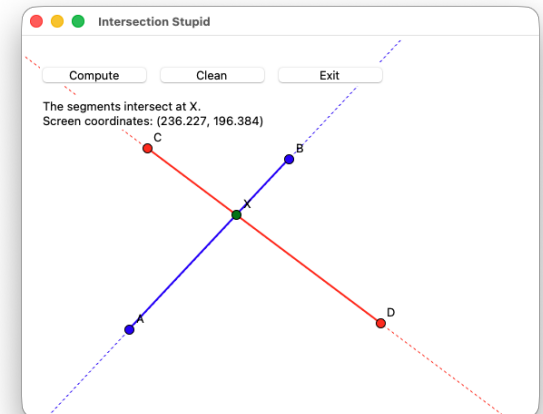
Is it easy ?



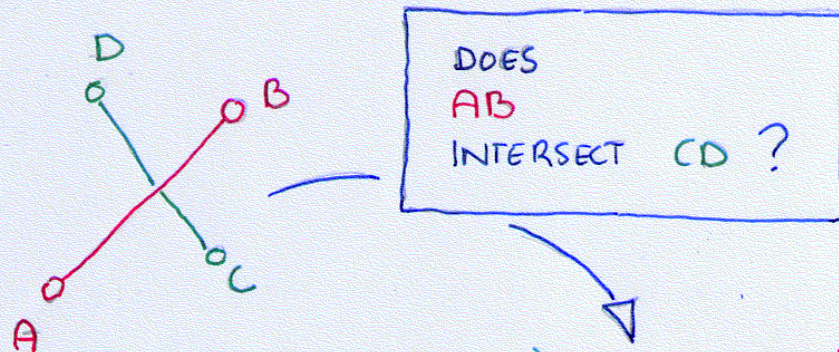
$$y = m_1x + b_1$$

$$m_1 = \frac{y_b - y_a}{x_b - x_a}$$

$$b_1 = y_a - m_1x_a$$

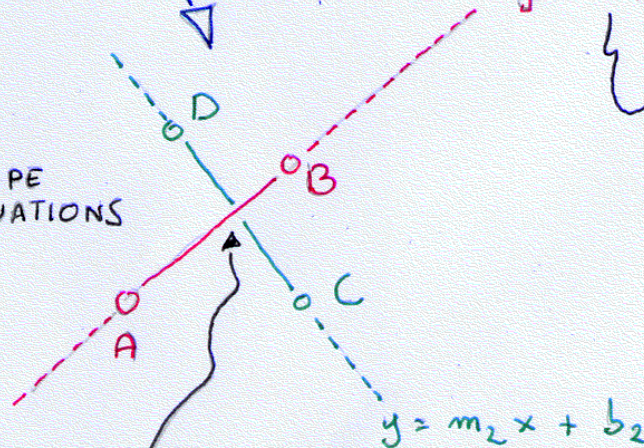


PROBLEMS WITH SLOPES



DOES
AB
INTERSECT CD ?

SLOPE
EQUATIONS



$$y = m_1x + b_1$$

$$m = \frac{y_B - y_A}{x_B - x_A}$$

$$b = y_A - m x_A$$

$$\left(\frac{b_1 - b_2}{m_2 - m_1} ; \frac{b_1 m_1 - b_2 m_2}{m_2 - m_1} \right)$$

SPECIAL CASES PHILOSOPHY

EASY !



2

WRITE CODE THAT HANDLES THE SPECIAL CASES



TEDIOUS AND ERROR-PRONE

3

FIND GEOMETRICAL SYSTEM WITHOUT SPECIAL CASES



WONDERFUL BUT NOT ALWAYS POSSIBLE !

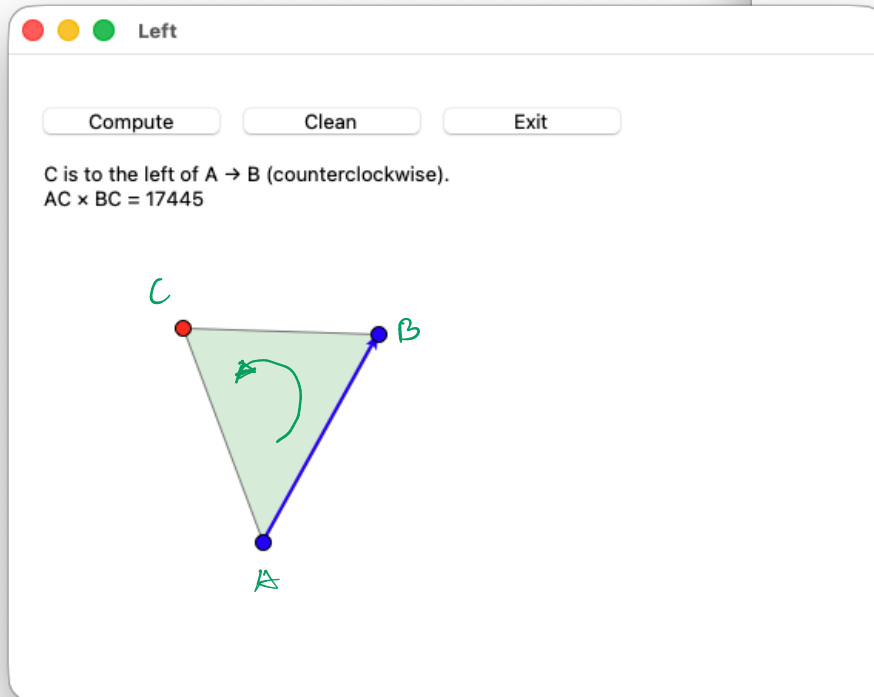
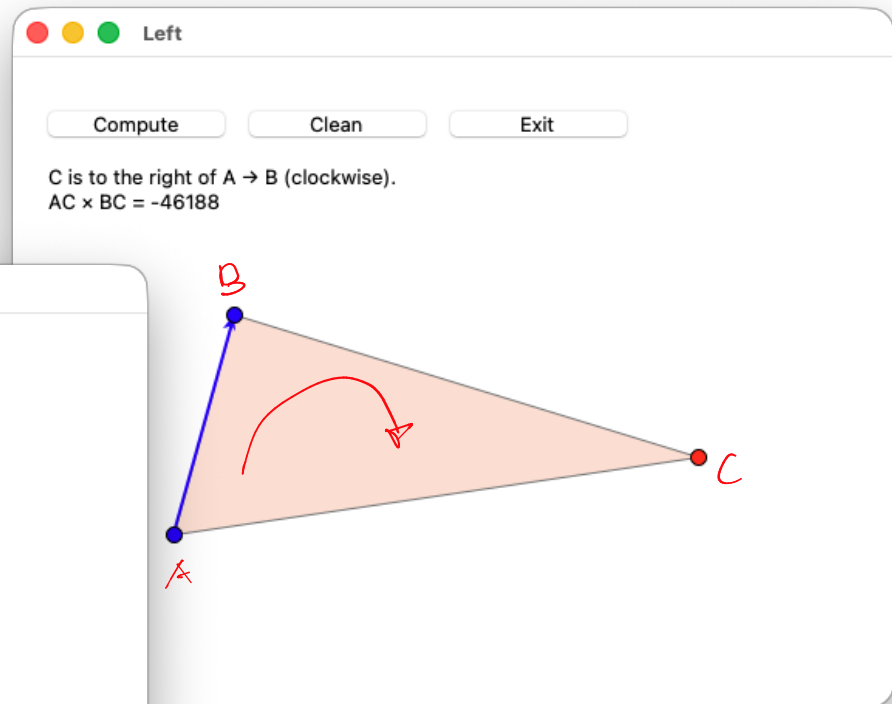
EX : HOMOGENEOUS COORDINATES IN COMPUTER GRAPHICS

4

FIND COMPUTER IMPLEMENTATION WITHOUT SPECIAL CASES



Left ?



```
def orient2d(a,b,c):  
    return ((c[0] - a[0]) * (c[1] - b[1])  
            - (c[1] - a[1]) * (c[0] - b[0]))
```

LEFT

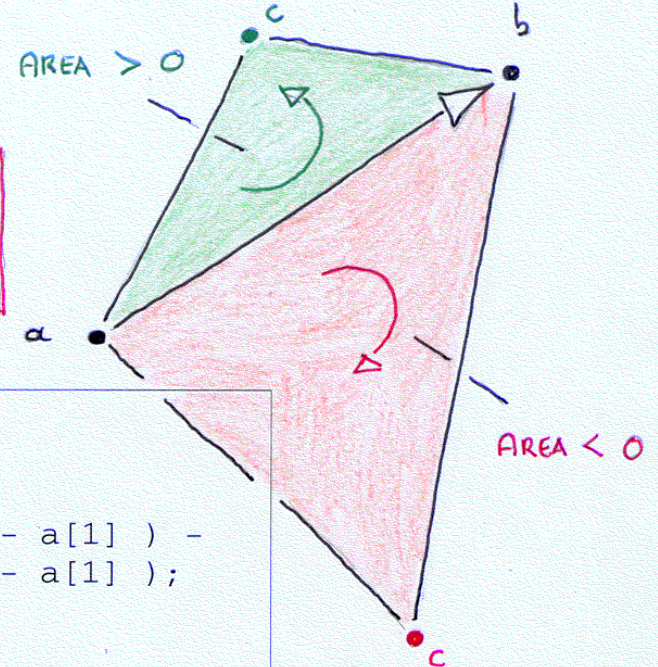
COLLINEARITY

EXCEPTIONAL
GEOMETRIC
SITUATION



EXCEPTIONAL
NUMERICAL
SITUATION

ZERO AREA



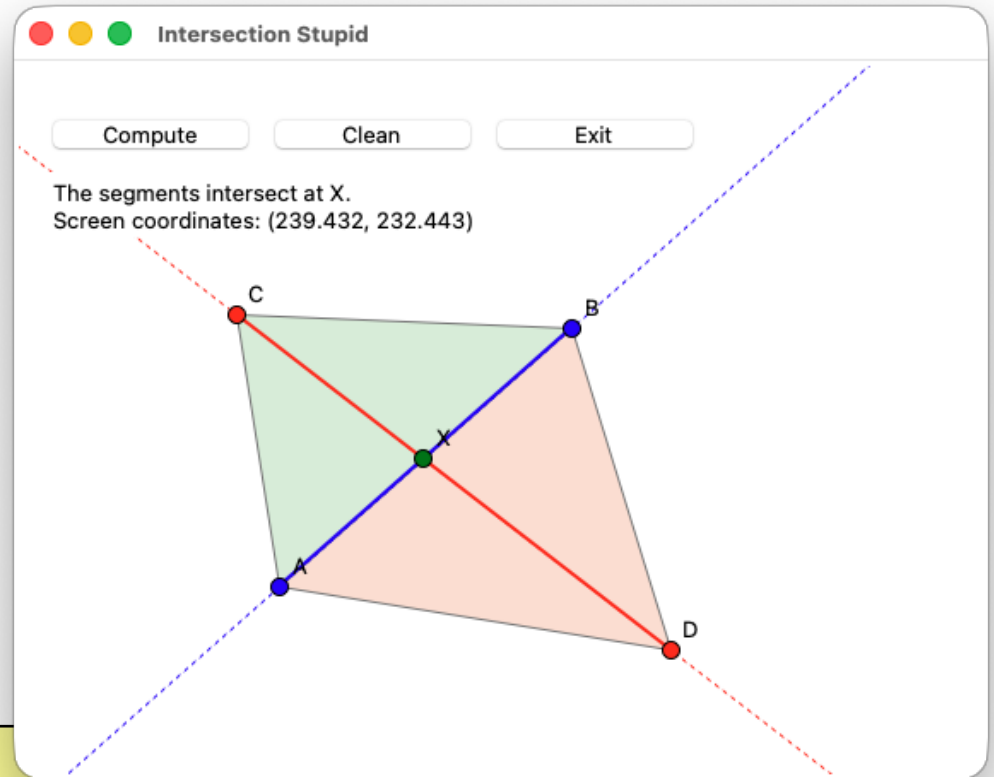
```
int    AreaSign(Pointi a,Pointi b,Pointi c)
{
    double area2;
    area2 = ( b[0] - a[0] ) * (double)( c[1] - a[1] ) -
            ( c[0] - a[0] ) * (double)( b[1] - a[1] );

    /* The area should be an integer. */
    if      ( area2 > 0.5 ) return( 1);
    else if ( area2 < -0.5 ) return(-1);
    else
        return( 0);
}

int    Left(Pointi a,Pointi b,Pointi c)
{
    return( AreaSign(a,b,c) > 0 );
}
```

Predicate $\text{orient2d}(a,b,c)$

```
def intersect(a,b,c,d):  
    abc = orient2d(a,b,c)  
    bad = orient2d(b,a,d)  
    t = abc / (bad + abc)  
    dca = orient2d(d,c,a)  
    s = dca / (bad + abc)  
    if not (0 < t < 1 and 0 < s < 1): return None  
    return ((1 - t) * c[0] + t * d[0],  
            (1 - t) * c[1] + t * d[1])
```



Let us do the math !

$$\begin{cases} x = (1-t) c_x + t d_x \\ y = (1-t) c_y + t d_y \end{cases}$$

$$\underline{x} = (1-t) \underline{c} + t \underline{d}$$

$$\underline{c} = (c_x \ c_y)$$

$$\underline{d} = (d_x \ d_y)$$

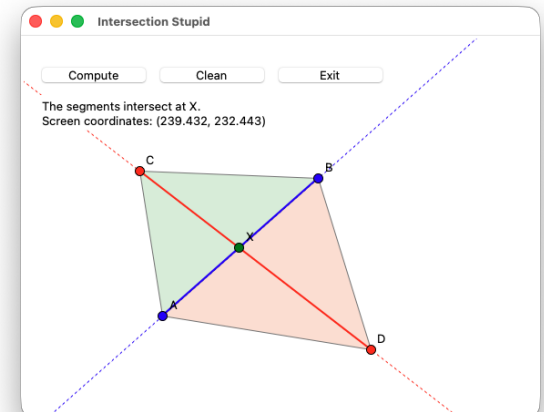
$$\underbrace{f(a, b, x)}_{=0} = (1-t) \underbrace{f(a, b, c)}_{abc} + t \underbrace{f(a, b, d)}_{\substack{abd \\ -bad}}$$

$$\underbrace{a} \underbrace{b} \underbrace{c} \quad \underbrace{a} \underbrace{b} \underbrace{d}$$

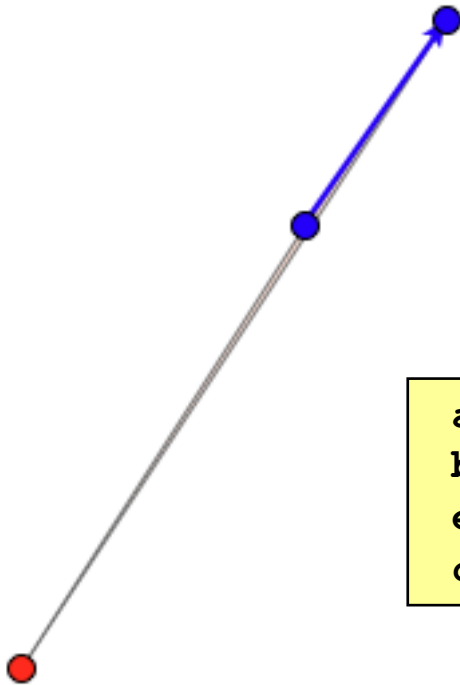
$$f(a, b, c)$$

$$abc = t(abc + bad)$$

$$t = \frac{abc}{abc + bad}$$



But it does not work !



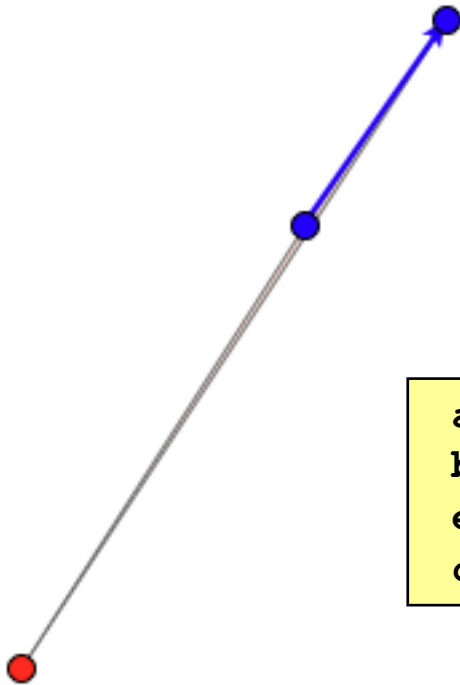
$$\underbrace{7./3} - \underbrace{4./3} - 1$$

$3/3$

```
a = (12.0, 12.0)
b = (36.0, 36.0)
eps = 7.0 / 3.0 - 4.0 / 3.0 - 1.0
c = (1.0 + i*eps, 1.0 + j*eps)
```

```
def orient2d_naive(a,b,c):
    return ((c[0] - a[0]) * (c[1] - b[1])
            - (c[1] - a[1]) * (c[0] - b[0]))
```

Computers are stupid !

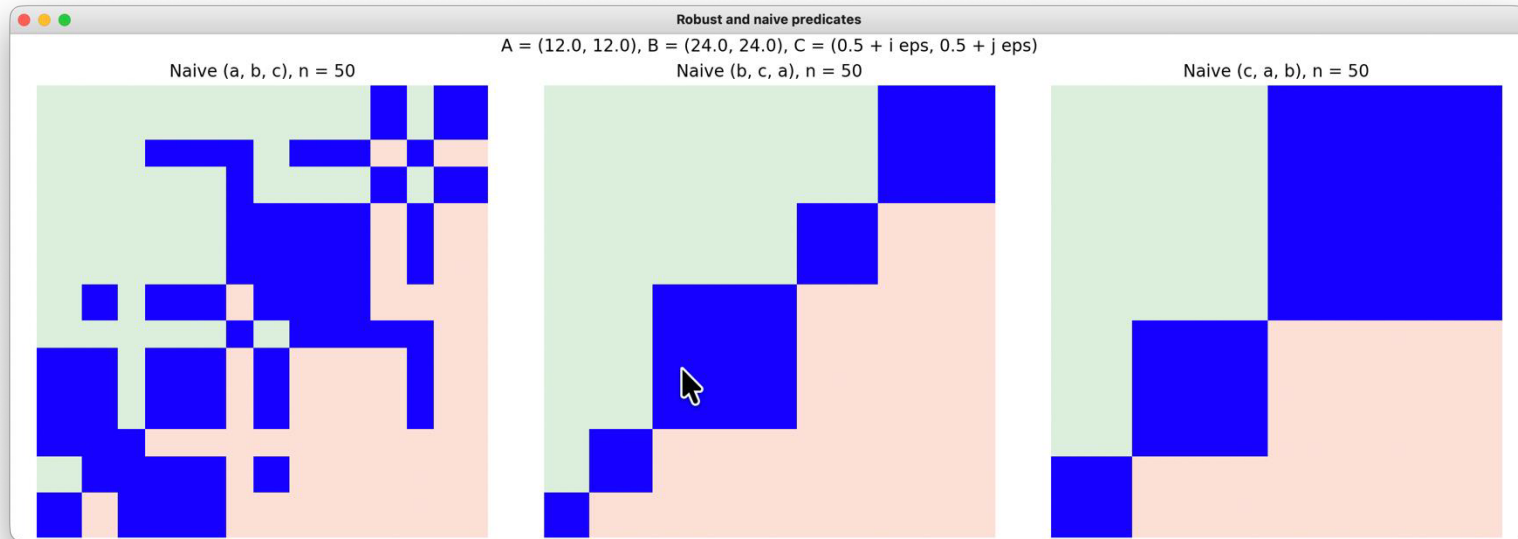


```
>>> eps = 7./3 - 4./3 -1
>>> eps
2.220446049250313e-16
```

```
a = (12.0, 12.0)
b = (24.0, 24.0)
eps = 7.0 / 3.0 - 4.0 / 3.0 - 1.0
c = (0.5 + i*eps, 0.5 + j*eps)
```

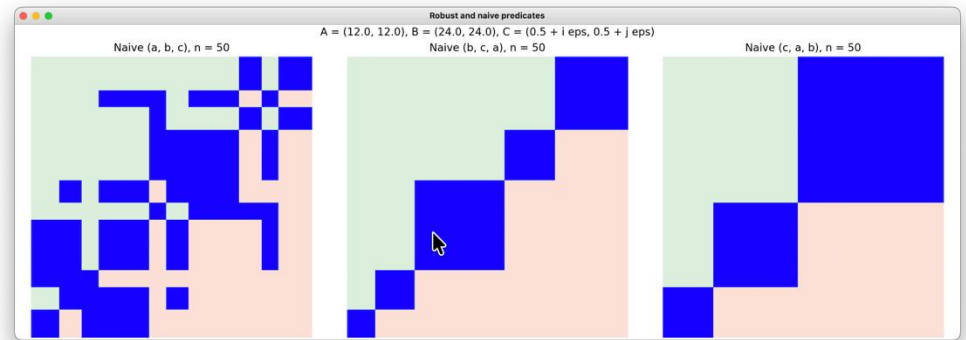
```
def orient2d_naive(a,b,c):
    return ((c[0] - a[0]) * (c[1] - b[1])
            - (c[1] - a[1]) * (c[0] - b[0]))
```

Permuting arguments !



```
def orient2d_naive(a,b,c):  
    return ((c[0] - a[0]) * (c[1] - b[1])  
            - (c[1] - a[1]) * (c[0] - b[0]))
```

Naive(a,b,c)/Naive (b,c,a)	disagreements:	642 / 2500
Naive(a,b,c)/Naive (c,a,b)	disagreements:	638 / 2500
Naive(b,c,a)/Naive (c,a,b)	disagreements:	1004 / 2500

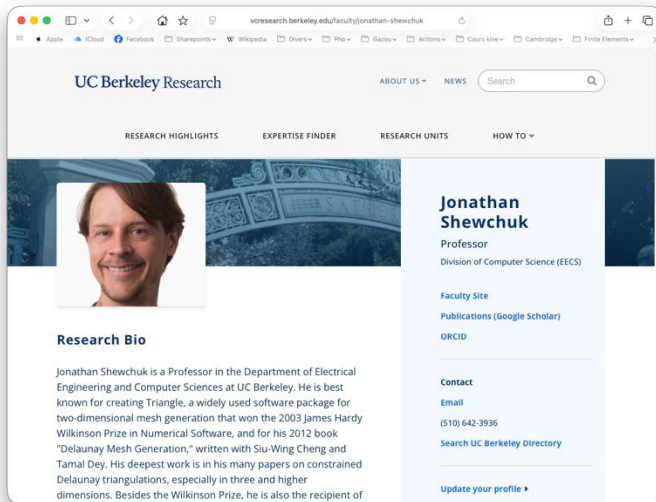


$$\begin{array}{cccc}
 \underbrace{(c_x - a_x)}_{-11.5 + i\epsilon} & \underbrace{(c_y - b_y)}_{-23.5} & \leftarrow & \underbrace{(c_y - a_y)}_{-11.5} \underbrace{(c_x - b_x)}_{-23.5 + i\epsilon}
 \end{array}$$

Why is it not robust ?

Why order of operations breaks the associativity rules ?

Robust predicates



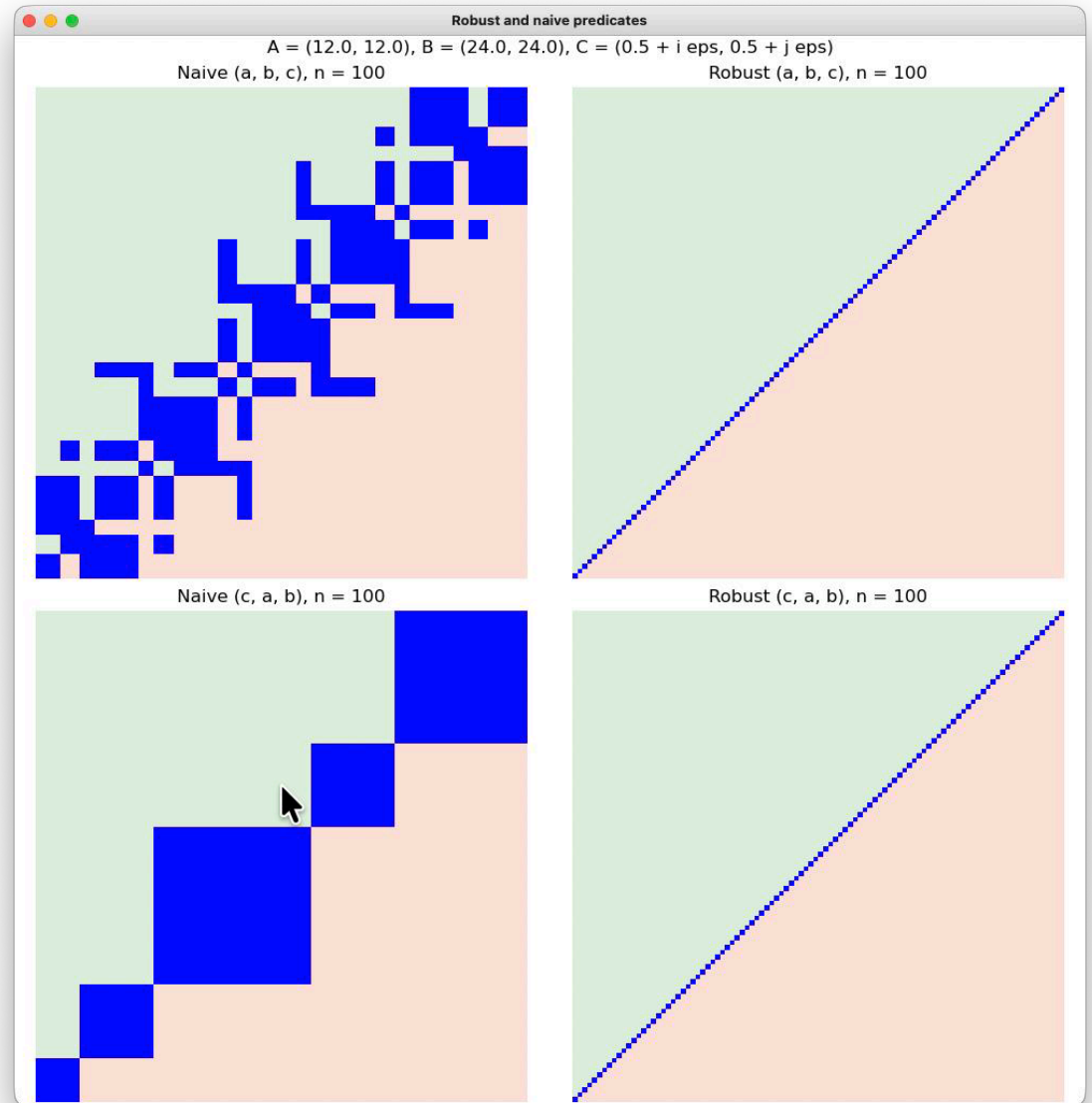
Shewchuk's algorithms use adaptive precision: simple cases are computed quickly using machine precision, and the precision of the computation is increased when the result is too close to zero for its sign to be guaranteed.

The goal is thus to obtain the exact sign while avoiding the cost of full-precision arithmetic in ordinary cases.

J. R. Shewchuk, *Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates*, Discrete & Computational Geometry, 18(3), 305–363, 1997. DOI: 10.1007/PL00009321

```
import numpy as np
import matplotlib.pyplot as plt
from geompreds import orient2d
```

Robust predicates



```
from geompreds import orient2d
```