# ENVELOPPES CONVEXES

Mountaz Hascoet, Univ. Montpellier

# PLAN

- Motivation
- Définitions et propriété de base
- Algorithmes de construction en dimension 2
  - Algorithme des demi-plans
  - Algorithme de Jarvis
  - Algorithme de Graham
- Exercices pratiques

# MOTIVATION

- Nombreuses applications
  - Simplifier un problème
    - En infographie
      - Détection de collision
      - Détermination des faces visibles/cachées
    - Dans les autres domaines
      - Programmation linéaire
- Objectifs
  - Maitriser quelques algorithmes assez différents en dimension 2
    - adaptés à des contexte et des styles de programmation différents
  - Les expérimenter en pratique
    - Défi 1 -> implémenter trois algorithmes, les tester sur des cas d'école
    - Défi 2 -> étudier les avantages et les limites de chaque algorithme

# DÉFINITIONS ET PROPRIÉTÉS DE BASE

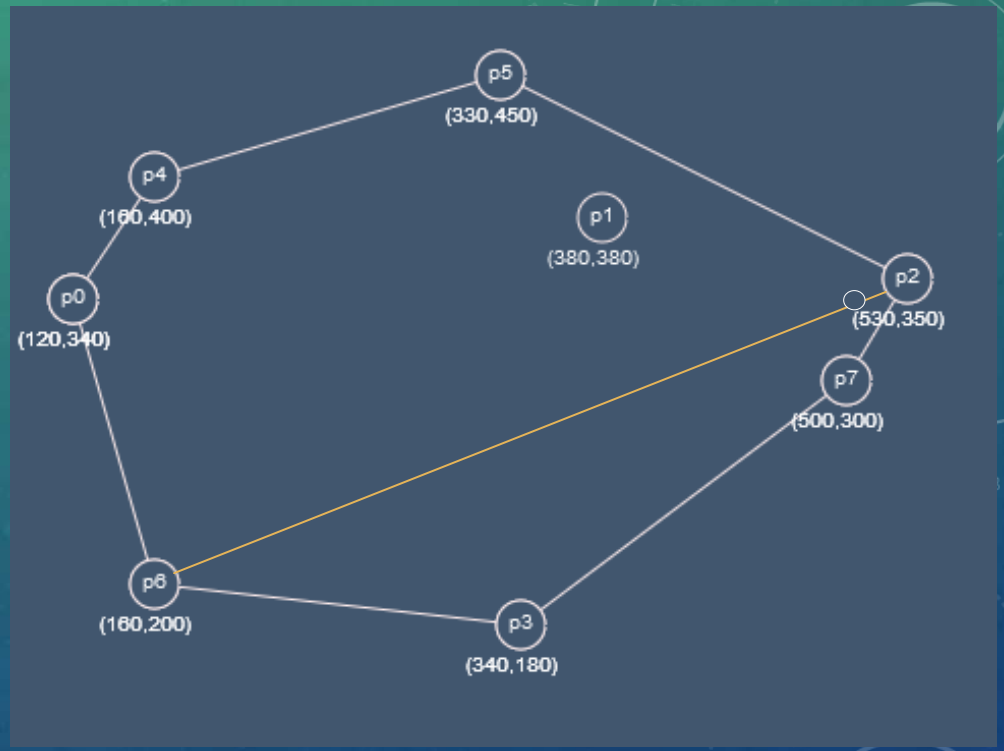
# CONVEXITÉ

- Ensemble convexe

  Un ensemble V de points est dit convexe lorsque pour tout couple (A,B) de V, pour tout point C du segment [A,B], C appartient à V

- Enveloppe convexe

  En dimension 2, l'enveloppe convexe d'un ensemble de points V est le plus petit polygone contenant V



# PROPRIÉTÉ EN DIMENSION 2

- Intersection de demi-plans
  - La région délimitée par une enveloppe convexe est l'intersection des demi-plans délimités par ces droites
  - Les segments qui forment l'enveloppe convexe d'un ensemble V de sommet du plan sont portés par des droites qui laissent tous les points de V du même côté.

# TROIS ALGORITHMES EN DIMENSION 2

DEMI-PLAN, JARVIS, GRAHAM

# ALGORITHME 1 – INTERSECTION DE DEMI-PLANS

# ALGORITHME 1 – ALGO DES DEMI-PLANS

- Principe
  - Chaque segment [A,B] de sommets de V est examiné
    - [A,B] fait partie de l'enveloppe convexe ⇔ tous les sommets de V (différents de A,B) sont du même côté de [A,B]
- Pour un ensemble de n sommets
  - n – 2 tests par segments et n (n-1)/2 segments à tester
  - => complexité en $O(n^3)$

# APPLICATION SUR UN ENSEMBLE DE POINTS

# ORIENTATION DANS LE PLAN

- Comment déterminer l'orientation d'un triangle ou de trois points du plan?
  - input

    trois points a, b et c
  - output

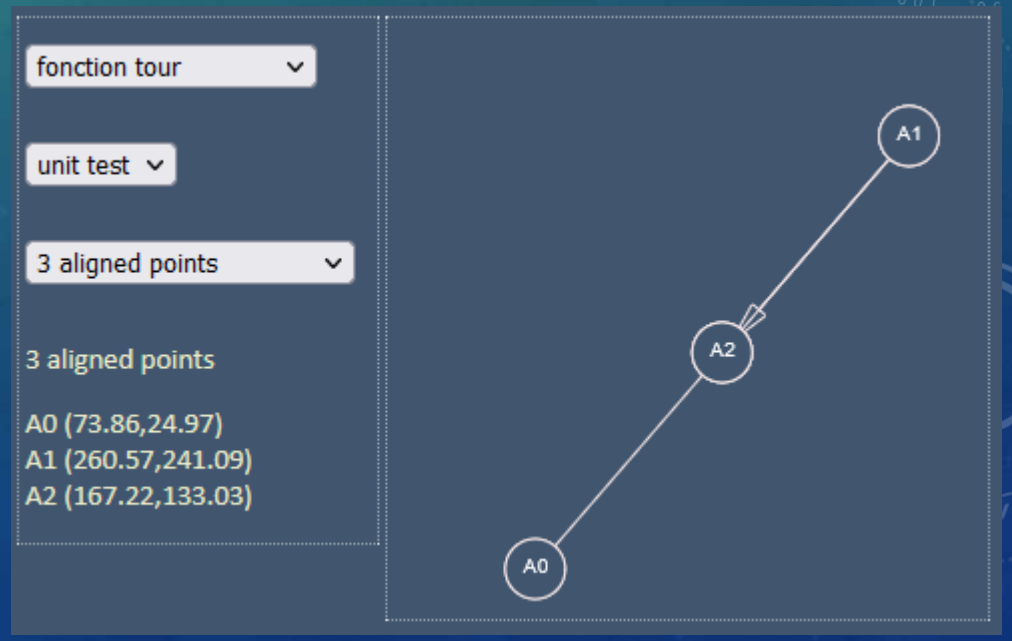
    +1 lorsque a, b, c forment un tour gauche,

    -1 lorsque a, b, c forment un tour droit et

    0 lorsque a, b, c sont alignés.
  - stratégie

    sans utiliser les fonctions de trigo

    utiliser le déterminant de deux vecteurs
- Remarque:
  - dans leur version de base les algorithmes ne gèrent pas les cas où plus de 2 sommets se trouvent sur un même droite (sommets alignés).



# ALGO DEMI PLAN

- input
  - un ensemble V de n sommets notés $v_i$ avec $i \in [0, n-1]$
- output
  - le tableau de paires *d'indices des extrémités des segments* de V qui forment son enveloppe convexe.
- prérequis
  - `tour`
  - `findFirst`
    - retourne l'indice du premier point de V différent de V[i] et V[j].
- complexité
  - $O(n^3)$

```
1  //initialisations
2        aop result = Ø                                          //aop pour a-rray o-f p-ai
3        int n = taille(V)
4        int current, previous, i, j, k, f
5        bool mc                                                 // mc pour m-ême c-ôté
6
7  //début
8  pour les couples (i, j) de 0 à n – 1 avec i < j
9        mc = true
10       f = findFirst(V, i, j)
11       previous = tour (V[i],V[j],V[f])
12       k = f + 1
13       faire
14             si (k ≠ i && k ≠ j)
15                   current = tour(V[i], V[j], V[k])
16                   si (current == 0)                           // exception points aligné
17                         exception ( "alignement", [V[i], V[j], V[k]])
18                   sinon si (current ≠ previous)
19                         mc = false
20                   fin si
21             fin si
22             k++
23       tant que (k < n && mc)
24
25       si (k == n && previous == current)
26             si (current > 0) result.add(i, j)
27             si (current < 0) result.add(j, i)
28             si (current == 0) exception ( "alignement", [V[i], V[j], V[k]])
29       fin si
30 fin pour
```

# ALGORITHME 2 – ALGORITHME DE GRAHAM

# ALGORITHME DE JARVIS

- Principe
  - Exploiter la relation entre deux sommets voisins de l'enveloppe convexe
  - Trouver un premier sommet trivial de l'enveloppe et construire les autres de proche en proche
- Pour un ensemble de n sommets
  - Trouver le premier sommet de l'enveloppe convexe demande n tests
  - A partir d'un sommet, trouver le suivant peut demander jusqu'à n tests dans le pire des cas
  - => complexité en $O(n^2)$
    - Ou O(kn) où k est le nombre de sommets de l'enveloppe convexe. En pratique, on trouve des cas où k << n et des cas où k = n. Pour cet algorithme, la complexité est donc très sensible à la nature des "input".

# ALGO JARVIS

- Input
  - un ensemble V de n sommets notés $v_i$ avec i ∈ [0, n-1]
- Output
  - le tableau des indices des sommets de V qui forment son enveloppe convexe.
- Prérequis
  - `tour` déjà décrite,
  - `minY`
  - `findNext`

```
1   //initialisations
2       min = minY(V)
3       result = [min]
4       current = undefined
5       previous = min
6
7   //début
8       faire
9           current = findNext(previous)
10          result.add(current)
11          previous = current
12      tant que (current ≠ min)
13
14  //résultat
15  retourner result
```

# ALGORITHME 3 – ALGORITHME DE GRAHAM

# ALGORITHME DE  GRAHAM

- Principe
  - Affine le principe de l'algo de Jarvis en utilisant la propriété suivante:

    dans le plan, les sommets d'un polygone se suivent dans l'ordre d'un tri angulaire à partir de n'importe quel point intérieur au polygone.
  - Faire ce tri permet de ne pas avoir à tester n sommets pour trouver le suivant de chaque sommet de l'enveloppe
- Pour n sommets
  - Choix d'un centre pour le tri et tri de n-1 sommets restants
  - Parcourir les sommets triés et sélectionner ceux qui font partie de l'enveloppe
  - => la complexité de l'algorithme dépend de la complexité du tri
    - O(nlog(n)) avec un algorithme de tri efficace

# ALGO GRAHAM

- Input
  - un ensemble V de n sommets notés $v_i$ avec i ∈ [0, n-1]
- Output
  - le résultat est la pile des sommets de V qui forment son enveloppe convexe.
- Prérequis
  - les fonctions `minY` et `tour` ont déjà été présentées.
  - La fonction `tri` est la principale difficulté de cet algorithme et fait l'objet d'un exercice de TD

```
1   //initialisations
2       min = minY(V)
3       candidates = [tri(V–{min},min),min]   // on trie sans le min, et on ajoute min en fin de liste
4       result = [min, candidates[0]]         // result est un tableau avec des fonctions de pile
5       n = taille(candidates)
6       m = taille(result)
7
8   //début de l'algo
9       pour k de 1 à n - 1
10          tant que ((t = tour(result[m-2], result[m-1], candidates[k]) <= 0 && m >= 2)
11              si (t == 0)
12                  exception ("alignement", [result[m-2], result[m-1], candidates[k]])
13              fin si
14              result.pop()
15              m = taille(result)
16          fin tant que
17          result.push(candidates[k])
18          m = taille(result)
19      fin pour
20
21  //résultat
22      retourner result                      // l'enveloppe convexe de V qui commence par min
23                                            // et termine par min
```

# BILAN DES ALGORITHMES DANS LE PLAN

- Algorithme des demi-plans
  - $O(n^3)$
- Algorithme de Jarvis
  - $O(n^2)$
- Algorithme de Graham
  - O(nlogn)
- D'autres algorithmes existent

# EXPÉRIMENTER CES ALGORITHMES

PAR L'IMPLÉMENTATION, L'INTERACTION ET L'ANIMATION

# IMPLÉMENTATION DES ALGORITHMES ET DES TESTS

- Algorithmes et fonctions utilitaires
  - 3 algorithmes et 3 fonctions utilitaires
- Jeux de données
  - 12 jeux de données dont certains sont des familles de jeux de données paramétrables
- Quelques paramétrages interactifs
  - Centre du tri, point de référence des comparaisons, etc.
- Deux styles de tests
  - Sans animation / avec animation
- La dimension de l'espace des tests possible
  - 12 x 6 x a x 12

# OBJECTIFS PRATIQUES ET DÉFI

- Implémentation et tests unitaires
  - des fonctions utilitaires
  - des algorithmes
- Implémentation de quelques ébauches d'animations d'algorithme
  - Pour comprendre, débugger et/ou expliquer
- En partant d'un code de départ
  - Ou pas
- En arrivant à un code commun
  - Servant de référence pour des exercices sur les enveloppes convexes à l'examen
- Le tout en javascript/html/css
- Pourquoi est-ce un défi?

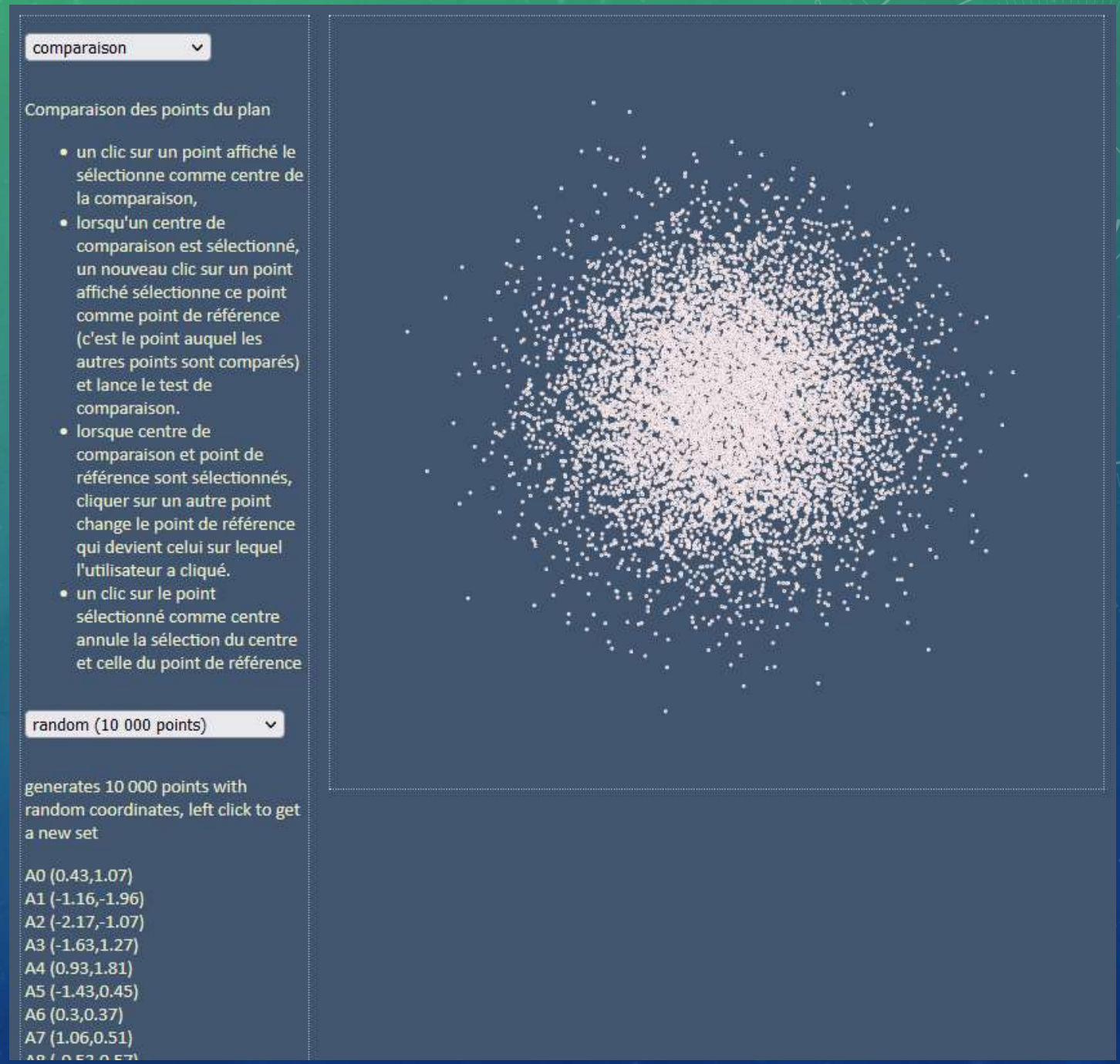
# QUELQUES EXPLICATIONS

# L'IMPLÉMENTATION

LE CODE DE DÉMARRAGE – PRINCIPES – STRATÉGIE – CONVENTIONS DE NOMMAGE

# UNE APPLICATION WEB

- Implémentation des algorithmes et fonctions utilitaires
- Programmation
  - Affichage
  - Interaction
  - Animations



# PRINCIPES

- Choix du langage de programmation
  - pas obligatoire mais "priorisé"
  - javascript avec le canvas et un peu de html et de css
  - programme de démarrage ouvert et à discuter
  - convention de notations minimalistes mais utiles
- Priorités
  - Lisibilité du code

    <= simplifier, optimiser pour la lisibilité

    => minimiser les commentaires
  - Les interactions et animations pour tester
  - Le langage javascript

Remarques

Approche client léger

Pas d'entrées/sorties fichier

Portabilité raisonnable

Ecriture des éléments html par le script js

# STRUCTURE

- Evolution continue
  - En chantier, mais en progrès
- Priorité de la structuration
  - Modularité
  - Simplicité/expressivité
  - Refactorings fréquents
- Utilisation des classes js
- Page web minimaliste
  - Tout est dans le code js

- Principaux éléments dans le désordre
  - EnveloppeConvexe
  - Displayer
  - Coord2D
  - Animation
  - ControlPanel
  - Test
  - Colors
  - TriRadial
  - DataSet

# LES INTERACTIONS ET L'AFFICHAGE

- ControlPanel
  - Paramétrage des tests
    - Jeux de données
    - Type de tests
    - Fonction à tester
- Test
- EnveloppeConvexe

- Displayer
  - Dessin des résultats
  - Interaction souris minimale
    - Sélection simple
    - Réinitialisation random de certains types de jeux de données