2D ELASTIC COLLISION SIMULATION USING PYTHON AND NVIDIA WARP

Table des matières

Table des figures

---

1. Introduction

Le but de ce projet était de simuler des collisions élastiques 2D entre plusieurs particules en utilisant Python. La simulation devait être animée, efficace et, éventuellement, hébergée sur un serveur local afin d'être accessible depuis des appareils du même réseau. Ce projet constituait à la fois un outil pédagogique et un défi technique, permettant d'approfondir la compréhension des simulations physiques, de la programmation GPU et de l'hébergement web.

Le programme a été divisé en trois parties :

1. Implémentation de la physique pour les collisions et le mouvement,
2. Animation des résultats avec Matplotlib,
3. Hébergement facultatif de la simulation sur un serveur local.

Nous nous sommes assurés que la simulation respectait les principes physiques, tels que la conservation de la quantité de mouvement et de l'énergie.

Bien que simplifiée (2D uniquement, collisions idéalisées, particules circulaires), la simulation illustre les concepts physiques fondamentaux utilisés dans l'ingénierie, le développement de jeux et la visualisation de données.

2. Materials

Élément	Description	Justification
Environnement de développement	Python, VS Code	Standard pour écrire et déboguer du code efficacement
Bibliothèque de simulation	Nvidia Warp	Exécution performante de kernels, fonctions vectorielles et de collisions utiles
Bibliothèque de rendu	Matplotlib	Traçage 2D et animation simple et efficace
Méthode d'installation	pip (via PyPi)	Méthode standard et fiable pour installer les packages Python
Matériel	Ordinateurs scolaires et personnels	Simulation légère fonctionnant sur la plupart des CPU ; même d'anciens PC peuvent héberger des serveurs
Contrôle de version	Git et GitHub	Suivi du développement et collaboration facilitée

Figure 2 : Main materials used in the simulation.

Python a été choisi pour sa simplicité et le support des bibliothèques. Nvidia Warp offre des calculs vectorisés et la gestion des threads, cruciaux pour les simulations. Matplotlib a été choisi pour le rendu grâce à sa facilité d'utilisation, bien qu'il soit moins performant que l'OpenGLRenderer de Warp.

---

3. Recherches

Les premières recherches ont porté sur les moteurs physiques et les techniques de détection des collisions. Nous avons examiné comment modéliser des collisions élastiques réalistes en utilisant les principes de conservation de la quantité de mouvement et de l'énergie cinétique.

La compréhension de l'architecture de Nvidia Warp était essentielle. Nous avons exploré son fonctionnement, la gestion des kernels (souvent accélérés par GPU), des threads et des calculs vectoriels. Bien que l'accélération GPU n'ait pas été disponible sur les ordinateurs scolaires, apprendre Warp nous a permis d'écrire du code efficace sur CPU.

OpenGLRenderer a été envisagé initialement, mais des problèmes de compatibilité (notamment avec pyglet) ont rendu son utilisation impossible. Nous avons donc opté pour Matplotlib pour le rendu, en nous basant sur la documentation et des exemples.

Nous avons également exploré la fonction enumerate() de Python pour itérer efficacement sur les objets, utile pour généraliser la simulation à un nombre quelconque de particules.

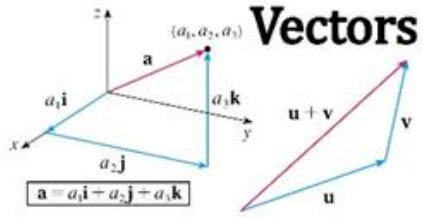

Figure 3 : Example of vector geometry used for collisions.

4. Résolution de problèmes

Plusieurs problèmes techniques et conceptuels ont été rencontrés pendant le développement :

- **Problèmes d'installation des bibliothèques** : restrictions sur les ordinateurs scolaires, résolu par la création d'un environnement virtuel.
- **Erreur "No CUDA Device" :** Warp nécessitait un GPU, indisponible sur le matériel ancien. Solution : forcer l'usage du CPU.
- **Crash "Illegal Instruction" :** dû à des instructions CPU non supportées. Solution : remplacer la machine par un PC plus récent.
- **Incompatibilité GPU :** Warp dépend de CUDA pour les performances complètes, indisponible sur le matériel scolaire. Warp a été exécuté en mode CPU.
- **Échecs de rendu OpenGL :** erreurs pyglet empêchant l'utilisation de OpenGLRenderer. Matplotlib a été utilisé à la place. *animation*.

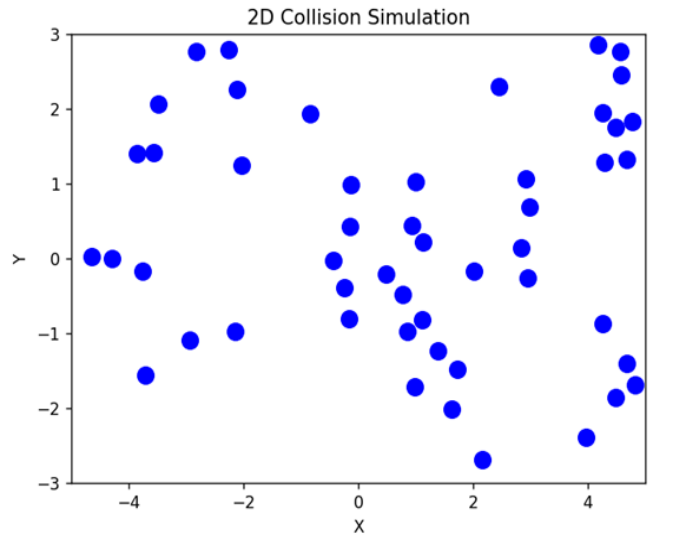

Figure 4 : Screenshot of Matplotlib

- **Fausse détection de collisions** : après une collision initiale, les particules se chevauchaient parfois. Correction : avancer légèrement le temps et vérifier la distance future entre particules.

Figure 5 : Code snippet preventing false collision detection.

```
diff = pos[i] - pos[j]
dist = wp.length(diff)

#Compute distance between points after a small time nudge to avoid false positives for collision
dist_nudge=wp.length(diff+(vel[i]-vel[j])*wp.float(dt/1000.0) )
```

5. Planification et calendrier du projet

Plan Initial :

- Semaines 1–2 : implémentation de la simulation et de la physique.
- Semaines 3–4 : rendu et animation.
- Semaines 5–6 : expérimentation de l'hébergement serveur et tests.

Modifications du plan :

Plus de temps a été consacré au débogage de la compatibilité matériel/logiciel, notamment lié à Warp et aux CPU anciens. La décision de faire tourner le serveur sur un ancien PC a réorienté les priorités, passant du rendu avancé à la faisabilité du déploiement. Le projet a été alors divisé entre simulation (Niklas) et hébergement (moi), permettant un développement concentré.

6. Éthique, sécurité et précautions

Nous avons été prudents concernant le code et les installations externes. Les extraits de code tiers n'ont été utilisés que pour les tests. Le code final a été écrit de zéro et vérifié.

L'installation des packages s'est faite via PyPi et a été limitée à des bibliothèques populaires et bien maintenues pour réduire les risques de sécurité. Nous savons que PyPi n'est pas complètement vérifié (Stack Overflow, 2017), donc seuls des packages fiables ont été utilisés.

Enfin, nous reconnaissons que la simulation n'est pas physiquement parfaite : espace 2D uniquement, particules circulaires, absence de frottement ou de forces autres que la collision, impacts parfaitement élastiques. C'est un projet éducatif, non adapté à l'ingénierie réelle sans améliorations importantes.

---

7. Conclusion

Nous avons construit avec succès une simulation fonctionnelle de collisions élastiques 2D avec un nombre arbitraire de particules. Bien que la portée du projet ait été réduite par rapport au rendu 3D et à l'hébergement serveur, nous avons atteint nos objectifs principaux. La simulation fonctionne de manière réaliste dans les hypothèses idéalisées, et l'animation montre clairement les interactions entre particules.

Nous avons beaucoup appris sur la simulation physique, le calcul basé sur les kernels, l'animation Python et les défis d'installation sur systèmes limités. Les améliorations futures pourraient inclure des collisions non élastiques, des visuels plus réalistes, l'interaction utilisateur et des extensions 3D.

Nous sommes fiers du résultat et des connaissances acquises grâce à la résolution de problèmes et au travail en équipe.

Bibliographie

Community, W. (n.d.). *Momentum*. Retrieved from https://en.wikipedia.org/wiki/Momentum

Gregorius, D. (2014). *Robust Contact Creation for Physics Simulations*. https://winter.dev/articles/physics-engine/DirkGregorius_Contacts.pdf

Matplotlib Development Team. (n.d.). *Matplotlib*. Retrieved May 13, 2025, from https://matplotlib.org/

NVIDIA. (n.d.). *Warp Core Tutorial: Basics*. Retrieved from https://github.com/NVIDIA/warp/blob/main/notebooks/core_01_basics.ipynb

NVIDIA. (n.d.). *Warp Core Tutorial: Points*. Retrieved from https://github.com/NVIDIA/warp/blob/main/notebooks/core_03_points.ipynb

Python Community. (2025). *PyPi*. Retrieved from https://pypi.org/project/pip/

Z, D. (2017). *Stack Overflow: Are PIP packages curated? Is it safe to install them?* https://stackoverflow.com/questions/38236366/are-pip-packages-curated-is-it-safe-to-install-them

ChatGPT

Ubuntu Forums

Reddit

Quora