Algorithmen für die Echtzeitgrafik

Algorithmen für die Echtzeitgrafik

Daniel Scherzer
scherzer@cg.tuwien.ac.at

LBI Virtual Archeology

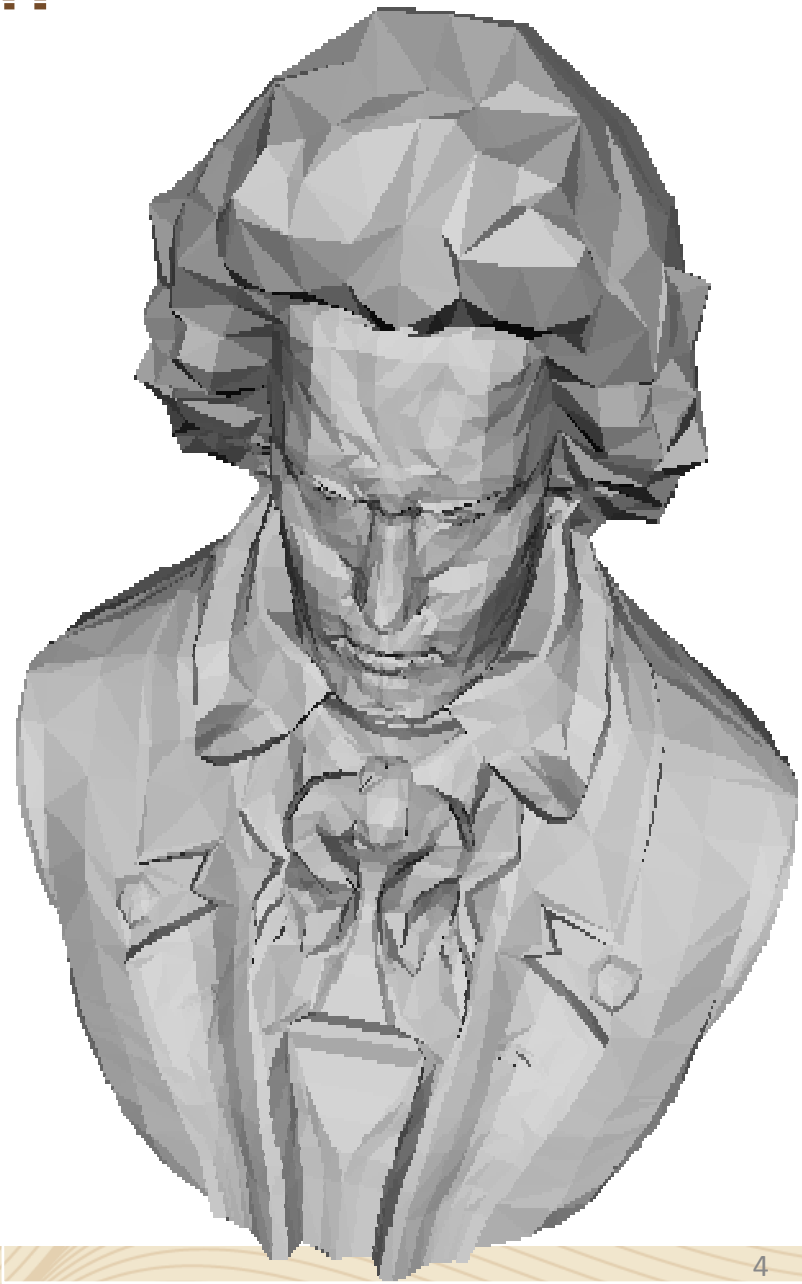
Math Basics

Computational Geometry

Question

- This **solid** model is made of **5030** triangles.
- No holes!
- How many vertices does it have?

1. 2517
2. 5032
3. 10,060
4. Cannot know



Descartes-Euler Polyhedral Formula

- For normal solid polyhedra:

  Vertices + Faces – Edges – 2 * Holes = 2

- So, for a simple model with 5030 Faces

  V + 5030 – E = 2

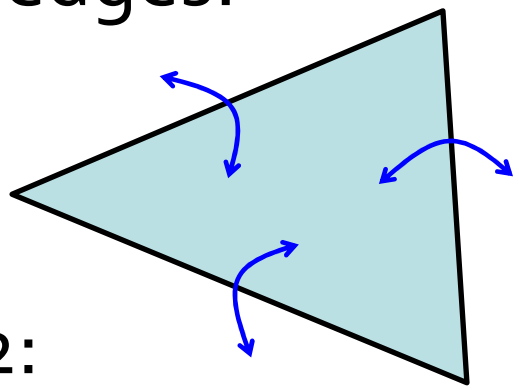
  V = E – 5028

- But now what?

Try Again

- We know that each edge is shared by 2 triangles, and each triangle has 3 edges:

  $E = 3/2 * F$



- So, substituting this into V+F-E=2:

  $V + F - 3/2 * F = 2$

  $V - F/2 = 2$

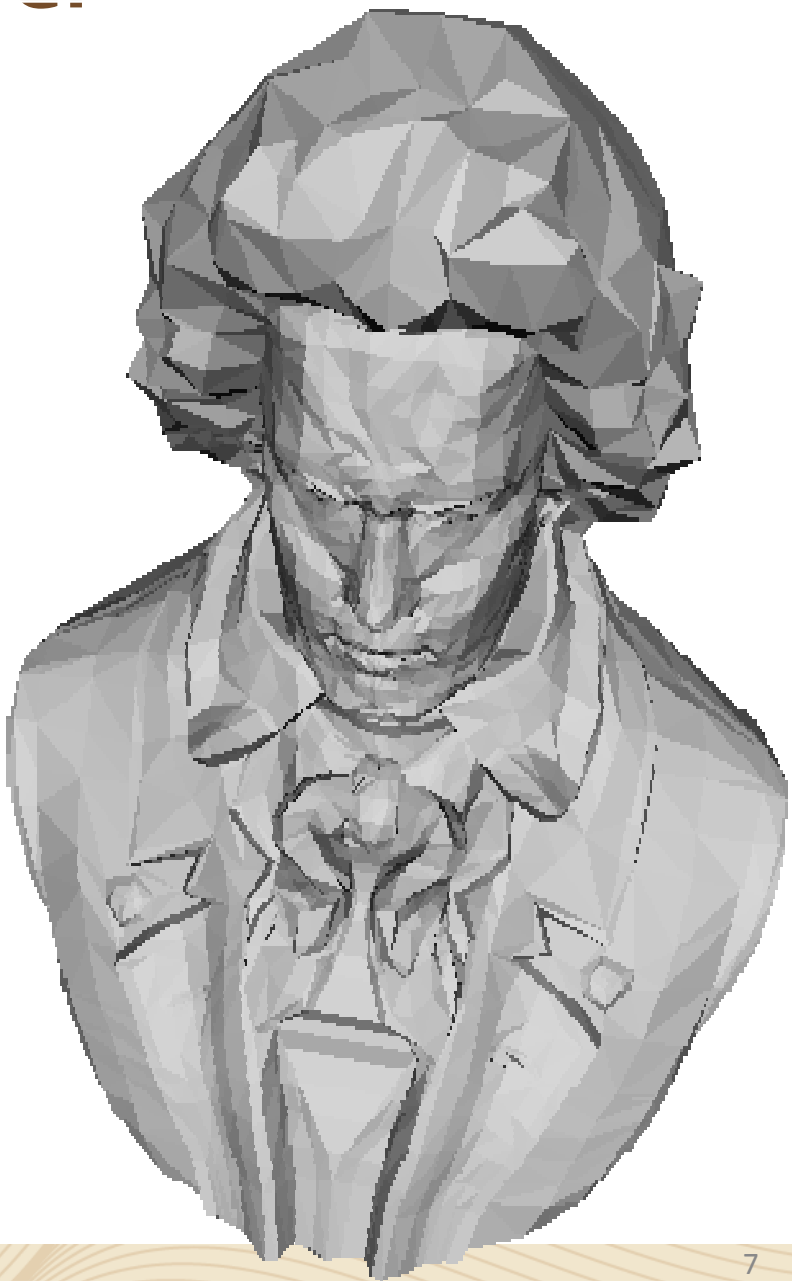
The Answer

- With V – F/2 = 2 and 5030 faces:

  V – 5030/2 = 2

  V = 2517

- Some more conclusions can be drawn…



Relationships

- Relationships for polyhedra made of triangles:

  E = 3/2 * F, derived for any triangle

  V – F/2 = 2, for a polyhedra

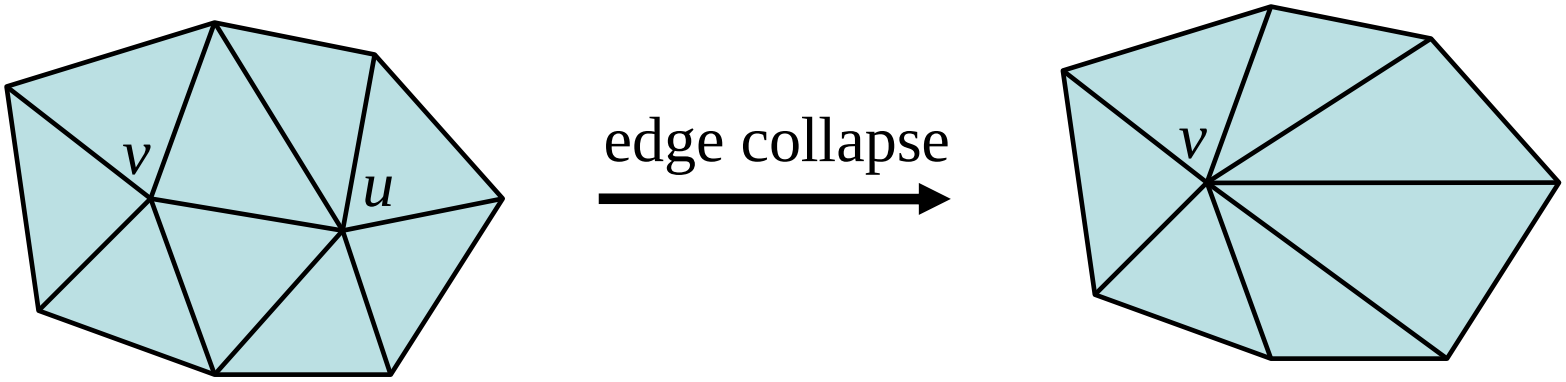
-> V – E/3 = 2

- *Knowing just the number of faces, edges, or vertices, you can get the other two.*

  V ~= F/2 ~= E/3

Decimation Example

Effect of Decimation



- Subtract vertex *u* by collapsing it and you get 1 less vertex, 2 less faces, and 3 less edges:

- Change in V = change in F/2 = change in E/3

Solid Mesh Properties, Part 1

- $V - F/2 = 2$

  Can F ever be an odd number?

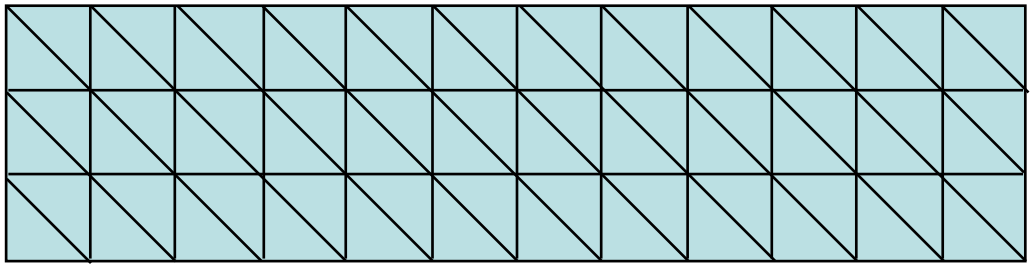
  No.

- So, the number of faces in a solid made of triangles must always be even. Surprising!

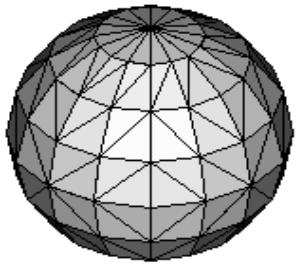
Solid Mesh Properties, part 2

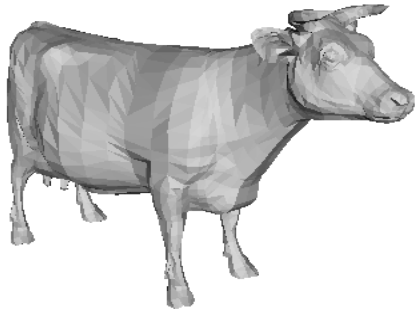
- $V - E/3 = 2$

  Each vertex will have an average of almost six edges meeting at it. Easy for a regular mesh:



  But, this is unobvious for an arbitrary mesh:





General Polyhedra

- For triangular faces, V ~= F/2 ~= E/3.
- For quadrilateral meshes, V ~= F ~= E/2.
- As V/F rises, the model has more quadrilaterals (or pentagons, hexagons, etc).

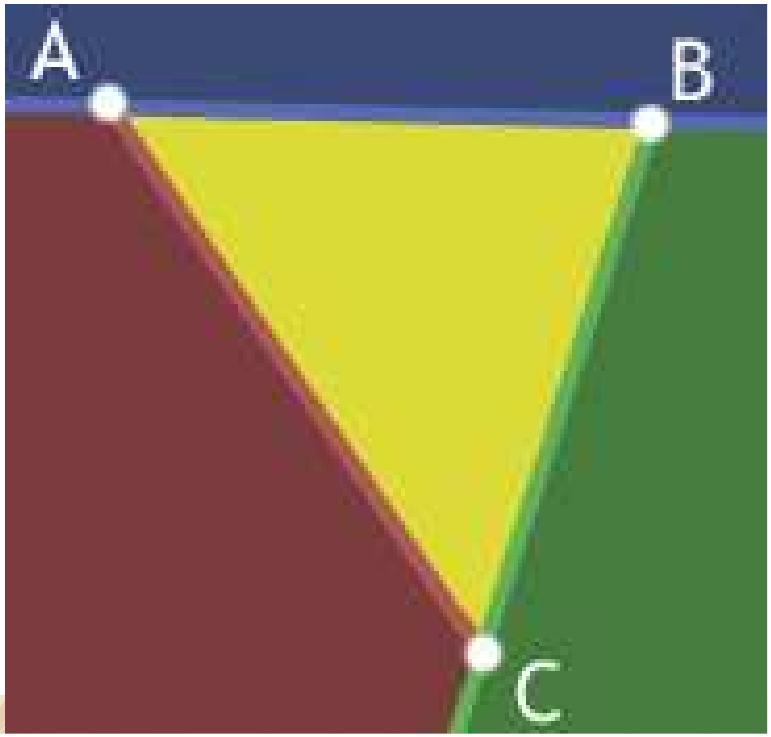
- **Warning:** these properties hold only when the model is a single solid polyhedron, with no extraneous "doubled" vertices or faces.

Relevance

- Pipeline development: you now know the expected amount and connectivity of vertex vs. face data in any mesh.
- Quick robustness check: you can quickly tell if a mesh is well-formed and so can be used for making shadow volumes and for other algorithms.

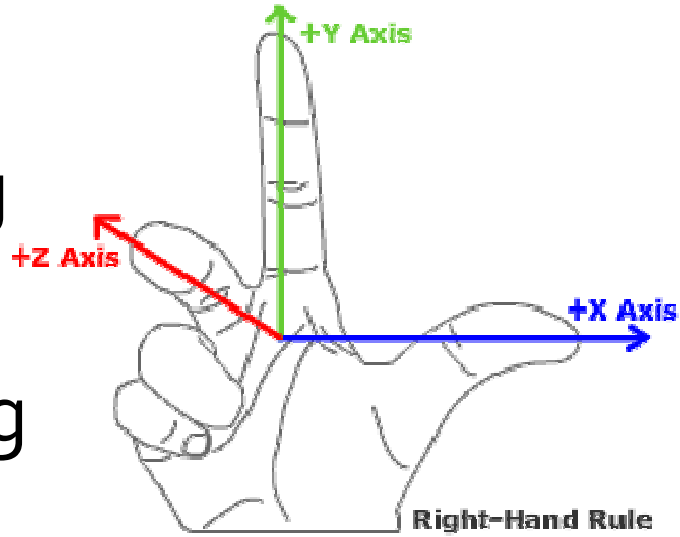
Point in Triangle Test

- Point p inside ABC iff (means if and only if)
  - "Below" AB and
  - "Left of" CB and
  - "Right of" CA
- It follows (contrapositive):
- Point p outside ABC iff
  - "Above" AB (area) or
  - "Right of" CB (area) or
  - "Left of" CA (area)



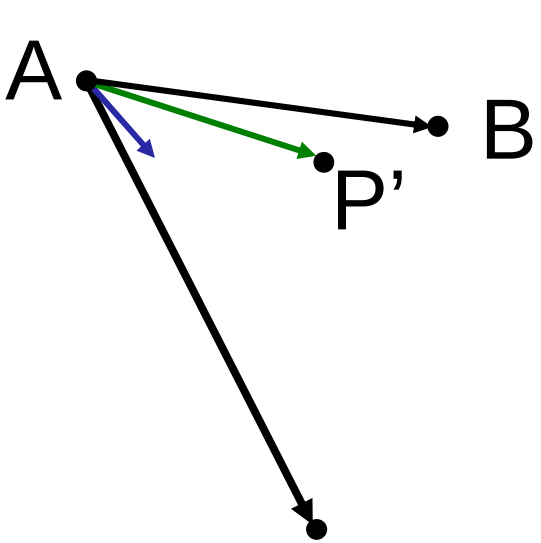
Point in Triangle Test

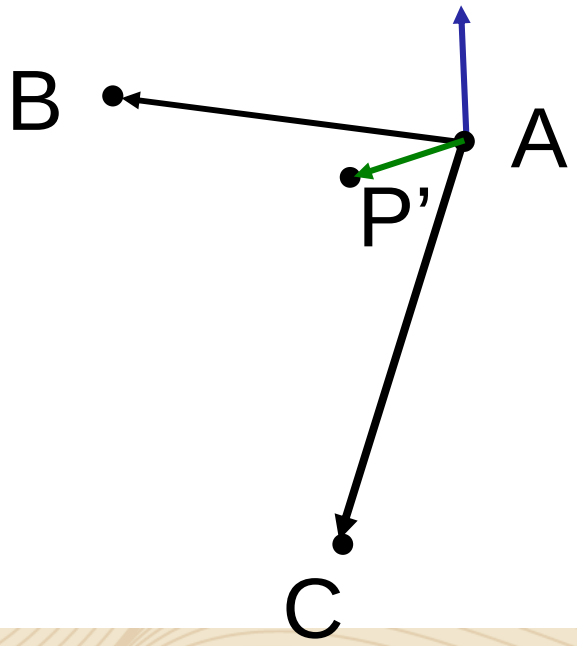
- Which side of line?
- [B-**A**] x [P-**A**] = vector pointing out of screen
- [B-**A**] x [P'-**A**] = vector pointing into screen
- Which direction indicates P is inside?



Point in Triangle Test

- Which direction indicates p is inside?
  - Problem: CW vs. CCW triangles
  - [B-**A**] x [P'-**A**] same direction as [B-**A**] x [C-**A**]
  - If CCW or CW => do not need [B-**A**] x [C-**A**]

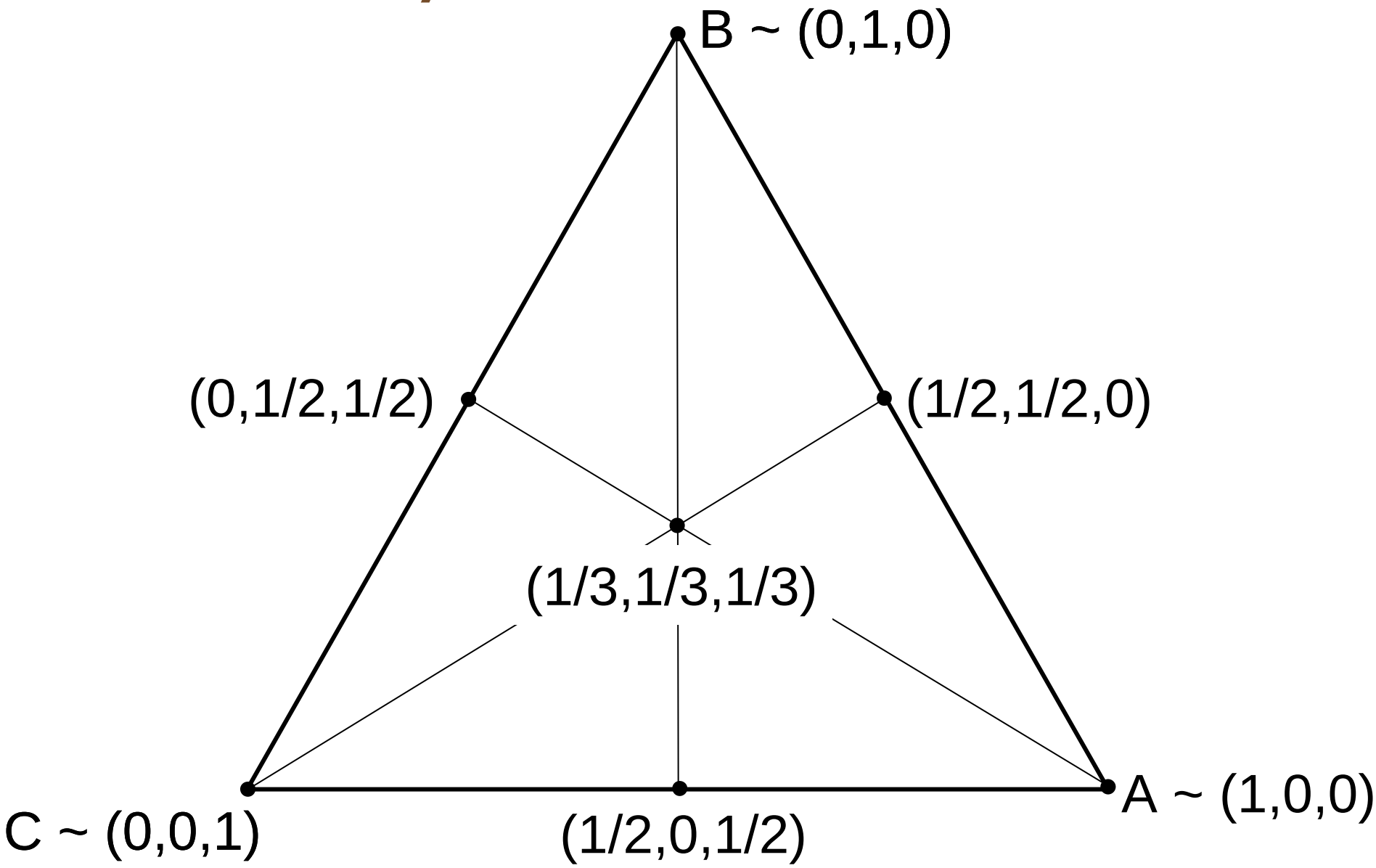




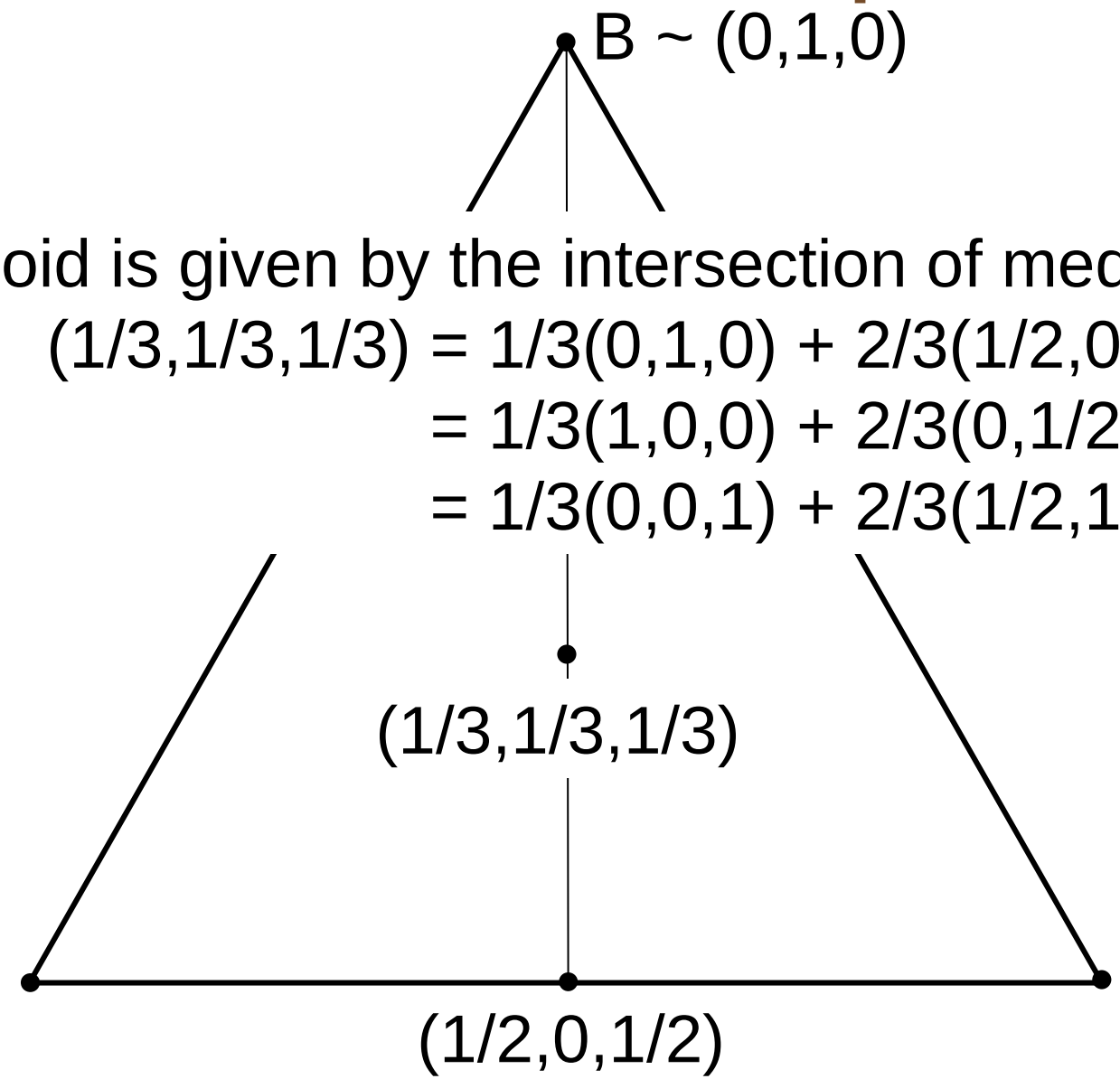
Barycentric Coordinates (*u*,*v*,*w*)

- Coordinates defined by the vertices (A,B,C,..) of a simplex (triangle, tetrahedron, …)
- We define A,B,C,… to be ordered CCW
- Local coordinate system
- P = *u*A+*v*B+*w*C 1= *u*+*v*+*w* (comb. of points)
- A ~ (1,0,0)
- B ~ (0,1,0)
- C ~ (0,0,1)
- Not independent: w=1-u-v
- Affinely invariant

Barycentric Coordinates

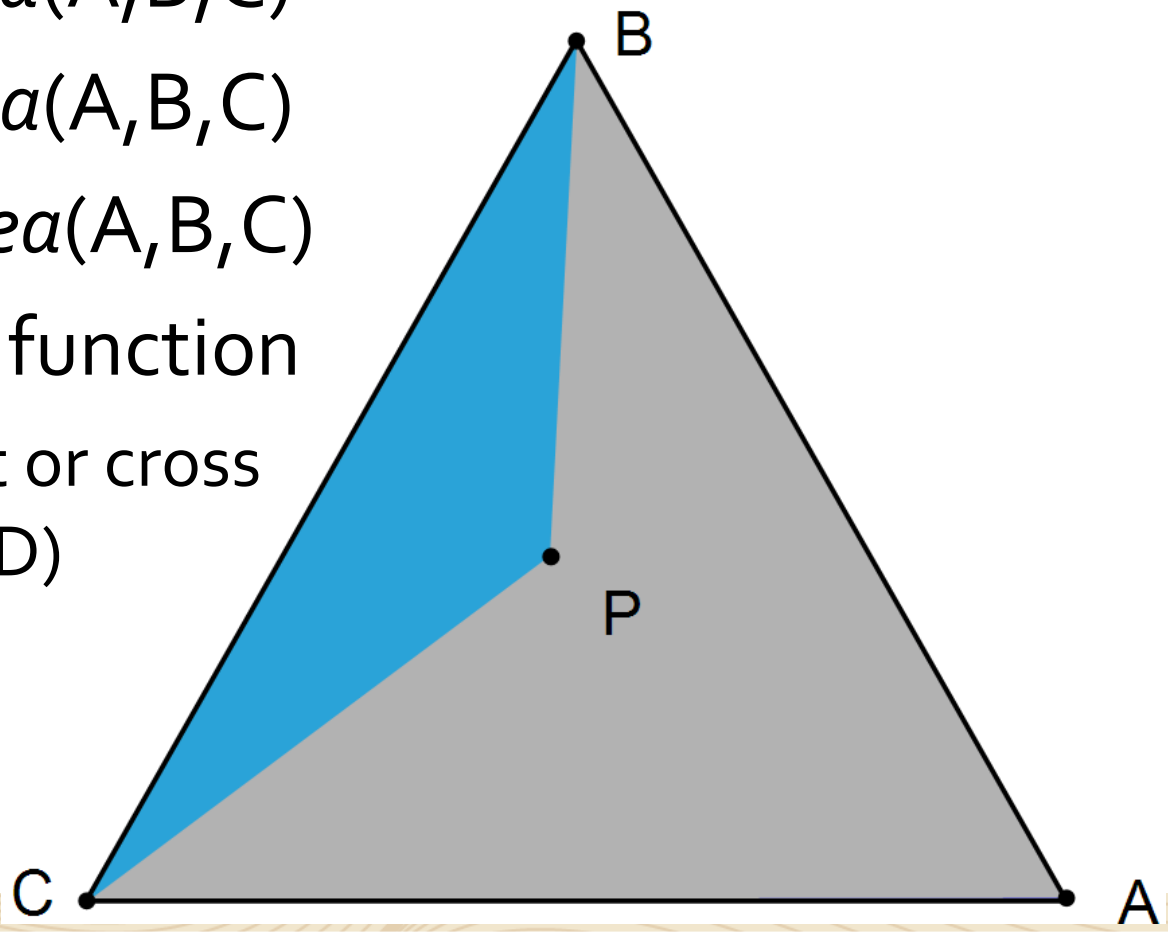


Barycentric Coordinates: Special Points



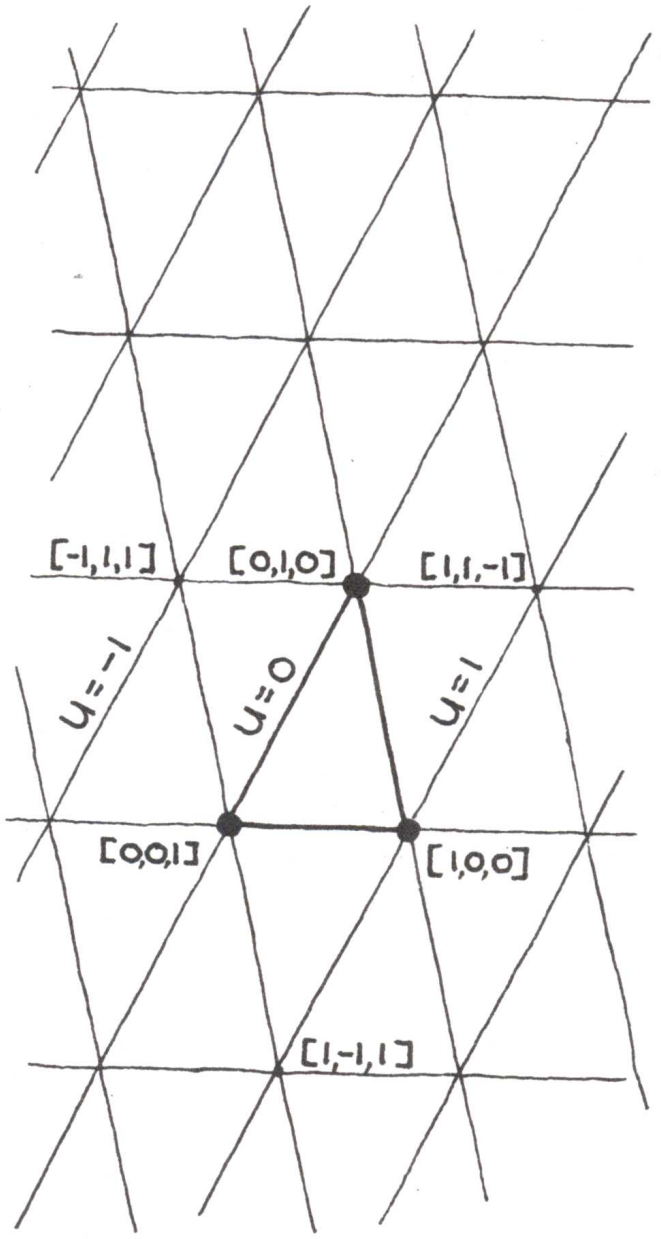
Barycentric Coordinates

- Barycentric Coordinates of P?
- $u = area(P,B,C)/area(A,B,C)$
- $v = area(P,C,A)/area(A,B,C)$
- $w = area(P,A,B)/area(A,B,C)$
- *area* is signed area function
  - With determinant or cross product (only in 3D)



Barycentric Coordinates

- Background for texture coordinate interpolation
  - Coordinates at vertices given
  - Every other point on surface interpolated



Barycentric Coordinates: Point in Triangle

- Point is in triangle if its barycentric coordinates
  (u,v,w) are all of the same sign
  - For CCW (u,v,w) >= 0
  - u = *area*(P,B,C)/*area*(A,B,C)
  - v = *area*(P,C,A)/*area*(A,B,C)
  - w = *area*(P,A,B)/*area*(A,B,C)
  - Division can be avoided (only sign is needed)!
- Ray triangle intersection
  - (u,v,w) gives the point of intersection on triangle

Ray Triangle Intersection

- Point on triangle

$$t(u,v) = (1-u-v)A + uB + vC$$

- Ray

$$r(t) = O + tD$$

- Intersection

$$O + tD = (1-u-v)A + uB + vC$$

Ray Triangle Intersection

- Intersection

$$O + tD = (1 - u - v)A + uB + vC$$

- Rearranged

$$O - A = \begin{pmatrix} -D & B - A & C - A \end{pmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix}$$

- Linear system!
- Solve with Cramer's rule

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{det(-D, B-A, C-A)} \begin{pmatrix} det(O - A, B - A, C - A) \\ det(-D, O - A, C - A) \\ det(-D, B - A, O - A) \end{pmatrix}$$

Ray Triangle Intersection: Implementation

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{det(-D,B-A,C-A)} \begin{pmatrix} det(O-A,B-A,C-A) \\ det(-D,O-A,C-A) \\ det(-D,B-A,O-A) \end{pmatrix}$$

- Rewrite using:

$$det(A,B,C) = -(A \times C) \cdot B = -(C \times B) \cdot A$$

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(D \times (C-A)) \cdot (B-A)} \begin{pmatrix} ((O-A) \times (B-A)) \cdot (C-A) \\ (D \times (C-A)) \cdot (O-A) \\ ((O-A) \times (B-A)) \cdot D) \end{pmatrix}$$

Ray Triangle Intersection: Implementation

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(D \times (C-A)) \cdot (B-A)} \begin{pmatrix} ((O-A) \times (B-A)) \cdot (C-A) \\ (D \times (C-A)) \cdot (O-A) \\ ((O-A) \times (B-A)) \cdot D) \end{pmatrix}$$

- Substituting :

$$E_1 = B - A \qquad E_2 = C - A \qquad S = O - A$$
$$P = D \times (C-A) \qquad Q = (O-A) \times (B-A)$$

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{P \cdot E_1} \begin{pmatrix} Q \cdot E_2 \\ P \cdot S \\ Q \cdot D \end{pmatrix}$$

Ray Triangle Intersection: Code

```
bool rayTriIntersect(in O,D, A,B,C, out u,v,t) {
    E1 = B-A
    E2 = C-A
    P = cross(D,E2)
    detM = dot(P,E1)
    if(detM > -eps && detM < eps)
        return false          0 == detM
    f = 1/detM
    S = O-A
    u = f*dot(P,S)
    if(0 > u || 1 < u)
        return false      u outside [0,1]
    Q = cross(S,E1)
    v = f*dot(Q,D)
    if(0 > v || 1 < u+v)
        return false
    t = f*dot(Q,E2)
    return true
}
```

vectors scalars

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{P \cdot E_1} \begin{pmatrix} Q \cdot E_2 \\ P \cdot S \\ Q \cdot D \end{pmatrix}$$

Computational Geometry

The study of algorithms for combinatorial, topological, and metric problems concerning sets of points, typically in Euclidean space. Representative areas of research include geometric search, convexity, proximity, intersection, and linear programming.

Online Computing Dictionary

Computational Geometry

- Previously: design and analysis of geometric algorithms
- Overlapping and merging with discrete geometry
- Now: **study of geometrical problems from a computational point of view**

Handbook of Discrete and Computational Geometry

Computational Geometry

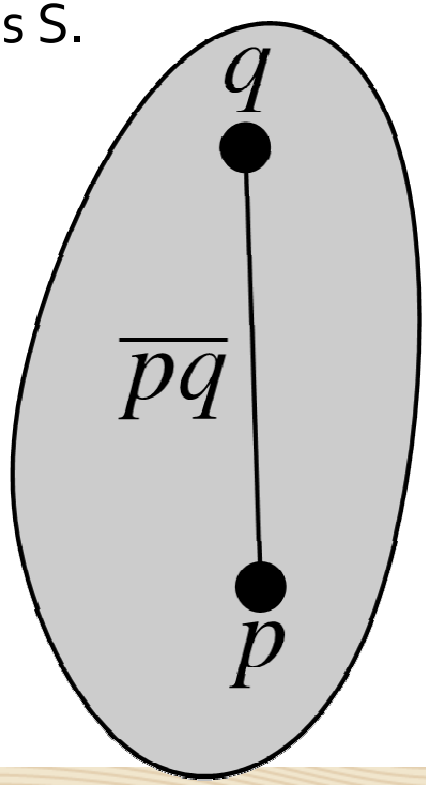
- **Basic objects**: points, lines, line segments, polygons, polygonal lines, embedded graphs
- **Computed objects**: convex hull, alpha hull, triangulation, arrangement, Voronoi diagram, Delauney triangulation.
- **Variations**: static, dynamic (discrete changes), kinetic (continuous motion)
- **Wanted: good algorithms**

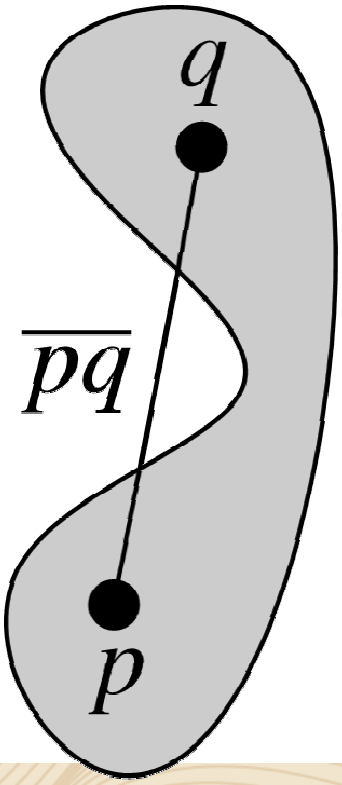
Planar Convex Hull

- Definition:

  A subset S of the plane is called convex if and only if for any pair of points p,q ∈ S the line segment pq is completely contained in S. The *convex hull* CH(S) of a set S is the smallest convex set that contains S.

convex



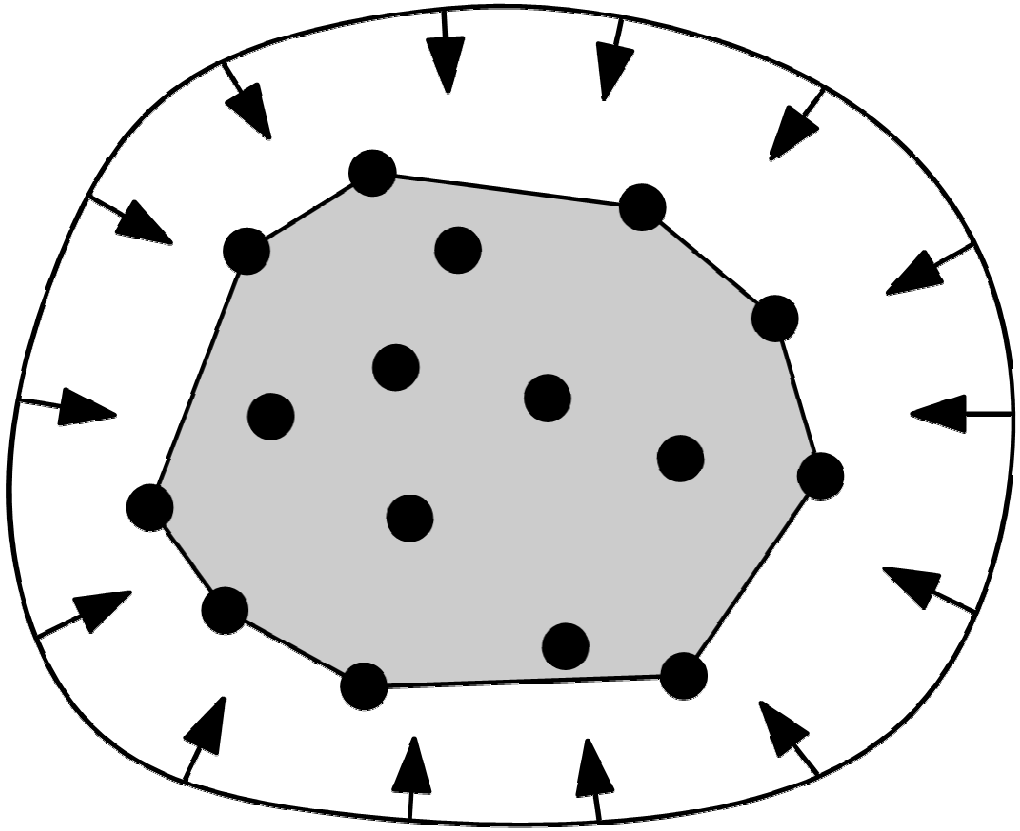


not convex

Planar Convex Hull

- Intuition:

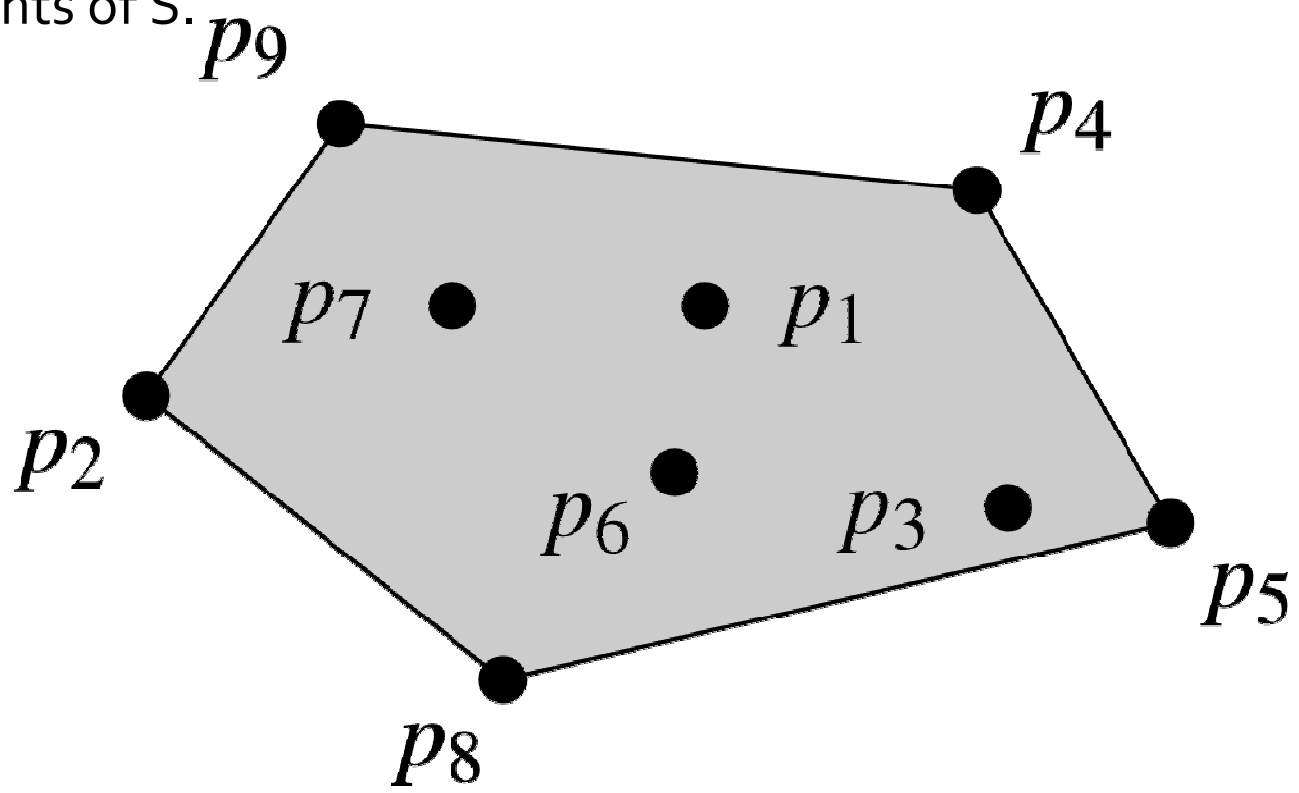
  If there is a board with nails sticking out from it. And S is the set of nails. Then the *convex hull* of S, CH(S) can be thought of the shape formed by a tight rubber band that surrounds all the nails.



Planar Convex Hull

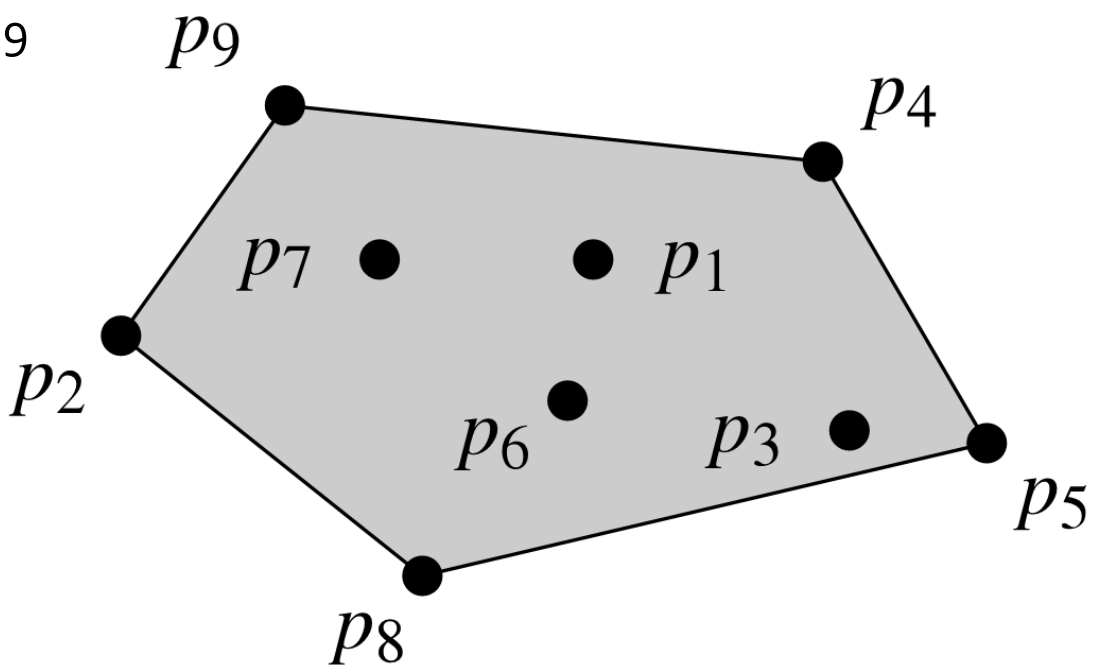
- Alternate definition (for finite sets):

  The *convex hull* of a finite set S of points in the plane is the unique convex polygon whose vertices are points from S and that contains all points of S.



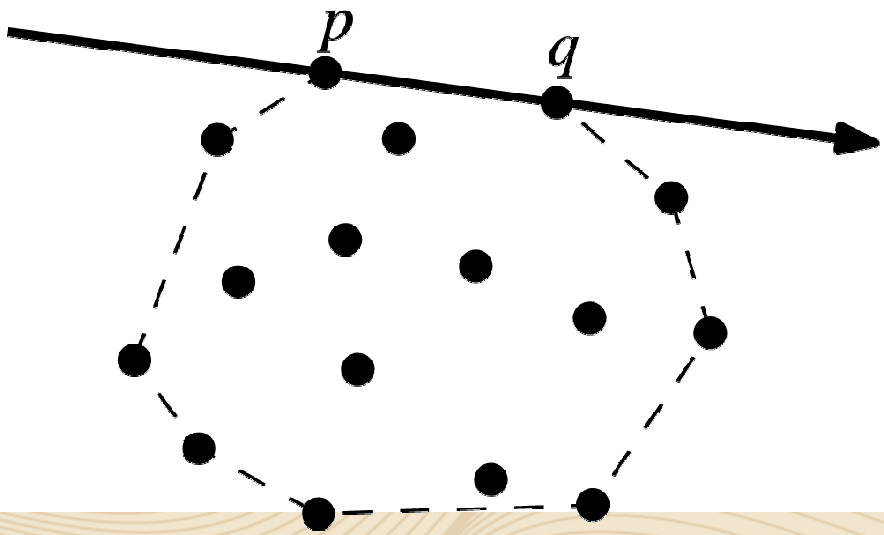
Planar Convex Hull: General Algorithm

- Input = set of n points
  - Here $p_1..p_9$
- Output = representation of convex hull
  - Here p_4, p_5, p_8, p_2, p_9



Planar Convex Hull: First Idea

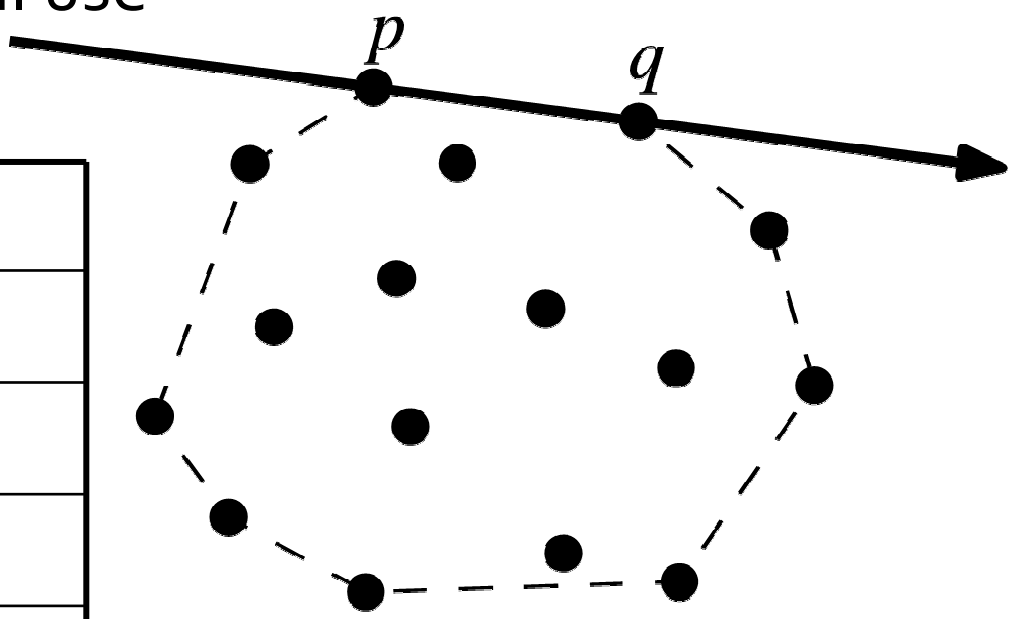
- Observation: (p,q) is an edge of the convex hull, if all other points lie to the right of the line.
- Idea: If we check this for every ordered pair, we find all edges of the convex hull.



Complexity

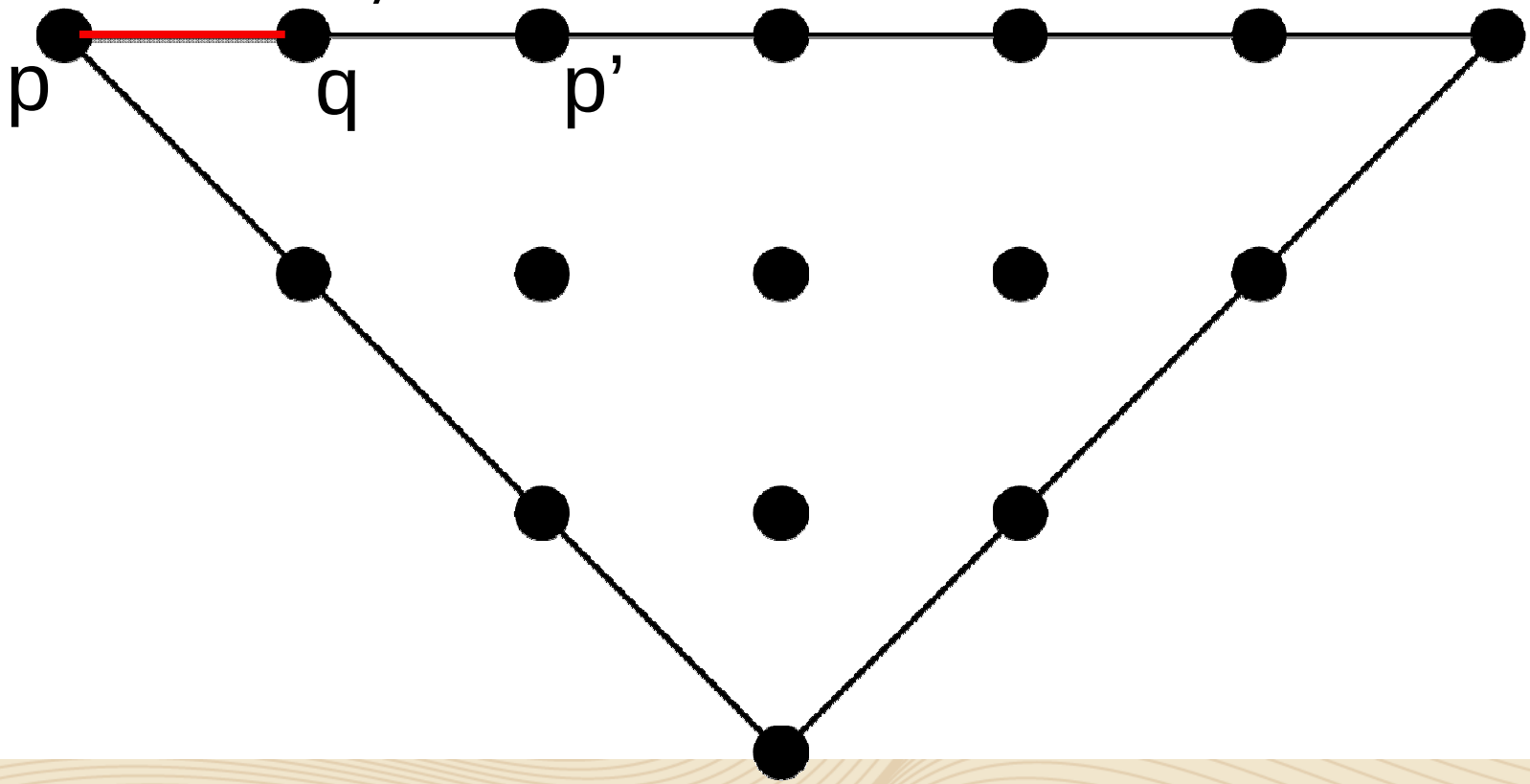
- Number of ordered pairs: $n*(n-1) = n^2-n$
- For each pair we have to check n-2 points
- Totals to $(n^2-n)*(n-2) = n^3-2n^2-2n \sim O(n^3)$
  - To slow for practical use

n	n^3
10	1.000
100	1.000.000
1.000	1.000.000.000
10.000	1.000.000.000.000



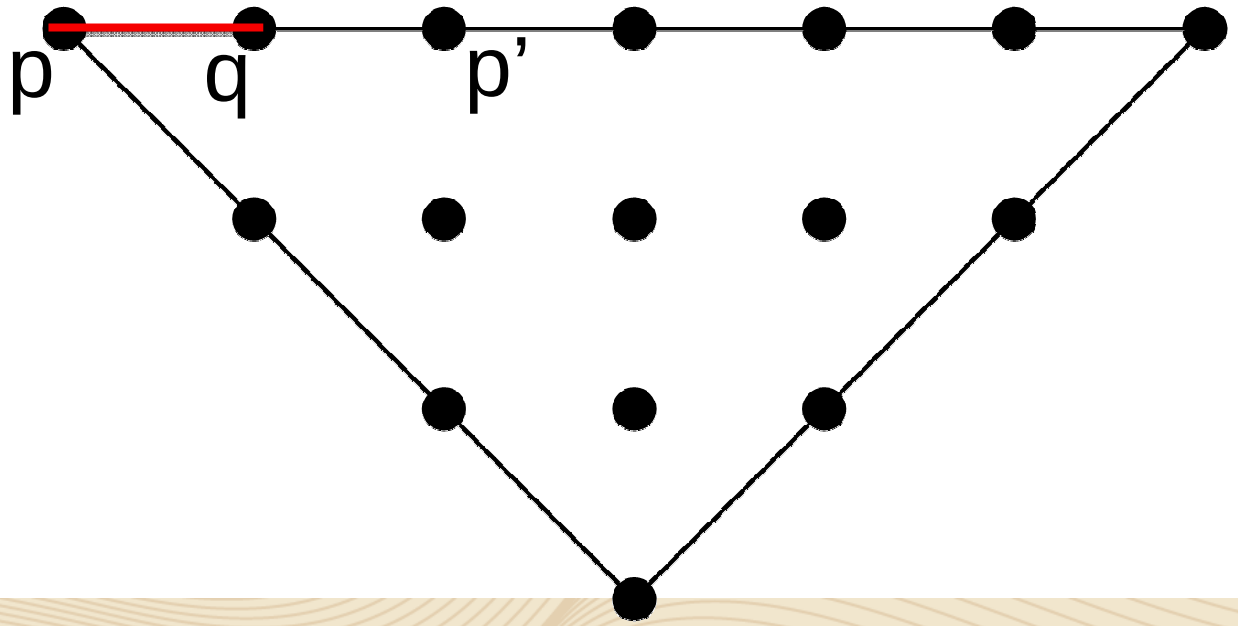
Correctness

- Observation: (p,q) is an edge of the convex hull, if all other points lie to the right of the line.
  - Is not always correct



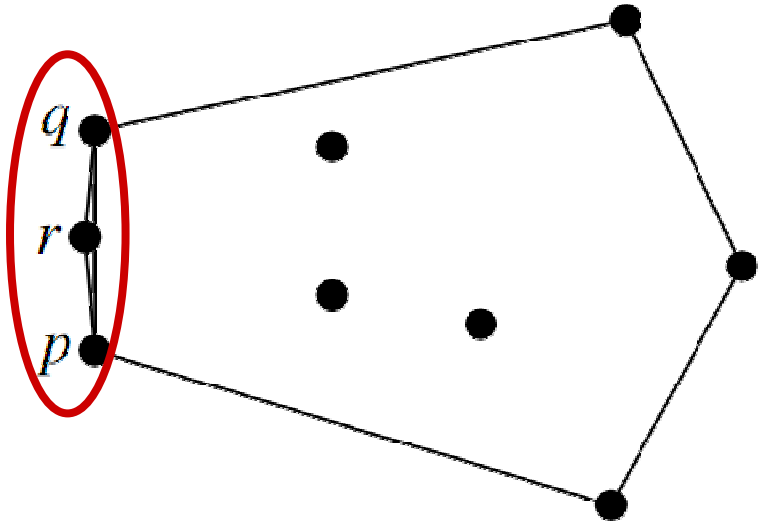
Degenerated Cases

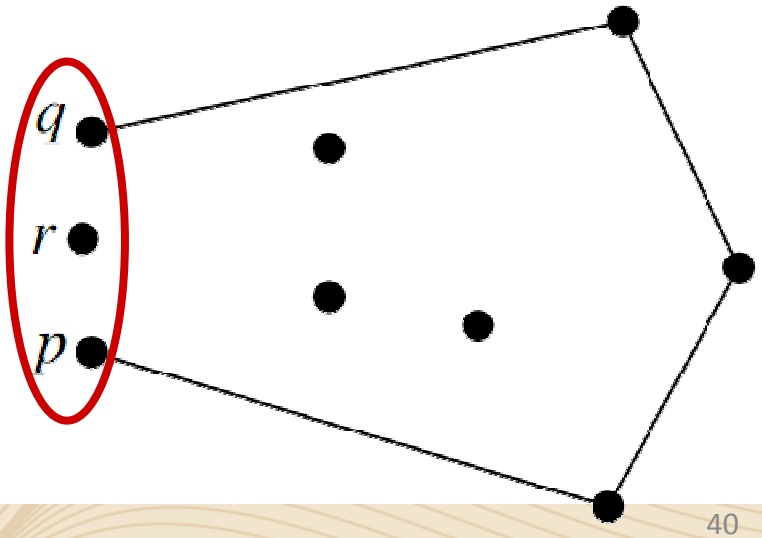
- Our algorithm does not handle degenerated cases so far
  - Maybe hole in convex hull
  - Maybe crash (division by zero)
  - Maybe incorrect Results



Floating Point Issues

- We test pairs (p,q), (p,r) and (r,q)
- Due to rounding errors
  - Maybe r right of (p,q)
  - Maybe p right of (r,q)
  - Maybe q right of (p,r)
  - Geometrical impossible
- Outcome: algorithm accepts
  - All 3 edges
  - Rejects all 3 edges
  - Some combination





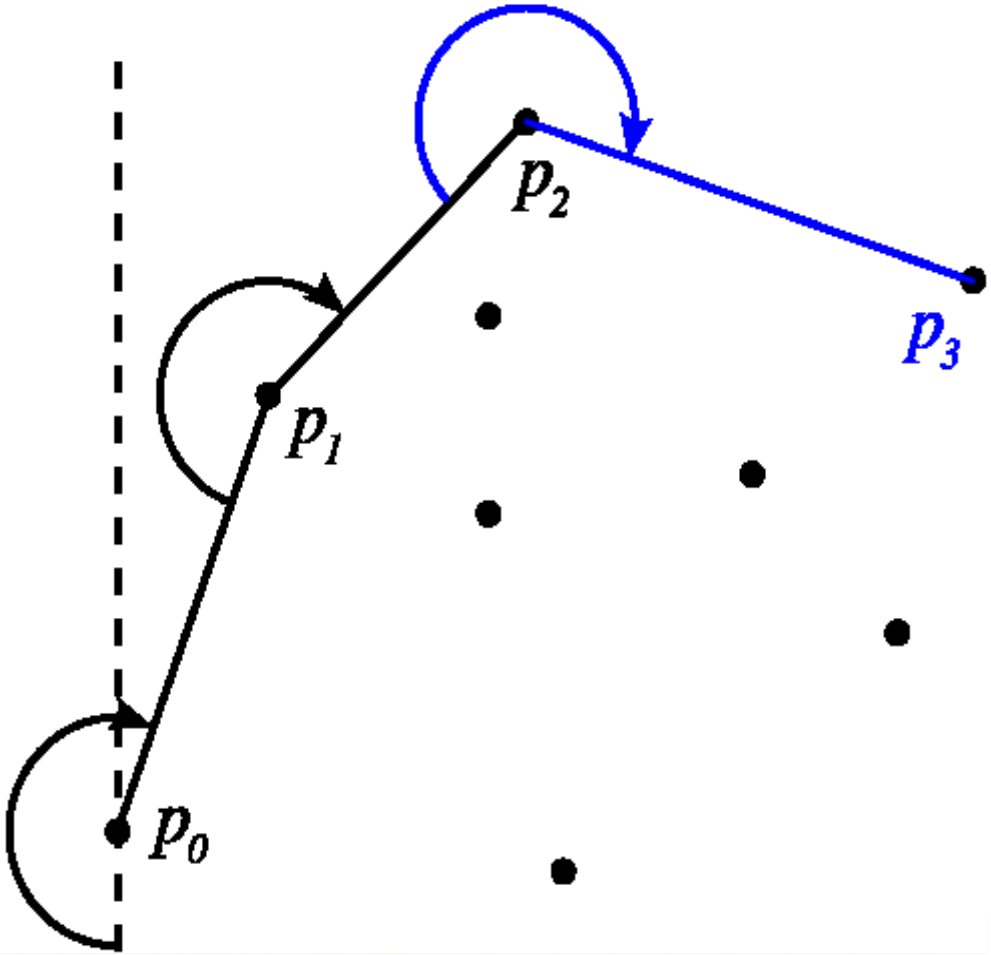
Planar Convex Hull: First Idea - Résumé

- Observation: (p,q) is an edge of the convex hull, if all other points lie to the right of the line.
- Idea: If we check this for every ordered pair, we find all edges of the convex hull.

- **Speed**: slow - $O(n^3)$
- **Correctness**: bad - does not handle collinear points
- **Robustness**: bad - small round-off errors can lead to a failing algorithm

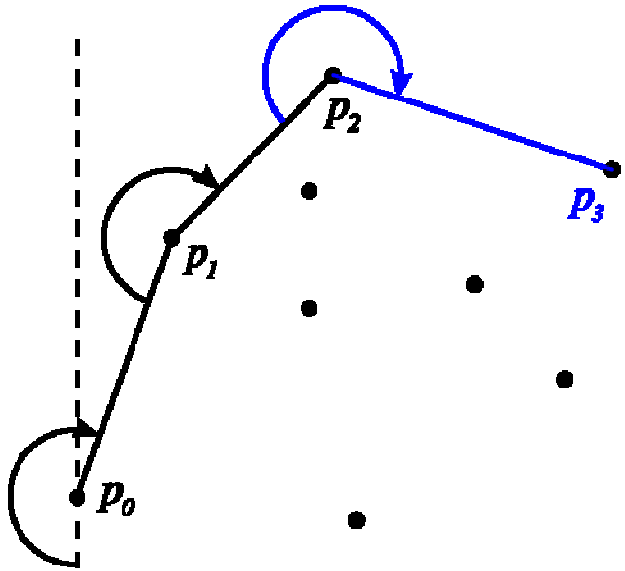
Jarvis's March (Gift Wrapping)

- Jarvis March computes the convex hull of a set S of **n** points by a technique called gift wrapping.
- Taut piece of paper wrapped around the set S
- Start with "anchor" point (point on convex hull
- Make line with every other point
- Select the one with the least angle
- Repeat



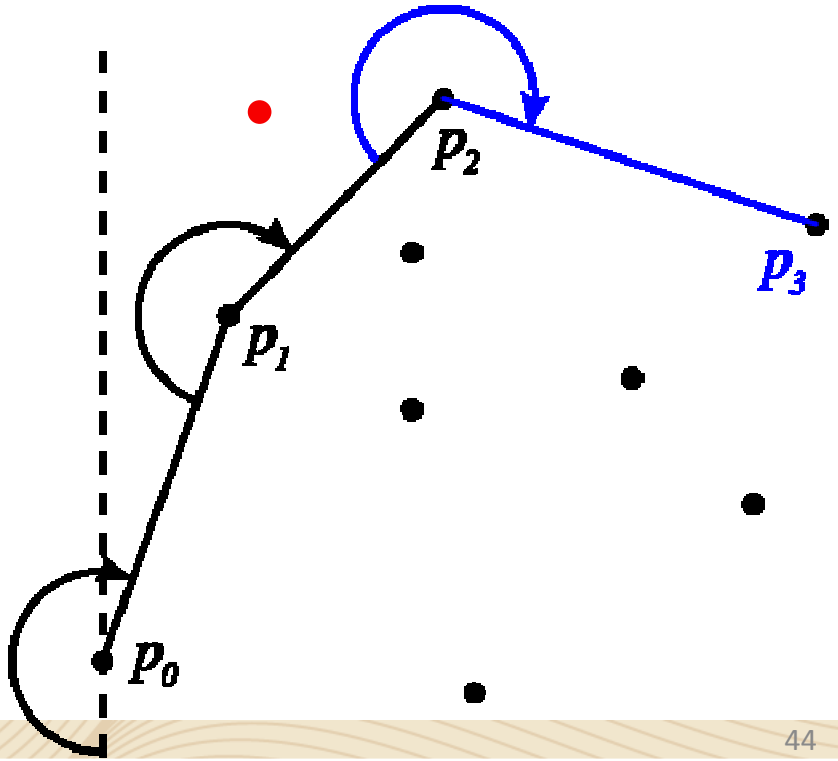
Jarvis's March (Gift Wrapping)

- Similar to previous approach
- But iterative
- Speed
  - Finding the anchor point: O(n)
  - For each point on convex hull check n other points
    h..number of points on convex hull
    O(n*h)
  - Total: O(n)+O(n*h) = O(n*h)



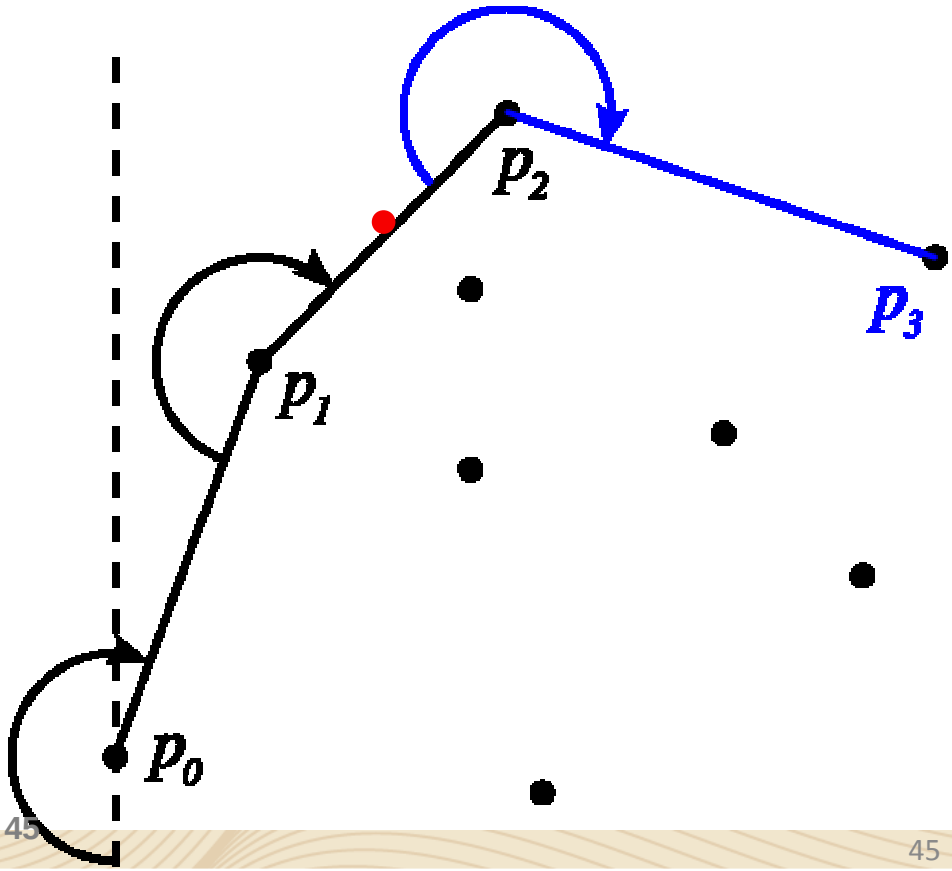
Jarvis's March (Gift Wrapping)

- Correctness - Good
  - Can handle collinear points
  - Rounding errors can lead to neglected points



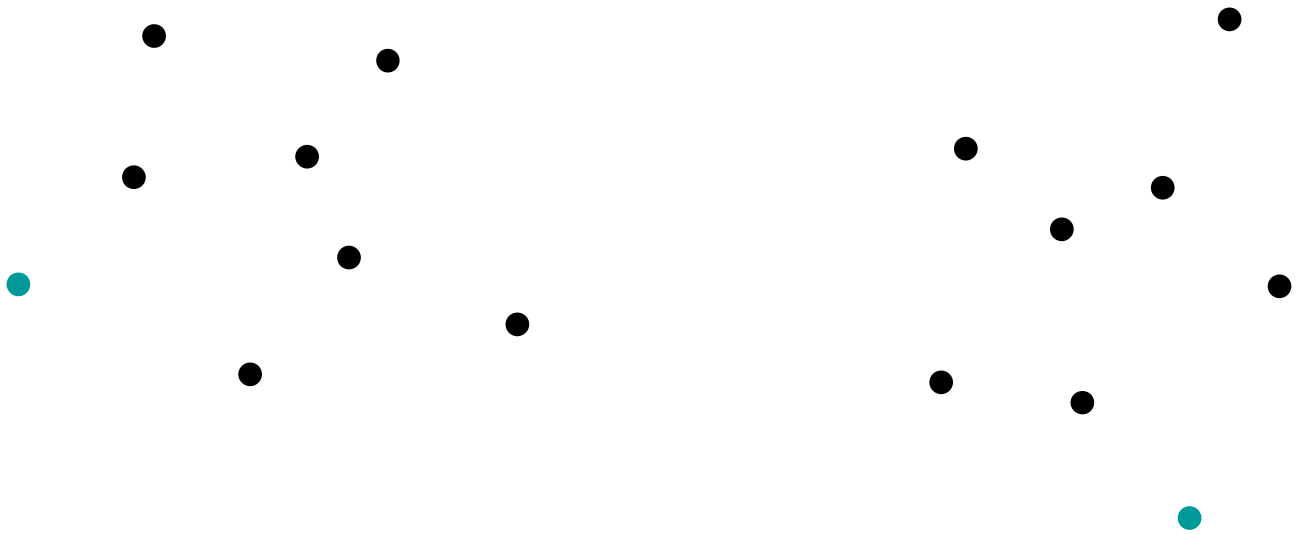
Jarvis's March (Gift Wrapping)

- Robustness - Good
  - Small round-off errors lead to small errors



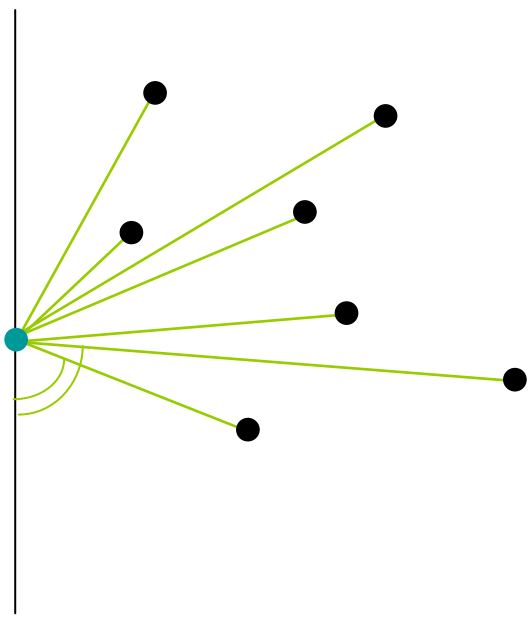
Graham's Scan

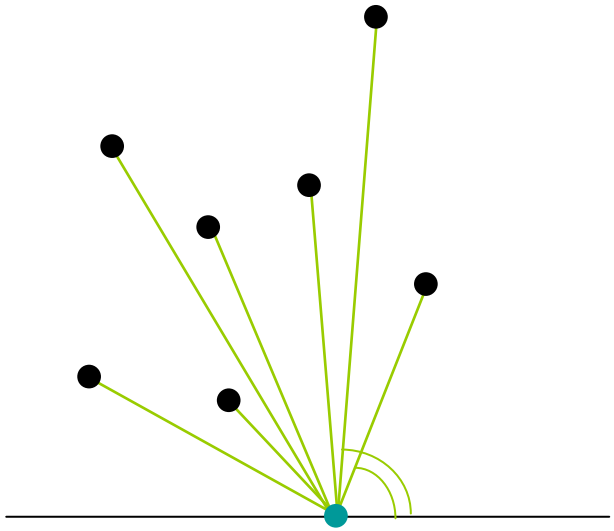
- Find an anchor point (leftmost, lowest,…)
  O(n)



Graham's Scan

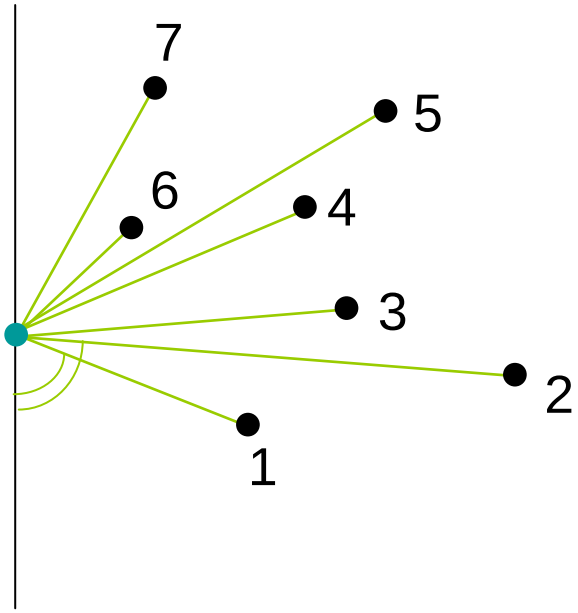
- From this point calculate angles to all other points

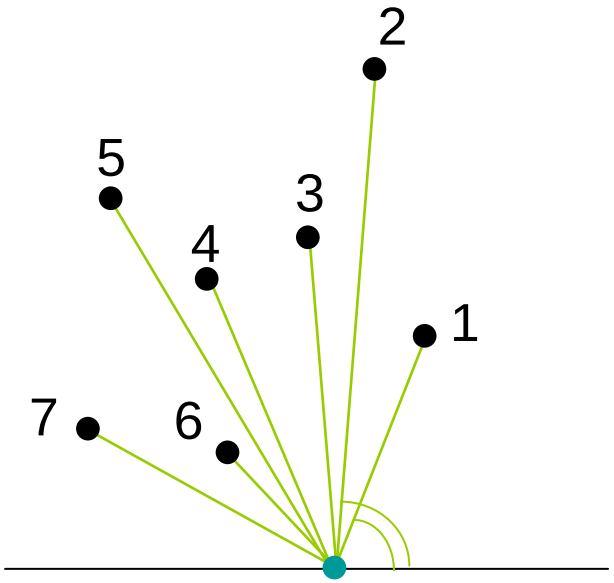




Graham's Scan

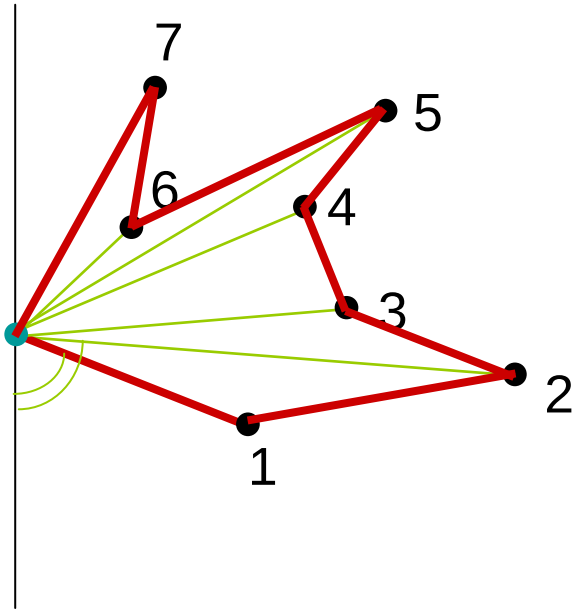
- Sort points by angles O(n*log(n))





Graham's Scan

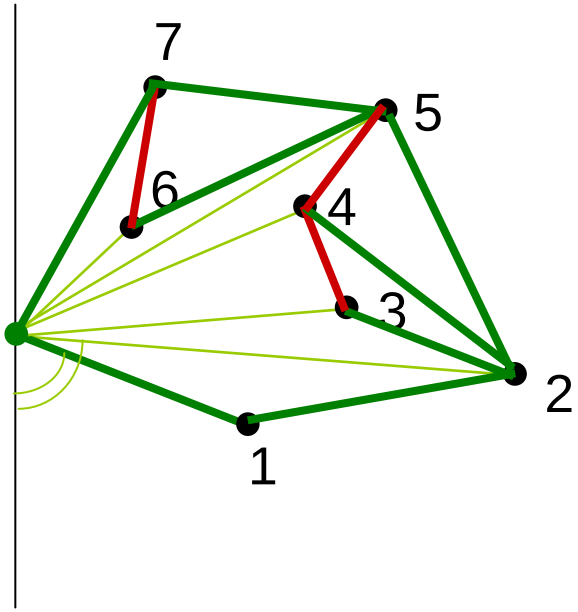
- Tries to follow points in order
- Results in walk along objects boundaries



1
2
3
4
5
6
7

Graham's Scan

- Whenever it takes right turns it backtracks (in the sorted list of points) and re-joins those points that makes the shortest path. O(n)



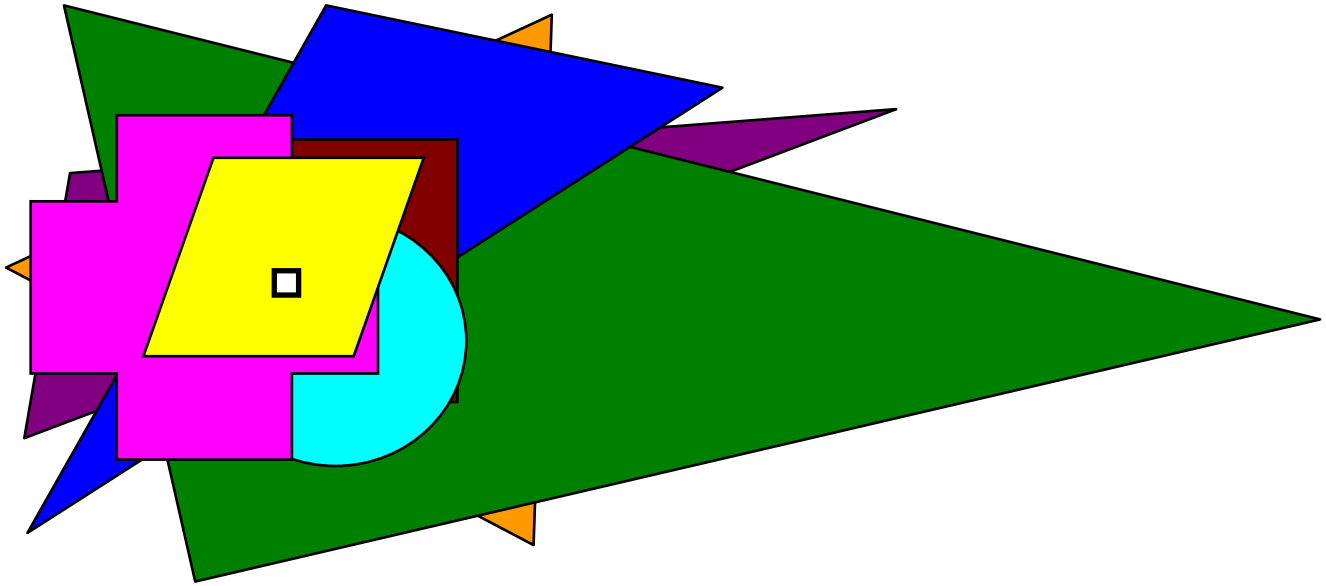
1
2
3
4
5
6
7

Graham's Scan

- Speed: O(n*log(n)) generally faster than gift wrapping for many points
- Correctness: good, calculations can be done using rational numbers
- Robust: very robust

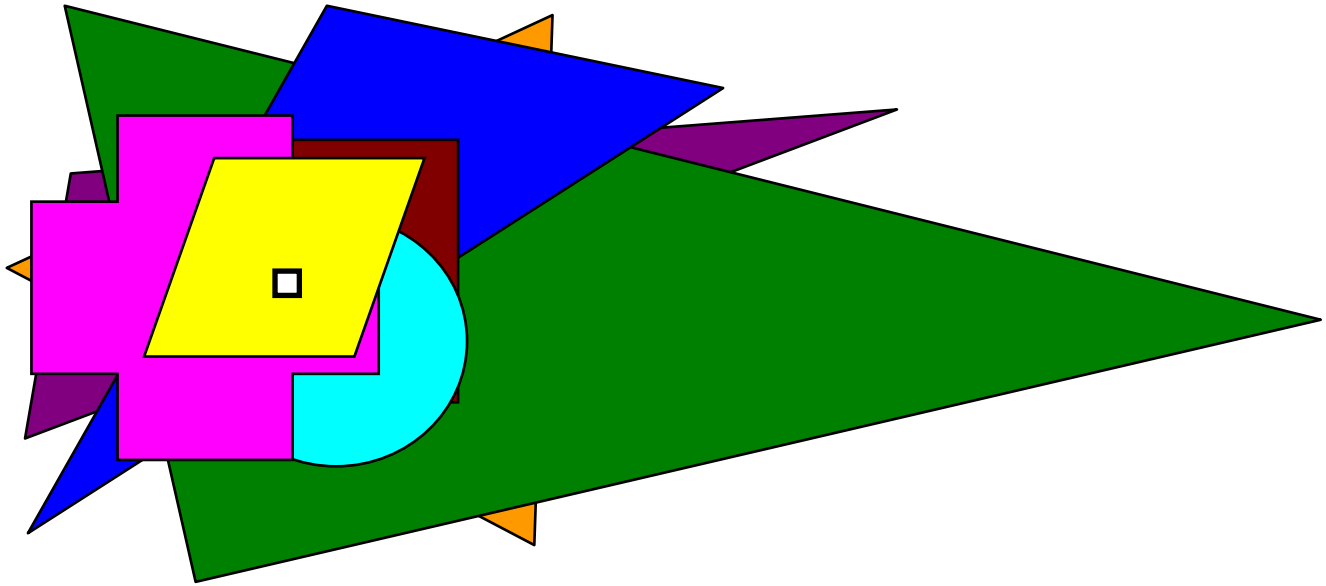
Question

- Ten 3D polygons cover a pixel. Drawing these in random order with a Z-buffer, what is the average number of times the color of the pixel is replaced?



Intuitive, and Wrong, Answer

- Drawing front to back, I would paint the pixel once. Back to front, I would paint it ten times. So, the average is 5.5 draws.



Right Answer

- First polygon drawn must cover pixel: 1.
- Second polygon drawn is in front or behind first polygon, 50/50, so chance is: ½.
- Third polygon has one chance in three of being the closest polygon drawn.

- 1 + 1/2 + 1/3 + …+ 1/10 = 2.9289…

Slowly Growing Function

- This function, the *harmonic series*, slowly grows to infinity:

1 poly	1 draw
4 polys	2.08 draws
11 polys	3.02 draws
31 polys	4.03 draws
83 polys	5.00 draws
…12,367 polys	10.00 draws