TP MPI: Boids

ENS-Rennes

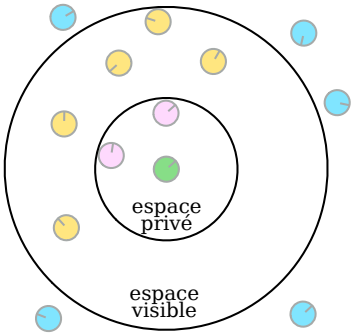
Boids est un système introduit par Raynolds en 1987 permettant une simulation réaliste de nuées d'oiseaux. Chaque *bird-oid* suit des règles simplistes, mais l'ensemble des oiseaux peut exhiber une cohérence de groupe. L'objectif du TP est de paralléliser une simulation de boids[1] avec MPI.

Objectifs pédagogiques:

- Parallélisation par décomposition des données;
- Problématique d'équilibrage de charge;
- Échanges de données collectives et point-à-point en MPI;

Chaque boid est représenté par une position (x, y) et une vitesse (vx, vy). Les forces auxquelles les boids sont soumis sont les suivantes:

1. **Séparation:** pour éviter les collisions, chaque boid tend à s'éloigner des boids qui se trouvent dans son espace privé.
2. **Alignement:** chaque boid tend à aligner sa vitesse avec celle des boids dans son espace visible.
3. **Cohésion:** chaque boid tend à se diriger vers le centre de gravité de la masse des boids autour de lui.
4. **Évitement d'obstacles:** quand il s'approche de la bordure de l'écran, un boid tend à modifier sa vitesse pour s'en éloigner.
5. **Contrôle de vitesse:** on ajuste la vitesse pour rester entre une limite haute et une limite basse



Le code fourni permet de simuler une nuée de boids et d'afficher leurs positions à l'écran avec CSFML. Il vous faut la version 3.35 de SimGrid (pour simplifier la compilation avec CMake et SMPI). Voici le pseudo-code de la mise à jour du tableau:

```
Pour chaque boid
  Mise à zéro de tous les accumulateurs:
    xpos_avg, ypos_avg, xvel_avg, yvel_avg, num_neighbors, close_dx, close_dy

  Pour chaque autre boid dans la nuée nommé ‘‘otherboid’’

    Si boid et other sont à portée de vue:
       Si other se trouve dans l’espace privé de boid, ce dernier va vouloir s’écarter:
          close_dx += boid.x - otherboid.x
          close_dy += boid.y - otherboid.y
       Sinon, calcul des accumulateurs de cohésion et d’alignement:
          xpos_avg += other.x       ;       xvel_avg += other.vx
          ypos_avg += other.y       ;       yvel_avg += other.vy
          num_neighbors += 1

  # Appliquer la séparation
  boid.vx += (close_dx * separation_f)  ;  boid.vy += (close_dy * separation_f)

  Si num_neighbors > 0: # appliquer la cohésion et l’alignement
    xpos_avg /= num_neighbors    ;    xvel_avg /= num_neighbors
    ypos_avg /= num_neighbors    ;    yvel_avg /= num_neighbors

    boid.vx += (xpos_avg - boid.x) * cohesion_f + (xvel_avg - boid.vx) * alignement_f
    boid.vy += (ypos_avg - boid.y) * cohesion_f + (yvel_avg - boid.vy) * alignement_f

  # évite les bordures
  si boid.x < leftmargin:   boid.vx += turnfactor
  si boid.x > rightmargin:  boid.vx -= turnfactor
  si boid.y > bottommargin: boid.vy -= turnfactor
  si boid.y < topmargin:    boid.vy += turnfactor

  si speed < minspeed: boid.vx= (boid.vx/speed)*minspeed; boid.vy= (boid.vy/speed)*maxspeed
  si speed > maxspeed: boid.vx= (boid.vx/speed)*maxspeed; boid.vy= (boid.vy/speed)*maxspeed

  boid.x += boid.vx     ;     boid.y += boid.vy
```

[1] Ce TP s'inspire librement de `https://vanhunteradams.com/Boids/Boids-predator.html`

Le code fourni suit le modèle MVC[2], en séparant les considérations d'affichage et celles de mise à jour du monde, et il est raisonnablement optimisé. Vous êtes cependant libres de modifier ce code comme vous le souhaitez. En particulier, les paramètres fournis ne sont pas optimaux pour de beaux comportements de troupeau. On pourrait aussi économiser quelques appels à la coûteuse fonction de racine carrée en approximant la vitesse avec l'algorithme "alpha max plus beta min", ou bien trier le tableau des boids en fonction de leur position et ne considérer que les boids dont les coordonnées en x laissent à penser qu'ils peuvent être proches. L'objectif est cependant de passer du temps sur les versions parallèles: ne perdez pas trop de temps avec la version séquentielle.

★ **Exercice 1: Parallélisation par individus** (obligatoire).

Une première idée pour paralléliser ce code entre N ranks est de découper le vecteur d'individus en N parties, et attribuer chaque partie à un rank donné. À chaque itération, l'intégralité du vecteur est synchronisé entre les participants avec un appel à `MPI_Allgather` (si la quantité de boids gérés n'est pas la même pour tous les ranks, vous pourriez avoir besoin de `MPI_Allgatherv`, mais ce n'est pas forcément conseillé). Le rank 0 se chargera en plus d'afficher l'état du monde.

▷ **Question 1:** Implémentez ce mode de parallélisation.

CMake est réglé pour compiler le code sans SMPI. Décommentez la ligne `smpi_c_target` et utilisez `smpirun` pour exécuter votre code sans segfault une fois que vous êtes passés à MPI.

▷ **Question 2:** Mesurez le speedup de votre solution. Il peut être utile d'agrandir la taille du monde quand le nombre de boids devient grand. Vers quelle valeur semble converger le speedup?

▷ **Question 3:** (optionnelle) Lors de l'affichage, donnez une couleur spécifique à chaque rank afin de voir graphiquement qui s'occupe de quels boids.

★ **Exercice 2: Parallélisation spatiale** (optionnelle mais recommandée – points bonus).

Comme l'impact potentiel des boids les uns sur les autres est limité dans l'espace et que la boucle coûteuse du calcul est en $O(n)$, il est assez naturel de proposer une répartition spatiale du travail : chaque rank est en charge d'une bande d'espace, et n'échange avec ses voisins que les coordonnées des boids à la frontière.

▷ **Question 1:** (recommandée) Implémentez un découpage 1D de l'espace de la simulation. Un découpage 2D n'est PAS demandé. Le rank 0 devant participer à plus de communications (tout le monde lui envoie toutes ses données) et faire l'affichage, il peut être dispensé d'animer une partie du monde.

▷ **Question 2:** (appréciée si le temps le permet) Mesurez le speedup de cette version (sans compter le rank 0), et comparez-le au speedup obtenu à l'exercice précédent.

▷ **Question 3:** (déraisonnable) Quand le monde est assez grand et que la force de cohésion est assez forte, tous les boids volent en quelques nuées compactes. La charge de travail entre les ranks tend alors à se déséquilibrer. La limite de cette tendance est que parfois, tous les boids sont placés dans la zone d'un seul rank.

L'objectif de cette extension est d'adapter dynamiquement la taille des zones de chaque rank. À chaque étape, les ranks échangent avec leurs voisins leur nombre de boids dont ils ont la charge, et si le ratio est trop différent entre deux voisins, celui le moins chargé devient responsable des boids à la frontière (qui est donc déplacée). Cet algorithme d'équilibrage de charge ne peut pas être optimal puisqu'il est basé sur une connaissance locale uniquement de la charge, mais l'espoir est qu'il converge assez vite pour s'adapter aux déséquilibrages créés par les mouvements de boids.

Si vous avez une meilleure idée pour équilibrer la charge, n'hésitez pas à la tester en pratique, bien sûr. L'algorithme de Barnes-Hut et ses quadtrees (pour le cas 2D) mérite le détour, même si vous n'avez clairement pas le temps de l'implémenter.

★ **Exercice 3: Procrastination et autres idées** (déconseillé)

▷ **Question 1:** (optionnelle) Implémentez des prédateurs, qui se déplacent librement sur la carte (pas de cohésion ou autre), mais que les boids fuient fortement. On pourrait aussi s'amuser à faire en sorte que les prédateurs cherchent à attraper des boids, et meurent de faim quand ils n'y arrivent pas pendant trop longtemps.

▷ **Question 2:** (optionnelle) Laissez vos boids se reproduire: quand deux boids sont suffisament proches, il y a une faible probabilité pour qu'un nouveau boid apparaisse à cet endroit, avec des caractéristiques proches de celles de ses géniteurs. À l'inverse, un boid qui n'a vu personne depuis trop longtemps pourrait mourir d'ennui.

▷ **Question 3:** (optionnelle) Dotez chaque boids de paramètres qui lui sont spécifiques en terme de force de cohésion, séparation et alignement. Pour calculer les caractéristiques lors de la reproduction, on peut utiliser la moyenne de celles des géniteurs, ou chercher sur internet ce que "algorithme génétique" veut dire (mais attention: gouffre temporel juste devant vous).

[2]Modèle-Vue-Contrôleur: `https://en.wikipedia.org/wiki/Model-view-controller`