

國立交通大學

資訊科學與工程研究所

碩士論文

複雜環境之影像式碰撞濾除運算

A Practical Collision Filtering for Complex Environments
Using Image-based Techniques

研究生：陳勇誠

指導教授：莊榮宏 教授

中華民國九十五年九月

複雜環境之影像式碰撞濾除運算
A Practical Collision Filtering for Complex Environments Using
Image-based Techniques

研究生：陳勇誠

Student：Yong-Cheng Chen

指導教授：莊榮宏

Advisor：Jung-Hong Chuang

國立交通大學
資訊科學與工程研究所
碩士論文



Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年九月

複雜環境之影像式碰撞濾除運算

學生： 陳勇誠

指導教授： 莊榮宏 博士

國立交通大學資訊工程學系

摘要

我們設計一個可快速且有效率的對複雜場景進行碰撞偵測的演算法，我們將針對影像式的碰撞濾除加以改進並將潛在碰撞集合(potentially colliding set, PCS)分割成數個子集合。在方法中我們延續 CULLIDE 的演算法[13]，在線性時間將那些屬於完全可見的物體從潛在碰撞集合內移除，並且我們在兩次的繪圖流程都給定一個從遠到近的順序，這將大幅度的降低因物體投影重疊而影響方法的效率；儘管有些無發生碰撞的物體被其他的物體給遮蓋著，在我們方法中大多情況下仍能有效濾除。另外，在執行能見度分析也同時計算是否存在分割曲面於潛在碰撞集合中，分割曲面可將潛在碰撞集合分割成數個小的集合，可減少配對碰撞偵測的次數。最後，我們利用 50~200 個由 800 個三角形構成的圓環測試執行效能，平均所花費的碰撞偵測時間約為 0.85~7 秒，實驗結果，證實我們的方法能有效率地應用於物理模擬或即時系統之中

A Practical Collision Filtering for Complex Environments using Image-based Techniques

Student: Yong-Cheng Chen

Advisor: Dr. Jung-Hong Chuang

Department of Computer Science and Information Engineering
National Chiao Tung University

ABSTRACT

We present an algorithm for efficient and exact collision detection between complex geometric models in large environments using graphics hardware. The purpose of our algorithm is to improve image-based collision culling and partition the potentially colliding set (PCS) into subsets. Our algorithm eliminates objects which are fully visible in PCS using occlusion query in linear time. It is based on CULLIDE and decides an order of rendering to supply a more efficient culling performance. We can conspicuously reduce the extent of culling to rely on the depth complexity of the scene along the view direction. Despite there are some collision-free objects are occluded by other objects, they may be pruned in our algorithm. Furthermore, our algorithm determines whether separating surfaces exist between the subsets of the PCS. The computation of collision pairs would be beneficial to divide the PCS into multiple sets during the collision culling. Finally, we measured the performance of our algorithm using 50-200 torii with 800 triangles. The average collision detection time is around 0.85-7 milliseconds. In the experiments, our approach had shown to be efficient for collision detection in physics simulation and applied to real-time applications.

Contents

List of Figures	iv
List of Algorithms	vii
List of Tables	viii
1 Introduction	1
1.1 Collision Detection	1
1.2 Collision Detection Algorithms	2
1.3 Motivation	4
1.4 Organization	5
2 Related Work	6
2.1 Ray Casting	6
2.2 Counting Boundary Crossings	8
2.3 Layered Depth Images	11
2.4 Collision Culling	14
3 Image-based Collision Filtering	16
3.1 Overview	16
3.2 Collision filtering with depth order	18
3.3 Collision Independent Sets Construction Using Separating Surfaces	20
3.3.1 First Pass in Collision Culling	27
3.3.2 Second Pass in Collision Culling	28
3.3.3 A detailed process of collision independent sets construction	28
3.4 Collision filtering in hierarchical structure	29

3.4.1	Bounding Volume Hierarchy Construction	29
3.4.1.1	Bounding Volume Hierarchy Optimization	36
3.4.2	Sorting and collision independent sets construction	37
4	Results	39
4.1	Performance analysis in simple environment	40
4.2	Performance comparison for environments of high geometry complexity and low depth complexity	45
4.3	Performance comparison for environments of high depth complexity and low geometry complexity	48
5	Conclusion	57
5.1	Summary	57
5.2	Future work	58
Bibliography		59



List of Figures

2.1	Passible Z-interval cases[BWS99]	7
2.2	Passible Z-interval cases[BWS99]	9
2.3	Semi-infinite rays from a point in a closed mesh[KP03]	9
2.4	Update the depth buffer[KP03]	10
2.5	Increases the stencil buffer when rendering "front-facing" polygons[KP03].	10
2.6	Decreases the stencil buffer when rendering "back-facing" polygons[KP03].	11
2.7	(a) VoI for A and B; (b) Extended VoI to obtain an outside face for A[HTG03].	12
2.8	LDI for two pixels, unsorted and sorted with respect to depth values[HTG03].	13
2.9	Intersection volume for Knot and Dragon[HTG03]	14
3.1	The flowchart of the proposed collision detection system.	17
3.2	The worst case of CULLIDE: (a) It shows that there are no intersections in the environment; (b) All the projections of objects overlap with each other; (C) We only can prune the object C using visibility-based pruning in CULLIDE.	19
3.3	Projection and sorting.	20
3.4	The result of our collision culling: (a) It shows that there are no intersec- tions in the environment; (b) All the projections of objects overlap with each other; (c),(d) There are no occlusion in two passes.	22
3.5	If S_1 and S_2 do not collide with each other, we can find a separating surface among them.	23
3.6	(a) Collision independent set; (b) The projection of green objects overlap with the collision independent set, and the projection of yellow objects dose not overlap with the set.	24

3.7	An example for collision independent sets construction.	26
3.8	An example, a scene consists of $O_1, O_2, O_3, O_4, O_5, O_6, O_7$, and O_8	30
3.9	Traverse projection elements of e_5, e_9 , and e_3 . The global probing surface PS_g is updated to b_3	30
3.10	Traverse projection elements of b_5 and b_8 , collision independent set CIS_1 is constructed at O_8	31
3.11	Traverse projection elements of e_4 and b_3 , and the expansion of CIS_1 is finish at b_3	31
3.12	Traverse projection elements of e_2 and b_4 , collision independent set CIS_2 is constructed at O_4	32
3.13	Traverse projection elements of e_1, e_6 and b_2 , and the expansion of CIS_2 is finish at b_2	32
3.14	Traverse projection elements of e_7 and b_6 , collision independent set CIS_3 is constructed at O_6	33
3.15	After the first pass, three independent colliding sets, $PCS_1 = \{O_8\}$, $PCS_2 = \{O_4\}$, and $PCS_3 = \{O_6, O_7\}$, and three separating surfaces, SS_1, SS_2, SS_3 , are found.	33
3.16	After the second pass, the three collision independent sets will be finalized as $CIS_1 = \{O_5, O_8\}$, $CIS_2 = \{O_3, O_4\}$, and $CIS_3 = \{O_2, O_6, O_7\}$, and there are two separating surfaces found.	34
3.17	Bounding volume hierarchy with a 8-ary tree.	36
3.18	Bounding volume hierarchy construction using (a) three eigenvectors of the covariance matrix; (b) optimizing principal components	37
4.1	Figures (a)-(f) show the simulating process in environment 1.	41
4.2	Performance comparison between our approach and CULLIDE in environment 1.	42
4.3	Performance comparison with CULLIDE, CULLIDE with hierarcical bounding volume, and our approach in environment 1 at frame buffer resolution 800×800	43
4.4	The number of collision independent sets constructed for environment 1 at frame buffer resolution 800×800	44
4.5	Figures (a)-(f) show the simulating process in environment 2.	46

4.6	Performance comparison between our approach and CULLIDE in environment 2.	47
4.7	Performance comparison with CULLIDE, CULLIDE with hierarcical bounding volume, and our approach in environment 2 at frame buffer resolution 800×800	49
4.8	Performance comparison between CULLIDE with hierarcical bounding volume and our approach in environment 2 at frame buffer resolution 800×800	50
4.9	The number of collision independent constructed for in environment 2 at frame buffer resolution 800×800	51
4.10	Figure shows the simulating process in environment 3.	52
4.11	Performance comparison between our approach and CULLIDE in environment 3.	53
4.12	Performance comparison with CULLIDE, CULLIDE with hierarcical bounding volume, and our approach in environment 3 for 200 torii at frame buffer resolution 800×800	55
4.13	The number of collision independent sets constructed for environment 3 for 200 torii at frame buffer resolution 800×800	56



List of Algorithms

2.1	The logical flow of RECODE	8
2.2	The algorithm of collision culling	15
3.1	Our collision culling algorithm	21
3.2	The algorithm of collision independent sets construction	35



List of Tables

3.1	Parameters of collision independent sets construction	27
4.1	Timing statistics for environment 1.	45
4.2	Timing statistics for environment 2.	48
4.3	Timing statistics for environment 3.	52



Acknowledgments

I would like to thank my advisors, Professor Jung-Hong Chuang, for his guidance, inspirations, and encouragement. I am grateful to Tan-Chi Ho for his comments and advices. Thanks to my colleagues in CGGM lab.: Yi-Chun Lin, Chao-Wei Juan, Roger Hong, Chen-Li Hao, Yu-Shuo Lio, Chia-Lin Ko, Yung-Cheng Chen, Zhi-Wen Zhang, and Ya-Jing Qiu. for their assistances and discussions. Lastly, I would like to thank my parents and my girl friend Si-Ning Li for their love, encouragement, and support.



CHAPTER 1

Introduction

In recent years, with the growth of the developments in games, virtual reality, physics simulation, surgery simulators, and robotics, the realistic and efficient physical simulation appears more and more important. Collision detection aiming to determine when and where objects colliding each other, plays an important role on the physical simulation system. Collision detection in general is a computationally expensive problem due to its $O(n^2)$ time complexity.

1.1 Collision Detection

The interference among objects is usually detected by using the special data structure and geometric computation, and is often categorized two phases:

- **Broad phase:** For broad phase, we regard an object or a group of objects as a base unit. We determine whether base unit collide with each other. The ultimate goal is to avoid testing $O(n^2)$ times for all potential pairs, where n is the number of base units. The outcome of the broad phase is a set of paired objects or base units that potentially collide each other.
- **Narrow phase:** For narrow phase, we perform collision detection between the primitives of objects, such as polygons. The goal is to greatly reduce the number of test for potential paired primitives. In narrow phase, in addition to accurately computing

the collisions among objects, one should capture other information, such as collision position, collision time, and penetrating depth etc.

Collision detection is a fundamental problem in interactive applications. Real-time performance is the ultimate goal for all collision detection algorithm. To this end, several coherence properties are generally applied, which can be classified into three categories as follow:

- **Spatial coherence:** The problem of collision detection is how to find the geometric overlapping information in 3D space. The spatial coherence depicts the relations among objects in the space. We can assume that an object usually spans only a relatively small portion of the space, and its collisions to the other are fairly rare, only when there are close to each other.
- **Temporal coherence:** In the dynamic environments, the location of objects will be changed with time. Under the assumption that the motion of objects is small among continues timestamps, if an interference occurs at a timestamp then it may occur at next timestamp.
- **Image coherence:** The problem of collision detection can be transferred from 3D to 2D image. The dimensional reducing will decrease the complexity of geometric computation. The intersection could be detected by using graphics hardware. For example, if an object is fully visible with respect to others then it does not collide.

1.2 Collision Detection Algorithms

The problem of collision detection has been extensively studied in recent years. We refer the readers to these recent surveys [JTT01, LG98]. The collision detection algorithms can be classified into two categories: object-space collision detection and image-based collision detection.

Most of the object-space approaches are proposed to accelerate collision detection by using hierarchical data structure and spatial partition. For narrow phase, hierarchical bounding volume is commonly used to speed up the interferences detection of paired objects. The representation of hierarchical bounding volume is a tree structure formed by some simple shapes, such as spheres, axis-aligned bounding boxes, oriented bounding

boxes, discrete orientation polytopes, quantized orientation slabs with primary orientations, spherical shells, etc. [GLM96, He99, Hub93, Hub95, Hub96, HDLM96, JTT01, KHM⁺98, KPLM98, PG95, Qui94, Zac95]. That hierarchical representations are frequently used for collision detection and based on data structures that can be more or less pre-computed. These representations are used to cull away portions of each object that are not in close proximity. However, the hierarchical bounding volumes are hardly used for deformable objects, because they must be rebuilt or updated when the shape of objects changes. Some extensions of hierarchical representation are proposed for handling deformable objects [JP04, KZ05, LAM01, van97], focusing on how to update hierarchy and bounding volumes efficiently.

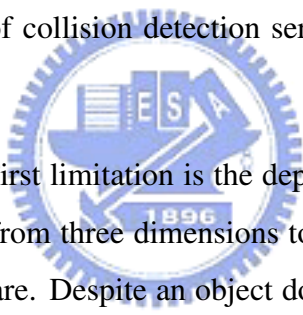
Spatial partition is usually used in either narrow phase or broad phase. Common spatial decomposition techniques are BSP trees, octrees, k-d trees, or voxel grid [CLMP95, GASF94, HKM95, MW88, NAT90, TN87, YT93].

Recently, the image-based technique supplies new avenues for collision detection. The conceptions of ray tracing are exploited for detecting the interference of two convex objects [BWS99, MOK95]. The relation among a ray with two convex objects is categorized into several situations. The interferences are found by checking whether the intervals overlap along the ray. However, these algorithms are limited by the depth complexity and only applicable to simple shape or convex objects. An algorithm is proposed for handling multi-body environments by exploiting the shadow volume technique [KP03]. This algorithm is very efficient for deformable and non-convex objects. Layered depth image technique is introduced for efficient collision detection of arbitrarily shaped, water-tight objects [HTG03, HTG04]. Layered depth images are used to approximate the volume of an objects. In this way, the volumetric intersection of objects will be detected. A reliable image-based collision culling algorithm is presented in [GRLM03, GLM04]. It computes a potentially colliding set using hardware occlusion queries. The techniques are based on CULLIDE for handling self-intersection [BM04, GLM05b, GLM05a]. The image-based collision detection is also applied to some specific applications, including cloth animation and virtual surgery [GLM05b, GLM05a, LCN99, VSC01]. In chapter 2, we will give a more detailed introduction to image-based techniques for collision detection.

1.3 Motivation

With the growth of the graphics hardware, many image-based techniques for collision detection have been proposed, and have attempted to maximally utilize the functionality of graphics hardware. Hardware-based techniques usually do not require the complex data structure and reduce the load of CPU by using graphics hardware. These techniques rely on rasterizing the objects of a collision query into color, depth, or stencil buffer, and are approximate collision detection method. However, frame buffer readbacks will become the bottleneck of these approaches.

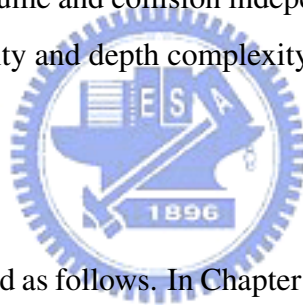
An image-based approach, called CULLIDE, does not perform frame buffer readbacks but readbacks a few bytes per object. It computes the potentially colliding set by using graphics hardware, and intersections will be determined on CPU. Since CULLIDE prunes some objects which do not collide using visibility analysis, it will produce many factors which affect the performance of collision detection seriously. There are four limitations described as follows:

- 
- (1) *Depth complexity*: The first limitation is the depth complexity of the scene. CULLIDE reduces the space from three dimensions to two dimensions by using rasterization of graphics hardware. Despite an object does not collide with any one, but it may be occluded by others. When the number of overlapping between orthographic projections of objects along view direction is frequent, it is able to decrease the performance of collision culling.
 - (2) *The coarseness of single PCS*: The second limitation in CULLIDE is that the PCS is too coarse. CULLIDE proposed a method to efficiently return the PCS. However, it is a pity that it can not partition PCS into subsets during collision culling. The computation of collision pairs will be simpler if a colliding set is divided into several subsets.
 - (3) *Non-volumetric intersections*: The third limitation is that it is not able to detect volumetric intersections, penetrating depths and distances between objects. It is only to determine whether overlapping triangles exist between different objects.
 - (4) *Precision issue*: The last limitation is that the accuracy of collision detection is extent of the frame buffer resolution and the depth buffer precision. In fact, collisions will

be missed or too conservative during rasterization due to the frame buffer is not used appropriately.

Therefore, the final purpose of this paper is to accelerate collision detection for real-time system. Our algorithm uses two-pass structure in CULLIDE and efficiently detects collision among rigid or deformable objects. We will discuss the first two limitations described above and propose some methods to improve CULLIDE, and expect to improve the culling efficiency by using sorted order. It can decrease the impact on collision culling from depth complexity of a scene along view direction. In our approach, we can efficiently prune an object which does not collide with others and is occluded or not. We also can partition the PCS into multiple collision independent sets in linear time such that the cost of exact collision computations will be reduced. Finally, we construct bounding volume hierarchy for each object, and the proposed algorithm will be applied to hierarchical structure. Hierarchical bounding volume and collision independent sets construction can reduce influence of geometry complexity and depth complexity, respectively.

1.4 Organization



The rest of the thesis is organized as follows. In Chapter 2, we give a survey of image-based collision detection techniques, including ray casting, counting boundary crossing, layered depth images, and collision culling. In Chapter 3, we give an overview of our framework and present our approach to determine multiple collision independent sets by using separating surfaces. In Chapter 4, we highlight the performance on different benchmarks, and make a comparison with CULLIDE and Quick-CULLIDE by physical simulation. Finally, in Chapter 5, we give the conclusion.

Related Work

With the growth of the graphics hardware, many statistics show that the computational power of the graphics processing units (GPUs) exceed that of the central processing units (CPUs) in some specific applications. Although there are a lot of restrictions on the manipulation of GPUs, and the frequencies of GPUs are usually lower than the frequencies of CPUs of the same period; However, GPUs have the parallel computation ability and support programable vertex and pixel processors. Such developments have greatly contributed to the techniques of rendering. Besides, there are many application, which are transferred to GPU. Therefore, the researches and discussions on GPU are more and more important in recent years. In this chapter, we will give a review of prior work on image-based techniques for collision detection.

2.1 Ray Casting

The basis of the ray casting is based on the conception of the ray tracing. It determines the relations among a ray with two objects, and categorizes the intersection of them into several situations. However, the graphics hardware dose not use the ray tracing to render the image but the ray casting. Therefore, in this section, we will describe how to achieve the intersection of the ray with two objects using the ray casting technique.

The intersection of a ray with two convex objects could be categorized into eight situations [BWS99, MOK95], as shown in Figure 2.1. Case 0 means that the ray dose not

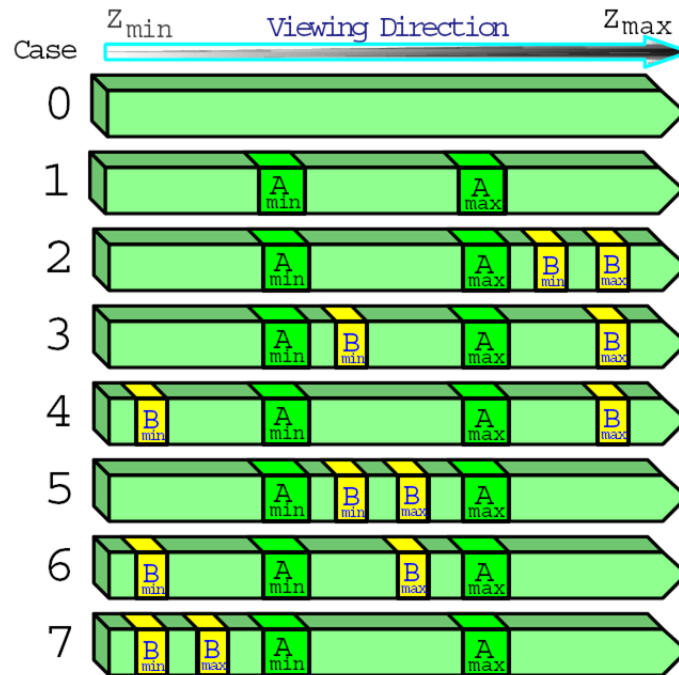


Figure 2.1: Possible Z-interval cases[BWS99]

overlap with any object. In case 1, the ray passes through A, but no face of B overlaps the ray. In both case 2 and 7, the range on the depth covered by A and B, but $[B_{min}, B_{max}]$ does not overlap the interval $[A_{min}, A_{max}]$. Hence, in cases 0, 1, 2, and 7, we can decide that A and B do not overlap.

In RECODE, it reduce the collision detection problem to a one-dimensional interval test along the z-axis. As show in Figure 2.1, first it determines MOR(A,B), which is the minimum overlapping region of the projection of the volume occupied by the overlapping bounding boxes of objects A and B. Then it sets the rendering viewport, which is the frame buffer viewport allocated for rendering object A, to correspond only to the region covered by the MOR(A,B). The maximum Z-values of object A are stored in the depth buffer by the function $RenderSetZ(A, \geq)$. This map is then used as reference in the first rendering pass in order to establish a class of the possible cases of B overlapping A along the Z-axis.

Then it tests against eight independent cases, as illustrated in Figure 2.2. First the stencil buffer is initialized to zero. If there are no objects at pixel (x, y) then the corresponding stencil value remains zero. If at least one object overlaps pixel (x, y) then the corresponding stencil value is incremented to one. In cases 0, 1, 2, and 7, there is no intersection between A and B that described above. In cases 3 and 4 the interval occupied by B overlaps $[Z_{min}, A_{max}]$. That is, only one face of B crosses over A_{max} . Hence, the stencil value at (x, y) is

Algorithm 2.1: The logical flow of RECODE

```

RECORD  $A, B \equiv RenderingArea \leftarrow MOR(A, B)$ ;
RenderSetZ( $A, \geq$ );
RenderTestZ( $B, \leq$ );
if  $zIntervalOverlap(A, B) = TRUE$  then
    return collision = TRUE;
end
if  $secondRendering = TRUE$  then
    RenderSetZ( $B, \geq$ );
    RenderTestZ( $A, \leq$ );
    if  $zIntervalOverlap(A, B) = TRUE$  then
        return collision = TRUE;
    end
end

```

incremented to two. If stencil values equal to two, then the interferences of A and B are detected in the first pass. However, cases 5 and 6 will be missed because the stencil values equal to three. In order to detect cases 5 and 6, it stores the maximum Z-values of B in the depth buffer first and then render object A in the second pass. The cases 5 and 6 will be captured by checking whether the stencil values equals to two in the second pass.

RECORD presented an image-based technique for collision detection. It determines whether the interferences between two convex object occur by reading the stencil buffer. However, it can not compute some collision responds, such as the positions and the normals. It only can handle convex objects and does not attend to self-intersections. The stencil buffer readbacks are the main bottlenecks in RECORD.

2.2 Counting Boundary Crossings

An approach exploit the shadow volume technique, a polygonal mesh is created that represents the volume of space that lies in the shadow cast by an object. Determining whether or not a point lies in shadow involves casting a ray from the viewer toward the point. This approach is called CInDeR [KP03]. Its collision detection algorithm is predicated on the following property: Two polyhedral objects are interfering with each other if and only if an

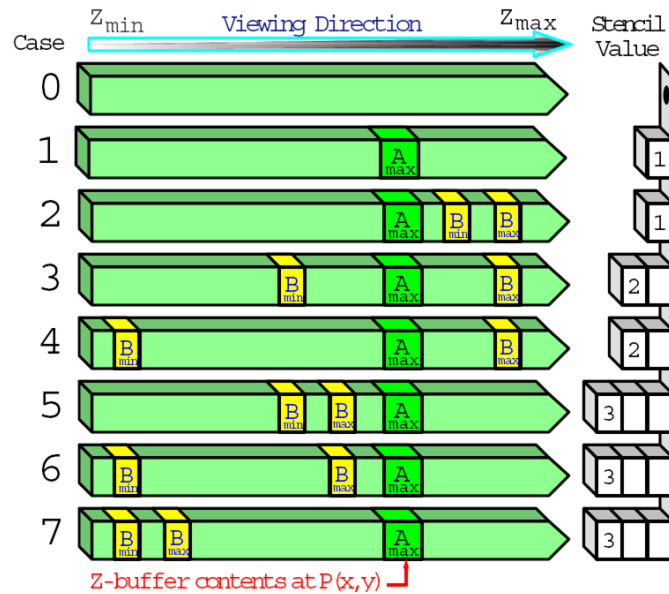


Figure 2.2: Possible Z-interval cases[BWS99]

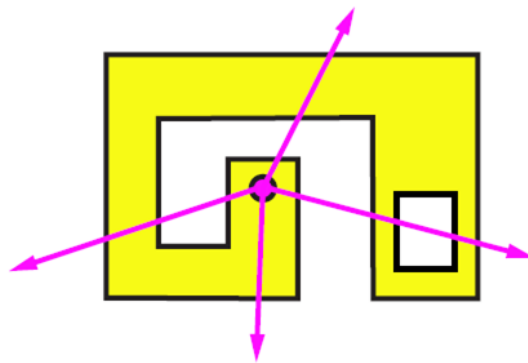


Figure 2.3: Semi-infinite rays from a point in a closed mesh[KP03]

edge of one object intersects the volume occupied by the other.

Counting boundary crossings technique is to observe a point relative to a closed mesh by casting a semi-infinite ray from the point. The number of the object's boundary that the ray passes through are counted. It is commonly used to solve the point-in-polygon problem. For a semi-infinite ray from the point, it can specify whether or not an intersection of the ray with a solid corresponds to the ray entering or leaving the solids volume. A semi-infinite ray cast from the interior of a solid will leave the solid one more time than it enters the solid, as shown in Figure 2.3.

Collision detection is performed by sampling the boundaries of objects and looking for object edges that are interior to other volumes. This is done using the rasterization of graphics hardware. Rays are cast through the pixels of the viewport toward objects of

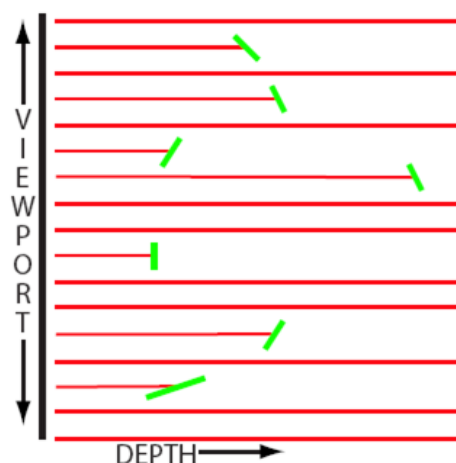


Figure 2.4: Update the depth buffer[KP03]

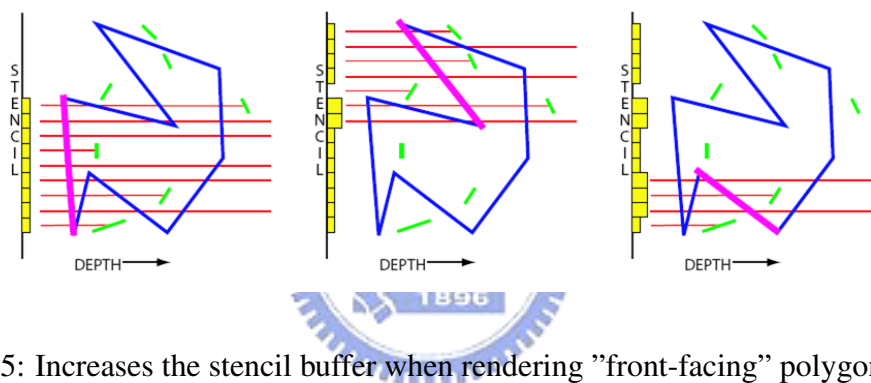


Figure 2.5: Increases the stencil buffer when rendering "front-facing" polygons[KP03].

interest. Rays that strike those objects edges are of particular interest.

First, it clears the depth buffer and sets the stencil buffer to zero. In the first rendering pass, it renders all of the edges of objects, and updates the depth buffer with their depth values, as shown in Figure 2.4. This ensures that all rays cast through pixels will be targeted at polygon edges.

In the second rendering pass, it draws only those polygons whose normals face toward the ray's origin, as shown in Figure 2.5. That is, all polygons will be rejected for which the dot product of the normal with the ray direction is positive. In the graphics hardware this corresponds to a back-face cull. If the pixel passes the depth test then increases the stencil value.

In the third rendering pass, it draws only those polygons whose normals face away the ray's origin, as shown in Figure 2.6. That is, all polygons will be rejected for which the dot product of the normal with the ray direction is negative. In the graphics hardware this

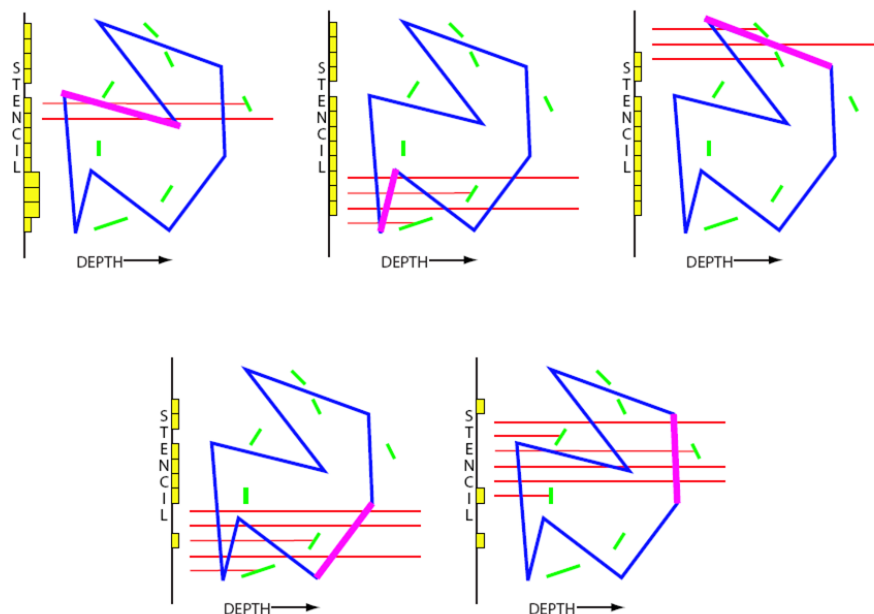


Figure 2.6: Decreases the stencil buffer when rendering "back-facing" polygons[KP03].

corresponds to a front-face cull. If the pixel passes the depth test then decreases the stencil value. After the third pass, the stencil buffer value at each pixel gives the results of the collision detection. Finally, it check interferences by scanning the values in the stencil buffer.

This approach is linear in both the number of objects and the number of polygons comprising those objects. It performs broad phase collision detection and narrow phase collision detection at the same time. The additional collision information, such as collision positions and identifications of objects, can be obtained using the depth buffer and the color buffer. However, It is seriously restricted by viewport resolution and is not robust in case of occluded edges.

2.3 Layered Depth Images

A Layered Depth Image (LDI) are used for the computation of volumetric intersections of complex polygonal meshes [HTG03]. LDIs have been introduced as an efficient image-based rendering technique. The depth values of a pixel are stored in multiple depth images. Such approach can conserve the occluded surfaces of the complex object.

First, it computes the axis-aligned bounding box (AABB) for a pair of objects. If the bounding boxes of the two objects do not collide then there is no intersection between

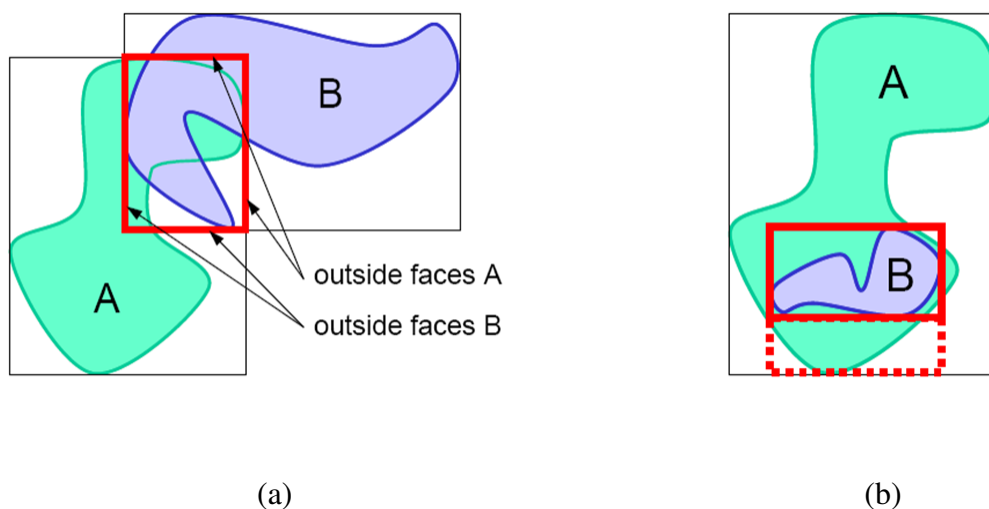


Figure 2.7: (a) VoI for A and B; (b) Extended VoI to obtain an outside face for A[HTG03].

them. If it is not, the LDIs computation is applied to the AABB intersection volume, called Volume-of-Interest (VoI). The VoI is bounded by pairs of their faces. Figure 2.7(a) shows a VoI with corresponding outside faces in two dimensions. In some cases, for instance when one box is entirely within another box, appropriate outside faces for both objects cannot be found. This problem is solved by extending the VoI. If the outside face of one object is fixed, the opposite face of the VoI can be scaled to touch the bounding box of the other object, as shown in Figure 2.7(b).

In order to generate an LDI, the object is rendered multiple times. The viewing parameters for the rendering process are determined by the selected outside faces of the VoI defining the near planes, and their opposite faces defining the far planes. Orthographic projection is used for rendering. Objects are rendered n_{max} times for LDI generation, where n_{max} denotes the depth complexity of the relevant part of the object within the VoI.

The first rendering pass generates a single LDI and computes n_{max} . First, depth testing and face culling are disabled. Only the stencil test is employed to discard fragments. The stencil test configuration allows the first fragment per pixel to pass, while the corresponding stencil value is incremented. Subsequent fragments are discarded by the stencil test, but still increment the stencil buffer. Hence, after the first rendering pass the depth buffer contains the first object layer per pixel and the stencil buffer contains a map representing the depth complexity per pixel. Thus, n_{max} is found by searching the maximum stencil value of the stencil buffer. If $n_{max} > 1$, additional rendering passes 2 to n_{max} generate the

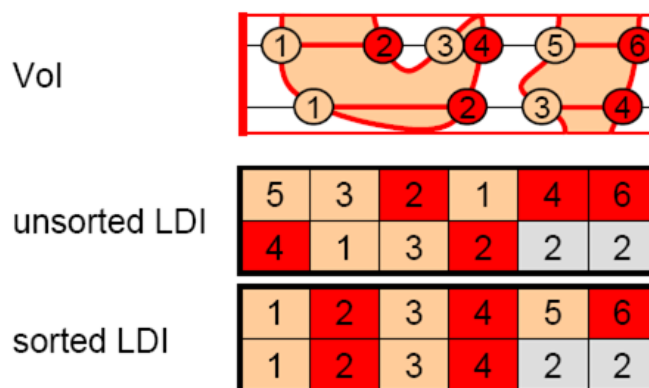


Figure 2.8: LDI for two pixels, unsorted and sorted with respect to depth values[HTG03].

remaining layers. The rendering setup is similar to the first pass. However, during the n -th rendering pass, the first n fragments per pixel pass the stencil test and, as a consequence, the resulting depth buffer contains the n -th LDI layer. During these passes, the stencil buffer is not incremented, if the stencil test fails.

It generates unsorted LDIs due to fragments are rendered in an arbitrary order. Therefore, the LDIs are sorted per pixel for further processing. Only the first n_p layers per pixel are considered, where n_p is the depth complexity of this pixel. n_p is taken from the stencil buffer as computed in the first rendering pass. If n_p is smaller than n_{max} , layers n_{p+1} to n_{max} do not contain valid LDI values for this pixel and are discarded, as shown in Figure 2.8.

The computed LDIs can be used to process a variety of collision queries. Two LDIs can be combined to compute an intersection volume, as shown in Figure 2.9. The LDIs for two colliding objects are discretized with the same resolution on corresponding sampling grids, but with opposite viewing directions. Hence, pixels in both LDIs represent corresponding volume spans. The intersection volumes can be computed by a pixelwise intersection along depth direction. Besides, it can perform another collision query for checking the vertex-in-volume, whether the vertex allocates between a closed interval in a fragment. This approach can handle deformable and non-convex objects. It uses multiple depth images to store the different depths of a mesh. The depth complexity is the major restriction due to the approach heavily relies on the frame buffer readbacks. The LDIs technique is improved for detecting the self-intersection in [HTG04].

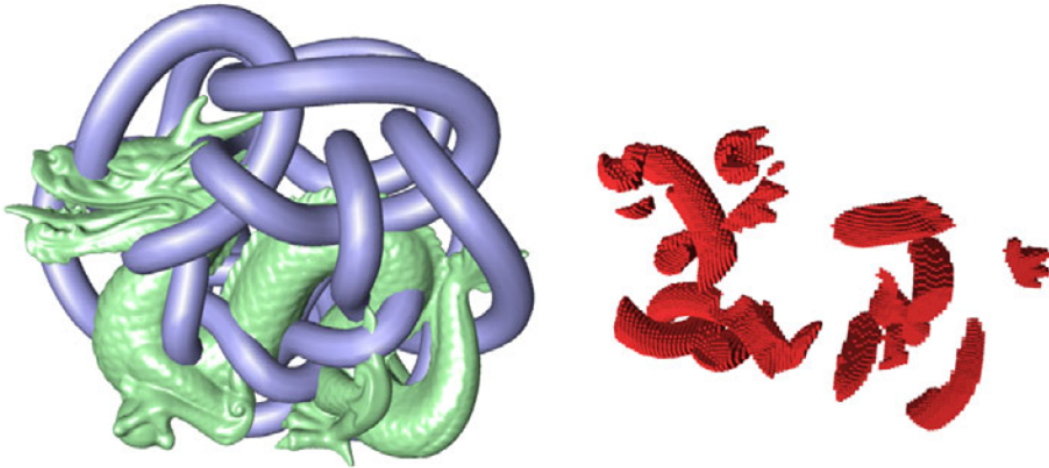


Figure 2.9: Intersection volume for Knot and Dragon[HTG03]

2.4 Collision Culling

Govindaraju et al. proposed CULLIDE which eliminated some objects that did not collide with any object, and returned a set, called potentially colliding set (PCS) [GRLM03]. Given a set O that composed of n objects, O_1, O_2, \dots, O_n . For each O_i , we divide O into two corresponding sets $S_{<i} = \{O_1, \dots, O_{i-1}\}$ and $S_{>i} = \{O_{i+1}, \dots, O_n\}$. If O_i is a colliding free object, then we can ensure that O_i will not collide with $S_{<i}$ and $S_{>i}$. Therefore, there is a trivial solution that we can determine whether O_i is colliding free. We render $S_{<i}$ and $S_{>i}$ to depth buffer and then perform visibility test for O_i using occlusion query. If O_i is fully visible with $S_{<i}$ and $S_{>i}$, then we can present that O_i dose not collide with others. By this way, we can conservatively prune some objects which are fully visible with corresponding sets. It is completed in $O(n^2)$, where n is the number of objects. An approach to PCS computation in $O(n)$ is presented in CULLIDE. It uses a two-pass rendering algorithm to perform linear time PCS pruning. In the first pass, the depth buffer is cleared to z-far and the objects are rendered in the order O_1, \dots, O_n along with occlusion queries. In other words, for each object in O_1, \dots, O_n , it renders the object and tests if it is fully visible with respect to the objects rendered prior to it. In the second pass, it clears the depth buffer and renders the objects in the opposite order along with occlusion queries. Then it perform the same operations as in the first pass while rendering each object. At the end of each pass, it tests if an object is fully visible or not. An object classified as fully-visible in both the passes does not belong to the PCS. The algorithm is shown in Figure 2.2. Finally, the exact

triangle-triangle intersection tests are performed on the CPU.

Algorithm 2.2: The algorithm of collision culling

```

//1st pass;
ClearBuffer( DEPTH );
foreach object  $O_i$ ,  $i=1, \dots, n$  do
    //performs visibility test;
    DepthMask( FALSE );
    DepthFunc( GEQUAL );
    RenderUsingOcclusionQuery(  $O_i$  );
    //updates to depth buffer;
    DepthMask( TRUE );
    DepthFunc( LESS );
    Render(  $O_i$  );
end
foreach object  $O_i$  do
    GetQuery(  $O_i$  );
end
//2nd pass;
//Same as First pass, except that the two "For each object" loops are run with
i=n, . . ., 1;

```



In CULLIDE, spatial relationships among objects are builded using occlusion queries, which are supported on graphics hardware. CULLIDE is suitable for complex environments and large objects in real-time application due to it can determine PCS rapidly in linear time. The PCS is computed using occlusion queries widely available on commodity graphics hardware. Occlusion queries involve very low bandwidth requirements in comparison to frame buffer readbacks. Quick-CULLIDE presents an extension to CULLIDE to perform inter- and intra-object collisions between complex models [HTG04]. It performs a visibility-based classification scheme to determine potentially colliding set and separate two collision-free subset from PCS, which considerably improves the performance of the collision culling.

Image-based Collision Filtering

3.1 Overview



In this chapter, we present a framework for the proposed collision filtering system. The framework consists of construction of the bounding volume hierarchies, projection and depth sorting, visibility pruning collision filtering with depth sorting, and collision independent sets construction using the hierarchical structure, as shown in Figure 3.1.

In preprocessing, we construct a bounding volume hierarchy in a top-down manner for each object using oriented bounding boxes. This bounding volume hierarchy describes an object at various levels of detail. During runtime, we perform hierarchy traversal for collision filtering. The bounding volumes of objects are projected on the view direction, and sorted in a near-to-far order. Such sorted order is used in visibility pruning and enhances the performance of CULLIDE in the cause of collision filtering. During hierarchical pruning, we can select different view direction at each level for collision filtering and construct multiple collision independent sets which will greatly reduce the pairwise computation and the times of occlusion query and rendering. Finally, the exact intersection test among potentially colliding triangles are performed on the CPU.

As stated before, the performance of CULLIDE is restricted by the extent of projection overlapping along the view, and the resulting single potentially colliding set implies a po-

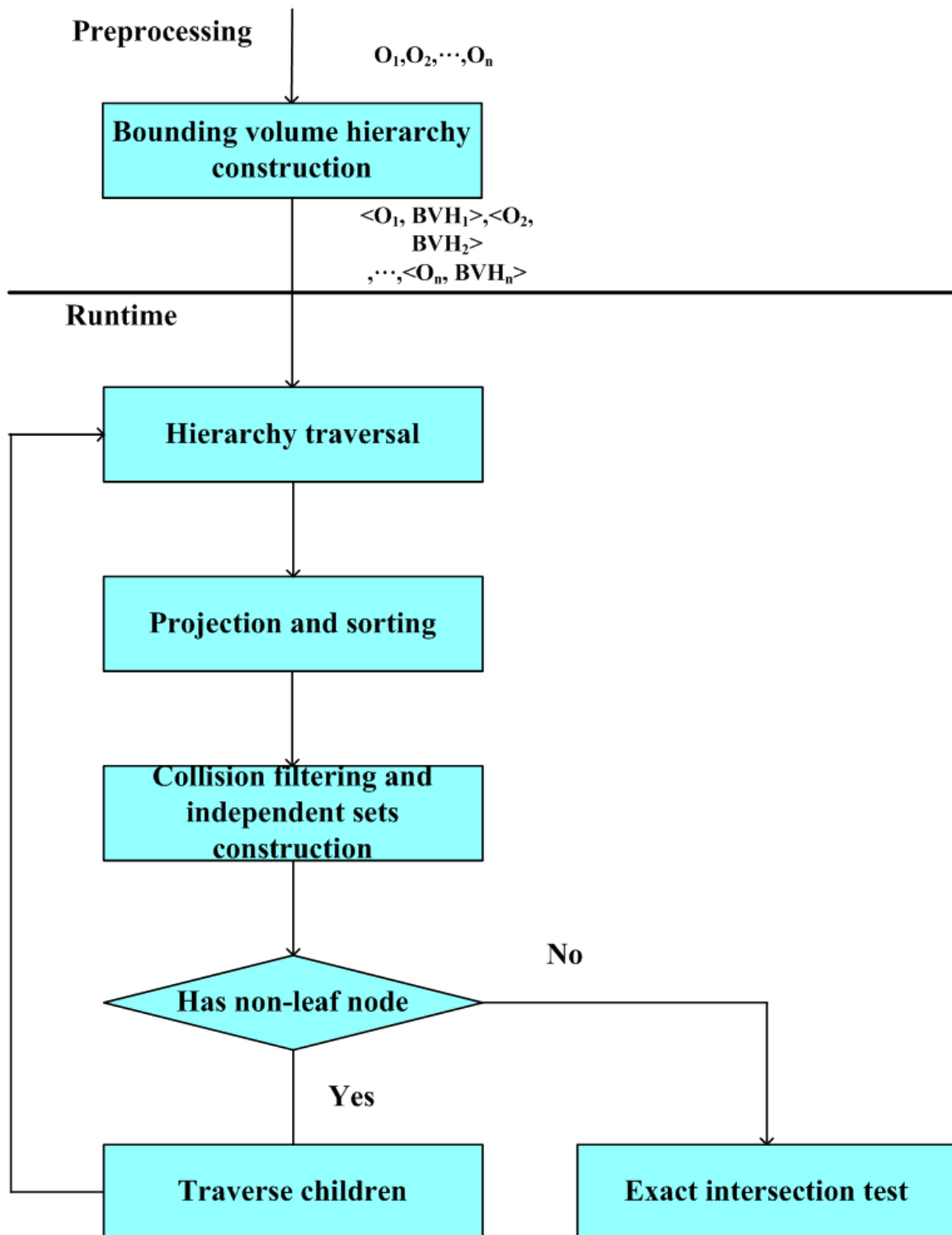


Figure 3.1: The flowchart of the proposed collision detection system.

tential large number of collision detections. For simplicity reason, in this chapter, we first introduce the collision filtering based on depth order, then incorporate collision independent sets construction into the collision filtering process. Finally, the hierarchical traversal for collision filtering is presented.

3.2 Collision filtering with depth order

The visibility-based collision filtering in CULLIDE analyses the projection of objects from 3D to 2D image using rasterization of graphics hardware. It is quite often that objects are disjoint in 3D space, but can not be claimed disjoint by CULLIDE, due to the projections could overlap with each other. As shown in Figure 3.2, all five objects are disjoint, but only object C can be excluded from the resulting potentially colliding set.

In order to amend the performance of visibility pruning in CULLIDE, we render objects in an order sorted by depth. The axis-aligned bounding box of each object is projected onto the view direction. Two extreme, Max_i and Min_i , of the projection of object O_i , for all objects, are sorted. In general, a sort would take $O(n \cdot \log n)$, where n is the number of objects. However, under the assumption of slow motion, temporal coherence can be applied. In addition to sorting, we need to keep track of the changes in overlap status of interval pairs. By doing so, the time complexity of the sorting will be reduced to $O(n + \varepsilon)$, where ε is the number of exchanges in the list [CLMP95]. Figure 3.3 is a result of projection and sorting.

Using hardware supported, occlusion queries, we can obtain how many pixels are rendered on the frame buffer [GRLM03]. and judge whether an object is fully visible with respect to the previously rendered objects. As in CULLIDE, fully visible objects are pruned in two passes from potentially colliding set. We decide a rendering order as O'_1, O'_2, \dots, O'_n , and let $Min_i > Min_j$, where $i < j$. To avoid the problem described in Figure 3.2, objects are rendered from far to near far-to-near in the first pass.

In the second pass, if objects are rendered in a near-to-far order, farer objects might be occluded by nearer ones. Instead, we move the view from z-near to z-far and render objects in a far-to-near order. In addition, we offset a tolerance distance to depth value in two passes, to make sure that the depth values of all pixels in the first pass are closer than depth values in the second pass. The depth buffer clearing is only performed once during collision pruning. Our algorithm is shown in Algorithm 3.1. Figure 3.4 shows a result of

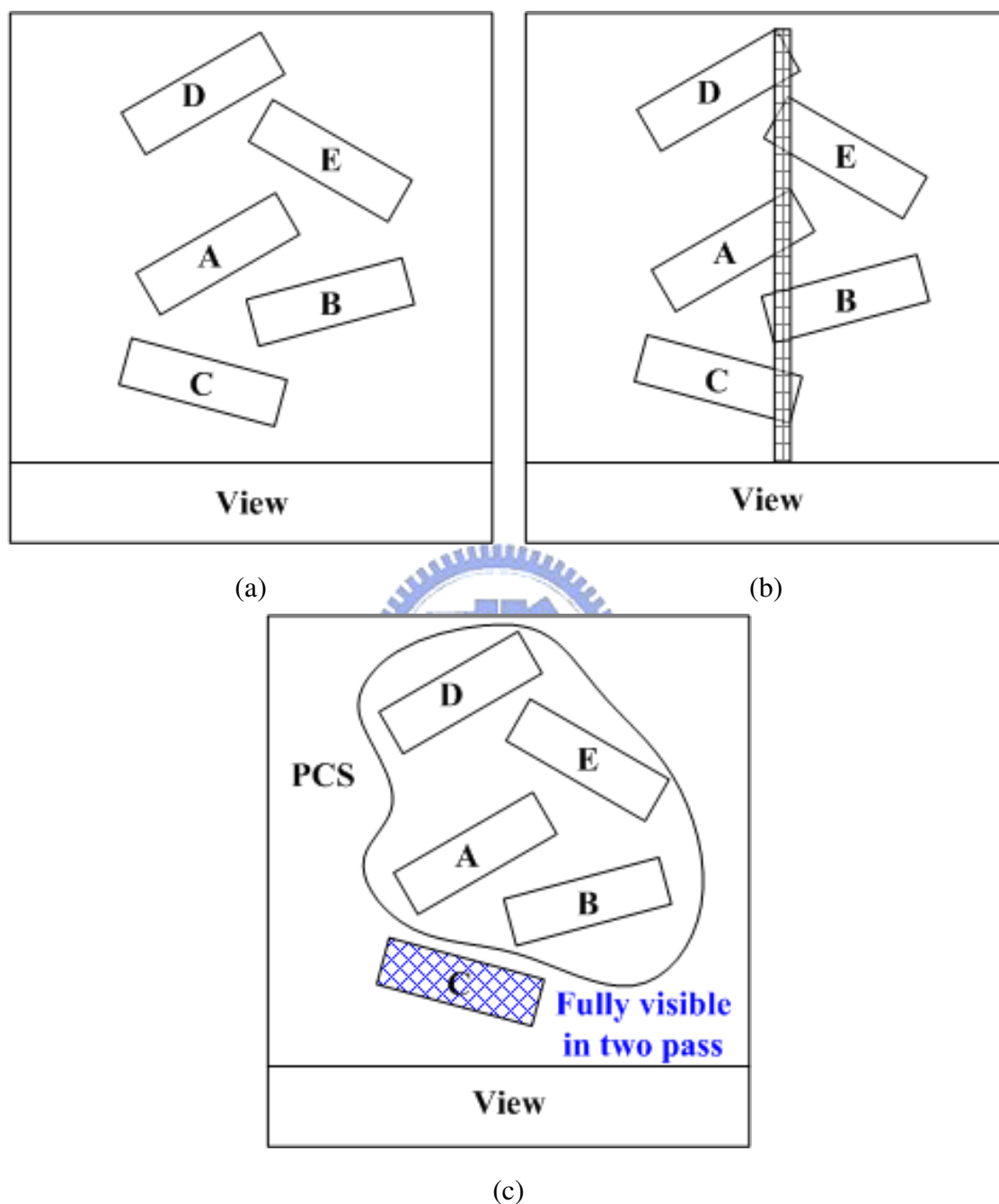


Figure 3.2: The worst case of CULLIDE: (a) It shows that there are no intersections in the environment; (b) All the projections of objects overlap with each other; (c) We only can prune the object C using visibility-based pruning in CULLIDE.

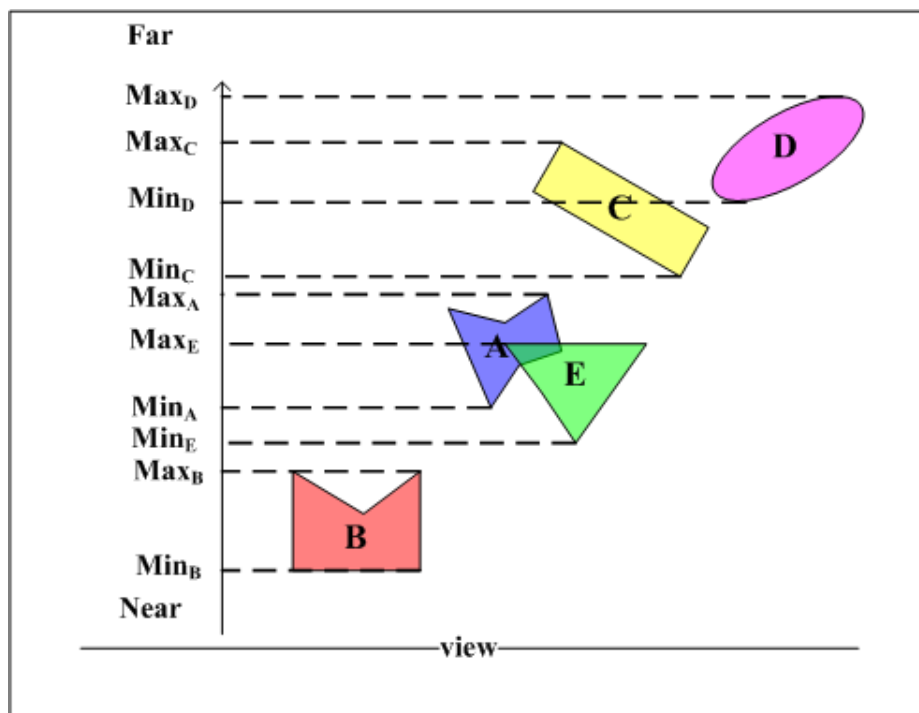


Figure 3.3: Projection and sorting.

our collision filtering with depth order.

3.3 Collision Independent Sets Construction Using Separating Surfaces

CULLIDE prunes away objects that do not collide with any object in the final PCS. Conceptually, for each pruned object, there is a separating surface between it and the PCS. Such a concept can be extended to partition the PCS into several collision independent sets, which are sets of objects satisfying the property that any pair of objects from two different sets will not collide. A separating surface partitions a set of objects into two nonempty subsets, placed at opposite side of the surface, as shown in Figure 3.5.

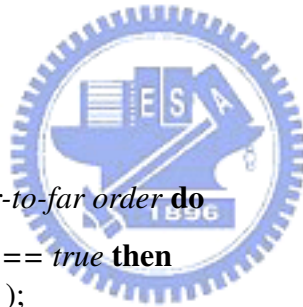
If a set S is a collision independent set, then it will not collide with any other objects, as shown in Figure 3.6(a). To determine whether S is a collision independent set, we perform the visibility test for the objects that are not in S . When all of these objects are fully visible with respect to S , we can conclude that objects in S do not collide with these objects, and hence S is a collision independent set. In the following, two lemmas are given. Lemma 1 implies that the visibility test is not required when S and O do not overlap along the

Algorithm 3.1: Our collision culling algorithm

```

//1st pass;
ClearBuffer( DEPTH );
DepthOffset( - $\epsilon$  );
foreach object  $O_i$  in a far-to-near order do
    DepthMask( FALSE );
    DepthFunc( GEQUAL );
    RenderUsingOcclusionQuery(  $O_i$  );
    DepthMask( TRUE );
    DepthFunc( LESS );
    Render(  $O_i$  );
end
foreach object  $O_i$  do
    GetQuery(  $O_i$  );
end
//2nd pass;
DepthOffset( + $\epsilon$  );
foreach object  $O_i$  in a near-to-far order do
    if IsVisibleIn1st(  $O_i$  ) == true then
        DepthMask( FALSE );
        DepthFunc( LEQUAL );
        RenderUsingOcclusionQuery(  $O_i$  );
    end
    if IsInvisibleIn1st(  $O_i$  ) == true then
        DepthMask( TRUE );
        DepthFunc( GREATER );
        Render(  $O_i$  );
    end
end
foreach object  $O_i$  do
    GetQuery(  $O_i$  );
    if IsVisibleInTwoPasses(  $O_i$  ) == true then
        RemoveFromPCS(  $O_i$  );
    end
end

```



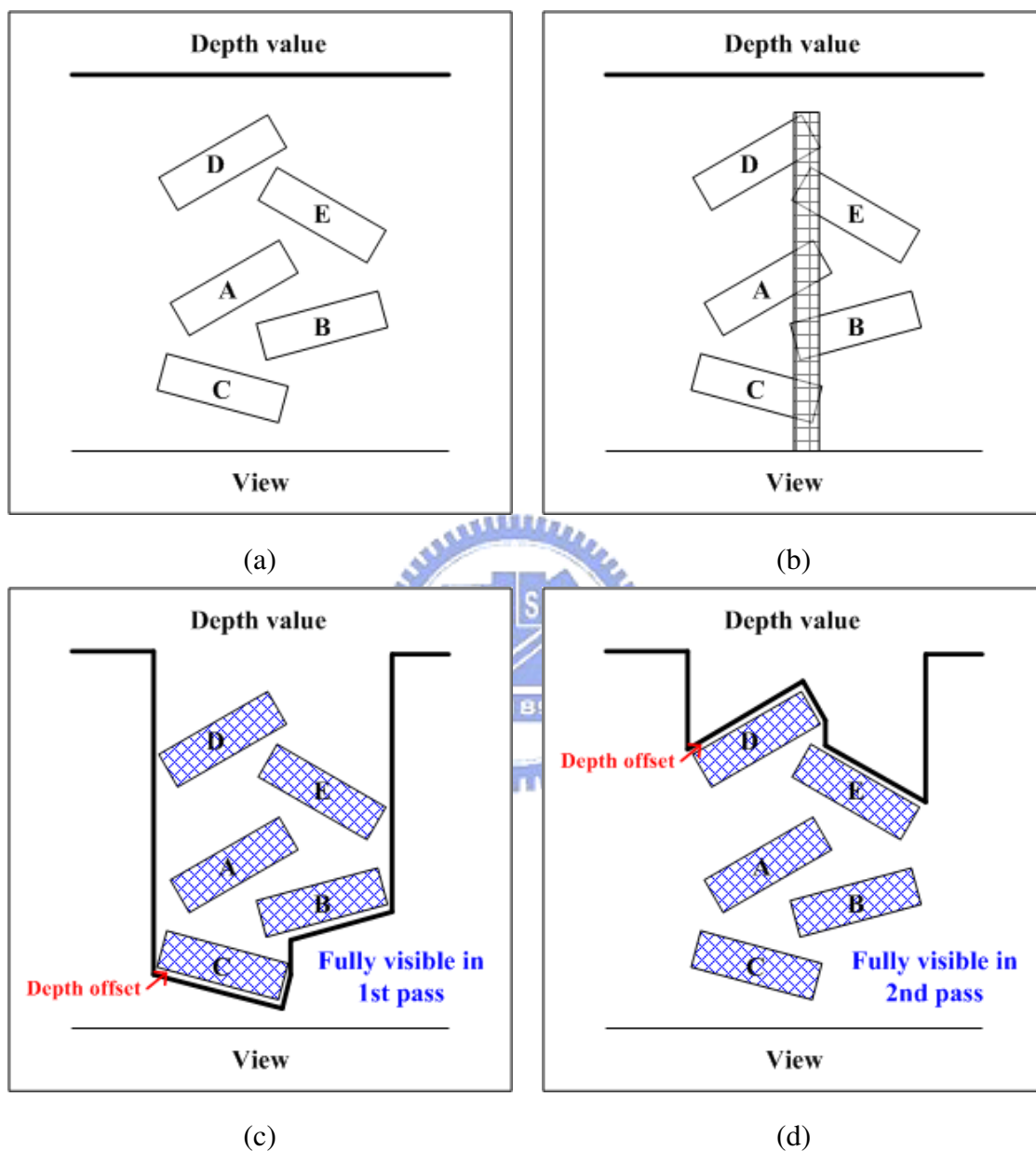


Figure 3.4: The result of our collision culling: (a) It shows that there are no intersections in the environment; (b) All the projections of objects overlap with each other; (c),(d) There are no occlusion in two passes.

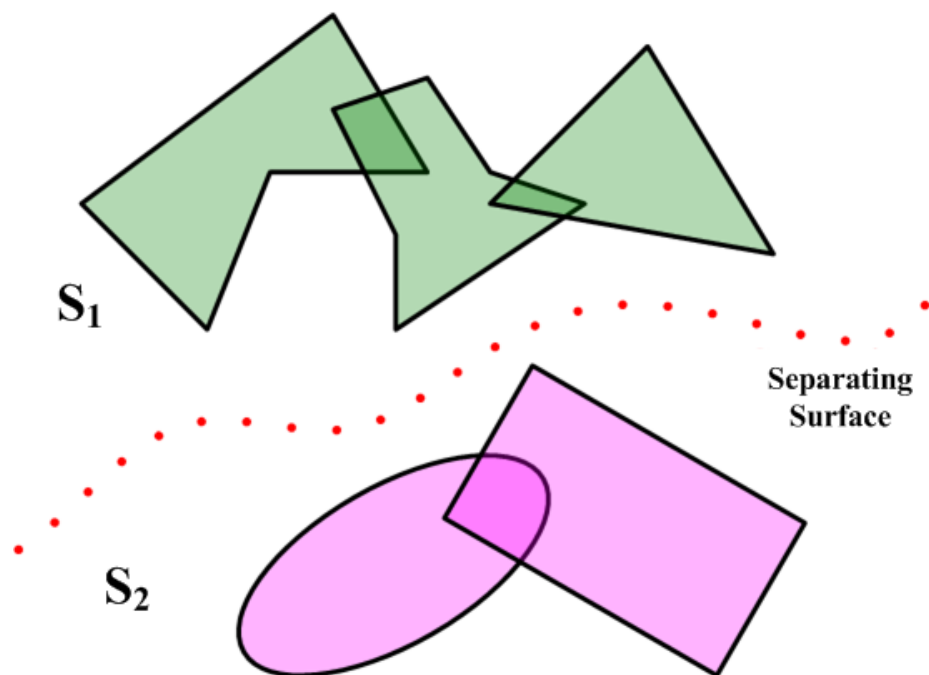


Figure 3.5: If S_1 and S_2 do not collide with each other, we can find a separating surface among them.

viewing direction.



Lemma 1. *If a set of objects S and an object O do not overlap along the viewing direction, then there exists a separating surface between S and O .*

Proof: By separating axis theorem, if projections of two objects do not overlap along a direction, then two objects do not collide with each other. Besides this, we can find a surface which separates these two objects. By the same way, if projections of a set of objects S and an object O do not overlap along the viewing direction, then any object in S and O do not collide with each other. Therefore, we can find a surface separating S and O .

Lemma 2. *If a set of objects S and an object O overlap along a direction, but O is fully visible with respect to S , then there exists a separating surface between S and O .*

Proof: It has been proven in [GRLM03] that there are no intersections between an object O and a set S , if O_i is fully visible with respect to S . Though the projections of O and S overlap along a direction, we will still find a separating surface between them.

In determining if S in Figure 3.6 is a collision independent set, by Lemma 1., O_2 and O_4 are free from the visibility test. Only the green objects O_1 and O_3 should be tested. If

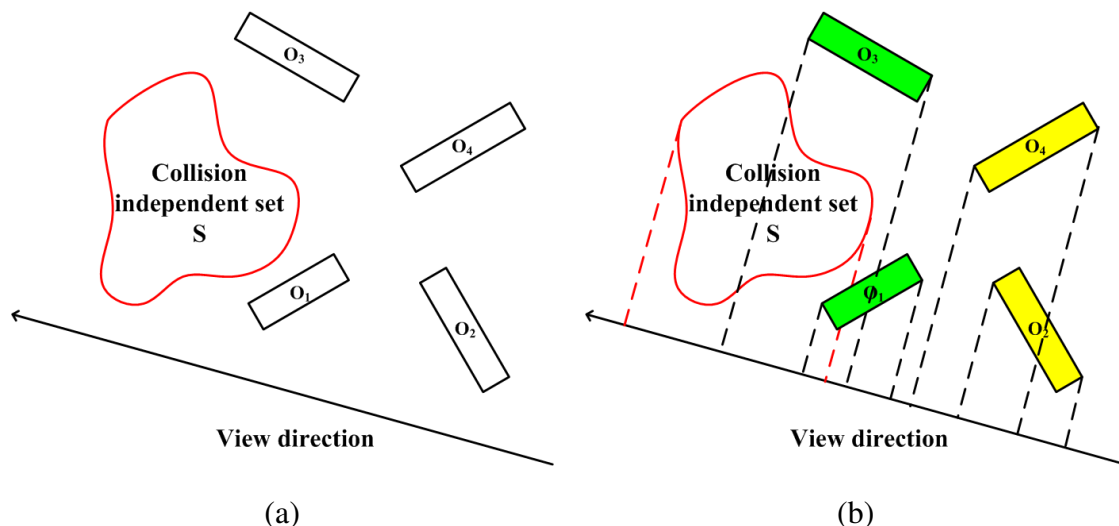


Figure 3.6: (a) Collision independent set; (b) The projection of green objects overlap with the collision independent set, and the projection of yellow objects do not overlap with the set.

all of the green objects are fully visible with respect to S , then S is a collision independent set. If any of the green objects is not fully visible, then we insert the object into S .

In the first pass, we reduce objects in far-to-near order based on the minimum depth Min_i of object O_i . If an object is fully visible, it is discarded; otherwise, it is the first element of a newly created collision independent set. Once a collision independent set S is constructed. We continue to render the objects that are overlapped with S along the view direction. We discard the one that is fully visible and insert to the S the one that is not fully visible. Note that once such an insertion occur, we need a way to indicate which objects are overlapped with the new S . After all the objects overlapped with S are rendered, we continue to render the remaining objects and search for the next object is not fully visible and construct the next collision independent set. After the first pass, we guarantee there is no overlapping among the interval of the projection of potential collision independent sets along the view direction. Furthermore, we indicate the separating surfaces at the nearest part of potential collision independent sets.

Collision independent sets constructed in the first pass generally are not complete. Those objects that are fully visible with respect to the previously rendered object may collide with objects in some collision independent sets. In the second pass, we move the view from near to far and render those objects in far-to-near order, and insert each object that is not fully visible to an appropriate collision independent set. Such a independent set

can be found in constant time by using separating surface indicated in the first pass.

In order to perform the collision independent sets construction in linear time, we expect to have one visibility test for each object in the first pass. We propose a method for updating the overlapping region with respect to the collision independent set under construction. We design a probing surface pointing to the minimum depth of objects that are overlapped with the object to be rendered. Probing surface will be assigned to overlapping region once an object becomes the first object in a newly created collision independent set or is inserted into a collision independent set.

Since we construct collision independent sets on the 1D view direction, if two collision independent sets overlap along the view direction then we can not distinguish them. In section 3.4, we will apply the algorithm to a hierarchical structure, and replace single view direction with multiple view directions to reduce this limitation.

Regard Figure 3.7 as the example. The visibility test of objects of O_5 , O_8 , O_3 , O_4 , O_2 , O_6 , O_1 , and O_7 are performed in the first pass. We disregard O_5 due to it is fully visible. O_8 is non-fully visible, we construct a collision independent set CIS_1 and insert O_8 into it set. The only overlapped object with CIS_1 is O_3 and O_3 is fully visible, we can confirm that CIS_1 will not collide with any objects that are not yet rendered. Similarly, CIS_2 containing O_4 is constructed since O_4 is not fully visible, and the overlapped object O_2 is fully visible. CIS_2 also dose not collide with remaining objects. After CIS_3 containing O_6 is constructed, we do visibility test for the overlapped objects O_1 and O_7 . Only O_7 is not fully visible, and inserted into CIS_3 . Separating surfaces SS_1 , SS_2 , and SS_3 point to the minimum depth of CIS_3 , CIS_2 , and CIS_1 , respectively. In the second pass, we only perform visibility test for O_1 , O_2 , O_3 , and O_5 , since these objects are fully visible in first pass. O_2 , O_3 , and O_5 are not fully visible, and inserted into CIS_3 , CIS_2 , and CIS_1 , respectively. As a result of three collision independent sets $PCS_1 = \{O_5, O_8\}$, $PCS_2 = \{O_3, O_4\}$, and $PCS_3 = \{O_2, O_6, O_7\}$ are constructed.

Some symbols are described before we introduce the proposed algorithms. E_i is a temp for traversing all projection elements in the sorted list. In each iteration, E_i scans the list from far to near in the first pass and then from near to far in the second pass. *ActiveCIS* is used to indicate the collision independent set that is being constructed. *SS* is a separating surface. It will separate the *ActiveCIS* from other collision independent sets. *PS* is a probing surface pointing to the minimum depth of objects that are overlapped with the

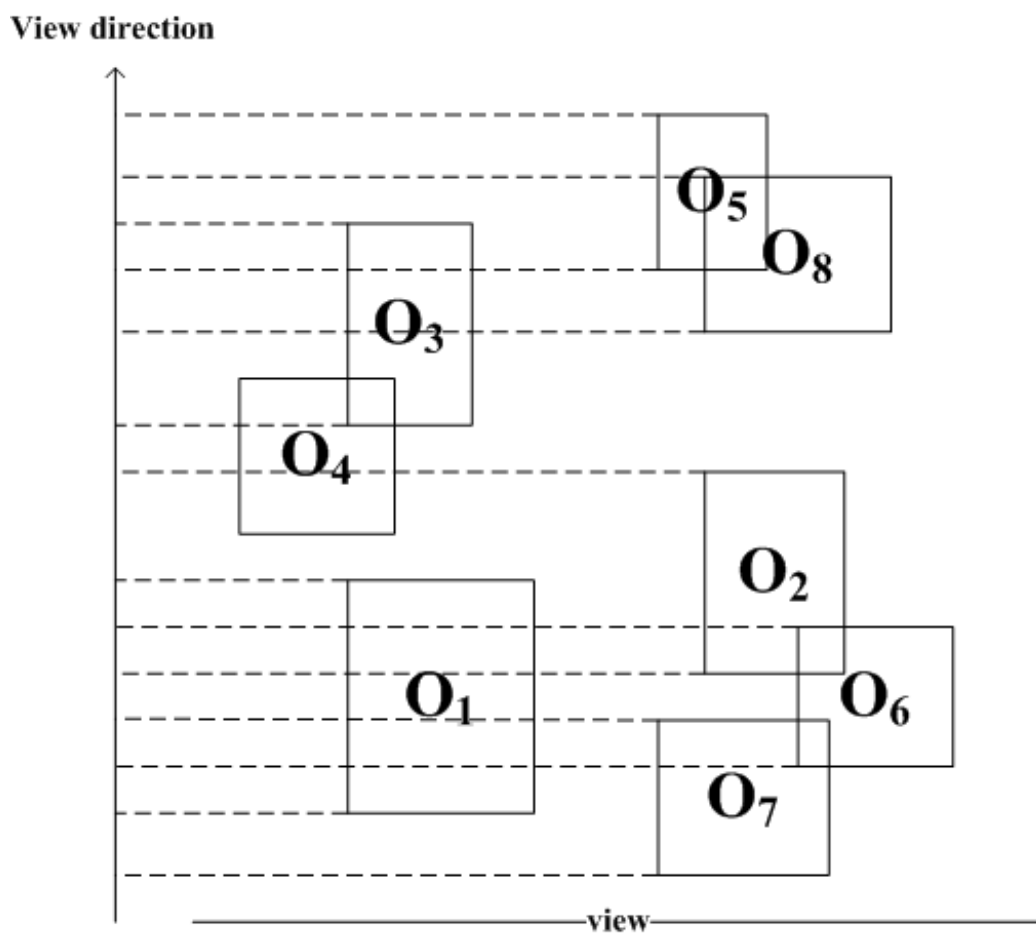


Figure 3.7: An example for collision independent sets construction.

Table 3.1: Parameters of collision independent sets construction

Notation	Description
E_i	each sorted element after projection
$ActiveCIS$	the constructing collision independent set
SS	the potential separating surface
OV	overlapping region corresponding to $ActiveCIS$
PS	probing surface for updating overlapped region

object to be rendered. OV is a depth indicating the overlapping region of $ActiveCIS$. PS will be assigned to OV once an object becomes the first object in a newly created collision independent set of is inserted into $ActiveCIS$. The algorithm of collision independent sets construction is shown in Algorithm 3.2.

3.3.1 First Pass in Collision Culling

PS is set to positive infinity initially. We traverse all projection elements from far to near and construct collision independent sets. If E_i points the maximum depth of an object O_i and PS is greater than the minimum depth of O_i , then we update PS to the minimum depth of O_i . If E_i is a minimum value of O_i , then O_i is performed visibility test. If O_i is not fully visible, we check whether the $ActiveCIS$ exists. If $ActiveCIS$ is nil, then we construct a new collision independent set, represented by $ActiveCIS$; otherwise we insert O_i into $ActiveCIS$, and update separating surface SS and overlapping region OV . The separating surface SS points to the minimum value of O_i showing that there might be a separating surface pass through it. OV points to the position of probing surface PS .

After O_i performed visibility test, if the $ActiveCIS$ exists ($ActiveCIS \neq nil$) and E_i point to a overlapping region OV , then we can make sure that there is a separating surface passing through SS and separating $ActiveCIS$ and other untested objects. We record the $ActiveCIS$ and SS into a list of collision independent set, and set $ActiveCIS$ to nil. This means that there is no potentially colliding set expanding.

3.3.2 Second Pass in Collision Culling

In the second pass, we detect objects that belong to the collision independent sets constructed in the first pass. It is completed in linear time using the separating surfaces which are derived in the first pass. These separating surfaces will avoid ambiguous situations and ensure that each potentially colliding object will be inserted into correct collision independent set. We scan projection elements from near to far. If E_i is a minimum value of O_i , then we check whether O_i is fully visible in the first pass. If the object is not fully visible in the first pass and E_i points to separating surface SS , then we assign the collision independent set corresponding to SS to *ActiveCIS*. If the object is fully visible in the first pass, then we perform visibility test for it, and insert it into *ActiveCIS* if the object is non-fully visible in the second pass.

3.3.3 A detailed process of collision independent sets construction

An example is shown in Figure 3.8, traverse the projection elements: $e_5, e_8, e_3, b_5, b_8, e_4, b_3, e_2, b_4, e_1, e_6, b_2, e_7, b_6, b_1,$ and b_7 , where e_i and b_i are the farthest and nearest part of O_i with respect to the view direction, respectively. Objects of $O_5, O_8, O_3, O_4, O_2, O_6, O_1,$ and O_7 are performed visibility test in the first pass.

In Figure 3.9, we sequentially traverse projection elements of $e_5, e_8,$ and e_3 . $e_5, e_8,$ and e_3 are farthest part of objects, and probing surface PS is updated to b_3 , the nearest position of $O_5, O_8,$ and O_3 . PS records the conservative overlapping region with respect to e_3 .

b_5 and b_8 are traversed, and O_5 and O_8 are performed visibility test in order. Only O_8 is non-fully visible, and then we construct a potentially collision independent set CIS_1 , insert O_8 into the set and assign CIS_1 to *ActiveCIS*. The separating surface SS_1 and overlapping region OV_1 point to b_8 and PS , respectively. This means that there may exist a separation surface between CIS_1 and untested objects. The result is shown in Figure 3.10. If all untested objects are fully visible before OV_1 , then CIS_1 is a collision independent set and there is a separating surface pass through b_3 .

e_4 and b_3 are traversed in order, PS is updated to b_4 and O_3 is performed visibility test. O_3 is fully visible and PS_1 points to b_3 . This means all overlapped objects with respect to CIS_1 are fully visible, and CIS_1 will not collide with any objects that nearer than SS_1 . we record CIS_1 and SS_1 , and assign nil to *ActiveCIS*, as shown in Figure 3.11.

We traverse e_2 and b_4 in order, update PS to b_2 and perform visibility test for O_4 . CIS_2 is constructed and assigned to $ActiveCIS$ due to O_4 is non-fully visible and $ActiveCIS$ is nil. SS_2 and OV_2 with respect to CIS_2 point to b_2 and PS , respectively, as shown in Figure 3.12. A new collision independent set is found at O_4 , and OV_2 is indicated by PS .

e_1 , e_6 and b_2 are traversed in order, PS is updated to b_1 and O_2 is performed visibility test. The expansion of CIS_2 is finished due to O_2 is fully visible and OV_2 points to b_2 , and we remove CIS_2 from $ActiveCIS$, as shown in Figure 3.13. O_2 is an only overlapped object with CIS_2 , and it is fully visible. This means CIS_2 dose not collide any objects that nearer than SS_2 .

We traverse e_7 and b_6 in order, update PS to b_7 and perform visibility test for O_6 . CIS_3 is constructed and assigned to $ActiveCIS$ due to O_6 is non-fully visible and $ActiveCIS$ is nil. SS_3 and OV_3 with respect to CIS_3 point to b_6 and PS , respectively, as shown in Figure 3.14. b_1 and b_7 are traversed, and we perform visibility test with O_1 and O_7 in order. Only O_7 is non-fully visible and $ActiveCIS$ is not nil, then we insert O_7 into CIS_3 . The result of the example in the first pass is shown in Figure 3.15. There are three collision independent sets $PCS_1 = \{O_8\}$, $PCS_2 = \{O_4\}$, and $PCS_3 = \{O_6, O_7\}$ found.

In the second pass, we traverse the projection elements in near-to-far order, and perform visibility test for objects of O_1 , O_2 , O_3 , and O_5 which are fully visible in first pass. O_2 , O_3 , and O_5 are non-fully visible, and inserted into CIS_3 , CIS_2 , and CIS_1 , respectively. There are three collision independent sets $PCS_1 = \{O_5, O_8\}$, $PCS_2 = \{O_3, O_4\}$, and $PCS_3 = \{O_2, O_6, O_7\}$, and two separating surfaces are found among collision independent sets. The result is shown in Figure 3.16.

3.4 Collision filtering in hierarchical structure

3.4.1 Bounding Volume Hierarchy Construction

The construction of bounding volume hierarchy can be done in either top-down, bottom-up, or incremental fashion. In order to build a complete tree and group neighboring primitives, we use top-down construction. Collision culling will be processed in a coarse to fine manner by traversing the bounding volume hierarchy which will substantially reduce the time required for visibility queries [BM04]. This should be handled by broad phase collision

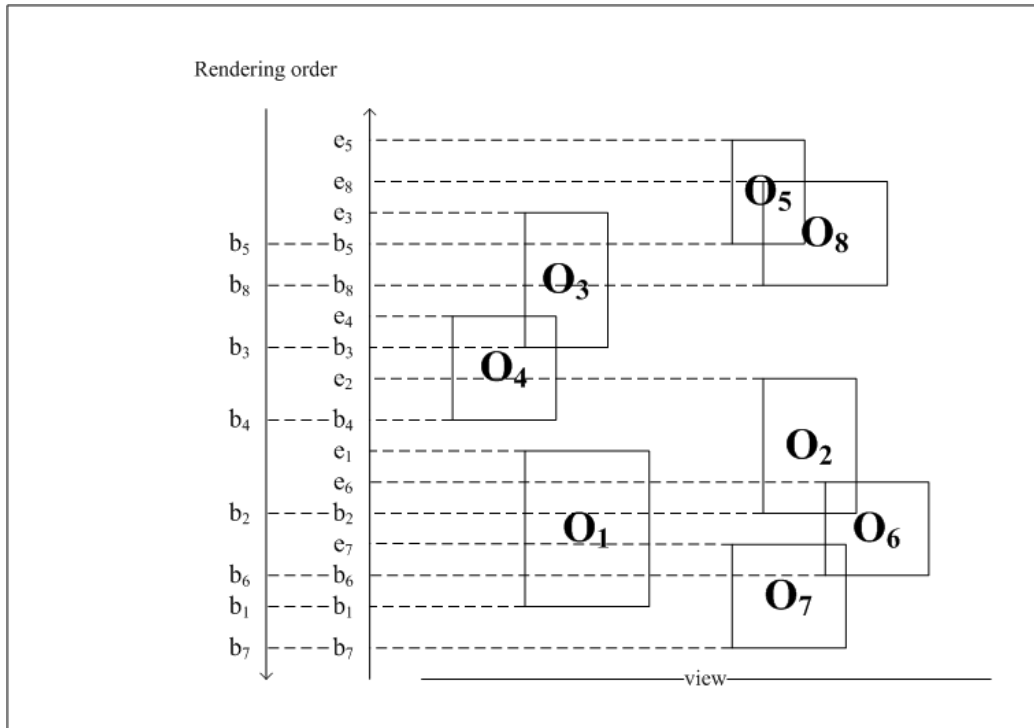


Figure 3.8: An example, a scene consists of $O_1, O_2, O_3, O_4, O_5, O_6, O_7,$ and O_8 .

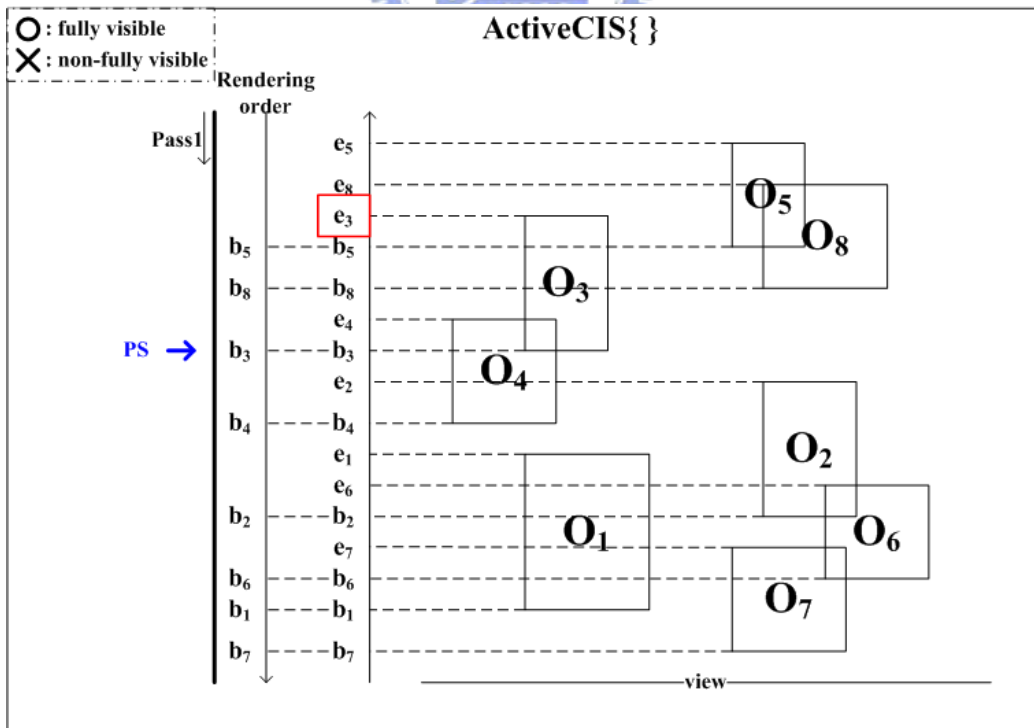


Figure 3.9: Traverse projection elements of $e_5, e_9,$ and e_3 . The global probing surface PS_g is updated to b_3 .

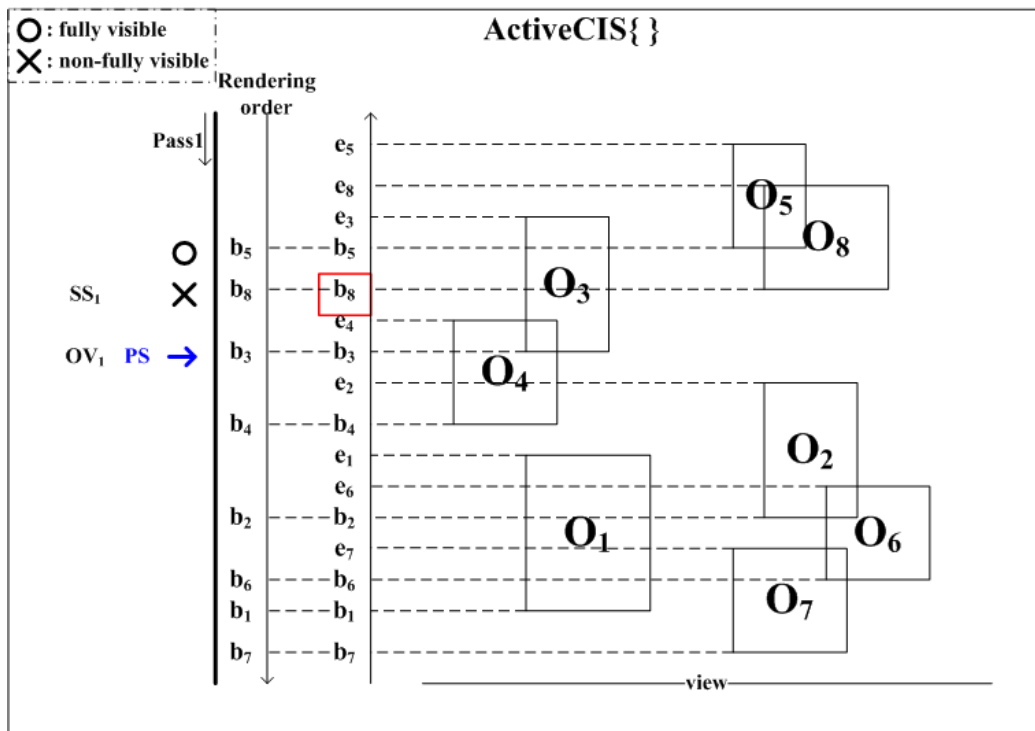


Figure 3.10: Traverse projection elements of b_5 and b_8 , collision independent set CIS_1 is constructed at O_8 .

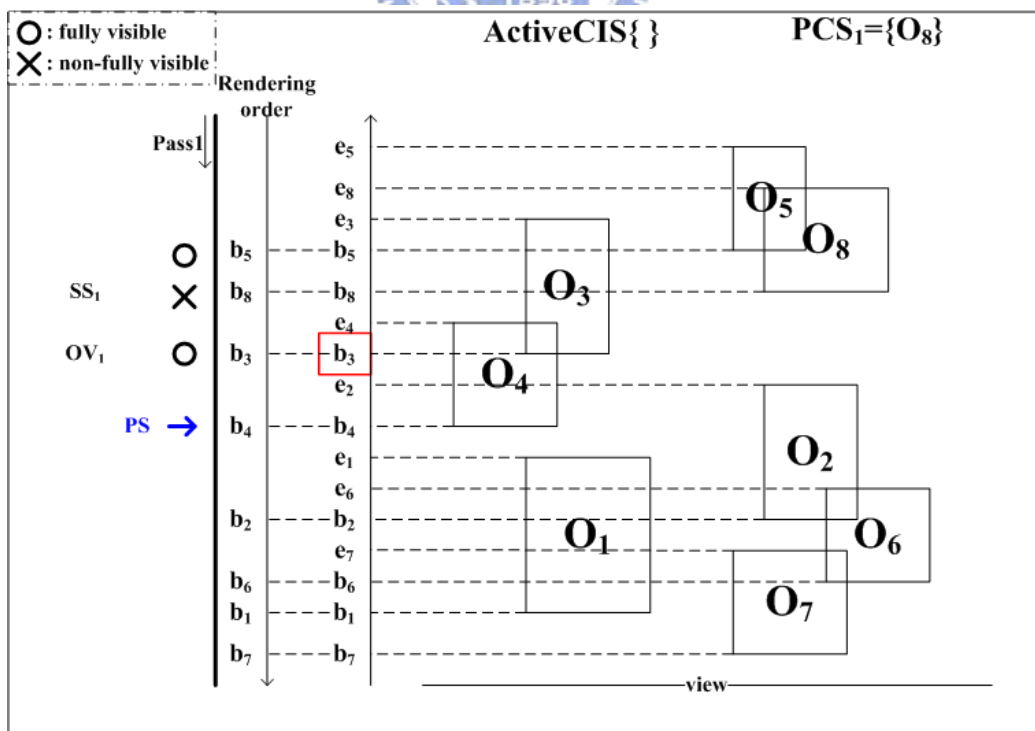


Figure 3.11: Traverse projection elements of e_4 and b_3 , and the expansion of CIS_1 is finish at b_3 .

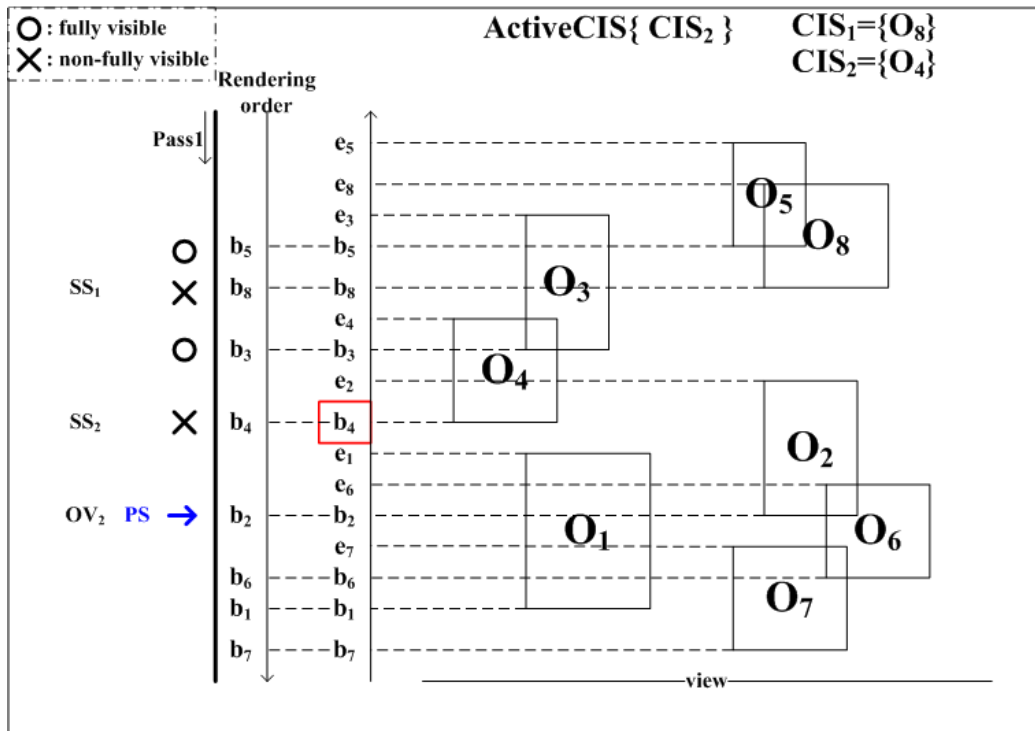


Figure 3.12: Traverse projection elements of e_2 and b_4 , collision independent set CIS_2 is constructed at O_4 .

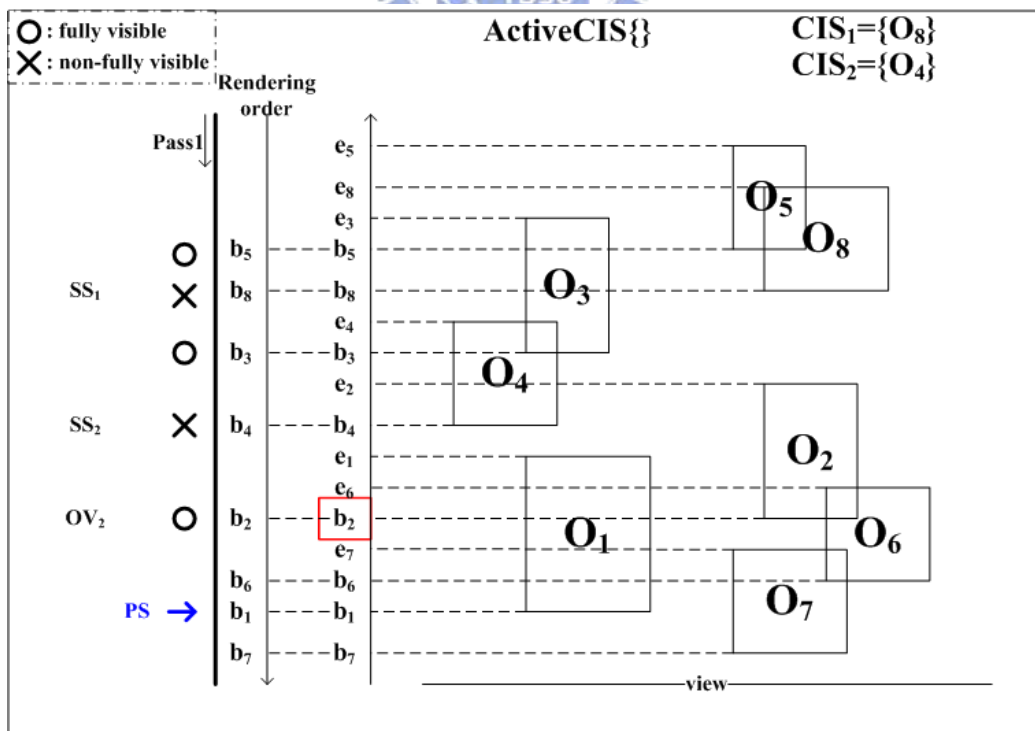


Figure 3.13: Traverse projection elements of e_1 , e_6 and b_2 , and the expansion of CIS_2 is finish at b_2 .

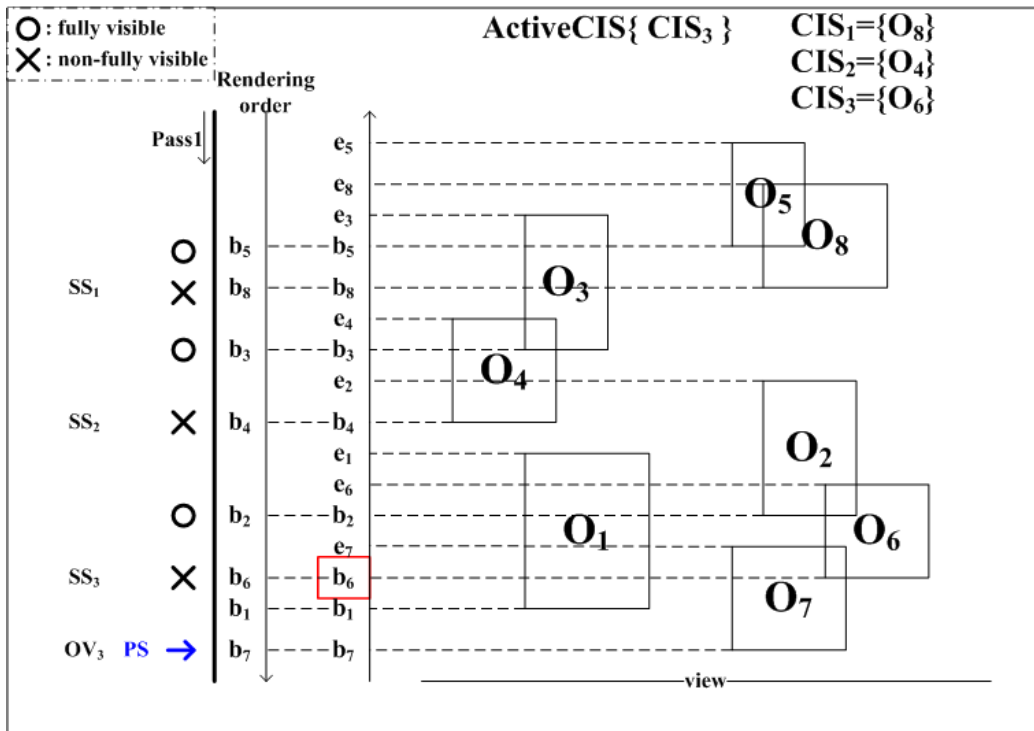


Figure 3.14: Traverse projection elements of e_7 and b_6 , collision independent set CIS_3 is constructed at O_6 .

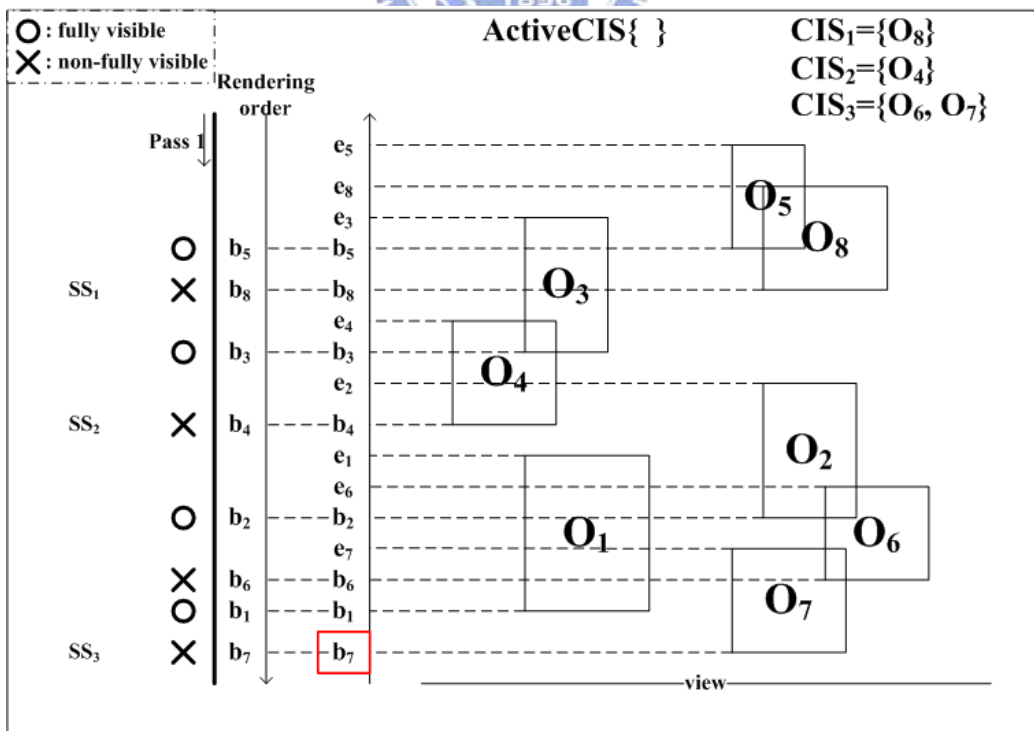


Figure 3.15: After the first pass, three independent colliding sets, $PCS_1 = \{O_8\}$, $PCS_2 = \{O_4\}$, and $PCS_3 = \{O_6, O_7\}$, and three separating surfaces, SS_1, SS_2, SS_3 , are found.

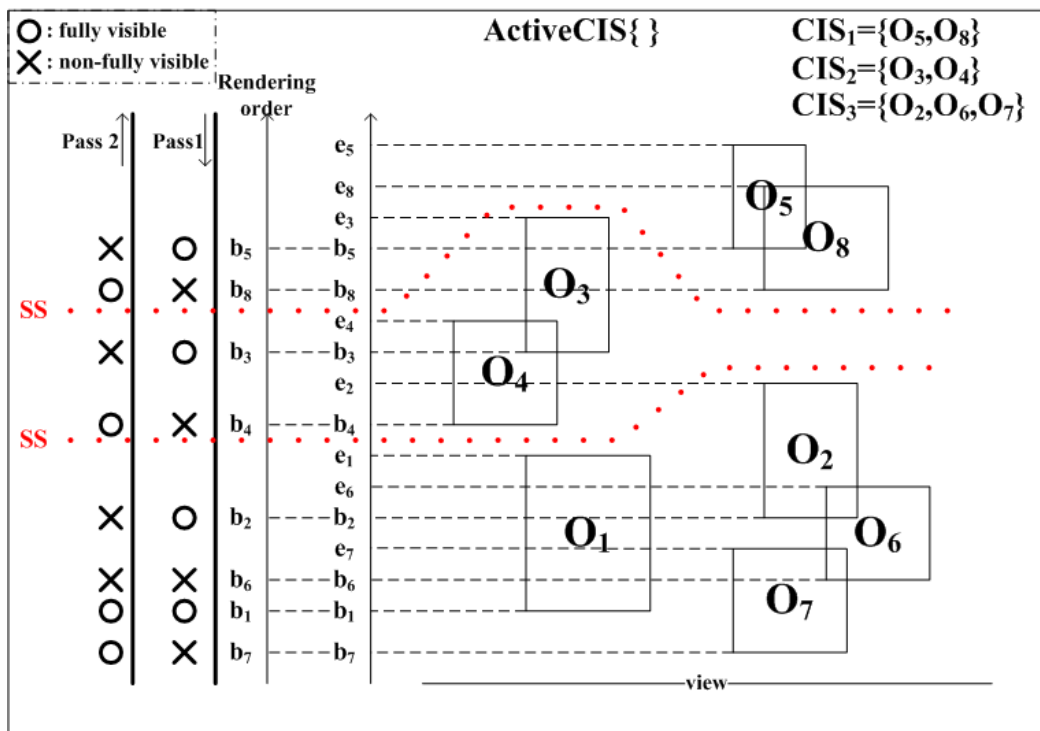


Figure 3.16: After the second pass, the three collision independent sets will be finalized as $CIS_1 = \{O_5, O_8\}$, $CIS_2 = \{O_3, O_4\}$, and $CIS_3 = \{O_2, O_6, O_7\}$, and there are two separating surfaces found.

Algorithm 3.2: The algorithm of collision independent sets construction

```

//1st pass;
ActiveCIS = nil, OV = nil, CISList = {}, PS = ∞;

foreach element  $E_i$  in a far-to-near order do

  if  $E_i == MAX$  and  $PS > GetMin( E_i )$  then
     $PS = GetMin( E_i );$ 

  else if  $E_i == MIN$  then
     $VisibilityTest( E_i );$ 

    if  $IsInvisible( E_i ) == true$  then

      if  $ActivePCS = nil$  then
         $ConstructCIS( ActiveCIS );$ 

      end

       $InsertIntoCIS( O_i, ActiveCIS );$ 

       $SS = E_i;$ 

       $OV = PS;$ 

    end

    if  $ActivePCS \neq nil$  and  $OV = E_i$  then
       $Complete( ActiveCIS, SS, CISList );$ 

       $ActivePCS = nil;$ 

    end

  end

if  $ActivePCS \neq nil$  then
   $Complete( ActiveCIS, SS );$ 

end

//2nd pass;

foreach element  $E_i$  in a near-to-far order do

  if  $E_i == MIN$  then

    if  $IsInvisibleInIst( E_i ) == true$  and  $SSExists( E_i )$  then
       $ActiveCIS = GetCIS( E_i, CISList );$ 

    else if  $IsVisibleInIst( E_i ) == true$  then
       $VisibilityTest( E_i );$ 

      if  $IsInvisible( E_i ) == true$  then
         $InsertIntoCIS( O_i, ActiveCIS );$ 

      end

    end

  end

end

```



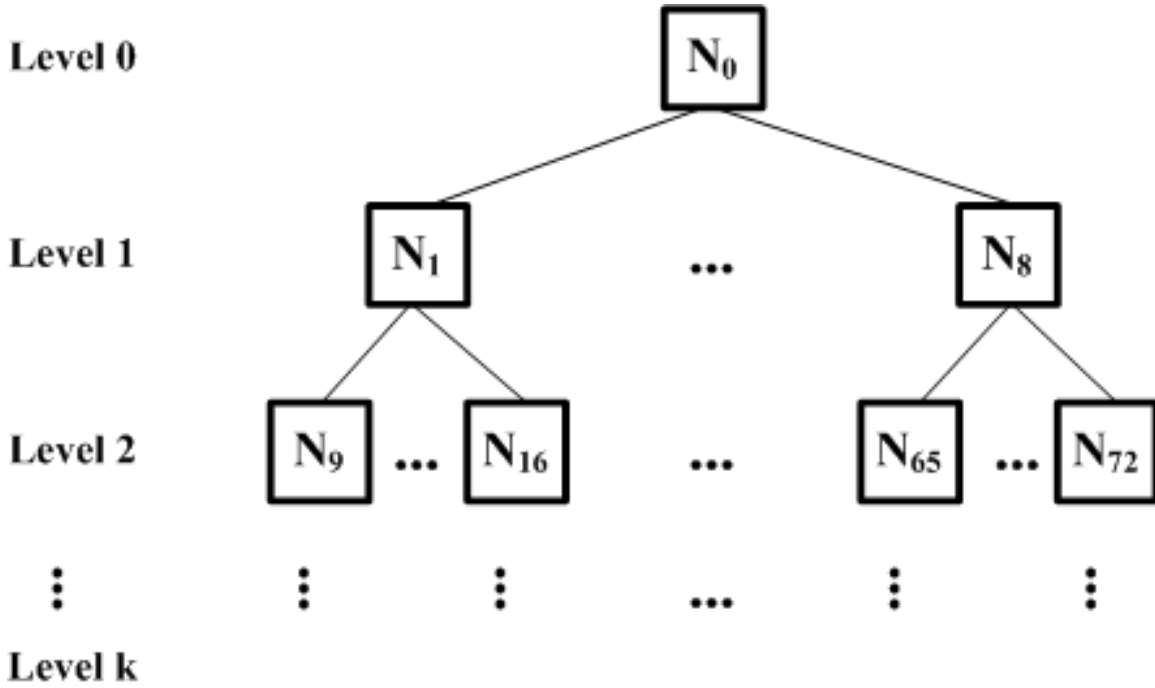


Figure 3.17: Bounding volume hierarchy with a 8-ary tree.

detection. But bounding volume hierarchy is mainly for narrow phase collision detection. Since a frame buffer clearing is called for each level of hierarchy, we replace the original binary trees for bounding volume hierarchy by the 8-ary trees to decrease the height of a tree, as shown in Figure 3.17. Moreover, we offset boundary of volumes by a small distance to avoid collisions missing due to insufficient frame buffer resolution in collision culling.

3.4.1.1 Bounding Volume Hierarchy Optimization

In this section, we will present how to construct oriented bounding boxes. The covariance matrix provides a measure of the correlation among the vertices in a set. The direction and density of the vertex distribution can be described using the eigenvalues and eigenvectors of the covariance matrix. The covariance matrix $[C_{ij}]$ of vertices of an object can be represented by

$$C_{jk} = \sum_{i=1}^n \frac{A^i}{12A^H} (9c_j^i c_k^i + p_j^i p_k^i + q_j^i q_k^i + r_j^i r_k^i) - c_j^H c_k^H$$

Let vertices p^i, q^i , and r^i are the three vertices of the i 'th triangle. where $A^i = \frac{1}{2} |(p^i - q^i) \times (p^i - r^i)|$ and $A^H = \sum_j^n A^j$ representing the area of triangle i and the summed area of all triangles, respectively, $c^i = (p^i + q^i + r^i)/3$ and $c^H = \frac{\sum_i A^i \cdot c^i}{A^H}$ representing the centroid of triangle i and all triangles, respectively. The three eigenvectors of the covariance matrix

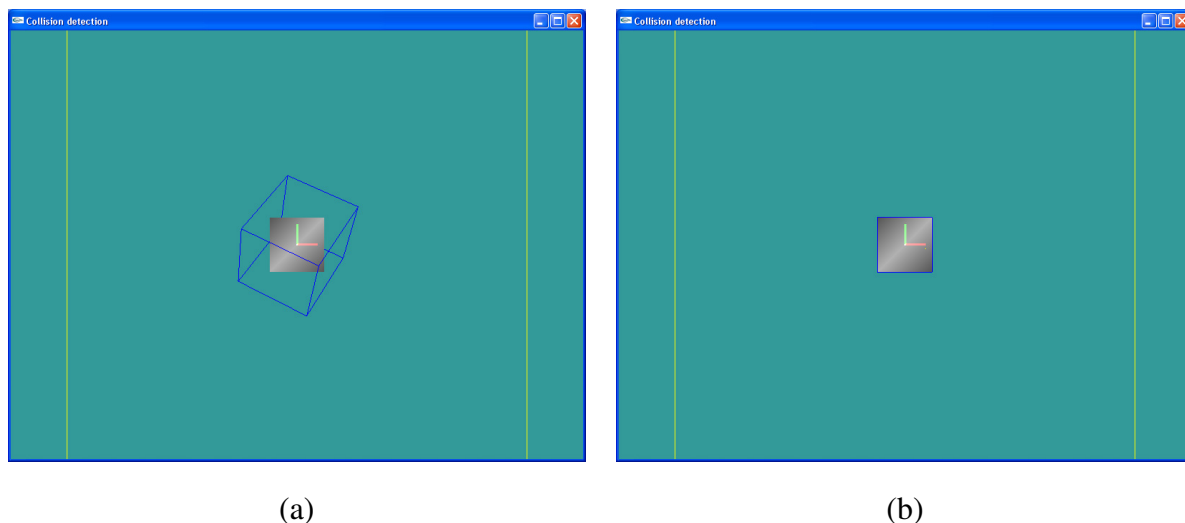


Figure 3.18: Bounding volume hierarchy construction using (a) three eigenvectors of the covariance matrix; (b) optimizing principal components

can be used as the three axes of the oriented bounding box. This, however, is not optimal. One improved approach, called optimizing principal components [Eri05], is to first align the box along one of the eigenvectors, and then determine two remaining eigenvectors by using the computed minimum-area bounding rectangle of the projection of vertices onto the plane perpendicular to the first axis. This method determines the best orientation of two remaining axes and produces the smallest volume oriented bounding box. Figure 3.18 shows the bounding volume hierarchy constructed using 3 eigenvectors of the covariance matrix and optimizing principal components, respectively.

3.4.2 Sorting and collision independent sets construction

In subsection 3.3, we present an approach to partition potentially colliding set into multiple subsets. We project axis-aligned bounding volumes of objects and find the separating surfaces along the view direction. However, if the intervals of the projections of two collision independent sets overlap along the view direction, these sets might be unable to distinguish. For this reason, we attempt to apply the proposed algorithm to a hierarchical structure.

During hierarchy traversal, we select the different view directions for performing collision culling for each level. The axis-aligned bounding volumes of remaining objects are rearrange along the view direction, and the projection elements are also scanned from far to near and then from near to far. For each independent set, we maintain an additional sorted

list for the projections of objects of the set, because we only consider the overlapping region in the same independent set, and the subsets are computed in each collision independent set. In this way, collision independent sets will be constructed more efficiently, we can reduce the influence of that the projections of collision independent sets overlap on the specific direction. Moreover, the first tested object of each collision independent set need not carry out visibility test, and the last tested object of each collision independent set need not update to the depth buffer.



CHAPTER 4

Results

We have implemented the proposed algorithm in C++ and OpenGL, and compare the performance of the algorithm with CULLIDE [GRLM03] on VC 7.0 with 3.0GHz Intel Pentium 4 CPU, 512 MB memory, and Geforce 6800 GPU. Three different scenes of varying benchmarks are tested. The dynamic computation is supported by the physics engine Tokama.

Since bounding volume hierarchy is part of the proposed approach and it is beneficial to the objects of high geometry complexity, performances of CULLIDE (called CULLIDE-w/o-BVH), CULLIDE with bounding volume hierarchy (called CULLIDE with BVH), and our approach are compared by collision detection time. We also implement CULLIDE-w/o-BVH in two stages, object level and sub-object level. For each level, we choose $+x$ -axis, $-x$ -axis, $+y$ -axis, $-y$ -axis, $+z$ -axis, and $-z$ -axis to regard as view direction. After potentially colliding set computation, potentially colliding pairs are determined by using sweep-and-prune algorithm, and exact intersections of potentially colliding pairs are computed on CPU eventually. For possible optimizations in CULLIDE, we perform off-screen rendering by using frame buffer object such that the resolution of testing buffer will not be restricted to window size. Testing on high resolution can be done using frame buffer object with cost lower than pbuffer. Additionally, we use vertex array object to place geometry of a model on the video memory beforehand to avoid a large number of geometric data transferring between CPU and GPU. In the experiments, two timing statistics are recorded.

The first is the average collision detection time, defined as

$$Average_CD_time = \frac{\sum_{i=1}^n CD_time_i}{n}$$

where n is the number of frames performed and CD_time_i is the collision detection time in i 'th frame. The second is the maximum collision detection time

$$Maximum_CD_time = \max_i \{CD_time_i\}.$$

If the maximum collision detection time is too large, that means the algorithm will not detect collisions efficiently when there are many intersections, and we can not calculate the cost of detecting collision in prediction.

4.1 Performance analysis in simple environment

Environment 1 is a scene of lower geometry and depth complexity, consisting of 50 torii and 2 planes. Each torus consists of 800 triangles, and each plane consists of 1800 triangles. All the torii move with gravity but two planes are fixed.

We perform physics simulation at 500-1200 frame buffer resolutions. We repeatedly simulate 10 times for each frame buffer resolution over a period of time, and some of the simulating process are shown in Figure 4.1. Timing statistics of simulation results are shown in Table 4.1 and Figure 4.2. It is shown that the average collision detection time is around 4-6 milliseconds and the maximum collision detection time is lower than 30 milliseconds in our approach. In CULLIDE-w/o-BVH and CULLIDE with BVH, the average collision detection time is around 9-12 and 7-10 millisecond and the maximum collision detection time is around 34-45 and 29-33 milliseconds, respectively. Our approach is about two times faster than CULLIDE-w/o-BVH.

Figure 4.3 depicts the collision detection time of the simulation over a short period of time. Figure 4.4 shows that the number of collision independent set corresponding to the same time period in Figure 4.3.

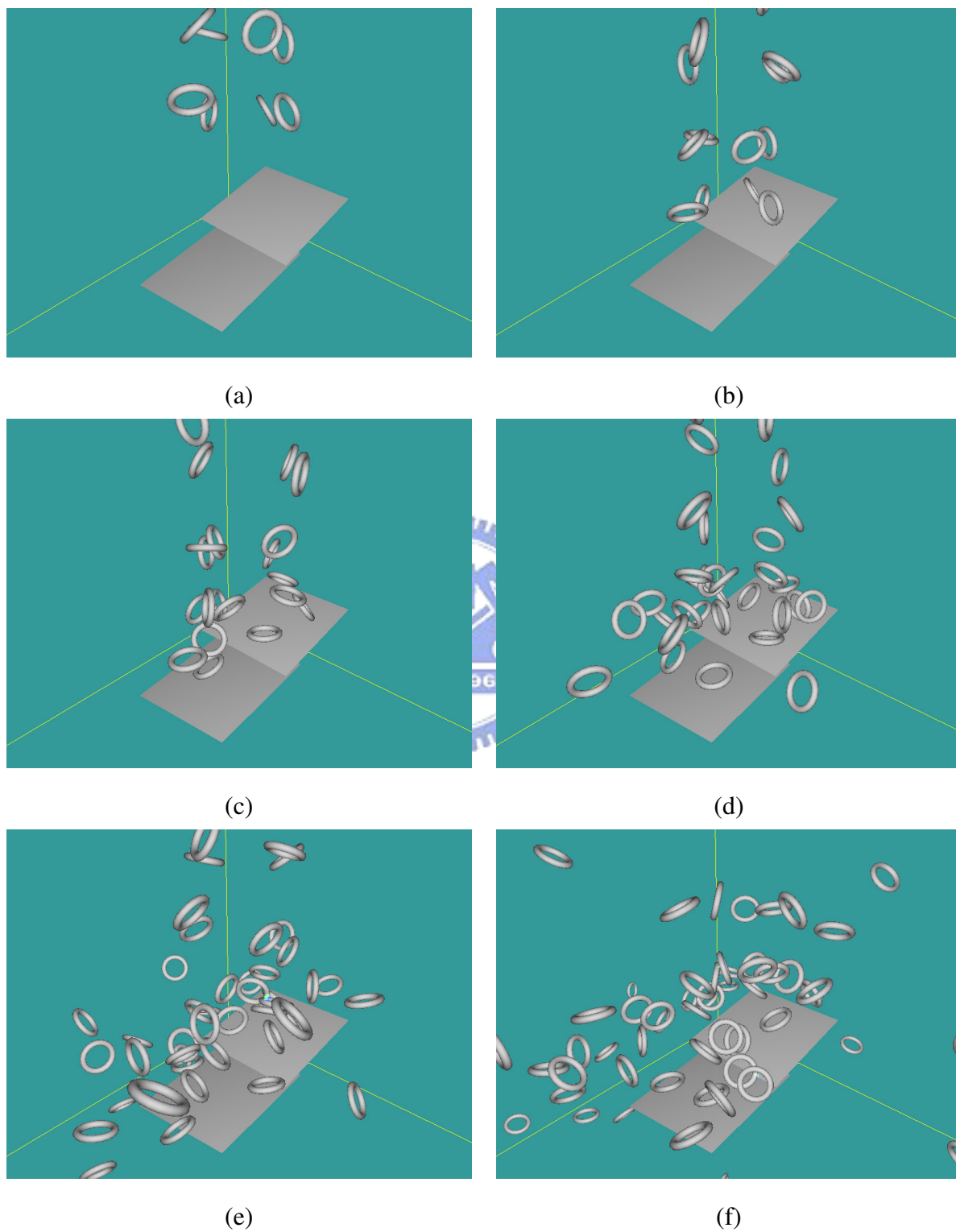
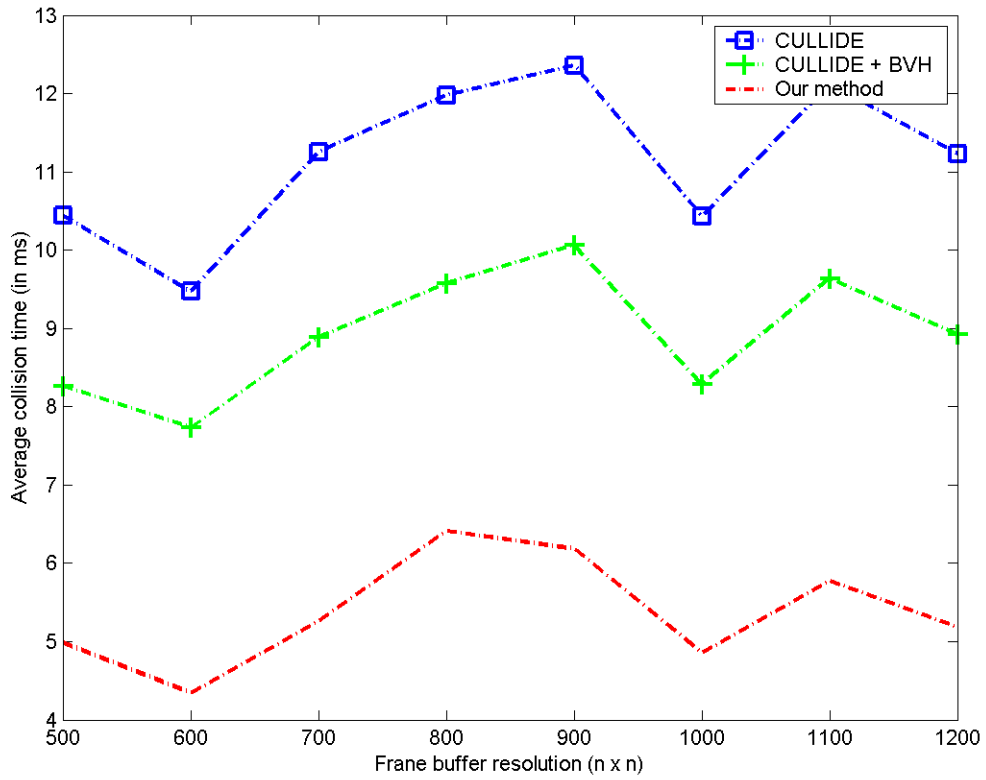
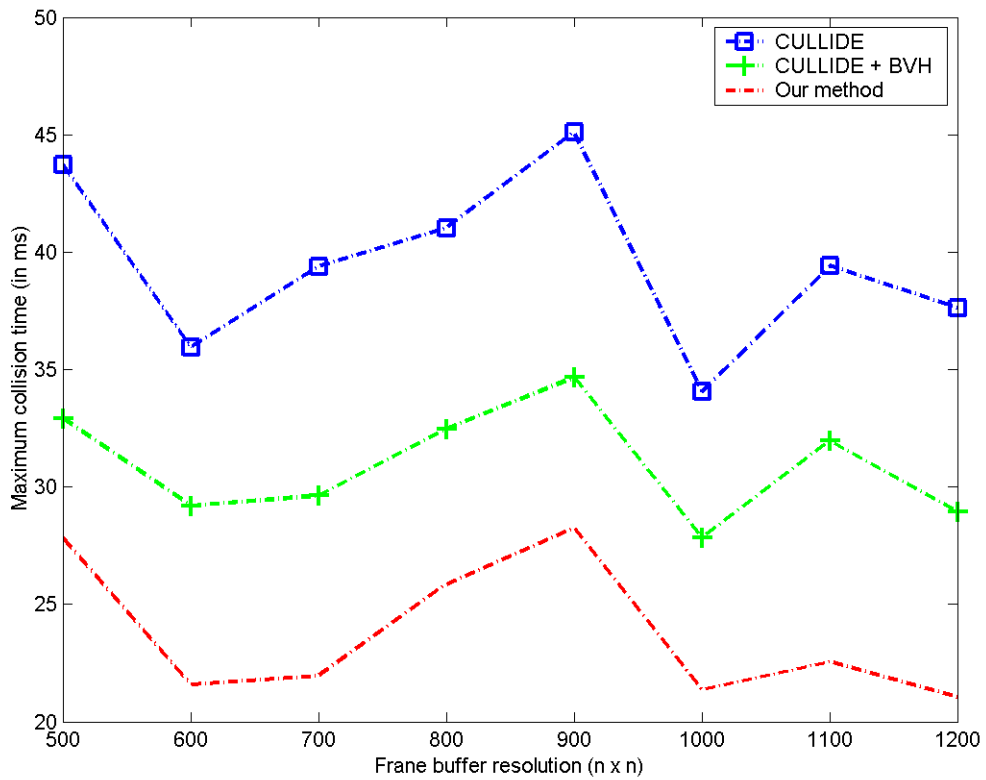


Figure 4.1: Figures (a)-(f) show the simulating process in environment 1.



(a) Average collision time.



(b) Maximum collision time.

Figure 4.2: Performance comparison between our approach and CULLIDE in environment

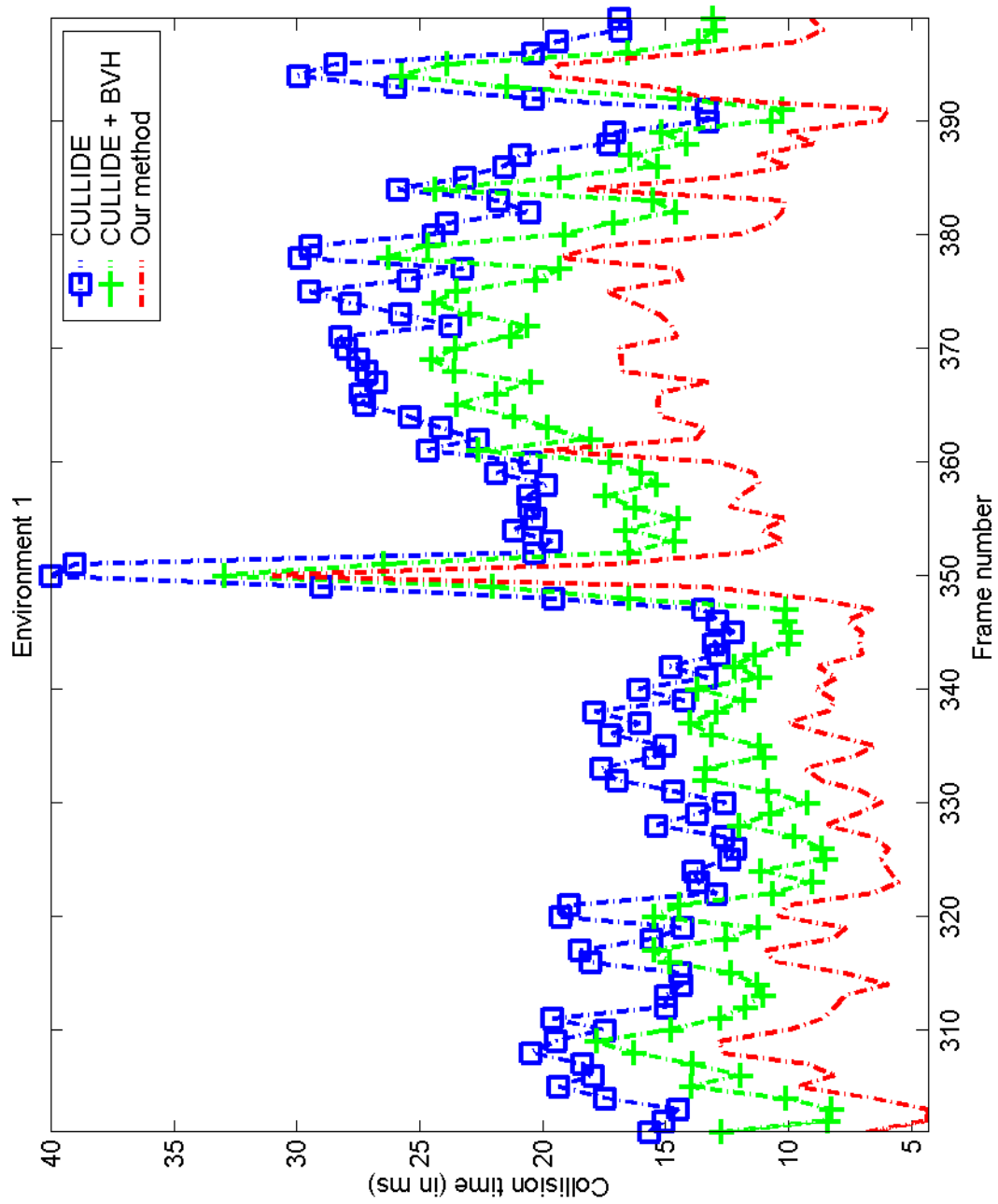


Figure 4.3: Performance comparison with CULLIDE, CULLIDE with hierarchical bounding volume, and our approach in environment 1 at frame buffer resolution 800×800 .

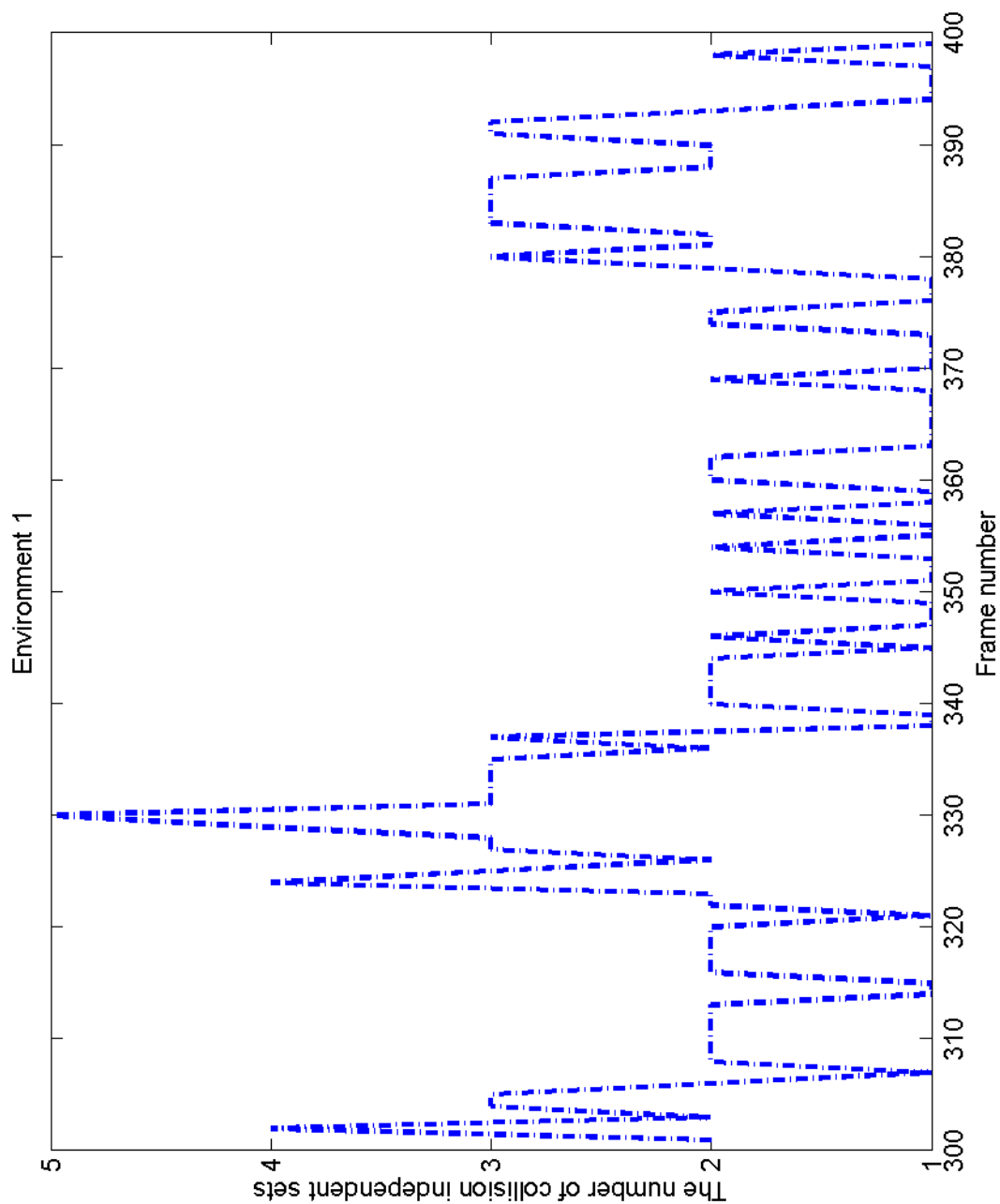


Figure 4.4: The number of collision independent sets constructed for environment 1 at frame buffer resolution 800×800 .

Resolution	500	600	700	800	900	1000	1100	1200
CULLIDE	10.4387	9.4688	11.2566	11.9801	12.3607	10.4257	12.1255	11.2271
CULLIDE + BVH	8.2661	7.7312	8.8909	9.5789	10.0679	8.2891	9.6388	8.927
Our approach	4.984	4.3461	5.258	6.4132	6.1873	4.8549	5.7734	5.186

(a)The average collision time (in ms).

Resolution	500	600	700	800	900	1000	1100	1200
CULLIDE	43.6992	35.9532	39.3979	41.0256	45.1085	34.0662	39.4205	37.6131
CULLIDE + BVH	32.9125	29.1949	29.6336	32.4783	34.6623	27.8242	31.967	28.9488
Our approach	27.8145	21.5791	21.9499	25.8569	28.2488	21.3527	22.5531	21.0549

(b)The maximum collision time (in ms).

Table 4.1: Timing statistics for environment 1.

4.2 Performance comparison for environments of high geometry complexity and low depth complexity

Environment 2 is a scene of high geometry complexity that consists of 12 dragons and two planes. Each dragon and plane are composed of 47794 and 1800 triangles, respectively. Environment setting is the same as environment 1, all the dragons are affected with gravity, and two planes are fixed. Collision responses between dragons and between dragons and the fixed planes are decided by the physics engine. Some of simulation process is shown in Figure 4.5. Table 4.2 shows the timing statistics of simulation at frame buffer resolutions of 500-1200. From Figure 4.6, we can see that there is a large difference on collision detection time between CULLIDE-w/o-BVH and CULLIDE with BVH. The maximum collision time of our approach is slower than CULLIDE with BVH at some resolutions, because the number of collision independent set is 1 and the construction of collision independent set brings additional overhead. However, the average performance of our approach is faster than CULLIDE with BVH.

From the simulation result, we observe that hierarchical bounding volumes can actually improve the performance. Figures 4.7 and 4.8 show the performance of our method and CULLIDE with BVH, we can see that the difference of performance is reduced. It suggests that hierarchical bounding volume is advantageous to the scene of high geometry complexity since the times of visibility queries will be reduced substantially by using the bounding volume hierarchy. In Figure 4.9, we observe that only a few collision independent sets are

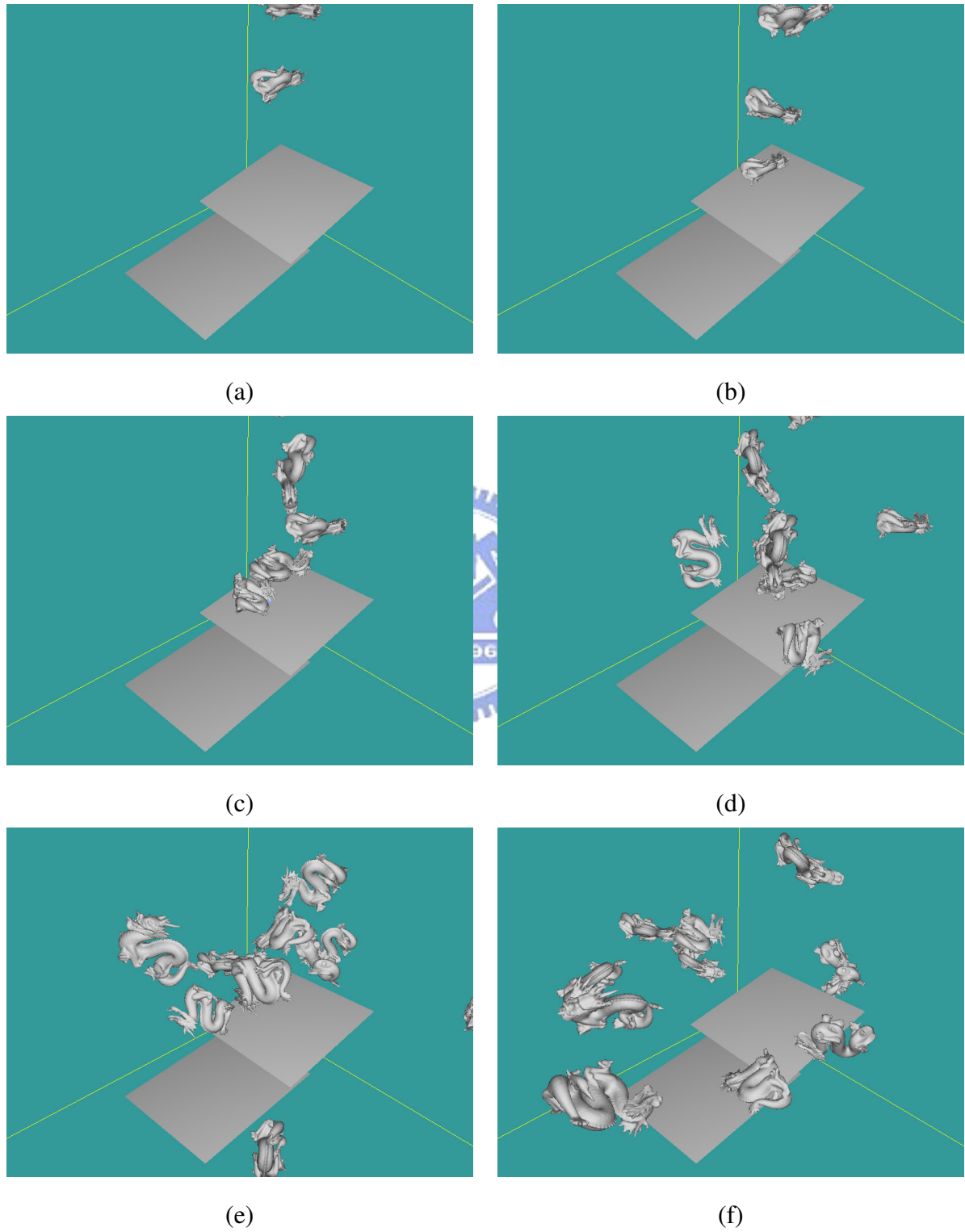
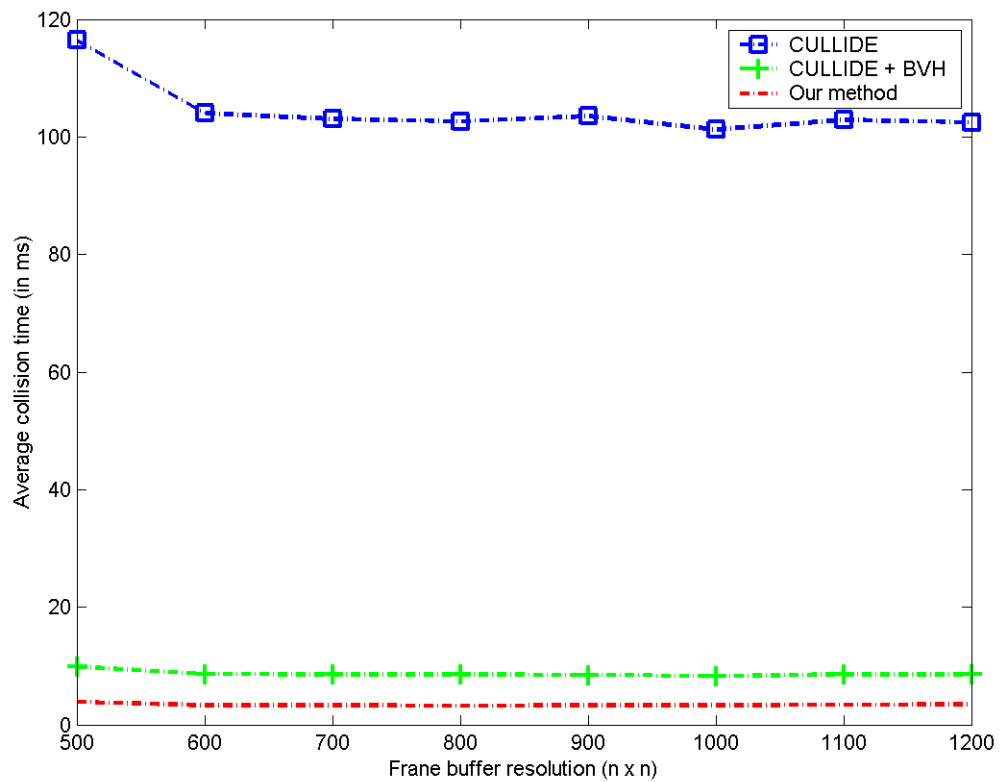
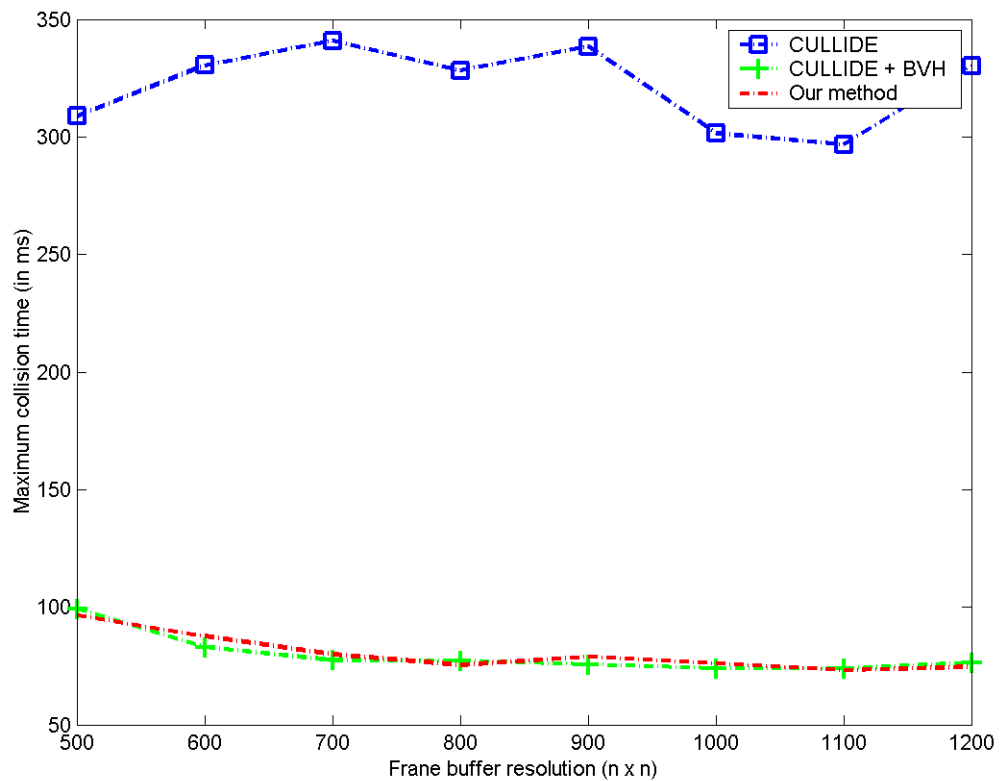


Figure 4.5: Figures (a)-(f) show the simulating process in environment 2.



(a) Average collision time.



(b) Maximum collision time.

Figure 4.6: Performance comparison between our approach and CULLIDE in environment

Resolution	500	600	700	800	900	1000	1100	1200
CULLIDE	116.4508	104.0282	103.0554	103.5537	102.5925	101.2312	102.8491	102.5063
CULLIDE + BVH	9.8627	8.6619	8.5765	8.5069	8.4425	8.3264	8.5565	8.5093
Our approach	3.8625	3.3018	3.2927	3.2294	3.3413	3.3237	3.4301	3.472

(a)The average collision time (in ms).

Resolution	500	600	700	800	900	1000	1100	1200
CULLIDE	308.8009	330.4723	340.8981	328.2975	338.5292	301.5541	296.7655	330.0577
CULLIDE + BVH	99.2663	82.9217	77.5231	77.2811	75.631	74.0776	74.105	76.3502
Our approach	96.4205	87.6988	79.9788	75.535	78.8733	76.117	73.2332	74.6011

(b)The maximum collision time (in ms).

Table 4.2: Timing statistics for environment 2.

found since there are fewer objects in the scene.

4.3 Performance comparison for environments of high depth complexity and low geometry complexity

Environment 3 consists of varying number of torii, each of them has 800 triangles. We record the time of collision detection performed at frame buffer resolution 800×800 . Each torus moves randomly in a cube with an initial velocity, but without gravity, as shown in Figure 4.10. This scene has higher depth complexity than previous two environments. Table 4.3 shows the timing statistics of simulation results for 50-200 torii. Our approach is 3.5-5.6 times faster than CULLIDE-w/o-BVH at the average collision detection time, as shown in Figure 4.11.

From the simulation result, we observe that depth sorting and collision independent sets construction are the major sources of performance improvement when the depth complexity of the scene is high. Figure 4.12 depicts the simulation over a period of time, and Figure 4.13 shows that the number of collision independent set corresponding to the same time period in Figure 4.12. The restriction of the depth complexity will be conspicuously reduced by our collision independent set approach, which will decrease the cost of exact collision detection and times of occlusion queries and rendering. From the experiments, we can observe that hierarchical bounding volume have better performance on dealing with scenes having high geometry complexity, and sorting and collision independent set construction

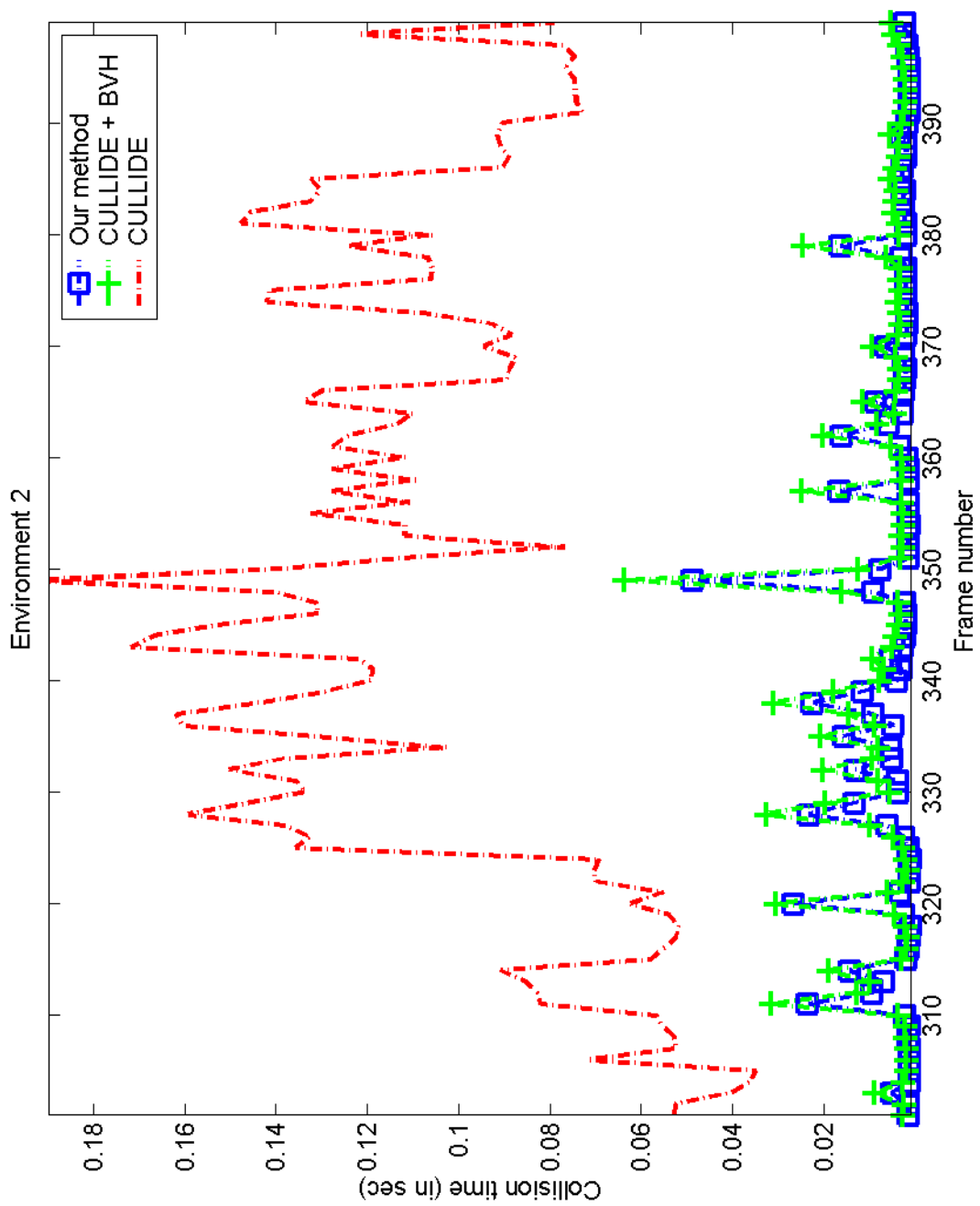


Figure 4.7: Performance comparison with CULLIDE, CULLIDE with hierarchical bounding volume, and our approach in environment 2 at frame buffer resolution 800×800 .

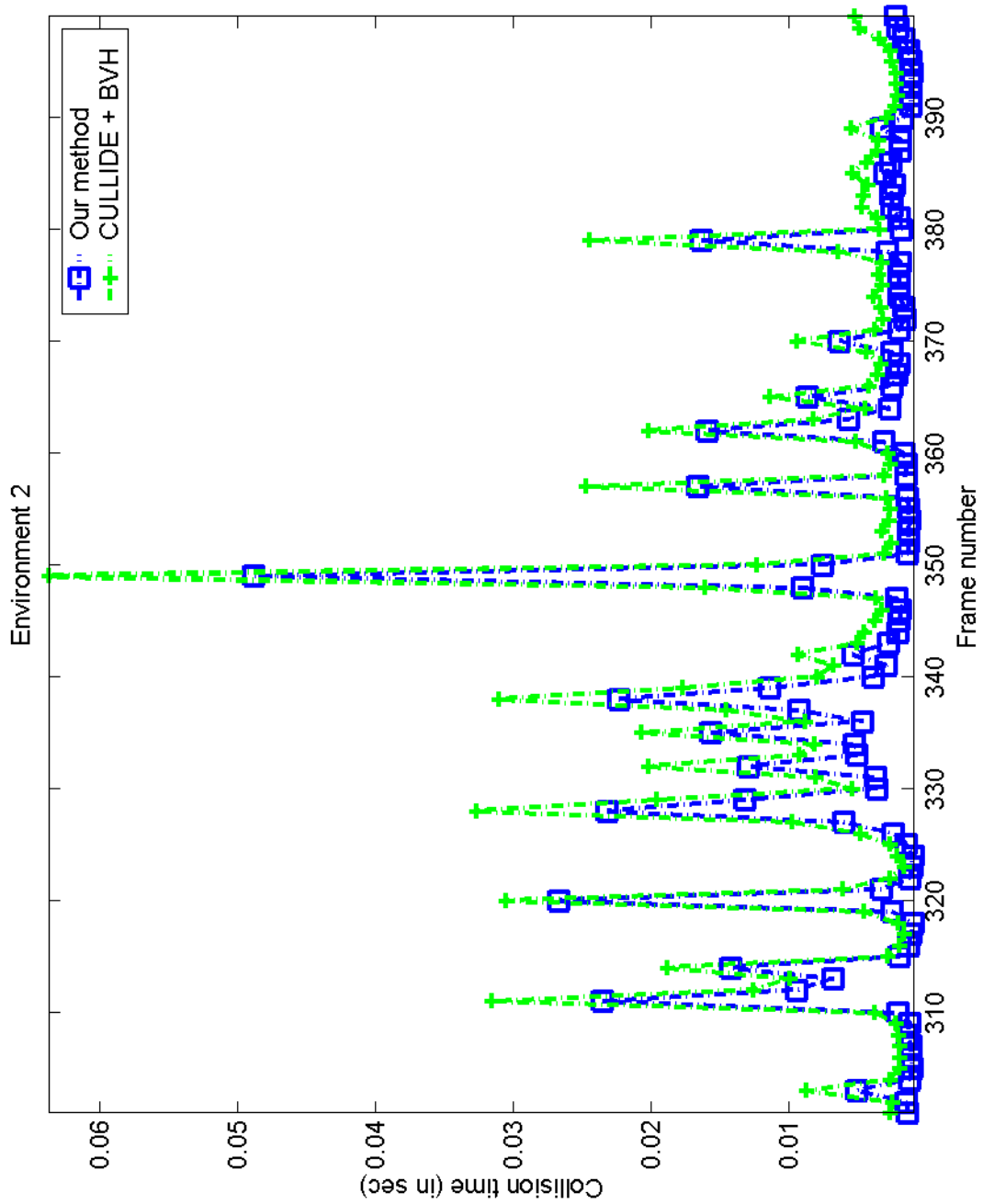


Figure 4.8: Performance comparison between CULLIDE with hierarcial bounding volume and our approach in environment 2 at frame buffer resolution 800×800 .

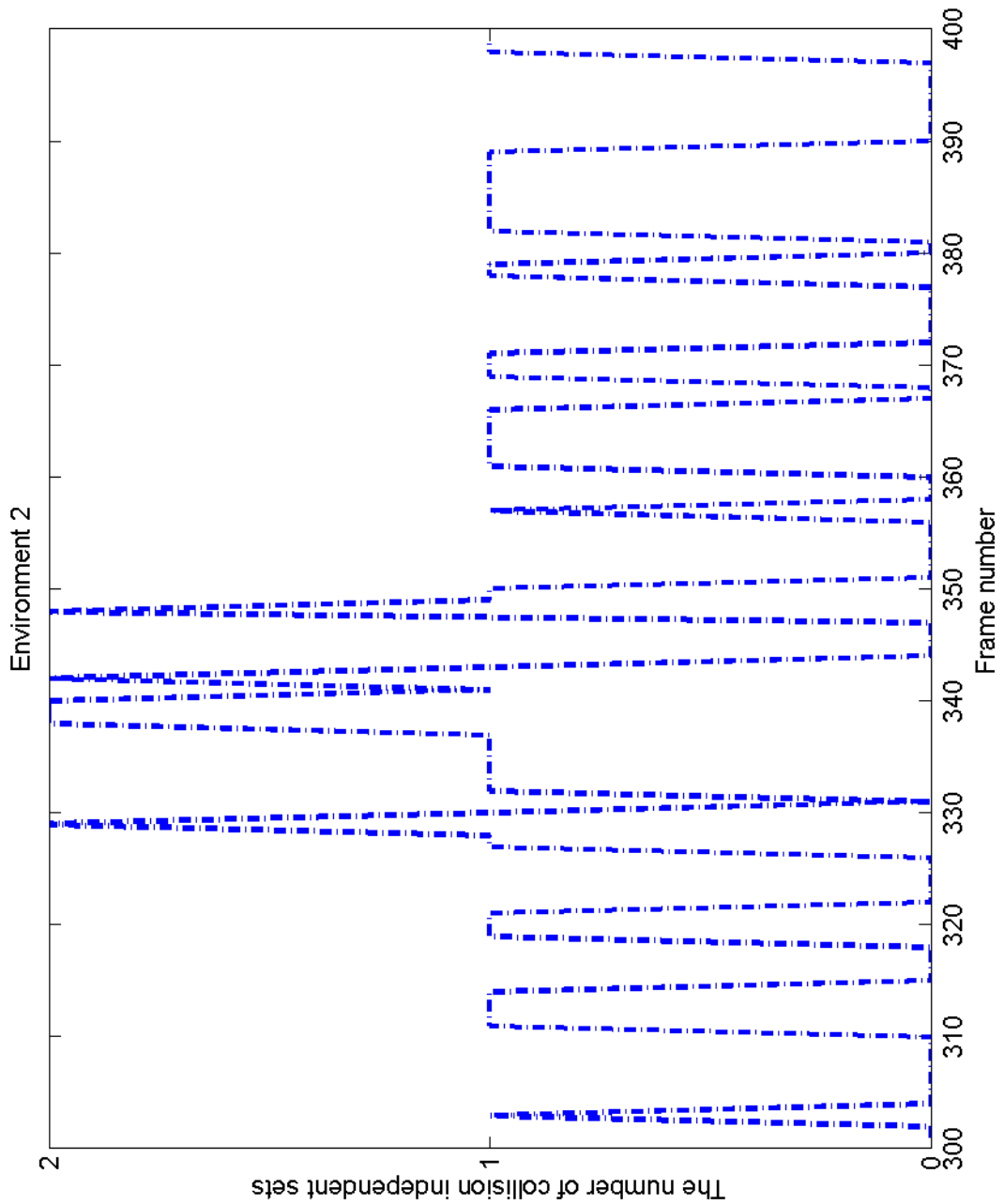


Figure 4.9: The number of collision independent constructed for in environment 2 at frame buffer resolution 800×800 .

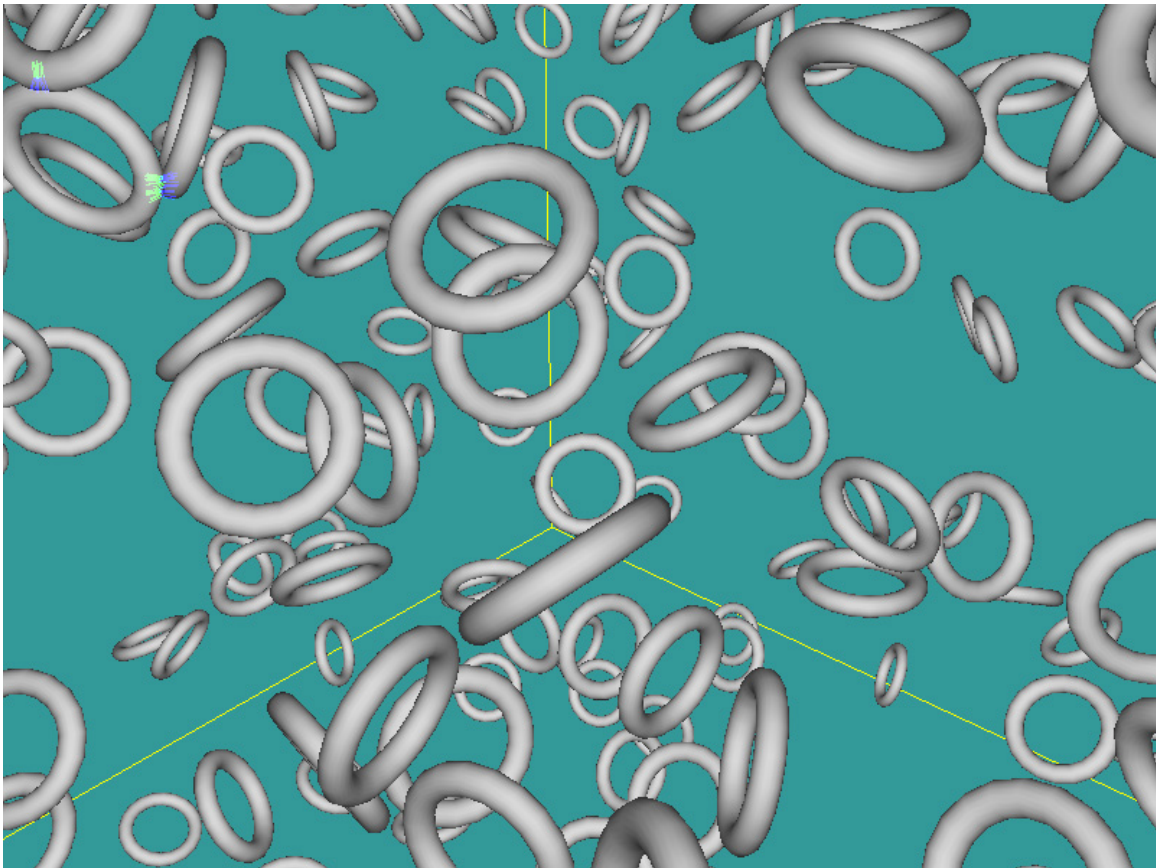


Figure 4.10: Figure shows the simulating process in environment 3.

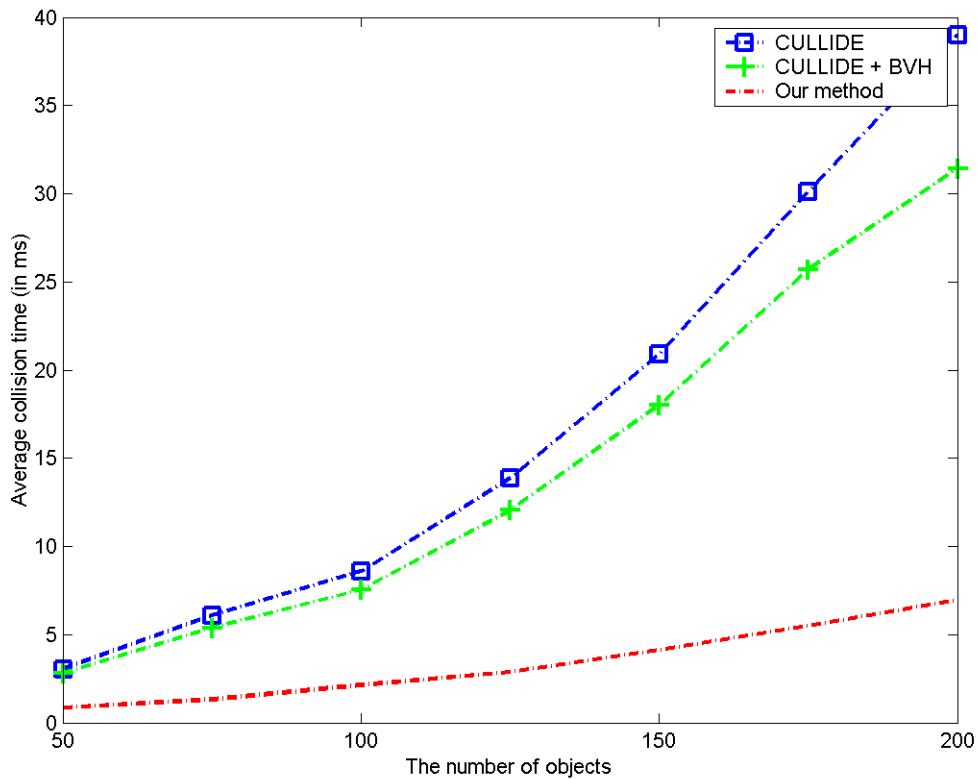
Number of objects	50	75	100	125	150	175	200
CULLIDE	3.0452	6.0724	8.5626	13.8683	20.8768	30.1283	39.0184
CULLIDE + BVH	2.8114	5.3933	7.5438	12.0604	18.0058	25.6932	31.4438
Our approach	0.8566	1.3087	2.1369	2.8623	4.1652	5.4974	6.9615

(a)The average collision time (in ms).

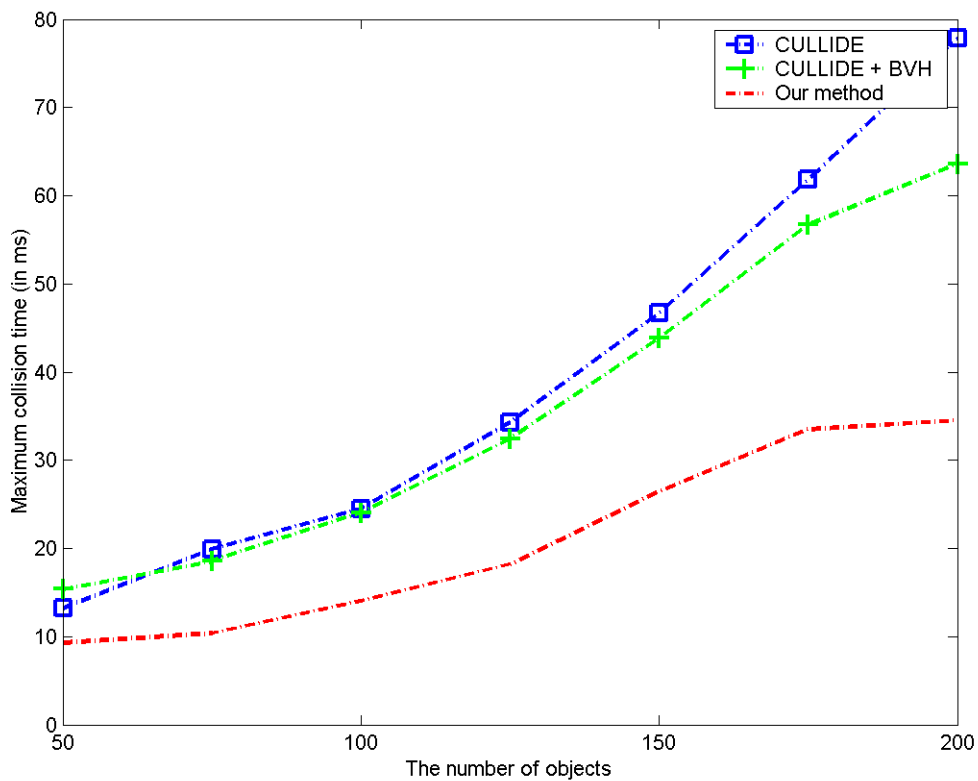
Number of objects	50	75	100	125	150	175	200
CULLIDE	13.2077	19.8889	24.4985	34.3188	46.6946	61.8026	77.8568
CULLIDE + BVH	15.4034	18.5633	24.0479	32.4148	43.821	56.7133	63.6322
Our approach	9.262	10.3844	14.0316	18.2389	26.452	33.5028	34.5121

(b)The maximum collision time (in ms).

Table 4.3: Timing statistics for environment 3.



(a) Average collision time.



(b) Maximum collision time.

Figure 4.11: Performance comparison between our approach and CULLIDE in environment 3.

can reduce the impact of high depth complexity.



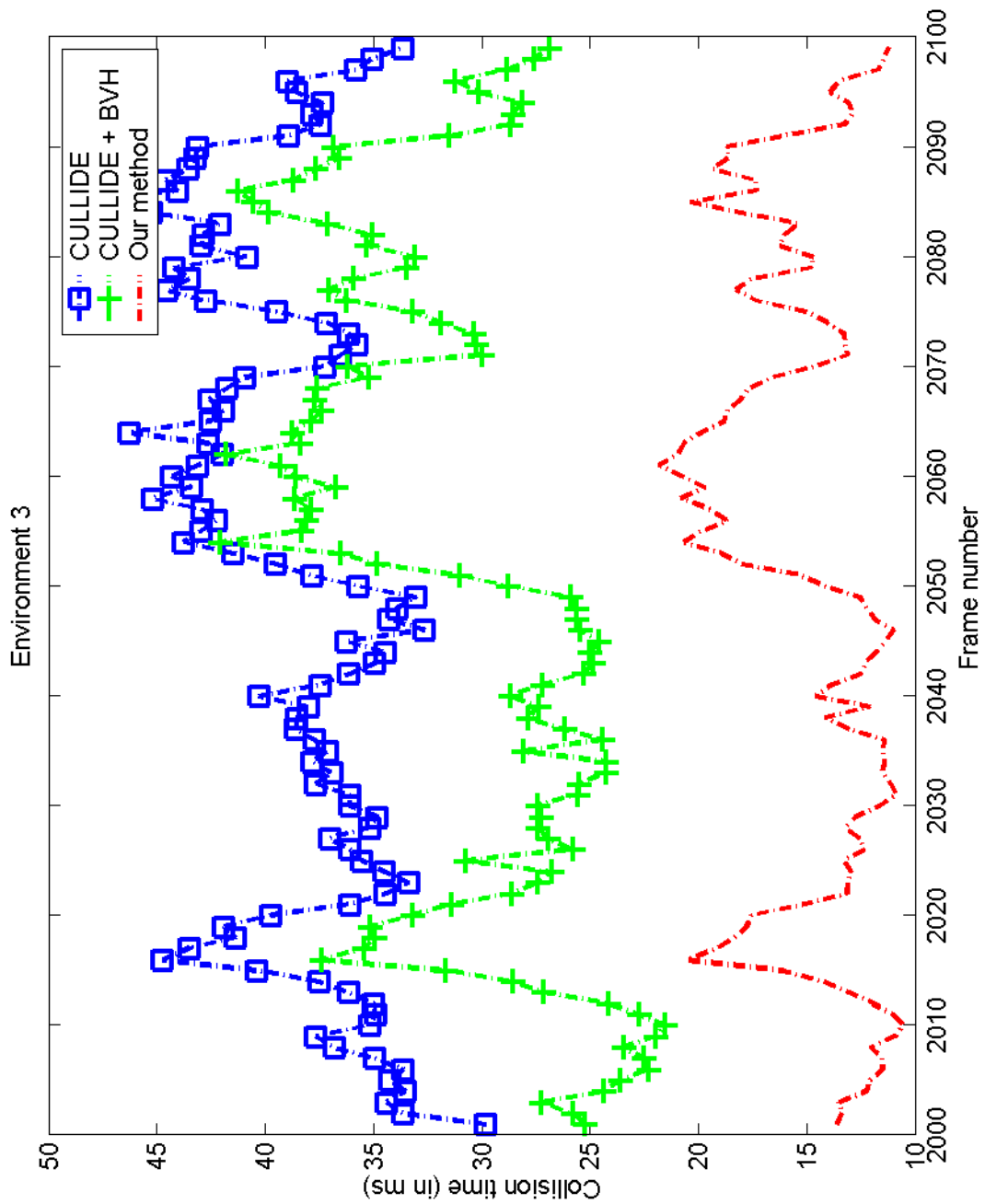


Figure 4.12: Performance comparison with CULLIDE, CULLIDE with hierarchical bounding volume, and our approach in environment 3 for 200 torii at frame buffer resolution 800×800 .

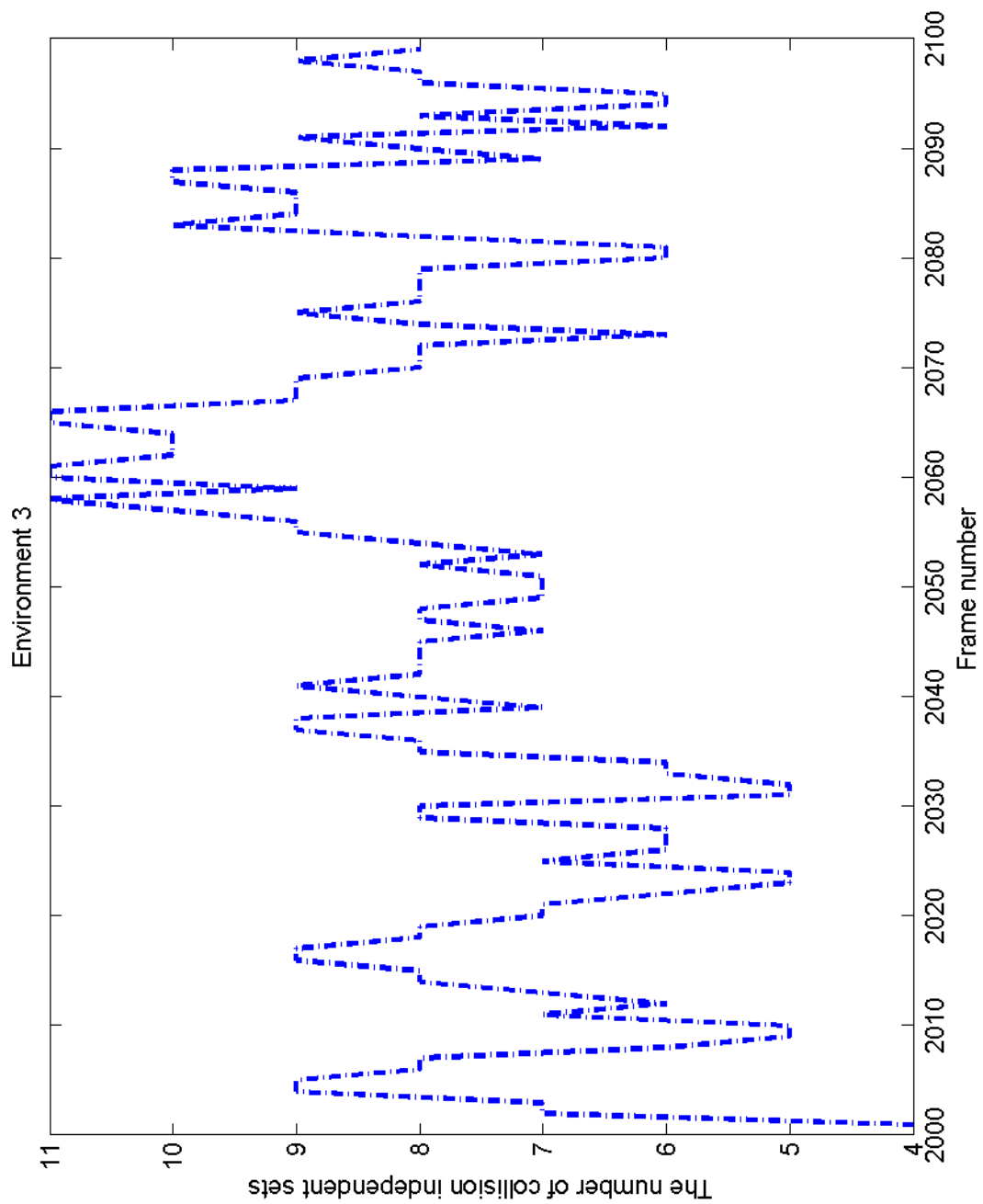


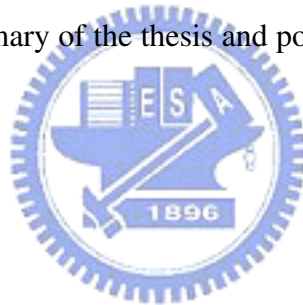
Figure 4.13: The number of collision independent sets constructed for environment 3 for 200 torii at frame buffer resolution 800×800 .

CHAPTER 5

Conclusion

In this Chapter, we give a summary of the thesis and point out some future research directions.

5.1 Summary



Collision detection is a fundamental problem in interactive application. However, no general collision detection algorithms are presented for all kinds of environments. For this reason, the collision detection algorithms are designed by using some coherences among objects. The coherences for collision detection can be classified into three categories such as spatial coherence, temporal coherence, and image coherence.

There are many image-based collision detection techniques are proposed due to the development of graphics hardware grows up rapidly. In this thesis, we present an algorithm for efficient and exact collision detection between complex geometric models in large environments using graphics hardware. Our approach is based on CULLIDE. It is constructed in several major includes construction of bounding volume hierarchies, projection and sorting, hierarchical pruning, and exact intersection test. In preprocessing, we construct bounding volume hierarchies for each object using oriented bounding boxes. These extra data structure will reduce the times of visibility queries substantially. In order to amend the performance of visibility pruning technique, we decide a suitable rendering order for collision culling. In this way, we can conspicuously reduce the extent of culling

to rely on the depth complexity of the scene along the view direction by using near-to-far rendering order in two-pass process. Furthermore, we design two lemmas for determining whether separating surfaces exist between the subsets of the potentially colliding set. Collision independent sets construction will reduce the cost of exact intersection test and times of occlusion queries and rendering.

The main contributions of our approach are that we propose a more efficient collision culling algorithm and partitioning potentially colliding set during collision culling. From simulation results, our approach is much faster than CULLIDE. It dose still not perform frame buffer readbacks but readbacks a few bytes per object. The polygon level intersections will be detected in interactive rate. Our collision detection approach is applicable to both high geometry complexity and depth complexity scenes. It is not restricted to closed and watertight models.

5.2 Future work

An approach uses temporal coherence in addition to a "sweep-and-prune" technique to reduce the pairs of objects that need to be considered for collision [CLMP95]. The hierarchical bounding volumes are updated fast by exploiting temporal coherence for deformable models [JP04]. We could also assume that the configurations of collision independent sets are changed very restrictedly. Collision independent sets could be constructed by using the results of successive time step.

We would like to extend our algorithm to perform other collision queries such as self-intersections, continuous collisions, volumetric intersection, and penetrating computation. An approach based on CULLIDE is proposed to detect self-collisions [BM04]. It is also restricted by depth complexity. At this point, we could apply our algorithm to its framework, and attempt to traverse nodes in hierarchy in a far-to-near order.

Bibliography

- [BM04] N. Boldt and J. Meyer. Self-intersections with Cullide. 23(3), 2004.
- [BWS99] G. Baciu, W.S. Wong, and H. Sun. Recode: an image-based collision detection algorithm. *The Journal of Visualization and Computer Animation*, 10(4):181–192, October - December 1999.
- [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.
- [Eri05] C. Ericson. *Real Time Collision Detection*. Morgan Kaufmann, 2005.
- [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.
- [GLM04] N. Govindaraju, M. Lin, and D. Manocha. Fast and Reliable Collision Culling using Graphics Hardware. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 2–9, 2004.
- [GLM05a] N. Govindaraju, M. Lin, and D. Manocha. Interactive Collision Detection between Deformable Models using Chromatic Decomposition. 24(3):991–999, July 2005.
- [GLM05b] N. Govindaraju, M. Lin, and D. Manocha. Quick-CULLIDE: Fast Inter- and

- Intra-Object Collision Culling Using Graphics Hardware. In *IEEE Virtual Reality Conference 2005 (VR'05)*, pages 59–66, 319, 2005.
- [GRLM03] N. Govindaraju, S. Redon, M. Lin, and D. Manocha. CULLIDE: Interactive Collision Detection Between Complex Models in Large Environments using Graphics Hardware. In *Proceedings of the Eurographics/SIGGRAPH Graphics Hardware Workshop*, 2003.
- [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.
- [HTG03] B. Heidelberger, M. Teschner, and M. Gross. Real-Time Volumetric Intersections of Deforming Objects. In *Proc. Vision, Modeling, Visualization VMV'03*, pages 461–468, 2003.
- [HTG04] B. Heidelberger, M. Teschner, and M. Gross. Detection of Collision and Self-Collisions using Image Space Techniques. In *Proc. Computer Graphics, Visualization and Computer Vision WSCG'04*, pages 145–152, 2004.
- [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.

- [JP04] D. JAMES and D. PAI. BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3), Aug. 2004.
- [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.
- [KP03] D. Knott and D.K. Pai. CInDeR: Collision and Interference Detection in Real-Time using Graphics Hardware. In *Proc. of Graphics Interface*, pages 73–80, 2003.
- [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.
- [KZ05] L. Kavan and J. ZARA. Fast Collision Detection for Skeletally Deformable Models Computer Graphics Forum. In *Proc. of Graphics Interface*, volume 24, pages 363–37, 2005.
- [LAM01] T. Larsson and T. Akenine-Mller. Collision detection for continuously deforming bodies. In *In Eurographics 2001*, pages 325–333, 2001.
- [LCN99] J.C. Lombardo, M.P. Cani, and F. Neyret. Real-Time Collision Detection for Virtual Surgery. In *Proceedings of Computer Animation '99*, pages 82–, 1999.
- [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.
- [MOK95] K. Myszkowski, O.G. Okunev, and T.L. Kunii. Fast Collision Detection between Complex Solids using Rasterizing Graphics Hardware. 11(9):497–512, 1995.

- [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.
- [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.
- [van97] G. van den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools: JGT*, 2(4):1–14, 1997.
- [VSC01] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast Cloth Animation on Walking Avatars. 20(3):260–267, Sept. 2001.
- [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.