Presented by Team 6

Geometric Algorithms 3D

KHTN2021

OUR TEAM

Trương Thanh Minh

Lê Châu Anh

TABLE OF CONTENTS

Contents

1

ELEMENTARY OPERATIONS

Chapter 1

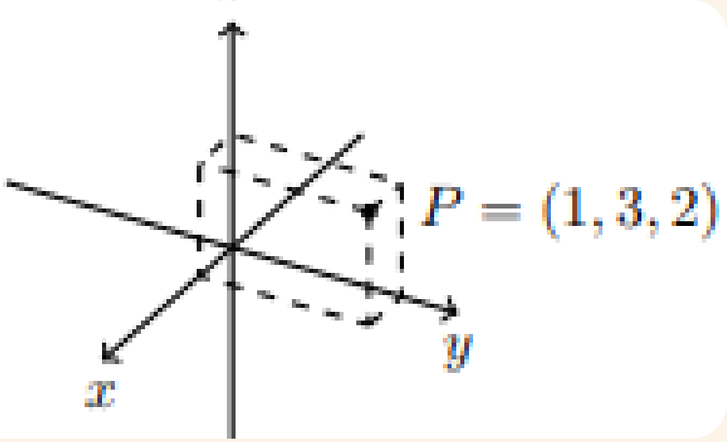
Chapter 1

1. POINT

1. POINT

- The point is represented in 3D space by its coordinates (x, y, z)

```python
class Point():
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
```



Chapter 1

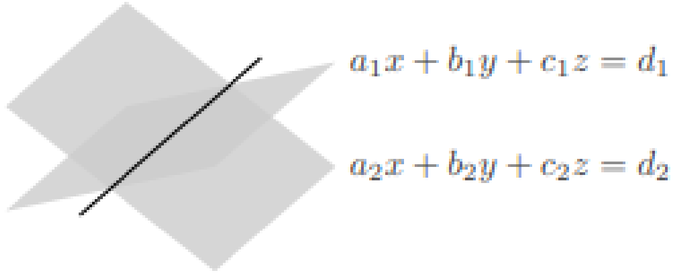
2. LINE

2. LINE

3D lines could be presented as the intersenction of two planes, like

$$a_1x + b_1y + c_1z = d_1$$

$$a_2x + b_2y + c_2z = d_2$$

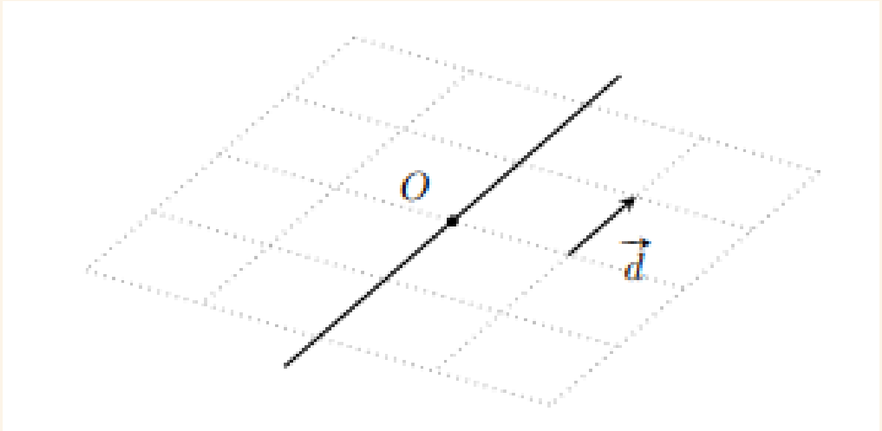


Nhưng cách biểu diễn này không thuận tiện lắm khi làm việc và có nhiều cặp mặt phẳng mà chúng ta có thể chọn.

2. LINE

Instead, work with a parametric representation: a point O on the line and a vector $\vec{d}$ parallel to the line and say that the points belonging to the line are all points (k is a real parameter)

$$P = O + k\vec{d}$$

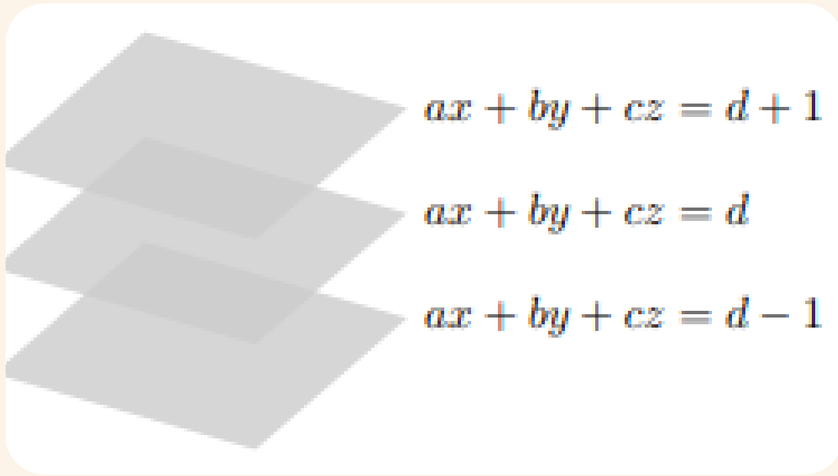


Chapter 1

3. PLANE

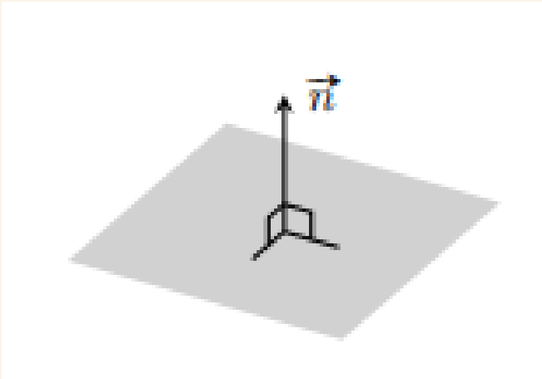
3. PLANES

- Planes are set of points (x, y, z) which obey an equation of the form ax+by+cz=d (a, b, c: orientation of the plane, d: position relative to the origin)



3. PLANES

- Vector $\vec{n} = (a, b, c)$ is perpendicular to the plane ans is called a normal of the plane.
- The equation can be rewritten as $\vec{n} \cdot (a, b, c) = d$: that is, the plane is formed by all points whose dot product with $\vec{n}$ = d



3. PLANES

Here are some other ways to define a plane, and how to find $\vec{n}$ and d in each case:

- know $\vec{n}$ and a point P belonging to the plane: can file d as $\vec{n} \cdot P$
- know a point P and tow (non-parallel) vectors $\vec{v}$ and $\vec{w}$ that are parallel to the plane: we can define $\vec{n} = \vec{v} \times \vec{w}$ the find d as above.
- know 3 non-colinear points P, Q, R on the plane: we first find 2 vectors $\vec{v} = \overrightarrow{PQ}$ and $\vec{w} = \overrightarrow{PR}$ that are parallel from the plane, then find $\vec{n}$ and d as above.

4. PRODUCTS, DISTANCES AND ANGLES

Chapter 1

4. PRODUCTS, DISTANCES AND ANGLES

4.1. PRODUCT

4.2. DISTANCES

4.3. ANGLES

Chapter 1

4.1. PRODUCTS

4.1.1 DOT PRODUCT

a. Coordinate definition

- The dot product of 2 vectors a = (a1, a2,a3) and b = (b1, b2,b3):

$$a.b = a_1.b_1 + a_2.b_2 + a_3.b_3$$

- If vector can be seen as a row matrix:

$$a.\,b = ab^T$$

4.1.1 DOT PRODUCT

a. Geometric definition

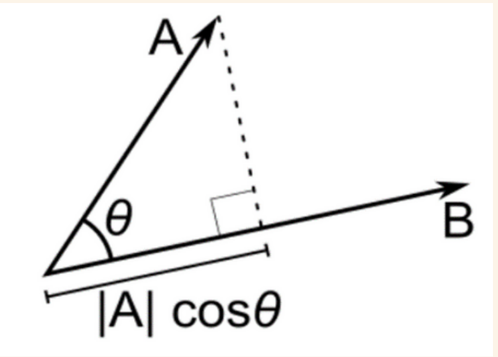
The dot product of 2 vectors a and b:

$$a.b = ||a||\,||b||cos(\theta)$$

Where:

- ||a|| and ||b|| are the lengths of vector a and b.
- $\theta \in [0, \pi]$ is the angle between the two vectors

In other words, the dot product of 2 vectors is the product of the length of one vector and the projection of another onto it.



4.1.1 DOT PRODUCT

c. Properties

- The length of vector a:

$$||a|| = \sqrt{a.a}$$

- The projection of vector a onto vector b:

$$p = \frac{a.b}{||b||}$$

- The angle between the two vectors:

$$\theta = arccos\left(\frac{a.b}{||a||\,||b||}\right)$$

4.1.2 CROSS PRODUCT

a. Coordinate definition

- The cross product of 2 vectors a = (a1, a2,a3) and b = (b1, b2,b3):

$$a \times b = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

- If vector can be seen as a row matrix:

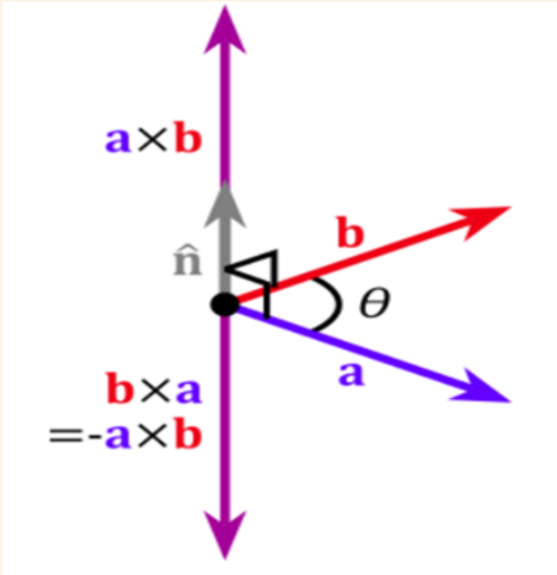
$$a \times b = \begin{matrix} (a_2b_3 - a_3b_2, \\ a_3b_1 - a_1b_3, \\ a_1b_2 - a_2b_1) \end{matrix}$$

4.1.2 CROSS PRODUCT

b. Geometric definition

- The cross product of 2 vectors a = (a1, a2,a3) and b = (b1, b2,b3):

$$a \times b = ||a||\,||b||sin(\theta)$$



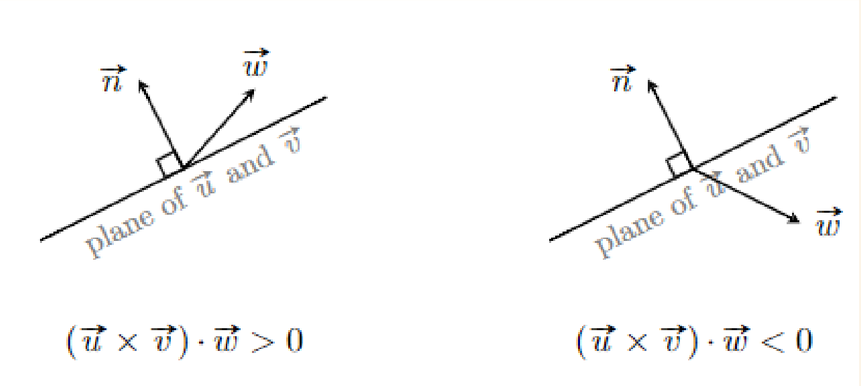
4.1.3 MIXED PRODUCT AND ORIENTATION

a. Mixed product

- The mixed product of 3 vectors a, b, c:

$$(a \times b) \cdot c$$

Let vector n = a x b is perpendicular to the plane containing a and b.

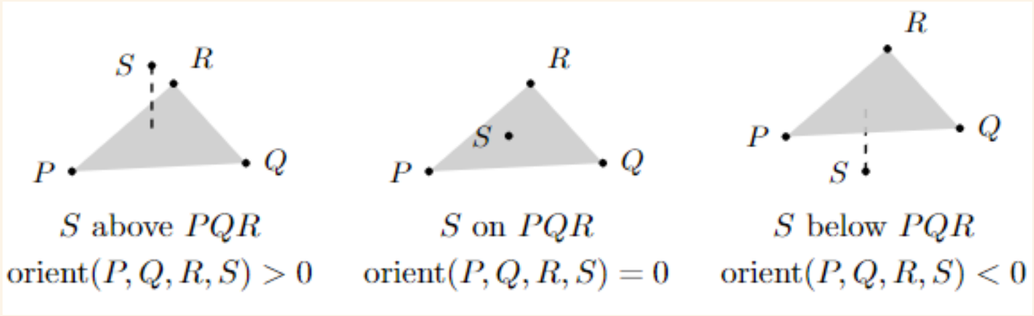


4.1.3 MIXED PRODUCT AND ORIENTATION

b. Orientation

- Given a point S, a plane PQR, we have a definition:

$$\text{orient}(P,Q,R,S) = \left(\overrightarrow{PQ} \times \overrightarrow{PR}\right) \cdot \overrightarrow{PS}$$



Chapter 1

4.2. DISTANCES

4.2.1 DISTANCE BETWEEN POINT AND PLANE

- Given a plane with equation Ax+By+Cz = D and a point P (x0,y0,z0). Distance from P to the plane:

$$\text{Distance} = \frac{|Ax_0 + By_0 + Cz_0 + D|}{\sqrt{A^2 + B^2 + C^2}}$$

- In an informal way, we have:

$$Distance = \frac{n \cdot P + D}{\|n\|}$$

Where: n is normal of the plane, n = (A, B, C)

Chapter 1

4.3. ANGLES

4.3.1 ANGLES BETWEEN A PLANE AND A LINE

- Given a plane (Q) and a line a. Angles between Q and a:

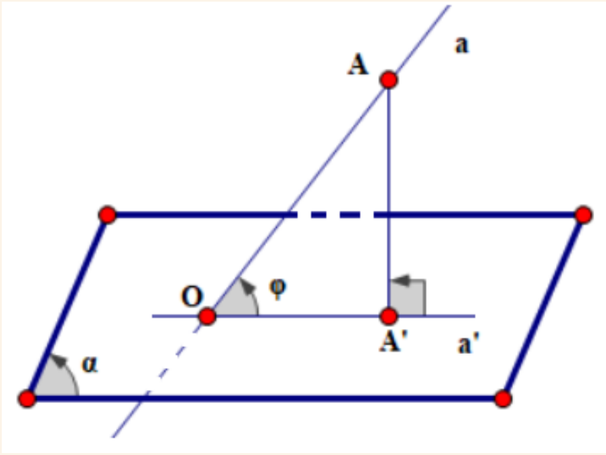
$$sin\varphi = sin(\widehat{a, (Q)}) = |\cos(\vec{n}; \vec{u})| = \frac{|\vec{u}.\vec{n}|}{|\vec{u}| \, . \, |\vec{n}|}$$

Where:

  - n is normal vector to the plane (Q).
  - u is direction vector to the line a.

- If vector n = (A,B,C) and vector u = (a,b,c) then:

$$sin\varphi = \frac{|A.a + B.b + C.c|}{\sqrt{A^2 + B^2 + C^2} \cdot \sqrt{a^2 + b^2 + c^2)}}$$



4.3.2 BETWEEN 2 LINES

- The angle between 2 lines is equal to the angle between their 2 direction vectors or 2 normal vectors:

$$cos\varphi = |cos(\vec{u}, \vec{n})| = \frac{|a_1a_2 + b_1b_2|}{\sqrt{a_1^2 + b_1^2} \cdot \sqrt{a_2^2 + b_2^2}}$$

The angle between two lines having slopes k1, and k2:

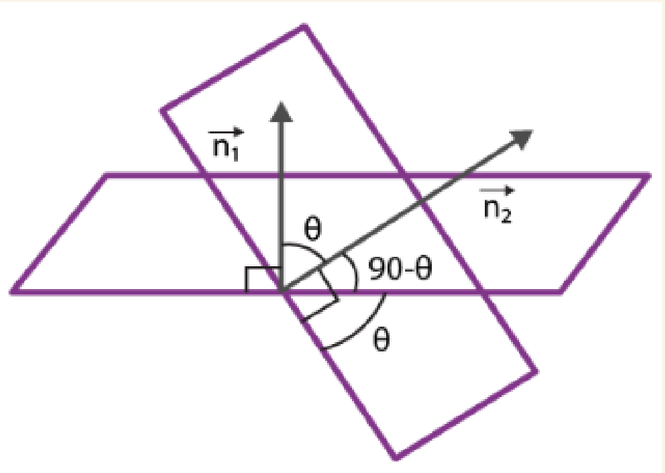
$$tan\varphi = \left| \frac{k_1 - k_2}{1 + k_1 \cdot k_2} \right|$$

4.3.3 BETWEEN 2 PLANES

- The angle between the two planes is equal to the angle between their normal:

$$cos\varphi = \left| \frac{\vec{n_1} \cdot \vec{n_2}}{|\vec{n_1}|\,|\vec{n_2}|} \right|$$

$$= \left| \frac{A_1A_2 + B_1B_2 + C_1C_2}{\sqrt{A_1^2 + B_1^2 + C_1^2} \cdot \sqrt{A_2^2 + B_2^2 + C_2^2}} \right|$$



Chapter 1

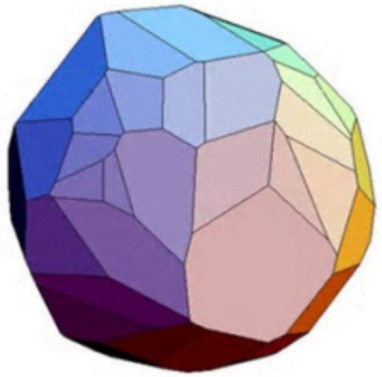
5. POLYHEDRON

5.1. POLYHEDRON

- Region of space whose boundary consists of vertices, edges and faces.
- Faces intersect properly
- Neighborhood of any point on P is homeomorphic to a disk
- Surface of P is connected.

5.2. CONVEXITY

- P is **convex** if for any p, q in P, the segment pq lies entirely in P.

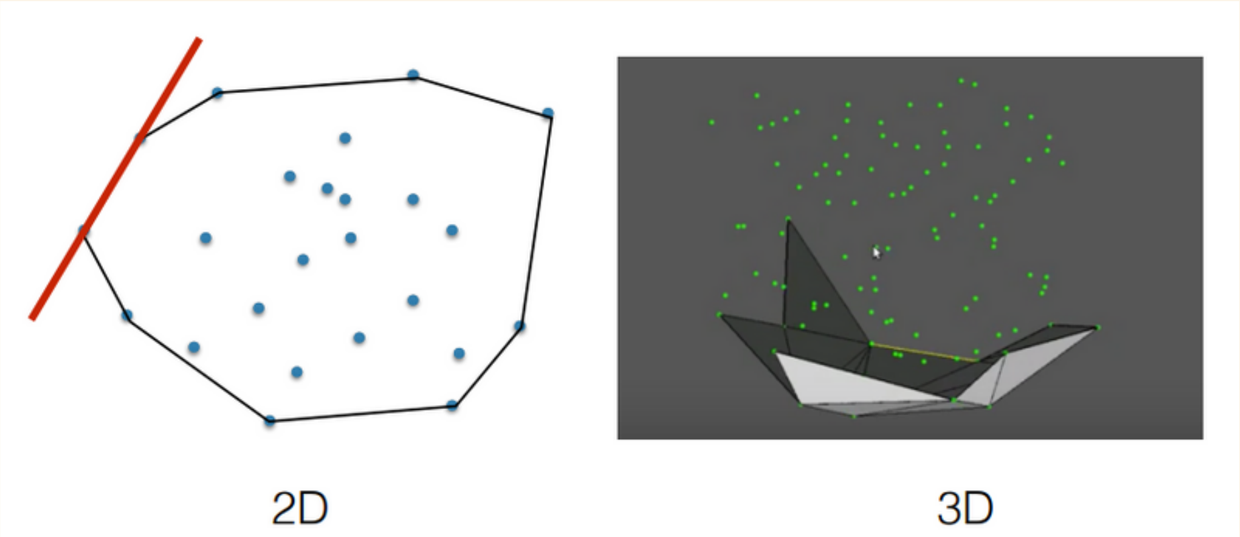


convex

non-convex

5.3. SOME PROPERTIES

- All faces of CH are extreme (and all extreme edges of P are on the CH)
- All internal angles between faces are < 180



2D 3D

II

ALGORITHMS

Chapter 2

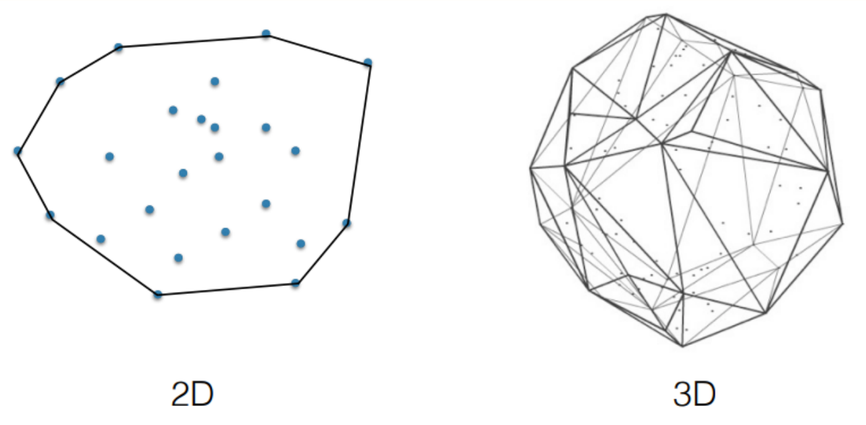
Chapter 1

1. CONVEX HULL 3D

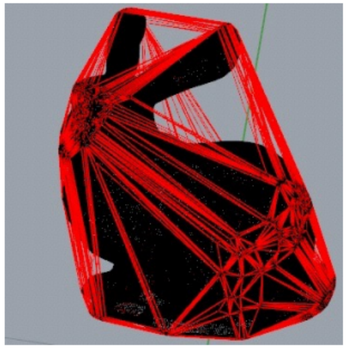
1.1. CONVEX HULL 3D

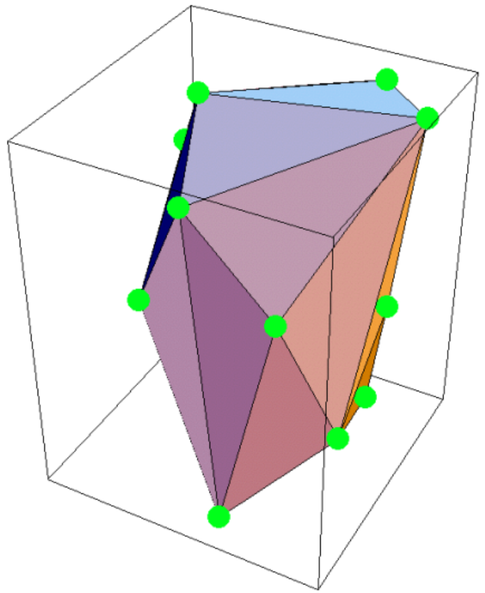
Problem statement

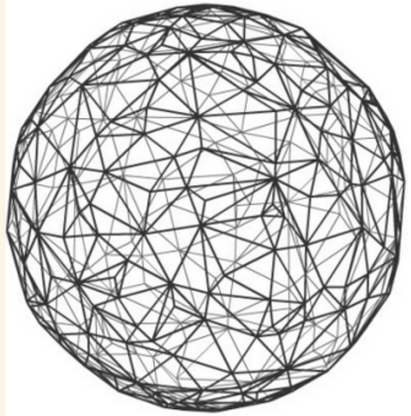
- Given P: set of n points in 3D
- Convex hull of P: CH(P), the smallest polyhedron s.t. all elements of P on or in the interior of CH(P)

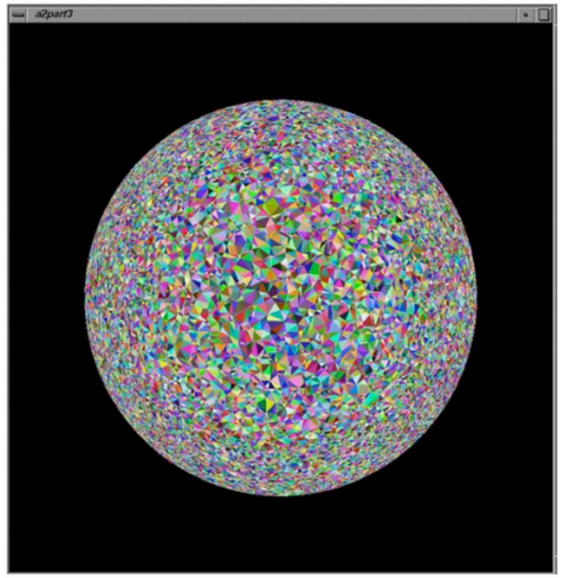


1.1. CONVEX HULL 3D









1.2. FROM 2D TO 3D

2D		3D
Naive	$O(n^3)$	$O(n^4)$
Gift wrapping	$O(nh)$	$O(n \times F)$
Graham scan	$O(n \lg n)$	no
Quickhull	$O(n \lg n)$, $O(n^2)$	yes
Incremental	$O(n \lg n)$	$O(n^2)$
Divide-and-conquer	$O(n \lg n)$	$O(n \lg n)$

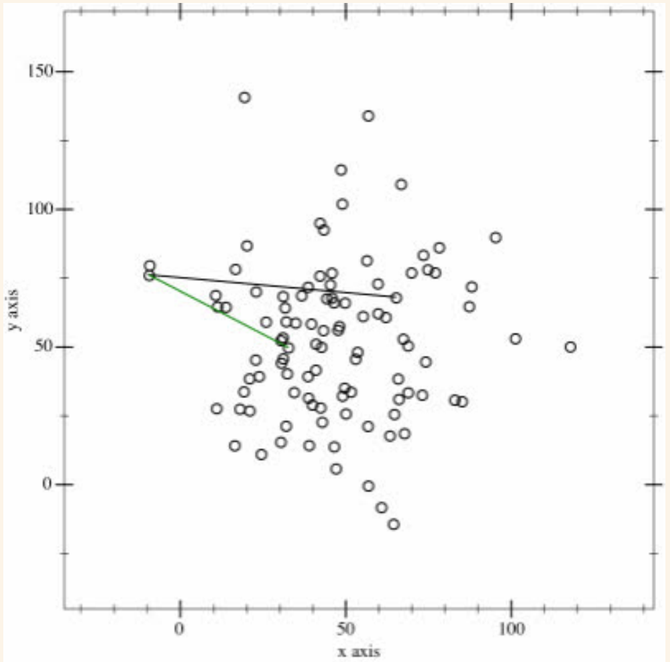
1.3. NAIVE ALGORITHM IN 3D

Algorithm

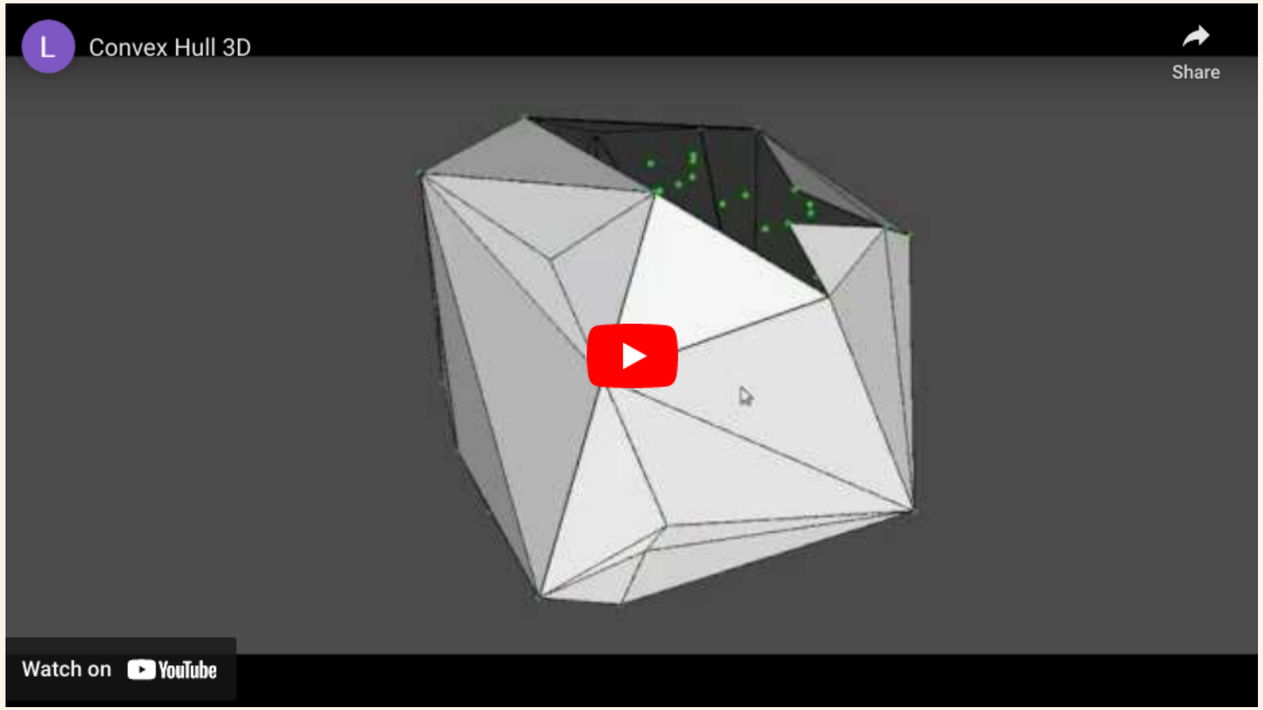
- For every triplet of points p_i, p_j, p_k
  - check if plane defined by it is extreme
  - if it is, add it to the list of CH faces
- Time complexity: $O(n^4)$

1.4. GIFT WRAPPING IN 3D

Ensure: H = Convex hull of point-set P
Require: point-set P
    $H = \emptyset$
    $(p_1, p_2) =$ HullEdge(P)
    $Q = \{(p_1, p_2)\}$
    while $Q \neq \emptyset$ **do**
        $(p_1, p_2) =$ Q.pop()
        if notProcessed((p_1, p_2)) **then**
            $p_3 =$ triangleVertexWithAllPointsToTheLeft(p_1, p_2)
            $H = H \cup \{(p_1, p_2, p_3)\}$
            $Q = Q \cup \{(p_2, p_3), (p_3, p_1)\}$
            markProcessedEdge((p_1, p_2))
        end if
    end while



1.4. GIFT WRAPPING IN 3D

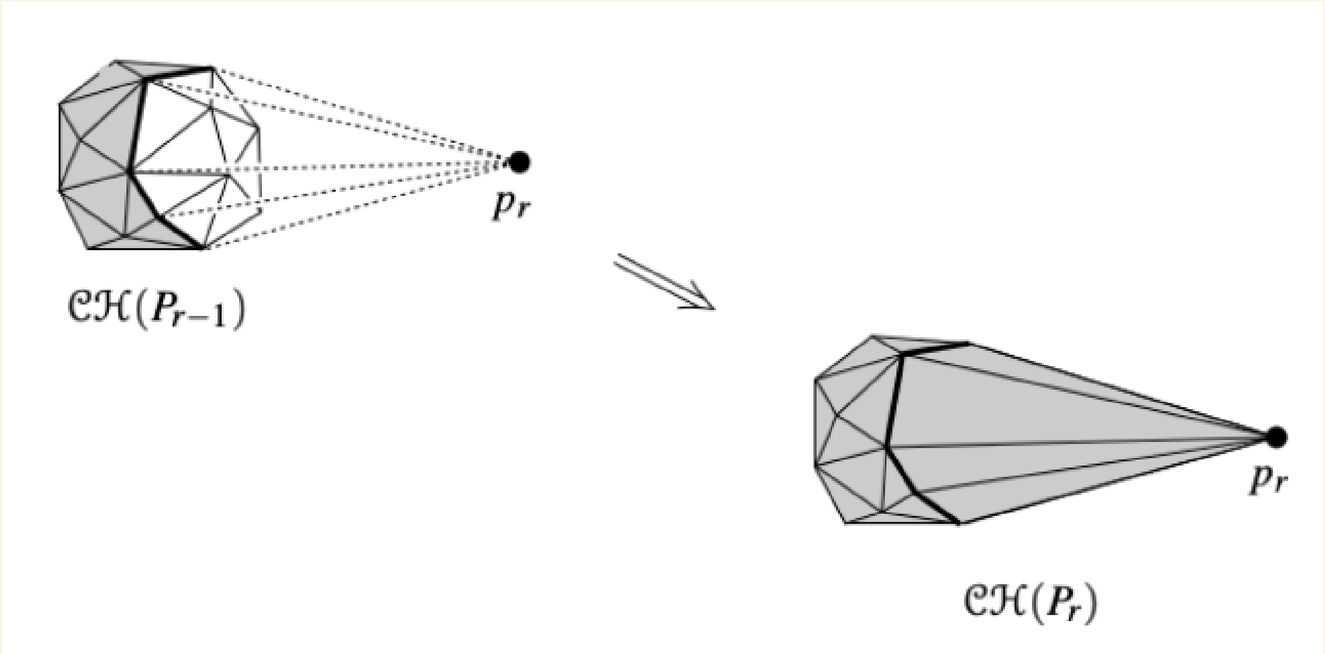


1.5. DIVIDE & CONQUER

- Too complex -> you can self-learning
- You can see here

1.6. INCREMENTAL 3D HULL

Incremental algorithm



1.6. INCREMENTAL 3D HULL

Incremental algorithm

1. Create tetrahedron(initial CH).
   - pick 2 points p_1 and p_2.
   - pick a point not lying on the line p_1p_2.
   - pick a point not lying on the plane $p_1p_2p_3$(if not found use a 2D algorithm).
2. Randomize the remaining points P.
3. For each $p_i \in P$, add p_i into the CH_{i-1}
   - if p_i lies inside or on the boundary of CH_{i-1} then do nothing.
   - if p_i lies outside of CH_{i-1} insert p_i.

1.6. INCREMENTAL 3D HULL

Incremental algorithm

Ensure: C Convex hull of point-set P
Require: point-set P
    $C = findInitialTetrahedron(P)$
    $P = P - C$
    for all $p \in P$ **do**
        if p outside C **then**
            $F =$ visbleFaces(C, p)
            $C = C - F$
            $C =$ connectBoundaryToPoint$(C,$ p$)$
        end if
    end for

1.6. INCREMENTAL 3D HULL

Find visible region

- Naive:
  - Test every face for each point $O(n^2)$.
- Conflict graph:
  - Maintain information to aid in determining visible facets from a point.

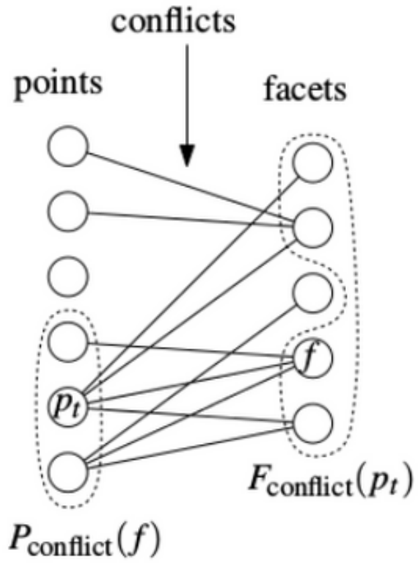
1.6. INCREMENTAL 3D HULL

Conflict graph

Relates unprocessed points with facets of CH they can see.

- Bipartite graph.
  - p_t unprocessed points.
  - f faces of the convex hull.
  - conflict arcs, point p_i sees f_j.
- Maintain sets:
  - $P_{conflict}(f)$, points that can see f.
  - $F_{conflict}(p)$, faces visible from p(visible region deleted after insertion of p).

At any step of the algorithm, we know all conflicts between the remaining points and facets on the current CH.

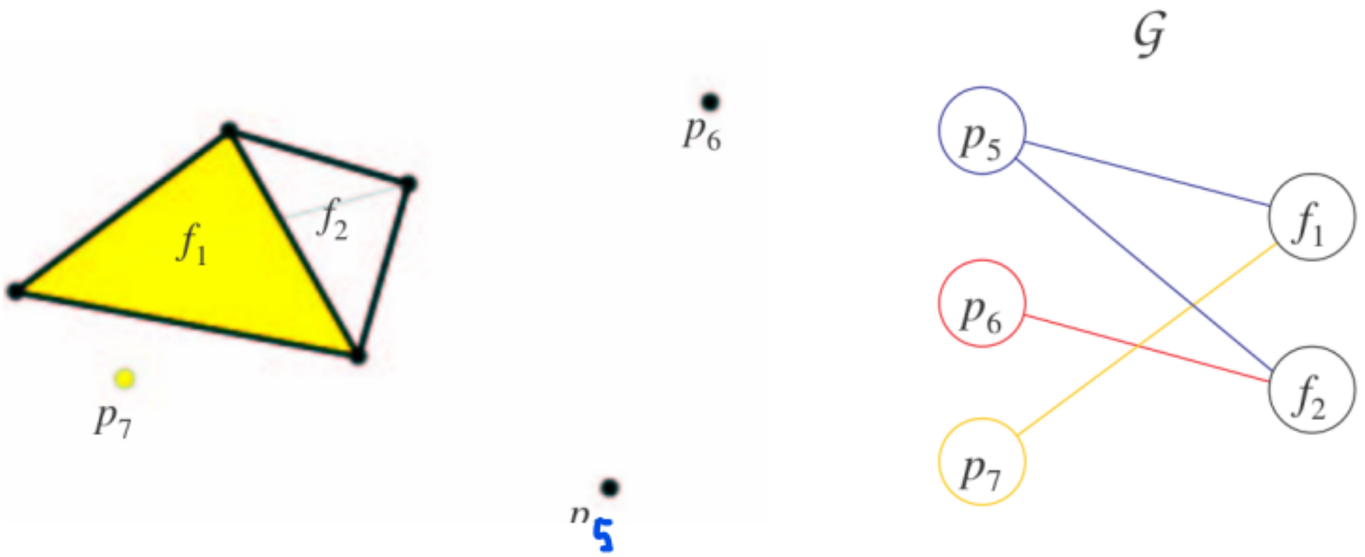


1.6. INCREMENTAL 3D HULL

Initialize conflict graph

Initialize the conflict graph G with $CH(P_4)$ in linear time.

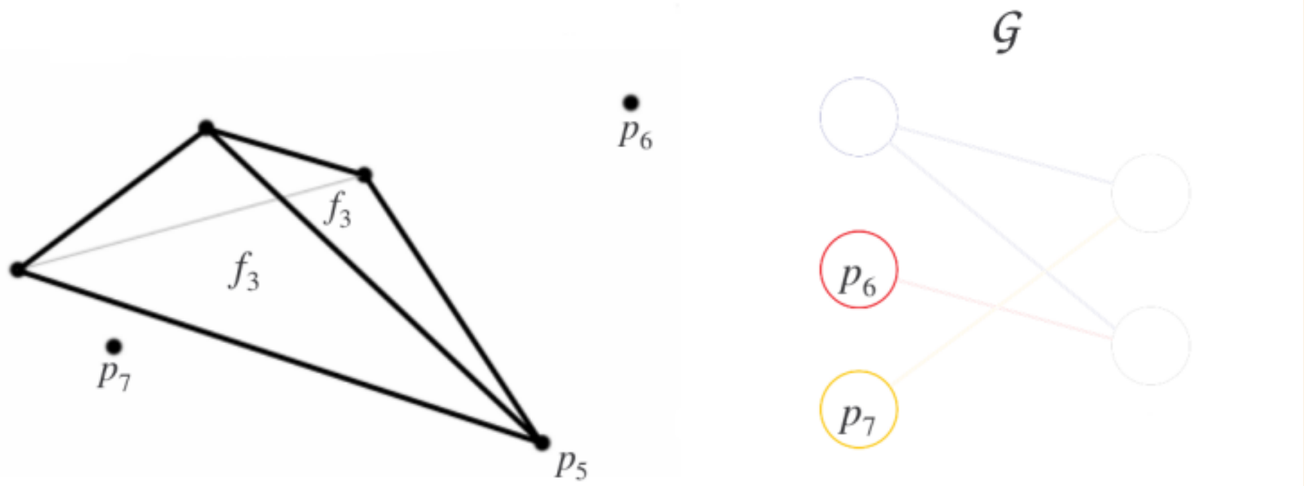
- Loop through all points to determine the conflicts.



1.6. INCREMENTAL 3D HULL

Update conflict graph

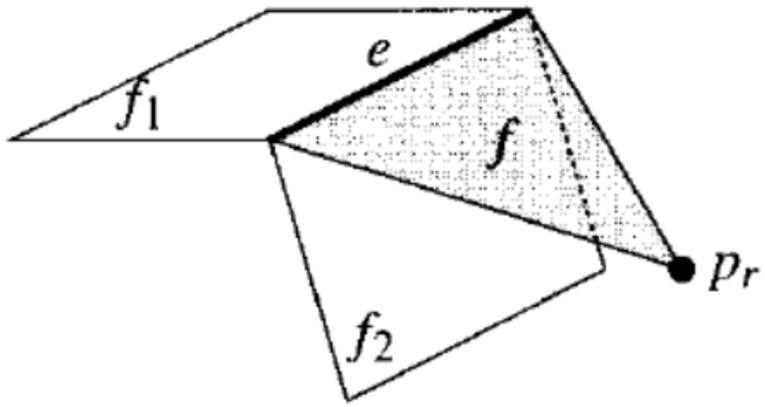
1. Discard visible facets from p by removing neighbors of $p \in G$.
2. Remove p from G.
3. Determine new conflicts.



1.6. INCREMENTAL 3D HULL

Determine new conflicts

- Check if $P_{conflict}(f_2)$ is in conflict with f.
- Check if $P_{conflict}(f_1)$ is in conflict with f.
- If p_r is coplanar with f_1, f has the same conflicts as f_1.



1.6. INCREMENTAL 3D HULL

Final incremental algorithm

Ensure: C Convex hull of point-set P
Require: point-set P
  $C = \mathit{findInitialTetrahedron}(P)$
  $\mathit{initializeConflictGraph}(P, C)$
  $P = P - C$
  for all $p \in P$ **do**
    if $F_{conflict}(p) \neq \emptyset$ **then**
      delete($F_{conflict}(p)$, C)
      $L =$ borderEdges(C)
      for all $e \in L$ **do**
        $f =$ createNewTriangularFace(e, p)
        $f' =$ neighbourFace(e)
        addFaceToG(f)
        if $\mathit{areCoplanar}(f, f')$ **then**
          $P_{conflict}(f) = P_{conflict}(f')$
        else
          $P(e) = P_{coflict}(f_1) \cup P_{coflict}(f_2)$
          for all $p \in P(e)$ **do**
            if $\mathit{isVisble}(f, p)$ **then**
              addConflict(p, f)
            end if
          end for
        end if
      end for
    end if
  end for

1.6. INCREMENTAL 3D HULL

TIme complexity

We want to bound the total number of facets created/deleted by the algorithm.

#faces

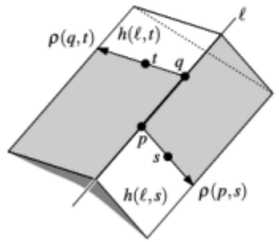
$$4 + \sum_{r=5}^{n} E|deg(p_r, CH(P_r))| \leq 4 + 6(n-4) = 6n - 20 = O(n)$$

1.6. INCREMENTAL 3D HULL

TIme complexity

We want to bound the total number of Points which can see horizon edges at any stage of the algorithm($\sum_e card(P(e))$).
Define:

- Set of active configurations $\tau(S)$.
- A flap $\Delta = (p, q, s, t)$.
- $\Delta \in \tau(S)$all edges in the flap are edges of the $CH(S)$.
- Set $K(\Delta)$ points which have the flap Δ as horizon edge.



1.6. INCREMENTAL 3D HULL

TIme complexity

We want to bound the total number of Points which can see horizon edges at any stage of the algorithm $\sum_e card(P(e))).$

$$\sum_e card(P(e)) \leq \sum_\Delta card(K(\Delta)) \leq \sum_{r=1}^{n} 16\left(\frac{n-r}{r}\right)\left(\frac{6r-12}{r}\right) \leq 96nlogn =$$

$$O(nlogn)$$

Chapter 1

2. CLOSEST PAIR OF POINTS 3D

2.1 CLOSEST PAIR OF POINTS 2D

a. Algorithm

Input: An array of n points P[]

Output: The smallest distance between two points in the given array.

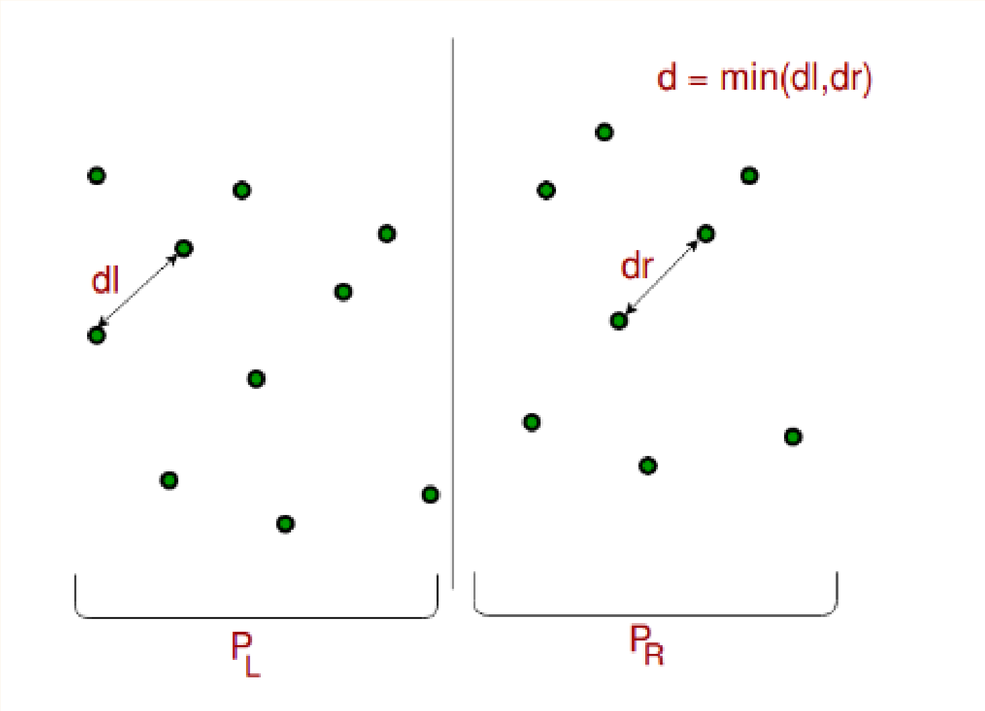
As a pre-processing step, the input array is sorted according to x coordinates.

1. Find the middle point in the sorted array, we can take P[n/2] as middle point.
2. Divide the given array in two halves. The first subarray contains points from P[0] to P[n/2]. The second subarray contains points from P[n/2+1] to P[n-1].

2.1 CLOSEST PAIR OF POINTS 2D

a. Algorithm

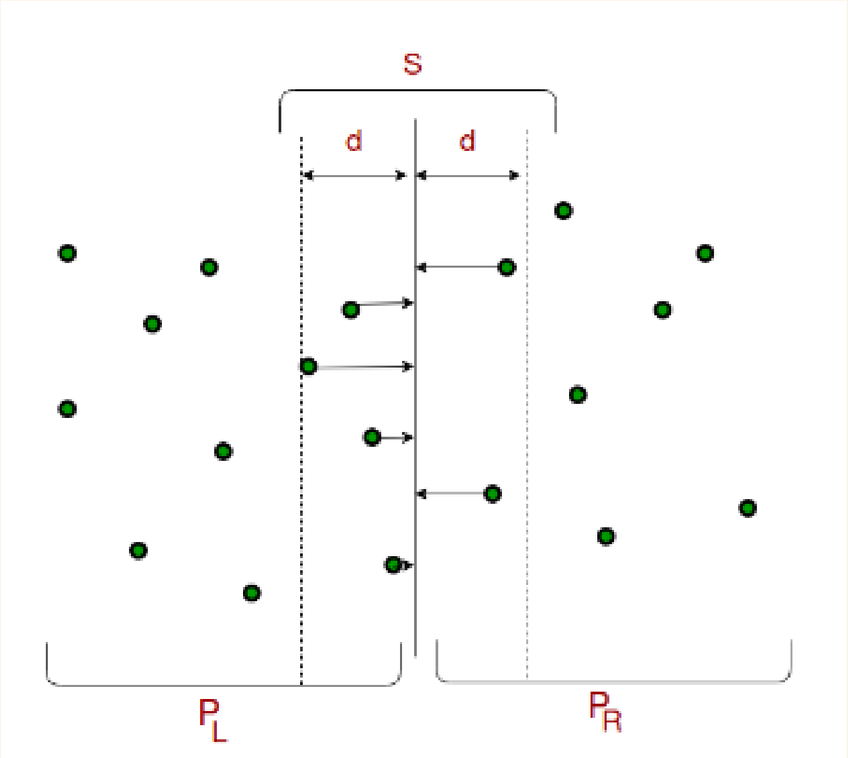
3. Recursively find the smallest distances in both subarrays. Let the distances be dl and dr. Find the minimum of dl and dr. Let the minimum be d.



2.1 CLOSEST PAIR OF POINTS 2D

a. Algorithm

Consider the pairs such that one point in pair is from the left half and the other is from the right half. Build an array strip[] of all such points.



2.1 CLOSEST PAIR OF POINTS 2D

a. Algorithm

5. Sort the array strip[] according to y coordinates.

6. Find the smallest distance in strip[]. It seems to be a O(n^2) step, but it is actually O(n). It can be proved that for every point in the strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate).

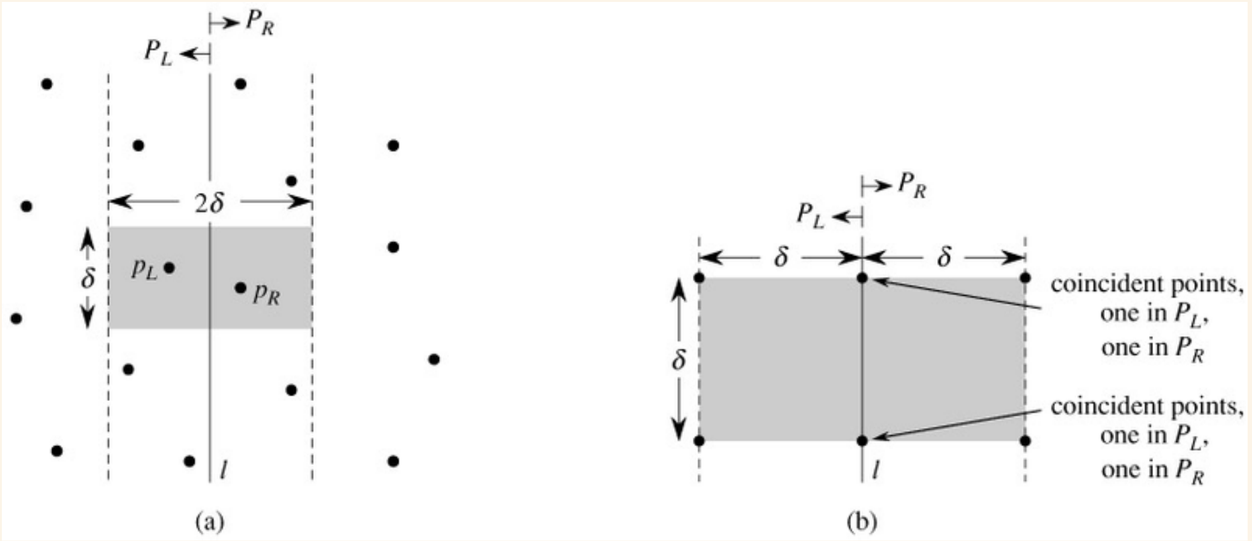
7. Finally return the minimum of d and distance calculated in the above step (step 6)

2.1 CLOSEST PAIR OF POINTS 2D

b. Analysis

For every point in the strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate).

- Sort coordinate: 0y
- Projection line/plane: 0x
- Bounding box: 2d x d



2.1 CLOSEST PAIR OF POINTS 2D

c. Time complexity

Let Time complexity of above algorithm be T(n). Assume that we use a O(nLogn) sorting algorithm. The above algorithm divides all points in two sets and recursively calls for two sets. After dividing, it finds the strip in O(n) time, sorts the strip in O(nLogn) time and finally finds the closest pair of points in strip in O(n) time. So T(n) can expressed as follows

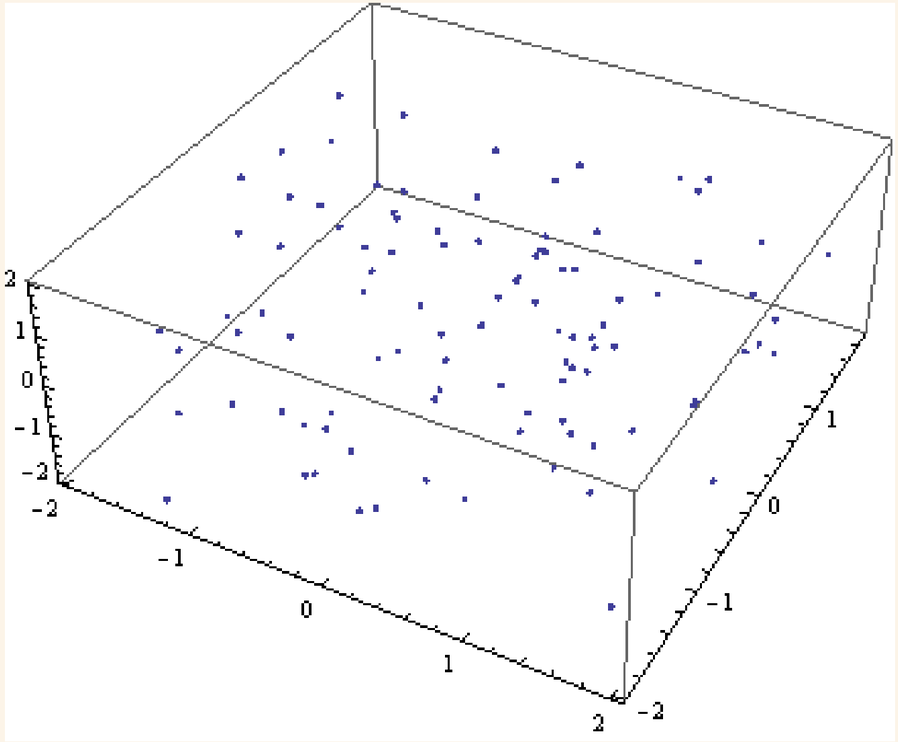
$T(n) = 2T(n/2) + O(n) + O(nLogn) + O(n)$

$T(n) = 2T(n/2) + O(nLogn)$

2.2 CLOSEST PAIR OF POINTS 3D

b. Algorithm

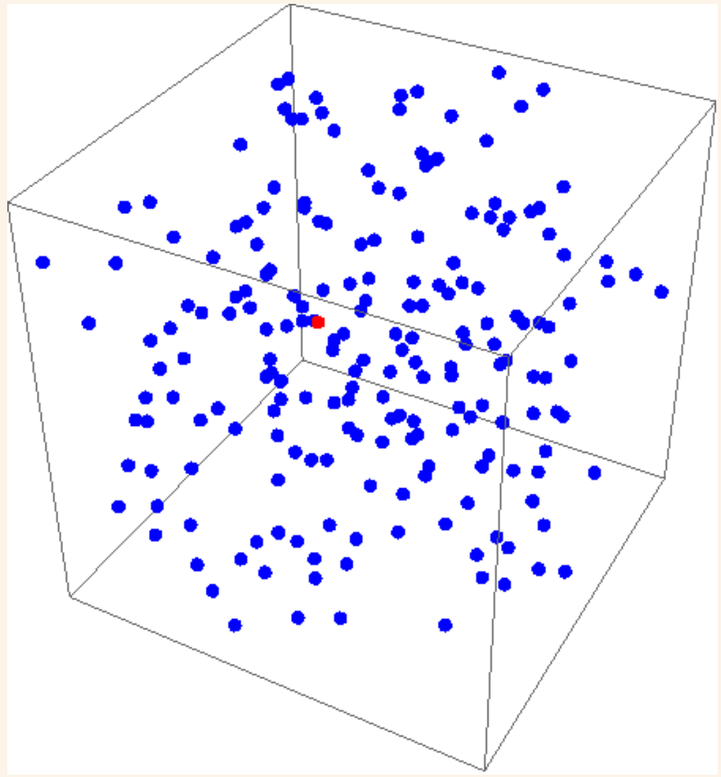
- Similar to 2D but there is a slight change.
- There are 3 coordinates x,y,z.
- Point p(x0,y0,z0)



2.2 CLOSEST PAIR OF POINTS 3D

b. Analysis

- Sort coordinates: Oy, Oz
- Projection plane: y = −z + c
- Bounding box: 2d x d x d
- Cube: d/2 x d/2 x d/2
- Number of cube: (2d^3) / ((d/2)^3) = 16
- Maximum points: 16 – 1 = 15



2.2 CLOSEST PAIR OF POINTS 3D

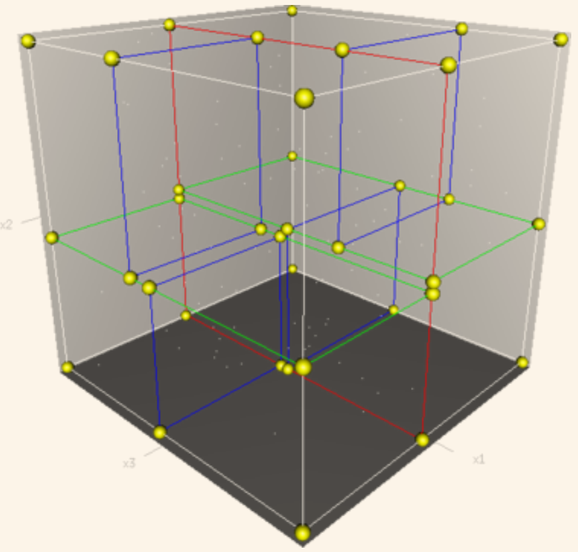
c. References

- Create an octree or a k-d tree, with all the points and then use it to find the nearest point for every point. That is O(N*log N) for the average case.
- References:

https://github.com/ashleshmahule/ClosestPairProblem
https://blog.krum.io/k-d-trees/

- Closest pair of points in d-dimensions

https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf

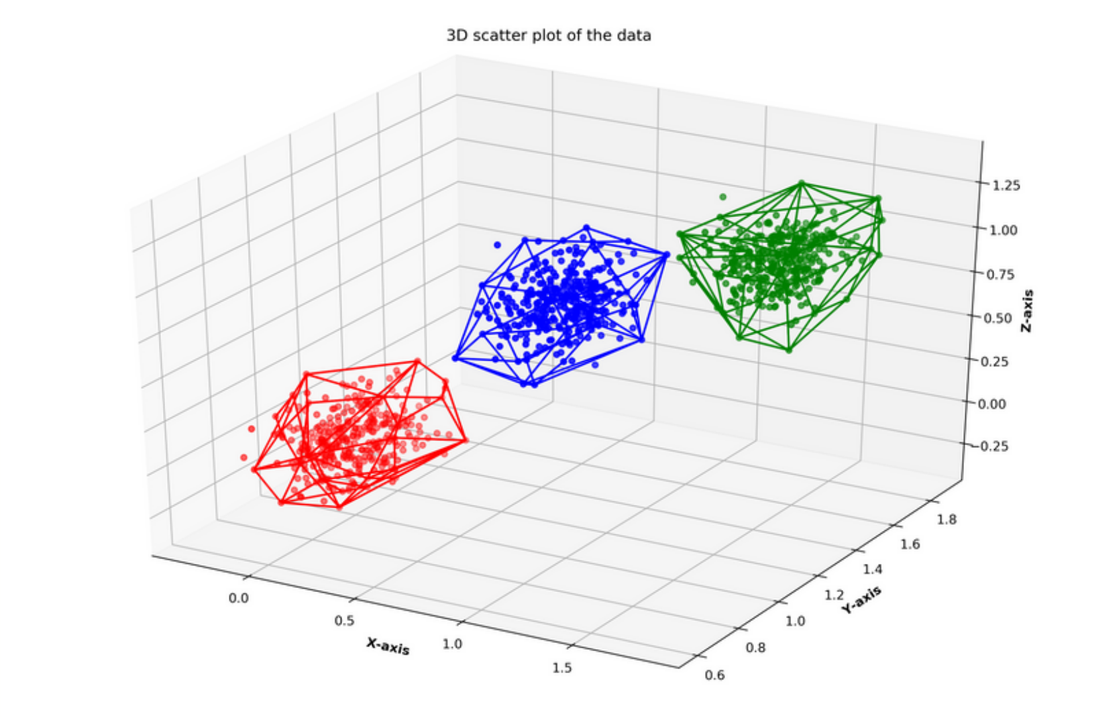


III

APPLICATIONS

Chapter 3

CLUSTERING


3D scatter plot of the data
X-axis
Y-axis
Z-axis
0.0
0.5
1.0
1.5
0.6
0.8
1.0
1.2
1.4
1.6
1.8
1.25
1.00
0.75
0.50
0.25
0.00
0.25

ARCHITECTURE

Three dimensional geometry plays an important role in deciding the shapes, their sizes and all strategic and technical decisions while constructing building, bridges and any other constructions.

APPLICATION IN MEDICINE

3D geometry is used in figuring out various decisions taken by doctors regarding body surgeries, treatments and manufacture of implants and prosthetics. Considerations of volumes, areas and lengths in prosthetics as per the requirement of human body are done by using three dimensional geometry.

TECHNOLOGY

Three dimensional geometry has wide range of applications in computer software.

Eg: Video games have structures of buildings and other objects which are designed using three dimensional geometry.

OTHER APPLICATIONS

https://www.inf.ed.ac.uk/teaching/courses/cl1/slides/14_out.pdf

IV

QUIZ AND HOMEWORKS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse quis enim pretium, bibendum ante ullamcorper, tincidunt augue.

Chapter 4

HOMEWORKS

Do your homeworks and upload in your github

1. Code convex hull 3D and submit in spoj . Write report and capture screen your submit AC in your report and write your solution.
2. Solve this problem in leetcode and capture your screen your submit AC in your report

Thank You

By Team 6