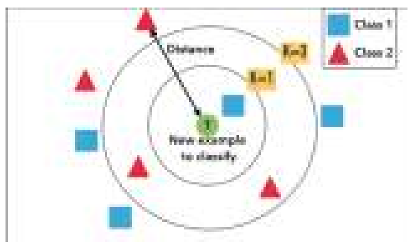
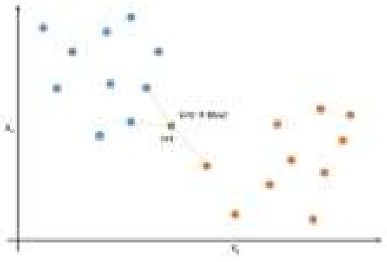
Computational Geometry

Mario A. Lopez and *Jen Robertson*
Department of Computer Science
University of Denver

A Proximity Problem

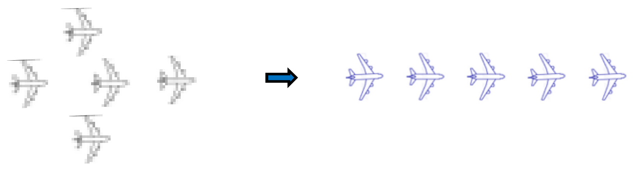
- Given a set S of points in $\mathbb{R}^d$ (such as feature vectors from people's faces), an integer k, and a query point q, find the k points from S closest to q
- Fundamental technique for data classification in machine learning (majority wins)
- Can you solve this in $o(n)$ time?





An Minimization Problem

- How do you move *n* drones from their current to a new set of locations as quickly as possible while avoiding collisions?

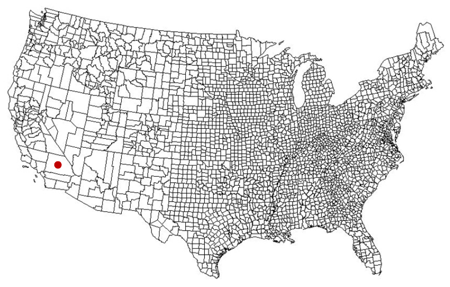


- Can you model this as a graph problem? Explain.
- Variants.
  - All drones are identical
  - Each drone has a specific position in the target formation

3

A Location Problem

- Given your longitude and latitude, which county are you in?

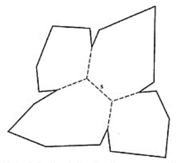
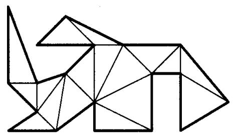


- What data structure(s) would you use to store the planar subdivision (the map)?

4

A Decomposition Problem

- Given a polygon *P,* break it into a small number of convex polygons (e.g., triangles, trapezoids, etc.)
- Basic step of many applications: rendering in computer graphics, point location, etc.

- One type of decomposition uses *diagonals* (segments connecting two vertices)
  - If the decomposition is *maximal*, the components are triangles
  - How many diagonals/triangles are needed for a given polygon?
  - Is a triangulation with diagonals *always* possible?

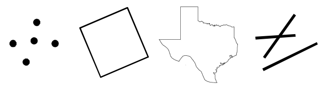
5

Introduction

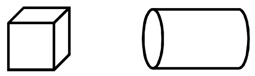
- *Computational geometry* is an area of CS devoted to the design and analysis of algorithms for problems dealing with *multidimensional data*

1D: social security number, phone number

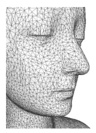
2D: points, polygons, line segments, polygonal partitions, etc.



3D: points, polyhedra, cylinders, spheres



- Emphasis is on:
  - Discrete problems
  - Computational complexity



6

Application Areas

- Computer-aided design:
  - VLSI design, solid modeling, manufacturing
- Computer graphics (movies, games, VR)
- Geographic information systems
- Robotics and computer vision
- Computational biology
- Machine learning
- Data mining in large databases
- World models (terrains, architecture, medical images)
- Music analysis and generation

7

Course Goals

- Describe the most common algorithmic design techniques for problems dealing with geometric objects
- Study efficient algorithms for classical geometric problems: convex hulls, intersection, decomposition, searching, proximity, interpolation, etc.
- Illustrate how the algorithms can be used in various application domains
- Learn how to write robust geometric code
- Practice reading and evaluating research literature on the subject

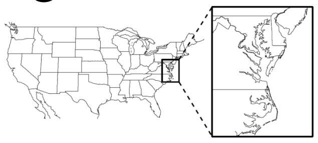
8

Expected Algorithmic Background

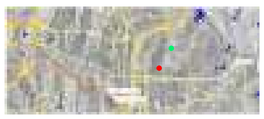
- Complexity using asymptotic notation: O, Ω, Θ, o
- Algorithms
  - Design techniques: incremental, DAC, greedy, augmentation
  - Sorting: reorder a list S with respect to an operator $<$
  - Searching: determine if $x \in S$
  - Order statistics: what is the i-th smallest value of S?
  - Lower bounds:
    - $\Omega(n \log n)$ comparisons are sometimes necessary to sort n values
    - $\Omega(\log n)$ comparisons are sometimes necessary to search for $x \in S$
- Data Structures: Stacks, queues, lists, heaps, hash tables, balanced search trees, graphs, and complexity of operations

Problem Categories

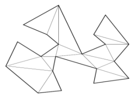
- Intersection
  Example: zoom into a graphical scene



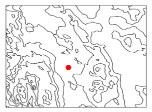
- Point location
  Example: find name of street at your current location
- Proximity:
  Example: find nearest gas station



- Decomposition
  Example: partition a polygon into triangles



- Interpolation
  Example: estimate elevation of a geographic location

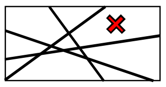


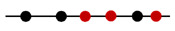
- Optimization
  Example: find *shortest* route from A to B avoiding obstacles

1

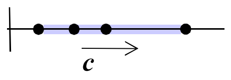
One Dimension vs. *d* Dimensions

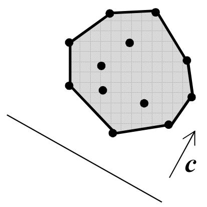
- Multidimensional problems are often generalizations of one dimensional problems:





sorting	planar subdivisions
merging	subdivision overlay
searching	point location
order statistics	convex hulls, linear programming





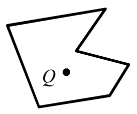
11

Problem Mode

Given a geometric data set *S* on which we want to perform queries *Q*

- *Single-shot*: we want to perform one or few queries
- *Repetitive-mode*: we want to perform many queries, using different values of *Q*
  - Preprocess *S* into a data structure *D*(*S*) to facilitate query
  - Static vs. dynamic

Example: is point *Q* inside polygon *S*?



12

Complexity

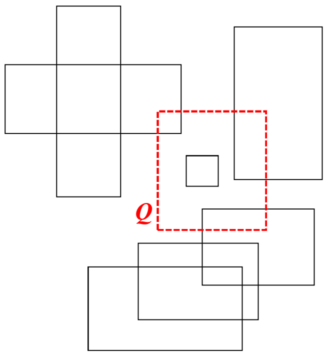
- Express running time $T(n)$ as a function of input size n
- When output size is variable, we may want to express the running time as a function $T(n,k)$ of both input size n and output size k for an *output-sensitive* algorithm
- For repetitive mode problems:
  - Query time $T(n)$ to perform query on $D(S)$
  - Preprocessing time $P(n)$ to construct $D(S)$
  - Update time $U(n)$ to change S
  - Memory $M(n)$ required to store $D(S)$

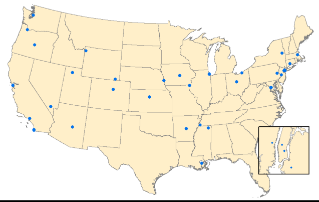
13

Intersection Problems

Given a set S of objects in d-dimensional space:

- *All-pairs:* report all pairs of objects from S that intersect
- *Intersection-searching:*
  - Store S in a data structure $D(S)$
  - Given a query object Q, use $D(S)$ to report the objects in S that intersect Q



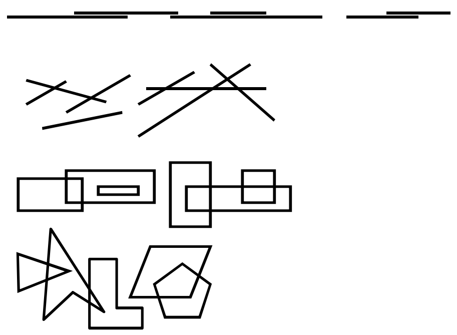


14

Intersection Variants

Different problems for different types of objects in S and different types of queries:

- Intervals
- Segments
- Rectangles
- Polygons

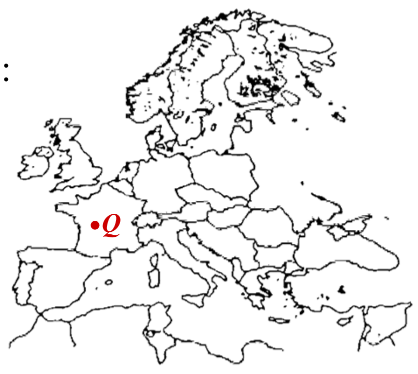


15

The Point Location Problem

Given a subdivision S of $\mathbb{R}^d$ (d-dimensional space):

- Store S in a data structure $D(S)$
- Given a query point Q, use $D(S)$ to find the region of S that contains Q

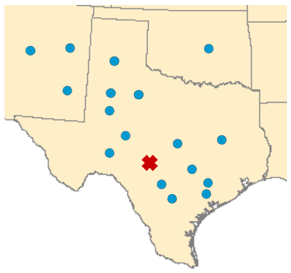


16

Proximity Problems

Given a set *S* of points in *d*-dimensional space:

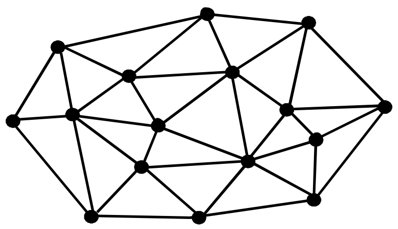
- *Closest-pair*: find two points of *S* that are closest.
- *Farthest-pair*: find two points of *S* that are farthest
- *Nearest-neighbor:*
  - Store *S* in a data structure *D*(*S*)
  - Given a query point *Q*, use *D*(*S*) to find the point in *S* that is closest to *Q*



17

The Triangulation Problem

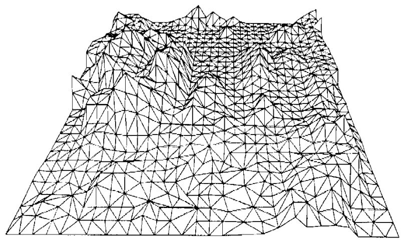
Given a finite set of points $S \subset R^2$, construct a *triangulation* of *S*, i.e., a maximal polygonal subdivision of the plane whose vertex set is *S*



18

An Application: Height Interpolation

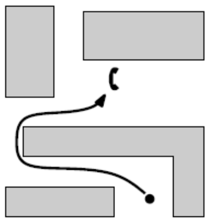
- Given a set of points $S \subset \mathbb{R}^2$ and a function $f : S \rightarrow \mathbb{R}$, construct a *polyhedral terrain* for *S*, i.e., a piecewise linear function that approximates the original terrain
- Triangulate *S* and elevate each point $p \in S$ to $f(p)$

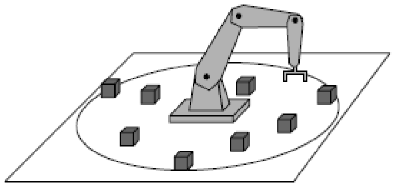


19

Geometric Optimization Problems

- Find the shortest route from *s* to *d* while avoiding a collection of polygonal obstacles
- Given a set of points in the plane, find the smallest enclosing disk





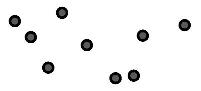
20

Algorithm Design

- Good solutions to geometric problem are based on two ingredients: a thorough understanding of the geometric properties of the problem, and a proper application of sound algorithm design techniques
- Design methodology:
  1. Understand the geometry of the *general case*, ignoring degenerate and boundary cases (input data is assumed to be in ***general position***)
  2. Design algorithm for the general case
  3. Extend the algorithm to handle degenerate cases
  4. Implement algorithm, including required primitives and predicates

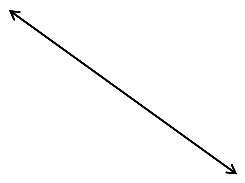
Geometric Primitives (2D)

- Point: two numbers (x, y)



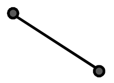
```
struct Point{ int x, y; };
Point A, B, RegistrarOffice;
```

- Line:
  - two numbers a and b: $y = ax + b$?
  - two points A and B: $p(t) = tA + (1-t)B$



```
typedef Point Line[2];  Line L;
```

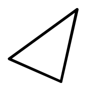
- Line segment: two points, $p(t) = tA + (1-t)B$, $0 \le t \le 1$



```
typedef Point Segment[2];  Segment Edge;
```

- Triangles

```
typedef Point Triangle[3];  Triangle T;
```



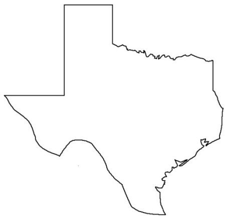
Geometric Primitives (2D)…

- Polygonal lines and polygons

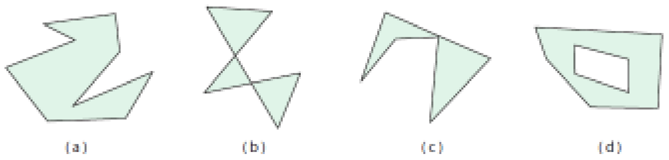
  ordered sequence of points $\langle p_1, p_2, \ldots, p_n \rangle$

  struct Polygon{ int n; Point *vertices; };

  Polygon FL, CO, TX, USA[50];



- Which of these are polygons?



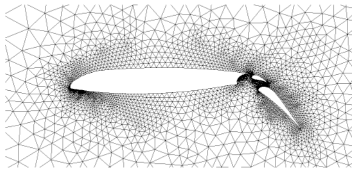
23

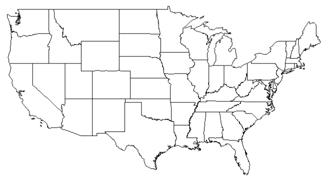
Geometric Primitives (2D)…

- Circle: a point and a number (the radius)

  struct Circle{ Point center; int radius; }; Circle mySensor;

- Other: ellipses, upright rectangles, polygonal meshes, etc.

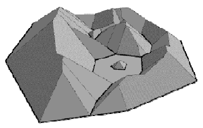


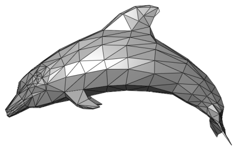


Geometric Primitives (3D)

- Point: three numbers (x, y, z)
- Line:
  - two points A and B: $p(t) = tA + (1 - t)B$
- Line segment: two points, $p(t) = tA + (1 - t)B$, $0 \le t \le 1$
- Polygonal lines, polygons

  ordered sequence of points $\langle p_1, p_2, \ldots, p_n \rangle$
- Sphere: a point and a number (the radius)
- Other: ellipsoids, polyhedra, polygonal meshes, etc.





25

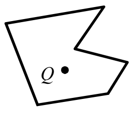
Primitive Operations and Predicates

- On which side of line does a point lie?
- Is a given point inside a given polygon?
- Which of two points is farther from a given point or a given line?
- Are three given points collinear?
- Which of two lines has the bigger slope?
- Do two line segments intersect?
- Do three points, in a given order, turn CCW?
- Is a given polygon simple? is it convex?
- Many more…

26

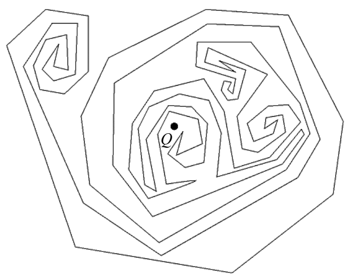
Problem 1: Point in Polygon

- A *polygon* is the closed region of $\mathbb{R}^2$ bounded by a finite set of line segments forming a closed curve that does not intersect itself.
- Is Q inside the polygon?



1	2	5	6	3	4
7	1	2	4	5	8

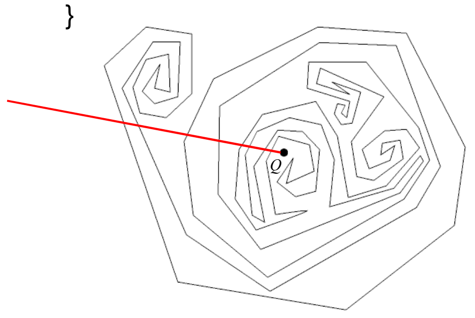
3
3



27

Solution1: Point in Polygon

```
bool PointInPolygon(int n, float ∗sx, float ∗sy, x, y) {
int i, j, in(false);
for (i = 0, j = n−1; i < n; j = i++)
     if ( ( (sy[i]<=y) && (y<sy[j]) || (sy[j]<=y) && (y<sy[i]) )  &&
                     ( x <  (sx[j]−sx[i])∗(y−sy[i])/(sy[j]−sy[i]) + sx[i] ) )
            in = !in;
      return in;
}
```



Jordan Curve Theorem. The boundary ∂P of a polygon P partitions $\mathbb{R}^2 \setminus \partial P$ into two disjoint sets: the bounded interior and the unbounded exterior.

28

Exercise

- The *Jordan Curve Theorem* implies that the boundary ∂P of a polygon P partitions $\mathbb{R}^2 \setminus \partial P$ into two parts: the odd-crossing set and the even-crossing set.
  1. Prove that if there is a path between the even- and the odd-crossing sets, then the path must cross ∂P
  2. Prove that if two points reside in the same set, then there is a path connecting them that does not cross ∂P.

29

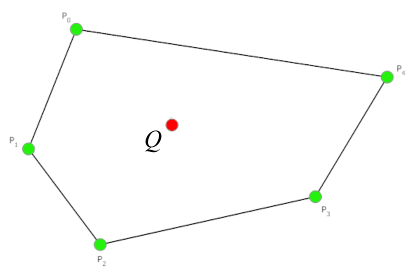
Exercise

- Let P be a convex polygon of size n. Describe an efficient algorithm to determine if a query point Q is inside or outside of Q. *Hint*: use binary search

$P =$

62	48	35	24	17	15	17	24	36	48	62	75	85	93	95	92	86
16	15	20	30	41	55	69	81	90	94	93	90	81	69	55	41	29

Is $Q = (16,50)$ inside P?

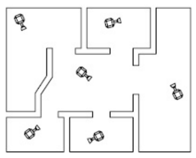


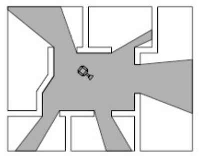
30

Problem 2: Guarding an Art Gallery

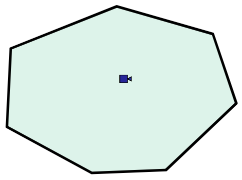
- How many stationary guards or cameras are needed to guard a gallery whose floor plan is modeled a simple polygon *P*?

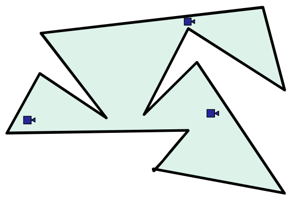
  Note: A camera at *p* can see a point *q* iff $\overline{pq} \subset P$





- The answer, of course, depends on the shape of *P*





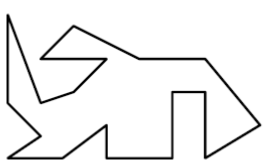
31

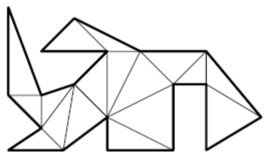
Guarding an Art Gallery…

- In 1973 Victor Klee posed the following problem:

  As a function of *n*, find the minimum number of guards that suffice to guard *any* polygon of size *n.*

- In other words,.we wish to find the maximum over all polygons of size *n* of the minimum number of guards needed to cover the polygon.
- Does it help to partition *P* into simpler objects that are easy to guard?

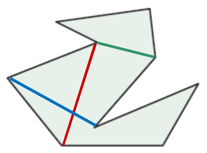




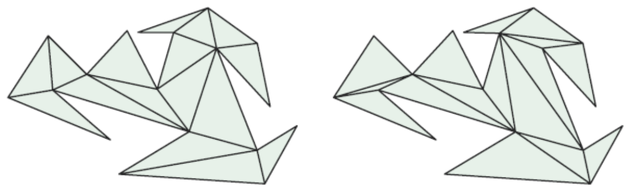
32

Triangulations

- Let P be a polygon. A ***diagonal*** of P is a line segment connecting two vertices of P and lying in the interior of P, not touching the boundary ∂P except at its endpoints



- Two diagonals are ***non-crossing*** if they share no interior points
- A ***triangulation*** of P is a decomposition of P into triangles using a ***maximal*** set of non-crossing diagonals

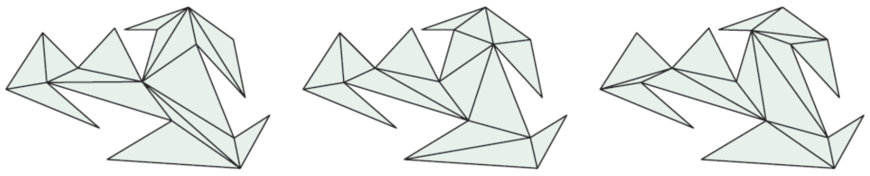


Triangulations…

- The notion of triangulation of a polygon suggests some interesting questions:
  1. Do all polygons admit a triangulation?
  2. What is the number of triangles in each triangulation of a polygon?
  3. How many triangulations does a given polygon have?

Counting Triangulations

Open problem. Identify features of polygons that lead to a formula for counting the number of triangulations of an arbitrary polygon P or, at least, to an efficient algorithm for computing this number.



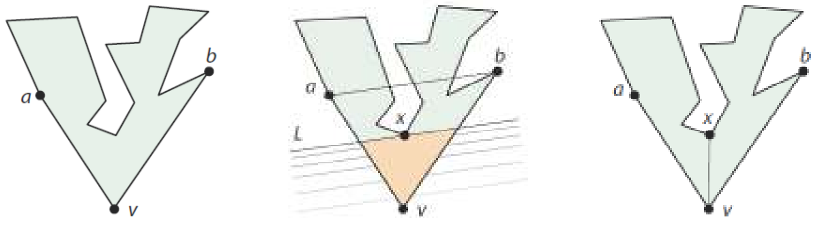
Exercise. What is the range of possible values for the number of triangulations of an n-gon?

35

Triangulations…

- Does *every* polygon admit a triangulation? If so, how many diagonals and triangles does it have?

Lemma. Every polygon with $n > 3$ vertices has a diagonal.



Theorem. Every polygon has a triangulation.

Proof. By induction on n.

Exercise. Prove that every polygon with holes has a triangulation.

36

Combinatorics and Algorithmics

Theorem. Every triangulation of a polygon P with n vertices has $n - 3$ diagonals and $n - 2$ triangles.

Proof. By induction on n.

Exercise. Describe an efficient algorithm to triangulate a polygon P with n vertices.

What helper functions does your algorithm require?

What is the running time of your algorithm?

```
TRIANGULATE(P)
1  if |P| = 3 ▷ Base case
2      then return P
   ▷ Divide
3  Find a diagonal d in P
4  Use d to split P into P_1 and P_2
   ▷ Conquer
5  T_1 ← TRIANGULATE(P_1)
6  T_2 ← TRIANGULATE(P_2)
   ▷ Combine
7  return T_1 ∪ T_2
```

37

Ears

Definition. Three consecutive vertices p, q, r, of a polygon P form an *ear* if pr is a diagonal of P. In this case, we call q the *ear tip*.

Lemma. Every triangulation of a polygon P with $n > 3$ vertices has at least two ears.

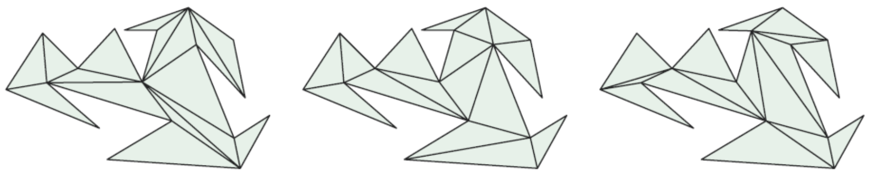
Proof. By the pigeonhole principle on the n edges of P and the $n - 2$ triangles of the triangulation of P.

We can now complete the proof of sufficiency by arguing inductively on the polygon P' obtained by removing an ear from P.

38

Exercise

1. Using an ear as a base case, design a decrease-and-conquer algorithm for computing a triangulation of an input polygon P. Explain the base case in detail
2. What is the complexity of your algorithm as a function of the number of vertices n in P?

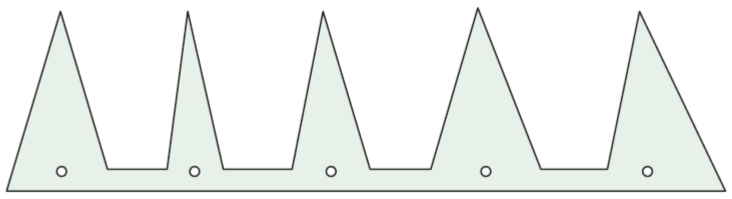


39

Guarding an Art Gallery…

Theorem. $\lfloor n/3 \rfloor$ guards are *sometimes necessary* and *always sufficient* to guard a polygon of size n.

Proof (of necessity).



Chvátal's comb for n = 15

40

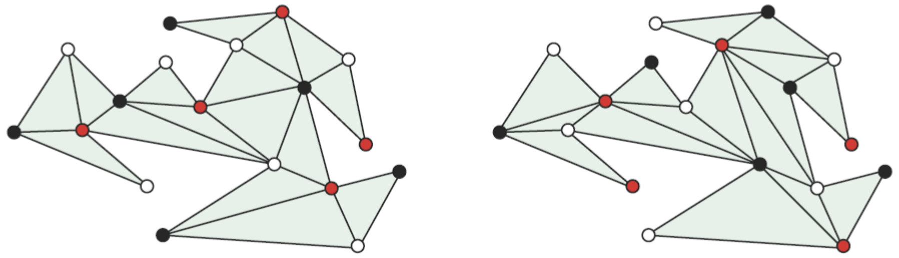
Guarding an Art Gallery…

Proof of sufficiency (Fisk 1977).

Let *T* be a triangulation of *P* viewed as a graph with vertices as nodes and polygon sides and diagonals as edges

Assign each node of *T* one of three colors (blue, red, or white) so that no two adjacent nodes have the same color.

Every triangle contains all three colors and the least frequently used color appears on at most $\lfloor n/3 \rfloor$ vertices.



Proof of Correctness

- By induction on number of vertices n
- Base case: P is a triangle
- Inductive step (for $n > 3$): triangulation T has an ear abc whose removal results in a polygon P' of size $n - 1$, and triangulation T' that can be 3-colored
- Place guards at any color occurring $\leq \lfloor n/3 \rfloor$ times, and color the ear tip b with the color not of a or c

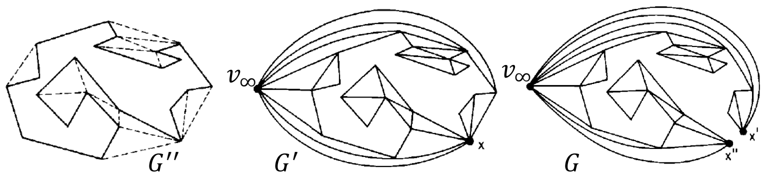
Exercise

- Does the coloring argument extend correctly to polygons with holes?

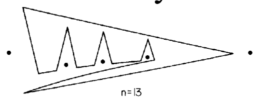
42

Exercise: The Fortress Problem

- Does guarding the *outside* of a polygon P of size n require more or fewer guards?
- Suppose that guards must reside at input vertices. Show that $\lceil n/2 \rceil$ are sometimes necessary and always sufficient to guard the *exterior* of a polygon of size n



- How many guards are sufficient and necessary if guards can be placed anywhere?



43

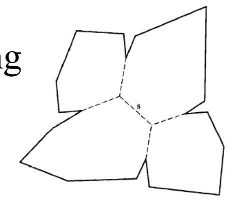
Open Problems

- Let P be a polygon. The ***visibility graph*** of P has one node per vertex of P and an edge between every pair of nodes that admit a diagonal. Find necessary and sufficient conditions to determine if a given graph is the visibility graph of some polygon
- An ***edge guard*** along an edge e sees $p \in P$, if there is $q \in e$ such that p is visible to q. How many edge guards suffice to always cover a polygon P of size n?
- Suppose the edges of P are perfect mirrors. Prove or disprove that only one guard is needed to cover P
- Are the answers different for orthogonal polygons?

44

Partitioning into Convex Polygons

- Consider now the problem of partitioning a polygon P into the smallest possible number of convex polygons



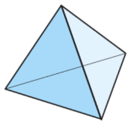
Definition. Let P be a simple polygon. A vertex v of P is a *reflex vertex* if the interior angle at v is $> 180°$

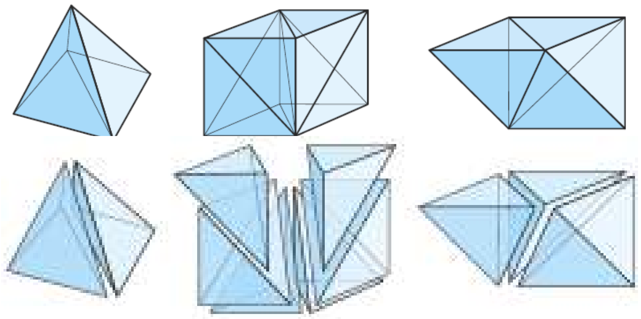
Exercise. Let P be a simple polygon and m the number of reflex vertices in P. Prove that the number ρ of polygons in an optimal convex partition of Psatisfies $\lceil m/2 \rceil + 1 \leq \rho \leq m + 1$

45

3D Decompositions and 3D Galleries

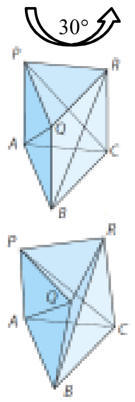
- A *polyhedron* is a 3D version of a polygon, a solid bounded by finitely many polygonal faces.
- A *tetrahedron* is the simplest polyhedron and generalizes the triangle.
- Can a polyhedron be decomposed into tetrahedra?





46

2D vs. 3D



- Not all polyhedra can be triangulated, as the “twisted prism” on the right shows.

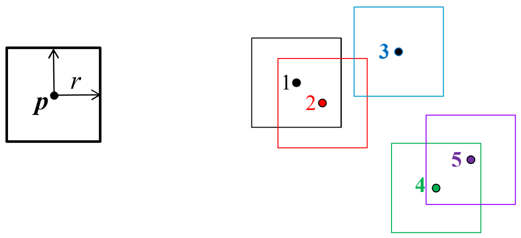
  Open problem. Identify key features of polyhedra that lead to a simple way to determine if a polyhedron can be triangulated.

- Different tetrahedralizations of the same polyhedron may have different number of tetrahedra.
- Placing a guard at every vertex of a polyhedron may not be enough to cover the interior.

47

Problem 3: Fixed Distance Neighbors

- Given a set P of n points in R^d and $r \geq 0$, find all k pairs $p,q \in P$ whose corresponding coordinates differ by at most r, i.e. $|p_i - q_i| \leq r$, for all $1 \leq i \leq d$
- Corresponds to L_∞ metric: $d_\infty(p,q) = \max_{1 \leq i \leq d} | p_i - q_i |$



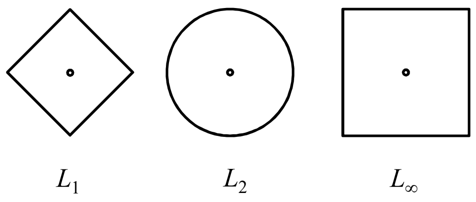
48

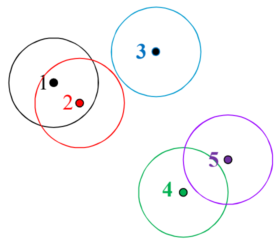
Metric

- A *metric* on a set S is a function d: $S \times S \rightarrow R$ that satisfies
  - *Non-negativity*: $d(p,q) \geq 0$
  - *Positive-definiteness*: $d(p,q) = 0$ iff $p = q$
  - *Symmetry*: $d(p,q) = d(q,p)$
  - *Triangle Inequality*: $d(p,q) \leq d(p,r) + d(r,q)$
- The metric defines the distance between elements of S

Other Common Metrics

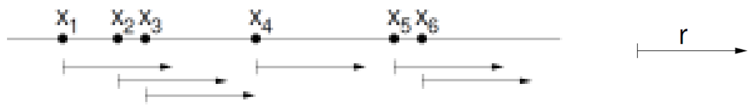
- In general, L_t distance: $$d_t(p,q) = \left(\sum_{1 \le i \le d} |p_i - q_i|^t \right)^{1/t}$$
- Our example uses L_∞: $$d_\infty(p,q) = \max_{1 \le i \le d} |p_i - q_i|$$





Fixed Distance Neighbors in 1D

- Get insight by working 1D case ($d = 1$) first
- Reporting vs. counting
  - k is output size, k_i = # of near successors of p_i
- Does sorting help?



$$T(n,k) = n\log n + \sum_{i=1}^{n}(k_i + 1) = n\log n + n + k = O(n\log n + k)$$

51

1D Solution with Bucketing

- Partition the real line into half-open "buckets" of length r

$$\ldots,[-3r,-2r),[-2r,-r),[-r,0),[0,r),[r,2r),[2r,3r),\ldots$$

- Bucket $[br, (b+1)r)$ has bucket index b
- x goes into bucket with index $b(x) = \lfloor x/r \rfloor$
- At most n buckets are occupied
- Store occupied buckets in a hash table of size $O(n)$

52

How does Bucketing Help?

- If p lies in bucket b, then every near-neighbor of p lies in buckets b, $b - 1$, or $b + 1$
  - Points in bucket b are near-neighbors
  - Points in $b - 1 \cup b + 1$ are uncertain

```
1. For each $p \in P$, insert $p$ in $H$ using key $b(p)$
2. For each $p \in P$ do
   a. Compute $b(p)$
   b. For each point $q$ in bucket $b(p)$ such that $q \neq p$,
      output $(p, q)$
   c. For each point $q$ in bucket $b(p) + 1$ such that
      $| q - p | \leq r$, output $(p, q)$
```

53

Analysis

Lemma. For any $x, y : xy \leq (x^2+y^2) / 2$

Theorem. Let D denote the total number of distance computations in step 2c. Then $D = O(n + k)$

Proof.

$$D = \sum_b n_b(n_{b+1}) \leq \sum_b \frac{n_b^2 + n_{b+1}^2}{2}$$

$$= \sum_b \frac{n_b^2}{2} + \sum_b \frac{n_{b+1}^2}{2} = \sum_b n_b^2 = O(n+k)$$

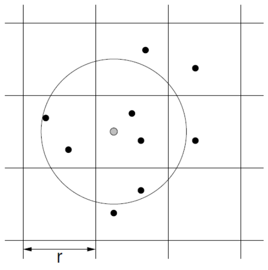
Corollary. The bucketing algorithm runs in $O(n)$ space and $O(n + k)$ expected time.

54

Generalization

- Generalize to d-dimensions and other L_t metrics

  Example: 2D ⇒ use grid of buckets

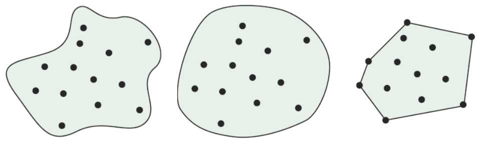


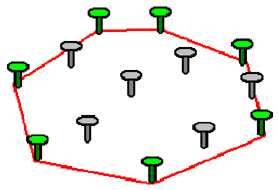
- Hash key now has d coordinates: $(\lfloor x_1/r \rfloor, \ldots, \lfloor x_d/r \rfloor)$
- Every near neighbor of p resides in one of 3^d buckets
- Algorithm still runs in $O(n + k)$ time

55

Problem 4: Convex Hulls

- The *smallest* convex set that contains a set of points P, denoted **conv(P)**, is called the ***convex hull*** of P
  - conv(P) is the intersection of all convex sets that contain P
  - The ***extreme points*** of P are the vertices of conv(P)
- Applications
  - Pattern recognition
  - Shape approximation
  - Collision detection
  - Geographic information systems
  - Image processing
  - Component of other geometric algorithms





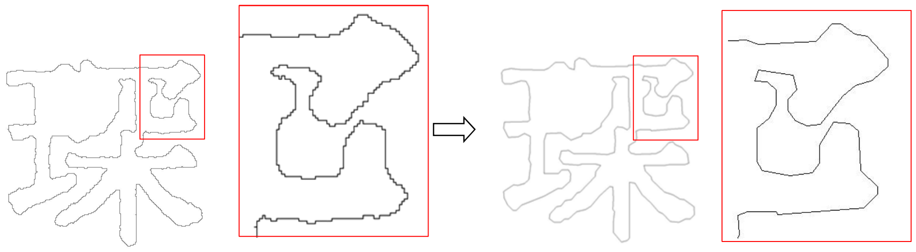
56

Applications

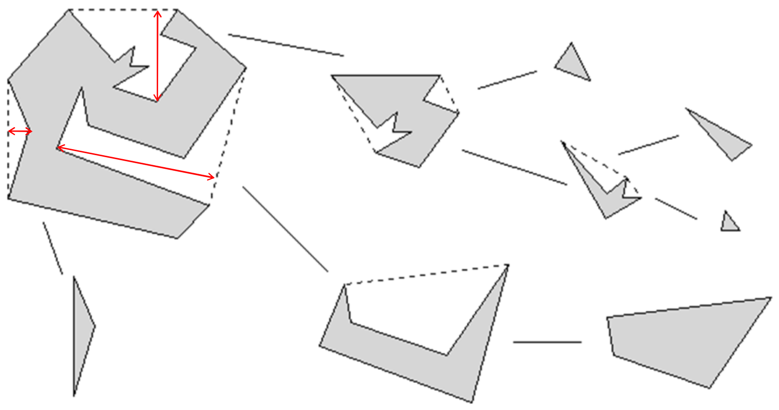
- Data classification

- Region analysis in GIS

- Scanned character smoothing

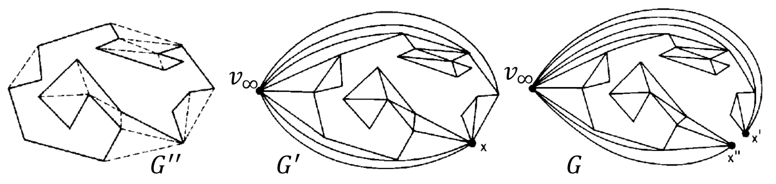


Scanned Character Smoothing

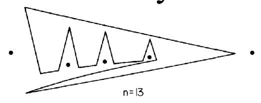


Exercise: The Fortress Problem

- Does guarding the *outside* of a polygon P of size n require more or fewer guards?
- Suppose that guards *must* reside at input vertices. Show that $\lceil n/2 \rceil$ are sometimes necessary and always sufficient to guard the *exterior* of a polygon of size n



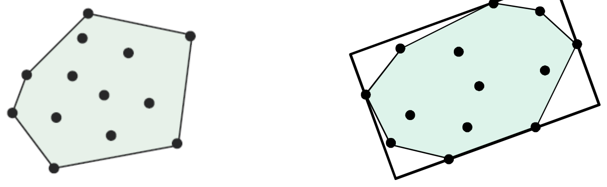
- How many guards are sufficient and necessary if guards can be placed anywhere?



59

Exercise

- Argue that $\text{conv}(P)$ is a convex polygon whose vertices belong to P
- Argue that $\text{conv}(P)$ has the smallest perimeter and smallest area among all polygons that contain P



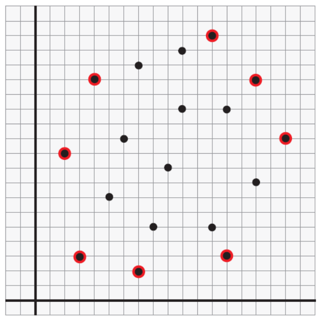
- Argue that the smallest-area rectangle enclosing P must contain one of its sides flush with an edge of $\text{conv}(P)$

60

Algorithms

- Our goal is to develop efficient algorithms to compute the convex hull of a finite set P of points in the plane
- What does it mean to compute the convex hull?
  - Definition of conv(P) is not computationally useful
  - It suffices to compute the polygonal boundary
  - How do we describe the boundary?

(2, 10)	(10, 17)
(3, 3)	(10, 13)
(4, 15)	(12, 5)
(5, 7)	(12, 18)
(6, 11)	(13, 3)
(7, 2)	(13, 13)
(7, 16)	(15, 8)
(8, 5)	(15, 15)
(9, 9)	(17, 11)



61

Recall…

- We adhere to the following design methodology:
  1. Understand the geometry of the *general case*, ignoring "degenerate" cases. Identify useful primitives (both data and functionality)
  2. Design algorithm for the general case
  3. Extend the algorithm to handle degenerate cases
  4. Provide a *robust* implementation of your algorithm, including required primitives and predicates
- The definition of conv(P) as the intersection of all convex sets containing P is not computationally useful. How do you apply the above methodology?

62

Recall…

- We adhere to the following design methodology:
  1. Understand the geometry of the *general case*, ignoring “degenerate” cases. Identify useful primitives (both data and functionality)
  2. Design algorithm for the general case
  3. Extend the algorithm to handle degenerate cases
  4. Provide a *robust* implementation of your algorithm, including required primitives and predicates
- The definition of conv(P) as the intersection of all convex sets containing P is not computationally useful. How do you apply the above methodology?

63

Convex Hull: Algorithm 1

Idea. If ℓ is the supporting line of an edge of conv(P) then all points of P are on the same side of ℓ

```
1. $E \leftarrow 0$
2. **for** all ordered pairs $(p,q) \in P \times P, p \neq q$ **do**
3.     valid ← **true**
4.     **for** all points $r \in P$ not equal to $p$ or $q$ **do**
5.         **if** $r$ lies to the right of directed line from $p$ to $q$
6.             **then** valid ← **false**
7.     **if** valid **then** add the directed edge $pq$ to $E$
8. Using set $E$ construct a list $L$ of vertices of conv($P$)
       sorted in CCW order.
```

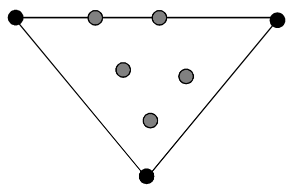
Time: $\Theta(n^3)$

64

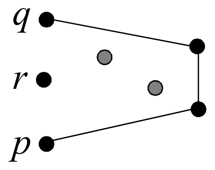
Practical Considerations

- Solution must handle *degenerate cases*.

  Example: three or more collinear points on an edge of conv(P)



- Solution must be *robust.* Example: inaccurate testing when using floating point arithmetic



65

Floating Point Example 1

```
% cat fperror1.c
#include <iostream.h>
int main(){
   double a,b;
      a = 1+2∗sqrt(2);
      b = sqrt(9+4∗sqrt(2));
      if (a == b)  cout << a << " is equal to " << b << endl;
      else  cout << a << " is NOT equal to " << b << endl;
}
% fperror1
3.82843 is NOT equal to 3.82843
```

66

Floating Point Example 2

```
% cat fperror2.c
#include <stdio.h>
main(){
   double f,a,b;
   a = 77617.0;  b = 33096.0;
   f = 333.75*b*b*b*b*b*b + a*a*( 11*a*a*b*b  −  b*b*b*b*b*b
        − 121*b*b*b*b −2) + 5.5*b*b*b*b*b*b*b*b + a/(2*b);
   printf("f is %g   ",f);
   f = 333.75*pow(b,6) + a*a*( 11*a*a*b*b − pow(b,6) −
      121*pow(b,4) − 2) + 5.5*pow(b,8) + a/(2*b);
   printf("f is %g\n",f);
}
% fperror2
f is 1.1726   f is −1.18059e+21
```

67

Conclusion

- Avoid floating point arithmetic as much as possible. Instead, use an arithmetic package that supports arbitrarily large integers or rational numbers
  - Generic: https://en.wikipedia.org/wiki/List_of_arbitrary-precision_arithmetic_software
  - Geometric support:
    - LEDA
    - CGAL
- Recommended reading

  What Every Computer Scientist Should Know About Floating-Point Arithmetic, by David Goldberg.
  ACM Computing Surveys, March 1991

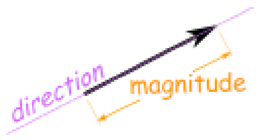
68

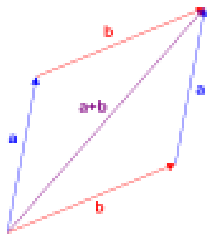
Vectors

- Another geometric primitive in $\mathbb{R}^d$
- Points and vectors are different:
  - A point is a location in space
  - A vector is a displacement in space
- Attributes: direction and magnitude
- Use the same representation as points

```
struct Vector{ int x, y; };
Vector W, NE, cardinal_directions[4];
```

- Some operations: $+$, $-$, scaling
- Also: $vector + point \rightarrow point$
  $point - point \rightarrow vector$
- How about vector multiplication?

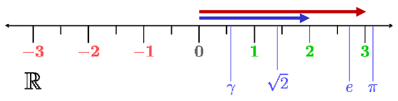




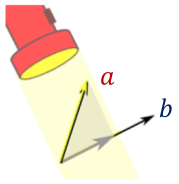
69

Multiplying Vectors

- Two operations:
  - Inner, scalar, or dot product ·
  - Cross product × (for 3D vectors only)
- The dot product is a generalization of scalar multiplication
  - ordinary scalar product can be interpreted as multiplying two vectors aligned in the same direction
  - for vectors, we use the contribution of one vector in the direction of the other, but what is this contribution?



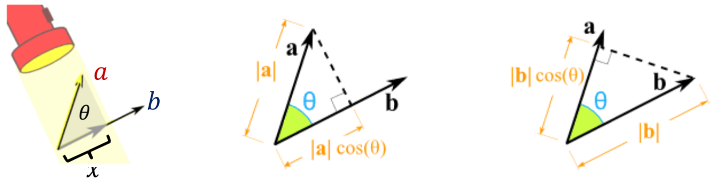
$a \cdot b = 6$



70

Dot Product ·

- If θ is the angle between a and b, then $\cos\theta = x/|a|$



- Thus, we want $a \cdot b \coloneqq |a||b|\cos\theta$
- Why is this useful to computation?
- Properties:
  - $u \cdot v = v \cdot u$ (commutativity)
  - For $\alpha \in \mathbb{R}, \alpha(u \cdot v) = (\alpha u) \cdot v = u \cdot (\alpha v)$
  - $(u + v) \cdot w = u \cdot w + v \cdot w$
  - $u \cdot v = u_x v_x + u_y v_y$, **prove it!**

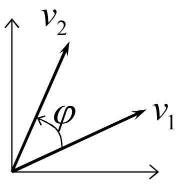
71

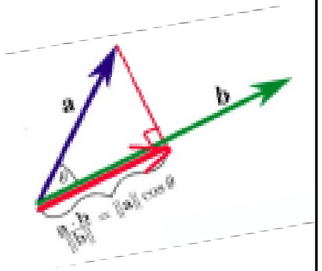
Dot Product ·

$$\text{dot}(v_1, v_2) = v_1 \cdot v_2 = x_1 x_2 + y_1 y_2 = |v_1||v_2|\cos\varphi$$

where $v_i = (x_i, y_i)$

$$\text{dot}(v_1, v_2)\begin{cases} > 0 & \text{if } 0 \le \varphi < \pi/2 \\ < 0 & \text{if } \pi/2 < \varphi \le \pi \\ = 0 & \text{if } \varphi = \pi/2 \end{cases}$$





Note:

$\text{dot}(v_1, v_2) = 0$ iff $v_1 \perp v_2$

$u^{\perp} = (-u_y, u_x)$

72

Exercise

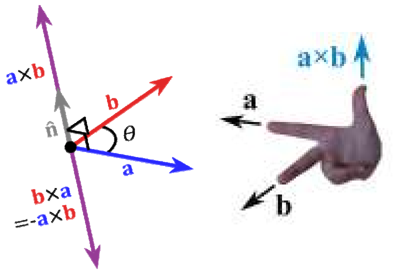
- Given four points A, B, P, Q determine which of P or Q is farther from the line through A and B, without explicitly computing distances
- Your solution should avoid divisions

73

Cross Product $\times$

- Defined for 3D vectors only
- Cross product of $a = (x_a, y_a, z_a)$ and $b = (x_b, y_b, z_b)$

$$a \times b = \det\begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_a & y_a & z_a \\ x_b & y_b & z_b \end{pmatrix}$$



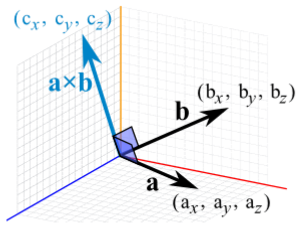
- Note: $a \times b = (|a||b| \sin\theta)\hat{n}$, where $\hat{n}$ is the unit normal to the plane subtended by a and b, in the direction given by the ***right hand rule***

74

Cross Product × in 2D

- Cross product of $\vec{v}_1 = (x_1, y_1, 0)$ and $\vec{v}_2 = (x_2, y_2, 0)$

$$\det\begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_1 & y_1 & 0 \\ x_2 & y_2 & 0 \end{pmatrix} = (x_1 y_2 - x_2 y_1)\vec{k}$$



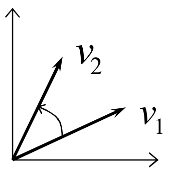
- Useful to define:

$$\text{cross}(v_1, v_2) = \det\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1$$

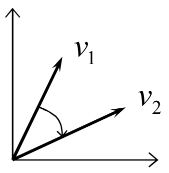
75

Cross Product Properties

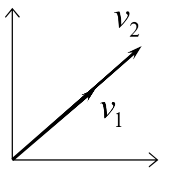
$$\text{cross}(v_1, v_2)\begin{cases} > 0 & \text{if } v_1 \text{ is clockwise from } v_2 \\ < 0 & \text{if } v_2 \text{ is clockwise from } v_1 \\ = 0 & \text{if } v_1 \text{ and } v_2 \text{ are collinear} \end{cases}$$



$\text{cross}(v_1, v_2) > 0$



$\text{cross}(v_1, v_2) < 0$

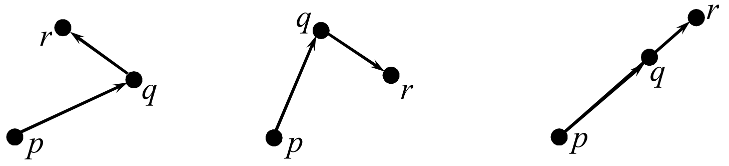


$\text{cross}(v_1, v_2) = 0$

76

A Basic Primitive: Turn

$$\text{turn}(p,q,r) = \begin{cases} \text{left} & \text{if cross}(q-p, r-p) > 0 \\ \text{right} & \text{if cross}(q-p, r-p) < 0 \\ \text{no\_turn} & \text{if cross}(q-p, r-p) = 0 \end{cases}$$

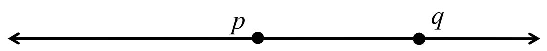


left turn　　right turn　　no turn

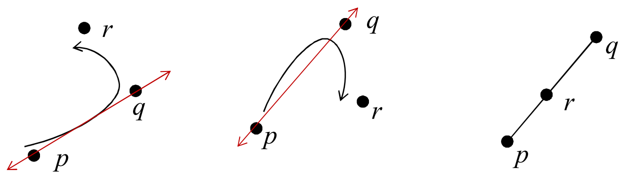
77

Orientation of Points

- In order to make reliable discrete decisions we need robust operators similar to the relational operators <, =, >



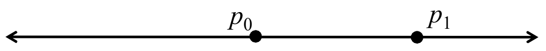
- A triple of points ⟨*p*,*q*,*r*⟩ has *positive orientation* iff the path *pqr* turns left (CCW)



positive　　negative　　zero

78

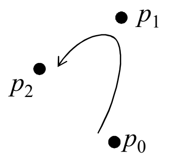
1D:



To determine the side of p_0 containing p_1 use the sign of $x_1 - x_0$

$$\text{orient}(p_0, p_1) = \det\begin{pmatrix} 1 & x_0 \\ 1 & x_1 \end{pmatrix} = x_1 - x_0$$

2D: $$\text{orient}(p_0, p_1, p_2) = \det\begin{pmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{pmatrix}$$



3D: $$\text{orient}(p_0, p_1, p_2, p_3) = \det\begin{pmatrix} 1 & x_0 & y_0 & z_0 \\ 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \end{pmatrix}$$

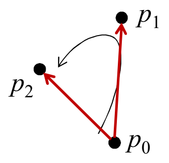
79

2D Orientation

$$\text{orient}(p_0, p_1, p_2) = \det\begin{pmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{pmatrix}$$

$$= x_1y_2 - x_2y_1 - x_0y_2 + x_2y_0 + x_0y_1 - x_1y_0$$

$$= (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$



80

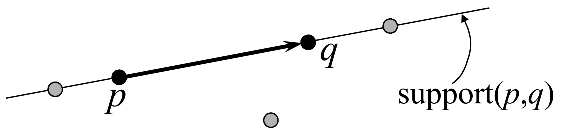
Convex Hull Edge Detection

The directed edge pq is invalid if $\exists\, r$:

$$\text{cross}(q-p, r-p) < 0$$

or

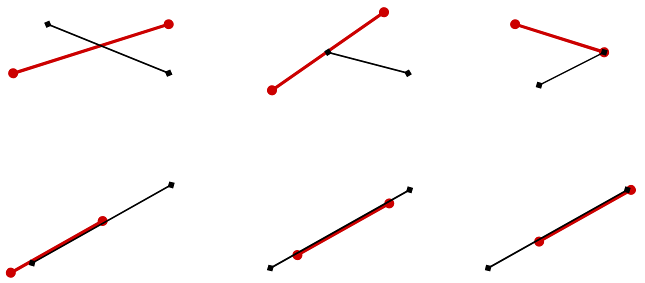
$$\text{cross}(q-p, r-p) = 0 \text{ and } \text{dot}(r-q, r-p) > 0$$



81

Example: Segment Intersection

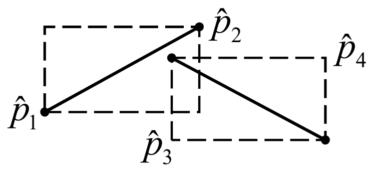
- Segments $p_1 p_2$ and $p_3 p_4$ intersect if they have at least one point in common



82

Bounding Boxes

- The bounding box of a segment p_1p_2 is the upright rectangle $[\hat{p}_1\hat{p}_2]$ where $\hat{x}_1 = \min(x_1, x_2)$ $\hat{x}_2 = \max(x_1, x_2)$, $\hat{y}_1 = \min(y_1, y_2)$, $\hat{y}_2 = \max(y_1, y_2)$



- Two bounding boxes $[\hat{p}_1\hat{p}_2]$ and $[\hat{p}_3\hat{p}_4]$ intersect iff the following is true:

$$(\hat{x}_2 \geq \hat{x}_3) \wedge (\hat{x}_4 \geq \hat{x}_1) \wedge (\hat{y}_2 \geq \hat{y}_3) \wedge (\hat{y}_4 \geq \hat{y}_1)$$

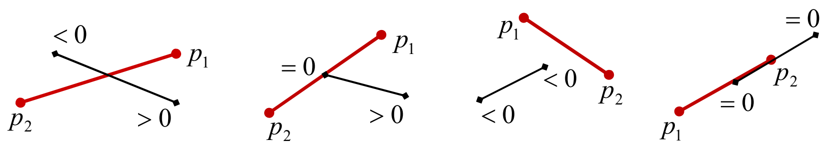
83

Segment Intersection

- Segments p_1p_2 and p_3p_4 intersect iff :

  1) bbox(p_1, p_2) intersects bbox(p_3, p_4)
  2) p_1p_2 straddles support(p_3p_4)
  3) p_3p_4 straddles support(p_1p_2)

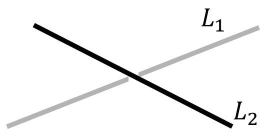
- p_3p_4 straddles support(p_1p_2) iff : orient$(p_1, p_2, p_3) *$ orient$(p_1, p_2, p_4) \leq 0$



84

Exercises

- Find the distance between 3D lines $L_1 \colon p_1 + a\vec{v}_1$ and $L_2 \colon p_2 + b\vec{v}_2$, for $a, b \in \mathbb{R}$



- Given a polygon *P* design an efficient algorithm to determine whether
  - *P* is convex or not
  - *P* is given in CW or CCW order
  - *P* is simple or not