2D Elastic Collision Simulation Using Python and Nvidia Warp

Table of Contents

Table of Figures

1. Introduction

The aim of this project was to simulate 2D elastic collisions between multiple particles using Python. The simulation had to be animated, efficient, and optionally hosted on a local server to make it accessible from devices on the same network. This project was both an educational tool and a technical challenge that gave insight into physics simulations, GPU programming, and web hosting.

The program was split into three parts: (1) implementing physics for collisions and motion, (2) animating the results using Matplotlib, and (3) hosting the simulation on a local server (optional feature). We focused on ensuring the simulation respected physical principles such as conservation of momentum and energy.

Although simplified (2D only, idealized collisions, and circular particles), the simulation reflects core physics concepts used in fields like engineering, game development, and data visualization.

2. Materials

Item	Description	Justification
Development Environment	Python, VS Code	Standard for writing and debugging code efficiently
Simulation Library	Nvidia Warp	High-performance kernel execution, useful vector and collision functions
Rendering Library	Matplotlib	Simple and effective 2D plotting and animation
Installation Method	pip (via PyPi)	Standard and trusted method for installing Python packages
Hardware	School and personal computers	Simulation is lightweight and runs on most CPUs; older PCs can host servers
Version Control	Git and GitHub	Tracked development and facilitated collaborative editing

Figure 2: Main materials used in the simulation.

Python was chosen for its simplicity and library support. Nvidia Warp provided vectorized math and thread support, crucial for simulations. Matplotlib was selected for rendering due to ease of use, despite being less powerful than Warp's OpenGLRenderer.

---

3. Research

The initial research focused on physics engines and collision detection techniques. We examined how to model realistic elastic collisions using principles like conservation of momentum and kinetic energy.

Understanding Nvidia Warp's architecture was key. We explored how it runs kernels (often GPU-accelerated), manages threads, and handles vector calculations. Although we couldn't fully utilize GPU acceleration on school computers, learning Warp's functionality helped us write efficient code using CPU.

OpenGLRenderer was initially considered, but compatibility issues with dependencies (notably pyglet) made it unfeasible in our environment. We instead switched to Matplotlib for rendering, which we learned via documentation and examples.

We also explored the Python enumerate() function to loop through objects more effectively—a helpful tool for generalizing simulations to any number of particles.

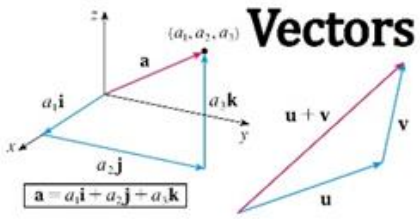

Figure 3: Example of vector geometry used for collisions.

---

4. Problem Solving

Several technical and conceptual issues arose during development:

- **Library Installation Issues:** Initially faced restrictions on school computers; solved by creating a virtual environment.
- **No CUDA Device Error:** Nvidia Warp required GPU access, which older hardware lacked. Workaround: forced CPU usage.
- **Illegal Instruction Crash:** Due to unsupported CPU instructions. We replaced the machine with a newer PC.
- **GPU Incompatibility:** Nvidia Warp depends on CUDA for full performance, which wasn't available on school hardware. We ran Warp in CPU mode.
- **OpenGL Rendering Failures:** Pyglet errors halted the use of OpenGLRenderer. We shifted to Matplotlib for robust, if basic, animation.

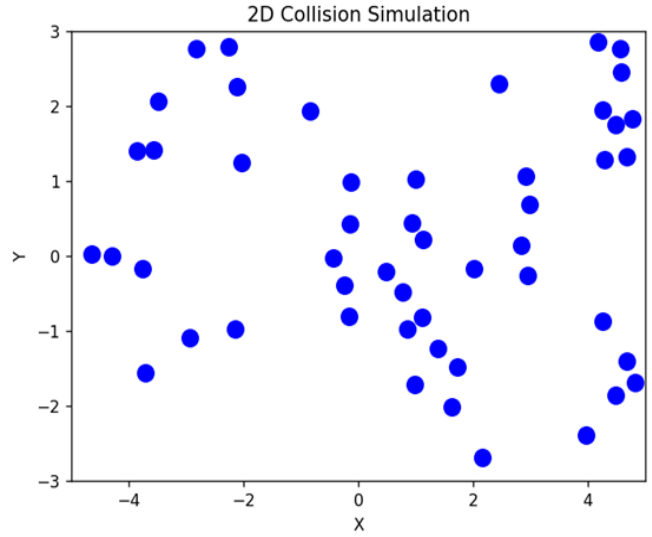

Figure 4: Screenshot of Matplotlib animation.

- **False Collision Detection:** After initial collision, particles occasionally overlapped again due to insufficient displacement. This was fixed by advancing time slightly and checking future distance between particles.

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. Project Timeline and Planning

Initial Plan:

- Week 1–2: Basic simulation and physics implementation.
- Week 3–4: Rendering and animation.
- Week 5–6: Experimental server setup and testing.

Changes to Plan:

More time was spent debugging hardware/software compatibility, especially related to Warp and older CPUs. The decision to run the server on an old PC became central to the project, shifting priorities from advanced rendering to deployment feasibility. The project was then divided between simulation (Niklas) and hosting (me), which allowed for focused development.

6. Ethics, Safety and Security

We were cautious about external code and software installations. Any third-party snippets were only used during testing. Final code was written from scratch and verified.

Package installation was done via PyPi and limited to popular, well-maintained libraries to reduce security risks. We understand PyPi is not curated (Stack Overflow, 2017), so we only used trusted packages.

Finally, we recognize that our simulation is not physically perfect. It assumes 2D space, circular particles, no friction or forces apart from collision, and perfectly elastic impacts. It is an educational project—not suitable for real-world engineering without extensive improvements.

---

7. Conclusion

We successfully built a working simulation of 2D elastic collisions with arbitrary numbers of particles. While the project's scope was scaled down from 3D rendering and server hosting, we achieved our core goals. The simulation performs realistically within ideal assumptions, and the animation clearly shows particle interactions.

We learned a lot about physics simulation, kernel-based computation, Python animation, and installation challenges on limited systems. Future improvements could include non-elastic collisions, more realistic visuals, user interaction, and 3D extensions.

We're proud of the result and the knowledge gained through problem solving and teamwork.

Bibliography

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