

Collision Detection and Motion of Rigid Bodies: A Literature Review

Jerold Pimentel^{1*}, Chuhan Zhong², Yulan Luo³

¹*Dulwich College Shanghai Puxi, Shanghai, China*

²*Basis Internation Shenzhen, Shenzhen, China*

³*Rosedale Global High School, Chengdu, China*

**Corresponding Author. Email: jerold.pimentel26@stu.dulwich.org*

Abstract. Collision detection is an integral part of many video games. Here, we review several collision detection methods and algorithms, examining the evolution of collision detection and evaluating the efficiency of methods to detect collision detection as well as methods of resolving collisions in the context of physics simulation.

Keywords: Collision Detection, Rigid Bodies, algorithms

1. Introduction

Collision detection involves the detection of penetration between two bodies. Collision detection algorithms aim to detect whether or not a collision has taken place, where such a collision has happened and when such a collision occurs.

Collision detection is vital within many video games, whether it be detecting if a player's bullet has hit a monster or whether a wrecking ball has hit a wall, many game mechanics (and video games) would not be possible without collision detection algorithms.

Efficient collision detection is needed to preserve the real-time nature of video games: detection must be precise enough for unhindered gameplay, yet it must be quick enough and cheap enough to execute so that more computer cycles can be dedicated to more intensive calculations or operations to give the player an optimal experience.

2. Background

2.1. Efficiency

The *efficiency* of an algorithm is defined by two things: how much time is taken to detect a collision and how much of the computer's resources it uses for the precision of collision detection being offered. The 'resources' of a computer is loosely defined as each method reviewed will focus on separate metrics, however in general, less time taken and less resources used means a more efficient algorithm.

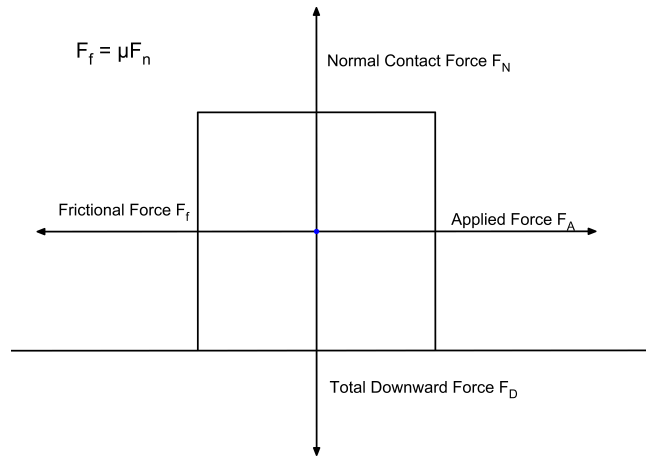


Figure 1. Friction acting on a body

2.2. Rigid bodies

A rigid body is a body that is unable to deform. More precisely, the distance between any two points on the surface of the body or within the volume bounded by the body will stay constant, no matter what deforming forces are applied onto the body. Therefore, a rigid body is one that is perfectly inelastic. In the context of collisions, in a physics simulation, the collision of two rigid bodies almost certainly means interpenetration, as the two bodies are unable to deform and therefore must penetrate.

2.3. Coulomb's friction model

Coulomb's friction model, or Coulomb's friction law, allows the force of dry friction F_f to be approximated based on:

- the coefficient of friction μ , a dimensionless scalar
- the normal force F_N , a vector measured in Newtons (N) as shown in 1. Specifically, the law states that

$$F_f = \mu F_N$$

Here, μ takes on different values depending on which type of friction is being calculated. Specifically, μ may be the coefficient of static friction μ_s (and the type of friction referred to in this article) when the two bodies are at rest, or μ may be the coefficient of dynamic friction μ_d when one or both bodies are in motion.

3. Basic methods in dealing with rigid body collision

In Baraff's paper [1], the "Non-penetration constraint" refers to preventing two rigid bodies from penetrating each other in physical simulation. Generally speaking, traditional physics simulators first detect whether two objects will penetrate each other, and then prevent them from penetrating each other. This method is widely used because of its simplicity. However, this method is computationally inefficient, requiring more simulation time, and requiring more adjustments under some simulation conditions. The article proposes that the simulator should not prevent rigid bodies

from penetrating, but should try to simulate all the actual forces generated in Newtonian physics and use these forces to deduce the real movement of objects.

An early method for dealing with rigid body collisions is known as the "Penalty Method". When objects penetrate, a restoring force (usually a spring force) is generated in the simulation to separate them. This method is very easy to implement and has been widely used in the physical simulation of rigid bodies and some soft bodies. But it has some drawbacks: although it is easy to implement, it has a high computational complexity, so it is inefficient; if this method is not adjusted, the spring force given to objects of different weights is the same, leading to unrealistic physics. More adjustments are needed to better simulate Newtonian mechanics.

The "Analytical Methods" proposed in the article calculates the contact and collision forces between objects through Newtonian dynamics. This can calculate more accurate physical simulations than "Penalty Methods" and reduces the time required for simulation. However, "Analytical Methods" is more difficult to use, especially when dealing with multiple objects or multiple contact points between two objects at the same time.

When two rigid bodies are in frictionless contact—that is, when the friction coefficient is assumed to be zero, so that no tangential friction forces act between the surfaces—the typical contact configurations are vertex-plane and edge (line)-edge contacts. This assumption of frictionlessness is an idealization used in theoretical models or situations where friction is negligibly small (for example, under highly lubricated conditions). The article proposes that the "vertex-plane" contact force should be perpendicular to the plane and act on the vertex along the normal vector to prevent vertex-plane contact, while the edge-edge can select an object to apply the same contact force along the normal vector. Similarly, by Newton's third law, the other object experiences an equal force in the opposite direction. However, in cases such as vertex-vertex contact, the determination of a unique normal vector is ambiguous, creating what we call a "singular contact point." In these situations, rather than requiring a human to manually adjust the force direction each time, our computational framework automatically applies a heuristic—based on local geometric considerations—to select a reasonable normal direction, thereby allowing the calculations to proceed consistently. When there are multiple contact points, especially when there are multiple singular contact points, the algorithm becomes too complicated. The solution proposed in the article is a "heuristic algorithm": the local temporary geometry of one of the objects in the singular contact point is expanded to make it a certain contact situation, and then the normal vector and force are calculated according to the normal calculation methods and applied on the object.

After determining the direction of the contact force, the author uses linear programming to calculate the magnitude of the contact force. This ensures that the contact force does not change drastically after the objects collide. This continuity satisfies the non-penetration constraint and conforms to the laws of Newtonian mechanics. In addition, to simplify the calculation, only the contact points at vertices and edges are considered in the simulation, not the contact points within polygons and line segments. This is also a heuristic algorithm. According to the article, this method will fail in some cases and cannot guarantee the most accurate simulation results. But this algorithm performs well in most practical applications, especially in the simulation of large-scale complex systems.

The author also raised some issues. When friction is introduced, the heuristic algorithm will cause the kinetic energy in the system to increase or decrease, thus affecting the accuracy of the simulation. The article proposes a treatment method to enable the system to quickly return to the correct mechanical state. However, the conventional method cannot handle the system with static

friction. By modifying the Coulomb friction model, the static friction can be handled in a few cases to obtain an approximate effect.

4. Methods of collision detection

"Bounding Object-Based Collision Detection" determines overlapping between objects at the level of their enclosing shapes—such as polygons—rather than based on precise object geometry [2]. While this method is computationally efficient and often serves as a preliminary filter in physics simulations, it is not sufficiently precise on its own for detailed physical interactions. In practice, such bounding volume techniques quickly identify potential collisions, which are then refined with more accurate, computationally intensive methods to fully resolve contact dynamics.



Figure 2. The bounding circle of a flower

Bounding Circle Collision Detection involves enclosing each object in the smallest possible circle. This method is one of the simplest collision detection methods, because it only applies one principle that two bounding circles only overlap when the distance between the centers of the circles is less than or equal to the sum of the radius. However, problems occur when the shape of the enclosed object leads to false collisions at its edges. If the circle is too big for an irregularly shaped object, like a rocket which has wide base but a small head, and the overlapping is based near the center, the collision is detected even if another object has not overlapped with the smaller area of the enclosed object or another object passes through the greater area of the bounding circle.



Figure 3. The bounding box of a rocket

Bounding Box-Based Collision involves enclosing objects within the tightest bounding rectangle aligned with the coordinate axes. Like the bounding circle collision detection, the collision check is performed by determining if the bounding boxes of two objects overlap, however strategies exist to reduce the false collision. For one, the shape of the bounding box allows some objects to have more realistic collisions. For another, enclosing an object in multiple bounding boxes allows collisions to be simulated more realistically.

However, this method has a drawback. When an object translates, its bounding box moves accordingly. However, axis-aligned bounding boxes becomes less effective when the object rotates.

The bounding box's size increases and no longer fits the object tightly leading to inefficient collision detection. Oriented Bounding Boxes, or OBB, addresses this issue. OBB has one difference: the box is rotated along with the object. In addition, there exists a hybrid bounding circle and box collision test, meaning that collision happens among different shapes of bounding objects. A collision occurs when the distance between the center of the bounding circle and any point on the bounding box is less than the radial distance of the bounding circle.



Figure 4. An example of a bounding polygon for an ellipse

Bounding polygons are constructed from arbitrary points to form the optimal enclosing shape for an object. This approach relies on a theorem known as the Separating Axis Theorem (SAT), which is used to determine whether two convex shapes are intersecting and to identify the minimum penetration vector required for collision resolution. The SAT operates by projecting the vertices of both polygons onto potential separating axes; according to the theorem, if there exists any axis along which the projected intervals do not overlap, then the two polygons are not colliding. Although the Separating Axis Theorem provides high accuracy, it is computationally expensive.

5. Hardware-accelerated collision detection for 3D virtual reality gaming

The paper [3] suggested that collision detection methods used in the 21st century differ greatly from their 20th century counterparts. Of the many computer-based developments that have occurred, the development of the Graphics Processing Unit (GPU) is one such development that has heavily impacted video game development. The method below outlines one such way in which hardware acceleration is used for collision detection in 3D virtual reality.

The algorithm uses a data structure known as a *scenegraph*. Such a graph encompasses all of the geometry and textures needed to display a scene, from material properties and mesh surfaces to transformations and bounding volumes.

5.1. The algorithm at a glance

The algorithm, like many collision detection algorithms today, is multi-phased, split into the broad phase and the narrow phase:

- In the broad phase, the hierarchical trees of the two objects being tested are traversed, and at each pair of nodes their bounding spheres are tested for any overlap. If there is no overlap, then the two objects do not collide and 'false' is returned. If there is overlap, then proceed onto the narrow phase. This is a quick test aiming to eliminate redundant checking.
- In the narrow phase, a viewing volume and occlusion query method is employed to directly pinpoint if a collision happens, and where such a collision has taken place.

5.2. Collision detection through viewing volumes

The viewing volume algorithm is described below:

Let objects A and B be the two colliding objects. Within the bounding box of object A, six frustums are constructed with the camera at the center. Each frustum correspond to the faces of the bounding box, and bases of the frustums define the near and far plane. These frustums make up the viewing volumes of object A.

Object B's rendering in a viewing volume is slightly different. Each face of object B is sent through the graphics pipeline with the following optimizations:

- Using GL SELECT: this render mode only sends the faces of B within the viewing volume of A through the graphics pipeline. Reduces the faces needed to be rendered
- Bounding Volume Hierarchy (BVH): The oriented bounding box (OBB) of B is rendered within every viewing volume in A. Occlusion queries check whether A penetrates the root bounding volume of B. If this does not happen, then there can be no collisions between A and B.

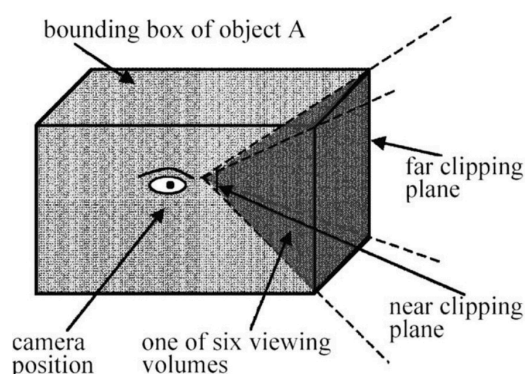


Figure 5. 1 of 6 viewing volumes. Diagram from paper [3]

- Object Convex Decomposition: Which involves breaking apart non-convex objects into simpler convex shapes. This allows the algorithm to be generalized to an object A that is non-convex.

Then, objects A and B collide if B occludes A in any of the 6 bounding boxes that have been constructed.

5.3. Conclusions and comparisons

The results shown within the article are promising. Commercially available methods of collision detection (RAPID, developed in 1996) were testing against the Hardware Accelerated Collision Detection (HACD) algorithms, benchmarked against HACD0 (brute force rendering of B). The time taken for collision detection in milliseconds was measured, and it was tested against 4 scenes. Of the HACD variations tested, HACD2 (using BVH and oriented bounding box tree optimization) were able to detect collisions within a shorter time of RAPID.

It is interesting to note that HACD was performed on devices with 512MB and 128MB of RAM, a far cry from the lowest RAM capacities commercially available today. Overall, the results from the testing shows that HACD (especially HACD2) scales extremely well with the number of triangles being rendered, its time extremely consistent throughout all test scenes.

Algorithms like HACD are what is most commonly used in gaming engines today. Such algorithms are much easier to understand and much more abstracted, unlike those relying heavily on physics simulations (like the algorithm proposed in the [1] paper) or otherwise. Therefore, we see

that they are much easier to implement. Given the vast advances in hardware that have been made since 2006, hardware accelerated collision detection algorithms are also much, much faster than their physics simulation counterparts, as the very task of collision detection contains a lot of repetition and iteration, growing in complexity with the number of points or faces. This is something CPUs and GPUs handle well, though in a non-hardware-assisted physics simulation this would cause the data and operations that need to be performed to grow to an unmanageable size (and quickly!).

6. Efficient library of rigid body physics simulation in games

The paper written by Felis [4] provides an overview of advanced algorithms used in the simulation and control of rigid body dynamics. In order to achieve this goal, the paper provided the basic equations required to determine the generalized coordinates, acceleration, external contact, collision, and motion of rigid multi-body systems. Additionally, they propose an open-source implementation of recursive methods in rigid body dynamics, known as RBDL (Rigid Body Dynamics Library). Recursive methods are functions that call themselves, breaking down a problem into smaller sub-problems. The essay also reports on a modeling framework for multi-body rigid body systems, serving as the primary structure of the RBDL library. When a single body is part of a rigid multi-body system, its rotation and motion can be calculated using the joint motion frame. The joint motion frame is a method that acts as an additional frame when the joint moves around parent bodies.

6.1. Algorithms in RBDL

In Part 5, the RBDL library includes dynamics algorithms that can efficiently compute the motion of bodies through predefined algorithms within the package, rather than requiring the generation of new code from additional mathematical support.

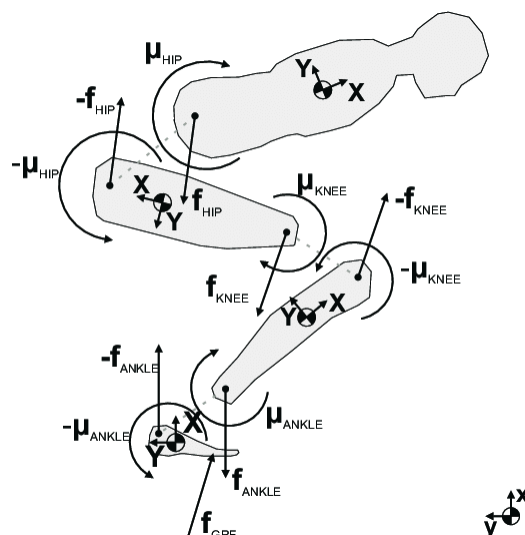


Figure 6. A visualization of the information needed for the Newton-Euler algorithm

The first dynamics algorithm is the Recursive Newton-Euler Algorithm. This algorithm operates recursively, starting from the base or root body and moving upwards through the kinematic chain. By utilizing the principle of Newton's second law and Euler's equations of motion, this algorithm

can determine the motion of each body based on both the internal and external forces acting upon it. The calculation operates in following way:

- Compute the position, velocity, and acceleration of all bodies.
- Use spatial equation of motion to compute for each body the net force that causes the previously computed acceleration.
- Compute the force that is transmitted over each joint.

This approach is highly efficient, especially for systems with a large number of bodies, as it avoids redundant calculations and simplifies the overall computation.

Next, the RBDL library also contains the Composite Rigid-Body Algorithm. This algorithm takes a slightly different approach, focusing on the overall motion of a composite rigid body rather than individual bodies within the system, by calculation of energy. The calculation of energy is the sum of all kinetic energy of individual bodies in the composited body, by the formula $K = I v^2 / 2$, which I is the spatial inertia and v is the spatial velocity. This algorithm is particularly useful in situations where the internal structure of the bodies is not critical, and the overall behavior of the system is the primary concern. By reducing the complexity of the system, the Composite Rigid-Body Algorithm allows for faster and more accurate simulations.

Finally, the Articulated Body Algorithm. This algorithm is specifically designed for simulating articulated systems, such as robotic arms or human skeletons, where bodies are connected by joints that allow for limited ranges of motion. While a body is connected with other bodies, the concept of articulated inertia is a dynamic property of a body when a force reacts with it. Compare with the Composite Rigid-Body Algorithm, the articulated inertia take the influence of other body into account. In the actual calculation, the method uses articulated inertia to calculate the bias force of bodies, and get the final result of joint accelerations. The Articulated Body Algorithm operates by representing each body as a rigid segment and each joint as a constraint that limits the relative motion between adjacent segments.

6.2. Comparison and conclusion

In comparison between RBDL and other rigid body physics implementation packages and emphasizes its benefits. Compared with SIMBODY in the computations on planar chains of varying lengths in different range of freedom, RBDL exhibits 9-38% higher efficiency compare with SIMBODY in all scenario. Also, compare with symbolic generation tool DYNAMOD in performance in humanoid robot model with 35 degrees of freedom, although RBDL has longer duration time, it might be more efficient because the it can seperately evaluates single part in the composited bodies. These methods in RBDL are contributes to quickly and efficiently simulate the reaction of rigid-body under collisions.

7. Parellelized engine structure to optimize collision detection

This paper, written by John Linford and Dr. Teng Moh [5], aims to optimize collision detection to improve user experience and game performance.

Firstly, to solve the specific problem of penetration in discrete collision detection, the minimum translation distance is found. The exact time of collision is found to maximum precision and the collision is resolved.

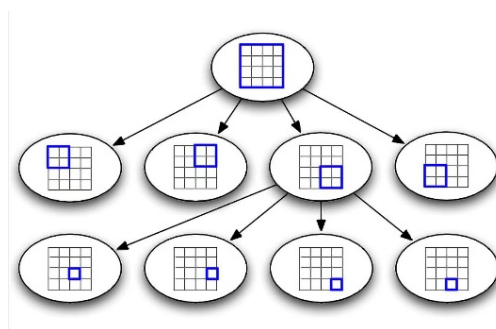


Figure 7. A diagram showing how the quad-tree is traversed when finding suitable nodes for object placement

Secondly, unnecessary collision checks are eliminated to increase speed. In this broad phase, a Quad-Tree is applied to reduce the time complexity. The space is divided into four parts represented by four nodes and then divided again into smaller four parts represented by four child nodes. Based on the quad-tree, they find a node, from child nodes to parent nodes, which can contain the object, and the object is placed in the node. Because the space is divided, two objects in different nodes can be excluded from collision checking.

Thirdly, parallelization is used, where data is divided into smaller parts processed by each processor. Many problems can occur in this step, for example, inconsistencies within each processors. The quad-tree and graphics segments are used to complete the data division.

Additionally, to avoid the possibility of two contradictory tasks, only the body is changed. In this case, even though sometimes there are duplicate processes for the collision (one object is in the parent node of another), the node they are enclosed in updates their data once.

Finally, they compare their engine with others. Though there is a huge difference between their data volumes, the parallel structure of physics engine significantly accelerates the process. With the development of technology and a demand for such algorithms, the authors believe such projects are useful in the future.

8. Conclusion

In this paper, We focused on researches of collision detection and rigid body simulation in gaming. We have discussed and compared the efficiency of various methods, including penalty methods, analytical methods, and heuristic algorithms to deal with rigid body collisions, as well as different object bounding techniques such as the bounding circle, box-based collision and bounding polygon, and collision detection techniques using the viewing volume method. Additionally, we have explored advanced algorithms such as the RBDL rigid body motion simulation. To further optimize collision detection, we highlighted the use of parallelization and data structures like Quad-Tree.

Acknowledgment

The authors would like to thank Professor Nace for his invaluable contributions to this paper, by providing guidance on our topic and reference selection. His expertise was crucial in shaping the direction and depth of our research. We are also thank for the encouragement and assistance provided by the teaching assistants. Their active participation in discussions fostered an supportive environment, significantly enhancing our grasp of the subject matter.

References

- [1] Baraff, D. (1989). Analytic methods for dynamic simulation of non-penetrating rigid bodies. *SIGGRAPH*, 10.1145/74333.74356, 10.
- [2] Mileff, P. (2023). Collision detection in 2D games. *University of Miskolc, Hungary, Institute of Information Technology. Production Systems and Information Engineering*, 10.32968/psaie.2023.3.2, 17.
- [3] Cai, Y., et al. (2006). Hardware-accelerated collision detection for 3D virtual reality gaming. *SAGE Journals*, 10.1177/1046878106293678, 16.
- [4] Felis, M. L. (2016). RBDL, an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, 10.1007/s10514-016-9574-0, 17.
- [5] Linford, J., & Moh, T. (2014). Parallelized rigid-body dynamics. *San Jose State University*, 10.1145/2638404.2638480, 6.