國立交通大學

資訊科學與工程研究所

碩 士 論 文

有 效 率 的 複 雜 模 型 之 連 續 碰 撞 偵 測

Efficient Continuous Collision Detection for Complex Models

研 究 生：陳永錚

指導教授：莊榮宏　教授

中 華 民 國　九 十 六　年 八 月

Efficient Continuous Collision Detection for Complex Models

Student: Yung-Cheng Chen Advisor: Dr. Jung-Hong Chuang

Department of Computer Science and Information Engineering
National Chiao Tung University

ABSTRACT

We present an efficient algorithm to perform continuous collision detection for complex-shaped models. Given two models and their initial and final configurations, we use linear interpolation to calculate their in-between motion and examine if there exists collision between the two objects. Our algorithm is composed of three stages:(1) dynamic collision detection with bounding spheres of models;(2) conservative advancement with upper levels of bounding volume hierarchy; (3)conservative advancement with exact distance between two models. Moreover, in our system, user can assign a distance threshold, and our system will claim the collision of two objects if the distance between two objects is below the threshold. The output of our algorithm is the time of contact and colliding feature of two models if there exists collision. We tested our algorithm in several scenarios and compared with the method proposed by Zhang. The performance of our algorithm is $2 \sim 4$ times faster. Furthermore, the required memory of our algorithm is less than [ZLK06] by a factor of $3 \sim 4$ on a Pentium4 2.8GHz PC for complex-shaped models composed of $10k \sim 70k$ triangles.

Acknowledgments

I would like to thank my advisors, Professor Jung-Hong Chuang and Professor Wen-Chieh Lin, for their guidance, inspiration, and encouragement. Thanks to my colleagues in CGGM lab.: Tan-Chi Ho, Yu-Shuo Lio, Chia-Lin Ko, Yung-Cheng Chen, Zhi-Wen Zhang, and Ya-Jing Qiu. for their assistances and discussions. I would like to appreciate my parents. Without their love and supports, I counld not achieve my M.S. degree. Finally, I would like to appreciate my girl frient, Alice Wang. Thank you for being together with me through these years.I love you.

Contents

List of Figures

List of Algorithms

List of Tables

CHAPTER 1

Introduction

Collision detection(CD) is an indispensable component in numerous domains including computer graphics, robotics ,video games, etc. With the advancement of computer hardware, more realistic dynamic simulations can be executed in interactive rate. One of the components to achieve this goal is collision detection. Collision Detection has been studied for many years, but there are several issues ,which still need to be studied, including CD for deformable bodies, continuous CD, etc. In this paper, we concentrate on continuous CD for rigid bodies.

1.1 Collision Detection Algorithms

Most of collision detection algorithms can be classified into two categories: discrete collision detection and continuous collision detection. Most previous collision detection algorithms are discrete. These algorithms sample an object's motion in discrete time steps and detect whether the object interpenetrate within a time step. Because of the discrete sampling method, collisions which occur between two discrete time steps could be missed. This phenomenon is called "tunneling effect". However, discrete collision detection is much faster than continuous CD, and it is still widely used in many domains, especially in games.

In contrast to discrete CD, continuous CD considers continuous motions of objects and determine the first time of contact between any two objects if there exists collision

within a time interval. Therefore, continuous CD can prevent interpenetration and tunneling effect which could occur in discrete CD algorithm. With this improvement, a dynamic simulation can be performed more accurately, and this is also the reason why continuous CD draws more and more attention recently. However, the performance of most continuous CD algorithms is too low to be executed in interactive rates. Therefore, achieving better has become the main concern of recent researches on continuous CD.

1.2 Motivation

As computer hardware advances in recent years, physical correctness and computational robustness of dynamic simulation for computer animation have drawn more and more attentions. In order to present a realistic animation, correctly producing the collision response between moving objects is no doubt an important issue. Therefore, providing colliding information for a physics engine is important for producing correct collision response. However, in animation and computer games, collision detection systems adopt a discrete CD algorithm due to its fast computation and easy implementation. As we stated in the above paragraph, discrete CD algorithm has two vital problems, interpenetration and tunneling effect, which could result in incorrect animation in a dynamic simulation. In contrast to discrete CD algorithms, continuous CD algorithms do not have these two problems, so it can correctly determine if two objects collide and provide correct information for physics engine. However, computational cost is the main problem for continuous CD algorithms, and this is also the reason that makes continuous CD algorithm not widely adopted in practical use. Fortunately, recent researches on continuous CD have good advance on the performance issue. Zhang[ZLK06] proposed a continuous CD algorithm based on conservative advancement that is first proposed by Mirtich[Mir96]. This method must apply a sophisticated preprocession, convex decomposition[Ea01], on a model first, and Zhang adopts the method proposed in [Ea01]. Compared with the previous methods, the method proposed by Zhang has better performance and could attain interactive rate. However, there are still some drawbacks in Zhang's method. First, Zhang's method requires more memory space than other continuous CD methods in runtime. In real-time applications, memory is a precious resource; therefore, it is necessary for a collision detection system to keep the usage of memory as low as possible. Second, Zhang's method is only applicable to 2-manifold models, which limits the application of Zhang's method. Since most models are

produced by 3D scanner and artists, which would not match the 2-manifold requirement without additional processing. Due to these two drawbacks, Zhang's method is not suitable for many applications. Although its performance can attain interactive rate. In order to design a continuous CD algorithm that is suitable for real-time application and does not have the same limitation as Zhang's method, we need to adopt a different preprocessing method and apply an efficient approach on this data structure produced by preprocessing. In the above continuous CD algorithm, conservative advancement approach is a proper choice due to its high performance. As for the preprocessing method, in our survey , the method proposed by Larsen[LGLM] performs better than the method[Ea01] adopted in Zhang's method when the motion of an object has translation and rotation at the same time. Furthermore, the method proposed in [LGLM] is applicable for any polygonal models and requires less memory than Zhang's method. However, it would not have better performance than Zhang's method to only apply conservative advancement approach on the data structure produced in [LGLM]. There are still some important issues related to performance for us to develop on this new continuous CD algorithm. With these improvements, we propose a continuous CD algorithm which does not only avoid the problem in Zhang's method but also has better performance than Zhang's method on models with highly complex shape.

1.3 Organization

The rest of the thesis is organized as follows. In Chapter 2, we briefly survey discrete collision detection methods and continuous collision detection methods. In Chapter 3, we give an overview of our framework and then explain our method in detail. In Chapter 4, we test the performance of our algorithm on different benchmarks and compare with Zhang's method. Finally, we conclude the thesis in chapter 5.

CHAPTER 2

Related Work

While collision detection is a topic that has been extensively studied for several decades, continuous CD algorithms are continuously proposed in recent years. The most important thing to trigger the researches on continuous CD is the great advance of computer hardware in recent years. In this chapter, we will give a brief survey of discrete CD algorithms first ,and then introduce continuous CD algorithms.

2.1 Discrete Collision Detection

All collision algorithms can be roughly classified into two categories: broad phase and narrow phase. Broad-phase algorithms is mainly used for acceleration. In other words, broad-phase algorithms prune those object pairs which would not collide with each other; therefore, it could prevent the occurrence of the worst case, $O(n^2)$(n is the number of objects in a space.). After the procession of broad-phase algorithm, narrow-phase algorithm further determine whether two objects collide or not and finally produce colliding information of two colliding objects. In this section, we first introduce broad-phase algorithm and then narrow-phase algorithm.

The main idea of broad-phase algorithm is spatial partition. By subdividing the space into numerous regions and correspond the objects into the regions with their position, we can determine which pairs of objects could not collide with each other and prevent the further collision check of these pairs. The broad-phase algorithms include uniform grids,

hierarchical grids, octree, quadtree, k-d tree, binary spatial partition tree and etc[GASF94, HKM95, MW88, NAT90, TN87, YT93]. Besides these methods, there is still a special method called sweep and prune[CLMP95]. This method does not partition the space but projects the bounding volume onto the three axes and check if the intervals intersect with each other.

The purpose of narrow-phase algorithm is to check whether the surfaces of two objects intersect with each other. In order to avoid checking each pair of triangles of two objects, an acceleration scheme called bounding volume hierarchy(BVH) is constructed for each object. The elements of BVH are bounding volumes, and there are several types of bounding volumes, including sphere, axis-aligned bounding box(AABB), object-oriented bounding box(OBB), k-DOP(discrete-orientation polytopes), convex hull and etc[GLM96, He99, Hub93, Hub95, Hub96, HDLM96, JTT01, KHM+98, KPLM98, PG95, Qui94, Zac95]. For further details of discrete collision detection, we refer the readers to [LG98, LM03] for excellent surveys on the field.

2.2 Continuous collision Detection

In recent years, there are many methods proposed in the field of continuous collision detection. However, most of these methods can only handle convex objects. Few of these methods are applicable to arbitrary-shaped objects, and they can be classified into five categories: swept volume approach, adaptive bisection approach, kinetic data structure approach and conservative advancement approach. In the following, we will introduce these approaches.

2.2.1 Swept Volume Approach

The basic idea of swept volume approach is to compute the volume swept by an object within a time interval and then check whether the swept volume intersects with other objects in the environment. The method to produce a swept volume is explained in the survey[AMBJ02]. The main computational cost of swept volume approach is producing the swept volume , which usually cannot be performed in interactive rate. Another drawback of swept volume approach is that if the intersection with other objects is found, it still needs a backtrack mechanism to search for the time of contact. In our survey, there

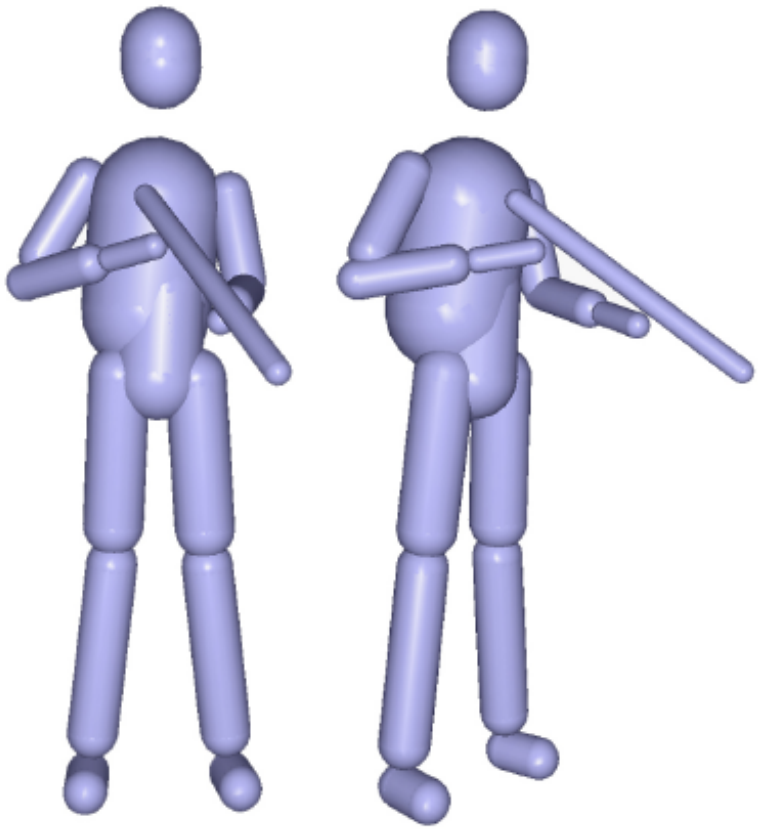


Figure 2.1: The avatar composed of several SSVs[RKLM04b].

is no good method to execute backtrack and this make the performance of swept volume approach worse. Redon et al. proposed a continuous CD method[RKLM04b] based on the swept volume method to detect whether an avatar collides with objects in a scene. However, the avatar is only composed of several simple convex shapes like capsule which is called sphere swept volume(SSV), as shown in figure2.1. During runtime, the swept volume formed by the avatar's movement is produced and is checked whether the swept volume intersect with the environment, as shown in figure2.2.

If the swept volume intersects with the objects in the environment, a backtrack mechanism is executed to find the time of contact. Because of the property of SSV, an efficient backtrack method is proposed by Redon, but it is only applicable to swept volume of SSV. For an avatar composed of several SSVs, this method can be performed in an interactive rate. In 2004, Redon also proposed a similar method for articulated models, such as robot arm. In this method, the swept volume of SSV is also used to check whether the articulated model is likely to collide with other objects. Therefore, an SSV is calculated to fit each link

Figure 2.2: The three steps to find time of contact for an avatar[RKLM04b].

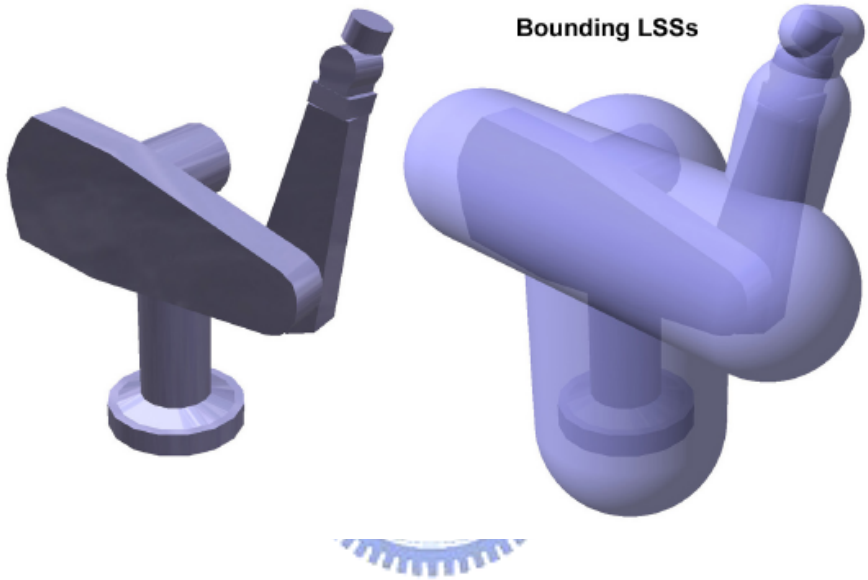


Figure 2.3: Each link of robot arm is bounded with a SSV [RKLM04a].

of the articulated model (shown in figure2.3). However, SSV is only the bounding volume of the articulated model. If the swept volume of SSV intersects with other objects, we need to use another continuous method[RKC02] to precisely check whether a link of the articulated model collides with other objects and find the time of contact. In this method, adaptive bisection approach[RKC02] is adopted to execute precise collision detection. The method runs in interactive rate if the movement of the articulated model is small, e.g., the rotational angle of each link is less than one degree in a time interval. If the movement of the articulated model is large, e.g. the rotational angle of each link is greater than thirty degrees, the method can not run in interactive rate.

2.2.2 Adaptive Bisection Approach

A few methods based on bisection were proposed, and we refer readers to an short survey in [RKC02]. The main idea of these method is to execute bisection test in a time interval to find the time of contact. However, these methods do not guarantee that the whole trajectory of the moving object is bounded and thus might miss a collision. In order to avoid this drawback, Redon et al. combined the interval arithmetic method with the bisection method in his continuous collision detection method[RKC02]. A good introduction to interval arithmetic for computer graphics can be found in [Sny92]. In this method, an OBB hierarchy[GLM96] is first constructed for each object. During runtime, the separating axis theorem of OBB is extended with interval arithmetic. In this way, the extent of the movement of an OBB is bounded with a larger box produced by interval arithmetic, and no collision would be missed. The method proposed by Redon et al. consists of two steps: continuous OBB hierarchy test and time interval subdivision. First, the continuous separating axis theorem is used to recursively test the intersection of OBBs in the hierarchy. After finishing the continuous OBB hierarchy test, some overlapped OBBs would be found at current time interval. At the second step, a heuristic method is used to determine whether the time interval needs to be subdivided or not. These two steps are continuously executed until the time interval is less than a threshold. The method proposed by Redon et al. showed good performance and was the most efficient method in the field of continuous collision detection before Zhang's method was proposed.

2.2.3 Kinetic Data Structure Approach

Kinetic data structure(KDS) is a framework for designing and analyzing algorithms for objects(e.g. points, lines, polygons)in motion [BGH97]. The KDS framework is an event-based algorithm which samples the state of different parts of the system as often as necessary for a specific task. For example, the task can be to preserve the separation of the moving objects, and it is called the attribute of the KDS. A KDS consists of a set of elementary conditions which are called certificates and guarantee the correctness of the attributes. For example, if a moving object exceeds a limited extent, it could collide with other objects. Here the limited extent is the certificate of the attribute. When a certificate fails, an event is inserted into the event queue. Its order in the queue is according to their earliest

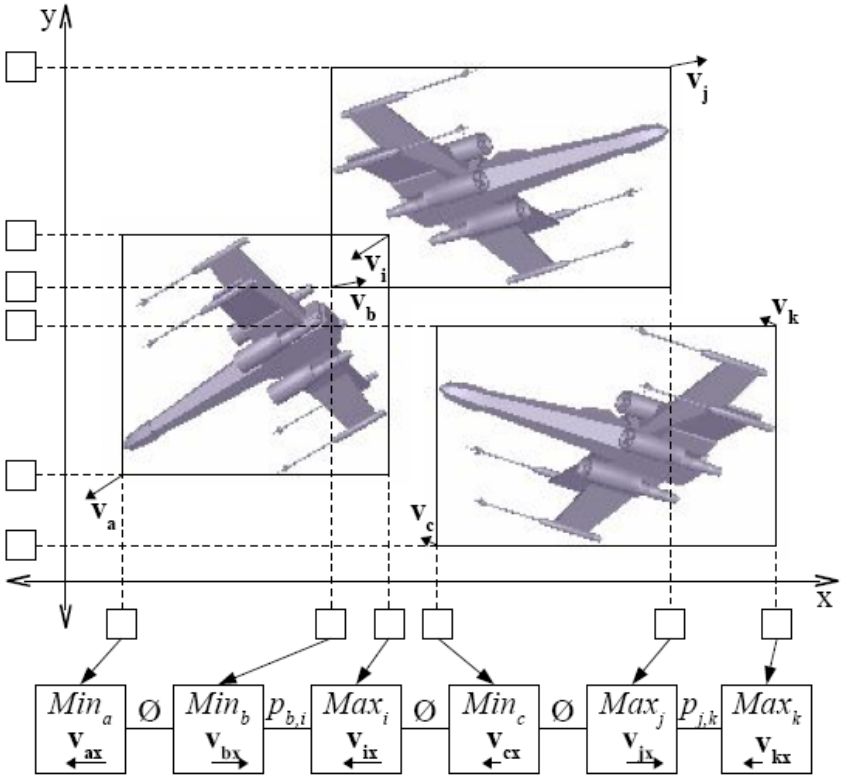


Figure 2.4: Project the vectors of velocity and bounding boxes of objects onto axes x and y [CS06].

failure time. In collision detection, the events are used to remind the system to produce response for collisions. Several methods baesd on KDS were proposed for continuous collision detection, but most of them are only applicable to convex objects or objects in 2D space. A method which combines sweep and prune and KDS was proposed in [CS06], and it can be applied to nonconvex objects. Traditional sweep and prune method projects the bounding box of objects onto three axises of 3D space and then check whether the three intervals projected on three axises intersect with the intervals of other objects at the same time. Moreover, the method is executed at discrete time step. The kinetic sweep and prune method projects not only the bounding box of an object but also the vector of velocity of an object onto three axises of 3D space. Therefore, it can predict when an interval of an object will intersect with other objects and insert an event in the order of the time of intersection into an event queue, as shown in figure2.4.

For each of the three axes, there is an independent event queue. The system would check the events in the three queues in the order of insertion time. An object is likely to collide with other objects if the system finds that the three intervals of the object intersect

with the intervals of other object at the same time. However, the method is designed as a broad-phased continuous collision detection algorithm, so a narrow-phased method is needed to check whether a possible colliding pair intersect with each other and find out the time of contact. There is another method based on KDS which is designed for deformable models[WZ06]. This method applies KDS on the AABB tree of a model and can predict when the verices of the model will move out of the AABBs which bound them. At first, the method checks the events from the leaf nodes of a AABB tree. If the vertices exceed the extent of their bound AABB, the method would update the AABB tree in a bottom-up manner. To detect collision with other objects, the method set a velocity which corresponds to the vertices for each face of an AABB; therefore, it can predict when an AABB will collide with other AABBs. However, there is an important requirement for the method: the flight plan of each vertex must be known. Without flight plan, the system could not correctly predict the time of contact. This is the limitation of the method.

2.2.4 Conservative Advancement Approach

The concept of conservative advancement(CA) approach was proposed by Mirtich [Mir96]. As Shown in figure2.5, the method is used to detect the collision of two convex objects and find out the time of contact in a time interval. Let $D_1(t)$ be an upper bound on the distance traveled by any point on body1 along **d** during the interval $[t_0, t]$. Let $D_2(t)$ similarly bound the distance traveled by any point on body2 along **d**. If a collision occurs at time t_c, then

$$D_1(t_c) + D_2(t_c) \geq d.$$

A lower bound on the time of contact can be found by replacing this inequality with an equality and solving for t_c. However, the method could not be applied on nonconvex objects. In order to make CA applicable to nonconvex objects, Zhang adopted a convex decomposition method[Ea01] in his method[ZLK06]. In Zhang's method, an object is first decomposed into numerous convex objects, as shown in figure2.6. Then, a convex hull tree is built in a bottom-up manner. During runtime, the tree traversal routine is executed on the two convex hull tree of two objects. When the tree traversal routine is executed, the bounding volume traversal tree(BVTT) is implicitly constructed (shown in figure2.7). The nodes of BVTT are composed of the nodes of two convex hull tree. Furthermore, the leaf nodes of BVTT is called front nodes which are most likely to collide with each other

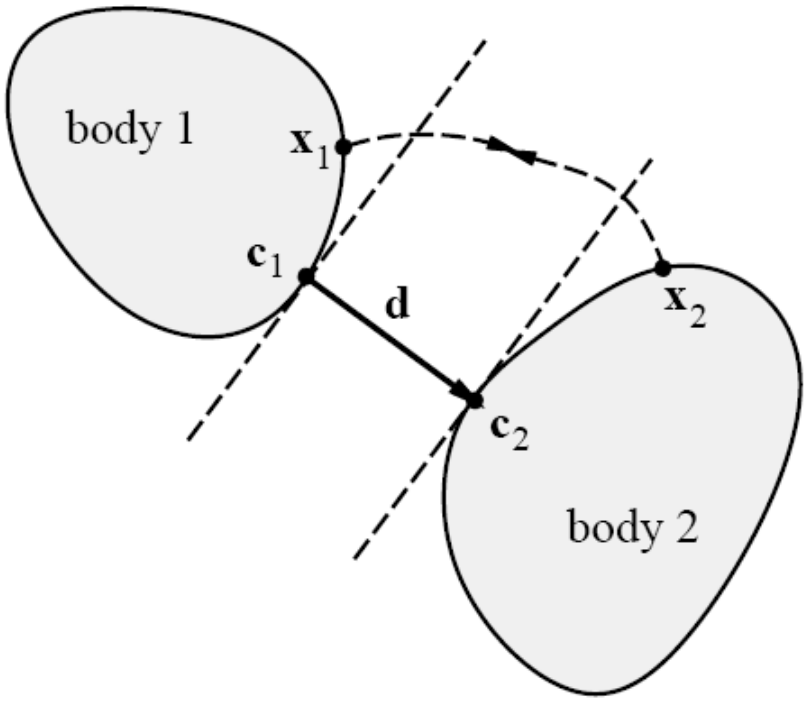


Figure 2.5: Finding a lower bound of the time of contact for convex objects [Mir96].

in these two convex hull tree. The concept of BVTT is first proposed in [LGLM] and is used to estimate the performance of a tree traversal algorithm. Zhang exploited BVTT to prevent the complexity of $O(n^2)$, where n is the amount of leaf nodes in a convex hull tree.

The algorithm of Zhang's method is shown in figure2.8. $\mathcal{A}$ and $\mathcal{B}$ are two nonconvex objects. $\mathbf{q}_0$ and $\mathbf{q}_1$ is the configurations of $\mathcal{A}$ when t = 0 and t = 1. In this algorithm, the loop is repeatedly executed until the distance between two objects is less than a threshold. In this paper, the threshold is set to 0.001 for an object with the diameter 100. Zhang also compared his method with the method proposed by Redon[RKC02], and his method is faster than Redon's method by a factor of 1.4 45.5 depending on benchmark scenarios. At present, Zhang's method has the best performance for the continuous collision detection of rigid nonconvex objects. However, Zhang's method requires more memory space to store the convex hull tree. This drawback makes Zhang's method less practicable. Some detail of Zhang's method will be mentioned in Chapter 3.

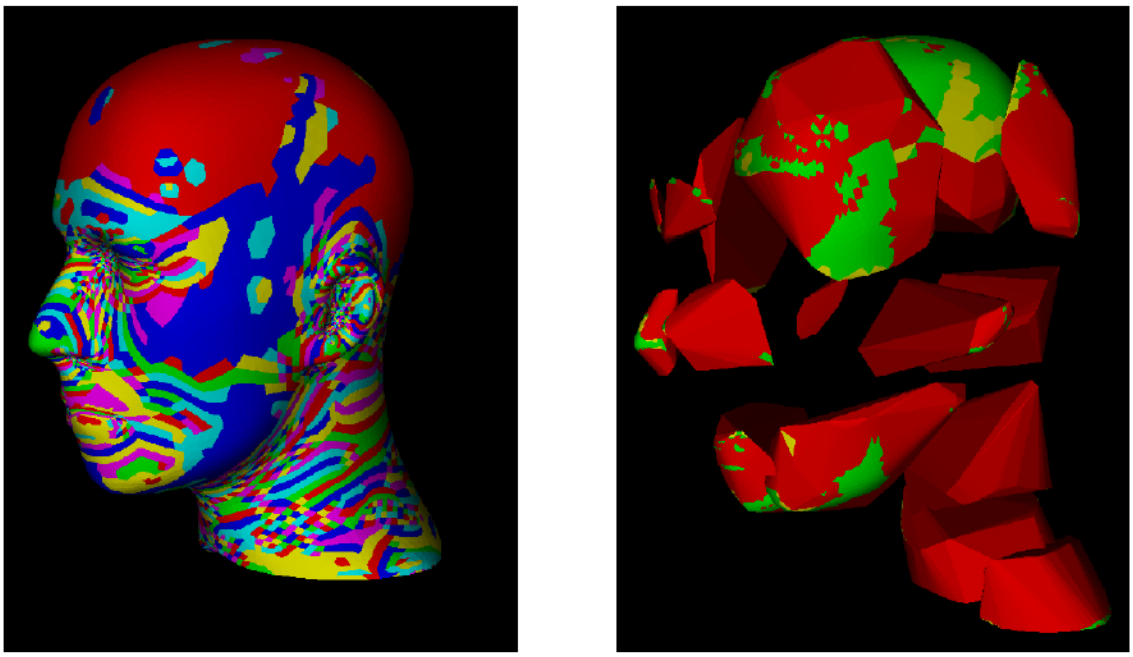


Figure 2.6: The left picture shows the leaf nodes of the convex hull tree with different color. The right picture shows an arbitrary level of the convex hull tree. [Ea01].

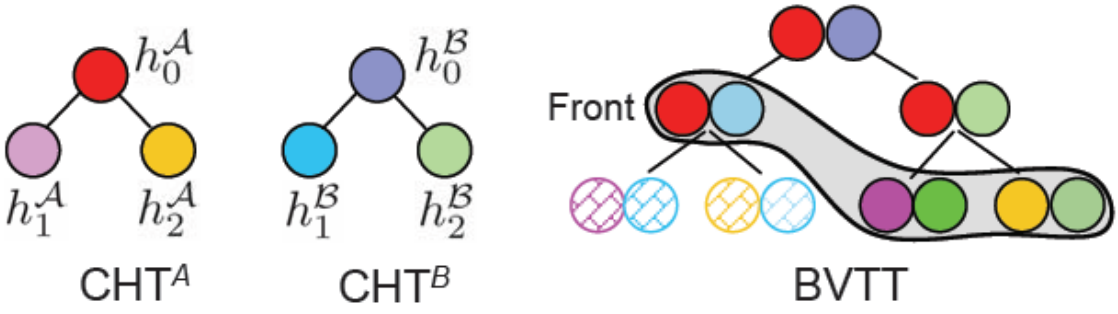


Figure 2.7: Two BVHs and their BVTT during distance query. Only solid-colored nodes are actually generated in BVTT [ZLK06].

Algorithm CCD using Convex Decomposition
Input: A moving polyhedron $\mathscr{A}$ and its initial and final configurations $\mathbf{q}_0, \mathbf{q}_1$, a fixed polyhedron $\mathscr{B}$.
Output: Calculate t_{TOC}.

1: As preprocess, decompose $\mathscr{A}$ and $\mathscr{B}$ into convex pieces and construct their convex hull trees, $CHT_{\mathscr{A}}, CHT_{\mathscr{B}}$.
2: Compute an interpolating motion $\mathbf{M}(t)$ from $\mathbf{q}_0, \mathbf{q}_1$.
3: **repeat**
4: Calculate $d(\mathscr{A}(t), \mathscr{B})$ between $\mathscr{A}(t)$ and $\mathscr{B}$ using $CHT_{\mathscr{A}}, CHT_{\mathscr{B}}$ and remember the pairs of front nodes $h_i^{\mathscr{A}}, h_j^{\mathscr{B}}$ in the trees that were traversed to calculate $d(\mathscr{A}(t), \mathscr{B})$. During the traversal, $d(h_i^{\mathscr{A}}, h_i^{\mathscr{B}})$ and closest direction vector $\mathbf{n}(i,j)$ are stored .
5: **for** each pair of $(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ **do**
6: Retrieve already calculated $d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ and $\mathbf{n}(i,j)$.
7: Calculate the motion bound $\mu(i,j)$ by projecting both the translational and rotation motion of $h_i^{\mathscr{A}}$ onto $\mathbf{n}(i,j)$.
8: Calculate $\Delta t_{i,j} = \frac{d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})}{\mu(i,j)}$.
9: **end for**
10: Find $\Delta t := \min(\Delta t_{i,j})$.
11: Advance $\mathscr{A}(t)$ by Δt.
12: **until** $d(\mathscr{A}(t), \mathscr{B})$ becomes less than a user-provided threshold
13: **return** t_{TOC}

Figure 2.8: The algorithm of Zhang's method [ZLK06].

CHAPTER 3

Continuous Collision Detection Based on Conservative Advancement

3.1 Overview

In this chapter, an overview of our continuous CD algorithm is first presented. Our algorithm consists of three stages, and the three stages form a pipeline shown in figure3.1. In processing stage, a bounding volume hierarchy composed of RSS(Rectangle Swept Sphere)[LGLM] is constructed for each model. The bounding sphere of each model is also calculated in processing stage. In run time, we execute collision detection as the order of the three stages. First, bounding spheres of two objects are checked whether they will intersect or not in this time interval. We can make use of the properties of sphere to easily solve the problem. After finishing first stage, we advance the moving object for a segment of the time interval which is the maximum advancement time that does not make the two bounding spheres intersect. If these two bounding spheres do not intersect during the whole time interval, we claim that the two objects will not collide and interrupt the execution pipeline. In second stage, we design a framework which is similar to [ZLK06]; however, a different bounding volume hierarchy is adopted in our framework. In this stage, we do not traverse to the bottom of BVH, but only traverse to the middle level of BVH. One of the reasons

is to use less execution time to filter the most cases in which two objects could not collide in this time interval. Because using the middle level of BVH to represent an object is tighter than using bounding sphere, most objects pairs which are determined as colliding pairs with bounding spheres could be determined as non-colliding pairs in second stage. In third stage, we make use of the traversal result of second stage and continue traversing to the bottom of BVH. In this way, we can prevent restarting the traversal of BVH from root node of BVTT and reduce the execution time of third stage. In third stage, we calculate the exact distance (the shortest distance) between two objects and execute conservative advancement with this distance. A distance threshold is assigned by user, in this thesis, the threshold is 0.001. If the distance between two objects is smaller than the threshold, we claim these two objects collide with each other. In the last two stages, an acceleration technique called front node tracking is applied in the traversal routine of BVH and will be introduced in the following paragraph.

3.2 Notations and Definitions

In this thesis, we use **boldface** to differentiate a **vector** from a scalar value (e.g., the origin, **o**). Let $\mathcal{A}$ and $\mathcal{B}$ be two polygonal models, and we assume that $\mathcal{A}$ is a moving object under rigid transformation **M** and $\mathcal{B}$ is fixed since we can find relative rigid transformation with respect to each other. Furthermore, the initial and final configurations of $\mathcal{A}$ are given as q_0 and q_1 at its corresponding time, t=0 and t=1. $\mathcal{A}(t)$ is the configuration of $\mathcal{A}$ under rigid transformation **M** at time t, i.e. $\mathcal{A}(t) = \mathbf{M}(t)\mathcal{A}$.

3.2.1 Conservative Advancement

The concept of conservative advancement(CA) is first proposed by Mirtich [Mir96]. It is used to calculate the time of contact of two convex objects under rigid motion. The execution of CA is repeatedly advancing $\mathcal{A}$ by $\triangle$t toward $\mathcal{B}$ and prevents the collision of $\mathcal{A}$ and $\mathcal{B}$. As shown in figure3.2, to finish an iteration of CA, we need the closest distance between $\mathcal{A}$(t) and $\mathcal{B}$ and the maximum motion track of $\mathcal{A}$(t) traced by some vertex **p** on $\mathcal{A}$. In each iteration of CA, we can advance $\mathcal{A}$ from time t to t + $\triangle$t without collision by calculating $\triangle$t with equation3.1. The proof of equation3.1 is presented in [Mir96], and we refer the readers to [Mir96] for detail. Note that CA proposed in [Mir96] is only applicable

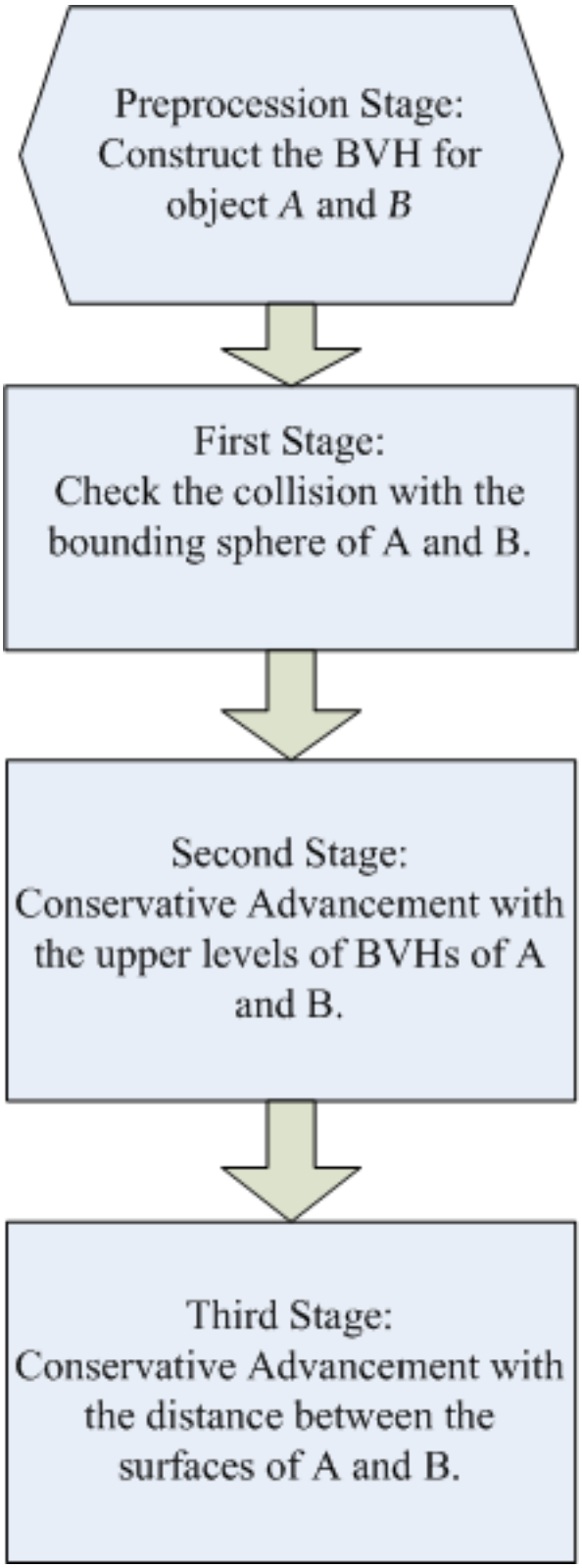


Figure 3.1: The flow chart of our framework.

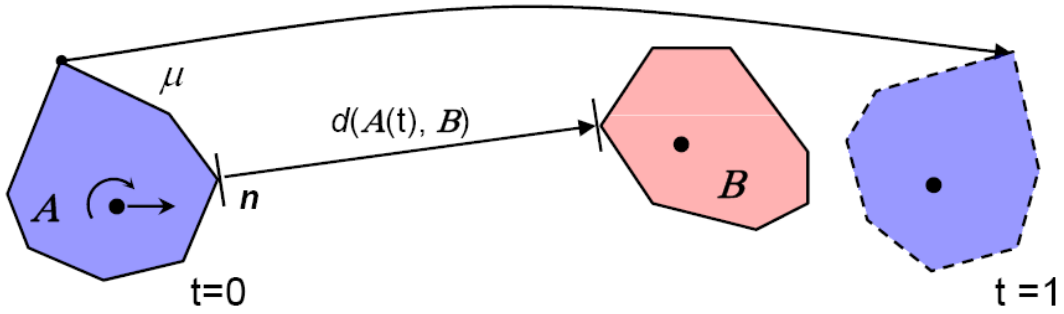


Figure 3.2: The concept of conservative advancement for convex objects.[ZLK06]

to convex objects; therefore, we can not directly apply CA to nonconvex objects. In our framework, each object has a BVH, and the components of BVH are convex polytopes. With this property, we can apply CA to the BVH of $\mathcal{A}$ and $\mathcal{B}$. The detail of our algorithm with CA approach will be explained in the following sections.

$$\mu \triangle t \leq d(\mathcal{A}(t), \mathcal{B})$$
$$\triangle t \leq \frac{d(\mathcal{A}(t), \mathcal{B})}{\mu} \tag{3.1}$$

3.3 First Stage

In this stage, we exploit the bounding spheres of two objects to advance the moving object for a segment of the time interval $\triangle$t and filter the cases in which two objects could not collide. One of the properties of sphere is rotation invariant. Therefore, we only need to consider the translation of the moving object. Figure3.3 depicts the condition of two spheres which are possible to collide with each other.

In figure3.3, θ is the angle between the translational vector **v** and the line segment formed by the centers of two bounding sphere. Two bounding spheres will collide in this time interval if the following equation is true.

$$\sin\theta \leq \frac{r_1 + r_2}{\|\mathbf{c}_1 - \mathbf{c}_2\|}$$

Therefore, before we calculate the maximum advancement time $\triangle$t of first stage, we can first calculate the angle θ and check whether it is greater than the maximum angle θ_M.

$$\theta_M = \arcsin \frac{r_1 + r_2}{\|\mathbf{c}_1 - \mathbf{c}_2\|}$$

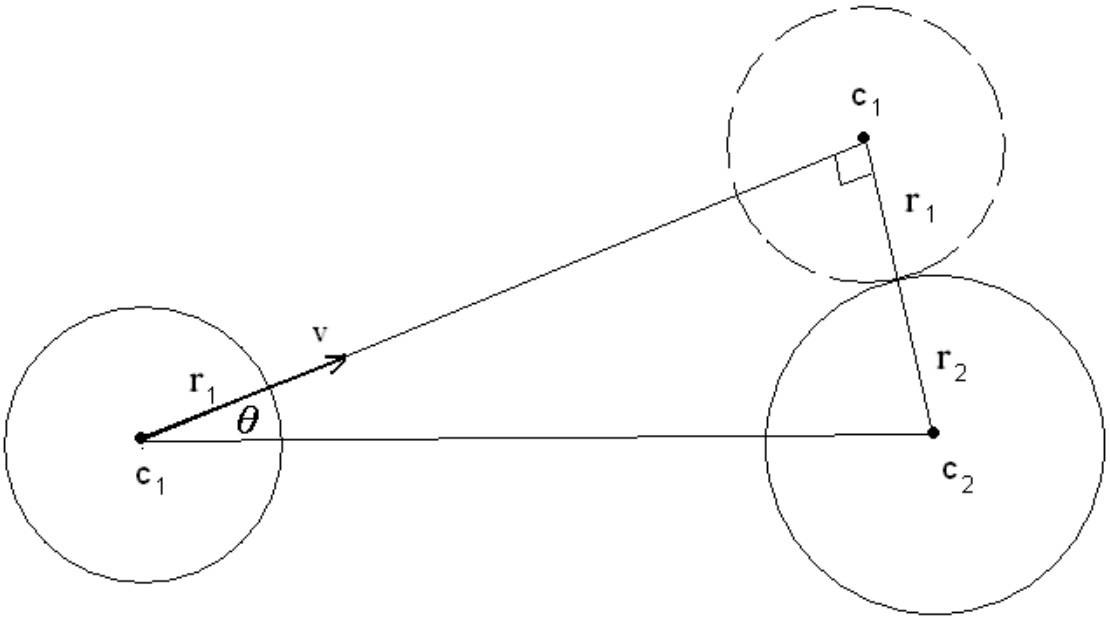


Figure 3.3: First Stage: Calculate the time of contact of two bounding spheres.

After the simple check, we must calculate the advancement time of the moving object. To determine whether two spheres intersect with each other, we only need to calculate the distance between the two centers and compare it with the sum of two radiuses. This approach can be represented with equation3.2.

$$\|(c_1 + s(t)) - c_2\|^2 = (r_1 + r_2)^2 \tag{3.2}$$

$$s(t) = \mathbf{v} \times t$$

In equation3.2, we must calculate the minimum value of t which is the first contact time of two spheres. In order to solve the value t, we can exploit some numerical methods ,e.g. Newton's method, to solve the equation. After calculating the maximum advancement time $\triangle$t in this stage, we set the moving object $\mathcal{A}$ to the configuration $\mathcal{A}(\triangle t)$ and start to execute the second stage.

3.4 Second Stage

The algorithm of this stage is shown in algorithm3.1. In the first iteration of the stage, we first traverse to half of the depth of BVH. For example, the depth of BVH is h, and half of the height is $\frac{h}{2}$. In other words, we treat the nodes of depth $\frac{h}{2}$ as leaf nodes when the tree traversal routine is executed. During the tree traversal, BVTT is implicitly constructed, and

the leaf nodes of BVTT are called front nodes. For each front node, we store the closest distance and normal vector in it and then make use of the information to calculate the advancement time $\triangle$t for each iteration. For each front node, we can use the information in it to solve equation3.1 to calculate an advancement time. After calculating the advancement times of all front nodes, the minimum advancement time is determined as the advancement time of the iteration. We use the minimum time to advance the whole object. In this way, we can ensure that any two objects will not collide after the advancement. In second iteration, we treat the nodes of the depth $\frac{h}{2}+1$ as leaf nodes and execute the same computation as first iteration. If the shortest distance of second iteration is less than the shortest distance of first iteration, we will continue to traverse the BVH down for one more level in third iteration, else we still traverse to the same level in third iteration. Therefore, as the distance becomes shorter, we will traverse to the deeper level of BVH. In this way, the exact distance between two objects would become shorter. We can roughly estimate the upper bound of the exact distance by calculating the sum of the shortest distance of the front node and the maximum radiuses of two BVs in the front node. However, when we traverse deeper, the computational cost of traversal routine also becomes higher. In our experiment, it has the best performance result for the overall continuous CD process to limit the maximum traversal depth of BVH to $\frac{3h}{4}$. After we traverse to the depth $\frac{3h}{4}$ in some iteration of second stage, we finish the second stage and start to execute the third stage.

3.4.1 Motion Bound Calculation

In conservative advancement approach, to calculate a tight motion upper bound μ is important, because it would effect the advancement time in each iteration. If the advancement time of each iteration is more accurate, the number of iterations could be reduced and the performance would be better. Therefore, we must estimate an upper bound as tight as possible to have a greater advancement time for each iteration. The method to estimate the upper bound of motion for an convex polytope is first proposed in [Mir96]; however, the equation is not efficient to calculate in run time. To make the calculation of motion bound more efficient, Zhang reduces the equation of motion bound and increase the efficiency of calculation in run time[ZLK06].

Given the interpolating motion $\mathbf{M}(t)$, a moving convex object $\mathcal{A}$ and a fixed object $\mathcal{B}$, we want to calculate the motion upper bound of $\mathcal{A}$. $\mathbf{M}(t)$ is the result of interpolating the

Algorithm 3.1: **Execution Loop of Second Stage**

Input: $\mathcal{A}$, $\mathcal{B}$ and $\mathbf{q}_0$, $\mathbf{q}_1$

Output: t_{TOC}

Initially, set the terminal level of the traversal routine to $\frac{h}{2}$;

while $d(\mathcal{A}(t), \mathcal{B}) \leq$ *the threshold* **do**

Traverse the BVHs of A and B to get $d(\mathcal{A}(t), \mathcal{B})$;

The pairs of front nodes$(n_i^{\mathcal{A}}, n_j^{\mathcal{B}})$ are stored during tree traversal. $d(n_i^{\mathcal{A}}, n_j^{\mathcal{B}})$ and $n(i, j)$ are also stored.;

foreach *pair of* $d(n_i^{\mathcal{A}}, n_j^{\mathcal{B}})$ **do**

Calculate μ with equation3.4;

Calculate $\triangle t_{ij} = \frac{d(n_i^{\mathcal{A}}, n_j^{\mathcal{B}})}{\mu}$;

end

$\triangle t = \min(\triangle t_{ij})$;

Advance $\mathcal{A}(t)$ by $\triangle t$;

if $d(\mathcal{A}(t), \mathcal{B}) \leq d(\mathcal{A}(t), \mathcal{B})$*of previous iteration* **then**

Increase the terminal level.;

end

end

return the total advancement time of second stage t_{second}

initial and final configuration of the moving object $\mathcal{A}$. Assume p_i is a vertex on $\mathcal{A}$, and p_i will trace out a trajectory $\mathbf{p}_i(t)$ when $\mathcal{A}$ undergoes **M**. We can calculate the velocity of $\mathbf{p}_i$ with the equation $\dot{\mathbf{p}}_i = \mathbf{v} + \omega \times \mathbf{r}_i(t)$. **v** is linear velocity ,and ω is rotational velocity. We can extract these two vectors from **M**, and they are constant during the time interval of $[0, 1]$. The undirected motion bound μ_u of $\mathcal{A}$ can be calculated by the equation:

$$\mu_u = \max_i \int_0^1 \|\dot{\mathbf{p}}_i\| dt \tag{3.3}$$

μ_u is the motion upper bound which can be used to calculate the advancement time, but we can still make the upper bound tighter by projecting $\mathbf{p}_i(t)$ onto **n**, which is the closest distance vector from $\mathcal{A}$ to $\mathcal{B}$. The equation of the new upper bound of motion is as follows:

$$\begin{aligned}
&\mu = \max_i \int_0^1 \|\dot{\mathbf{p}}_i \cdot \mathbf{n}\| dt \\
&\leq \max_i \int_0^1 \|\mathbf{v} \cdot \mathbf{n} + \omega \times \mathbf{r}_i \cdot \mathbf{n}\| dt \\
&\leq \mathbf{v} \cdot \mathbf{n} + \max_i \int_0^1 \|\omega \times \mathbf{r}_i \cdot \mathbf{n}\| dt \\
&\leq \mathbf{v} \cdot \mathbf{n} + \int_0^1 \max_i \|\omega \times \mathbf{r}_i \cdot \mathbf{n}\| dt \\
&\leq \mathbf{v} \cdot \mathbf{n} + \int_0^1 \max_i \|\mathbf{n} \times \omega \cdot \mathbf{r}_i\| dt \\
&\leq \mathbf{v} \cdot \mathbf{n} + |\mathbf{n} \times \omega| \max_i \|\mathbf{r}_i\| dt
\end{aligned} \tag{3.4}$$

In equation 3.4, **n** is stored in each front node and $\mathbf{r}_i$ can be precomputed and stored with each BV during the preprocessing stage.

3.5 Third Stage

In this stage, we will execute conservative advancement with the exact distance between two complex-shaped objects and finish the stage when the exact distance is below a threshold. In order to calculate the exact distance, we must traverse to the leaf nodes of BVH and calculate the distance between triangles. If we execute the traversal routine from the root node of BVH in each iteration, the computational cost of this stage is too expensive. Therefore, we keep track of the front nodes computed in the last iteration of second stage

and start our traversal routine from these front nodes. In this stage, we apply conservative advancement on the whole object which is not convex, so the algorithm in second stage can not be directly applied on this stage. For two convex objects, we can project motion upper bound onto the normal vector to calculate a tighter motion upper bound and guarantee the correctness of conservative advancement. However, for two nonconvex objects, we cannot guarantee the correctness of motion bound projection. To apply conservative advancement approach to nonconvex objects, we compute the length of the curve of motion upper bound and use the length to compute advancement time. This concept is also used in robotics to compute a conservative moving distance for a robot arm. In our observation, two complex-shaped objects will not intersect as long as the length of maximum motion trajectory of the moving object is equal to the exact distance. We can prove this observation and develop a lemma as follows.

Lemma 1. *If the length of maximum motion trajectory of the moving object is equal to the exact(shortest) distance between two objects with complex shape, these two objects will not intersect in this advancement.*

Proof: Assume that these two objects intersect under in this advancement. Therefore, we must reduce the motion trajectory and make it less than the shortest distance . This means the motion trajectory of all vertices on the moving object is less than the shortest distance. In this way, we can obtain a distance which is even shorter than the shortest distance from the motion trajectory of some vertex on the moving object, so the assumption contradicts with the base condition.

With this lemma, we can calculate an advancement time $\triangle$t in each iteration of third stage and develop a new motion bound equation , as shown in equation 3.5. In equation 3.5, r is the maximum radius of the moving object $\mathcal{A}$, that is, the radius of the bounding sphere of $\mathcal{A}$.

$$\begin{aligned} &\int_0^1 \|\mathbf{v}\| + r \cdot \|\omega\| dt \\ &= \|\mathbf{v}\| + \int_0^1 r \cdot \|\omega\| dt \\ &= \|\mathbf{v}\| + 2\pi r \cdot \frac{\theta}{2\pi r} \end{aligned}$$

$$= \|\mathbf{v}\| + r \cdot \theta = \mu \tag{3.5}$$

As long as the distance between two objects is positive, there must be a positive advancement time to make the moving object forward. Therefore, we can develop our algorithm base on lemma1.. The algorithm of this stage is shown as follows.

Algorithm 3.2: **Execution Loop of Third Stage**

Input: $\mathcal{A}$, $\mathcal{B}$ and $\mathbf{q}_0$, $\mathbf{q}_1$

Output: t_{TOC}

while *Distance between two object is greater than the threshold* **do**

$t_{TOC} = 0$;

Step1:Calculate the shortest distance between $\mathcal{A}$ and $\mathcal{B}$.;

{During this step, we also calculate the maximum effective radius.};

Step2:Calculate the undirected motion bound for $\mathcal{A}$ with equation 3.5.;

Step3:Calculate the advancement time $\triangle t$ with equation3.1.;

Step4:Advance $\mathcal{A}$ by $\triangle t$.;

$t_{TOC} = t_{TOC} + \triangle t$;

$\mathcal{A} \Rightarrow \mathcal{A}(t_{TOC})$;

end

return the time of contact t_{TOC}

At the first step of algorithm3.2, we calculate the shortest distance between two objects with the front nodes computed in second stage. The detail of the traversal algorithm will be explained in Section 3.5.1 . At the second step, we calculate the motion bound of $\mathcal{A}$ with equation3.5, and then the advancement time of the iteration is calculated. Finally, $\mathcal{A}$ is advanced with $\triangle t$. We finish the execution loop when the shortest distance is less than the threshold.

3.5.1 First Step of Third Stage: Distance Query

The first step of algorithm3.2, distance query, is to calculate the shortest distance between two objects. In order to prevent redundant BV tests during BVH traversal, we exploit the front nodes which is the result of BVH traversal computed in the last iteration of second stage. Because these front nodes are the internal nodes which depths are less than $\frac{3h}{4}$ in the BVH, we must further traverse to the leaf nodes of the BVH which contain the triangles

of the model. In this step, reducing the BV tests is a key factor which would improve the performance of traversal routine. The algorithm of a distance query routine is as follows.

Algorithm 3.3: **The Routine of Distance Query in Third Stage**

Input: A queue with front nodes inside

Output: The shortest distance d_s

Traversal from the front node f_s in which the shortest distance of the previous iteration exists .;

After the traversal of f_s, we get d_s.

foreach *front node* f_i *in the queue* **do**

  if $d(f_i.BV1, f_i.BV2) \leq d_s$ **then**

    Traverse f_i and get the distance d_i;

    if $d_i \leq d_s$ **then**

      $d_s = d_i$;

      $f_s = f_i$;

    end

  end

end

return d_s;

At first, we input an initial distance value as upper bound and exploit the upper bound to reduce the BV tests. Therefore, we must assign a distance upper bound as small as possible to efficiently execute the distance query routine. At the beginning of this step, we will start traversal from the front node in which the shortest distance exists. After the first traversal, we obtain an initial distance upper bound and then traverse other front nodes with this distance upper bound. If we find a distance result of other front node is smaller than the initial upper bound, we will replace the upper bound with this result and continue to traverse other front nodes. After finishing all the traversals of front nodes, we obtain a front node with the shortest distance and then make use of this front node to calculate distance upper bound in the next iteration. In our distance query routine, each front node computed in second stage will implicitly construct a BVTT, and the leaf nodes of the smaller BVTT are the front nodes of third stage. In the first few iterations of third stage, we calculate the difference of front nodes of third stage between two successive iterations. If the difference is small enough, we will keep track of all front nodes of third stage and start our traversal from these front nodes in the next iteration. The reason is that the advancement time is

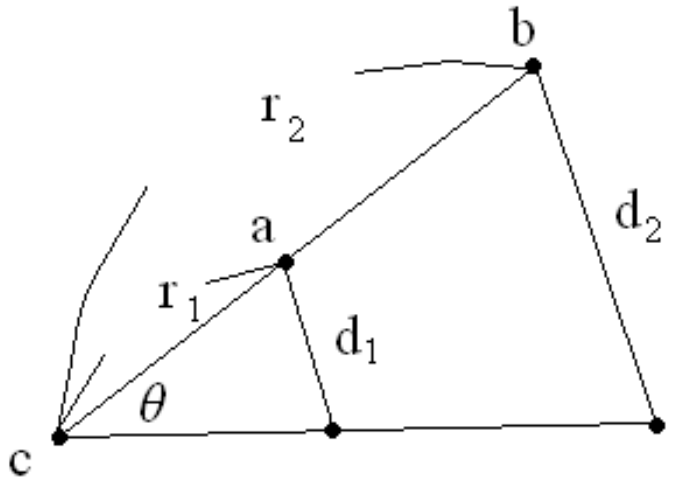


Figure 3.4: Moving distance of two vertices with different radiuses after the rotation.

smaller as the distance between two objects become shorter; therefore, the position of the moving object only change a little bit. In other words, the traversal results of the two successive iterations would not change too much, so there exists high coherence between the two iterations. The coherence is called temporal coherence. With this coherence, we can further improve the performance by starting our traversal from the front nodes of third stage. In our experiments, the performance would have a remarkable improvement when the difference is less than 20%.

3.5.2 Maximum Effective Radius Query

In equation3.5, the linear velocity **v** is constant for all vertices on a model while the rotation speed is a variable for each vertex. In order to estimate a tiger upper bound μ, we have to calculate a radius value r as small as possible. Consider the case shown in figure3.4, two vertices **a** and **b** rotate around the center **c**. After rotate with an angle θ, the values of moving distance of **a** and **b** are d1 and d2, and the ratio of r1 to d1 is equal to the ratio of r2 to d2. For two BVs of the moving object, we can use the property from figure3.4 to determine whether it will collide with the other object after the rotation of angle θ. Given two BVs, B1 and B2,from the moving object $\mathcal{A}$ and their shortest distance, d1 and d2, to the other object $\mathcal{B}$, we assume that the radius of B1, r1, is smaller than the radius of B2,

r2. To avoid intersection with object $\mathcal{B}$, we have to consider the shortest distance of BV1 and BV2. The safe moving distance of BV1 is its shortest distance to object $\mathcal{B}$, and so is BV2. If we rotate BV1 with some angle and make the moving distance equal to d1, the moving distance of BV2 can be calculated with the property from figure3.4. The angle is $d2' = d1 \times \frac{r2}{r1}$. If d2' is greater than d2, it is possible that the vertices in BV2 could intersect with $\mathcal{A}$. Therefore, the radius r1 is not a good candidate for the calculation of the motion upper bound, so we choose the radius of BV2, r2. If d2' is less than d2, BV2 could not intersection with object $\mathcal{B}$, and the radius of BV1, r1, is a good candidate. In order to determine the best radius to calculate the motion upper bound, we must examine the moving object $\mathcal{A}$ thoroughly. However, it is not a good idea to check all vertices of $\mathcal{A}$, and the cost is too expensive. In our observation, the front nodes produced from distance query is the parts which are most likely to collide between two objects, and the quantity of front nodes is far less than the leaf nodes of two objects. Therefore, we can determine a best radius among these front nodes. In our experiments, with this improvement we can reduce about 10% of iterations in third stage.

CHAPTER 4

Results

We have implemented our method by using C++ and OpenGL on a PC equipped with Intel P4 2.8GHz CPU and 512MB main memory. In our program, we adopt the library PQP[LGLM] to calculate the distance between nonconvex models. We compared the performance of our method with FAST[ZLK06]. Fortunately, the program FAST[ZLK06] could be downloaded from their website. Therefore, we can compare the two methods in the same environment.

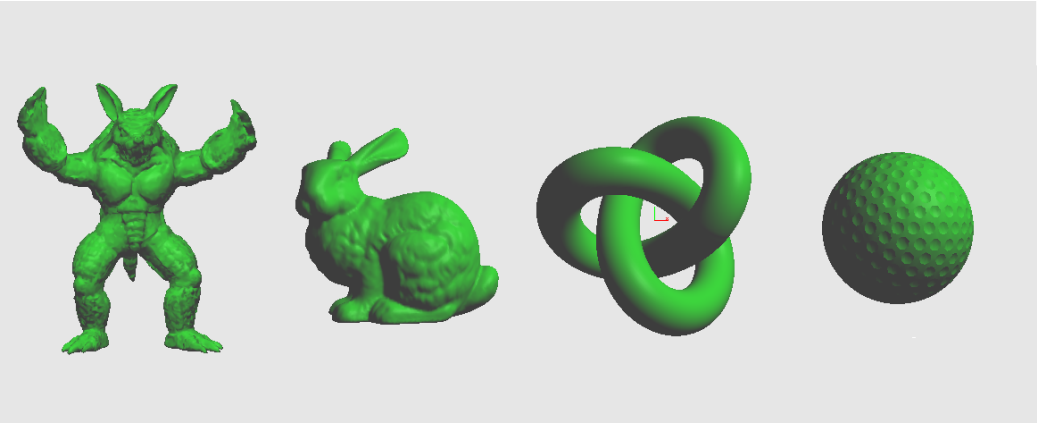


Figure 4.1: Models used in our benchmark.

4.1 Performance Comparison with 2-manifold Models

In order to measure the performance of our method and compare with FAST, we benchmarked our method with 2-manifold models which contain different numbers of polygons and different shapes, as shown in figure4.1. We adopt the benchmark scenario from [ZLK06] to compare the performance of our method with the performance of FAST. In the benchmark scenario, an object is shot from a random configuration $\mathbf{q}_0$(the red object) to another random configuration $\mathbf{q}_1$(the blue object). The rotation angle between the two configurations ,$\mathbf{q}_0$ and $\mathbf{q}_1$, is $\frac{\pi}{4}$ around a random axis. We want to detect whether the moving object will collide with the fixed object(the yellow object) and find the time of contact. The inbetween configuration of the moving object is calculated with linear interpolation. In the benchmark scenarios, we test the performance of our method for 200 frames, and the orientation of the yellow object is changed with the frame number. The green object is the inbetween configuration found by our method, and its distance to the yellow torus is less than the user-defined threshold which is set to 0.001. Moreover, the models are scaled so that their diameter are 100 units in OpenGL. The attributes of the models used in our benchmark and the comparison of performance and memory usage are described as follows.

1. torusknot vs torusknot (Figure4.3): Each torusknot consists of 35000 triangles. The performance result of our method is shown in figure4.3. The average time to detect the collision with our method is 5.09ms, and the time of the worst case is 13.19ms. The average time of FAST is 10.13ms, and the time of the worst cast is 103ms. In average, the performance of our method is 2 times better than FAST. In our method, to store a torusknot in main memory needs 11.5MB. In FAST, it needs 39MB for a torusknot, and it is about 3 times more than our method.

2. armadillo vs armadillo (Figure4.4): The model, armadillo, consists of 43000 triangles. The performance result is shown in figure4.4. The average time to detect the collision in our method is 6.6 ms, and the time of the worst case is 30.8ms. The average time of FAST is 14.46ms, and the time of the worst cast is 102.8ms. In average, the performance of our method is 2.19 times better than FAST. As for the memory issue, our method needs 14.5MB for an armadillo while FAST needs 50MB.

Test Model	Torusknot	Armadillo	Bunny	Golf
Our method Avg.	5.09	6.6	5.56	4.25
Our method Worst	13.19	30.8	15.7	8
FAST Avg.	10.13	14.46	21.4	10.6
FAST Worst	103.3	102.8	296.4	300

(a)The average time and the time of the worst case (in ms).

Test Model	Torusknot	Armadillo	Bunny	Golf
Our method	11.5	14.5	22.5	3.3
FAST	39	50	86	19

(b)The memory space for each model (in MB).

Table 4.1: The comparison between our method and FAST.

3. bunny vs bunny (Figure4.5): A bunny consists of 70000 triangles. The performance result is shown in figure4.5. The average time to detect the collision in our method is 5.56ms, and the time of the worst case is 15.7ms. The average time of FAST is 21.4 ms, and the time of the worst cast is 296.4ms. In average, the performance of our method is 3.8 times better than FAST. The memory space to store a bunny in our method is 22.5MB while it needs 86MB in FAST.

4. golf vs golf (Figure4.6): The model, golf, consists of 10000 triangles. The performance result is shown in figure4.6. The average time to detect the collision of our method is 4.25 ms, and the time of the worst case is 8 ms. The average time of FAST is 10.6 ms, and the time of the worst cast is 300 ms. In average, the performance of our method is 2.5 times better than FAST.

The comparison of the performance between our method and FAST is shown in table4.1(a), and the comparison of the memory space needed for each model in our method and FAST is shown in table4.1(b).

4.2 Performance Tests with Non-Manifold Models

In this section, we benchmark our method with some non-manifold models. Because FAST is not applicable to non-manifold models, we do not compare the performance in this section. We use the same benchmark scenario as section4.1 to measure the performance of our

method for non-manifold models. The attributes of models and the performance of each test are described as follows.

1. teapot vs teapot (Figure4.7): A teapot consists of 32000 triangles. The average time to detect the collision in our method is 5.4ms, and the time of the worst case is 16ms.

2. dragon vs dragon (Figure4.8): A dragon consists of 47000 triangles. The average time to detect the collision in our method is 5.4 ms, and the time of the worst case is 17ms.

3. car vs car (Figure4.9): A bunny consists of 57000 triangles. The average time to detect the collision in our method is 7.5ms, and the time of the worst case is 26ms.

4.3 Performance Comparison under Dynamic Simulation

In this section, we integrate our method with a physics engine, ODE, to detect the collisions in a dynamic environment. We also compare the performance of our method with the performance of FAST in dynamic simulations. The benchmark scenario is shown in figure4.2. In the figure, we drop a red object to collide with a yellow object and use ODE to deal with the response after collision. In our benchmark, we change the orientation of the yellow object and then drop the red object for 200 times. The attributes of the models used in our benchmark and the comparison of performance are described as follows.

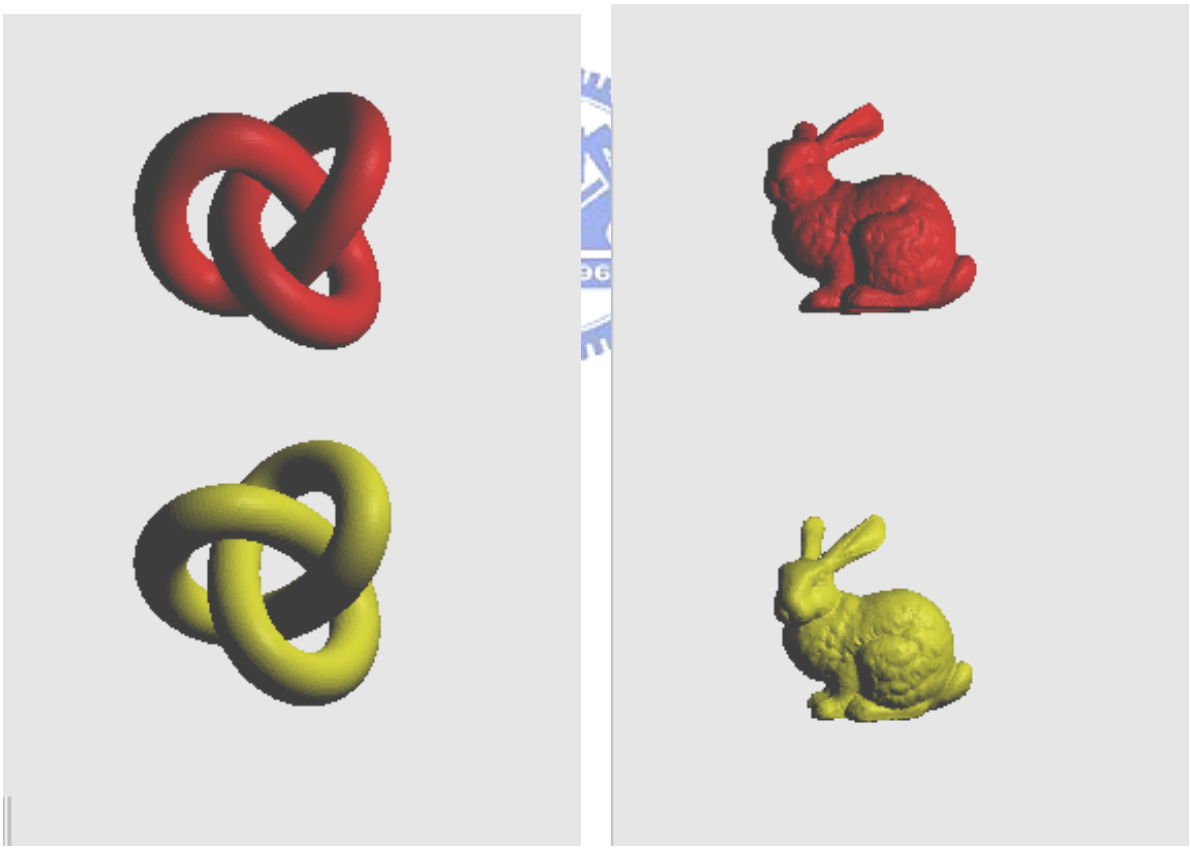
1. Dynamic torusknot (Figure4.2(a)): A torusknot consists of 35000 triangles. The performance result of our method is shown in figure4.10. The average time to detect the collision in our method is 8.13ms, and the time of the worst case is 28.6ms. The average time of FAST is 17.88ms, and the time of the worst cast is 363.23ms. In average, the performance of our method is 2.2 times faster than FAST.

2. Dynamic bunny (Figure4.2(b)): A bunny consists of 26000 triangles. The performance result is shown in figure4.11. The average time to detect the collision in our method is 7.6ms, and the time of the worst case is 39.4ms. The average time of FAST is 20.8 ms, and the time of the worst cast is 299.76ms. In average, the performance of our method is 2.7 times faster than FAST.

The comparison of the performance between our method and FAST is also shown in table4.2.

Test Model	Torusknot	Bunny
Our method Avg.	8.13	7.6
Our method Worst	28.6	39.4
FAST Avg.	17.88	363.23
FAST Worst	20.8	299.76

Table 4.2: The comparison between our method and FAST.



(a) (b)

Figure 4.2: (a) A dropping torusknot collides with a torusknot; (b) A dropping bunny collides with a bunny.

Test Model	Torusknot	Armadillo	Bunny	Golf
Triangles	35000	43000	70000	10000
Convex hulls	4888	10278	16673	4117

Table 4.3: The number of convex hulls for each model.

4.4 Analysis

Table4.3 shows the convex pieces for each model which is produced with [Ea01]. These convex pieces are the leaf nodes of the convex hull tree of each model created in FAST. Comparing the convex pieces with the number of triangles of each model, we can find that the number of triangles is only $4 \sim 7$ times more than the number of convex pieces. Moreover, the computational cost of distance query for convex hulls is much higher than the BV used in our method. Therefore, our method is more efficient in the distance calculation for the models in the benchmarks. The detail of the algorithm to compute the distance between two convex hulls is explained in [Ea00]. A technique called front nodes tracking is used in FAST to reduce the number of distance query of convex hulls; however, in some cases of the benchmarks, the coherence of front nodes is not good enough. In the performance results of different models, we can find that there exist some spikes for all the benchmarks of FAST, and the values of these spikes are much higher than the average time. Another factor to cause the phenomenon might be the effect of cache miss of CPU. FAST needs a large amount of memory space for each model in these benchmarks. If the cache memory of CPU is not enough to store the data for the computation, there could be a large amount of cache misses and context switches. In contrast to FAST, our method require relatively low memory space ,and the computational cost to refresh the cached front nodes is lower than FAST. Therefore, we can prevent the phenomenon which happens in the benchmarks of FAST.

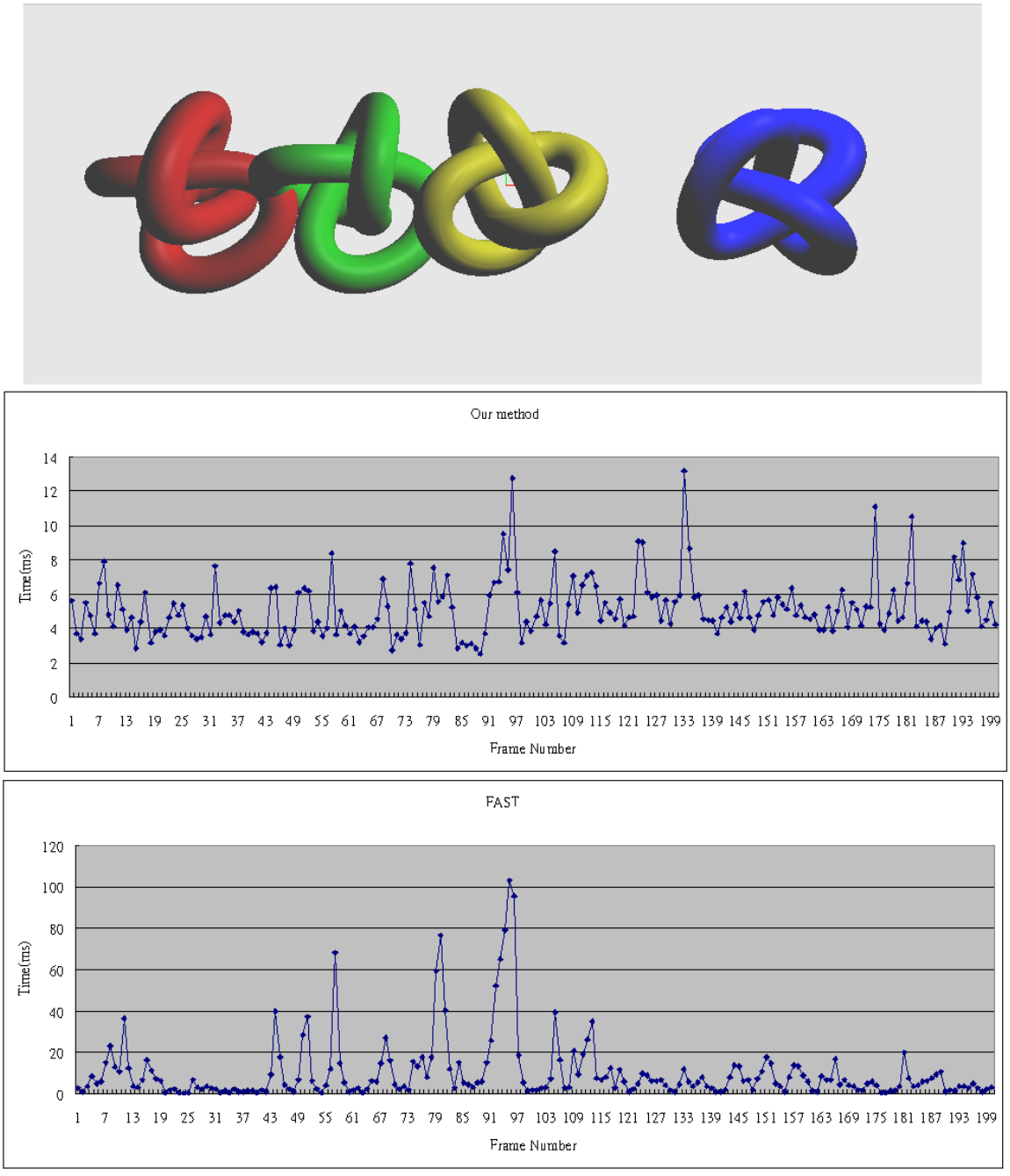


Figure 4.3: Torus vs Torus. The test results of our method and FAST.

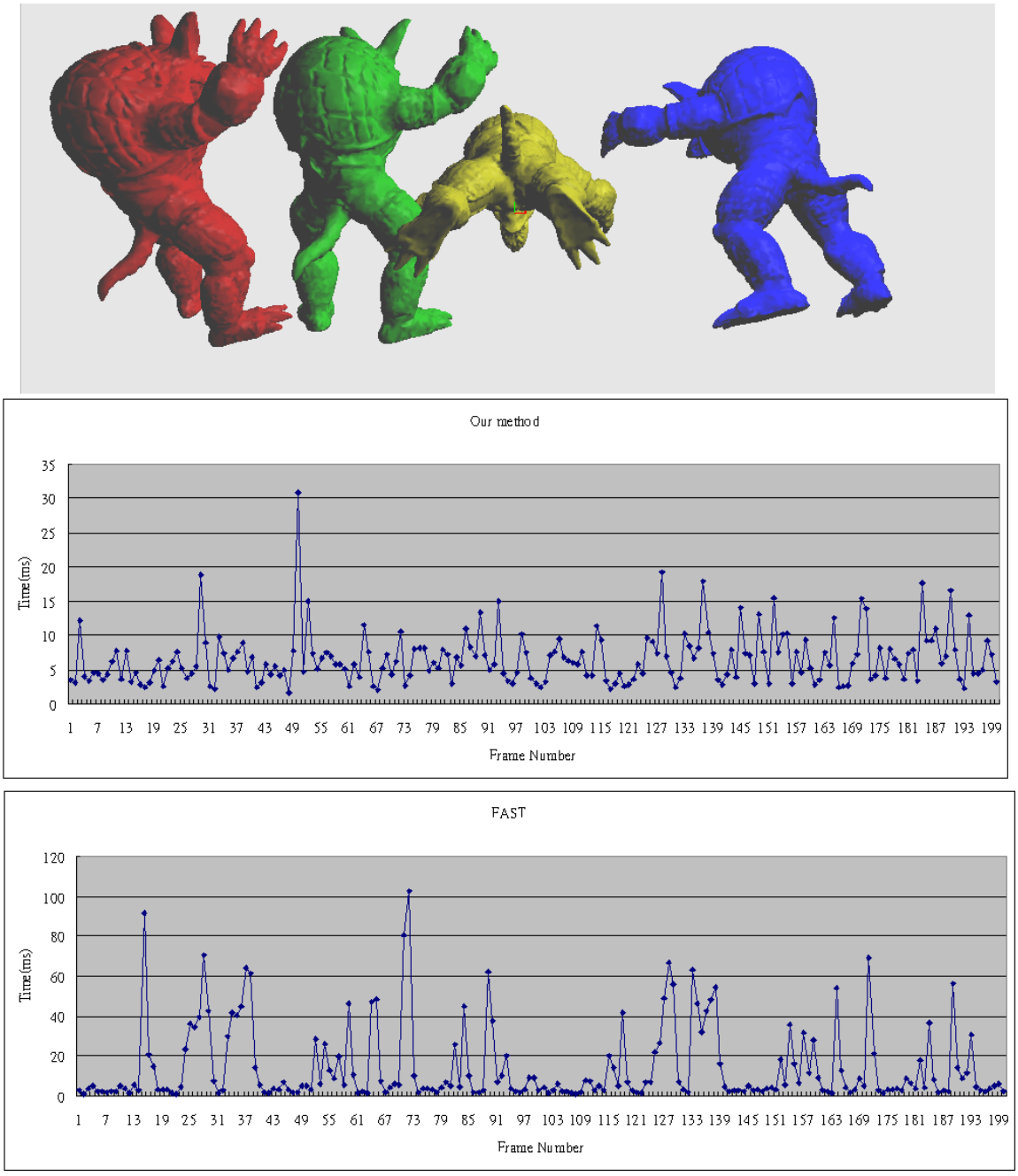


Figure 4.4: Armadillo vs Armadillo. The test results of our method and FAST.

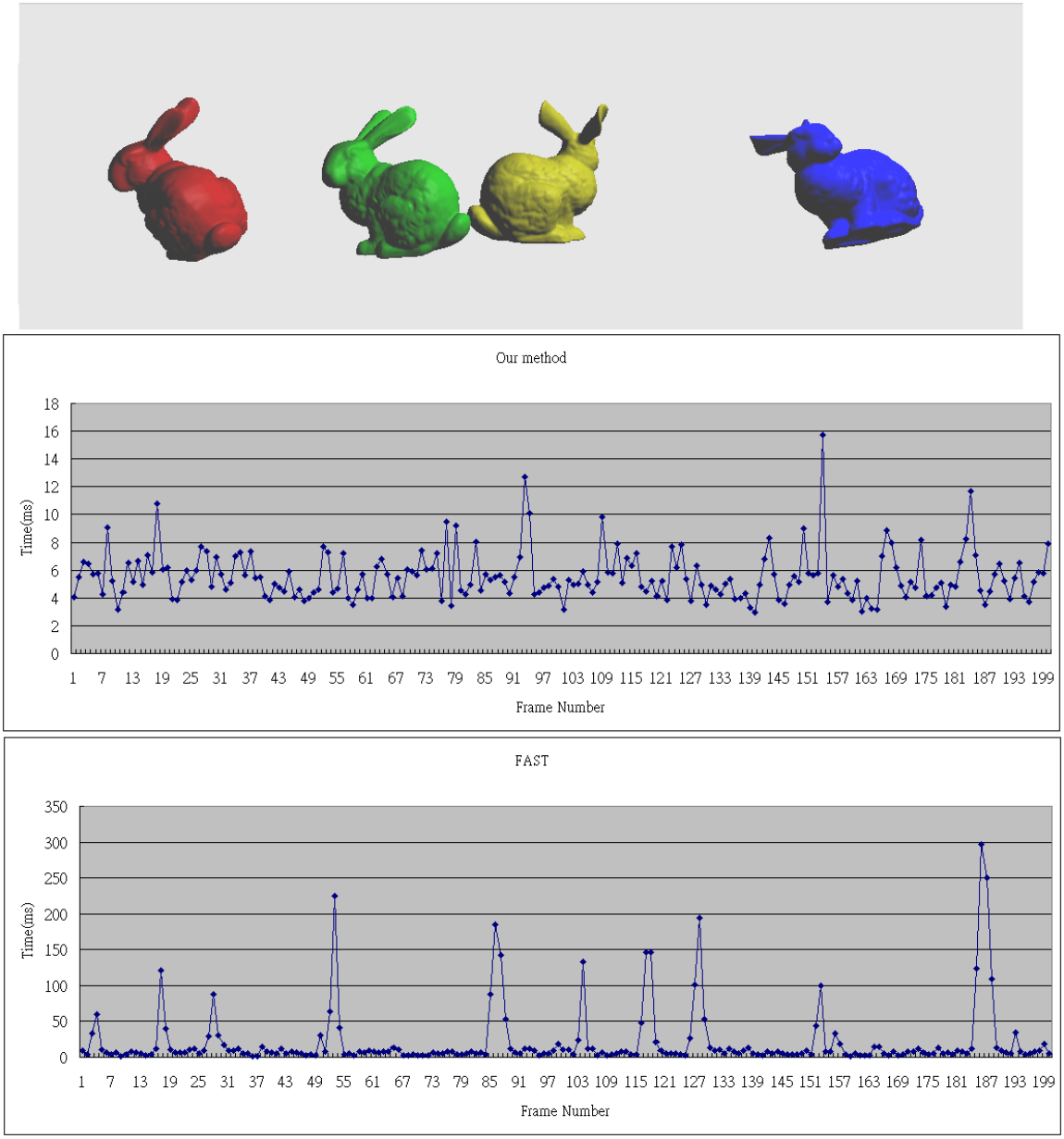
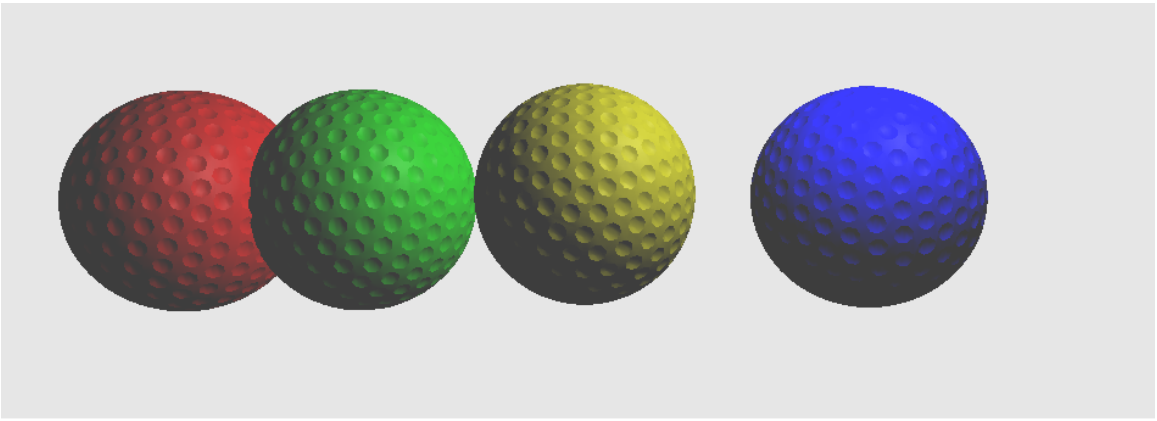
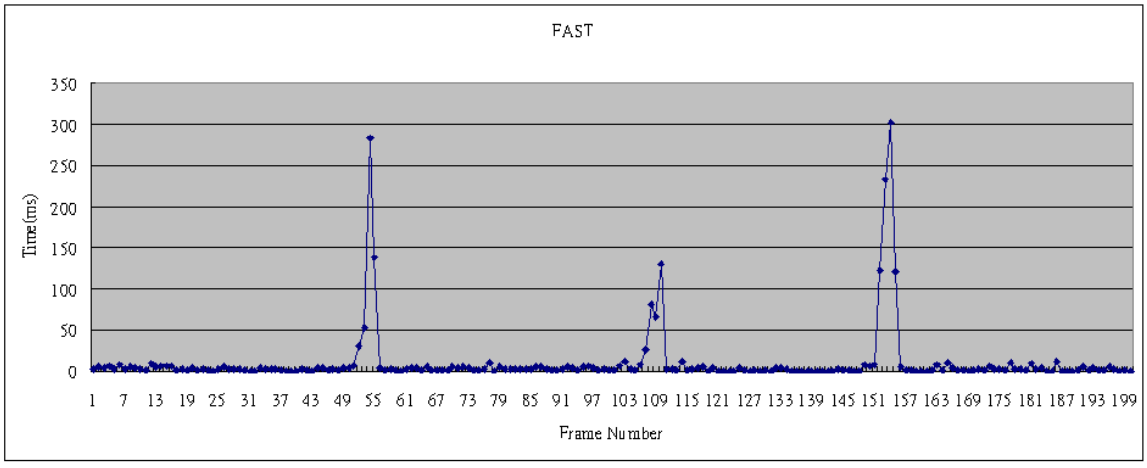


Figure 4.5: Bunny vs Bunny. The test results of our method and FAST.





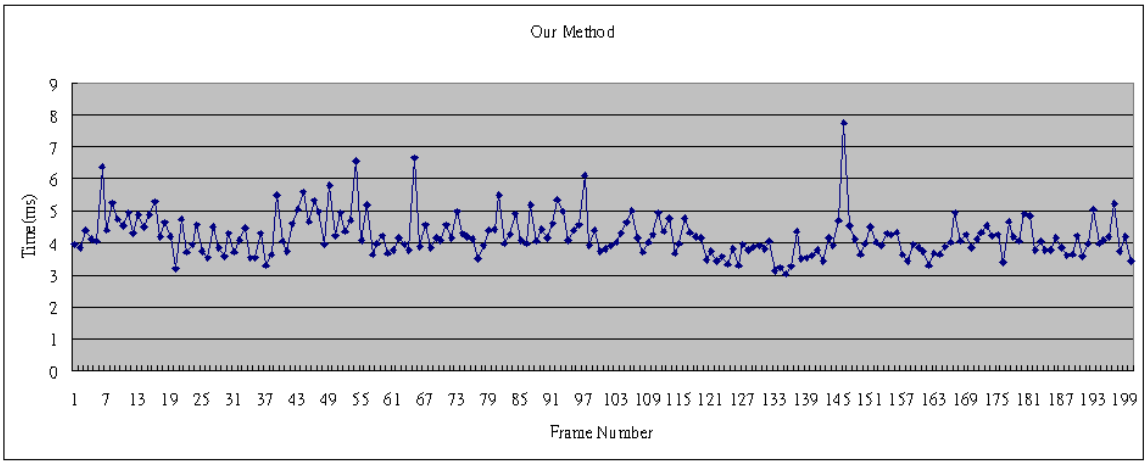


Figure 4.6: Golf vs Golf. The test results of our method and FAST.

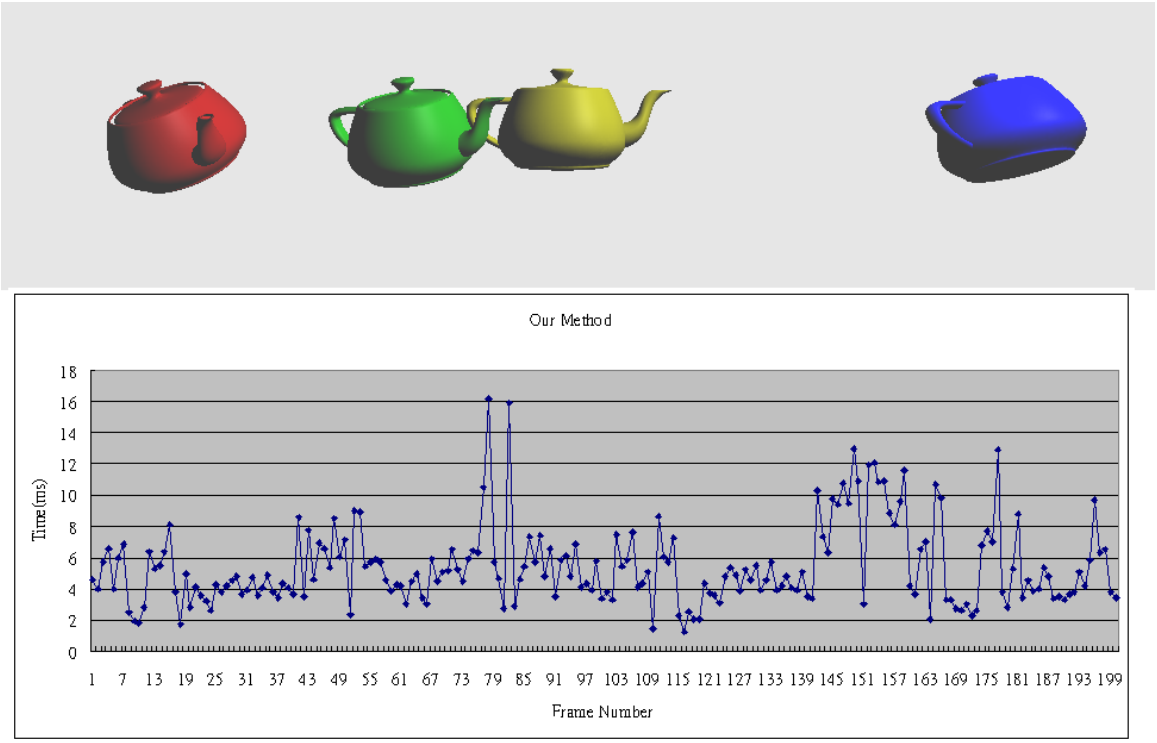


Figure 4.7: Teapot vs Teapot. The test results of our method.

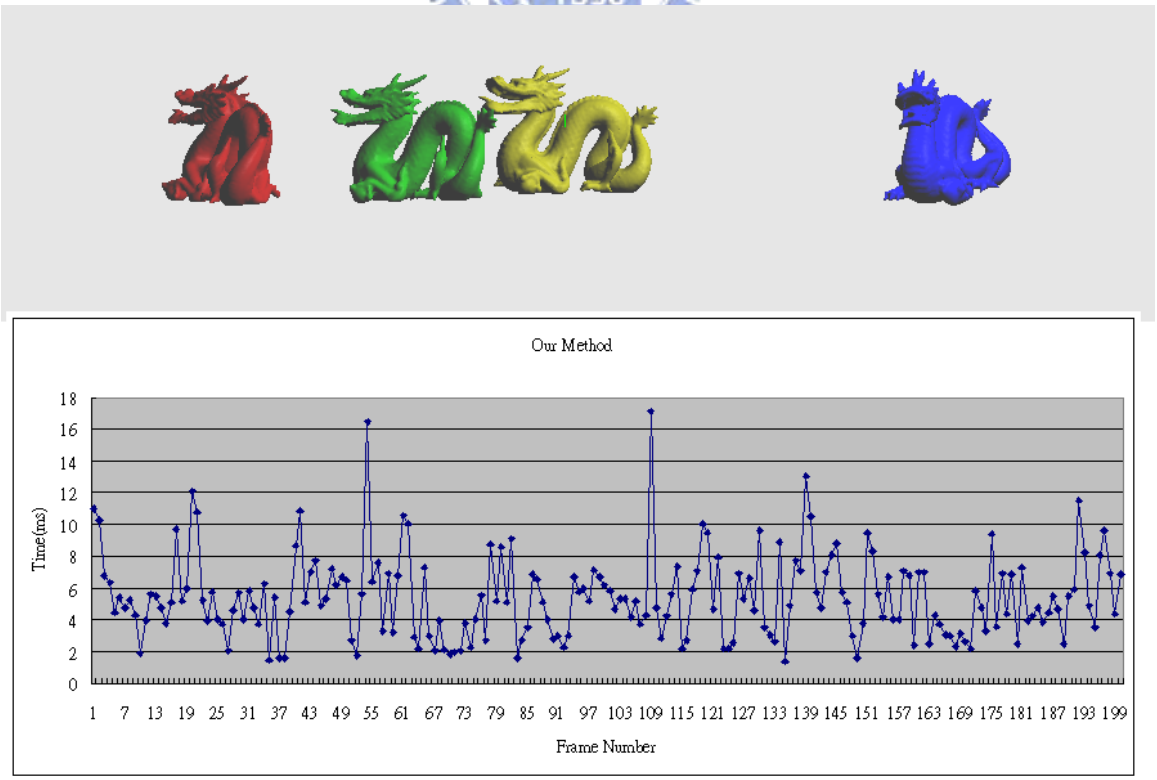


Figure 4.8: Dragon vs Dragon. The test results of our method.

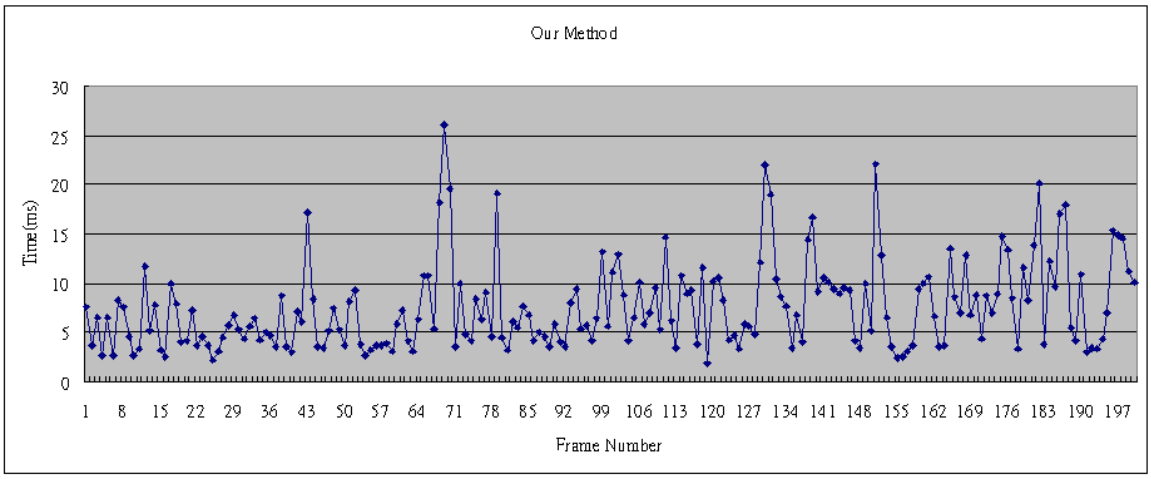


Figure 4.9: Car vs Car. The test results of our method.

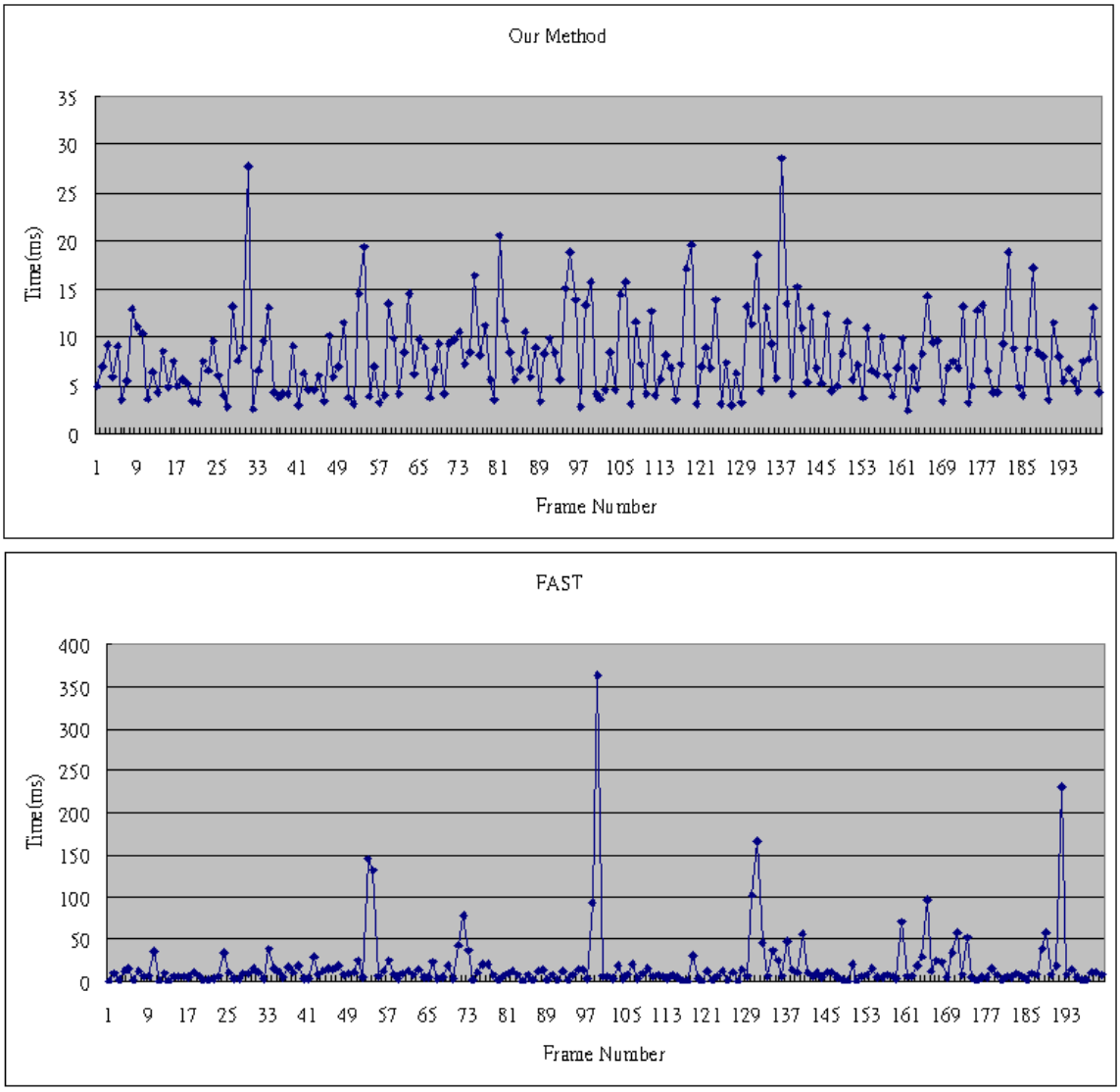


Figure 4.10: The performance result of dynamic torusknot.

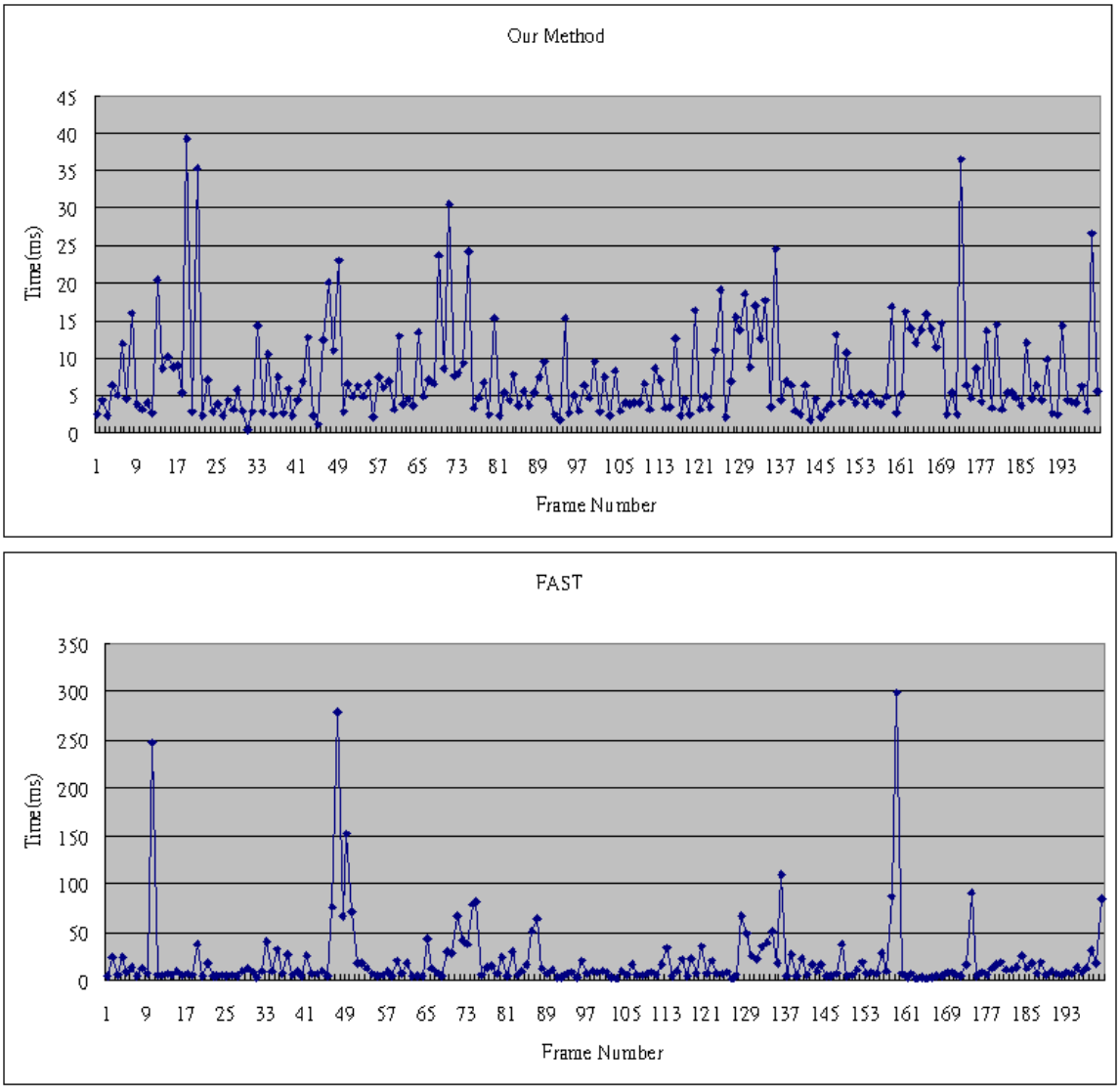


Figure 4.11: The performance result of dynamic bunny.

CHAPTER 5

Conclusion

In this Chapter, we give a summary of the thesis and propose the directions for further research of our method.

5.1 Summary

In this thesis, we present a highly interactive continuous collision detection algorithm for complex models. Our method is based on conservative advancement approach[Mir96] and an efficient distance query method[LGLM]. Given two models and their initial and final configurations, we use linear interpolation to calculate their in-between motion and examine if there exists collision between the two objects in a time interval. Our algorithm is composed of three stages, and the three stages form a pipeline. For each stage, we use different resolutions of a model to detect the collision in a time interval. In our system, user can assign a distance threshold. If the distance between two objects is below the threshold, our system claims that the two objects collide with each other. The contributions of our method are:

1. Efficient for highly nonconvex models: As shown in chapter 4, our method is more efficient than FAST[ZLK06] for highly nonconvex models.

2. Memory efficient: Due to the data structure which we adopt, our method uses less memory space and could achieve good performance.

3. No limitation on the topology of triangle mesh: In contrast to FAST, our method could be applied on polygon soups, not only 2-manifold models.

5.2 Future Work

Our method is to solve the problem of continuous collision detection for a pair of rigid bodies. Therefore, in order to apply our method to various kinds of dynamic simulations, there still needs some efforts to enhance our method. First, we would like to enhance our method to solve the problem of continuous collision detection for articulated bodies. Although Each link of an articulated body is a rigid body, it needs an broad-phased collision pruning algorithm to avoid the collision checks of these pairs of links which could not collide. In the field of discrete collision detection, the problem has been extensively studied. However, there still exists some issues for us to study in the field of continuous collision detection. Second, we would like to extend our method to multi-body collision detection. This problem has been also extensively studied In the field of discrete collision detection, but it is a more complex problem in the field of continuous collision detection. Because an object could collide with several objects during a time interval, the simulation could be incorrect if we only detect the first collision of the object. There are some other issues for this enhancement to study. Finally, we would like to study whether our method could be extended to deformable body. To efficiently solve this problem, we have to update our BVH in an efficient way. To produce the inbetween motion of a deformable body is also another important issue.

Bibliography

[AMBJ02] K. Abdel-Malek, D. Blackmore, and K. Joy. Swept Volumes: Foundations, Perspectives and Applications. *International Journal of Shape Modeling*, 2002.

[BGH97] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[CLMP95] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments. In *In Proceedings of Symposium on Interactive 3D Graphics*, pages 189–218, 1995.

[CS06] D.S. Coming and O.G. Staadt. Kinetic sweep and prune for multi-body continuous motions. *Computer and Graphics*, 30(3), 2006.

[Ea00] S. Ehmann and M. Lin and. Accelerated proximity queries between convex polyhedra using multi-level voronoi marching. *Tech. Report TR00-026, Department of Computer Science Univ. of North Carolina,*, 2000.

[Ea01] S. Ehmann and M. Lin and. Accurate and Fast Proximity Queries between Polyhedra using Convex Surface Decomposition. *Computer Graphics Forum*, 20(3):500–510, 2001.

[GASF94] A. Garcia-Alonso, N. Serrano, and J. Flaquer. Solving the Collision Detection Problem. *IEEE Computer Graphics and Applications*, 13(3):36–43, July 1994.

[GLM96] S. Gottschalk, M. Lin, and D. Manocha. OBBTree: A Hierarchical Structure

for Rapid Interference Detection. In *Proc. of ACM Siggraph'96*, pages 171–180, 1996.

[HDLM96] M. Hughes, C. DiMattia, M.C. Lin, and D. Manocha. Efficient and Accurate Interference Detection for Polynomial Deformation and Soft Object Animation. Technical Report TR96-001, 2 1996.

[He99] Taosong He. Fast Collision Detection using QuOSPO trees. In *Symposium on Interactive 3D Graphics*, pages 55–62, 1999.

[HKM95] M. Held, J. Klosowski, and J. Mitchell. Evaluation of Collision Detection Methods for Virtual Reality Fly-throughs. In *In proceedings Seventh Canadian Conference on Computational Geometry*, 1995.

[Hub93] P.M. Hubbard. Interactive collision detection. In *In Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, number TR96-001, 2 1993.

[Hub95] P.M. Hubbard. Collision Detection for Interactive Graphics Applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.

[Hub96] P.M. Hubbard. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[JTT01] P. Jimnez, F. Thomas, and C. Torras. 3D Collision Detection: A Survey. *Computers and Graphics*, 25(2):269–285, apr 2001.

[KHM+98] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[KPLM98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical Shells: A Higher-Order Bounding Volume for Fast Proximity Queries. In *In Proceedings of WAFR 98*, pages 287–296, 2 1998.

[LG98] M.C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *In Proc. of IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

[LGLM] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast Proximity Queries with Swept Sphere Volumes. In *Tech. Rep. TR99-018*. Department of Computer Science, University of North Carolina.

[LM03] M.C. Lin and D. Manocha. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*, 2003.

[Mir96] B. Mirtich. *Impulse-Based Dynamic Simulation of Rigid Body Systems*. Ph.D. thesis, University of California, Berkeley, 1996.

[MW88] M. Moore and J.P. Wilhelms. Collision Detection and Response for Computer Animation. In *Computer Graphics (SIGGRAPH 88)*, pages 289–298, 1988.

[NAT90] B. Naylor, J. Amanatides, and W. Thibault. Merging BSP Trees Yields Polyhedral Set Operations. In *Computer Graphics (SIGGRAPH 90)*, pages 115–124, 1990.

[PG95] I. Palmer and R. Grimsdale. Collision Detection for Animation using Sphere-Trees. *Computer Graphics Forum*, 14(2):105–116, 1995.

[Qui94] S. Quinlan. Efficient Distance Computation between Non-Convex Objects. In *IEEE Intern. Conf. on Robotics and Automation*, pages 3324–3329. IEEE, 1994.

[RKC02] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 2002.

[RKLM04a] S. Redon, Y.J. Kim, M.C. Lin, and D. Manocha. Fast continuous collision detection for articulated models. In *Proceedings of ACM Symposium on Solid Modeling and Applications*, 2004.

[RKLM04b] S. Redon, Y.J. Kim, M.C. Lin, and D. Manocha. Interactive and continuous collision detection for avatars in virtual environments. In *Proceedings of IEEE Virtual Reality*, 2004.

[Sny92] J. Snyder. Interval analysis for computer graphics. *Computer Graphics*, 26(2):121–130, July 1992.

[TN87] W.C. Thibault and B.F. Naylor. Set Operations on Polyhedra Using Binary Space Partitioning Trees. In *Computer Graphics (SIGGRAPH 87)*, pages 153–162, 1987.

[WZ06] R. Weller and G. Zachmann. Kinetic separation lists for continuous collision detection of deformable objects. In *Third Workshop in Virtual Reality Interactions and Physical Simulation*, 2006.

[YT93] Y. Yang and N. Thalmann. An Improved Algorithm for Collision Detection in Cloth Animation with Human Body. In *Proc. First Pacific Conf. Computer Graphics and Applications*, pages 237–251, 1993.

[Zac95] G. Zachmann. The BoxTree: Enabling Real-Time and Exact Collision Detection of Arbitrary Polyhedra. In *In Informal Proc. First Workshop on Simulation and Interaction in Virtual Environments, SIVE 95*, pages 104–112, July 1995.

[ZLK06] X. Zhang, M. Lee, and Y. J. Kim. Interactive Continuous Collision Detection for Non-Convex Polyhedra. In *Visual Computer Vol 22*, pages 9–11, 2006.