Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/37/20230505

Cutting-edge research in physics engines: Exploring parallel technologies, algorithm insights, and upcoming trends

Chang Che

College of Computer Science, Beijing University of Post and Telecommunication, Beijing, 100876, China

chechang@bupt.edu.cn

Abstract. Physics engines, a quintessential product of computer science, have become indispensable tools in simulating physical phenomena through mathematical computations. Their prominence in game development, engineering simulations, and various physics applications is unquestionable. Ensuring their optimization is pivotal for meeting the rigorous demands of practical experiments. The significance of this study is twofold: theoretical foundations and practical necessities. From a theoretical standpoint, this paper delves into the core principles and architectural underpinnings of physics engines. We venture into the integration of multithreading and the application of niche algorithms within these engines. Further analysis illuminates how leveraging these technologies can drastically enhance the performance of a physics engine. In culmination, we propose a robust, feasible theoretical framework for physics engines, filling in the pivotal details. This exploration aims to galvanize and steer the evolution of physics engine design. Addressing practical necessities, our research emphasizes real-world applications of various physics engines. We examine tailored optimization strategies suited for specific needs and investigate methodologies to elevate the operational efficiency of specialized physics engines. This dimension of our study holds substantial real-world implications.

Keywords: Physics Engine, Optimization, Multithreading.

1. Introduction

Physics engines can be broadly categorized into two types: high-precision and real-time engines. High-precision engines aim for the utmost accuracy in their simulations, albeit often at the expense of speed. Conversely, real-time engines prioritize swift performance, sometimes compromising on physical fidelity. An essential metric for evaluating real-time engines is their execution speed. This paper primarily delves into the optimization techniques tailored for real-time engines.

Real-time physics engines find their utility across a spectrum of applications, from video games and 3D animation to scientific simulations. However, to effectively straddle the delicate balance between accuracy and scalability, these engines must be impeccably optimized. A significant computational chunk of a physics engine is dedicated to collision detection and resolution. Hence, our discussion predominantly orbits around harnessing parallel technologies and algorithmic nuances to optimize this segment.

© 2023 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/37/20230505

At its core, collision detection and processing replicate real-world object interactions using intricate geometric relationships. Given the profusion of convex polygons and the sheer graphical complexity inherent to physics engines, collision detection presents a formidable challenge. Parallel computing emerges as a salient solution, addressing aspects like synchronization, task distribution, and harmonious CPU/GPU collaboration. From an algorithmic standpoint, the objective is twofold: to sidestep redundant operations and to prune unnecessary calculations, sometimes even trading a sliver of accuracy for a notable boost in efficiency.

2. Fundamental Principles and Architecture

The main task of the physics engine is to provide the simulation of physical systems, such as: rigid body simulation, software simulation, fluid simulation, etc. Each system will adopt different ways to obtain the most suitable results. This paper mainly focuses on rigid body simulation, but the optimized scheme has certain universality and is not necessarily limited to a certain use.

2.1. Architecture

Regardless the type of engine, the core of the physics engine can be summed up as a loop. The form of this loop is shown in Figure 1.

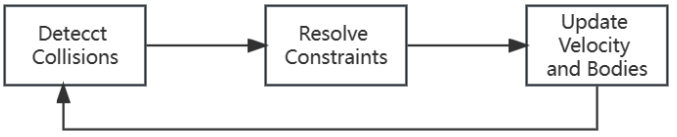

Figure 1. Simulating loop of physics engine (Photo/Picture credit: Original).

The first is collision detection, the algorithm of collision detection explores all the collision or contact points in all the graphs. This step can be broken down into two phases: The Broad Phase, which aims to screen out groups of objects that are likely to collide, and the Narrow Phase, which looks for the set of points in these groups of objects that actually collide. After the point of collision is found, the physical laws of motion need to be applied to the colliding object. Then it is also necessary to add the effect of friction, and because the computer operation is discontinuous, there may be overlapping problems between objects, and avoiding the overlap of objects is also a concern in this step. This Phase can be called the Resolve Phase and is used to resolve the constraints in a collision. After resolving the change in the state of the object caused by the collision, the state and speed of the object need to be updated and applied. Until the next collision detection, all objects in the engine will continue to move at the updated speed and state. The time between two collision detections is called dt. The physics engine's render frame does not have to be bound to a logical frame for collision detection, it can render only once for every several collision detection, or vice versa, or assign a separate thread to the render. Rendering and collision detection are independent of each other. Dt, which records the time between two collision detection, is an important indicator for judging the performance of a physics engine. dt can not only explain the time consuming of a single collision detection, but also show the accuracy of the engine from the side, because the more frequent the collision detection of a physics engine, the more accurate the simulated physical effect.

2.2. Collision Detection

Collision detection can be implemented in many forms, the simplest of which is to check whether the vertices of each rigid body exist in another rigid body. This form of judgment results in every vertex of every rigid body needing to be checked against all other rigid bodies, with a time complexity of O(n^2). Since the number of objects in physics engines is generally large, this form of detection can lead to low efficiency, so the next few sections are needed to optimize collision detection.

DOI: 10.54254/2755-2721/37/20230505

2.2.1. Discrete Collision Detection. Since computer system are discrete, the physics engine is also discrete, meaning that collisions between objects do not actually touch each other. The collision of the physics engine is shown in Figure 2, in the frame before the collision, the two objects are separated, and in the frame when the collision occurs, the two objects overlap each other. However, rigid bodies in the real world do not overlap each other, so the overlap problem needs to be solved before the velocity and state of the objects are updated. In order to solve this problem, we need to find a minimum translation distance, to push the two objects along the opposite direction of the velocity vector and let them just touch each other, to avoid the object overlap problem when the object collision.

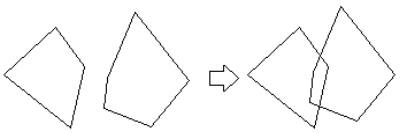

Figure 2. Collision diagram (Photo/Picture credit: Original).

On the other hand, in the physical world, rigid bodies do not overlap. But in a physics engine, as long as the object is not moving too fast, the error caused by the overlap in the event of a collision is not big. What's more, even if the two objects are pushed to the position of just contact, there will still be some error caused by the loss of movement time, and the error cannot be completely eliminated. Therefore, whether to correct the accuracy of discrete collision detection depends on the accuracy and efficiency needs of the physics engine.

3. Analysis of Typical Algorithms in Physics Engines

As mentioned above, the most important part of the physics engine is the collision algorithm. The collision algorithm can be divided into three phases: Broad Phase, Narrow Phase and Resolve Phase. Next, the algorithm optimization of these three parts will be carried out.

3.1. Broad Phase

In the Broad Phase, the program will first conduct a round of detection of all potential collision object pairs, and eliminate the object pairs that are unlikely to collide, so as to avoid these object pairs from participating in the calculation in the next two stages.

3.1.1. Bounding Box. A processing strategy in this stage is to wrap each object with bounding box, and then judge whether each bounding box generated by the object pair overlaps, instead of describing the object with the complex geometric properties of the object itself. This can greatly improve the efficiency of collision detection.

3.1.1.1. Bounding box classification. Common bounding boxes include spherical sphere, axial bounding box (AABB), directional bounding box (OBB) and discrete oriented polyhedron (K-DOP) [1]. The spectra of common bounding boxes are shown in Figure. 3.

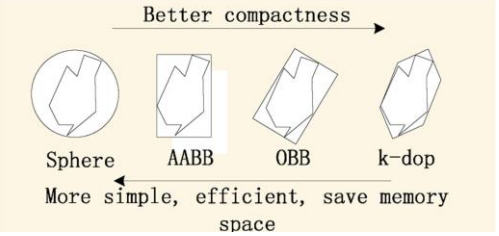

Figure 3. Typical bounding box [2].

DOI: 10.54254/2755-2721/37/20230505

3.1.1.2. Bounding box performance. For bounding box, in order to reduce calculation amount and improve calculation accuracy, the following two conditions need to be met [3]: Simplicity: All bounding boxes should be relatively simple, at least simpler than the surrounded objects, in order to ensure that this algorithm can improve efficiency rather than reduce efficiency. Compactness: All bounding box should be compact around the object, the closer the collision detection effect is better. The general relationship between performance and tightness of four common bounding boxes is shown in Figure 3, and a more specific comparison can be obtained from Reference 3.

3.1.1.3. Bounding box application. The basic application principle of Bounding box is very simple, that is, the object is wrapped with bounding box, and then the bounding box is used to replace the original object for a collision detection. The collision object is the object that needs actual collision detection in the next stage, otherwise it can be eliminated. The optimization of Bounding box mainly focuses on the selection of bounding box. Different adjustments can be made according to different objects mainly processed by the physics engine, and a variety of bounding boxes can also be combined with each other, namely, sphere screening and OBB screening [4]. However, it is obvious that since all objects must be detected, the time complexity of bounding box method must be O(N^2), and for physics engines with a large number of objects, such time complexity is unacceptable, so how to reduce the time complexity is also a major problem in collision detection.

3.1.2. Sort and Sweep

In order to reduce the time complexity, the Sort and Sweep algorithm can be used for further optimization. In this algorithm, it is still necessary to wrap all graphs with a cuboids (the length, width and height of the rectangle must be parallel to the coordinate axis), and then map the cuboids to the coordinate axis, and use an interval [b, e] to represent the large and small values mapped by the cuboid on the coordinate axis, and then put them into a list for sorting, as shown in figure 4.

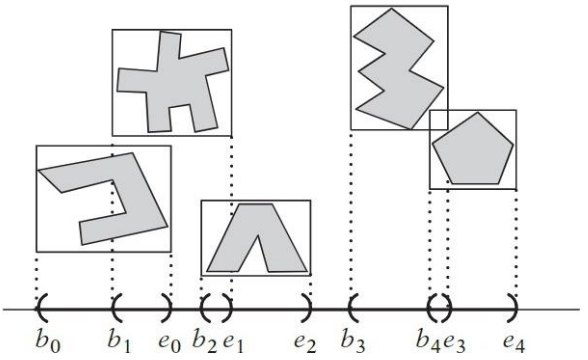

Figure 4. Schematic diagram of Sort and Sweep algorithm (Photo/Picture credit: Original).

The sorted list is then swept from front to back, and when point b is encountered, the interval is added to a current active interval list. Point e of the interval is encountered again, and the interval is removed from the active interval list. When point b is encountered, check the current activation interval list. The interval corresponding to b overlaps with all the intervals in the current activation interval list. A pair of cuboids that overlap on all axes is an overlapping pair of cuboids.

3.2. Narrow Phase

In the Narrow Phase, the program will conduct accurate collision judgment on the objects that may collide in the Broad Phase, so as to determine whether these objects really collide. Because of this stage, all collision detection should be accurate, so there is no optimization space for spherical, cube and other shapes of the object, these shapes have a clear and concise collision judgment mode, no need to optimize. For convex polyhedra collision detection, there is optimization space. The most basic multi-body collision detection is to detect the collision of all the faces of two convex polyhedra, if there is a collision

DOI: 10.54254/2755-2721/37/20230505

face, then find the intersection line and so on. The algorithm is completely mathematical, and the desired result can be obtained by the expression of the face vector and its side. For collision detection of polygons, there is a more efficient algorithm, GJK algorithm [5]. The core of this algorithm is that the graph formed by the Minkowski difference of two convex polyhedra contains the coordinate origin, and the pseudo-code is as follows.

```
function GJK_ intersection (shape p, shape q, vector init_ direction)
  vector A = Support (p, init_ direction) – Support (q, -init direction)
  simplex s = {A}
      vector D= -A
      loop:
        A = Support(p, D)-Supprot(q,-D)
        if dot (A, D) <0
              return false
        s = s ∪ A
            s, D, constains_ origin = NearestSimplex (s)
            if constains_origin
              return true
        return false
```

In addition to GJK algorithm, there are also some feasible convex polyhedral collision detection algorithms, such as Lin-Canny algorithm and V-Clip algorithm [6, 7], but these algorithms have no obvious advantages over GJK algorithm in dealing with convex polyhedral problems [8]. At the same time, GJK algorithm can still be further optimized by reducing sub-operations or by combining with other algorithms to reduce the number of iterations [9, 10].

4. Application of Parallel Technologies in Physics Engines

For the optimization of the physics engine, multithreading technology is one aspect that can be considered. But parallel programming itself has a number of critical issues to consider - Amuda's law, task granularity, the overhead of task switching, whether the program itself is parallel, synchronization of shared resources and CPU load allocation.

4.1. Regional Quadtree

To parallelize the physics engine, we need to introduce a concept - regional quadtree. A quadtree is a tree-like data structure in which each node has four sub-stages. The quadtree used in this article is called a regional quadtree, and its nodes are distinguished according to position. The whole world is divided into four regions, each subdivision is represented by a node, and then repeated for each subdivision, you can generate a quadtree of regions, each node of the tree has a list of objects that exist in the part that the node represents. When inserting an object into a tree, first check the center of the object and find its corresponding leaf. Then check whether the leaf boundary contains the object, if it is, add it to the node list, otherwise move to the parent node and repeat the above operation, so that you can ensure that each object is assigned to the smallest area possible.

4.2. Parallelism

With the support of regional quadtree, parallel operation can be easily implemented. The physics engine can assign one thread to the root node of the quadtree and one thread to each of its four children. Each thread processes only operate all objects that exist on its list, and when an object touches the boundary of the part which node represent, it moves up to the parent node. Therefore, when an object is likely to have a cross-thread collision, it is upgraded to the object of the root node, avoiding inter-thread communication. When the center of an object moves from the region corresponding to a node to the region corresponding to another node, it only needs to remove and reinsert the object, and the suitable location of the child node of the object can be found through the insertion function of the quadtree of the region. As for synchronization between threads, you can set up a barrier or simulate the effect of a

DOI: 10.54254/2755-2721/37/20230505

barrier by comparing the time of threads and adding meaningless loops (or inserting operations outside the physics engine, depending on the purpose of the physical engine) to keep the time difference between threads within a certain range.

5. Conclusion

A well-optimized naive physics engine, as discussed in this article, promises efficiency improvements of several magnitudes. Beyond this, its design facilitates scalability, leveraging multithreading technologies such as MPI to seamlessly expand across multiple cores. While the market boasts a plethora of mature, finely-tuned physics engines, the optimization techniques elucidated here serve a dual purpose. First, they shed light on the intricate operational principles underpinning physics engines. Second, they enhance development efficacy for developers and engineers working with these systems. As computer hardware continues to evolve, so too will the applications of physics engines. Their range of applications is set to widen, making notable inroads into domains like artificial intelligence and aerospace, among others. With this surge in relevance comes an inevitable increase in the demand for optimization strategies tailored for these engines. The future holds exciting prospects, and it's the hope that the insights offered in this paper will play a pivotal role in catalyzing the progression of physics engine development.

References

[1] Wang, W., Ma, J., & Liu, W. (2009). Research and Application of Collision Detection Based on Oriented Bounding Box. Computer Simulation, China, 26(9), 180-183.

[2] Wei, Y. W., & Wang, Y. (2001). Research on Fixed Direction Hull Bounding Volume in Collision Direction. Journal of Software, China, 12(7), 1056-1063.

[3] Ding, L. Z. (2006). The Research on Fast collision detection based on Oriented Bounding Box. Lanzhou University of Technology, China, pp. 13-54.

[4] Shen, Y. C., & Sun, X. Y. (2011). Research and improvement of collision detection based on oriented bounding box in physics engine. In 2011 IEEE 3rd International Conference on Communication Software and Networks (pp. 27-29).

[5] Gilbert, E. G., Johnson, D. W., & Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE Journal on Robotics and Automation.

[6] Lin, M. C., & Canny, J. F. (1991). A Fast Algorithm for Incremental Distance Calculation. In International Conference on Robotics and Automation.

[7] Mirtich, B. (1998). V-Clip: Fast and Robust Polyhedral Collision Detection. ACM Transactions on Graphics.

[8] Cameron, S. (1997). A Comparison of Two Fast Algorithms for Computing the Distance between Convex Polyhedra. IEEE Transactions on Robotics and Automation.

[9] Montanari, M., Petrinic, N., & Barbieri, E. (2017). Improving the GJK Algorithm for Faster and More Reliable Distance Queries Between Convex Objects. ACM Transactions on Graphics.

[10] Montaut, L., Lidec, Q. L., Petrik, V., et al. (2022). Collision detection accelerated: An optimization perspective. arXiv preprint arXiv:2205.09663.