Projet C++

Rapport C++ / Réponses aux questions

Auteur :
Nasrallah Aghelias
Abdessamad Badaoui

20 mai 2023

Table des matières

Chapitre 1

LAB 2 : Introduction

1.1 Comparaison des performances des différentes collections

En C++, les collections sont des structures de données qui permettent de stocker et de manipuler des éléments de différentes manières. Voici une comparaison entre les différentes types de collections :

List : Une liste est une collection de données ordonnées qui peut être modifiée à tout moment en ajoutant ou en supprimant des éléments. Elle est implémentée sous forme de liste doublement chaînée. Les avantages de l'utilisation de la liste incluent une insertion et une suppression rapide des éléments au milieu de la liste. Cependant, l'accès à un élément de la liste se fait en parcourant la liste depuis le début, ce qui est moins efficace que l'accès aléatoire des autres collections.

Vector : Un vecteur est une collection de données dynamique qui peut être modifiée en ajoutant ou en supprimant des éléments à la fin du vecteur. Les éléments sont stockés dans un tableau contigu en mémoire, ce qui permet un accès rapide aux éléments aléatoires. Cependant, l'insertion ou la suppression d'un élément au milieu du vecteur peut nécessiter un décalage de tous les éléments suivants dans le tableau, ce qui peut être coûteux en termes de performances.

Set : Un ensemble est une collection de données non ordonnées qui ne contient pas de doublons. Les éléments sont stockés dans un ordre arbitraire, ce qui signifie qu'il n'y a pas de garantie sur l'ordre dans lequel ils sont stockés. Les avantages de l'utilisation d'un ensemble incluent l'accès rapide et la recherche des éléments, ainsi que la garantie qu'il n'y aura pas de doublons. Cependant, l'ensemble n'offre pas de fonctionnalités d'insertion ou de suppression rapides au milieu de l'ensemble.

Deque : Un deque est une double file d'attente (deque abréviation de "double-ended queue" en anglais) est une collection de données dynamique qui permet d'ajouter ou de supprimer des éléments à la fois au début et à la fin de la file d'attente. Les éléments sont stockés dans une séquence de blocs contigus en mémoire, ce qui permet un accès rapide aux éléments aléatoires. Les avantages de l'utilisation d'un deque incluent une insertion et une suppression rapide des éléments aux deux extrémités de la deque.

En utilisant la bibliothèque `chrono`, nous avons pu réaliser les courbes suivantes représentant les performances en termes de temps d'exécution des différents types de collections pour un nombre croissant de particules.

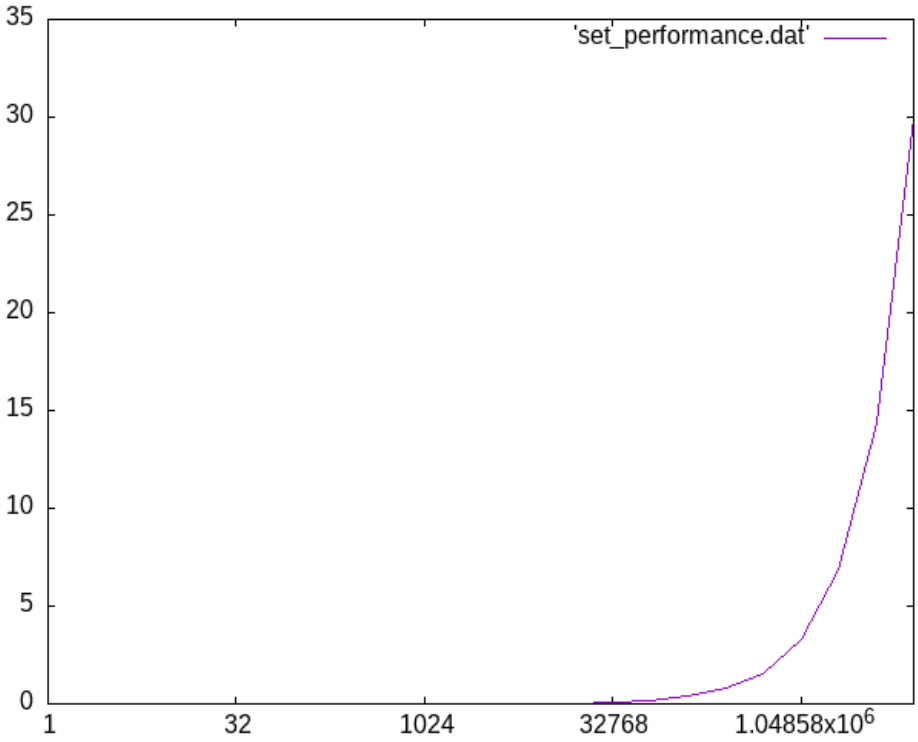


FIGURE 1.1 – **Collection Set** :Temps d'exécution (en secondes) en focntion de nombre de particules

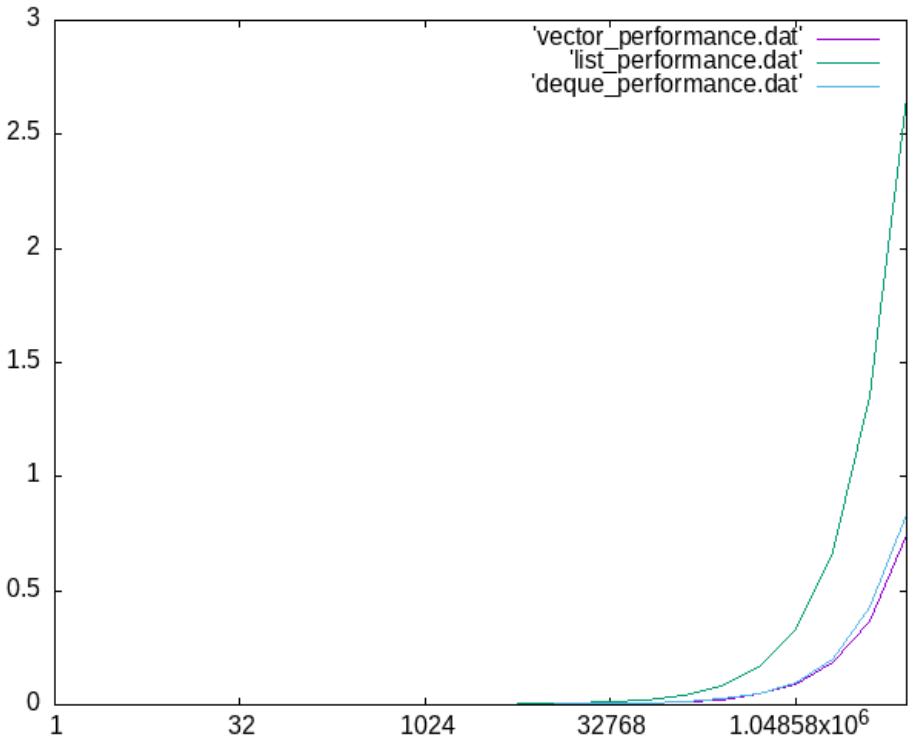


FIGURE 1.2 – **Collection Vector, Deque, et List** :Temps d'exécution (en secondes) en focntion de nombre de particules

la sélection d'une collection dépend de la situation et des exigences spécifiques du problème. Si la rapidité de l'insertion et de la suppression des éléments est une priorité, une liste ou une deque peut être une bonne option. Si l'accès aléatoire aux éléments est important, un vecteur peut être

préférable. Si la suppression de doublons est essentielle, un ensemble est une bonne option.

Dans notre cas, nous avons choisi de travailler avec un vector, car nous aurons un nombre important d'accès aléatoires aux éléments. Nous souhaitons donc que ces opérations soient aussi rapides que possible(avec une complexité de temps constante verb|O(1)|).

1.2 Méthode de Störmer-Verlet

Après avoir implémenté l'algorithme de Störmer-Verlet pour l'évolution du système de particules, nous avons pu vérifier notre implémentation et tracer l'évolution des différents corps du système gravitationnel proposé :

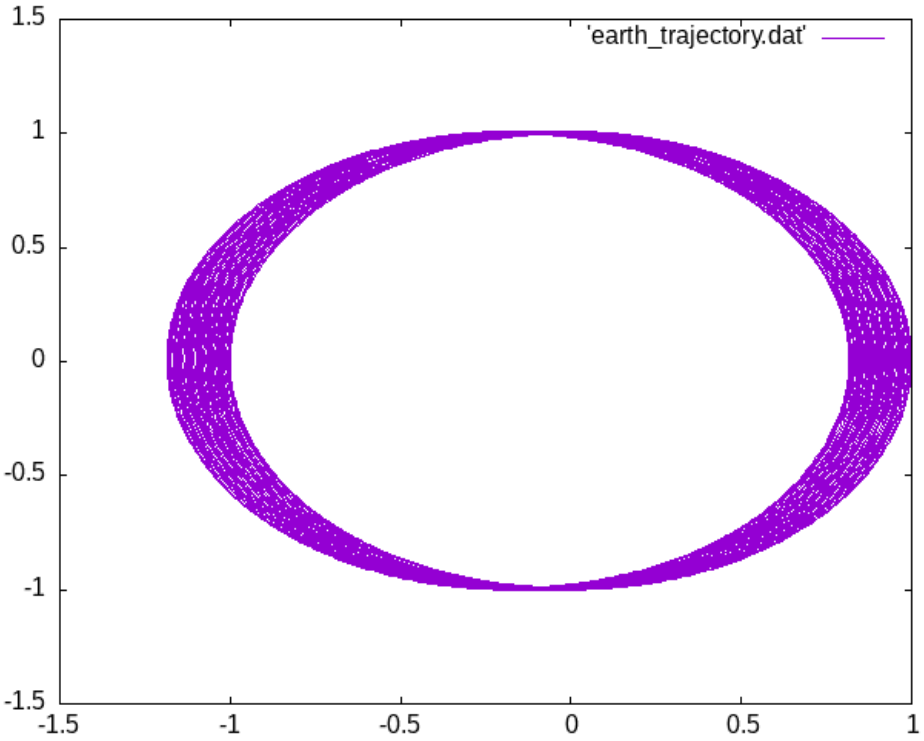


FIGURE 1.3 – **Earth trajectory**

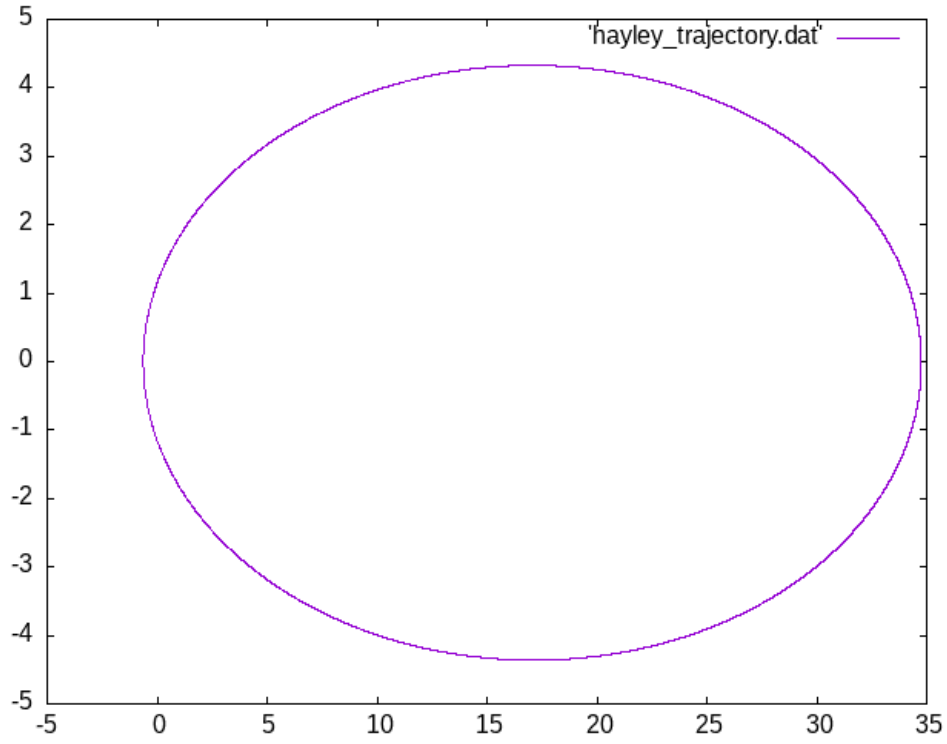


FIGURE 1.4 – **Hayley trajectory**

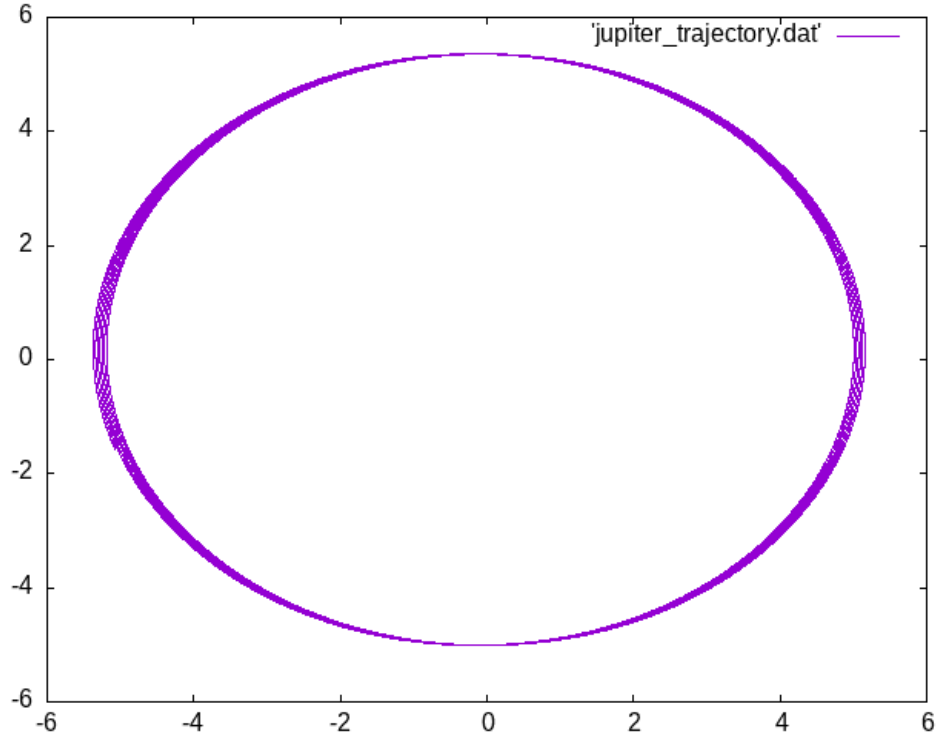


FIGURE 1.5 – **Jupiter trajectory**

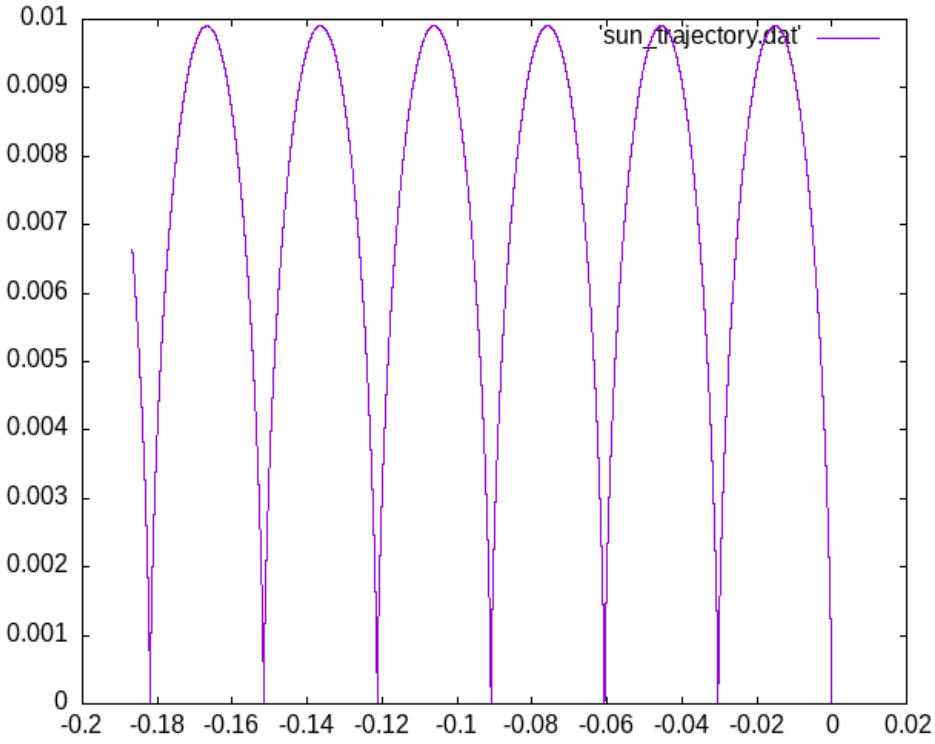


FIGURE 1.6 – **Sun trajectory**

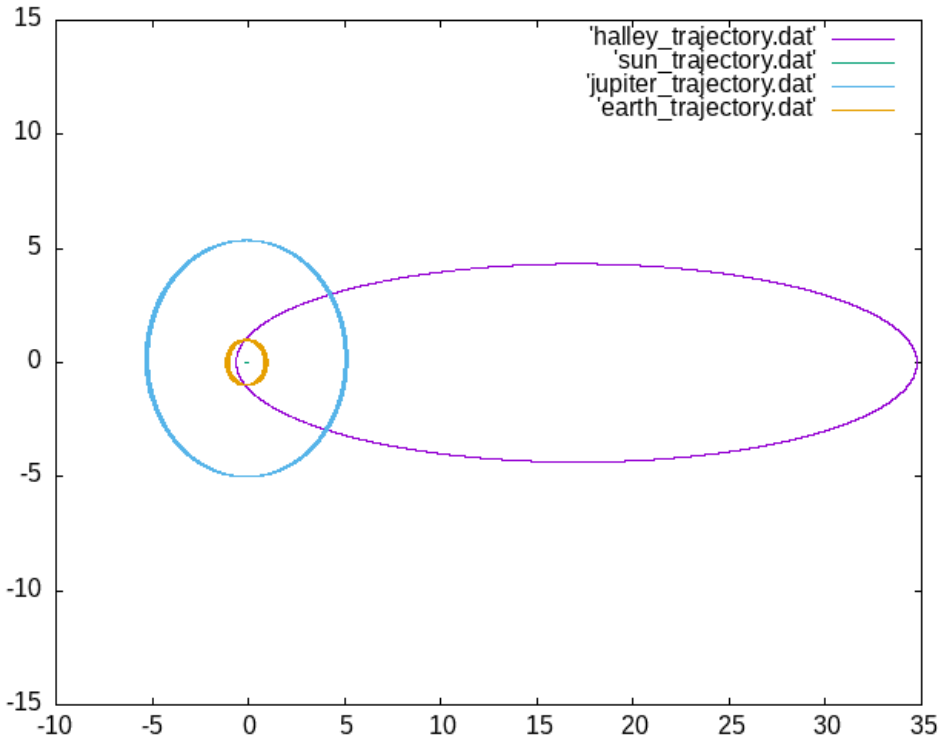


FIGURE 1.7 – **Solar system**

Chapitre 2

LAB 3 : Utilisation des opérateurs

2.1 La classe Univers

Dans notre cas, nous avons choisi de travailler avec un vector pour stocker nos particules, car nous aurons un nombre important d'accès aléatoires aux éléments. Nous souhaitons donc que ces opérations soient aussi rapides que possible(avec une complexité de temps constante $O(1)$).

2.2 Une modification pour diviser le temps de calcul par 2.

On peut diviser le temps de calcule par 2 en évitant les calculs répétés de la même force entre deux particules. Les forces sont calculées une fois et appliquées symétriquement aux deux particules impliquées. :

1 - Lorsque on parcoure les paires de particules pour calculer les interactions entre eux, on s'assure de ne les traiter qu'une seule fois. Par exemple, on peut parcourir les particules en utilisant deux boucles imbriquées, mais en s'assurant que la deuxième boucle ne commence qu'à partir de l'indice suivant de la première boucle.

2 - On calcule la force entre les deux particules une seule fois, en utilisant les formules appropriées pour notre modèle d'interaction.

3 - On applique la force calculée de manière symétrique aux deux particules.

2.3 Analyse de performances :

Le fichier `CMakeLists.txt` principale continet l'option de compilation `-pg` qui permet de faire mesurer les performances d'une implémentation. D'aprés l'analyse des résultats obtenus, on remarque que la modification effectuée a considérablement réduit le temps de calcul, le divisant par 2. En travaillant avec des pointeurs vers les particules, nous évitons de copier les objets de particules à chaque interaction. Au lieu de cela, nous utilisons des références mémoire vers ces objets, ce qui permet une gestion plus efficace de la mémoire. Les pointeurs nous permettent de manipuler directement les adresses des particules, évitant ainsi les surcharges dues à la copie des objets.

Cela réduit non seulement le temps de calcul, mais également les besoins en mémoire, car nous n'avons pas à stocker plusieurs copies des objets de particules. De plus, cela permet également de gérer dynamiquement les particules, en ajoutant ou en supprimant des particules pendant l'exécution de la simulation.

2.4 Autres simplifications

On peut utiliser une méthodes de troncature de potentiel : Si on utilise par exemple un potentiel de longue portée, tel que le potentiel de Lennard-Jones, on peut appliquer une méthode de troncature pour réduire la portée du potentiel. Cela signifie que on ve ignorer les interactions au-delà d'une certaine distance critique, car elles deviennent négligeables. Cela permet de réduire le nombre de calculs nécessaires (ce qui va être fait dans le lab suivant en combinaison avec les cellules).

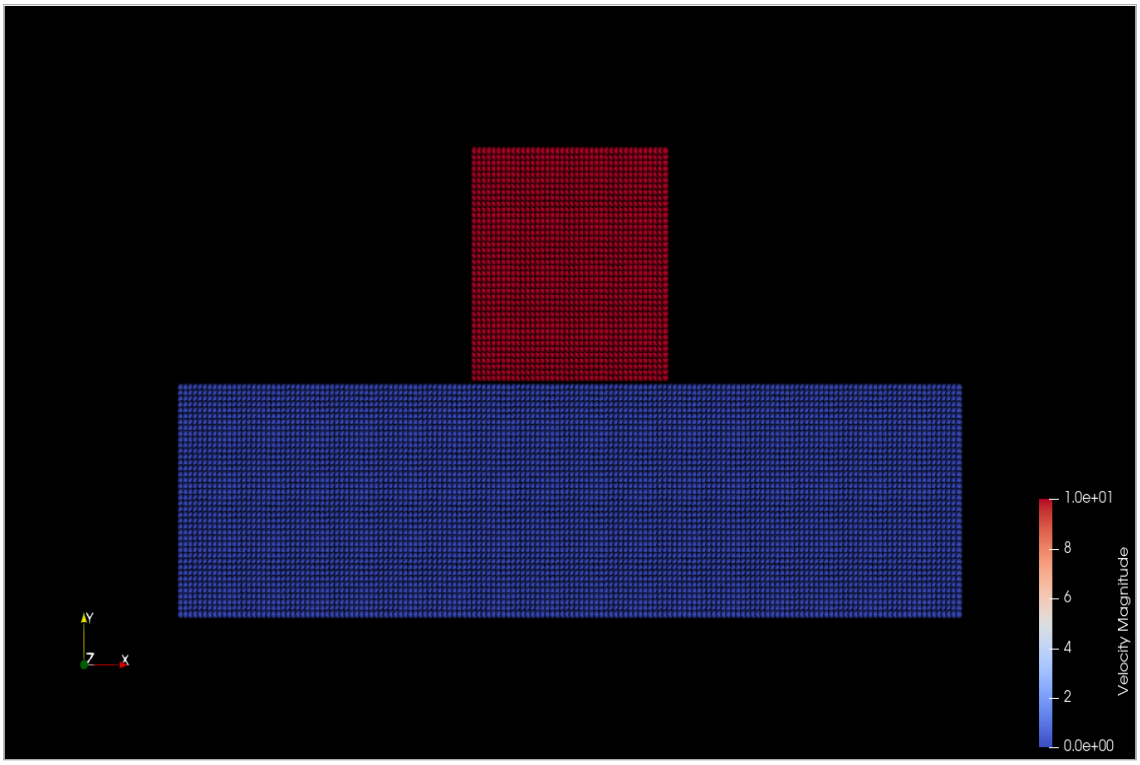
Chapitre 3

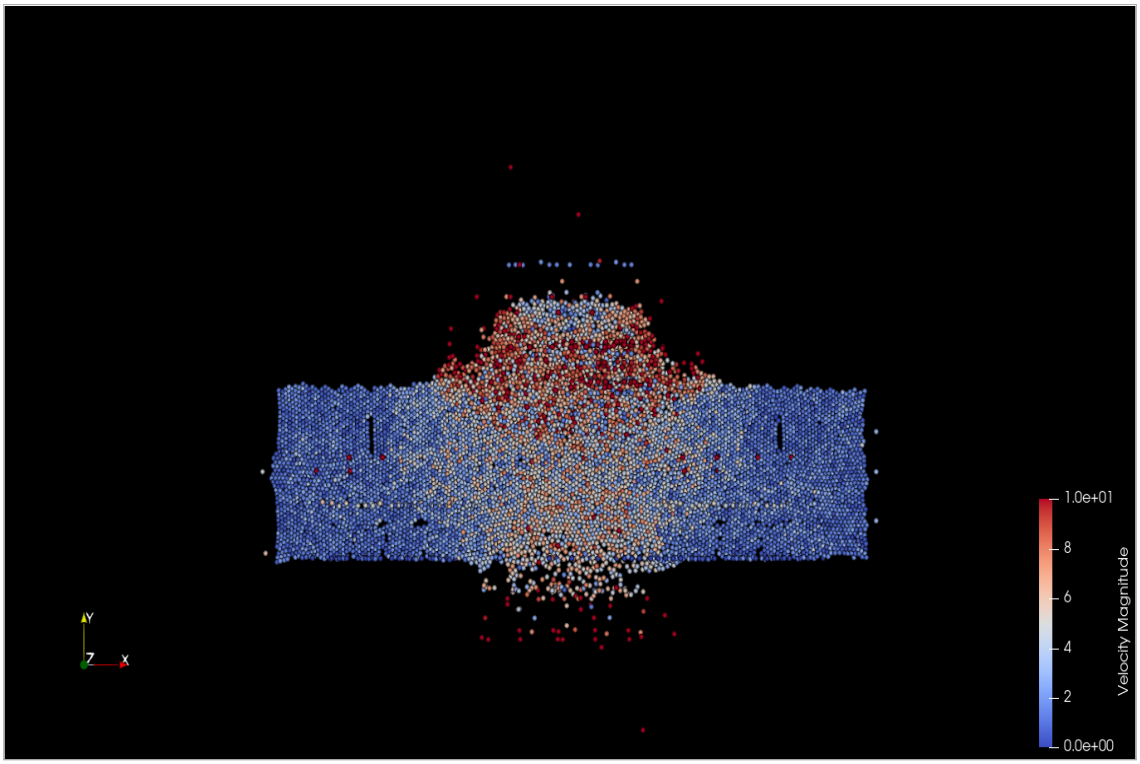
LAB 4 : Découpage de l'espace

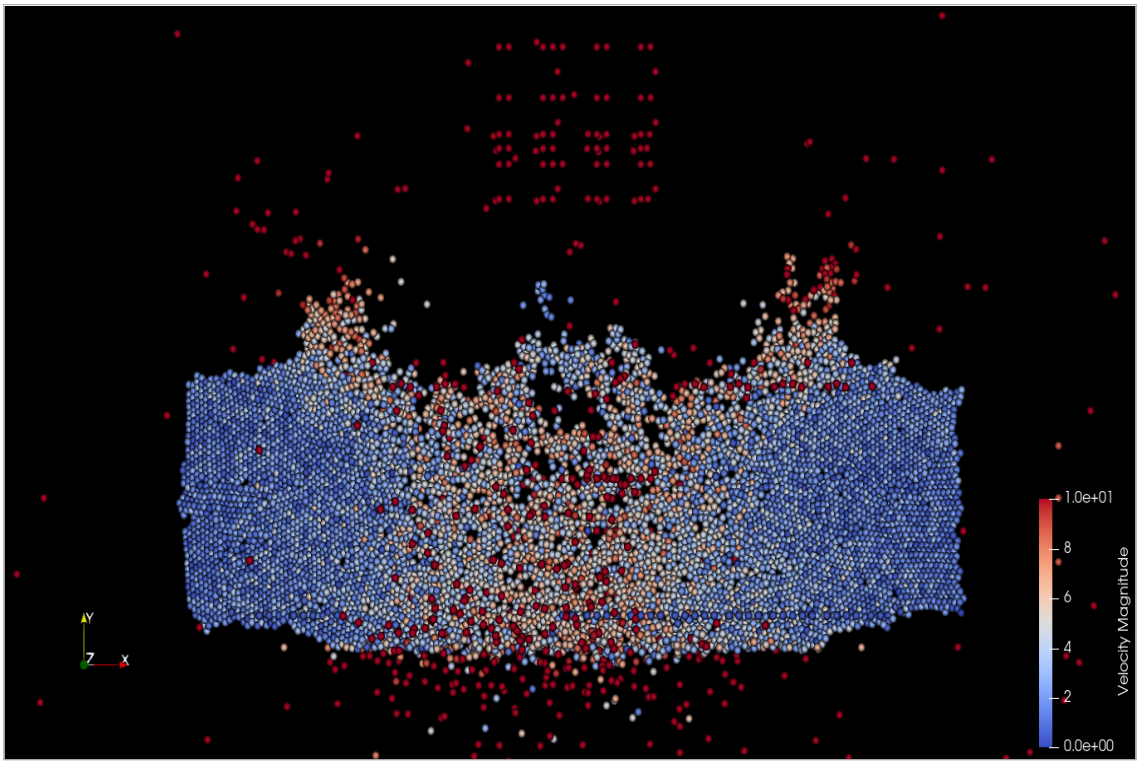
3.1 Application : Collision de deux objets

On considère les paramètres suivants :
$L_1 = 250$, $L_2 = 40$,
$\epsilon = 5$, $\sigma = 1$,
$m = 1$, $v = (0, 10)$,
$N_1 = 1600$, $N_2 = 6400$,
$r_{cut} = 2.5\sigma$, $\delta t = 0.00005$
avec v la vitesse initiale des particules du carré rouge. Le carré contient 40×40 particules équidistribuées et le rectangle contient 160×40 particules équidistribuées. La distance entre les particules est $2^{1/6}/\sigma$.

La simulation de la collision des deux objets en utilisant Paraview donne les résultats suivantes :




Y
Z
X
1.0e+01
8
6
4
2
0.0e+00
Velocity Magnitude


Y
Z
X
1.0e+01
8
6
4
2
0.0e+00
Velocity Magnitude

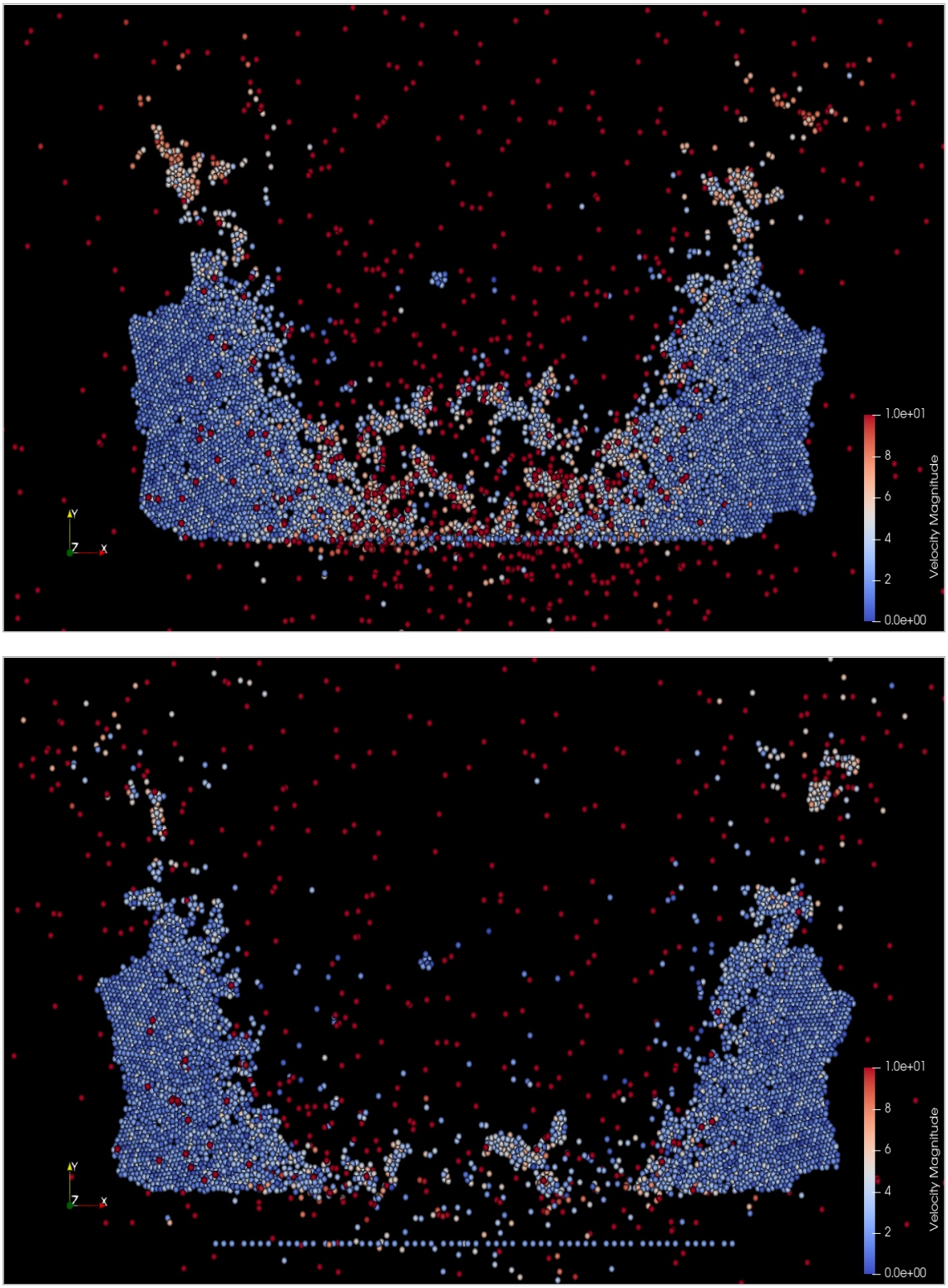


FIGURE 3.1 – **Différentes frames entre le début et la fin de la simulation**

Chapitre 4

LAB 5 : Test et visualisation

4.1 ACVL

4.1.1 Diagramme de cas d'utilisation.

Nous avons proposé le diagramme de cas d'utilisation suivant :

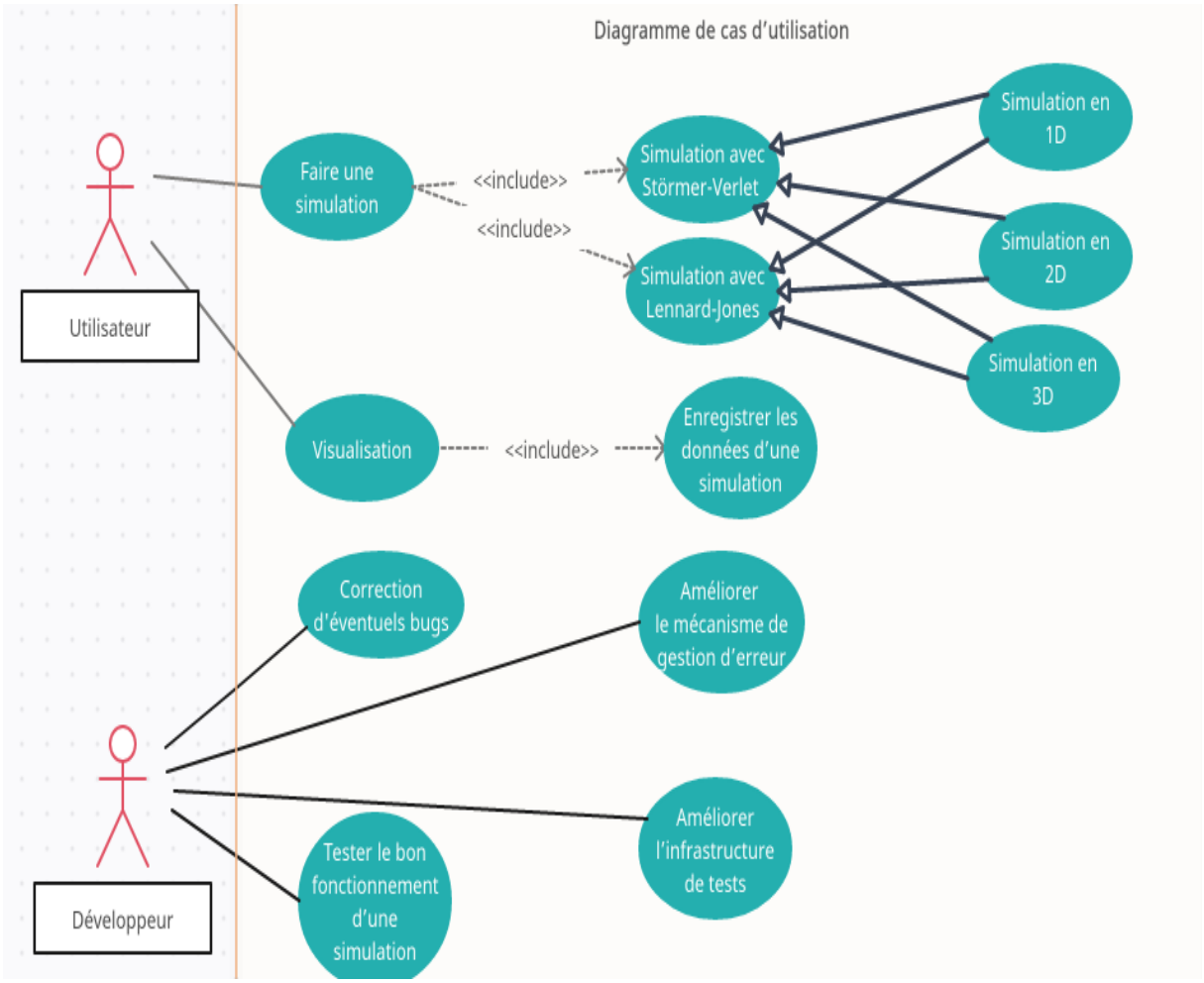


FIGURE 4.1 – **Diagramme de cas d'utilisation**

Le diagramme de cas d'utilisation décrit les principales fonctionnalités offertes par le système à la fois pour les utilisateurs et pour les développeurs. Voici une description des cas d'utilisation identifiés :

Faire une simulation (Utilisateur) : L'utilisateur peut utiliser cette fonctionnalité pour configurer les paramètres de la simulation, tels que la dimension et la méthode de simulation.

Visualiser les résultats (Utilisateur) : l'utilisateur peut visualiser les résultats de la simulation en enregistant les données de la simulation dans des fichiers vtk.

Tester les méthodes de simulation (Développeur) : Le développeur peut utiliser cette fonctionnalité pour effectuer des tests sur les différentes méthodes de simulation implémentées. Cela implique de configurer des scénarios de test spécifiques et de vérifier que les résultats obtenus sont conformes aux attentes.

Améliorer la gestion des erreurs (Développeur) : Le développeur peut travailler sur l'amélioration de la gestion des erreurs dans le système. Cela peut inclure l'identification, le suivi et la résolution des erreurs qui se produisent lors de l'exécution de la simulation. L'objectif est d'optimiser la fiabilité et la stabilité du système.

Corriger les bugs (Développeur) : Le développeur est responsable de la correction des bugs qui peuvent survenir dans le système. Cela implique l'identification des problèmes, la reproduction des erreurs, la recherche de la cause fondamentale et la mise en œuvre de solutions pour résoudre ces problèmes.

4.1.2 Diagramme de séquence

La figure suivante présente le diagramme de séquence correspondant au scénario d'utilisation du système :

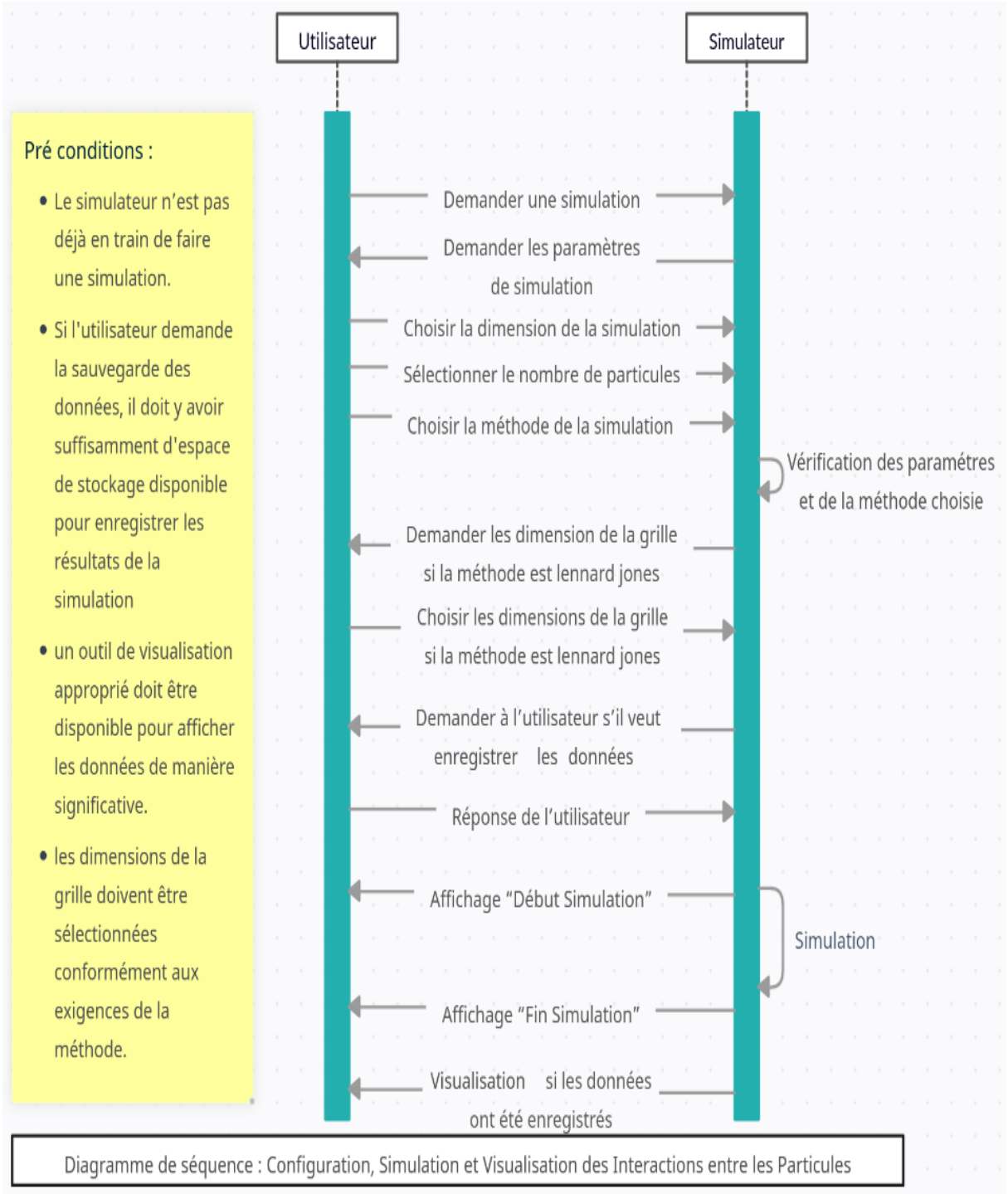


FIGURE 4.2 – **Diagramme de séquence**

Dans ce diagramme de séquence, nous avons l'Utilisateur qui interagit avec le Simulateur pour configurer la simulation. L'Utilisateur sélectionne la dimension de la simulation, la méthode de simulation à utiliser, les dimensions de la grille (si la méthode Lennard-Jones est choisie), et le nombre de particules. Ensuite le système demande à l'utilisateur s'il veut enregistrer les données de la simulation.

Le simulateur exécute ensuite la simulation en utilisant les paramètres configurés. Une fois la simulation terminée, une visulation des résultats est faite par un outil approprié (Paraview par exemple).

4.1.3 Diagramme de transition

Le diagramme de transition suivant représente les différents états et transitions de notre simulateur :

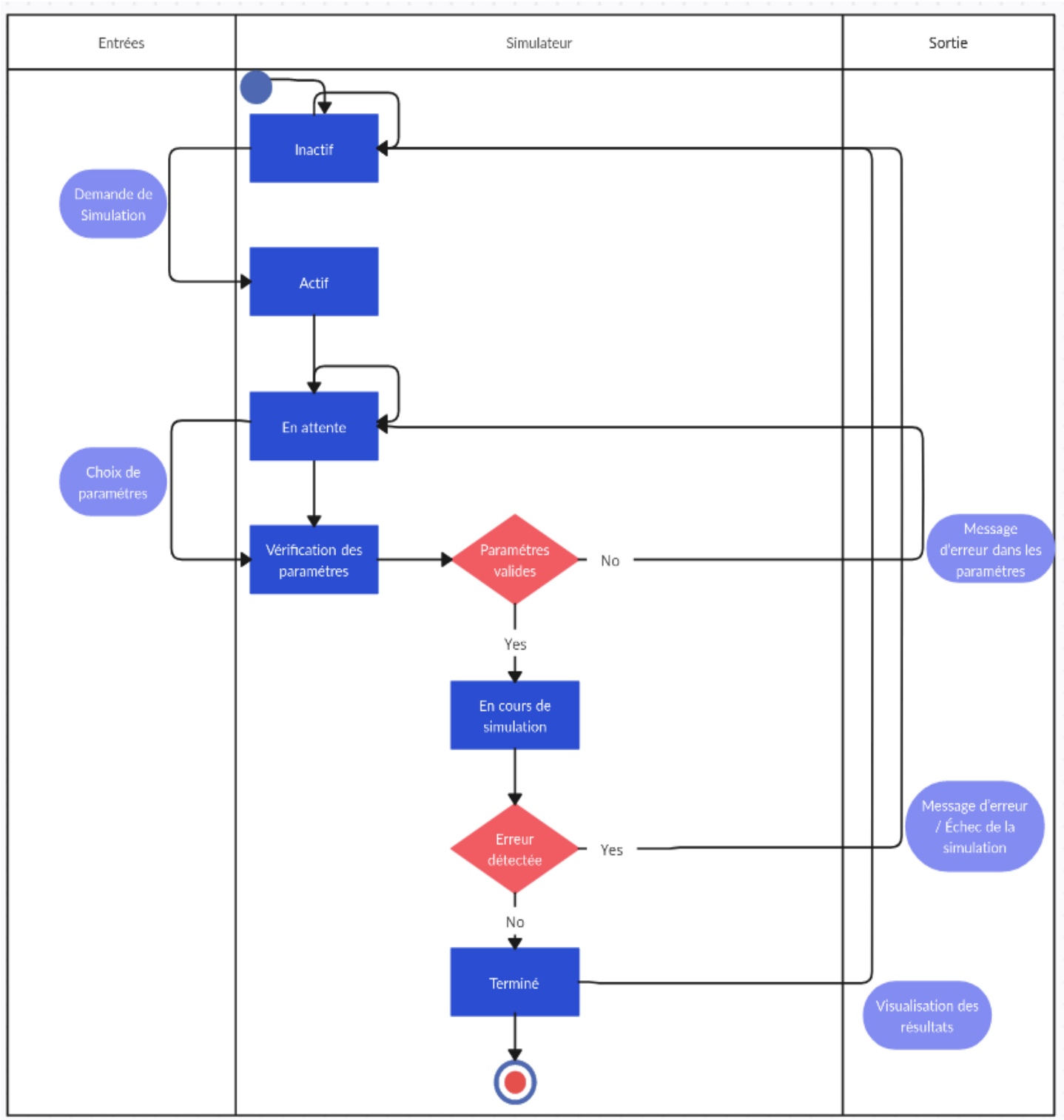


FIGURE 4.3 – **Diagramme de transition**

Dans ce diagramme d'état, nous avons les états suivants :

Inactif : C'est l'état initial du système où il est inactif et attend des événements.

Actif : Le système passe à cet état lorsqu'il reçoit l'événement "Demande de la simulation" depuis l'utilisateur. Il est alors prêt à effectuer la simulation.

En attente : Cet état représente une attente du système. Il attend de nouvelles entrées qui correspondent aux paramétres nécessaires pour faire la simulation.

Vérification des paramétres : Durant cette état, le système vérifie la cohérence des paramétres. Si les paramétres ne sont pas valides on revient vers l'état "En attente" en renvoiant un message d'érreur à l'utilisatuer pour qu'il rentre à nouveau les paramétres.

En cours de Simulation : Le système est dans cet état lorsqu'il effectue la simulation. Si une erreur est détectée, un message d'erreur/ échec de la simulation est affiché à l'utilisateur et le système passe à son état initiale "inactif".

Terminé : Le système passe à cet état lorsque la simulation est terminée et il est prêt à visualiser les résultats.

4.1.4 Diagramme de classes d'analyse

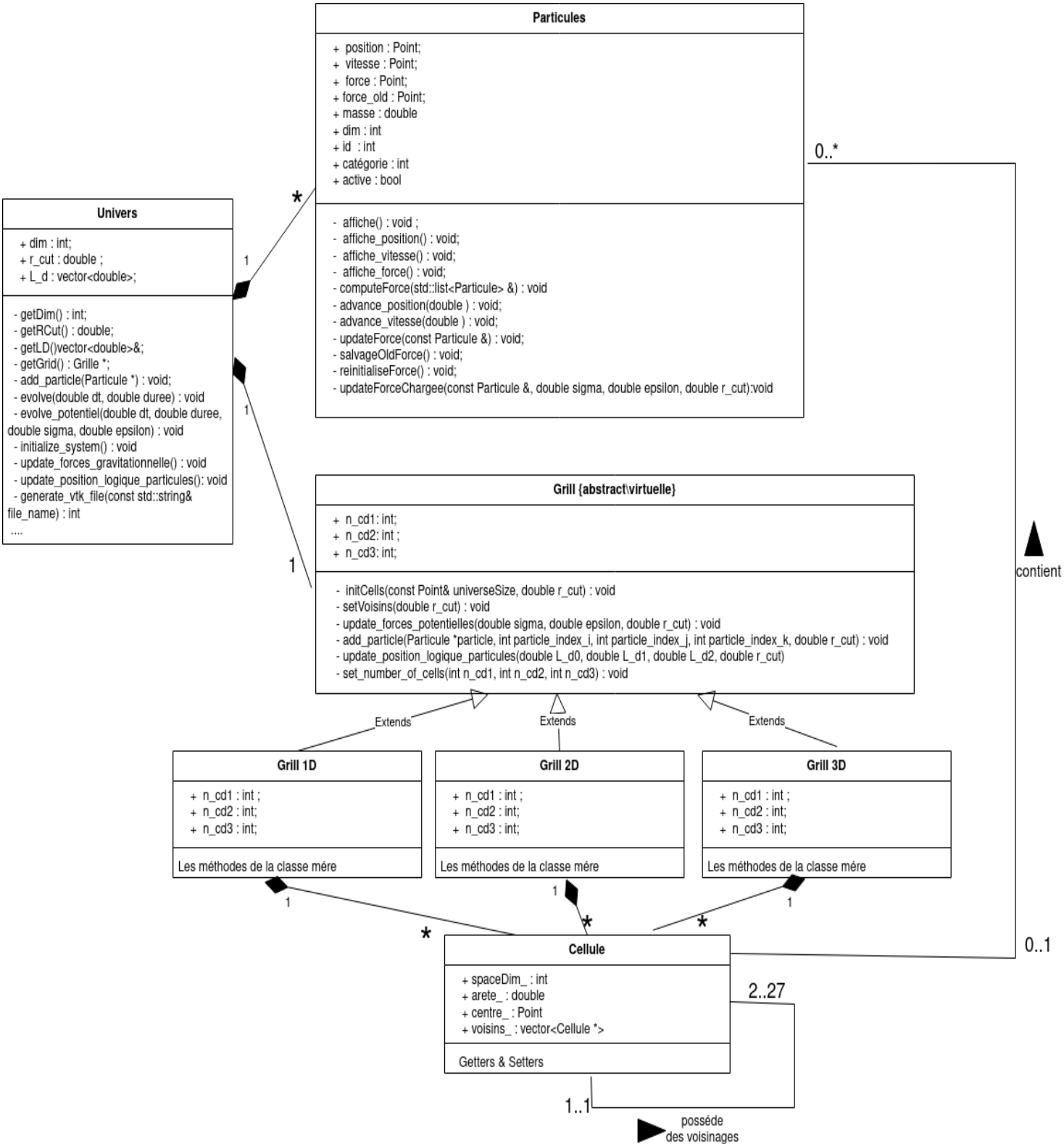


FIGURE 4.4 – **Diagramme de classe d'analyse**

Les particules et la grille sont des entités faibles de l'univers. Chaque univers est constitué d'une

seul grille et d'un ensemble de particules, qui peut éventuellement être vide. Les classes Grille 1D, Grille 2D et Grille 3D étendent la classe virtuelle Grille. La classe Cellule est une entité faible des différentes classes de grille, ce qui signifie qu'une cellule ne peut pas être définie sans une grille. De plus, chaque cellule a entre 2 (pour une cellule située sur le bord dans une grille 1D) et 27 (nombre maximal de voisinages pour une cellule dans une grille 3D) voisinages. Enfin, chaque cellule possède entre 0 et n particules, et chaque particule peut soit être incluse dans une cellule, soit ne pas en faire partie (cela dépend de la manière dont la gestion des bords a été réalisée : dans le cas de l'absorption, les particules qui sortent de l'univers n'appartiennent plus à une cellule, tandis que dans les autres types de gestion des bords, une particule appartient toujours à une cellule spécifique).

Chapitre 5

Lab 6 : Raffinement du modèle

5.1 Conditions aux limites réflexives

La comparaison entre les deux méthodes de conditions aux limites réflexives :
Paroi réfléchissante symétrique :
-La particule rebondit de manière symétrique par rapport à la paroi.
-Les composantes de la vitesse perpendiculaires à la paroi changent de signe lors du rebond.
-Cette méthode est simple à implémenter et ne nécessite pas de calcul supplémentaire.
-Elle simule une collision élastique avec la paroi, où l'énergie cinétique est conservée.
Conditions aux limites réflexives avec potentiel :
-Une force est appliquée aux particules lorsqu'elles s'approchent de la paroi.
-La force devient de plus en plus grande à mesure que la particule se rapproche de la paroi.
-Cette méthode permet de modéliser des interactions plus complexes avec la paroi, en utilisant un potentiel spécifique.
-Elle peut simuler des forces de répulsion, des forces magnétiques ou d'autres interactions particule-paroi.
-Elle nécessite des calculs supplémentaires pour déterminer la force en fonction de la distance entre la particule et la paroi.

Choisir entre ces deux méthodes dépend des objectifs de la simulation et des comportements qu'on souhaite reproduire. Si on souhaite simplement simuler des rebonds élastiques et que la symétrie par rapport à la paroi est suffisante, la paroi réfléchissante symétrique est généralement utilisée en raison de sa simplicité. En revanche, si on a besoin de modéliser des forces plus complexes ou des interactions spécifiques avec la paroi, les conditions aux limites réflexives avec potentiel offrent une plus grande flexibilité.

5.2 Application : collision de deux objets

On considère les paramètres suivants :
$L_1 = 250$, $L_2 = 180$,
$\epsilon = 1$, $\sigma = 1$,
$m = 1$, $v = (0, 10)$,
$N_1 = 395$, $N_2 = 17227$,
$r_{cut} = 2.5\sigma$, $\delta t = 0.00005$
G = -12, $E_c^D = 0.005$

La simulation de la collision des deux objets en utilisant Paraview donne les résultats suivantes (on a décidé de travailler avec une vitesse un peu plus élevée puisque la simulation prend beaucoup de temps) :

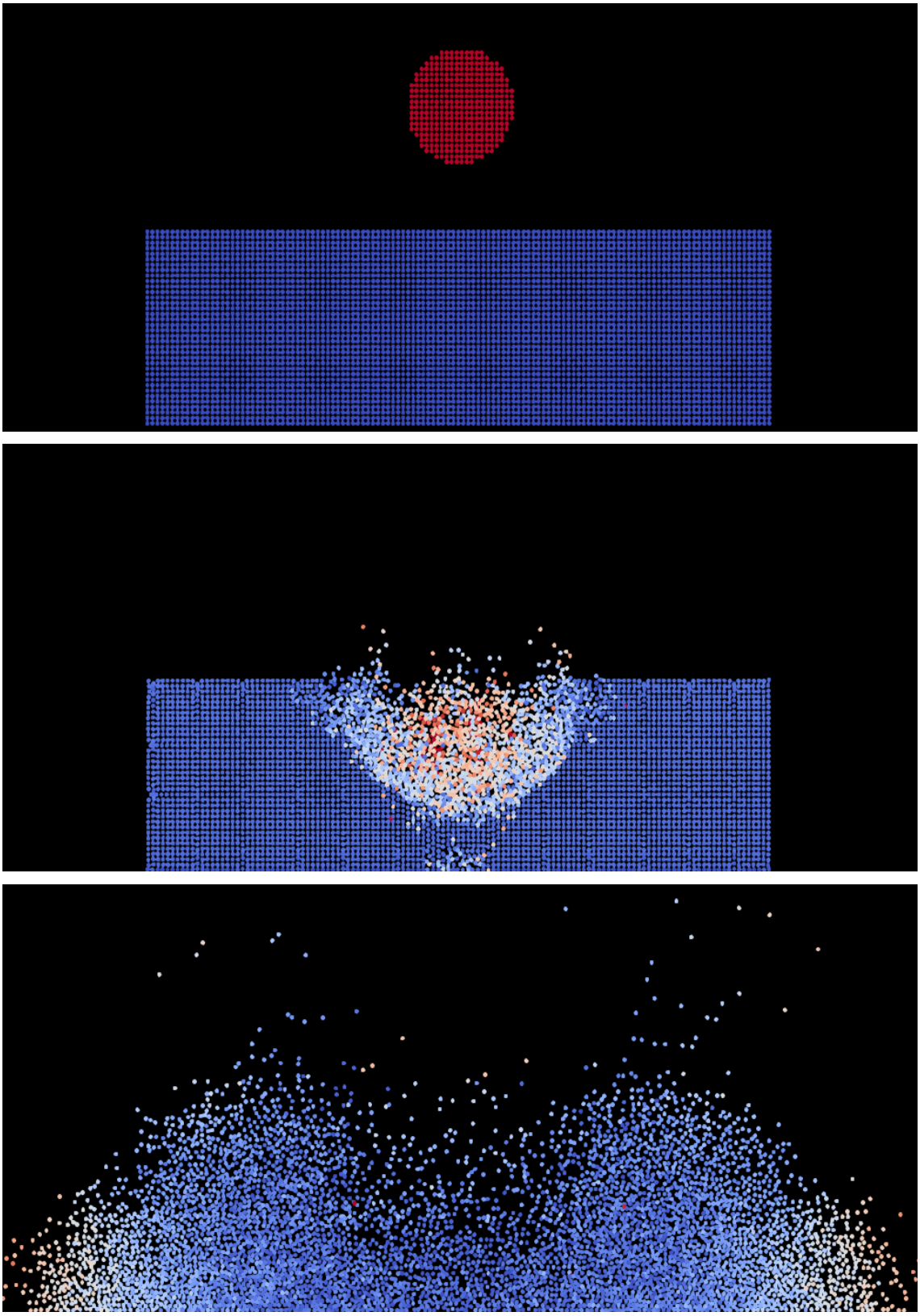


FIGURE 5.1 – **Différentes frames entre le début et la fin de la simulation**