RIGID BODY SIMULATION OF CONVEX POLYGON COLLISIONS

Darren Cleeman

ABSTRACT:

This project builds on the foundational concepts explored in Assignment 3, extending its scope to simulate dynamic interactions between multiple polygons of varying sizes, shapes, and physical properties. By adopting an object-oriented design, the simulation was generalized to handle complex scenarios involving collision detection, response, and stability under gravitational and external forces.

Key enhancements include the use of advanced mathematical models and algorithms to ensure accurate geometric and physical interactions. Convex hull algorithms validate polygon shapes for correctness, while the Separating Axis Theorem ensures precise collision detection. For collision response, impulse-based and Linear Complementarity Problem (LCP) formulations dynamically resolve forces, integrating rotational and linear effects. Additional features, such as customizable polygon creation and drag forces, were introduced to simulate real-world dynamics like friction and air resistance.

These refinements culminate in a robust and scalable simulation framework capable of rendering realistic interactions between rigid bodies in a two-dimensional plane. The project showcases the synergy between computational geometry, rigid body dynamics, and algorithmic optimization to achieve realistic physics simulations.

Introduction:

The study of rigid body dynamics is essential for understanding and simulating interactions between objects in physical systems. This project builds on the foundation established in Assignment 3, expanding its scope to simulate interactions among multiple rigid polygons with diverse physical and geometric properties. The objective was to develop a generalized and scalable framework for handling complex collisions and dynamic behaviors while maintaining computational efficiency.

Central to this work is the implementation of accurate geometric and physical models. Convex polygons were validated using computational geometry techniques such as convex hull algorithms, ensuring robust and precise shape handling. Collision detection leveraged the Separating Axis Theorem (SAT), enabling efficient detection of intersections between objects. Collision responses were governed by impulse-based models and Linear Complementarity Problem (LCP) formulations, facilitating realistic simulations of contact forces, friction, and rotational dynamics.

Beyond extending prior work, this project introduced enhancements to improve realism and flexibility, such as drag force modeling, energy damping mechanisms, and customizable polygon generation. These additions allow for the simulation of real-world behaviors, including complex interactions involving momentum, energy dissipation, and external forces.

The core challenge revolves around developing accurate and computationally efficient algorithms to handle polygonal collisions, contact resolution, and surface interactions in dynamic simulations. These problems are compounded when dealing with things such as non-uniform polygon shapes and real-time constraints. By addressing these challenges, this project seeks to enable robust and scalable physical simulations that accurately model the behavior of polygonal objects under dynamic conditions.

Methods:

This project focuses on developing a robust and scalable framework for simulating the dynamics of rigid polygons. The methods employed address key challenges, such as collision detection, contact resolution, and energy dissipation, while maintaining computational efficiency and physical realism. Below, we detail the assumptions, models, and approaches used throughout the project.

***Assumptions*:**

1. Rigid Body Approximation: Polygons are treated as rigid bodies with fixed shapes and uniform mass distribution, simplifying

dynamic calculations while preserving physical accuracy.
2. Convexity of Shapes: All polygons are assumed to be convex. This assumption enables the use of computationally efficient algorithms like the Separating Axis Theorem (SAT) for collision detection and simplifies physical modeling.
3. Global Parameters: Simulation constants such as gravitational acceleration (g) and time step (dt) are global. Object-specific properties, such as mass and friction, are encapsulated in individual polygon instances for modularity.
4. Energy Dissipation: Drag forces and friction are incorporated to emulate energy loss, ensuring polygons naturally come to rest in the absence of external forces.

***Polygon Generation and Validation*:**

To support user-defined shapes, the create_polygon_from_vertices function validates input vertices using Chan's convex hull algorithm [1]. The convex hull represents the smallest convex polygon enclosing a given set of points. By leveraging computational geometry techniques, the algorithm ensures that only valid convex polygons are processed.

Implementation: Input vertices are sorted lexicographically and processed to construct the lower and upper hulls iteratively. The orientation test, implemented using a cross-product calculation, ensures that each point added to the hull maintains convexity.

Purpose: This step guarantees geometric accuracy and robustness for user-defined shapes, which is critical for accurate collision detection and response.

***Collision Detection*:**

The is_touching function determines whether two polygons are in contact by leveraging the Separating Axis Theorem (SAT) [4]. SAT is particularly well-suited for convex shapes due to its efficiency and mathematical rigor.

Approach: The function computes potential separating axes using the edge normals of both polygons. Each polygon's vertices are projected onto these axes, and the intervals of their projections are checked for overlap. A lack of overlap on any axis indicates no collision.

Advantages: This method provides an efficient mechanism for detecting collisions, as the number of axes to test scales with the total number of edges in the polygons.

Results: By ensuring accurate and efficient detection of overlaps, this method supports realistic collision responses while minimizing computational overhead.

***Collision Response*:**

The project implements two distinct methods for resolving collisions: handle_collision_with_surface for static boundary interactions and collide_with_other for dynamic collisions between polygons. Both methods rely on impulse-based models to ensure physically realistic outcomes.

Collision Response Part 1: Static Collision Response

The handle_collision_with_surface method computes normal and tangential impulses at the contact point using a Linear Complementarity Problem (LCP) formulation [2][3].

Mathematical Model: The contact Jacobian matrix relates velocity changes to contact forces. The LCP solver computes these forces while respecting non-penetration constraints and friction limits.

Angular Dynamics: The calculated impulses generate torques, updating the polygon's angular velocity based on its moment of inertia. This captures rotational effects induced by boundary collisions.

Purpose: This method provides a robust mechanism for simulating interactions with static surfaces, incorporating both frictional and rotational effects.

Collision Response Part 2: Dynamic Collision Response

The collide_with_other method resolves collisions between moving polygons by computing relative velocities and applying normal and tangential impulses [2][3].

Impulse Calculation: The method calculates normal impulses to resolve relative velocities along the collision normal and tangential impulses to model frictional effects. These impulses are applied to update linear velocities.

Rotational Effects: Impulses displaced from the center of mass induce torques, updating the polygons' angular velocities. Angular velocity is clamped to prevent instability due to excessive spinning.

Results: This method ensures realistic collision behaviors, capturing momentum exchange and rotational dynamics between polygons of varying properties.

Energy Dissipation:

The apply_drag function models aerodynamic drag to simulate energy dissipation in moving polygons.

Linear Drag: The drag force is calculated using $F_d = 1/2 * \rho * C_d * A * v^2$, where ρ is the fluid density, Cd is the drag coefficient, A is the cross-sectional area, and v is the velocity magnitude. This force opposes the polygon's motion, reducing its velocity over time.

Rotational Drag: Rotational drag is modeled as $\tau_d = -C_r * \omega$, where Cr is the rotational drag coefficient and ω is the angular velocity. This gradually reduces spinning.

Outcome: The drag model ensures polygons lose energy and come to rest under realistic conditions, preventing perpetual motion.

Rotational Dynamics:

The calculate_inertia_matrix_inverse function computes the inverse of the moment of inertia for polygons, supporting both regular and custom shapes.

Regular Polygons: The moment of inertia is computed using a formula derived from symmetry and mass distribution: $I = 1/2 * mR^2*(1 + \cos^2(\pi/n))$, where mmm is the mass, R is the radius, and n is the number of sides.

Custom Polygons: For irregular shapes, the moment of inertia is calculated using an area moment integral, summing contributions from triangular slices of the polygon.

Importance: Accurate calculation of rotational resistance is essential for simulating angular dynamics during collisions, ensuring realistic updates to angular velocity.

Experiments:

The experimentation phase involved iterative refinements to the simulation, addressing key challenges and evaluating the results to ensure realistic and efficient behavior. The experiments focused on improving polygon representation, collision detection and response, and physical accuracy through drag and friction modeling. Below, the main steps and observations from the experiments are outlined.

***Polygon Representation and Generalization*:**

The initial implementation handled only a single square with a hardcoded size and global variables for simulation parameters such as gravity (g), mass (m), and timestep (dt). To expand the scope, a Polygon class was introduced, allowing each polygon to have its own attributes while retaining some global constants like g and dt for simplicity. The class was extended to support any regular polygon, requiring updates to mathematical operations such as the radius of gyration (rg_squared). The updated calculation incorporated the number of sides, ensuring proper mass distribution across polygons. Additionally, a generate_polygon_vertices function allowed dynamic generation of regular polygons, while the

calculate_inertia_matrix_inverse function enabled accurate computation of rotational inertia.

***Collision Detection and Response*:**

One of the primary challenges was ensuring accurate detection and response during polygon collisions. The is_touching function initial approach used a simple radius-based detection method that compared the distance between polygon centers. This led to inaccuracies, especially for polygons with fewer sides (e.g., triangles or squares). To address this, the Separating Axis Theorem (SAT) was implemented, which significantly improved collision detection by projecting polygons onto potential separation axes and identifying overlaps. This method ensured precise detection even for irregular shapes.

For collision response, the collide_with_other function initially used a basic algorithm to compute linear separation forces but neglected rotational effects. This led to unrealistic behaviors, such as polygons gliding or spinning unnaturally during collisions. To resolve this, a torque calculation was introduced based on the contact point and collision impulse. The update considered both linear and angular velocities, ensuring more realistic post-collision dynamics.

***Surface Interaction and Boundary Collisions*:**

To handle collisions with boundaries, the handle_collision_with_surface method was implemented using the Linear Complementarity Problem (LCP). Initially, issues arose with polygons passing through walls or jittering against surfaces. These problems were addressed by dampening horizontal velocities during position correction and incorporating energy damping to stabilize bouncing behaviors. The improved algorithm generalized surface interactions for all boundaries, ensuring consistent handling of ground, walls, and ceiling collisions.

***Concurrent Trajectory Simulation*:**

Early versions of the simulation used sequential updates for each polygon, which caused desynchronization during collisions. Polygons would often respond to outdated positions or velocities of other polygons, leading to visually incorrect interactions. To address this, a threading approach was introduced, allowing polygons to compute trajectories concurrently. Using Python's Barrier class ensured that all threads synchronized at each timestep, resolving timing issues and enabling accurate collision handling.

***Energy Dissipation and Friction Modeling*:**

During experimentation, it was observed that polygons could spin indefinitely without external forces, revealing the need for drag modeling. The apply_drag method was developed to introduce both linear and rotational drag, based on fluid dynamics principles. The drag equations helped dissipate energy naturally, allowing polygons to come to rest over time. However, slight vibrations persisted after polygons settled on surfaces, prompting the introduction of the apply_static_friction method. This method, grounded in Newtonian mechanics, effectively stopped polygons at rest and marked them as "stationary" to prevent unnecessary computations.

***Customization and Scalability*:**

The final phase of experimentation focused on enabling user-defined convex polygons while ensuring geometric validity. Chan's convex hull algorithm was integrated into the create_polygon_from_vertices method, enforcing convexity by reordering vertices. This addition allowed diverse shapes to be simulated without compromising collision accuracy or computational efficiency. The implementation demonstrated scalability, supporting multiple polygons with varying attributes in real-time simulations.

Discussion:

The project successfully achieved its primary goal of simulating realistic and dynamic collisions between convex polygons while maintaining computational efficiency. This section reflects on the strengths and limitations of the approach, as

well as potential improvements and extensions for future work.

***What Worked*:**

The simulation demonstrates a robust application of computational geometry and physics-based modeling. The physics of collisions are realistic enough to provide a convincing depiction of interactions between objects. For instance, collisions with the screen borders—handled by the handle_collision_with_surface function—exhibit good realism, capturing both energy dissipation and appropriate bounce dynamics.

The integration of the Separating Axis Theorem (SAT) for collision detection significantly enhanced accuracy compared to the initial radius-based approach. By leveraging SAT, the program reliably detects collisions between polygons of varying shapes and sizes, even in complex scenarios. Similarly, the implementation of drag and static friction modeling enabled the simulation to depict the slowing and eventual rest of polygons more accurately, contributing to the overall physical believability of the simulation.

Given the limitations of the matplotlib.animation framework, the program still achieves dynamic behavior for both collisions and trajectories. It effectively demonstrates how computational geometry and physics algorithms can be integrated to simulate physical systems. The framework was sufficient to validate the algorithms and showcase the potential of the approach.

***What didn't work and limitations*:**

While the project successfully addressed many challenges, several limitations became evident during development and experimentation. One notable limitation is the collide_with_other method, which does not implement the Linear Complementarity Problem (LCP) algorithm. As a result, collision responses between polygons are less realistic than they could be, particularly in scenarios where rotational dynamics and contact points play a critical role. For example, if two polygons have the same x-coordinates and one lands on top of the other, the collision does not account for the specific corner or edge involved, leading to unnatural vertical bounces.

Another limitation lies in the implementation of apply_static_friction and is_flat functions. While these functions are mathematically sound, their integration into the update method of the Polygon class was rushed. Although the goal of making polygons "settle" on the ground was achieved, the resulting behavior is not as realistic as it could be. The transitions from movement to rest sometimes appear abrupt or mechanically incorrect.

The handle_collision_with_surface function, while effective, is not as generalized as desired. The current implementation only supports horizontal and vertical surfaces, restricting its applicability. A more generalized approach, capable of handling collisions with arbitrary surfaces defined by directional vectors or vertices, would significantly enhance the versatility of the simulation.

Finally, the use of matplotlib.animation posed inherent challenges for creating highly dynamic behavior. While sufficient for prototyping and visualization, a framework like pygame would have been better suited for handling real-time interactions and achieving greater performance and flexibility.

***Future Directions*:**

This work highlights the potential for integrating computational geometry, physics-based modeling, and algorithmic optimization in physical simulations. However, there is ample room for improvement and expansion. Future work could focus on extensions such as implementing LCP in collide_with_other. Incorporating the LCP algorithm would improve the realism of dynamic collisions, particularly for scenarios involving rotational dynamics and complex contact points.

References:

[1] Chan, T. M. *Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions*, Springer-

Verlag, 1996, link.springer.com/article/10.1007/BF02712873.

[2] Kavan, Ladislav. *Rigid Body Collision Response*, Faculty of Mathematics and Physics Charles University Prague / Czech Republic, users.cs.utah.edu/~ladislav/kavan03rigid/kavan03rigid.pdf.

[3] Patil, Sachin. *Collision Response and Contact Handling for Articulated Body Dynamics*, 29 Oct. 2007, www.cs.unc.edu/~lin/COMP768-F07/LEC/20.pdf.

[4] Liang, Cheng, and Xiaojian Liu. *The Research of Collision Detection Algorithm Based on Separating Axis Theorem*, School of Computer and Software Engineering, Xihua University, Chengdu, China, 2015, www.ijscience.org/download/IJS-2-10-110-114.pdf