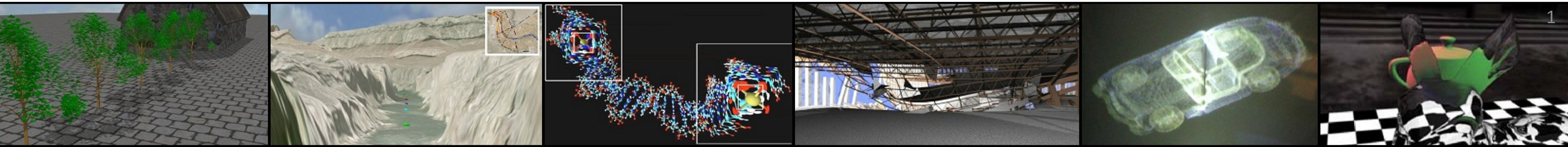
COT 4521: INTRODUCTION TO COMPUTATIONAL GEOMETRY

Introduction and Preliminaries

Paul Rosen
Assistant Professor
University of South Florida

Some slides from Valentina Korzhova



2004

Problem Domain

Computer Graphics

Problem Solving Approach

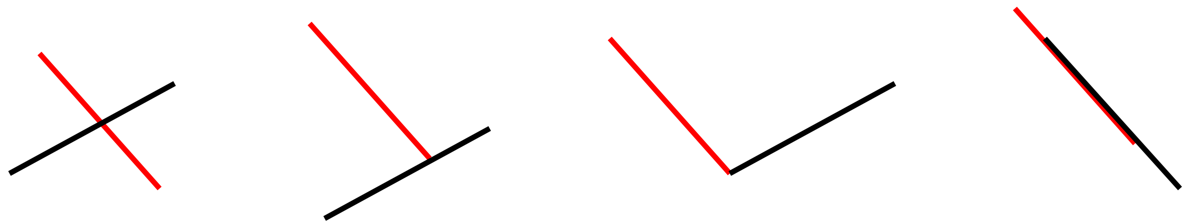
Geometric Approaches

Introduction

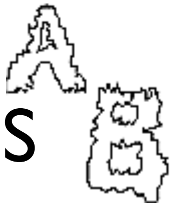
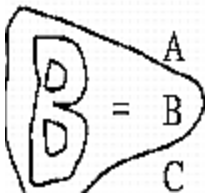
- What is computational geometry?
  - A subfield of the Analysis of Algorithms
  - Design of efficient algorithms and data structures for geometrical problems
- Started in mid 70's
  - started out by developing solid theoretical foundations, but became more and more applied over the last years
- Deals with geometrical structures
  - Points, lines, line segments, vectors, planes, etc.
- We'll focus mostly on 2-dimensional geometry, but 3D will be covered when possible

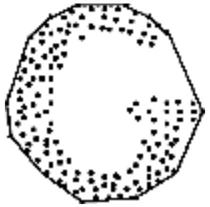
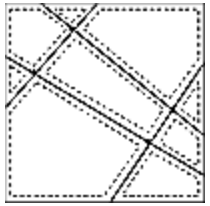
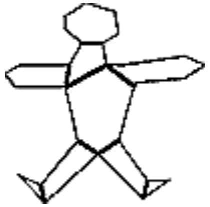
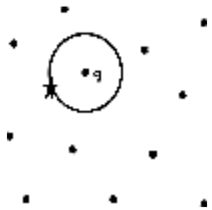
What makes Computational Geometry problems so Interesting/annoying?

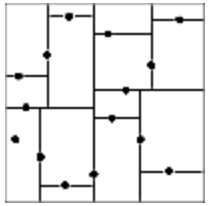
- Complexity
  - Curse of dimensionality
- Precision error
  - Avoid floating-point computations whenever possible
- Degeneracy
  - Boundary cases
  - For example, imagine how two line segments can intersect



Typical Computational Geometry Problems

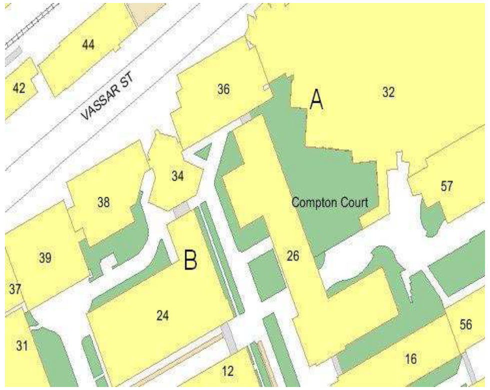
- Voronoi diagram 
- simplifying polygons 
- shape similarity 

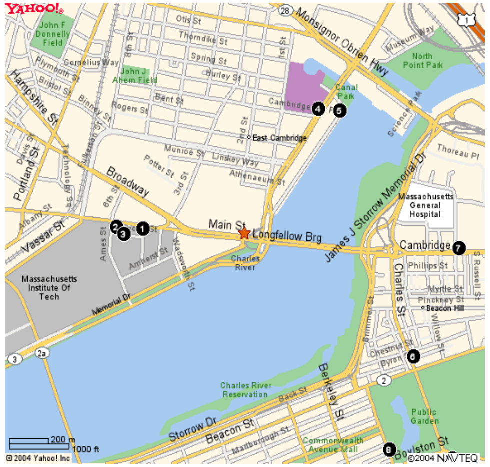
- convex hull 
- maintaining line arrangements 
- polygon partitioning 
- nearest neighbor search 

- kd-trees 

Applications

- Computer Graphics
- Robotics motion planning
- Geographic Information
- CAD/CAM
- Collision detection
- Computer vision
- Molecular modeling
- VLSI design
- Data Mining, Machine Learning, and Visualization



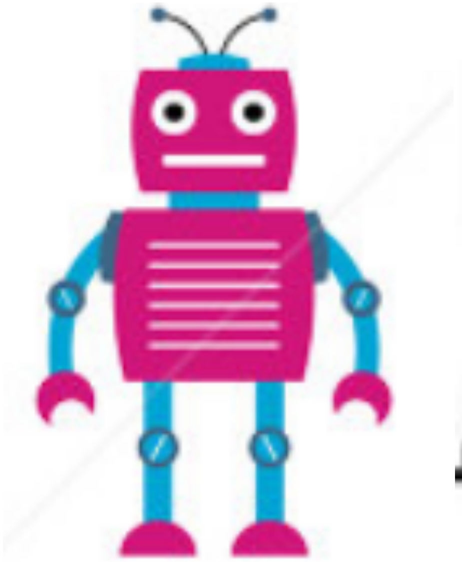


Applications of Computational Geometry

- Computer Graphics
  - For 2-dimensional graphics, typical questions involve the intersection of certain primitives, determining the subset of primitives that lie within a particular region, etc.
  - For 3-dimensional graphics, hidden surface removal

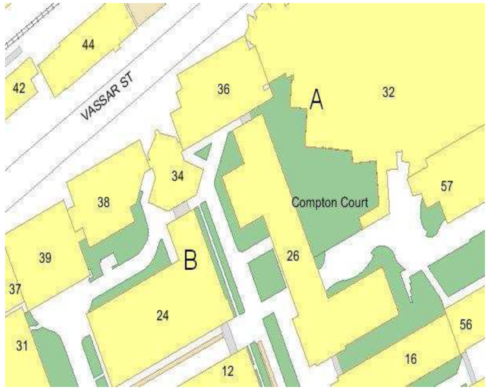
Applications of Computational Geometry

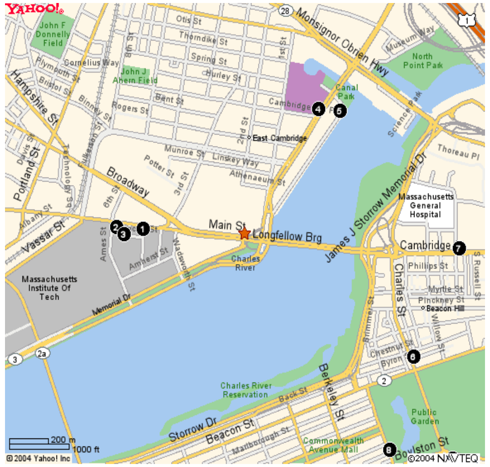
- Robotic motion planning
  - The geometric problem arise at many places because robots are geometric objects that operate in a 3-dimensional space



Applications of Computational Geometry

- Geographic Information system (GIS)
  - Stores geographical data
  - Can be used to extract information about certain regions and to obtain information between different types of data



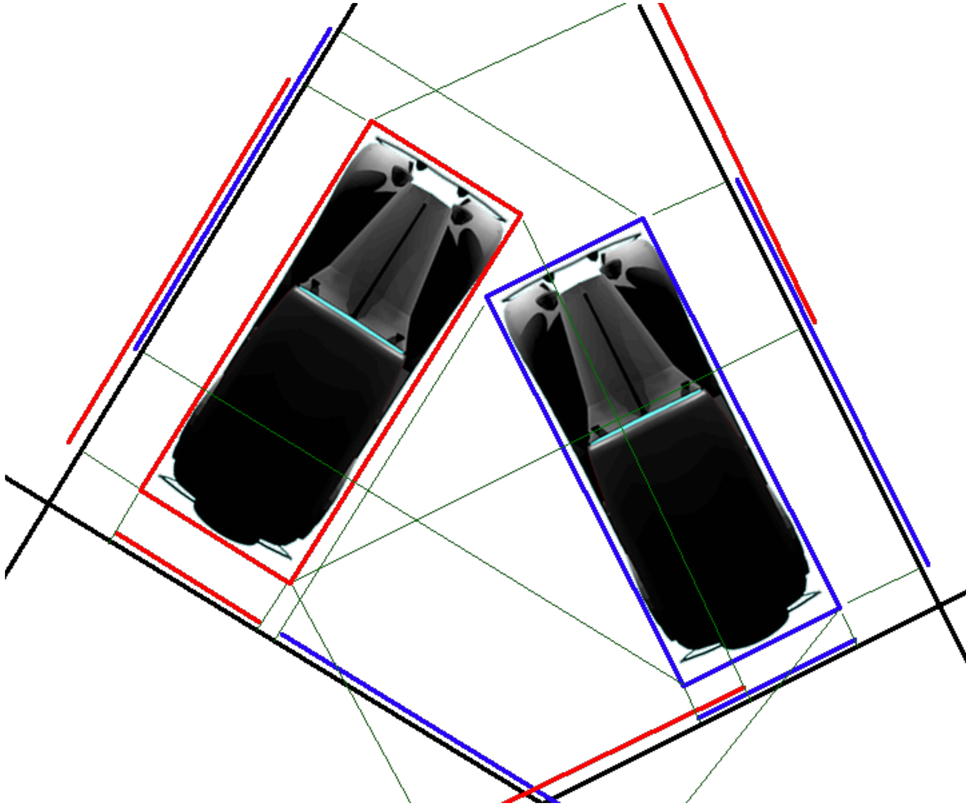


Applications of Computational Geometry

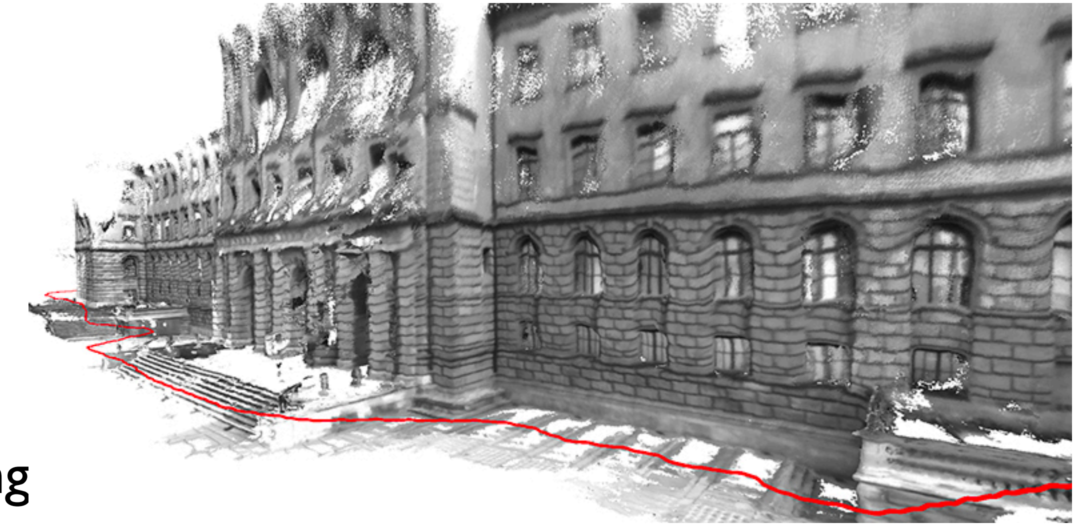
- CAD/CAM
  - Computer-Aided Design
    - Deal with intersections and unions of objects, with decompositions objects and object boundaries into simpler shapes, and with visualizing the designed products
  - Computer-Aided Manufacturing
    - Computer controls the actual manufacturing process

Applications of Computational Geometry

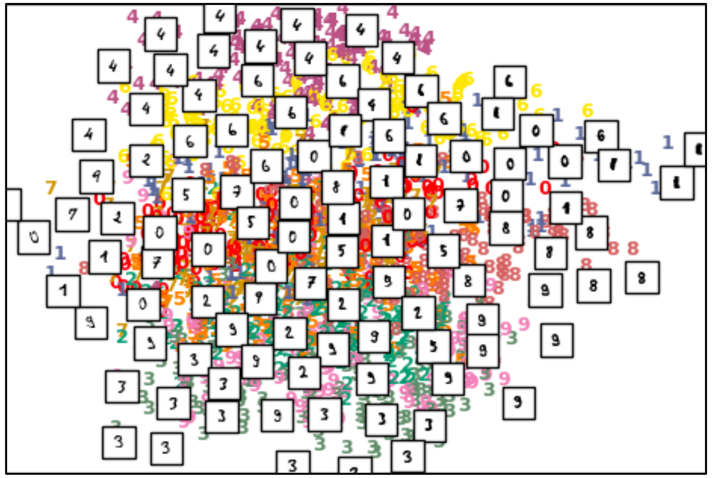
- Collision detection
  - Check whether two (possibly complicated) 3d objects intersect!
    - Approximate the objects by simple ones that enclose them (bounding volumes)
    - Popular bounding volumes: boxes, spheres, ellipsoids,...
    - If bounding volumes don't intersect, the objects don't intersect, either
    - *Only* if bounding volumes intersect, apply more expensive intersection test(s)



APPLICATIONS OF COMPUTATIONAL GEOMETRY

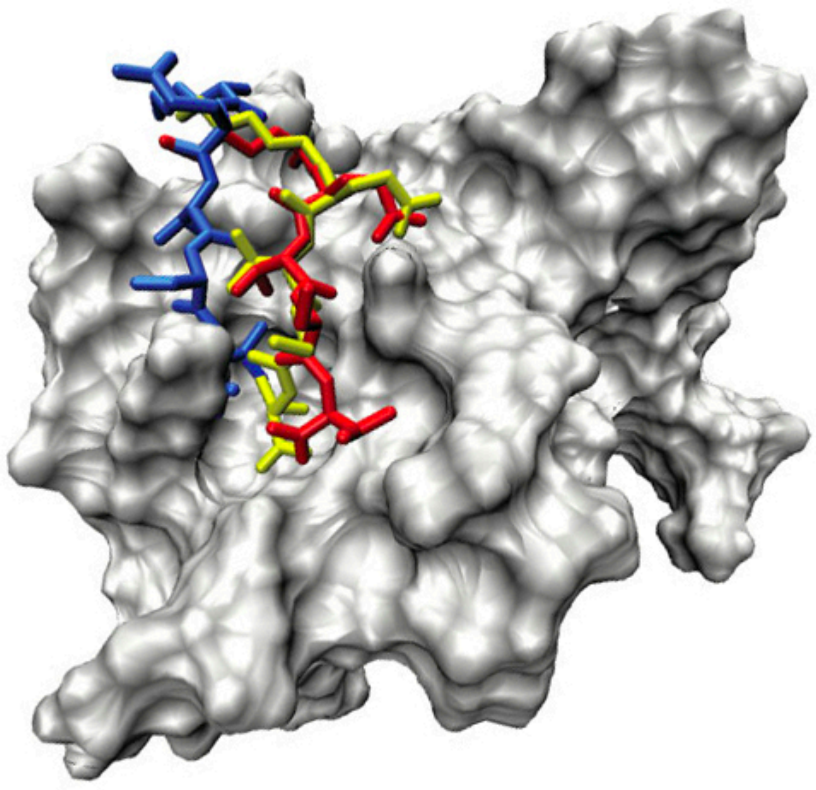


- COMPUTER VISION
  - Surface Reconstruction
    - Step 1: Scan the object (3d laser scanner)
    - Step 2: Create a triangulation
    - Step 3: process the triangulation (rendering *smooth* surface in $\boldsymbol{R}^3$
    - Major Computational Geometry task: Create a "good" triangulation
  - Pattern recognition
    - Example: an optical character recognition system



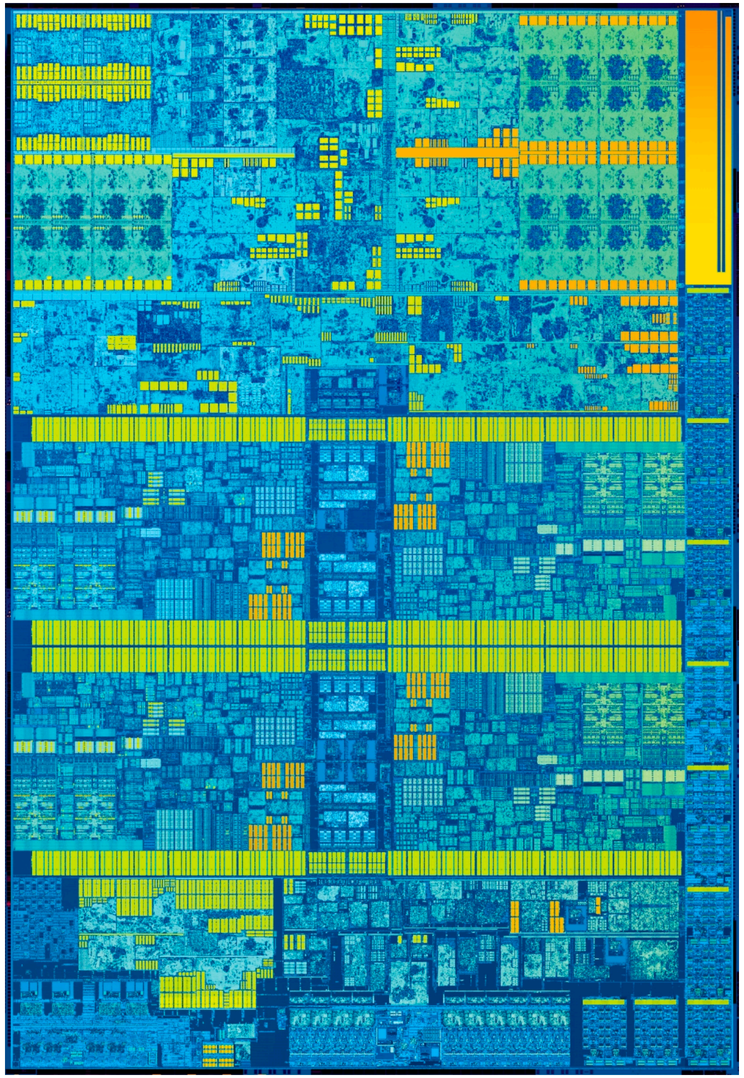
Applications of Computational Geometry

- Molecular modeling
  - Typical questions are to compute the union of the atom balls to obtain the molecule surface or to compute where two molecules can touch each other



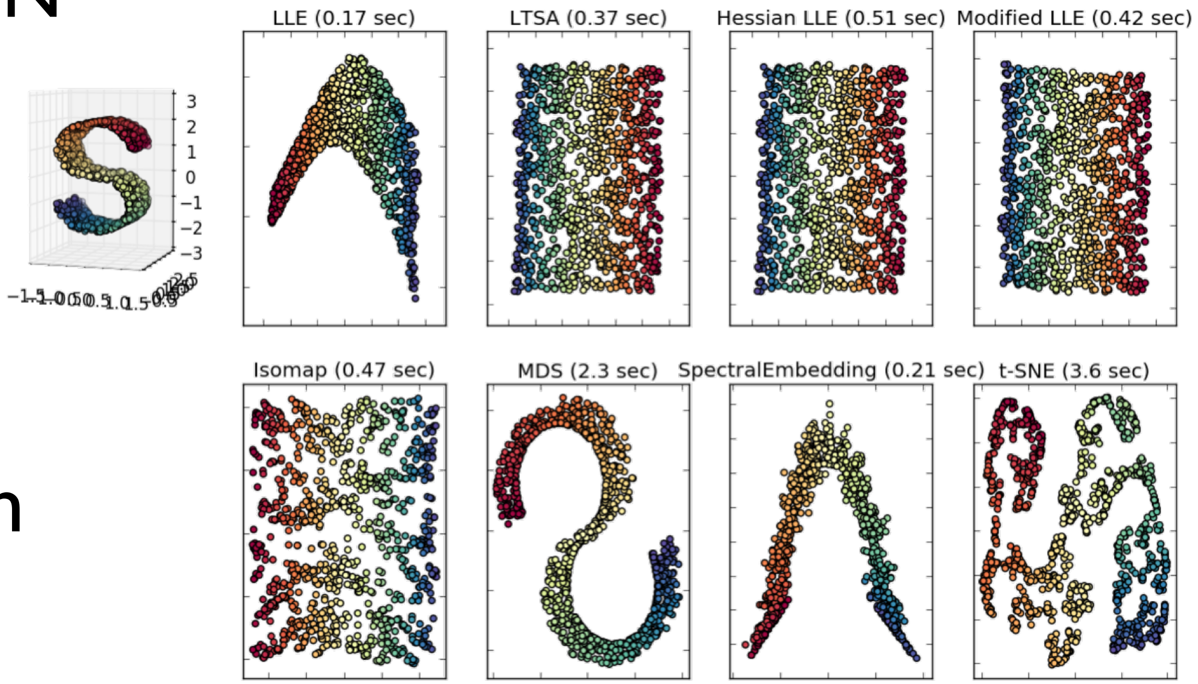
Applications of Computational Geometry

- VLSI design
  - The process of creating an integrated circuit (IC) by combining thousands of transistors into a single chip



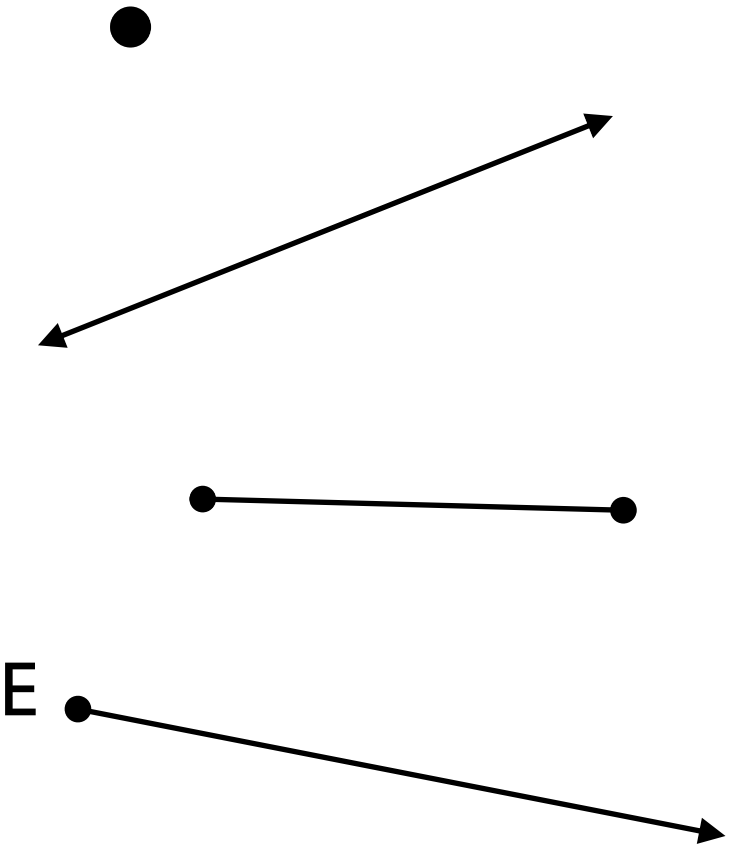
APPLICATIONS OF COMPUTATIONAL GEOMETRY

- DATA MINING, MACHINE LEARNING, AND VISUALIZATION
  - Many algorithms for operations such as clustering, dimension reduction, classification, and segmentation are based upon geometric approaches.

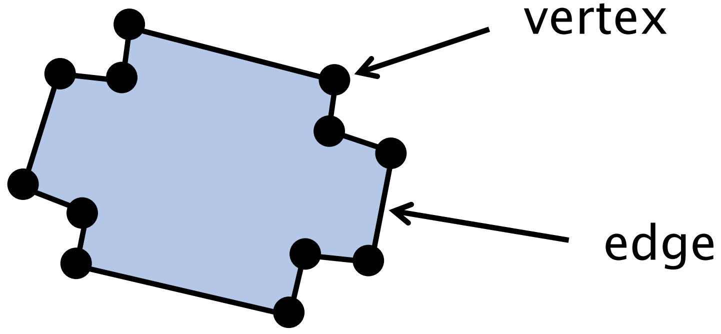


Basic definitions

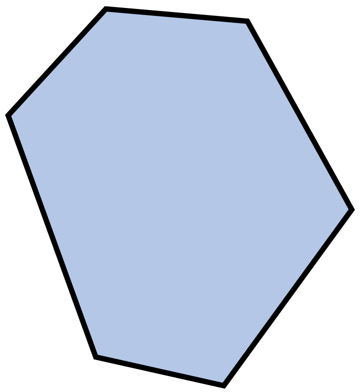
- Point—Specified by two coordinates (x,y)
- Line—Extends to infinity in both directions
- Line segment—Specified by two endpoints
- Ray—Extends to infinity in one direction



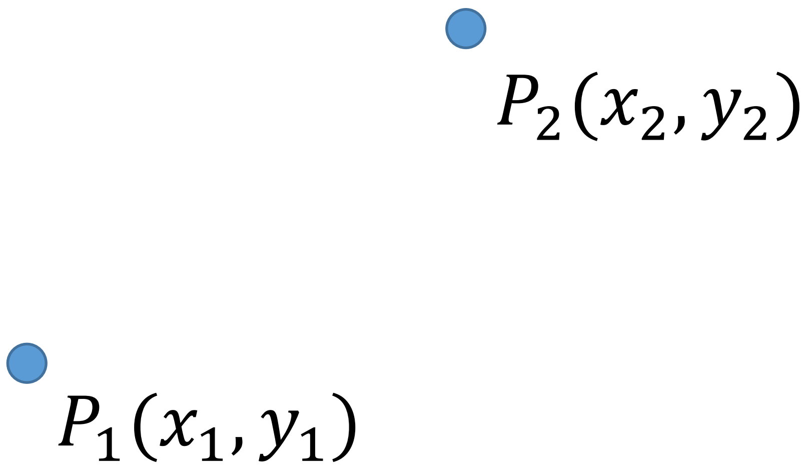
BASIC DEFINITIONS



- POLYGON
  - We assume edges do not cross
- CONVEX POLYGON
  - Every interior angle is at most 180 degrees
  - Precise definition of *convex*: For any two points inside the polygon, the line segment joining them lies entirely inside the polygon (we'll cover this later)

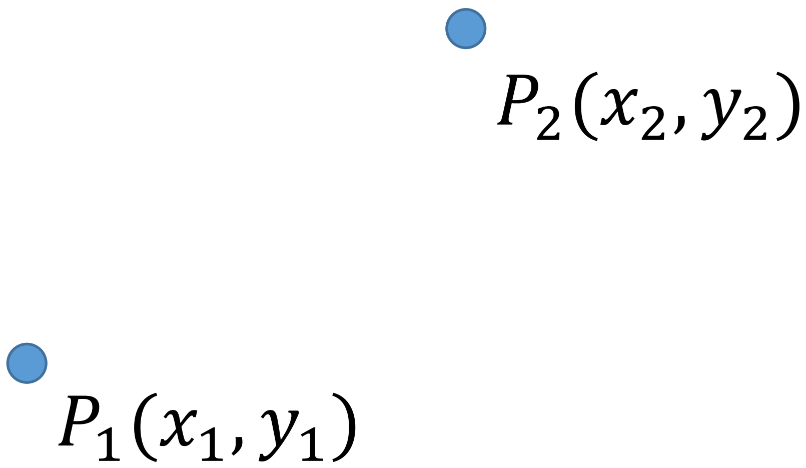


Points



Euclidean Distance

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



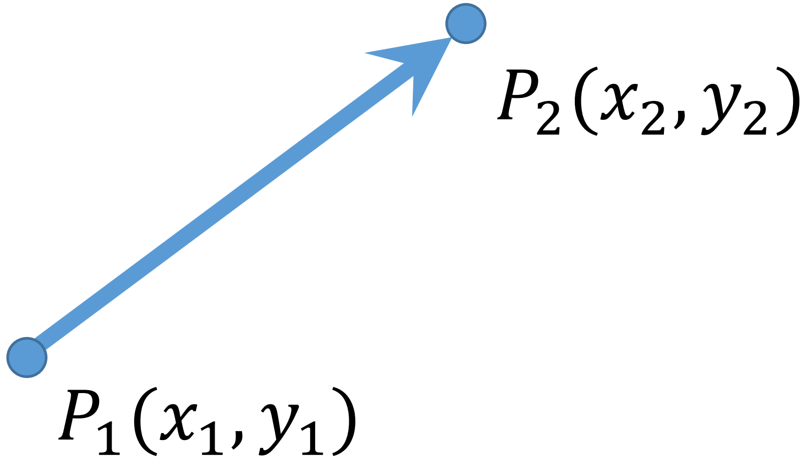
Euclidean Distance

- Square root is kind of slow and imprecise
- If we only need to check whether the distance is less than some certain length, say R

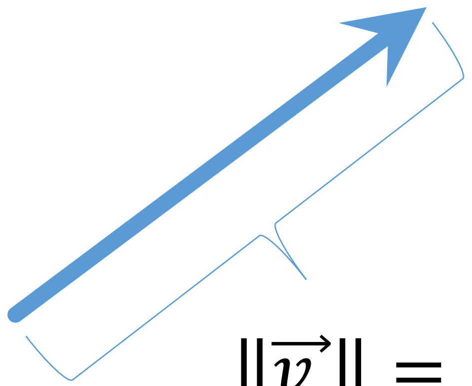
$$\text{if } \left((x_2 - x_1)^2 + (y_2 - y_1)^2 < R^2\right) \ldots$$

VECTORS

$$\overrightarrow{v_{12}} = P_2 - P_1$$
$$= < x_2 - x_1, y_2 - y_1 >$$

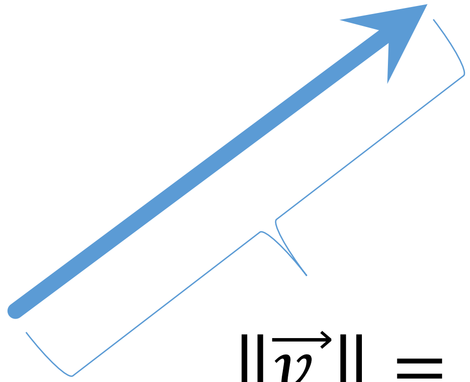


Vector Length/Magnitude



$$\|\vec{v}\| = \sqrt{v_x^2 + v_y^2}$$

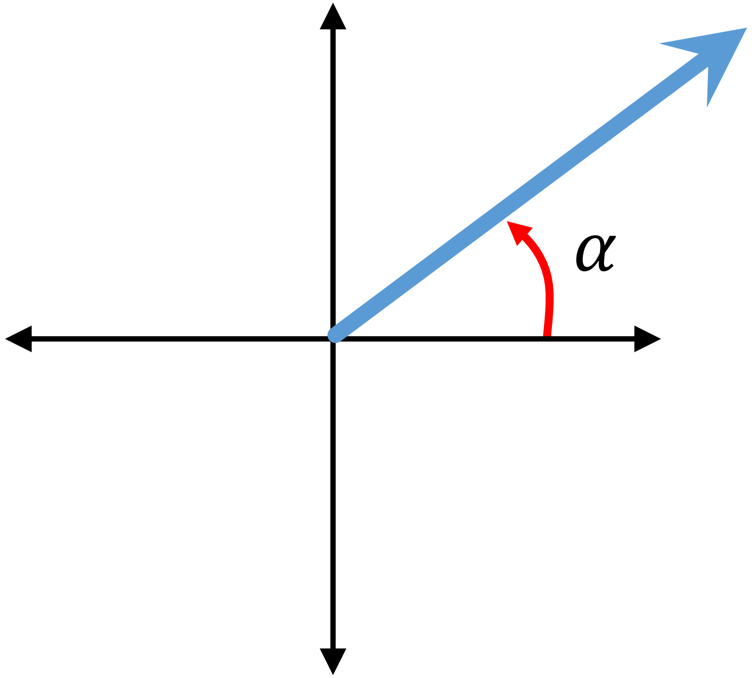
VECTOR LENGTH/MAGNITUDE



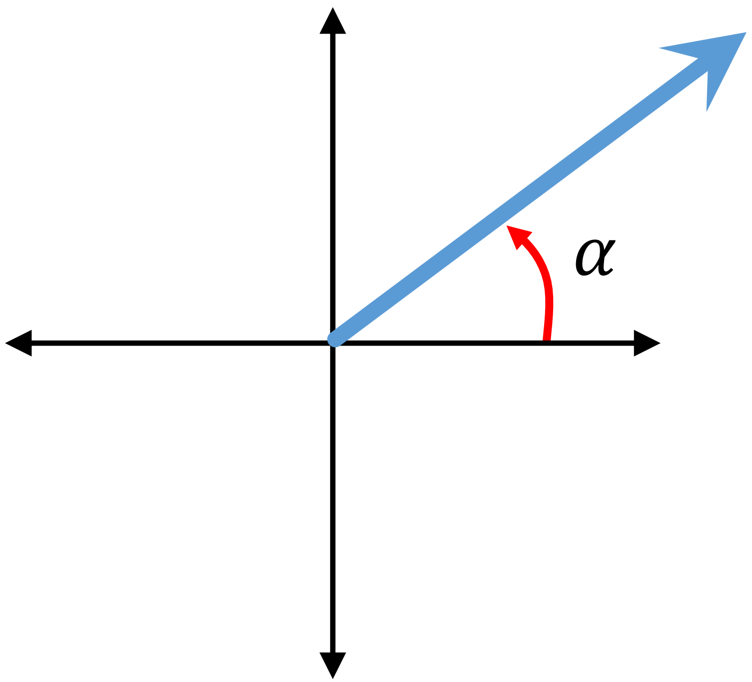
$$\|\vec{v}\| = \sqrt{v_x^2 + v_y^2}$$

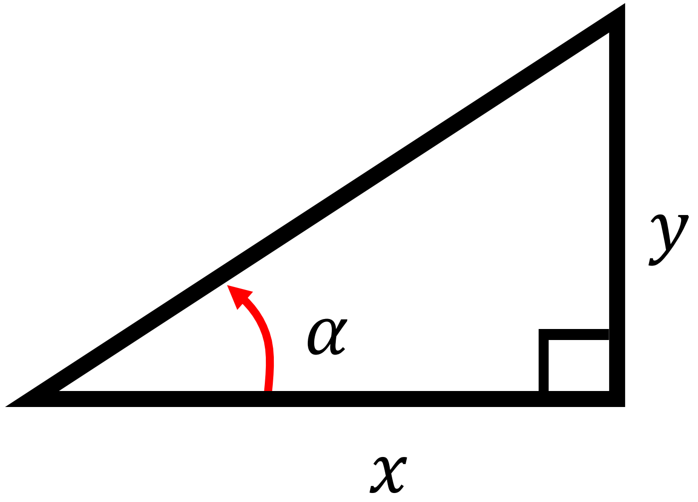
Equivalent to Euclidean distance…

Vector Angle



Vector Angle





$$\tan(\alpha) = \frac{y}{x}$$

Vector Angle

$$\alpha = \tan^{-1} \frac{y}{x}$$

or

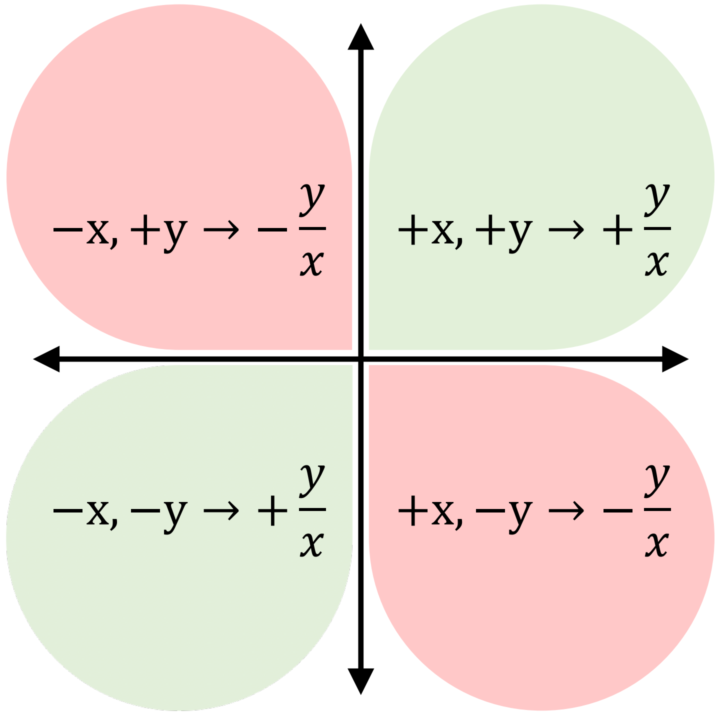
$$\alpha = \operatorname{atan}\left(\frac{y}{x}\right)$$

(results in radians, not degrees)

Problems?

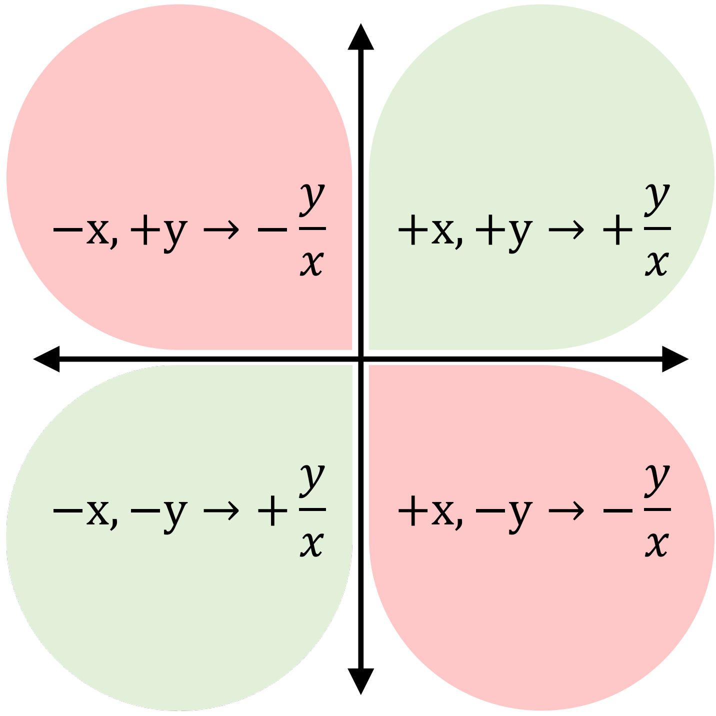
Vector Angle

- Problem 1: Division by zero
  - When α is $\frac{\pi}{2}$ or $-\frac{\pi}{2}$
- Problem 2: $\frac{y}{x}$ doesn't give a 1-to-1 mapping

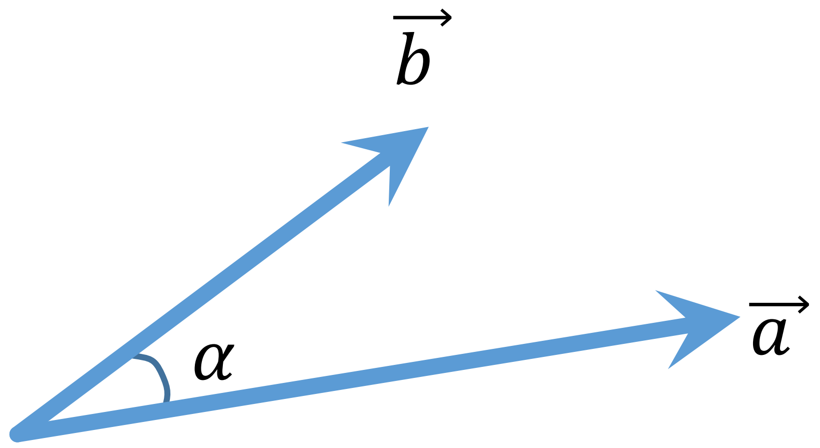


Vector Angle

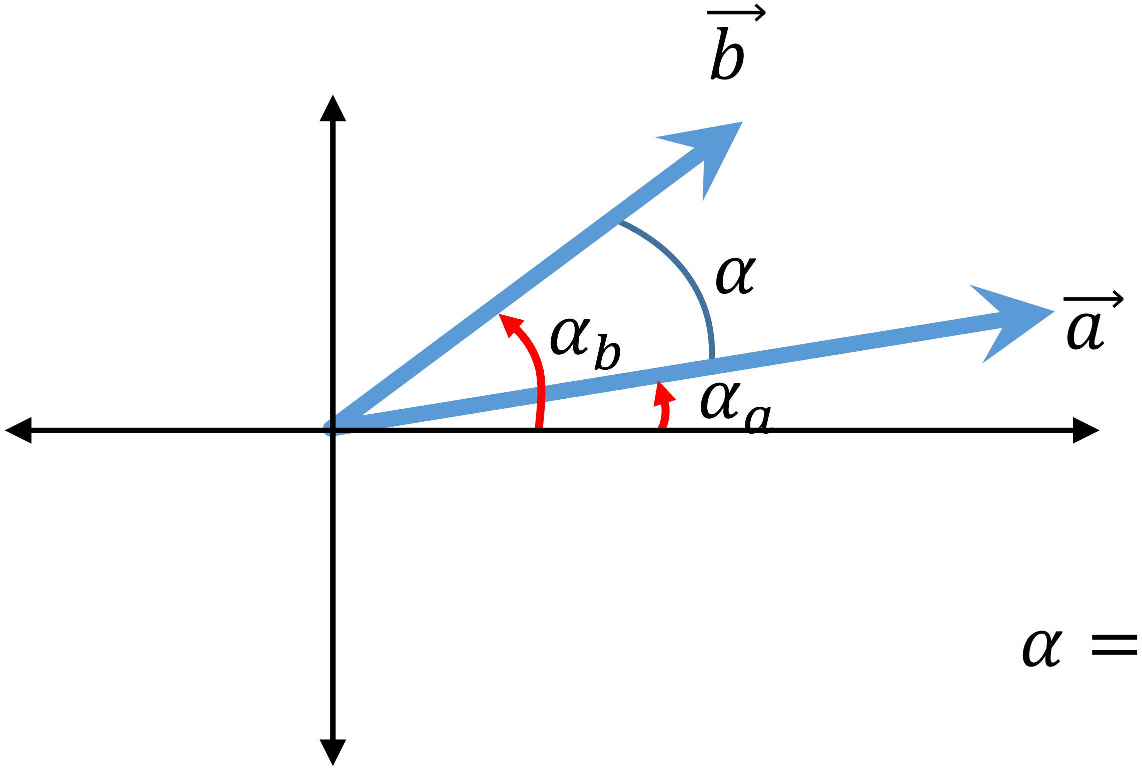
- Problem 1: Division by zero
  - When α is $\frac{\pi}{2}$ or $-\frac{\pi}{2}$
- Problem 2: $\frac{y}{x}$ doesn't give a 1-to-1 mapping
- Solution: $\boldsymbol{\alpha = \mathrm{atan}\,2(y, x)}$
  - Note: the arguments are $(\mathbf{y}, \boldsymbol{x})$, not (x, y)!!!



Angle Between 2 Vectors

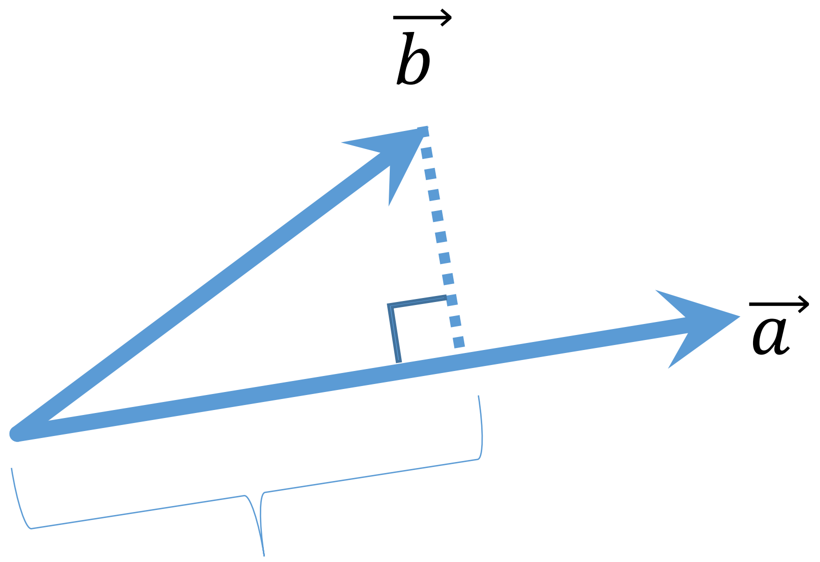


Angle Between 2 Vectors



$$\alpha = \alpha_b - \alpha_a$$

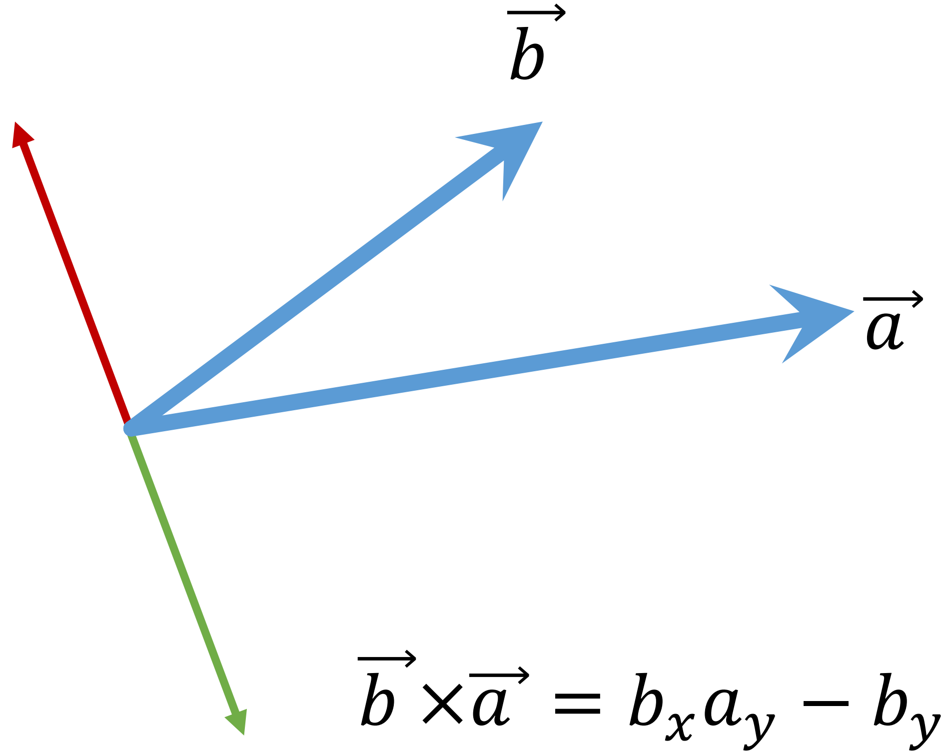
Dot Product



$$\vec{a} \cdot \vec{b} = a_x b_x + a_y b_y$$

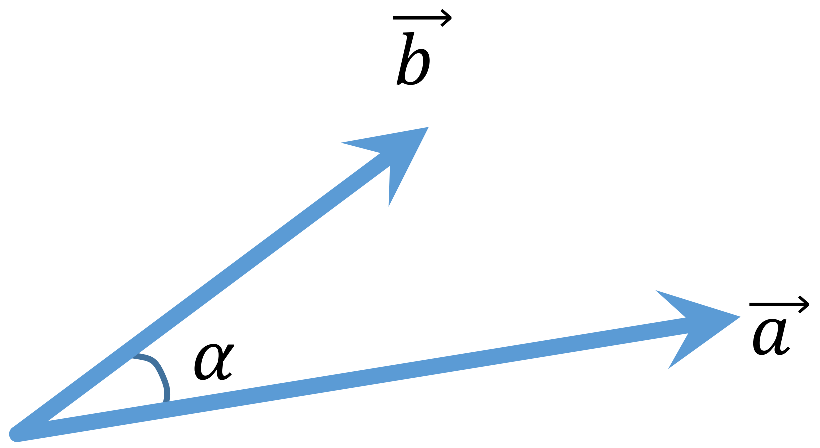
Cross Product

$$\vec{a} \times \vec{b} = a_x b_y - a_y b_x$$



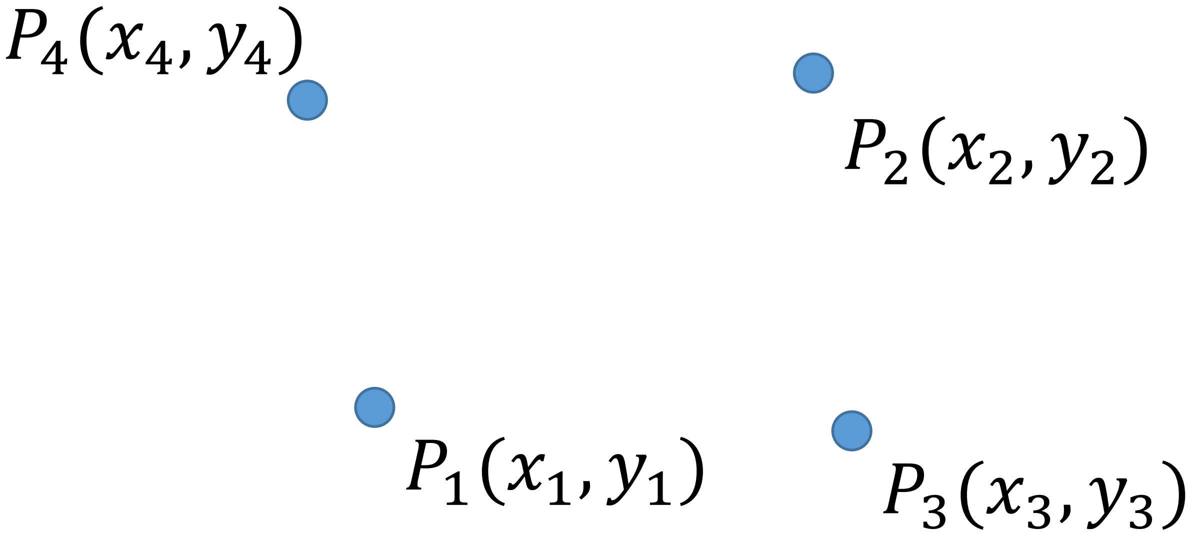
$$\vec{b} \times \vec{a} = b_x a_y - b_y a_x$$

Angle Between 2 Vectors



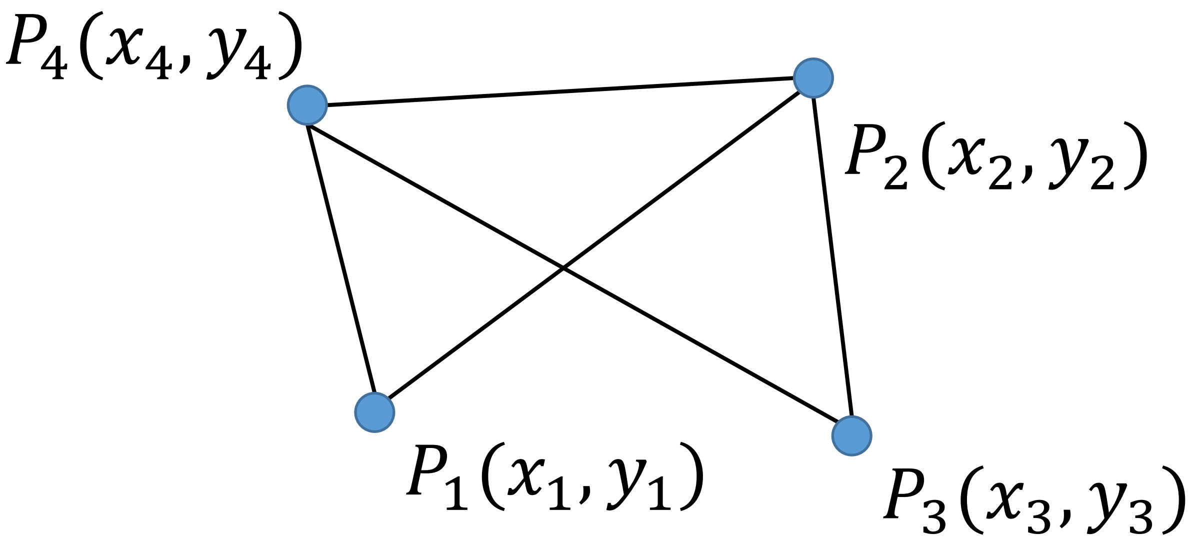
$$\cos\alpha = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\|\vec{b}\|} \qquad \sin\alpha = \frac{\|\vec{a}\times\vec{b}\|}{\|\vec{a}\|\|\vec{b}\|}$$

Storing Points



Point List
(x_1, y_1)
(x_2, y_2)
(x_3, y_3)
(x_4, y_4)
…
(x_n, y_n)

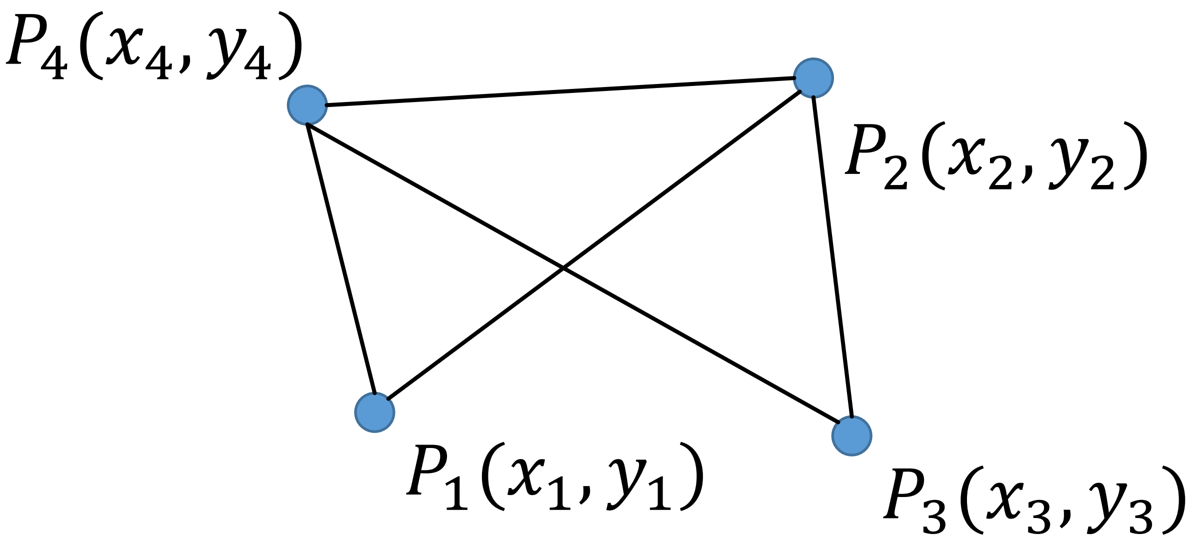
Storing Edges as Indices

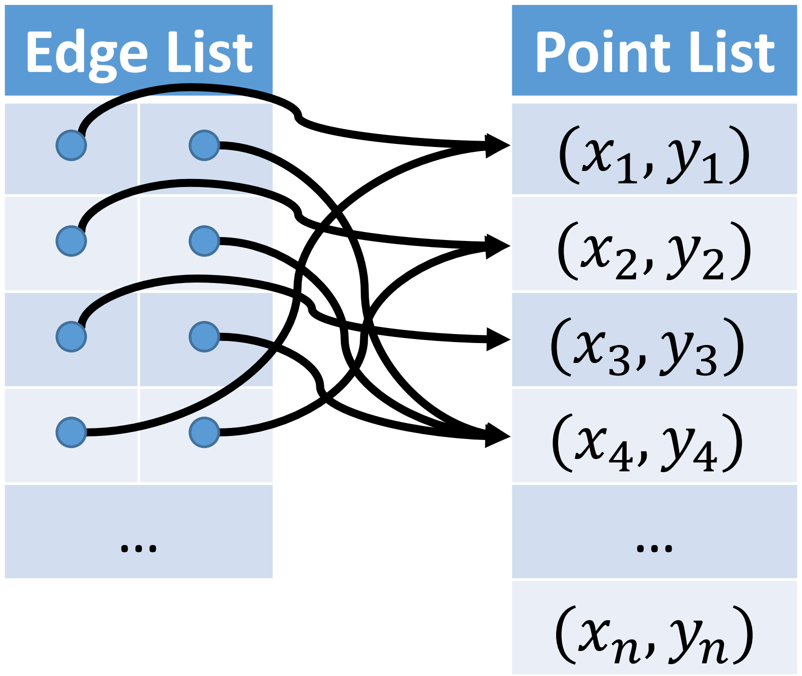


Edge List	
0	3
1	3
2	3
0	1
…	

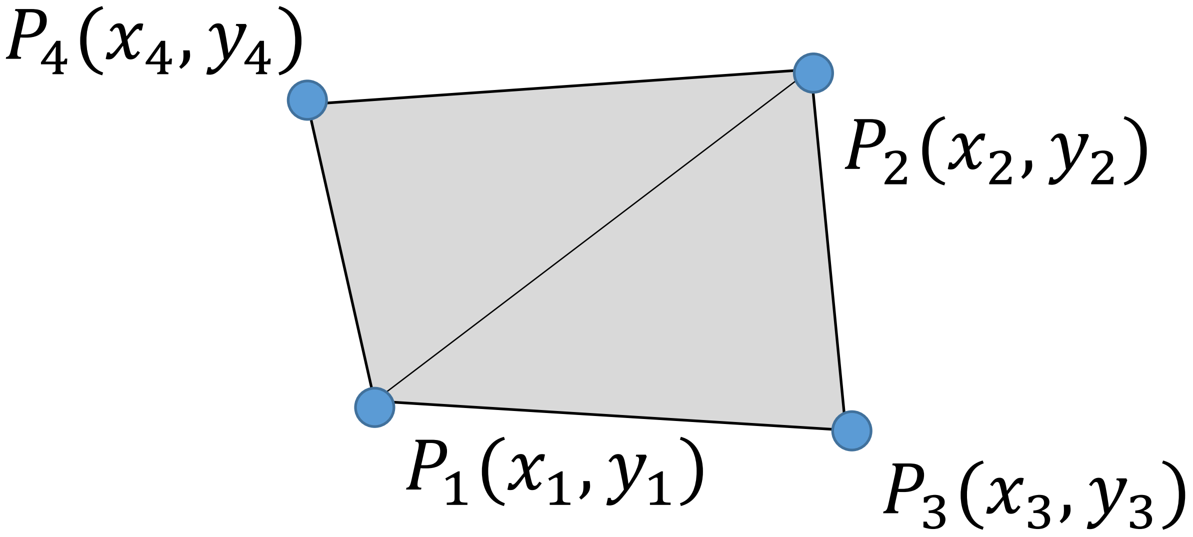
Point List	
0	(x_1, y_1)
1	(x_2, y_2)
2	(x_3, y_3)
3	(x_4, y_4)
…	…
n-1	(x_n, y_n)

Storing Points as Pointers





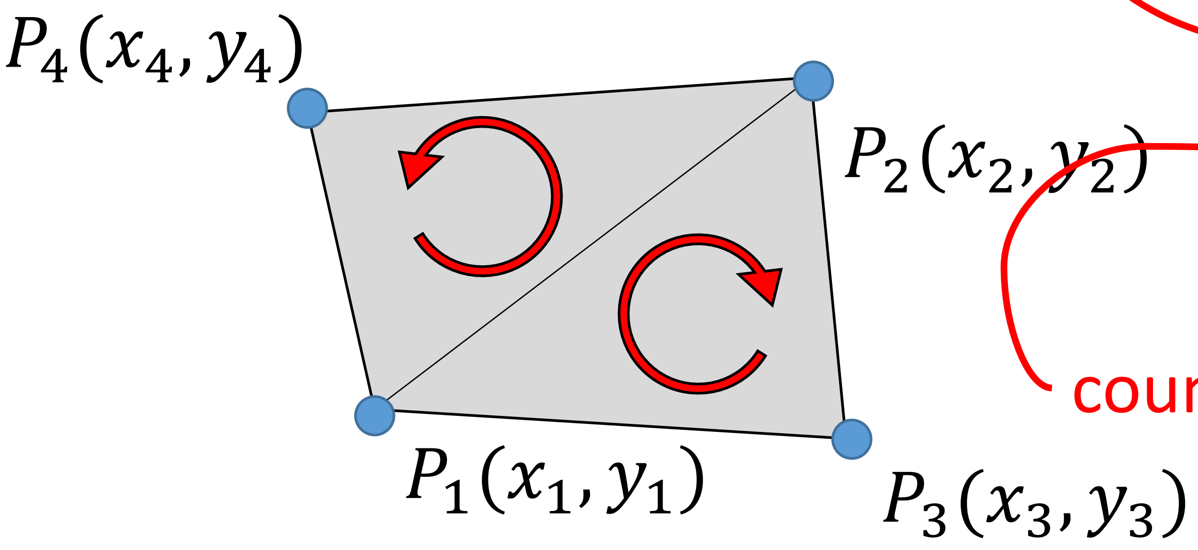
Storing Triangles with Indices



Triangle List		
0	1	2
0	1	3
…		

Point List	
0	(x_1, y_1)
1	(x_2, y_2)
2	(x_3, y_3)
3	(x_4, y_4)
…	…
n-1	(x_n, y_n)

STORING TRIANGLES WITH INDICES



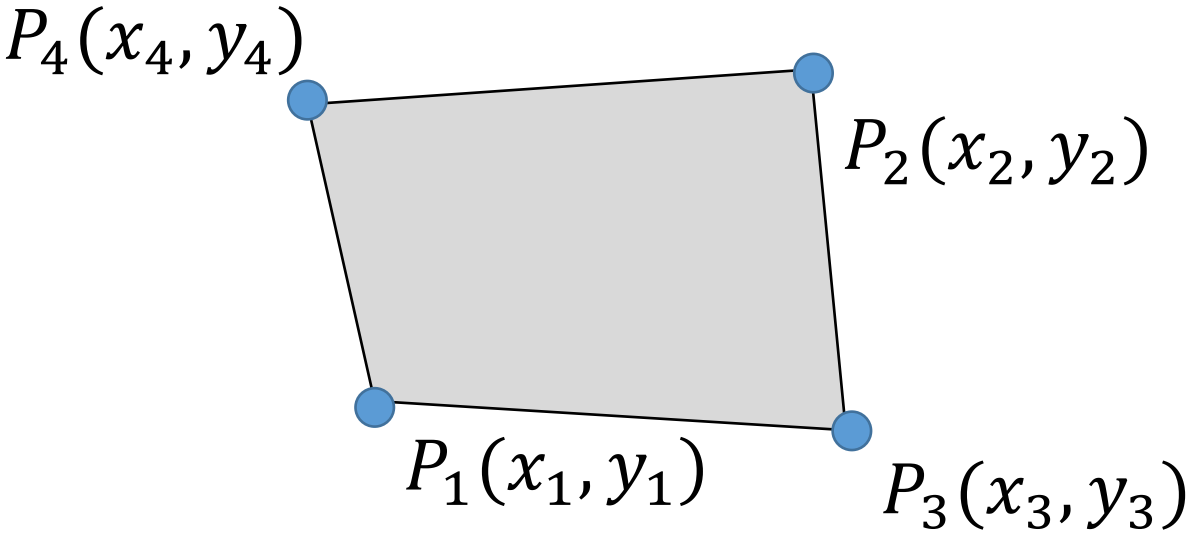
clockwise

counterclockwise

Triangle List		
0	1	2
0	1	3
…		

Point List	
0	(x_1, y_1)
1	(x_2, y_2)
2	(x_3, y_3)
3	(x_4, y_4)
…	…
n-1	(x_n, y_n)

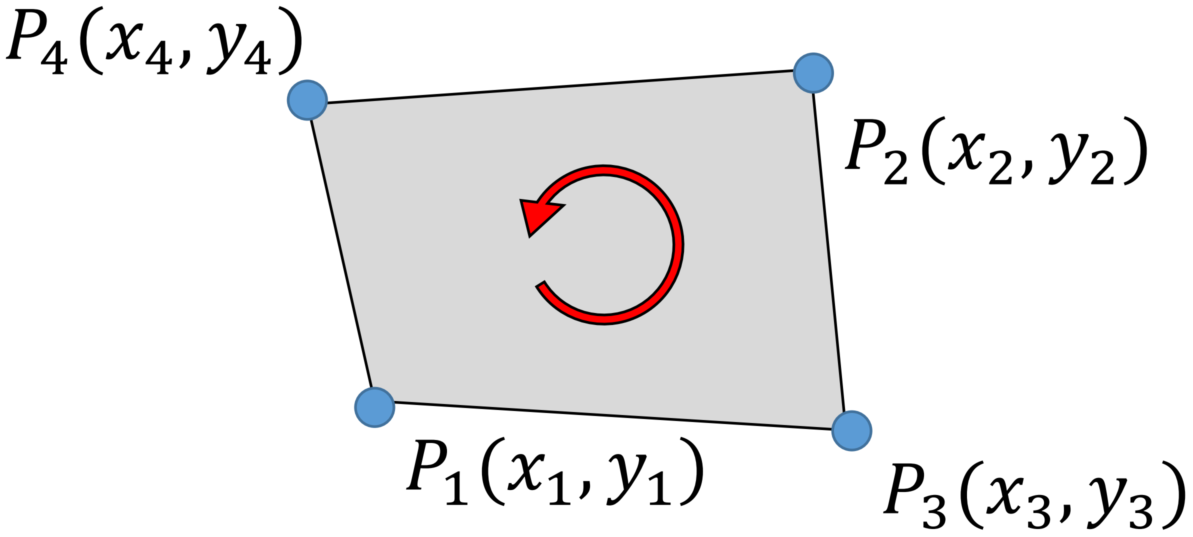
STORING POLYGONS WITH INDICES



Polygon List			
0	2	1	3
…			

Point List	
0	(x_1, y_1)
1	(x_2, y_2)
2	(x_3, y_3)
3	(x_4, y_4)
…	…
n-1	(x_n, y_n)

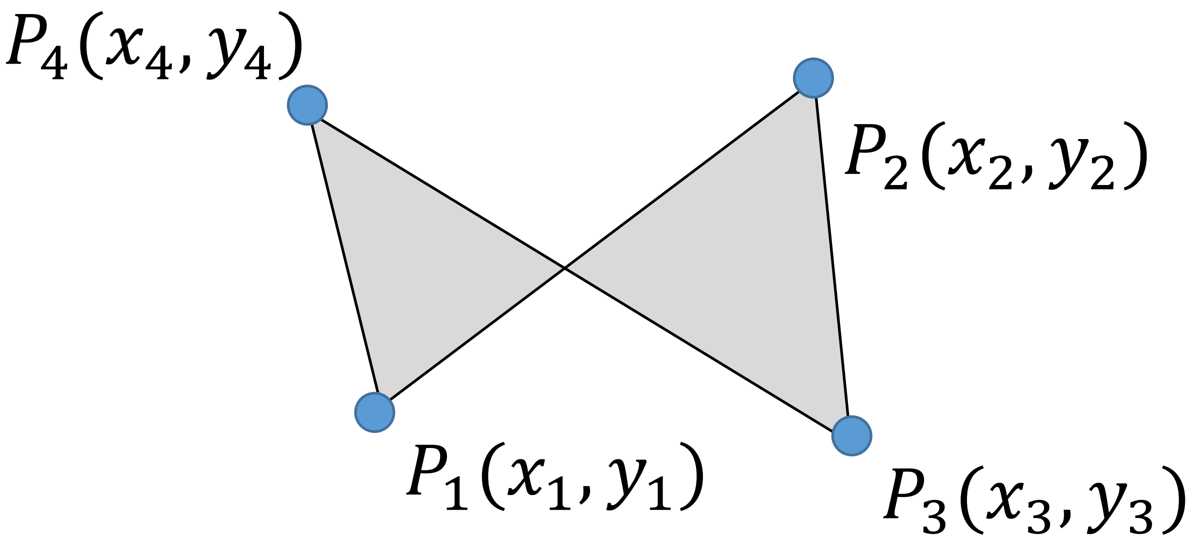
Storing Polygons with Indices



Polygon List			
0	2	1	3
…			

Point List	
0	(x_1, y_1)
1	(x_2, y_2)
2	(x_3, y_3)
3	(x_4, y_4)
…	…
n-1	(x_n, y_n)

Storing Polygons with Indices

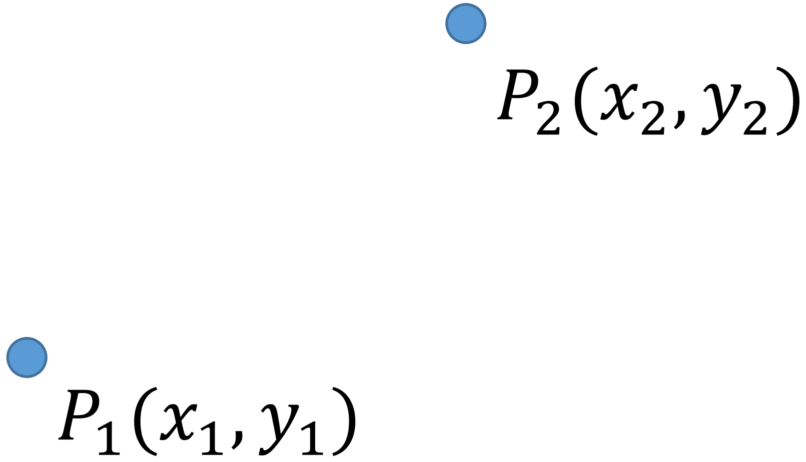


Polygon List			
0	1	2	3
…			

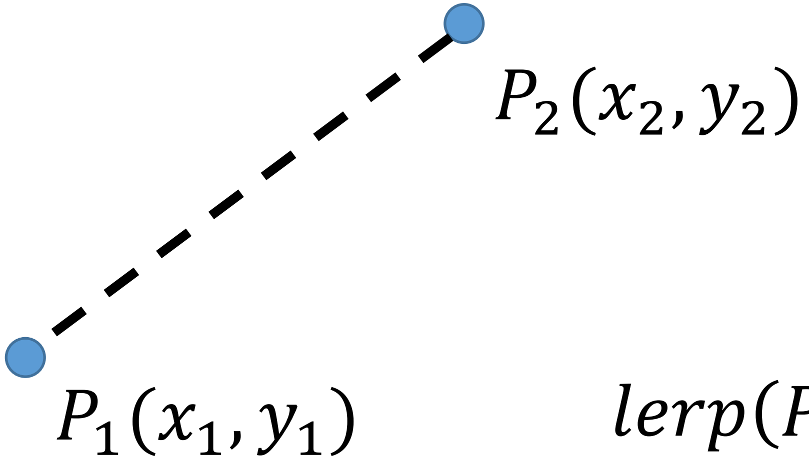
Take care with order

Point List	
0	(x_1, y_1)
1	(x_2, y_2)
2	(x_3, y_3)
3	(x_4, y_4)
…	…
n-1	(x_n, y_n)

Linear Interpolation

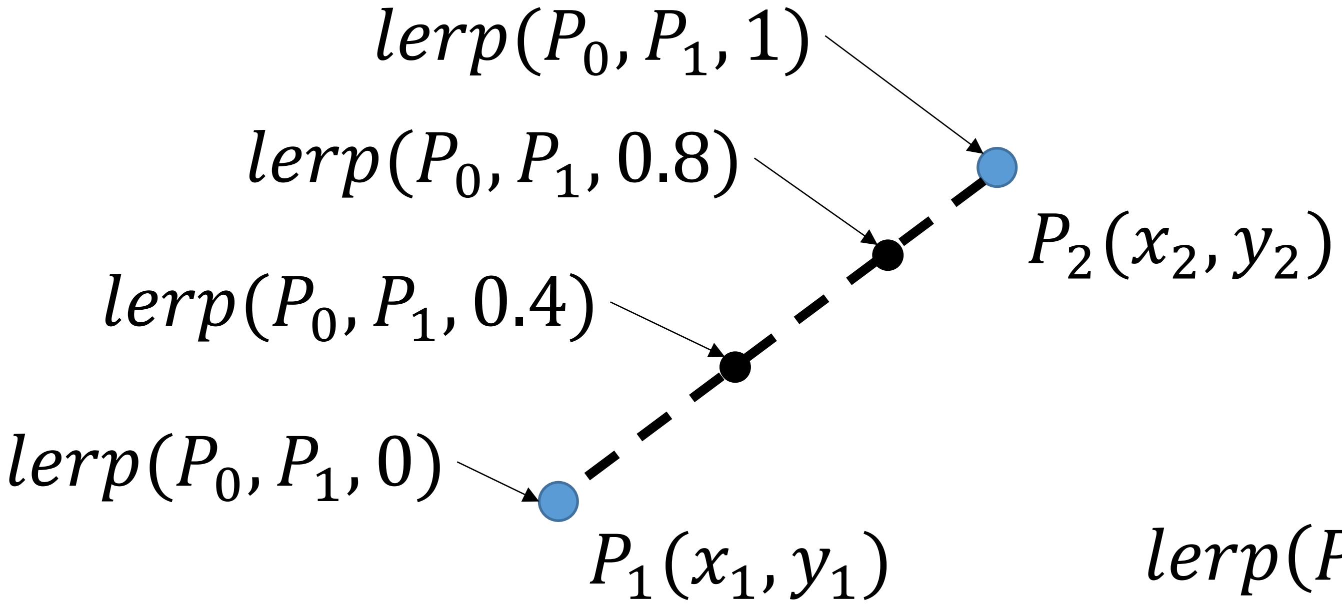


Linear Interpolation



$$lerp(P_1, P_2, a) = P_1(1 - a) + P_2 a$$

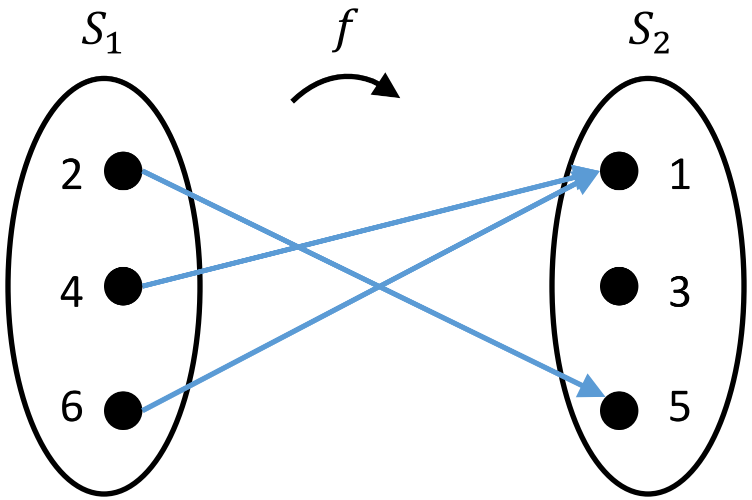
Linear Interpolation



$$lerp(P_1, P_2, a) = P_1(1 - a) + P_2 a$$

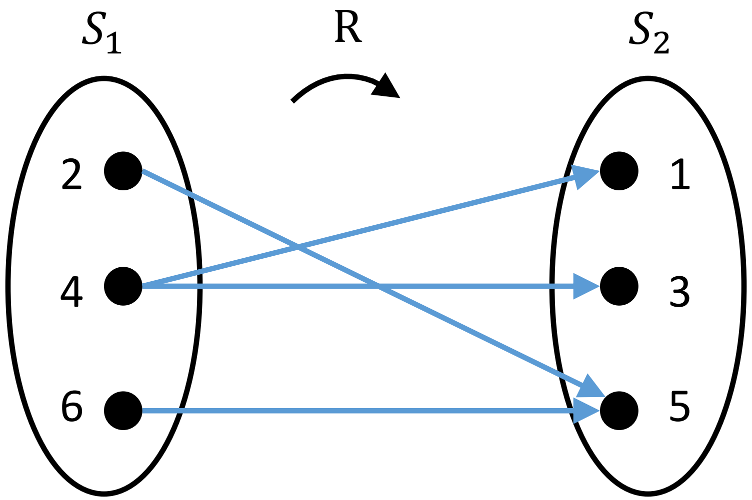
FUNCTIONS

- A FUNCTION IS A RULE THAT ASSIGNS TO ELEMENTS OF ONE SET A UNIQUE ELEMENT OF ANOTHER SET
  - $f: S_1 \rightarrow S_2,$
  - Where the domain of f is a subset of S_1 and the range of f is the subset of S_2
  - f is a total function on S_1 if the domain of f is all of S_1; otherwise f is a partial function



RELATIONS

- Some functions can be represented by a set of pairs $\{(x_1, y_1), (x_2, y_2), \ldots\}$, where x_i is an element in the domain of the function, and y_i is the corresponding value in its range
- For such a set to define a function, each x_i can occur at most once as the first element of a pair.
- If this is not satisfied, the set is called a relation.



FUNCTIONS

- THE BEHAVIOR OF FUNCTIONS:
  - Big O
    - f has order at most g
    - $f(n) \leq c|g(n)| \rightarrow f(n) = O\big(g(n)\big)$
  - Big Omega
    - f has order at least g
    - $|f(n)| \geq c|g(n)| \rightarrow f(n) = \Omega(g(n))$
  - Big Theta
    - f has the same order of magnitude as g
    - $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)| \rightarrow f(n) = \Theta\big(g(n)\big)$

Function Examples

- $f(n) = 2n^2 + 4n$,
- $g(n) = n^3$,
- $h(n) = 9n^2 + 300$

Function Examples

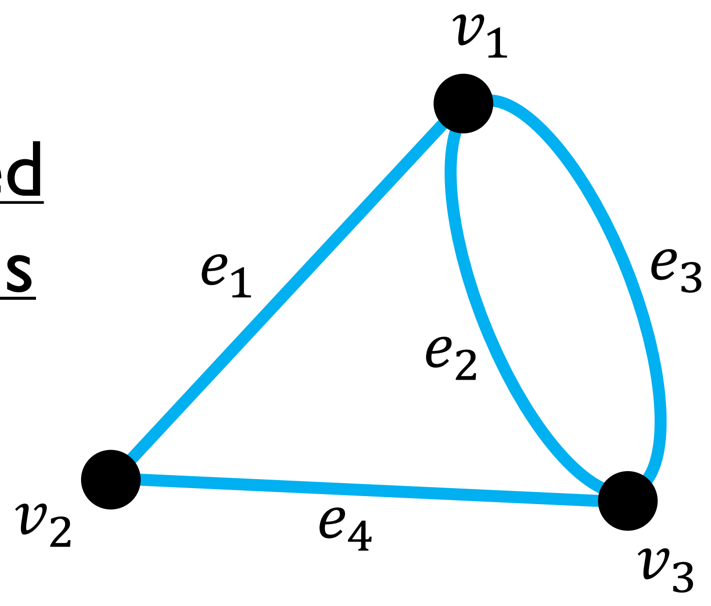
- $f(n) = 2n^2 + 4n$,
- $g(n) = n^3$,
- $h(n) = 9n^2 + 300$

- $f(n) = O(g(n))$,
- $g(n) = \Omega(h(n))$,
- $f(n) = \Theta\big(h(n)\big)$

$$O(n) + O(n) = 2(O(n))?$$

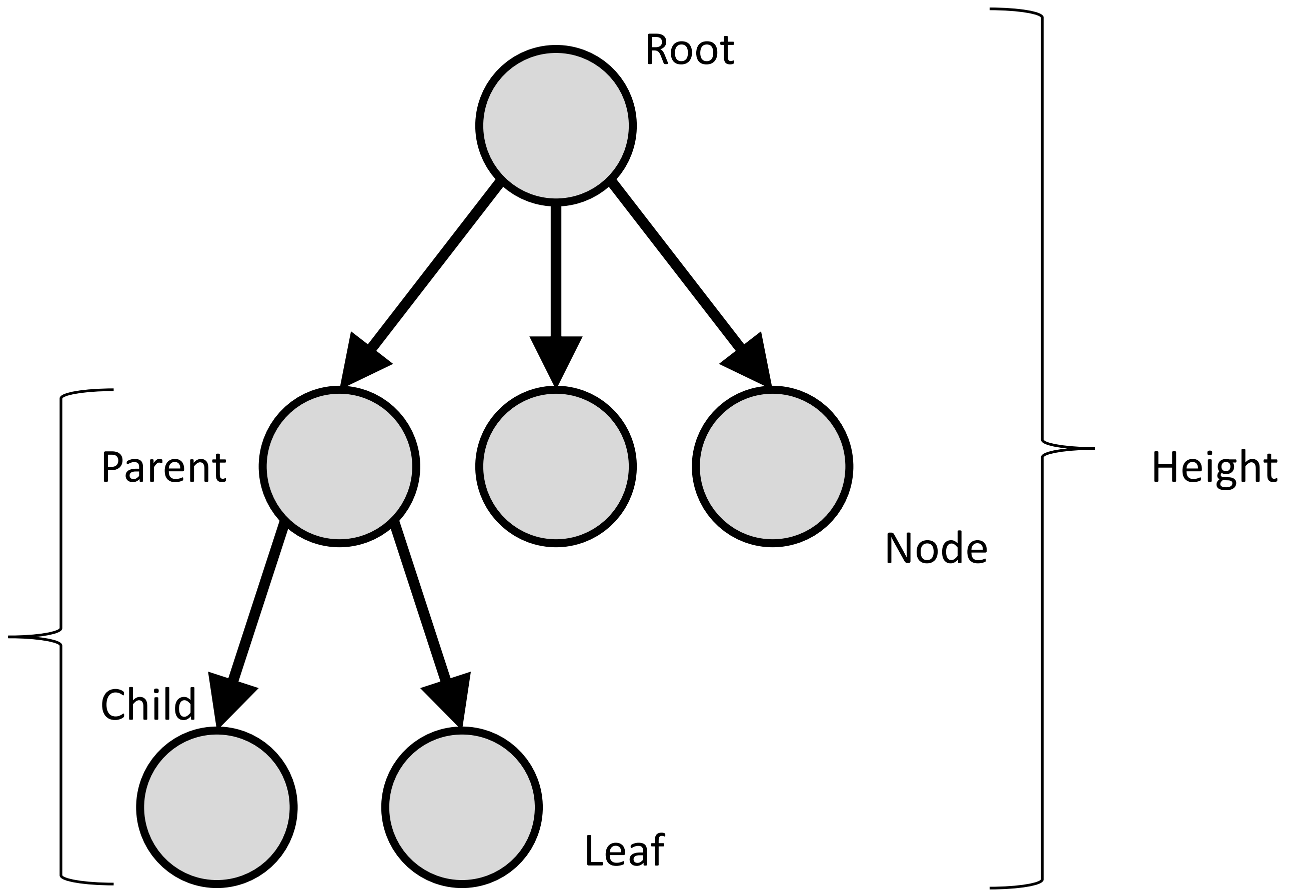
Graphs and Trees

- A graph is construct of two finite sets
  - Vertices, $V = \{v_1, v_2, \ldots, v_n\}$; Edges, E= $\{e_1, e_2, \ldots, e_m\}$
- A ***walk*** from v_i to v_n is a sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$
- A ***path*** is a walk in which <u>no edge is repeated</u>
- A ***simple path*** is a path in which <u>no vertex is repeated</u>
- A ***cycle*** with base v_i is a walk from v_i to itself with <u>no repeated edges</u>
- A ***loop*** is an edge from a <u>vertex to itself</u>



TREES

- A tree is a directed graph that has no cycles, and that has one distinct vertex (the root)



Proof techniques

- A proof is a sequence of steps that lead from some known facts to the desired conclusion; each step must be obviously correct
- Proof by contradiction:
  - To prove some fact P, we show that "not P" is false
  - That is, we suppose "not P" and demonstrate that it leads to an obviously wrong result
  - E.g.: Prove that $\sqrt{2}$ is not rational. Suppose that is rational, that is $\sqrt{2} = \frac{m}{n}$, where n and m do not have common factors

Proof techniques

- Proof by induction
  - We show that some fact is true for every natural number n, using two arguments:
    - Base: It is true for $n = 1$ (or for some small number)
    - Step: If it is true for n, then it is true for $n + 1$
  - E.g.: prove that $1 + 2 + \cdots + n = n(n+1)/2$