PAF : Multi-scale tracking of Collective Movement

Groupe 16.2:
Mathis DUPONT, Vincent HERFELD, Benjamin TERNOT, Inès VARHOL
Encadrante : Ada DIACONESCU

June 2022

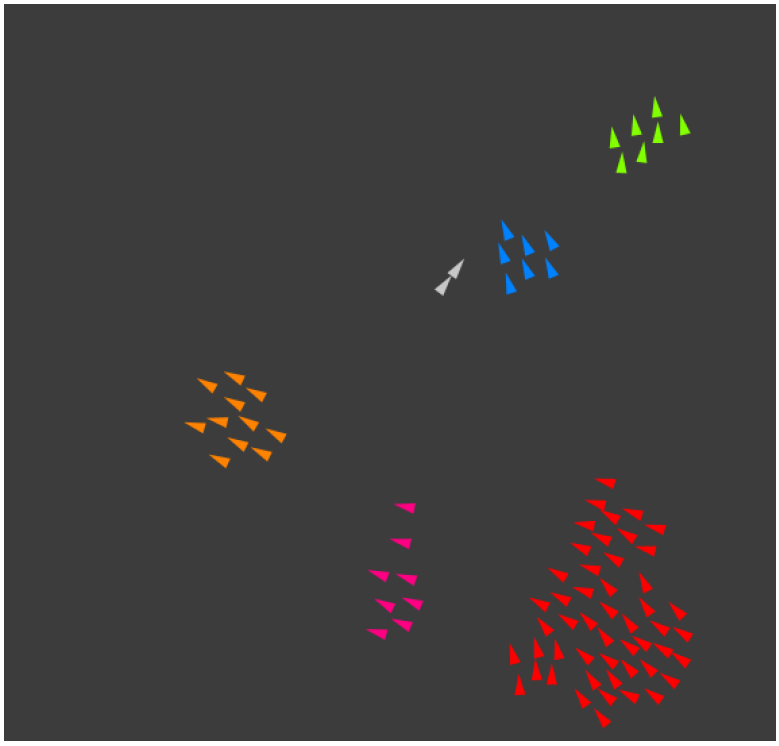


FIGURE 1 – Flocking

Table des matières

I Prise en main, généralités

Pour utiliser notre simulation, il faut télécharger Processing (`https://processing.org/download`) et notre dépôt git (`https://gitlab.telecom-paris.fr/PAF/2122/gr16-2`).

Il faut ouvrir Main.pde du dossier Main dans Processing, puis lancer le programme. On peut mettre en pause et exporter les données en appuyant sur espace puis sur la touche "s" dans la fenêtre de simulation. Les données (les positions des barycentres des groupes à chaque frame) sont exportées dans Main/traj.csv.

On peut choisir les paramètres suivants (variables globales en haut du fichier Main.pde) :

— `NBOIDS` : le nombre de boids

— `FRAMERATE` : le nombre de frames par seconde (c'est-à-dire le nombre d'update de positions des boids)

— `NMIN` : le nombre de voisins nécessaires pour être un boid noyau

— `STARTPOS` : les positions de départ des boids

- 0 : positions aléatoires
- 1 : 2 clusters
- 2 : 4 clusters

— `DISTNEIGHBORS` : la distance maximale pour être considérés voisins

— `BOIDRADIUS` : la taille des boids

— `OBSTACLERADIUS` : la taille des obstacles

— `DISTINTERACT` : la distance d'intéraction des boids

— `MAILLAGE` : le ratio entre la grille de pixels et la grille de stockage des données

Dans la fonction setup de Main.pde, le positionnement initial des boids est déterminé, ainsi que la taille de la fenêtre de simulation.

Voici la liste des commandes dans notre simulation finale :

— touche "espace" : met la simuation en pause

— touche "s" : exporte les données dans Main/traj.csv (*attention* l'export ne fonctione que si la simulation est en pause !)

— clic de souris : ajoute un boid

— touche "o" : place un obstacle à l'endroit du curseur

— touche "r" : réinitialiser la simulation

— touche "t" : affiche/cache les trajectoires des groupes

II Déplacement des boids

II.1 Réutilisation d'un code

Nous avons réutilisé une simulation présente sur le site de Processing (`https://processing.org/examples/flocking.html` codé par Daniel Shiffman). Cette simulation affiche des boids qui se déplacent sous l'effet de trois forces :

— cohésion : les boids ont tendance à se rapprocher des boids proches

— séparation : les boids gardent une distance minimale entre eux

— alignement : les vecteurs vitesse des boids proches ont tendance à être dans la même direction

Dans Boid.pde, les méthodes `cohesion, separate, align` renvoient un vecteur qui représente la force associée. Les méthodes `flock, applyForce` et `update` mettent à jour la position des boids en fonction des forces.

Les boids se déplacent comme sur un tore, si un boid sort du côté droit, il réapparait du côté gauche (méthode `borders` dans Boids.pde).

Les boids sont affichés par la méthode `render`.

II.2 Vision des voisins

Dans notre simulation, nous gardons ces trois forces, mais nous avons changé les calculs de distance entre boids et nous avons ajouté une 4ème force qui modélise des obstacles (voir V.Obstacle).

A l'origine, dans le calcul de chaque force pour chaque boid A, les distances entre A et tous les autres boids sont calculées pour déterminer quels sont les voisins de A. Nous utilisons une carte (un `ArrayList<Boid> Map` attribut de `Flock`) qui retient les positions de tous les boids à chaque frame. Pour connaître ses voisins, un boid parcourt toutes les cases proches de lui dans la carte grâce à la méthode `searchNeighbours` dans Boid.pde.

Techniquement, l'utilisation de `Map` n'est pas décentralisée : toutefois, elle remplace simplement les yeux des oiseaux ou des poissons que l'on pourrait vouloir étudier.

II.3 Distance sur un tore

Nous avons changé le calcul de distance du programme d'origine pour que la distance corresponde à la "réalité physique" du tore.

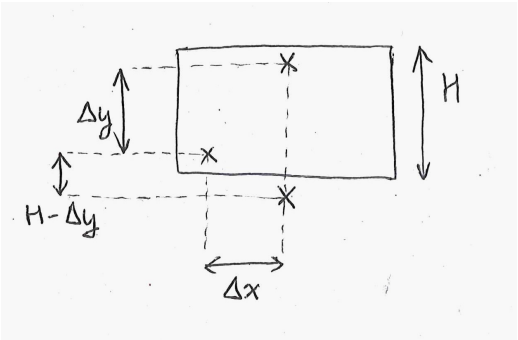


FIGURE 2 – Calcul de la distance entre deux points

Par exemple, dans la figure 2, on a pour les deux points choisis :

— distance classique = $\sqrt{\Delta x^2 + \Delta y^2}$

— distance réelle = $\sqrt{(\Delta x)^2 + (H - \Delta y)^2}$

La formule générale pour calculer la distance sur notre écran de taille (width = W, height = H) est donc :

$$\mathrm{d} = \sqrt{min(\Delta x, W - \Delta x)^2 + min(\Delta y, H - \Delta y)^2}$$

Cette distance est calculée dans la méthode `distance` de Boid.pde.

III Clustering

Nous avons longuement réfléchi aux règles pour identifier les groupes de boids. Voici les différentes façons que nous avons envisagé : considérer que deux boids suffisamment proches sont dans le même groupe, ou bien considérer en plus si leurs vecteurs vitesse sont suffisamment alignés, ou bien définir les groupes à partir de boids "noyaux" avec un nombre minimal de voisins.

III.1 Clustering par distance : versions 1.x

Nous avons implémenté deux versions de “groupement par distance” qui fonctionnent sur ce modèle. Une première version 1.1 (commit SHA 05a71e) qui a l’inconvénient de ne pas attribuer un indice unique à chaque groupe, et une deuxième version 1.2 (commit SHA 20f0cc) qui corrigeait ce défaut. Nous avons défini les règles suivantes pour qu’un groupe A conserve le même identifiant :

- il doit contenir la majorité des membres du groupe A du frame précédent
- les anciens membres du groupe A doivent être majoritaires dans le nouveau groupe

L’inconvénient du “groupement par distance”, c’est que parfois deux groupes de boids avec des directions opposées peuvent se rapprocher sans pour autant que les groupes ne fusionnent.

III.2 DBSCAN : version 2

Le DBSCAN se base sur la densité pour identifier des groupes. On repère des boids "noyaux" qui ont plus de `NMIN` voisins (on peut changer ce paramètre dans Main/Main.pde, il faut `NMIN > 2`, sinon le clustering revient à trouver les composantes connexes d’un graphe).

Les boids noyaux voisins sont automatiquement dans le même groupe : on considère de plus que tout voisin d’un boid noyau fait partie du même groupe que lui.

Les boids non-voisins d’un boid noyau ne font partie d’aucun groupe et ne sont pas étudiés dans notre simulation 2.0.

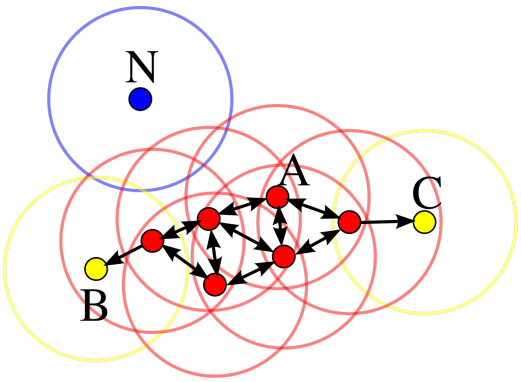


FIGURE 3 – principe du DBSCAN (src : `https://fr.wikipedia.org/wiki/DBSCAN#/media/Fichier:DBSCAN-Illustration.svg`)

Par exemple, sur la figure 3, on a `NMIN = 3`, les boids rouges représentent les boids noyaux, les boids jaunes ne sont pas des noyaux mais ont au moins un voisin qui l’est, et le boid bleu n’a pas de voisin noyau. Le groupe créé par DBSCAN est donc l’ensemble des boids rouges et jaunes, avec comme noyau les boids rouges.

L'avantage du DBSCAN par rapport au groupement par distance est que lorsque des chaînes de boids se forment entre les groupes, les groupes sont toujours considérés comme étant différents.

La création d'un groupe se fait grâce aux méthodes `createGroup`, `propagateGroup` et `tellOthers` dans Main/Boid.pde, expliquées comme ceci :

— La méthode `createGroup` est appelée pour tous les boids, mais elle a un effet seulement pour les boids noyaux qui n'ont pas déjà reçu une affectation à un nouveau groupe. Dans ce cas, cette méthode appelle la méthode récursive `propagateGroup` au boid actuel, et récupère le groupe retourné par celle-ci.

```
1  int createGroup(int gMin, HashMap<Integer, ArrayList<PVector>>
       trajectoriesAll,  HashMap<Integer, ArrayList<PVector>>
       trajectoriesCore, ArrayList<Group> groups, int scale){
2  if (this.neighbors.size() >= NMIN && !this.activeCreate){
3    Group g = this.propagateGroup(this);
4    ...
5  } else {
6    return gMin;
7  }
8  }
```

— La méthode `propagateGroup` consiste en un parcours en profondeur d'arbre, avec récupération d'information, de sorte que le groupe se construit lors de la remontée d'information au boid père. Afin d'assurer la terminaison de la méthode, les boids déjà appelés sont marqués par une variable booléenne. De plus, un boid propage la méthode à ses voisins si et seulement si il est lui-même un boid noyau, de manière à conserver le principe du DBSCAN. Si le boid n'est pas un boid noyau, il s'ajoute au groupe uniquement si le boid père qui lui a propagé la méthode est son plus proche voisin.

```
9  Group propagateGroup(Boid father){
10 Group g = new Group();
11 if (this.activeCreate == false){
12   this.activeCreate = true;
13   if (this.neighbors.size() >= NMIN){
14     for (Boid neighbor : this.neighbors){
15       g.merge(neighbor.propagateGroup(this));
16     }
17     g.addBoid(this);
18   } else {
19     for (Boid neighbor : this.neighbors){
20       if (this.distance(neighbor) < this.distance(father)){
21         this.activeCreate = false;
22         return g;
23       }
24     }
25     g.addBoid(this);
26   }
27 }
28 return g;}
```

— Dès lors, le boid de la racine récupère son groupe g au complet, et appelle ensuite la méthode `g.defineId`, de sorte que le groupe g va s'attribuer lui-même son id en fonction de la répartition des anciens groupes de ses membres.

```
1   int createGroup(...){
2       if (this.neighbors.size() >= NMIN && !this.activeCreate){
3         Group g = this.propagateGroup(this);
4           int n = g.defineId(gMin);
5           if (n > gMin){
6               groups.add(g);
7           }
8           g.addTrajectories(trajectoriesAll, trajectoriesCore);
9           groups.set(g.id, g);
10          if(show){
11              g.renderCenter(scale);
12              g.renderVector(scale);
13          }
14          this.tellOthers(g, this);
15          return n;
16      } else {
17        return gMin;
18      }
19  }
```

— Pour finir, il ne reste plus qu'à transmettre à tous les boids du groupe la valeur finale du groupe grace à `tellOthers`, à nouveau de manière récursive par le même parcours d'arbre que `propagateGroup`.

```
20  void tellOthers(Group g, Boid father){
21      if (this.activeTell == false){
22          this.activeTell = true;
23          if (this.neighbors.size() >= NMIN){
24              for (Boid neighbor : this.neighbors){
25                  neighbor.tellOthers(g, this);
26              }
27              this.newGroup = g;
28          } else {
29              for (Boid neighbor : this.neighbors){
30                  if (this.distance(neighbor) <  this.distance(
                        father)){
31                      this.activeTell = false;
32                  }
33              }
34              if (this.activeTell){
35                  this.newGroup = g;
36              }
37          }
38      }
39  }
```

III.3 Décentralisation

Que ce soit le groupement par distance ou le groupement type DBSCAN, nous avons implementé des versions décentralisées du code.

Pour cela, nous utilisons la méthode `searchNeighbours` de Boid.pde qui permet à chaque boid de repérer les boids voisins. Cette méthode se base sur la lecture d'un tableau qui stocke les boids par position et qui est mis à jour à chaque frame. Cela permet notamment de réduire la complexité comparé à un calcul de distance entre tous les boids ($\mathcal{O}(n)$ contre $\mathcal{O}(n^2)$).

Au début, nous avions implémenté des méthodes plus centralisées, que ce soit la formation des groupes par distance (parties connexes d'un graphe), leur attribution d'un id en fonction des groupes précédents, comme le calcul des barycentres (méthode `findCenter` dans Flock.pde).

Par la suite, en créant les groupes de manière décentralisé par les méthodes récursives de la classe Boid, il nous est paru logique de créer une classe Group afin de faciliter l'échange d'informations de types divers dans le parcours d'arbre. Dès lors, on a décidé que les calculs liés à un groupe pouvaient s'effectuer uniquement par des méthodes qui lui sont propres, et ainsi tout est devenu décentralisé. La méthode `defineId` permet à un groupe de s'identifier ou non à un ancien groupe, comme expliqué en III.1, et les barycentres sont désormais calculés, stockés et affichés en interne.

IV Outils d'analyse

Dans cette section nous allons aborder les différents outils construits permettant d'analyser le comportement des boids une fois les groupes formés.

IV.1 Calcul de barycentre et vecteur vitesse de groupe

Nous avons décidé dans un premier temps, de caractériser un groupe par son isobarycentre. Pour un groupe de taille N avec $b(i)$ le vecteur position du i-ieme boid du groupe :

$$barycentre = \frac{1}{N} \sum_{i=1}^{N} b(i)$$

Toutefois comme nos boids se déplacent sur un espace torique, le calcul du barycentre dans un plan ne correspond pas toujours au centre du groupe que l'on voit sur l'écran. C'est pourquoi nous avons calculé un barycentre adapté.

On considère séparément les abscisses et les ordonnées des points. En temps normal, on place les valeurs sur un segment [0, width] et on calcule leur valeur moyenne : ici, on place les valeurs sur un cercle et on calcule la direction moyenne.

Ainsi, l'abscisse du barycentre d'une liste de points d'abscisse x_i est

$$barycentre(x_i) = \frac{width}{4} + \frac{width}{2\pi} * arctan\left(\frac{\sum_{i=1}^{N} cos(\frac{2\pi x_i}{width})}{\sum_{i=1}^{N} sin(\frac{2\pi x_i}{width})}\right) + \frac{width}{2} * int(\sum_{i=1}^{N} sin(\frac{2\pi x_i}{width}) < 0)$$

On a utilisé la fonction $atan2$ pour implémenter le calcul dans le code.

IV.2 Conservation des trajectoires, groupement par distance

Une fois les barycentres calculés il est simple de conserver les trajectoires de chaque groupe. Il suffit de placer dans un `ArrayList<PVector>` l'ensemble des barycentres associés à chaque groupe. Nous avons pris un dictionnaire `trajectories` qui a le numéro de groupe pour clef à laquelle on associe la liste des barycentres du groupe.

Lorsque l'on fixe un état initial il peut arriver que l'on place plusieurs boids sur une même case (exemple : séparation de tous les boids en deux points). Sachant que l'algorithme de recherche de voisins `searchNeighbours()` pour un boid b ne considère seulement les boids qui ne sont sur une autre case que celle de b, nous avons décidé de négliger les groupes formés qui disparaissent avant la 30^{eme} frame de l'acquisition. Après cette frame, nous conservons tous les groupes (qu'ils disparaissent ou non) dans le dictionnaire décrit plus haut.

Une dimension importante pour nous est le temps. Nous avons décidé de rester fidèle à cette notion et pour cela avant la création d'une nouvelle trajectoire de groupe (c'est-à-dire à la création d'un nouveau groupe) nous plaçons dans le dictionnaire des trajectoire au niveau de la clef du nouveau groupe, des points (-10,-10) afin de faire du padding. On place exactement `frameCount - 30` points (-10,-10).

A chaque frame le dictionnaire est rempli, et lorsque l'on appuie sur la touche espace et sur la touche "s", la simulation se met en pause et une image (le dictionnaire des trajectoires) de l'état actuel est sauvée dans un document `.csv`.

Ce document pourra donc etre exploité dans un tableur classique de type Excel. Cependant, nous avons aussi mis au point un programme Processing permettant d'observer seulement les trajectoires de groupes acquisent dans un intervalle de temps donné par l'utilisateur. Il suffit

de modifier les valeurs de `startFrame` et de `endFrame` dans `Graphs.pde`. A noter que le `endFrame` sera automatiquement limité par la durée de l'acquisition dont est issu le document `traj.csv`.

IV.3 Conservation des trajectoires, DBscan

Dans la version DBSCAN, les trajectoires d'un barycentre sont stockées dans un attribut de la classe `Groupe`, et leurs calculs sont effectués dans cette même classe (paradygme OO java + décentralisation).

Cela permet à chaque groupe de récupérer l'historique de ses barycentres de façon à tracer lui-même ses propres trajectoires, toujours de manière décentralisé.

IV.4 Mode analyse en temps-réel

Nous avons mis au points un mode d'affichage qui permet d'observer en temps-réel le tracé des trajectoires de groupes ainsi que les vecteurs vitesse de chaque groupe. Pour activer (ou désactiver) ce mode il suffit de tapper sur la touche `"t"` de son clavier.

L'épaisseur des trajectoires est proportionnelle au nombre de membres dans le groupe considéré. Seulement les trajectoires des groupes existants sont affichées, c'est un choix qui est effectué, de façon à pouvoir garantir l'aspect décentralisé de notre projet. (pour les versions 1, le choix est effectué dans la méthode `run` de Main/Flock.pde dans la boucle ligne 107). En effet, chaque groupe s'occupe d'afficher ses barycentres, et de ce fait, les groupes disparus ne sont plus jamais considérés dans la boucle `run`.

Il est possible d'afficher toutes les trajectoires (même des groupes qui n'existent plus), mais nous conseillons de plutôt le faire depuis Graphs/Graphs.pde (qui affiche automatiquement tous les groupes), de manière à pouvoir choisir un intervalle de temps précis.

IV.4.1 Exemple d'utilisation

Lancez une acquisition en appuyant sur le bouton `"play"` (▶). Ensuite vous pourrez appuyer sur la touche `"t"` du clavier, pour activer puis désactiver le mode d'analyse en temps-réel.

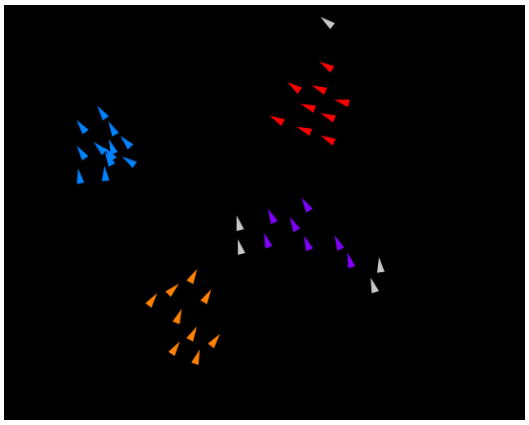


FIGURE 4 – Mode désactivé

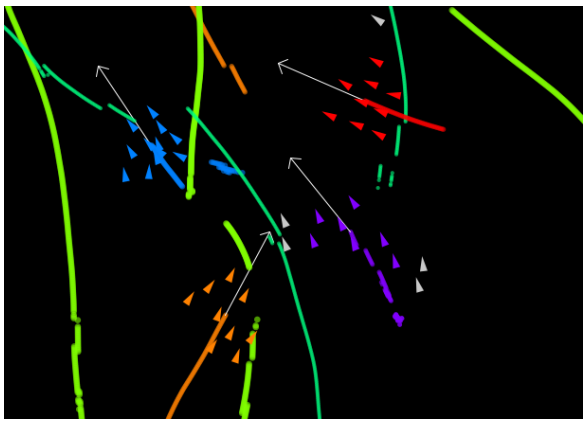


FIGURE 5 – Mode activé

IV.5 Lentilles (scales)

Pour analyser de différentes manieres, nous mettons à dispotion une lentille qui peut s'appliquer sur la vue de l'observateur. Cette lentille va permettre soit de visualiser tous les boids et leurs trajectoires de groupe (scale 0) soit seulement afficher les boids dit de coeur (avec un nombre minimum de voisins, cette valeur est fixé dans `Main.pde` sous la variable `NMIN`) ainsi que les trajectoires de groupes adaptés à cette vue (scale 1).

La lentille joue aussi sur le calcul des barycentres, ceux si sont soit calculés à partir de tous les boids d'un groupe (scale 0), soit uniquement à partir de ses boids noyaux (scale 1)

L'utilisateur peut très facilement appliquer la lentille ou non en tappant sur les touches `0` ou `1` sur son clavier.

IV.5.1 Exemple d'utilisation

Pour bien mettre en évidence cette fonctionalité, il est conseillé de réduire la distance maximale de voisinage. Pour cela il suffit d'aller dans `Main.pde` et modifier la valeur de `DISTNEIGHBOUR`. Pour cet exemple `DISTNEIGHBOUR = 30`. Ensuite lancez l'acquisition en appuyant sur le bouton "`play`" (▶). Vous observerez des groupes avec des boids de couleur (ceux du noyau) et des gris (bruits ou exterieurs au noyau). Lorsque vous changerez de lentille en particulier avec la lentille scale 1 (touche `1` sur le clavier) les boids gris vont disparaitre et les calculs de trajectoires se feront sans eux.

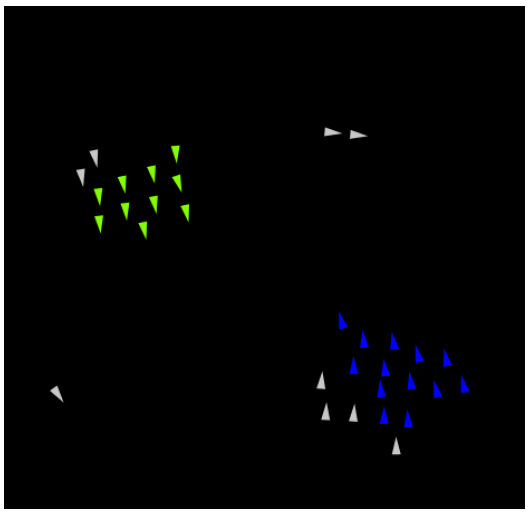


FIGURE 6 – Scale 0

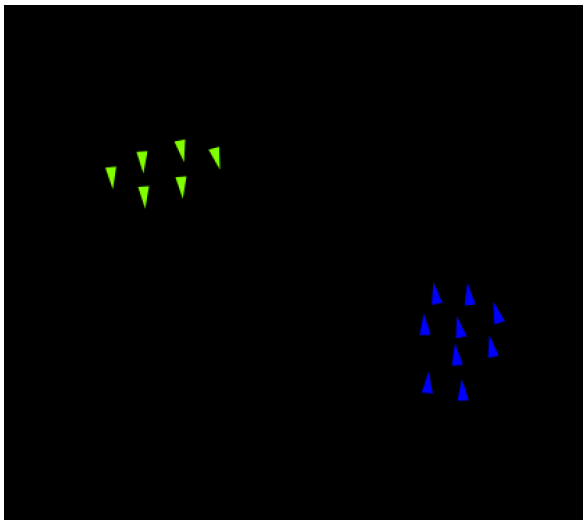


FIGURE 7 – Scale 1

IV.5.2 Remarque

Avant d'implémenter les différentes échelles de vue (lentilles), nous pensions que le scale 1 permettrait de lisser les trajectoires car nous éliminons le bruit et donc qu'il y aurait peu de variations. Finalement les trajectoires du scale 0 sont plus lisses que celle générées en scale 1. Cela est dû à la forte mobilité des boids qui cause des variations rapides entre boids de coeur et boid exterieurs.

V Obstacle

Nous avons implémenté des obstacles sous la forme d'une 4ème force qui agit sur les boids. Un obstacle est défini par sa position et sa force de répulsion.

A chaque frame, la méthode `searchNeighbours` de Obstacle.pde va déterminer quels boids sont dans le voisinage de l'obstacle et leur appliquer une force de répulsion dans la méthode `run`.

Nous voulions appliquer une force aux boids de façon à pouvoir séparer un groupe temporairement. Seulement, nous n'avons pas réussi à implémenter une force de ce type. Cette force necessite de conserver pendant un temps (entre l'entrée et la sortie de la zone d'application de l'obstacle) le vecteur vitesse du groupe, afin de le lui appliquer comme force. Cela l'entrainera à avancer dans la direction initiale et donc au groupe séparé de se regrouper.

Nous avons choisi de faire une force qui dévie les boids à une certaine distance de l'obstacle. La déviation choisie dépend du point d'entrée dans la zone d'activation des boids. Nous avons séparés les cas afin de faire passer un boid d'un coté ou l'autre de l'obstacle de la manière la plus naturelle. Le résultat obtenu est très similaire à la réaction d'un poisson face à une main humaine qui se rapproche par exemple. On rappelle tout de meme que les obstacles sont fixes.

Nous avons aussi mis en place un code couleur permettant d'identifier les obstacles les plus "repulsifs" (faible : vert, moyen : bleu, élevé : rouge). L'utilisateur pourra placer des obstacles de deux forces différentes en appuyant sur la touche "`o`" et "`O`" ("`o`" majuscule) de son clavier. Les variables globales `newObsForce1` et `newObsForce2` (modifiables dans `Main.pde`) définissent respectivement la force de l'obstacle placé grace à "`o`" ou "`O`". L'obstacle sera placé là où se trouvait la souris au moment de l'appuis sur la touche.

V.0.1 Exemple d'utilisation

Modifiez comme vous le souhaitez `newObsForce1` et `newObsForce2` dans le fichier `Main.pde`. Ensuite lancez l'acquisition en appuyant sur le bouton "`play`" (▶). Maintenant placez votre souris sur l'ecran et appuyez soit sur "`o`" soit sur "`O`". Un obstacle apparaitera là où se trouvait votre souris.

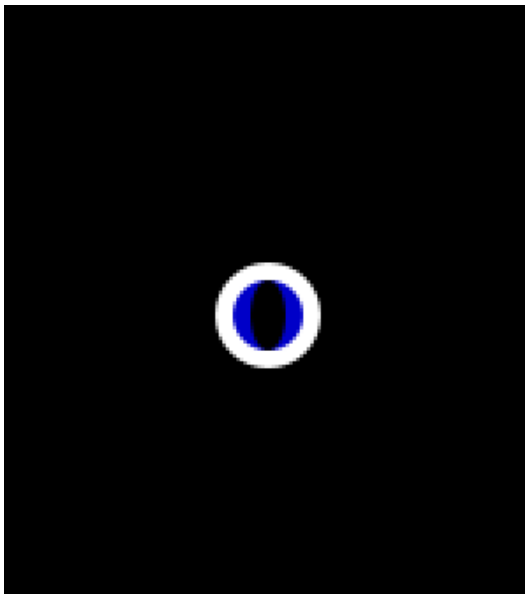


FIGURE 8 – Obstacle de force moyenne

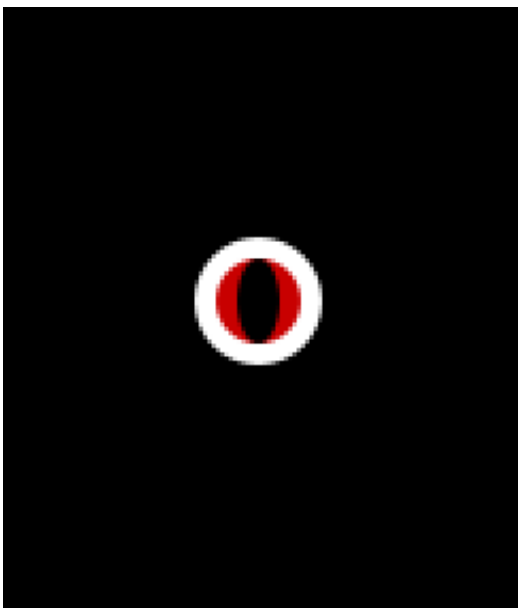
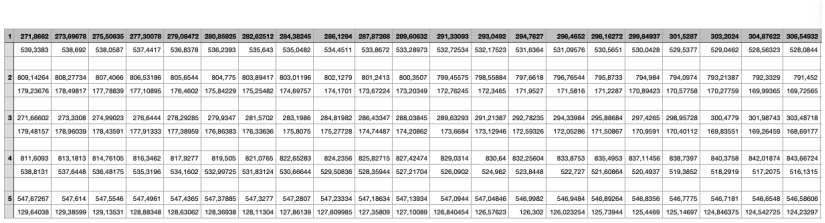


FIGURE 9 – Obstacle de force élevée

VI Tracé de graphiques

Une fois une acquisition lancée l'utilisateur pourra mettre l'acquisition en pause et sauvegarder l'état en appuyant sur la touche `s`. Un fichier .csv sera généré dans le dossier `Trajectories/` avec des couples de lignes la premiere représentant les abscisses des barycentres et la deuxieme les ordonnées. On conserve aussi le numéro de groupe qui permettra de reconstruire la couleur du tracé.



	1	271,8662	273,69678	275,50635	277,30078	279,08472	280,85925	282,62512	284,38245	286,1294	287,87268	289,60632	291,33093	293,0492	294,7627	296,4652	298,16272	299,84937	301,5287	303,2024	304,87622	306,54932
		539,3383	538,692	538,0587	537,4417	536,8378	536,2393	535,643	535,0482	534,4511	533,8672	533,28973	532,72534	532,17523	531,6364	531,09576	530,5651	530,0428	529,5377	529,0462	528,56323	528,0844
	2	809,14264	808,27734	807,4066	806,53186	805,6544	804,775	803,89417	803,01196	802,1279	801,2413	800,3507	799,45575	798,55884	797,6618	796,76544	795,8733	794,984	794,0974	793,21387	792,3329	791,452
		179,23676	178,49817	177,78839	177,10895	176,4602	175,84229	175,25482	174,69757	174,1701	173,67224	173,20349	172,76245	172,3465	171,9527	171,5816	171,2287	170,89423	170,57758	170,27759	169,99365	169,72565
	3	271,66602	273,3308	274,99023	276,6444	278,29285	279,9347	281,5702	283,1986	284,81982	286,43347	288,03845	289,63293	291,21387	292,78235	294,33984	295,88684	297,4265	298,95728	300,4779	301,98743	303,48718
		179,48157	178,96039	178,43591	177,91333	177,38959	176,86383	176,33636	175,8075	175,27728	174,74487	174,20862	173,6684	173,12946	172,59326	172,05286	171,50867	170,9591	170,40112	169,83551	169,26459	168,69177
	4	811,6093	813,1813	814,76105	816,3462	817,9277	819,505	821,0765	822,65283	824,2356	825,82715	827,42474	829,0314	830,64	832,25604	833,8753	835,4963	837,11456	838,7397	840,3758	842,01874	843,66724
		538,8131	537,6448	536,48175	535,3196	534,1602	532,99725	531,83124	530,66644	529,50836	528,35944	527,21704	526,0902	524,962	523,8448	522,727	521,60864	520,4937	519,3852	518,2919	517,2075	516,1315
	5	547,67267	547,614	547,5546	547,4961	547,4365	547,37885	547,3277	547,2807	547,23334	547,18634	547,13934	547,0944	547,04846	546,9982	546,9484	546,89264	546,8356	546,7775	546,7181	546,6548	546,58606
		129,64038	129,38599	129,13531	128,88348	128,63062	128,36938	128,11304	127,86139	127,60985	127,35809	127,10089	126,840454	126,57623	126,302	126,023254	125,73944	125,4469	125,14697	124,846375	124,542725	124,23297

FIGURE 10 – .csv file

Nous avons écrit un second programme contenu dans le dossier Graphs qui permet de lire des .csv formatés comme le notre (les positions des barycentres sont sur deux lignes, une pour l'abscisse et une pour l'ordonnée, les colonnes représentent les différents moments).

Pour lire un .csv il faut modifier le `path` dans `Graphs.pde`.

Ce programme permet d'afficher les trajectoires sur les intervalles de temps souhaités : on visualise les trajectoires entre le n° de colonne `startFrame` et `endFrame`. Les trajectoires de différents groupes seront séparés par couleur identique à celle du groupe lors de l'acquisition. La couleur est choisie à partir de la valeur de l'identifiant de groupe (un entier).

VI.0.1 Exemple d'utilisation

Lancez une acquisition en appuyant sur le bouton "`play`" (▶) avec `Main.pde` ouvert. Laissez les trajectoires se former (vous pouvez les observer en appuyant sur "`t`"). Lorsque vous etes satisfait, tappez sur la barre d'espace, cela mettra la simulation en pause. Vous pouvez ensuite en pause appuyer sur la touche "`s`" de votre clavier pour sauvegarder l'état actuel dans un fichier `.csv`. VOus verrez apparaitre un nouveau fichier ("traj DATE HEURE.csv") dans le dossier `Trajectories/`. Vous pouvez alors ouvrir ce fichier dans le tableur de votre choix et afficher des nuages de points. Vous pouvez aussi ouvrir le fichier `Graphs/Graphs.pde`, modifier les valeurs de `startFrame` et de `endFrame` ainsi que le chemin vers le fichier `.csv` que vous souhaitez ouvrir dans la variable globale `.path`. Finalement appuyez sur le bouton "`play`" (▶).

VI.0.2 Remarque

On voit sur la Figure 11 des trajectoires vertes qui n'apparaissent pas sur la Figure 10. Cela est dû à la disparition du groupe qui a tracé cette trajectoire. La reconstruction conserve l'historique des groupes donc vous verrez apparaitre des trajectoires passées qui n'apparaissent plus dans l'acquisition.

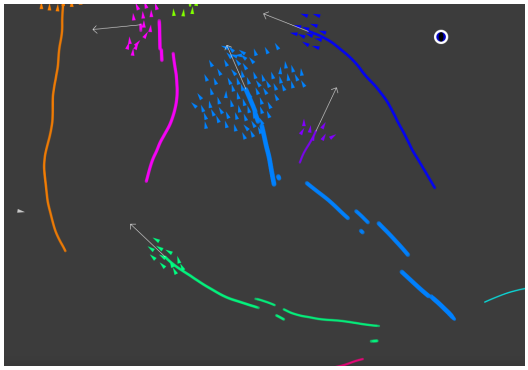


FIGURE 11 – Acquisition

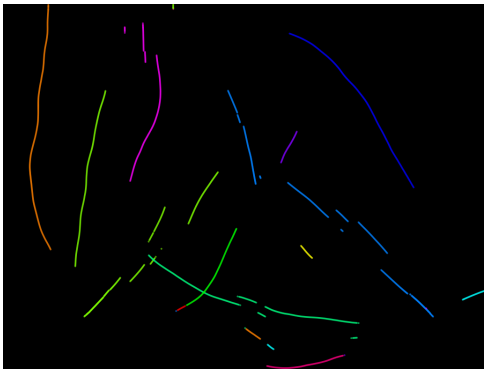


FIGURE 12 – Reconstruction avec startFrame = 0 et endFrame = 500

FIGURE 13 – Reconstruction avec startFrame = 150 et endFrame = 200