Introduction to Algorithm Design

Lecture Notes 5

ROAD MAP

- **Divide And Conquer**
  - Binary Search
  - Maximum Subsequence Sum Problem
  - Merge Sort
  - Quick Sort
  - Binary Tree and Its Properties
  - **Multiplication of Large Integers**
  - **Strassen’s Matrix Multiplication**
  - Closest Pair of Points
  - Convex Hull

Multiplication of Large Integers

- Some applications require manupilation of large integers (over 100 decimal digits long)
  - Such as cryptology
- Such integers are too long to fit in a special word of a modern computer
  - They require special treatment
  - Does not take unit time

Multiplication of Large Integers

- Classical pen-pencil algorithm for multiplying two *n-digit* integer
  - Each of *n* digits of the first number is mulipled by each of *n* digits of second number
- The total is n^2 digit multiplications
- Is it possible to design an algorithm with fewer than n^2 digit multiplication?

Multiplication of Large Integers

Example: multiply 23 and 14

$$23 = 2 \cdot 10^1 + 3 \cdot 10^0 \text{ and } 14 = 1 \cdot 10^1 + 4 \cdot 10^0.$$

$$\begin{aligned} 23 &= (2 \cdot 10^1 + 3 \cdot 10^0) * (1 \cdot 10^1 + 4 \cdot 10^0) \\ &= (2 * 1)10^2 + (3 * 1 + 2 * 4)10^1 + (3 * 4)10^0 \end{aligned}$$

- There are 4 multiplications in total
- The middle term can also be calculated as

$$3 * 1 + 2 * 4 = (2 + 3) * (1 + 4) - (2 * 1) - (3 * 4)$$

- So the result can be obtained by three multiplications only

Multiplication of Large Integers

In general:

For any pair of two-digit integers $a = a_1 a_0$ and $b = b_1 b_0$, their product ***c*** can be computed by the formula

$$c = a*b = c_2 10^2 + c_1 10^1 + c_0$$

where

$c_2 = a_1 * b_1$ → product of their first digits

$c_0 = a_0 * b_0$ → product of their second digits

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$ → product of the sum of the *a's* digits and the sum of the *b's* digits minus the sum of c_2 and c_0

Multiplication of Large Integers

- **<u>Approach :</u>**

  If we want to multiply two *n-digit* integers a and b where a is positive even number

  - Divide both numbers in the middle
  - Denote first half of the a's digits by a_1 and second half by a_0
    - Same notations for b
  - $a = a_1 a_0$ implies that $a = a_1 10^{n/2} + a_0$ and

    $b = b_1 b_0$ implies that $b = b_1 10^{n/2} + b_0$

Multiplication of Large Integers

- We get

  $c = a*b = (a_1 10^{n/2} + a_0) * (b_1 10^{n/2} + b_0)$

  $c = (a_1*b_1) 10^n + (a_1*b_0 + a_0*b_1) 10^{n/2} + (a_0*b_0)$

  $c = c_2 10^n + c_1 10^{n/2} + c_0$

  where

  $c_2 = a_1*b_1$ → product of their first halves

  $c_0 = a_0*b_0$ → product of their second halves

  $c_1 = (a_1+a_0)*(b_1+b_0)-(c_2+c_0)$ → product of the sum of the *a's* halves and the sum of the *b's* halves minus the sum of c_2 and c_0

Multiplication of Large Integers

- If *n/2* is even, we can apply same method for computing products of c_2, c_1 and c_0.
- Thus we have a recursive algorithm to compute product of two *n-digit* integers
- Recursion is stopped
  - when n becomes 1
  - when we deem *n* small enough to multiply the numbers of that size directly

Multiplication of Large Integers

- <u>**Analysis :**</u>

  How many digit multiplications does this algorithm make?

Multiplication of Large Integers

- **Analysis :**

  Multiplication of n-digit numbers requires three multiplications of *n/2* digit number

  So

  $$M(n) = 3M(n/2) \text{ for } n > 1, \quad M(1) = 1$$

  solving it by backward substitution for n = 2^k yields

  $$M(2^k) = 3M(2^{k-1}) = 3[3M(2^{k-2})] = 3^2 M(2^{k-2})$$
  $$= \cdots = 3^i M(2^{k-i}) = \cdots = 3^k M(2^{k-k}) = 3^k.$$

  since k = $\log_2 n$

  $$M(n) = 3^{\log_2 n} = n^{\log_2 3} \approx n^{1.585},$$

Multiplication of Large Integers

Discussion :

- Used in many problems today
  - Cryptography
  - Security units of mobile devices
- Divide and conquer algorithm outperform the pen-and-pencil method on integers over 600 digits long

Matrix Multiplication

- **Problem Definition :**

Find product *C* of two *n-by-n* matrices *A* and *B*

- We will see that matrix multiplication can be done using less than n^3 scalar multiplications

Matrix Multiplication

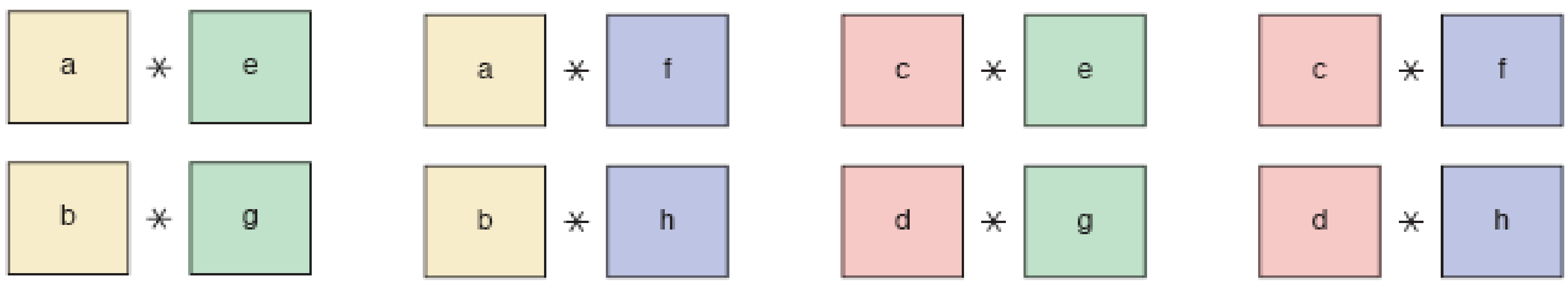
A simple divide and conquer strategy:

- Let *A* and *B* be two *n-by-n* matrices where *n* is a power of *2*
- We can divide *A, B* and their product *C* into four *n/2-by-n/2* submatrices each as follows

$$\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

Matrix Multiplication

<u>8 Sub-Problems:</u>



<u>Analysis:</u>

- 8 multiplication operation → (n/2)-by-(n/2) matrix
- 4 addition operation → (n/2)-by-(n/2) matrix

- $T(n)=8*T(n/2)+\Theta(n^2) = \Theta(n^3)$

Strassen's Matrix Multiplication

- To perform matrix multiplication using less than n^3 scalar multiplications
- First lets consider the case of *2-by-2* matrix multiplication
  - We will show that this can be done using 7 multiplications instead of 8 multiplications required by brute-force algorithm.

Strassen's Matrix Multiplication

We can use the following formulas

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ b_{10} & b_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

where

$$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$$
$$m_2 = (a_{10} + a_{11}) * b_{00}$$
$$m_3 = a_{00} * (b_{01} - b_{11})$$
$$m_4 = a_{11} * (b_{10} - b_{00})$$
$$m_5 = (a_{00} + a_{01}) * b_{11}$$
$$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$$
$$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11}).$$

Strassen's Matrix Multiplication

- There are 7 multiplications.
- But how many additions are there?
- Is it good idea to use this method for *2-by-2* matrices?

Strassen's Matrix Multiplication

$$\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

7 multiplication operation

P1=a*(f-h) P2=(a+b)*h P3=(c+d)*e P4=d*(g-e)
P5=(a+d)*(e+h) P6=(b-d)*(g+h) P7=(a-c)*(e+f)

Solution:

a*e+b*g = P5+P4-P2+P6

a*f+b*h = P1+P2

c*e+b*h = P3+P4

c*f + d*h = P5+P1-P3-P7

Strassen's Matrix Multiplication

Approach:

- Let *A* and *B* be two *n-by-n* matrices
  - where *n* is a power of *2*
- Divide *A* and *B* into four *n/2-by-n/2* submatrices
- Calculate 7 submatrix multiplications recursively
- Perform required additions to obtain the matrix C

Strassen's Matrix Multiplication

- **Analysis :**

$$M(n) = 7M(n/2) \text{ for } n > 1, \quad M(1) = 1.$$

Since $n = 2^k$,

$$\begin{aligned} M(2^k) &= 7M(2^{k-1}) = 7[7M(2^{k-2})] = 7^2 M(2^{k-2}) = \cdots \\ &= 7^i M(2^{k-i}) \cdots = 7^k M(2^{k-k}) = 7^k. \end{aligned}$$

Since $k = \log_2 n$,

$$M(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807},$$

Strassen's Matrix Multiplication

Discussion :

- Saving in # of multiplications was achieved at the expense of making extra additions
  - We must check # of additions A(n)
  - $A(n) \in \Theta(n^{\log_2 7})$
  - Same order of growth as # of multiplication
- Efficiency is better than brute force
  - Brute force algorithm is n^3
- Is it good for memory efficiency?
- It is not the best algorithm for matrix multiplication
  - Coopersmith and Winogrand algorithm's efficiency is $O(n^{2.376})$

ROAD MAP

- **Divide And Conquer**
  - Binary Search
  - Maximum Subsequence Problem
  - Merge Sort
  - Quick Sort
  - Multiplication of Large Integers
  - Strassen’s Matrix Multiplication
  - **Closest Pair of Points**
  - **Convex Hull**

Closest-Pair Problem

- **Problem Definition :**

  Find the closest points in a set of *n* points.

  - We consider the two dimensional case of the problem
  - We assume that points in question are specified in a standard fashion by their *(x,y)* Cartesian coordinates
  - We assume that distance between two points Pi = (xi, yi) and Pj = (xj,yj) is the standard Euclidean distance

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Closest-Pair Problem

Approach :

1. Divide points given into two subsets S_1 and S_2 of n/2 points each by drawing a vertical line x=c

- c is be the median μ of x coordinates
- So, n/2 points lie on the left or on the line itself, and n/2 points lie on the right

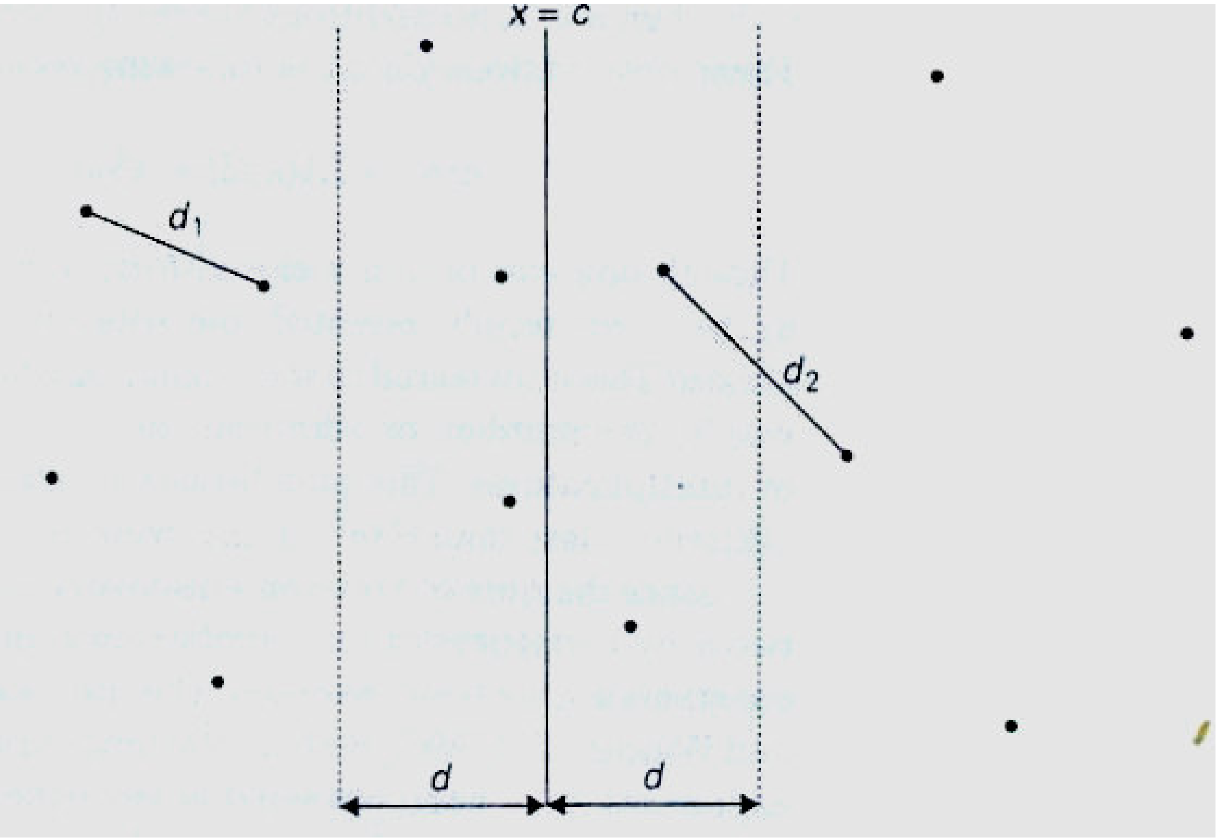
Closest-Pair Problem

Approach :

2. Find recursively the closest pair in the left subset S_1 and the right subset S_2

- let d_1 and d_2 be the smallest distances between pairs of points in S_1 and S_2
- let $d = \min\{d_1, d_2\}$
- *d* is not necessarily the smallest distance between all pairs of points in S_1 and S_2
  - A closer pair of points can lie on the opposite sides of the seperating line
- Should also consider the points in symetric vertical strip of width *2d*
  - the distance for any other pair of points is greater that *d*

Closest-Pair Problem



Idea of the d&c algorithm for the closest-pair problem

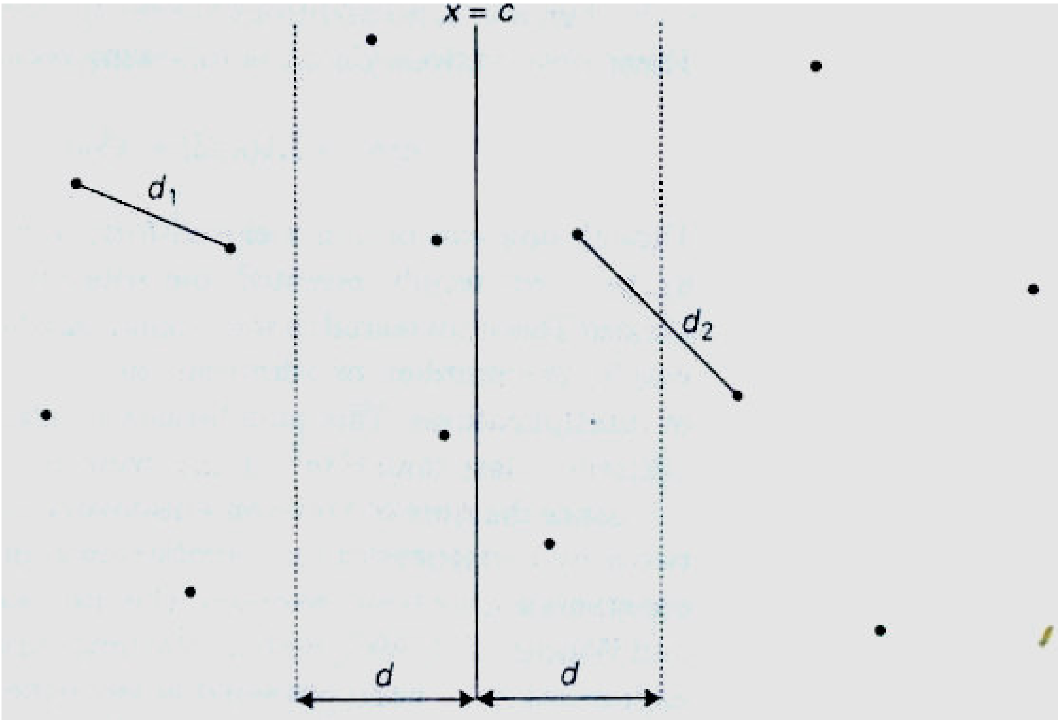
Closest-Pair Problem

Approach :

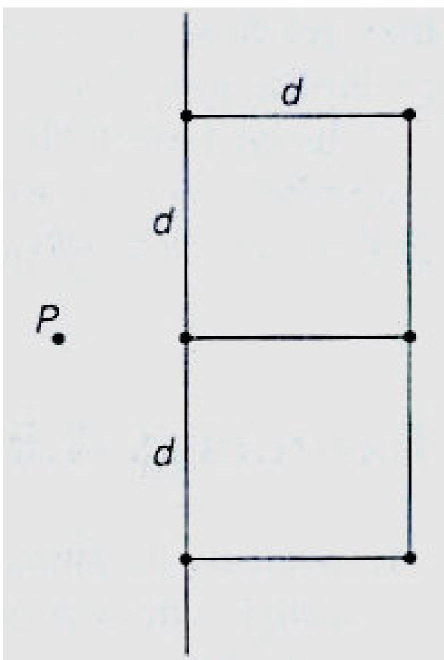
Let C_1 and C_2 be the subsets of points in the left and right parts of the strip respectively

4. For every point P(x,y) in C_1, inspect points in C_2 that may be closer to the *P* than *d*

- Such points must have their y coordinates in the interval [y−d, y+d]
- So there can be no more than such 6 points



(a) Idea of the d&c algorithm for the closest-pair problem



(b) The six points that may need to be examined for point *P* (worst case)

Closest-Pair Problem

Approach :

6. **Maintain list of points in C_1 and C_2 in ascending order of their y coordinates**
   - **This ordering can be maintained by merging two previously sorted lists**
7. **Process C_1 points sequentially while a pointer into the C_2 list scans an interval of width *2d***

Closest-Pair Problem

- **Analysis :**

The recurrence for T(n), on *n* presorted points :

$$T(n) = 2\,T(n/2) + M(n)$$

$$T(n) \in O(n \log n)$$

Convex-Hull Problem

- **Problem Definition :**

  Find the smallest convex polygon that contains n given points in a plane

Convex-Hull Problem

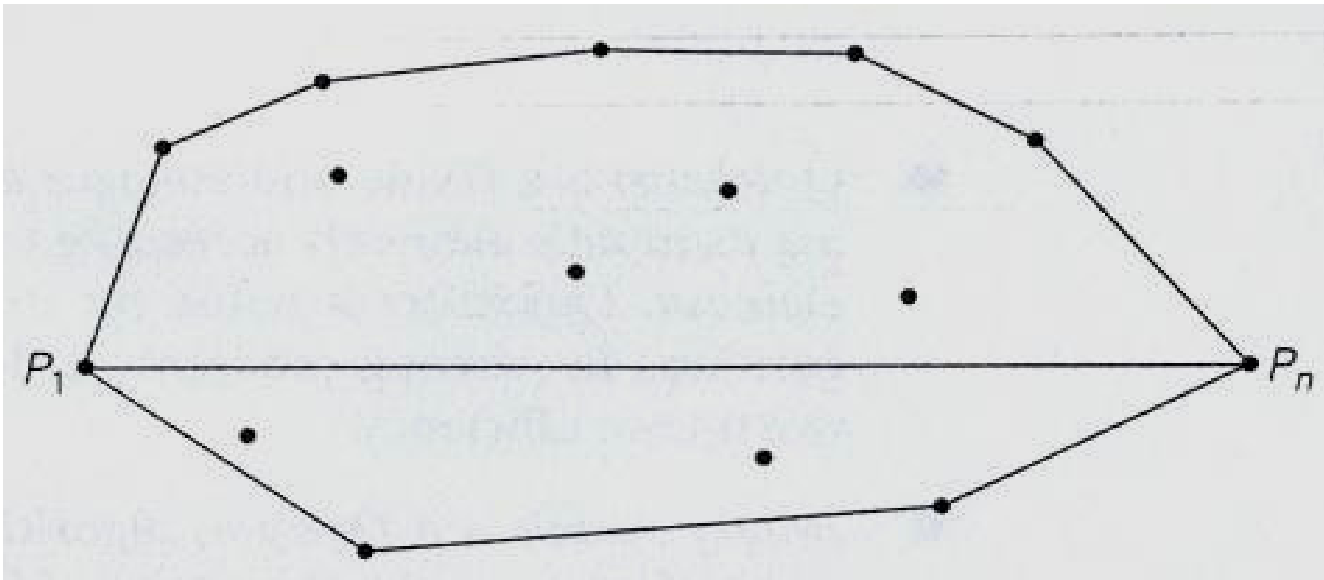
- There are several divide&conquer algorithms for the convex-hull problem.
- We will look at the simplest one !
- This algorithm is sometimes called quickhull
  - its operations resemble those of quicksort

Convex-Hull Problem

Any İdea????

Convex-Hull Problem

- Let $P_1 = (x_1, y_1), \ldots, P_n=(x_n, y_n)$ be a set of $n>1$ points in the plane
- We assume that points are sorted in increasing order of their *x* coordinates
- Leftmost point P_1 and rightmost point P_n are two distinct *extreme points*



Convex-Hull Problem

- Let $\overrightarrow{P_1P_n}$ be the straight line through point P_1 and P_n
- This line seperates the points of S into two sets
  - S_1 is the set of points on the left of this line
  - S_2 is the set of points on the right of this line
- The points of S on the line $\overrightarrow{P_1P_n}$ other than P_1 and P_n can not be extreme points of convex-hull

Convex-Hull Problem

- Boundary of convex hull of S is made of two polygonal chains
  - *Upper* boundary called *upper hull* is a sequence of line segments including
    - Line segment connecting P_1 and P_n
    - Line segments connecting some points in S_1
  - *Lower* boundary called *lower hull* is a sequence of line segments including
    - Line segment connecting P_1 and P_n
    - Line segments connecting some points in S_2
  - Convex-hull of the entire set is composed of the upper and lower hulls

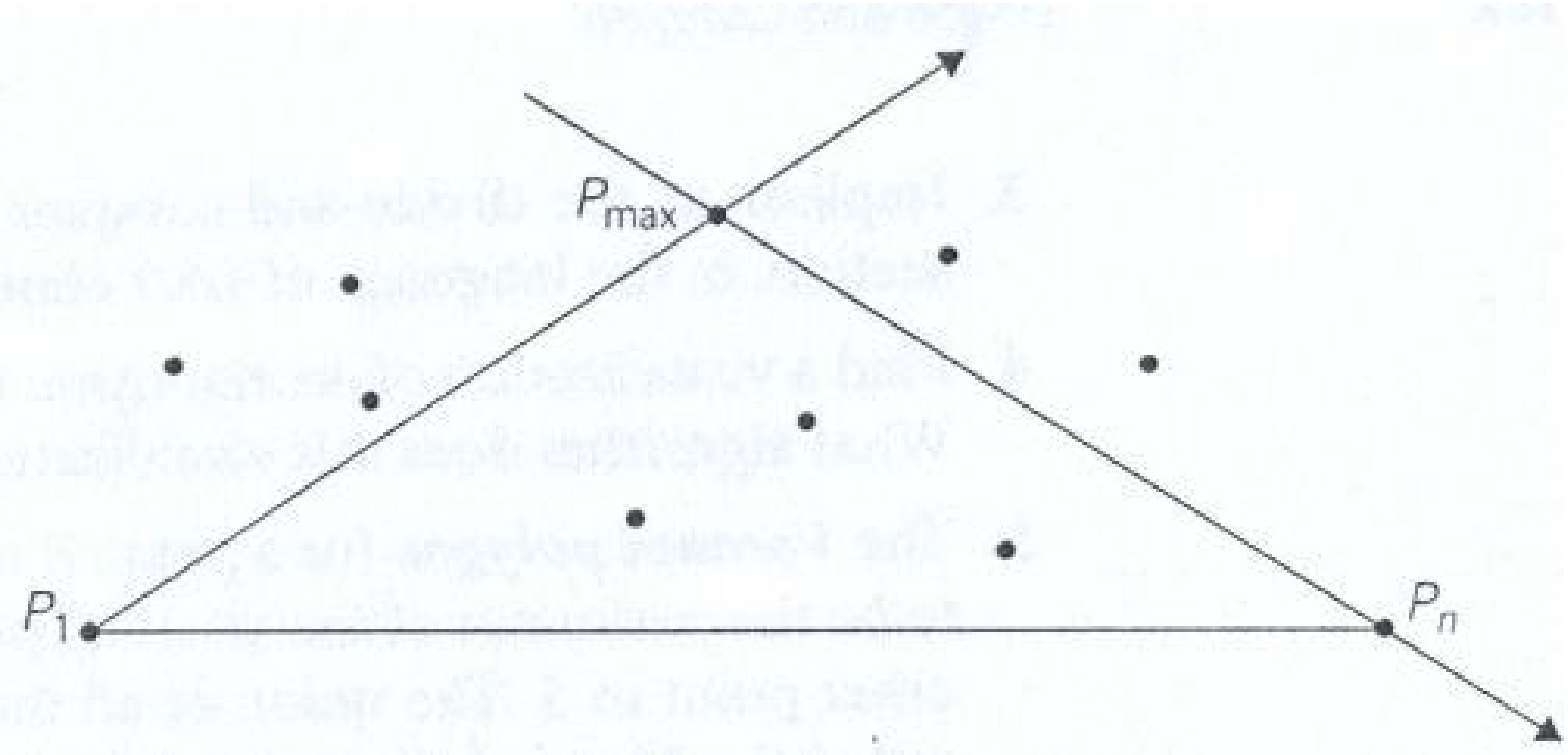
Convex-Hull Problem

- How to construct the upper hull and lower hull?
- Is the problem same as the original convex-hull problem?
- Where is the divide and conquer technique?

Convex-Hull Problem

- We will use divide-and-conquer technique to construct the upper hull and lower hull.
  - Then the algorithms is called quickhull

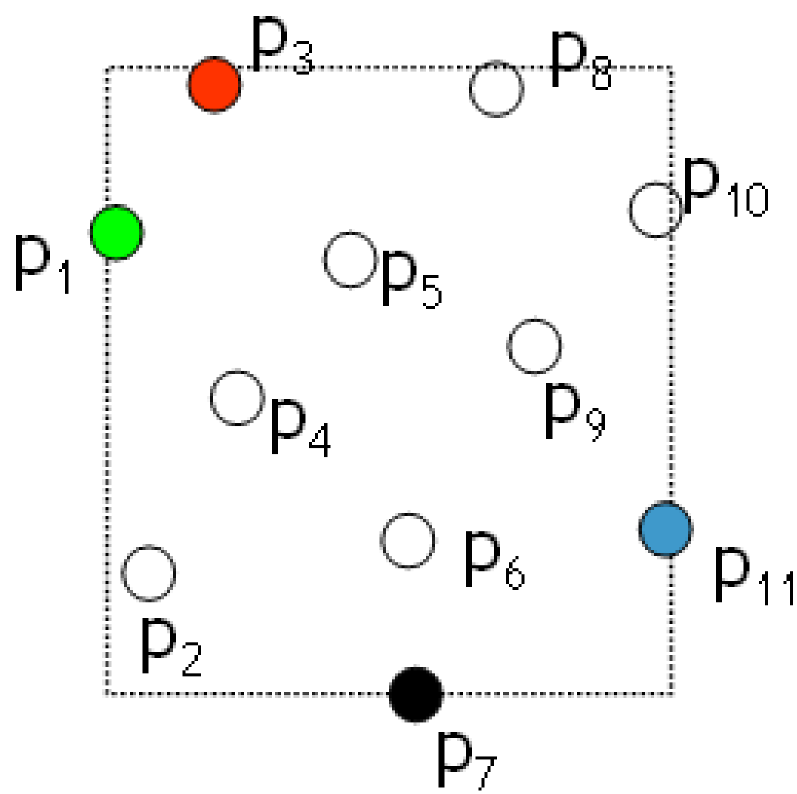
Convex-Hull Problem



Idea of quickhull

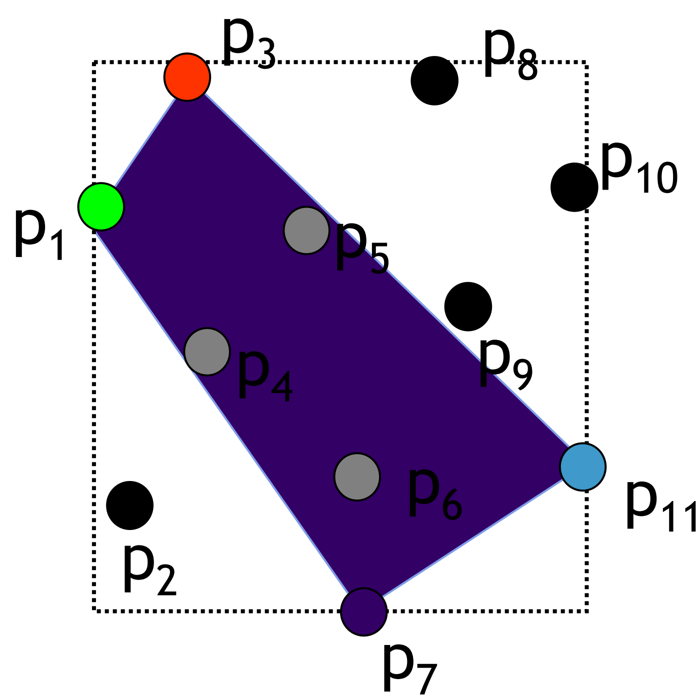
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



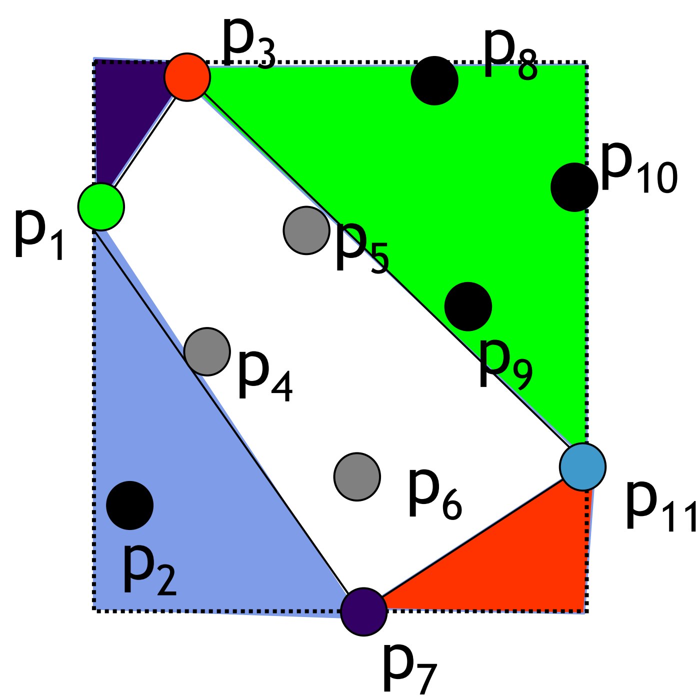
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points



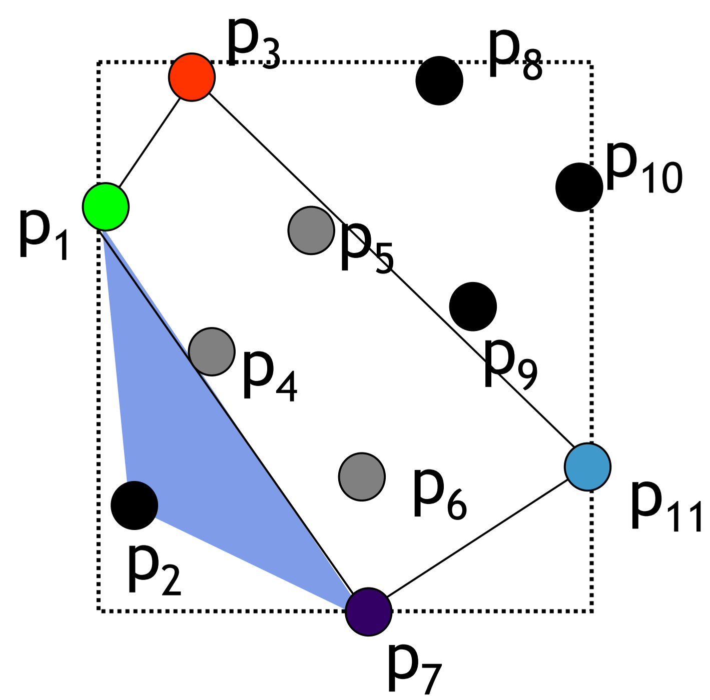
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively



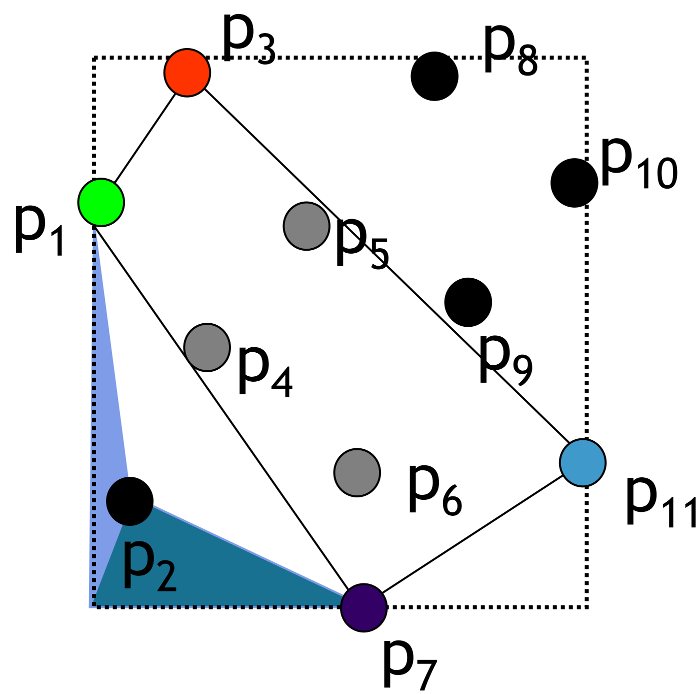
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles



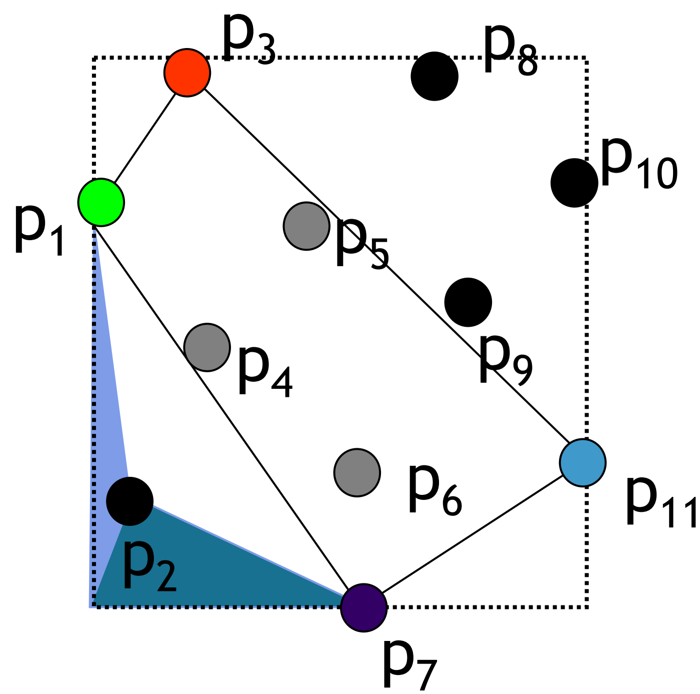
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



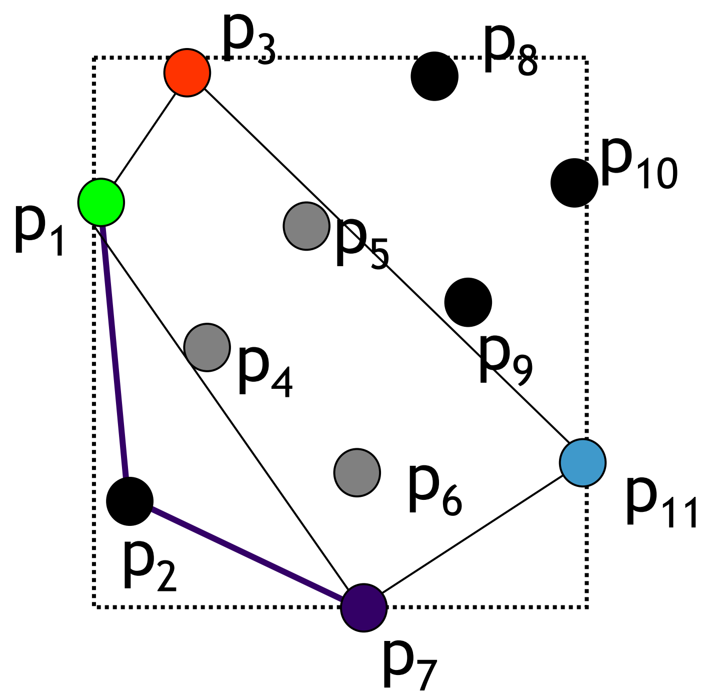
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



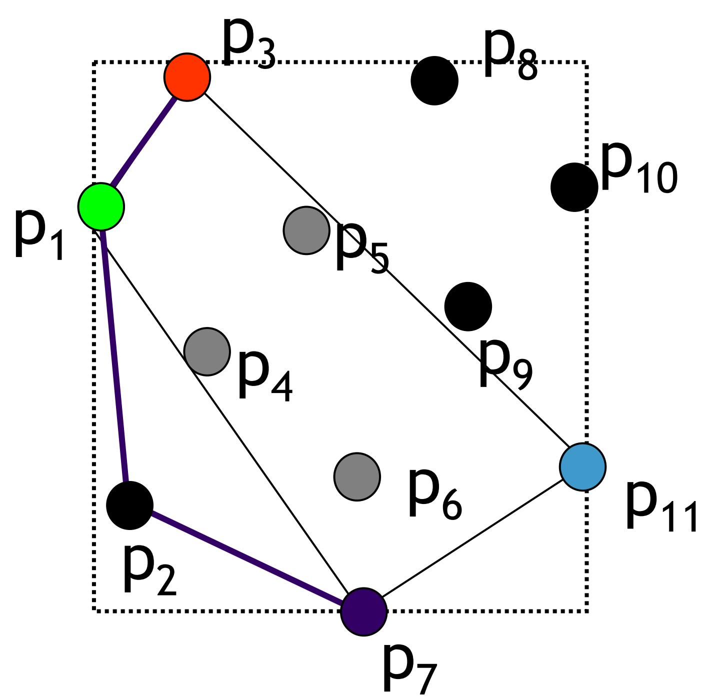
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



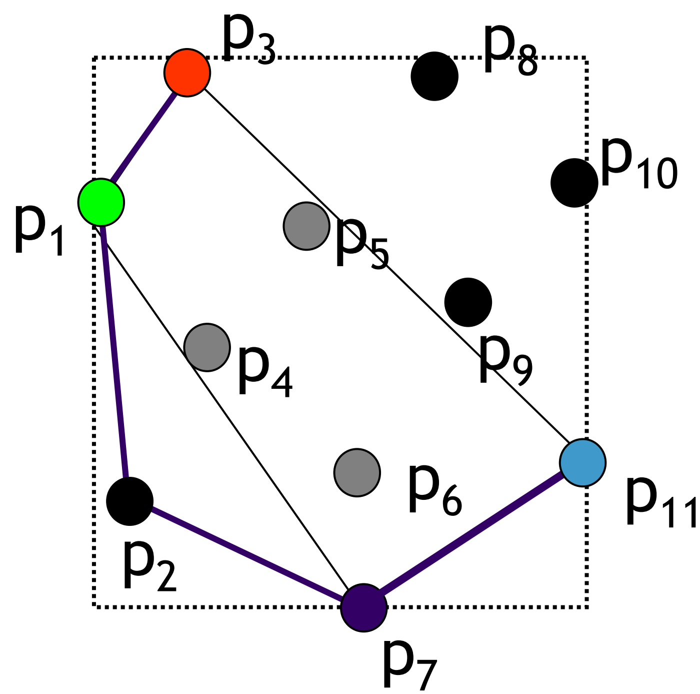
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



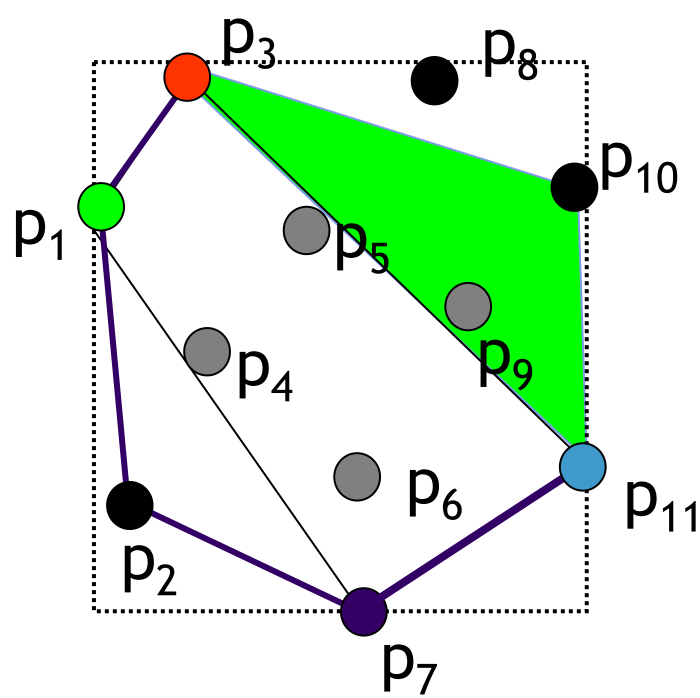
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



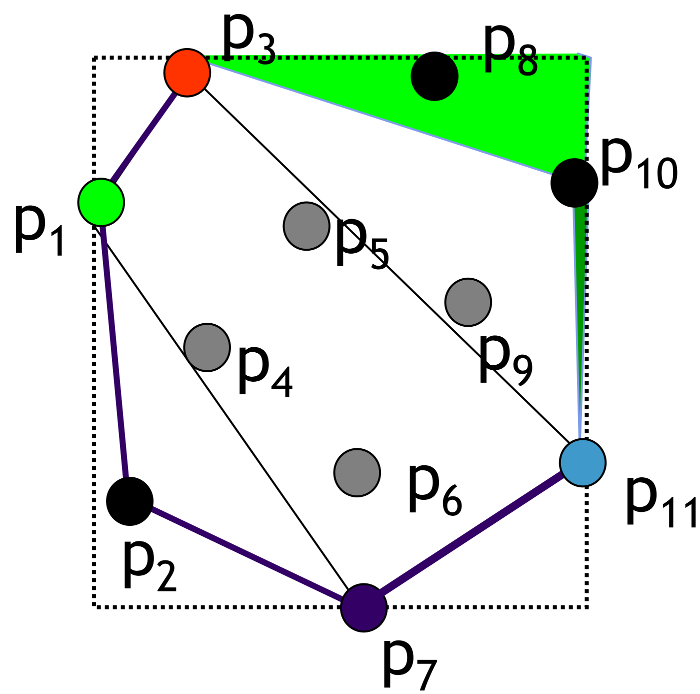
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



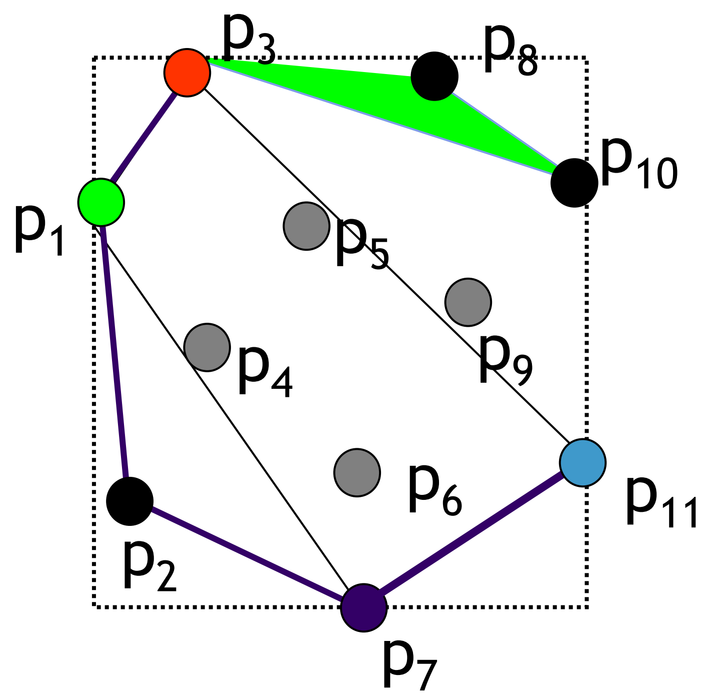
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



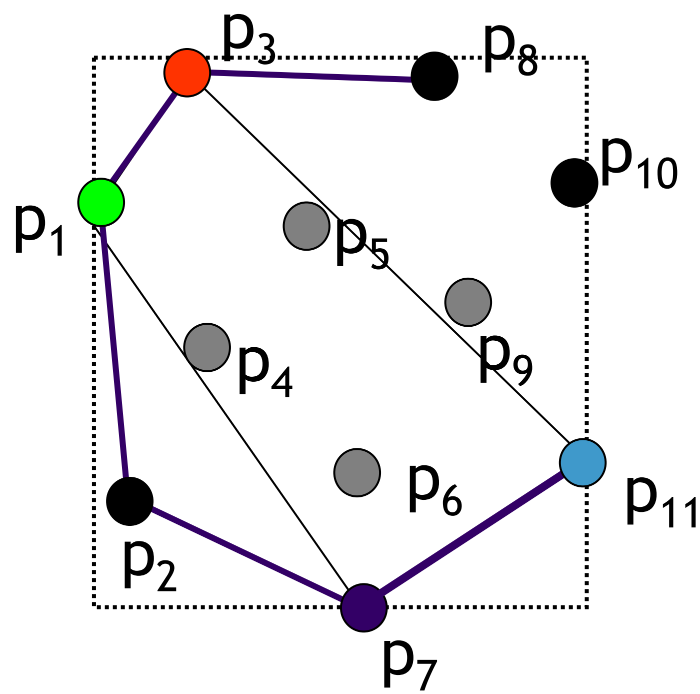
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



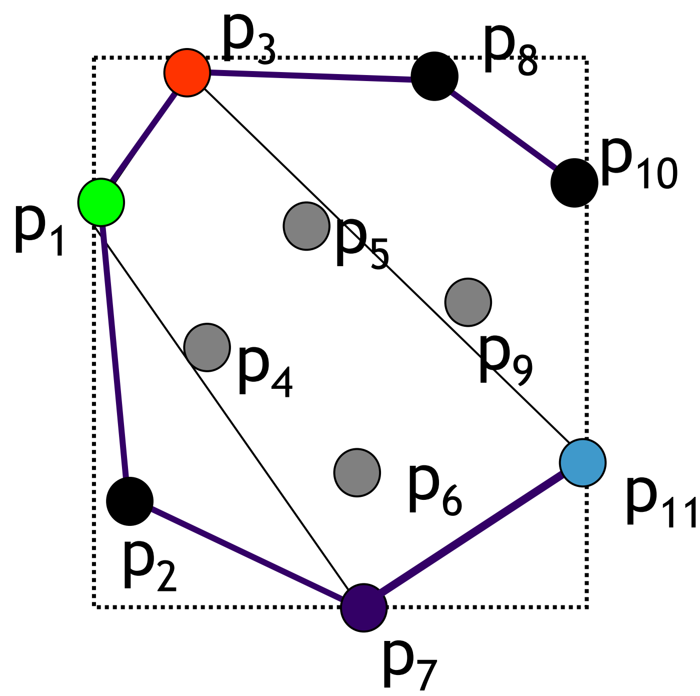
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



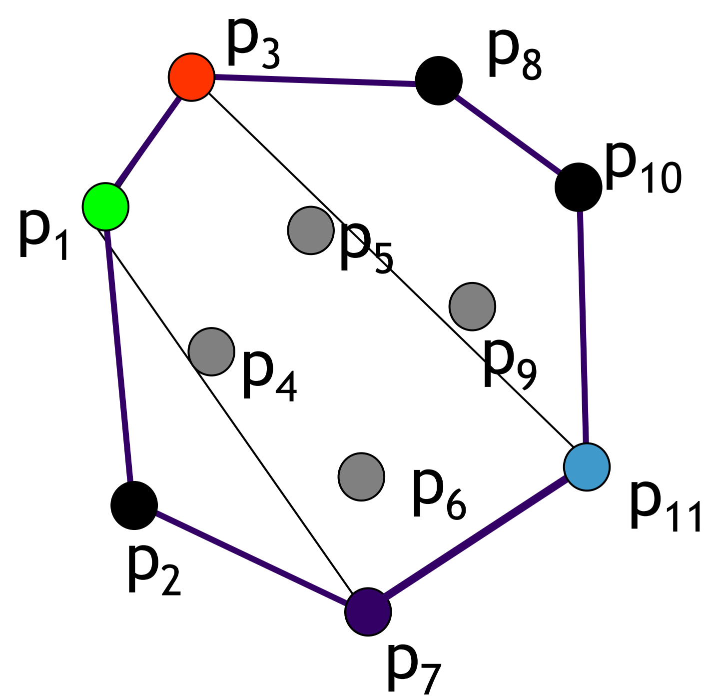
Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



Quickhull Example

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



Convex-Hull Problem

- **Analysis :**

  - Worst case efficiency is $\Theta(n^2)$
    - As quicksort

  - In average case we expect a better performance
    - Under a natural assumption that points given are chosen randomly from a uniform distribution over some convex region, average case turns to be **Θ(nlogn)**

Convex-Hull Problem

Discussion :

- Quickhull has the same efficiency with quicksort
  - Θ`(nlogn)` in average case but $\Theta(n^2)$ in worst case
- There exists more sophisticated d&c algorithms for this problem with the worst case efficiency Θ`(nlogn)`

Divide & Conquer

- **Discussion :**

  There are 3 criterias for efficiency of D&C algorithms

  - # of subproblems
  - Proportion of the main problem and subproblem
  - Time to divide the problem and combine the sub-solutions