

# Sujet TP (cours 1)

## Notation

### Modalités

- Travail en binôme seulement
- **À rendre :**
  - Le code source complet
  - Un fichier `README.md` avec vos choix d'implémentation et leur justification

## Barème (/20)

### 1. Implémentation des fonctions manquantes (5 points)

- Fonctionnalités complètes et correctes

### 2. Optimisation du code (10 points)

- Optimisations techniques (7 points)
- Justification des choix et tests réalisés (3 points)

### 3. Performance (5 points)

- Classement selon le speed-up obtenu par rapport au code initial
- Si nécessaire, un code de référence servira de base de comparaison

## Conseils

- Documentez vos optimisations
- Gardez un code lisible
- Conservez l'ensemble des modifications essayées
- Testez avec différentes tailles de données

## Code donné

- fonction de benchmark (`rdtsc`, `fps`)
- initialisation des tableaux & valeurs par défaut
- système de rendu graphique (`SDL`)
- `Makefile`

## Ajout du code manquant

### Fonctions basiques

- `scale_vector` : multiplie un vecteur avec un nombre réel
- `sub_vectors` : soustrait deux vecteurs
- `mod` : calcule la distance euclidienne d'un vecteur
- `add_vectors` : additionne deux vecteurs

## Fonctions de simulations

- compute\_accelerations

La fonction `compute_accelerations` permet de calculer l'accélération d'une particule. Cette valeur change en fonction des forces appliquées par les autres particules ainsi que de la masse et la gravité présente. N'hésitez pas à changer les paramètres pour voir l'évolution du système. La fonction vous est donnée ci-dessous. ( $mod(a)$  correspond à la distance euclidienne du vecteur a)

$$mod(a) = \sqrt{(a.x)^2 + (a.y)^2}$$

$$a(t_1)' = a(t_0) * \frac{gravity * masse}{mod(pos_i - pos_j)^3 + 1e7} * pos_j - pos_i$$

- compute\_positions

La fonction `compute_positions` calcule la position de la particule en fonction de sa vitesse et de sa position à t-1. La formule vous est donnée ci-dessous.

$$p(t_1) = p(t_0) + (v(t_0) + \frac{a(t_1)}{2})$$

- compute\_velocities

La fonction `compute_velocities` permet de calculer la vitesse (vitesse) de chaque particule pour chaque pas de temps. La formule vous est donnée ci-dessous.

$$v(t_1) = v(t_0) + a(t_1)$$

- resolve\_collision

Pour la fonction `resolve_collision`, le principe est de simuler la collision entre 2 particules. Il existe plusieurs algorithmes et implémentations pour cette fonction, avec plus ou moins de précision. Vous êtes libre de choisir la fonction que vous voulez. Il vous est cependant demandé de justifier ce choix.

- check\_position (*bonus*)

La fonction `check_position` permet de faire ré-entrer une particule sortante par le côté opposé à sa sortie.

- Toutes les implémentations sont attendues en C et en ASM. N'hésitez pas à dupliquer les fonctions pour avoir toutes les versions dans le fichier `kernels.c`

e.g.     “c void mod(vector a){...} void mod\_asm(vector a){...} void mod\_version\_x(vector a){...}