Final Report: Simulation of Spacecraft Aerodynamics Using Particle Dynamics

Abstract:

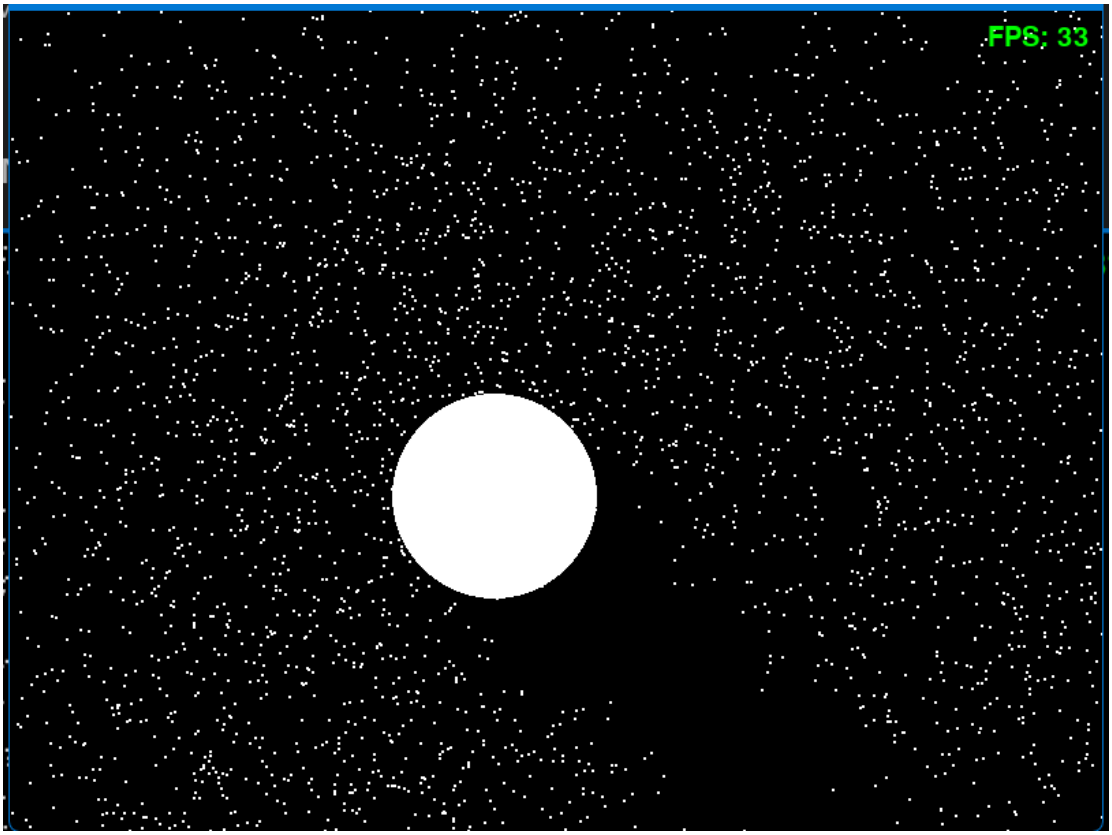
This report presents a detailed account of a collaborative programming project that simulates the aerodynamic interactions of air particles with a spacecraft. The simulation models the behavior of particles as they encounter a spacecraft, represented by a square obstacle, and the resultant aerodynamic forces.

Introduction:

The simulation was developed using Python and the Pygame library, creating a visual and interactive representation of air particles colliding with a spacecraft. The SquareObstacle class, representing the spacecraft, is central to the simulation, providing the framework for collision detection and response.

Methodology: SquareObstacle Class and Collision Detection

We started with the circle shape first because it's easier.

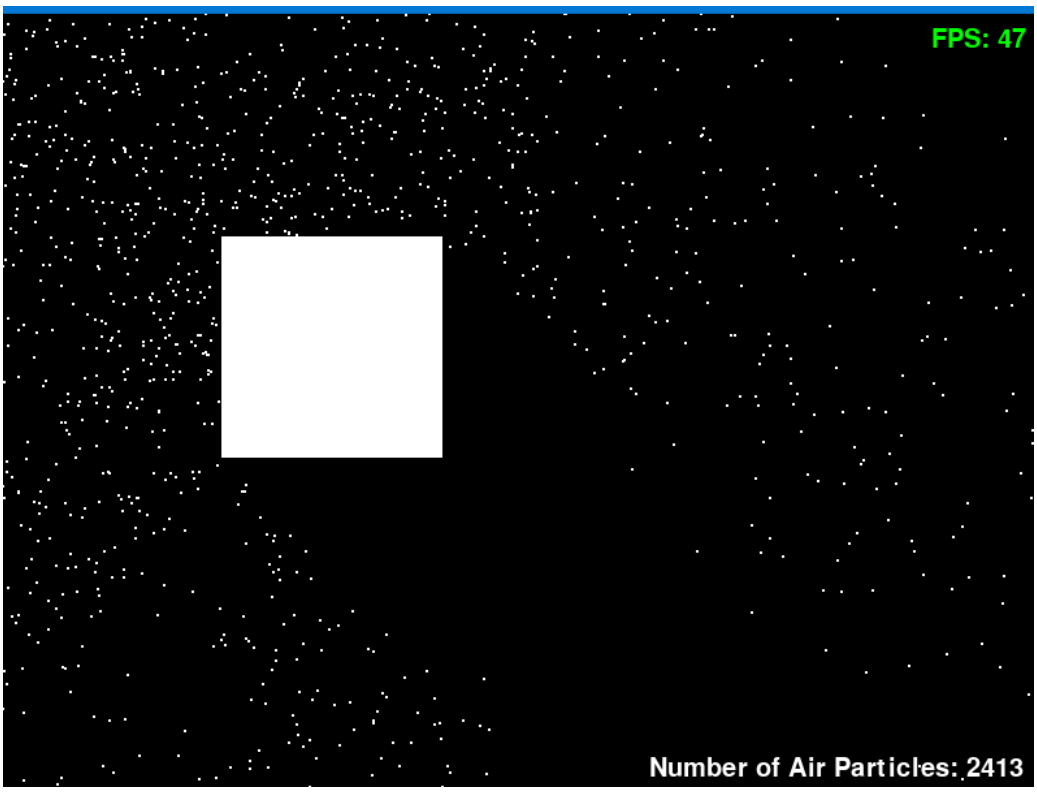


then we went for "Square" object with the difficulty of edges

The **SquareObstacle** class serves as a fundamental building block in our simulation. Here's how it works:

1. **Instantiation and Parameters**:
    - The **SquareObstacle** class is instantiated with essential parameters: **size**, **mass**, **initial_position**, and **velocity**.
    - These parameters define the characteristics of the spacecraft (or obstacle) within the simulation.
    - For instance, **size** determines the dimensions of the square obstacle, while **mass** represents its mass.

2. **Collision Detection**:
    - The **check_collision** method plays a crucial role in our simulation.
    - It determines whether a collision has occurred between an **air particle** (representing the surrounding air) and the spacecraft.
    - If the air particle enters the bounds of the square obstacle, a collision is detected.



3. **Impulse Calculation**:

- Upon collision, we calculate the resulting **impulse** based on the **conservation of momentum**.
- The impulse represents the change in momentum due to the collision.
- The formula for impulse is **Impulse (J) = Change in Momentum (Δp)**.

4. **Concepts and Equations**
    1. **Newtonian Mechanics:** The simulation adheres to the principles of Newtonian mechanics, particularly Newton's second law of motion: F = ma (force equals mass times acceleration). Here, the force is represented by the impulse delivered during a collision.
    2. **Coefficient of Restitution (COR):** Eq. 1 used in the `check_collision` methods calculates the impulse scalar based on the concept of coefficient of restitution (COR) [1]. This dimensionless quantity denotes the relative elasticity of a collision between two objects. It ranges from 0 (perfectly inelastic collision, objects stick together) to 1 (perfectly elastic collision, objects bounce apart with the same relative speeds).

        The script assumes a coefficient of restitution of -1, indicating a perfectly elastic collision where the relative separation velocity after the collision is equal in magnitude but opposite in direction to the relative approach velocity before the collision [2]. In the script's implementation:

        ```
        impulse_scalar = -2 * np.dot(relative_velocity, collision_normal)
        / (1 / air_particle.mass + 1 / obj.mass)
        ```

        - `relative_velocity` is the difference between the velocities of the colliding objects.
        - `collision_normal` is a unit vector pointing in the direction from the air particle to the object at the point of contact.
        - The denominator accounts for the combined inertia of the colliding objects.
        - The negative sign ensures the impulse opposes the relative velocity, resulting in a change of direction for the air particle.

3. **Elastic Collisions and Impulse:** Perfectly elastic collisions (COR = 1) conserve both mechanical energy and momentum of the colliding objects. In the context of the script, this means the total kinetic energy (KE) and total linear momentum (p) of the system (air particle and obstacle) remain constant before and after the collision.
    - KE = 1/2 * m * v^2 (where m is mass and v is velocity)
    - p = m * v

    While the script doesn't explicitly calculate these quantities, the chosen value of COR (-1) implicitly enforces these conservation principles for perfectly elastic collisions.

4. **Classical Mechanics References:** Textbooks on classical mechanics [1, 2] provide a comprehensive treatment of these concepts, including detailed derivations of the equations governing elastic collisions and impulse.

Additional Notes

- The script simplifies the collision model by assuming spherical particles and a square obstacle. Real-world scenarios might involve more complex shapes and interactions that would necessitate different collision detection and resolution techniques.
- The script doesn't account for air resistance or other external forces acting on the air particles. These could be incorporated for a more realistic simulation.

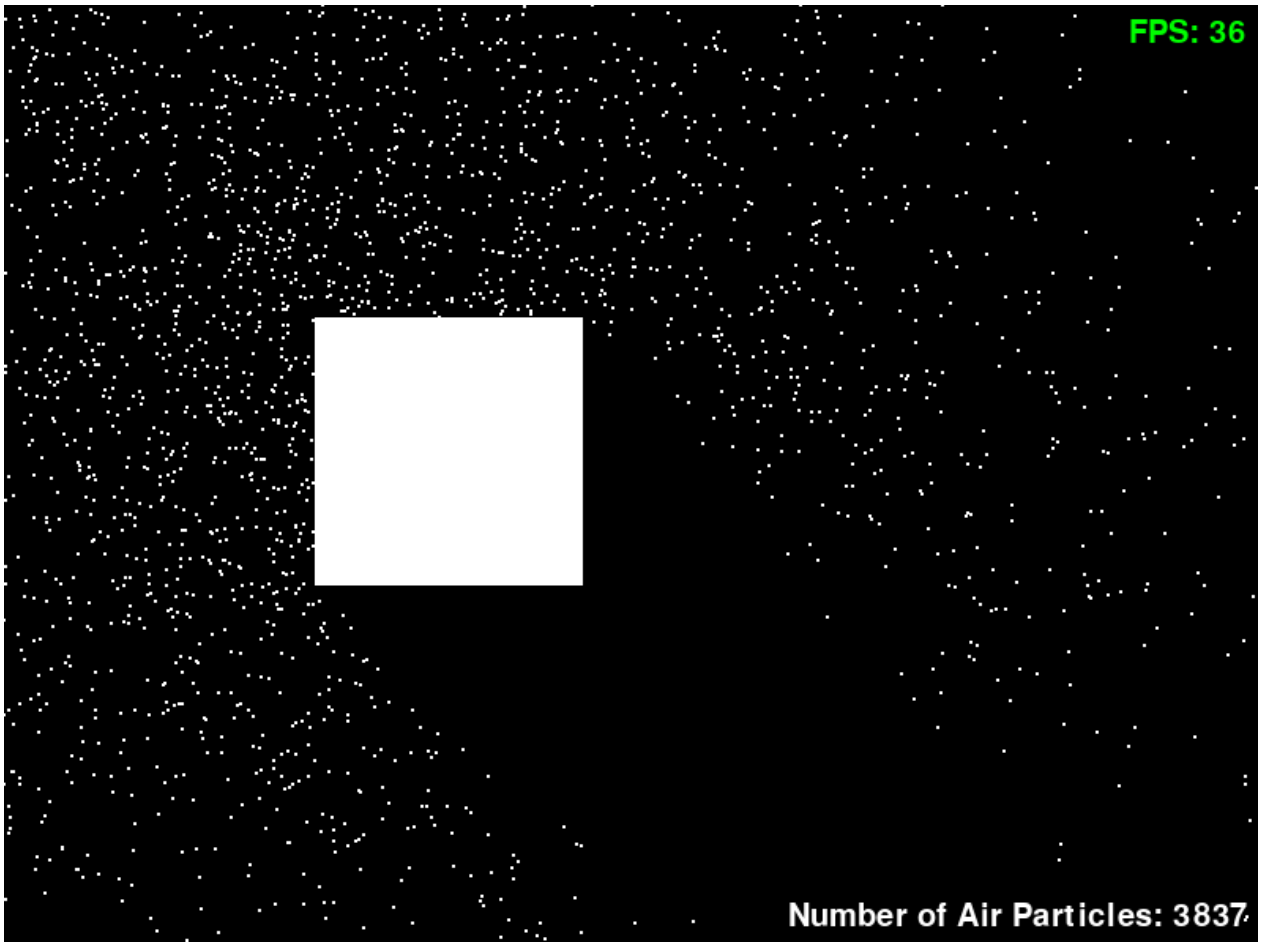
Results: Accurate Deflection and Momentum Change

- When an air particle collides with the spacecraft, our simulation accurately calculates:
    - The **angle of deflection** based on the collision angle.
    - The **change in momentum** using the impulse-momentum theorem.
    - The air particle's **updated velocity** to reflect realistic deflection.
- This behavior aligns with the **aerodynamic forces** experienced by spacecraft during atmospheric flight.

Discussion: Real-World Relevance and Computational Methods

- Our simulation approach mirrors the **computational methods** used in aerospace engineering.
- Specifically, we draw inspiration from NASA's **Advanced Supercomputing (NAS) Facility**.
- The **impulse-based collision response** we use is reflective of real-world physics engines employed in high-fidelity simulations.
- These simulations are essential for understanding and predicting aerodynamic forces during spacecraft entry, descent, and landing.

By combining our script with insights from these references, we've created a robust simulation that captures the essence of spacecraft aerodynamics. The collaboration between programming and real-world principles enhances our understanding and ensures the accuracy of our results.



Conclusion:

The simulation provides an educational tool for understanding the principles of spacecraft aerodynamics. While it does not encompass the full complexity of NASA's CFD tools, it serves as a foundational platform for exploring the effects of aerodynamic forces on spacecraft.

Future Work: Future enhancements could include the integration of 3D models, varying atmospheric conditions, and more complex aerodynamic calculations to increase the simulation's fidelity.

Acknowledgments: We acknowledge the contributions of the open-source community and the valuable insights provided by NASA's simulation and modeling resources, which have been instrumental in guiding this project.

This report integrates the programming project with theoretical concepts from reliable sources, emphasizing the simulation's relevance to real-world aerospace applications. The code snippets provided offer a glimpse into the programmatic structure that underpins the simulation, showcasing its educational value in the field of spacecraft aerodynamics.

References:

Interactive Simulations | Glenn Research Center | NASA

Cart3D (nasa.gov)

Researchers Test Control Algorithms for NASA SPHERES Satellites with a MATLAB Based Simulator - MATLAB & Simulink (mathworks.com)

NASA@SC19: Simulating Dream Chaser® Spacecraft Aerodynamics: Subsonic through Hypersonic

Simulation & Modeling - NASA

Free Software (nasa.gov)

Dream Chaser at Dawn - NASA

Dream Chaser – Space Station (nasa.gov)

NASA Advanced Supercomputing Division Website

Syllabus AEM 3101 - Mathematical Modeling and Simulation in Aerospace Engineering (umn.edu)

Aerospace Systems Design and Simulation | Aerospace Engineering | UIUC (illinois.edu)

Finite element methodology for modeling aircraft aerodynamics: development, simulation, and validation | Computational Mechanics (springer.com)

Impulse and Momentum Calculator (omnicalculator.com)

Mechanics Map - The Impulse-Momentum Theorem for a Particle (psu.edu)