Du P, Zhao JY, Pan WB *et al.* GPU accelerated real-time collision handling in virtual disassembly. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(3): 511–518 May 2015. DOI 10.1007/s11390-015-1541-2

GPU Accelerated Real-Time Collision Handling in Virtual Disassembly

Peng Du[1] (杜 鹏), Jie-Yi Zhao[2] (赵杰伊), Wan-Bin Pan[1] (潘万彬), and Yi-Gang Wang[1] (王毅刚), *Member, CCF, ACM*

[1]*School of Media & Design, Hangzhou Dianzi University, Hangzhou 310018, China*
[2]*College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*

E-mail: {dp, su27}@zju.edu.cn; {panwanbin, yigang.wang}@hdu.edu.cn

Received November 25, 2014; revised February 18, 2015.

Abstract Previous collision detection methods for virtual disassembly mainly detect collisions at discrete time intervals, and use oriented bounding boxes to speed up the process. However, these discrete methods cannot guarantee no penetration occurs when the components move. Meanwhile, because some of the components are embedded into each other, these components cannot be separated in the subsequent process. To solve these problems, we propose an approach for real-time collision handling by utilizing the computational power of modern GPUs. First we present a novel GPU-based collision handling framework for virtual disassembly. Second we use a collision-streams based continuous collision detection to guarantee no collision missed. Finally we introduce a triangle intersection detection algorithm to solve the problem that collision cannot be detected when the components are embedded into each other at the initial configuration. The experimental results show that our method can improve the overall performance of collision detection and achieve real-time simulation.

Keywords GPU-based virtual disassembly, continuous collision detection, discrete collision detection, collision handling

1 Introduction

Virtual disassembly process allows us to decompose the assembly into parts, while its inverse process makes the parts reassembled again. The virtual disassembly process can train users without destructing the actual assembly[1]. Virtual disassembly is becoming more important due to the environmental and economic pressures for industrial recycling and reusing.

One can design an interactive virtual disassembly system by employing a 3D projector to make users immersed in the virtual environment. The system then allows users to dismount the assembly with interactive devices such as head sensors and 3D mouses. The disassembly system can also guide users to disassemble the 3D components according to the computed disassembly sequence, as shown in Fig.1. Additionally, we can record the disassembly path and then automatically install the components into the original assembled

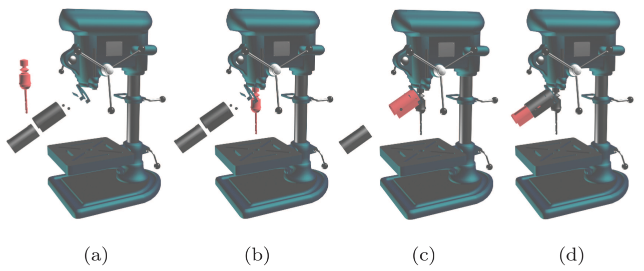


Fig.1. Microscope. (a) Disassembly result. (b)∼(d) Assembly process. Our system can record the disassembly path and then automatically install the components by taking the reverse operations to the disassembly path.

configurations (Fig.1(d)) by following its reversed path.

To perform the virtual disassembly, collision detection and response between different components is an important task. Previous collision detection methods for virtual disassembly mainly detect collisions at discrete time intervals[2], and use oriented bounding

Regular Paper
Special Section on Computational Visual Media
This work was supported by the National Natural Science Foundation of China under Grant No. 61472111, the Zhejiang Provincial Natural Science Foundation of China under Grant No. LQ13F020016, the Foundation of Zhejiang Educational Committee under Grant No. Y201224034, and the Scientific Research Start Foundation of Hangzhou Dianzi University under Grant No. KYS225613032.

boxes (OBBs)[3] to accelerate the virtual process. Unfortunately, these discrete methods cannot guarantee no penetration among different components as they move. Furthermore, in practice, some components are intersected at the initial configuration, but the users think there is no intersection at the initial configuration. Therefore, collision detection cannot get the consistent configuration as the users expected, and these components cannot be separated in the subsequent process.

To solve these problems, we present a hybrid collision detection approach. Firstly we present a GPU-based continuous collision detection (CCD) to guarantee no collision missed. Secondly we present a triangle intersection detection (TID) algorithm to solve the embedded problem. The contributions of the paper can be summarized as follows.

• We present a novel GPU-based collision handling framework for virtual disassembly. This framework can guarantee no collision missed by utilizing the capacities of CCD.

• We present an improved collision-streams based CCD method which can improve the overall performance of collision detection, and achieve real-time simulation.

• We integrate TID into our parallel CCD system to solve the problem that collision cannot be detected when the components are embedded into each other at the initial configuration.

2 Related Work

In this section, we discuss related work on virtual disassembly and GPU-based collision detection algorithms.

2.1 Virtual Disassembly

The virtual disassembly process on the design of a product is an essential aspect for recycling, maintenance, repair, and component/material reusing in the sustainable development.

For simulating efficient disassembly process, Popescu and Lacob proposed a method of automatically generating dismounting sequences[4]. In this method, the valid disassembly trajectories of a component relative to its surrounding ones are modeled based on the mobility operator, and the swept volume approach is used to evaluate collisions between components. Finally, the disassembly sequences are determined through an improved layer peeling technique. For designing non-destructive disassembly path, Pomares *et al.* proposed an object-oriented model required for developing the disassembly process in a flexible way[5]. The authors used local and global strategies for the removal of one component based on the features of the components.

2.2 Continuous Collision Detection

Collision detection methods can be classified into discrete collision detection (DCD) and continuous collision detection (CCD). DCD[3,6-11] can check collisions only at discrete time steps, and may result in collision misses as the models undergo a fast, continuous movement, whereas CCD is used to compute the first-time-of-contact between deforming triangles along continuous trajectories. In most cases, we use linear interpolation to depict the trajectories of the triangle vertices for improving the efficiency of collision detection. The CCD test between two deforming triangles reduces to performing nine vertex-face (VF) and six edge-edge (EE) elementary tests, where each elementary test is reduced to find the roots of a cubic equation.

The research on CCD can be mainly divided into two aspects: high-level culling[12-16] and low-level culling[17-22]. High-level culling, especially bounding volume hierarchies (BVHs), performs triangle-level overlap tests to reduce the potential colliding pair-wise set (PCS). Some widely used high-level culling methods include continuous normal cone tests[13], star-contour tests[14], and subspace min-norm certification tests[15]. Low-level culling is to remove redundant primitive pairs (VF or EE pair) and reduce elementary tests from PCS. Recently, representative triangles[22] and orphan sets[13] have been proposed to remove redundant primitive pairs. Continuous separating axis theorem[17], deforming non-penetration filter[18] and parallel filter in subspace[20], was proposed to reduce a large number of false positives.

2.3 GPU-Based Collision Detection

Current GPUs are regarded as high-throughput processors, which operate on an SIMD (single-instruction multiple data) basis, and the computations are performed simultaneously by executing a large number of threads. On current GPUs, the multiple threads are executed by the kernels which are organized into a two-level hierarchy: blocks and threads. At the top level of the hierarchy, a grid is organized as a two-dimensional

(2D) array of blocks. At the bottom level, all blocks of a grid are organized into an array of threads. All the threads in the same block can access a small, high-speed shared memory.

GPU can accelerate geometry computation[23-24]. Recently, some algorithms of collision detection exploiting the parallel computing capability of modern many-core GPUs have been designed[25-29]. For example, Tang *et al.* presented a parallel continuous collision detection method based on a collision-streams based data-level parallelism and a front-based task-level parallelism, by utilizing the capacities of GPUs[29]. Additionally, Zhang and Kim evenly partitioned the front to make load balance among multiple cores[25]. Pabst *et al.* presented an adaptive space grid based parallel collision detection method[28]. Wong *et al.* used the advantage of octree grid to address the issue of performance reduction caused by uneven model subdivision[26].

3 Motivation and Overview

3.1 Motivation

When designing a real-time virtual disassembly system which allows users to manipulate different components with the consideration of collision, we have to ensure collision detection and response can achieve 30 or more frames per second. To achieve the requirement, previous virtual disassembly systems mainly use OBBs instead of actual components to check collision.

However, there are some drawbacks in the method. First, as we abstract the components with OBBs, the details such as holes and irregular corners will be erased and the simulation effect cannot satisfy the visual requirement. Second, penetration may occur as users move the components. Third, some of the components are designed to be embedded into each other at the initial configuration. Therefore, the intersected components cannot be removed from the assembly with previous collision detection methods.

To solve these problems, we present an approach of hybrid collision detection. First we present an improved collision-streams based continuous collision detection (CCD) to guarantee no collision missed, and second we present a triangle intersection detection (TID) method to solve the embedded problem.

3.2 System Pipeline

Our virtual disassembly system has the following parts: components removing, collision culling, and exact collision detection and collision response, which can be summarized as Fig.2. The exact detection also can be divided into two phases in terms of whether intersection occurs: CCD and TID. For large assembly with millions of primitives, first we employ QSlim library to make some simplifications for real-time requirement.

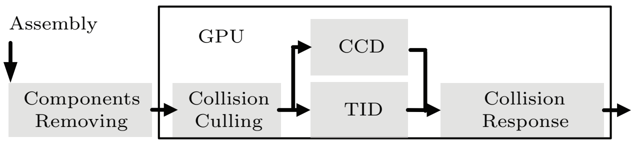


Fig.2. Overview of the virtual disassembly system. The pathway and components of the system: components removing, collision culling, and exact collision detection and collision response.

4 Parallel Collision Detection

First our algorithm builds a BVH for the assembly, then builds an intra-collision detection tree and performs a top-down traversal of the tree to check for collision between different components. In terms of CCD computation, the algorithm assumes linearly interpolating motion between the vertices and the first-time-of-contact is computed by performing elementary tests between the features.

4.1 Algorithm Overview

Initially, we construct the following streams on GPU: vertex stream S_{v}, intersection stream S_{i} to get the pairs of intersected components, BV stream S_{bv} to save the bounding volume of each triangle, BVH stream S_{bvh} to save the bounding volume of each intermediate node in BVH, front stream S_{f} to save the nodes of the bounding volume traversal tree (BVTT), and triangle pair stream S_{t} to save the non-adjacent triangle pairs whose bounding volumes overlap. And we construct transformation matrix stream $\boldsymbol{S}_{\mathrm{tm}}$ to save the transformation matrices of the components on CPU. Then in each simulation time step, first we use high-level culling method to cull the false positives. For the unsettled triangle pairs, if it belongs to S_{i}, we use TID to test intersection, otherwise we use CCD.

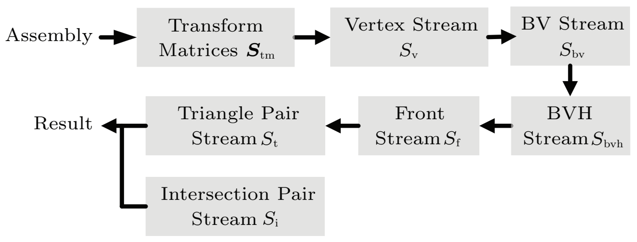
In addition, if the intersected components are not separated, we allow the components to move freely. Once the collision pair is separated, we remove the component pair from S_{i} and use CCD to guarantee no collision for this pair in the next frame. If collisions occur, we make the configuration of the assembly return to previous non-collision configuration. For more details, please refer to Algorithm 1.

Algorithm 1. Improved Collision-Streams Based CCD
```
 1: for each simulation time step do
 2:    Send $\boldsymbol{S}_{\rm tm}$ from CPU to GPU
 3:    Update $S_{\rm v}$ with $\boldsymbol{S}_{\rm tm}$
 4:    Update $S_{\rm bv}$ and $S_{\rm bvh}$ with $S_{\rm v}$
 5:    Update $S_{\rm f}$ with $S_{\rm bvh}$ and generate $S_{\rm t}$
 6:    for each triangle pair $t$ in the triangle pair stream $S_{\rm t}$ do
 7:       if the triangles in $t$ belong to the same components
          then
 8:          Continue
 9:       if $t$ in $S_{\rm i}$ is available then
10:          Perform TID for $t$
11:          if intersection occurs for $t$ then
12:             Make the intersection pair available in the next
                time step
13:       else
14:          Perform CCD for $t$
```

4.2 Collision-Streams

In order to fully utilize the powerful parallel computation ability of current GPUs, we improve the collision-streams based CCD[29], in which the authors treat the GPU as a collection of stream processors that can perform large-scale fine-grained parallel computation on stream data. In terms of the overall CCD algorithm, the geometric data are represented as stream data, and the underlying functional modules used in the algorithm (i.e., updating BVHs, front traversal, elementary tests, etc.) are mapped to computation kernels.

We inherit this method and abstract the primary data streams as vertex stream $S_{\rm v}$, BV stream $S_{\rm bv}$, BVH stream $S_{\rm bvh}$, front stream $S_{\rm f}$ and triangle pair stream $S_{\rm t}$. In addition, we design some assistant streams, i.e., intersection stream $S_{\rm i}$ and transformation matrix stream $\boldsymbol{S}_{\rm tm}$. These data streams can be summarized as Fig.3.



Assembly
Transform Matrices $\boldsymbol{S}_{\rm tm}$
Vertex Stream $S_{\rm v}$
BV Stream $S_{\rm bv}$
BVH Stream $S_{\rm bvh}$
Front Stream $S_{\rm f}$
Triangle Pair Stream $S_{\rm t}$
Result
Intersection Pair Stream $S_{\rm i}$

Fig.3. Collision-streams. All the procedures are mapped to a set of computation kernels. By executing these kernels, the streaming CCD algorithm runs entirely on a GPU.

• *Vertex Stream* $S_{\rm v}$. It corresponds to the geometric coordinates of the vertices of the components. In order to perform CCD between two discrete time steps, two vertex streams $S_{{\rm v}_n}$ and $S_{{\rm v}_{n+1}}$ are used to store the vertex coordinates corresponding to time t_n and t_{n+1}, respectively. In addition, we use $S_{{\rm v}_0}$ to represent the initial vertex coordinate stream.

• *BV Stream* $S_{\rm bv}$ *and BVH Stream* $S_{\rm bvh}$. All the bounding volumes for triangles are represented by $S_{\rm bv}$. All of the components are enclosed by a single BVH $S_{\rm bvh}$. $S_{\rm bv}$ and $S_{\rm bvh}$ are updated at each simulation time step based on $S_{\rm v}$.

• *Front Stream* $S_{\rm f}$. At the initial configuration, we perform BVTT traversal to get the non-intersected nodes or the leaf nodes where the tree traversal terminated, and then put them into the front stream $S_{\rm f}$. In the following process of CD, we can traverse $S_{\rm f}$ instead of BVTT to get the intersected triangle pairs.

• *Triangle Pair Stream* $S_{\rm t}$. During $S_{\rm f}$ traversal, all the non-adjacent triangle pairs whose bounding volumes overlap are collected into $S_{\rm t}$.

• *Intersection Stream* $S_{\rm i}$. In the initial stage, we record all the intersected component pairs into the intersection stream $S_{\rm i}$. In the following process, if the intersected components are not separated, we allow the components to move freely. Once they are separated, we remove the component pair from $S_{\rm i}$ and use CCD to guarantee no collision for this pair in the next frame.

• *Transformation Matrices Stream* $\boldsymbol{S}_{\rm tm}$. When the coordinates of components change, we send the transform matrices $\boldsymbol{S}_{\rm tm}$ into GPU, and compute the new vertex stream $S_{{\rm v}_{n+1}}$ with $S_{{\rm v}_0}$ and $\boldsymbol{S}_{\rm tm}$.

4.3 Global-Lock Based Front Tracking

Our collision culling algorithm is based on front tracking[19]. At the initial stage, we construct a front stream from BVTT as shown in Fig.4. As the components move during the simulation, the front needs to be updated correspondingly. In order to maintain a valid front, two operators "sprouting" and "pruning" are used. By using the sprouting operator, a front is sprout to the level where bounding volumes are disjoint or leaf nodes in the tree are reached. The sprouting of each front node is independent of each other. During the tracking of the BVTT front, the nodes that need to be sprouted are flagged as invalid, and their descendants will be traversed in a depth-first manner to check for collisions. This makes it suitable for parallel processing. However, the pruning operator needs to search for sibling nodes in the BVTT front, which is hard for parallel execution. Thus in practice, we do not use any pruning operator since it will affect the parallel performance.

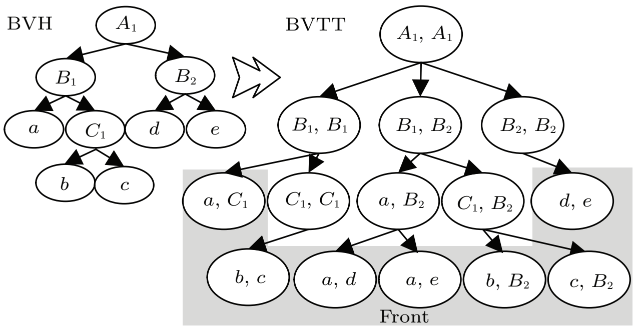


Fig.4. Front construction. In the initial stage, we perform BVTT traversal to get the non-intersected nodes or the leaf nodes where the tree traversal terminates, and then put them into the front.

In each process of CD, we represent the front stream from the previous simulation time step as $S_{\mathrm{f}(t_0)}$. After updating the BVH stream, we can traverse $S_{\mathrm{f}(t_0)}$ to get a new front stream $S_{\mathrm{f}(t_1)}$. First, we allocate each node in $S_{\mathrm{f}(t_0)}$ to a thread. If the two bounding volumes of the node intersect, we will sprout the level until the bounding volumes are disjoint or the leaf nodes in the tree are reached. And then we insert the BVTT nodes into the rear of the front stream $S_{\mathrm{f}(t_1)}$. If the reached node is a leaf node, we insert the corresponding triangle pair into the triangle pair stream. When we insert a new node into the stream, we use atomic operations (e.g., *atomicAdd*() in CUDA or *atom_add*() in OpenCL) to specify the position in the stream.

4.4 Triangle Intersection Detection

In the initial process, we put the pairs of intersected components into the intersection stream. In the following process, we detect whether the collision pairs of the intersection stream have intersected at time t_1 with TID. If the previous intersected components have no intersection at the current time, we flag it as invalid, and then we use CCD to guarantee no penetration occurs. The triangle-triangle intersection test can be decomposed into a small number of 3D orientation tests based on the following Sign Theorem[30].

Theorem 1 (Sign Theorem[30]). *Given four points a, b, c, and d, we define the discriminant:*

$$\det(a,b,c,d) = \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{vmatrix}$$

$$= \begin{vmatrix} a_x - d_x & a_y - d_y & a_z - d_z \\ b_x - d_x & b_y - d_y & b_z - d_z \\ c_x - d_x & c_y - d_y & c_z - d_z \end{vmatrix}. \qquad (1)$$

The sign of (*a,b,c,d*) *has two geometric interpretations, each corresponding to a right-hand rule. If the right-hand screw of vectors* ***ba*** *and* ***ca*** *points to d* (*Fig.*5(*a*))*, or if the right-hand screw of* ***ab*** *directs along* ***cd*** (*Fig.*5(*b*))*, the sign of* (1) *is above zero.*

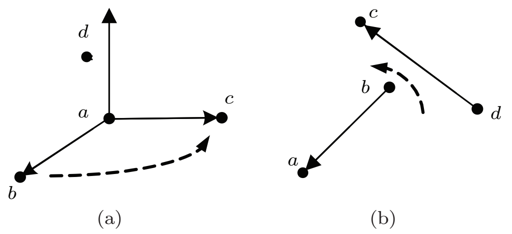


Fig.5. Sign discriminant. We can determine whether the vertex d is above, on or below a plane (Fig.5(a)) and whether $\boldsymbol{cd}$ is along the right-hand screw of $\boldsymbol{ab}$ (Fig.5(b)).

Based on Theorem 1, triangle intersection detection can be summarized as Algorithm 2. First we check whether the vertices of one triangle are on the same side of the other triangle. If so, we conclude no intersection occurs. If all of the vertices are in the same plane, we use line-line intersection detection to judge whether intersection occurs and then terminate the algorithm. Otherwise we make p_1 be the only vertex in the positive half space induced by T_2 (line 11 in Algorithm 2). Finally, we test edge pairs (p_1q_1, p_2q_2) and (p_1r_1, r_2p_2) with Theorem 1. As shown in Fig.6, L is the intersection line of triangles T_1 and T_2, which are represented with *ij* in T_1 and *kl* in T_2 respectively. If all these discriminants are not above zero, that means *ij* intersects with *kl*, namely intersection occurs.

Algorithm 2. Triangle Intersection Detection
```
 1: Compute $\det(p_2, q_2, r_2, p_1)$, $\det(p_2, q_2, r_2, q_1)$,
    $\det(p_2, q_2, r_2, r_1)$
 2: if the signs are all above zero then
 3:     Return false
 4: if the signs are equal to zero then
 5:     Return check intersection with line-line intersection test
        in the mutual plane
 6: Compute $\det(p_1, q_1, r_1, p_2)$, $\det(p_1, q_1, r_1, q_2)$,
    $\det(p_1, q_1, r_1, r_2)$
 7: if the signs are above zero then
 8:     Return false
 9: Make $p_1$ be the only vertex in the positive halfspace in-
    duced by the plane of triangle $T_2$
10: Compute $\det(p_1, q_1, p_2, q_2)$, $\det(p_1, r_1, r_2, p_2)$
11: if the signs are not above zero then
12:     Return true
13: Return false
```

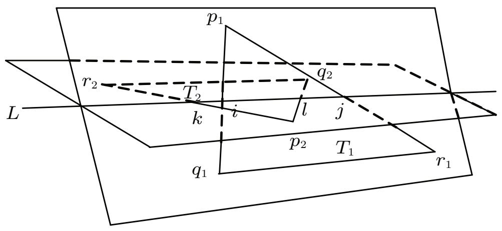


Fig.6. Triangle intersection detection. Based on Theorem 1, if all these discriminants are not above zero, intersections occur.

5 Implementation and Results

In this section, we describe our implementation and present the performance of our algorithm on several benchmarks.

Implementation. We implemented our algorithm on a standard Intel Core machine on a 64-bit Windows 7 platform (3.0 GHz, 8 GB RAM, NVIDIA Tesla K20c GPU). We used CUDA toolkit 5.5 and Visual studio 2008 as the development environment. k-DOPs (specifically 16-DOPs) was used as bounding volumes because they provide a good balance between tight fitting and rapid updating[11]. Representative triangle[22] was employed to reduce the redundant elementary tests. We used deforming non-penetration filter (DNF)[18] to cull the false positives.

Benchmarks. In order to test the performance of our algorithm, we have tested our method on four benchmarks.

• *Microscope.* As shown in Fig.1, the assembly has 19k triangles, and 2.9k simplified triangles with QSlim library.

• *Tooling.* As shown in Fig.7, the assembly has 43.5k triangles, and 5.1k simplified triangles with QSlim library.

• *Lathe.* As shown in Fig.8(a) and Fig.8(b), the assembly has 120k triangles, and 4.3k simplified triangles with QSlim library.

• *Robot.* As shown in Fig.8(c) and Fig.8(d), the assembly has 43k triangles, and 3.8k simplified triangles with QSlim library.

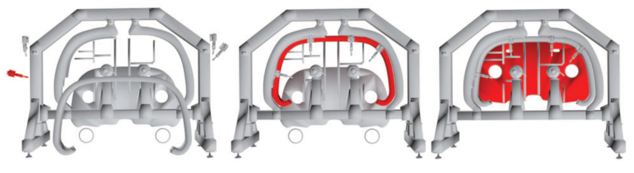


Fig.7. Tooling. (a) Disassembly result. (b) (c) Assembly process.

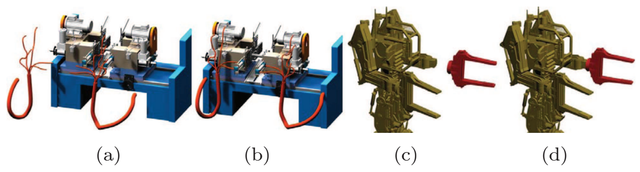


Fig.8. Benchmarks. (a) (c) Disassembly process.

Table 1 highlights the performance of our algorithm. In addition, we have compared the speedup with front-based CCD under a 4-core PC[19], as shown in Fig.9.

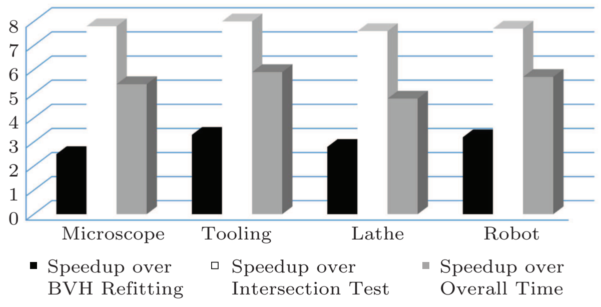


Fig.9. Speedup over front-based CCD under 4 cores.

Table 1. Executed Time (ms) of Our Method on BVH Refitting, Intersection Tests and Overall Time

Model	BVH Refitting	Intersection Test	Overall Time
Microscope	9	11	20
Tooling	14	17	31
Lathe	13	14	27
Robot	10	12	22

6 Comparison and Analysis

In this section, we compare our algorithms with previous GPU and multi-core based CD algorithms and highlight some of the benefits.

Comparison. The previous virtual assembly systems mainly check for collision between OBBs instead of actual components to improve the performance. As opposed to these methods, our approach provides object-space accuracy and is quite fast. Our algorithm is a purely GPU-based approach and there is almost no communication bottleneck in terms of repeated data transfer between CPU and GPU. Moreover, our framework is more flexible and makes it easy to incorporate with the previous virtual disassembly systems.

We have implemented some previous methods (including local-lock based CCD[29], front-based multi-core CCD[19]). In the following of the section, we would like to compare our method with them.

• *Local-Lock Based CCD*. They abstract the stream data into vertex stream, BVH stream, front stream, triangle pair stream, feature pair stream, etc.[29] When new nodes are inserted into the front stream or triangle pair stream, they are firstly reserved in the shared memory with atomic operation, and then merged into the global memory. Experiments show we can get almost the same performance as the local-lock method. But our method is more compatible with low-end graphics card for no limitation on the capacity of shared memory. The result illustrates our global-lock method is better for the small assembly.

• *Front-Based Multi-Core CCD*. At the initial configuration, they insert the non-intersected or leaf nodes of BVTT into a front list. In the following process, they can traverse the front parallel with multi-core[19]. Experiments prove that our method achieves 6∼8 times speedup compared with the front-based method under a 4-core PC.

Limitation. In this paper, we design an interactive virtual disassembly system which employs 3D projector to immerse users into the virtual environment, and allows people to dismount the assembly with interactive devices. When one component is surrounded by another tightly, it is not easy to separate the components with 3D mouse, because many collisions occur.

7 Conclusions

In this paper, we imported CCD into virtual disassembly and presented a hybrid collision handling approach. We solved the problem that collision cannot be detected when the components are embedded into each other at the initial configuration. The experiment results show that our method can improve the overall performance of collision detection and response, and achieve real-time simulation.

In the future work, we would like to improve the performance by integrating new research achievements, such as the improved front-based CCD with a load-balancing static task partition strategy[25].

Acknowledgement We would like to thank Sung-eui Yoon, Zhen-Dong Wang and Min Tang for useful discussion and paper writing. We also thank NVIDIA for the Hardware Donation Program (Tesla K20 GPU) and the anonymous reviewers for their helpful feedbacks.

References

[1] Srinivasan H, Shyamsundar N, Gadh R. A framework for virtual disassembly analysis. *Journal of Intelligent Manufacturing*, 1997, 8(4): 277–295.

[2] Coutee A S, Bras B. Collision detection for virtual objects in a haptic assembly and disassembly simulation environment. In *Proc. ASME 2002 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, September 29–October 2, 2002, pp.11–20.

[3] Gottschalk S, Lin M, Manocha D. OBBTree: A hierarchical structure for rapid interference detection. In *Proc. the 23rd ACM SIGGRAPH*, August 1996, pp.171–180.

[4] Popescu D, Lacob R. Disassembly method based on connection interface and mobility operator concepts. *The International Journal of Advanced Manufacturing Technology*, 2013, 69(5/6/7/8): 1511–1525.

[5] Pomares J, Puente S T, Torres F, Candelas F A, Gil P. Virtual disassembly of products based on geometric models. *Computers in Industry*, 2004, 55(1): 1–14.

[6] Wu J, Dick C, Westermann R. Efficient collision detection for composite finite element simulation of cuts in deformable bodies. *The Visual Computer*, 2013, 29(6/7/8): 739–749.

[7] Fan W S, Wang B, Paul J C, Sun J G. An octree-based proxy for collision detection in large-scale particle systems. *Science China Information Sciences*, 2013, 56(1): 1–10.

[8] Hubbard P M. Interactive collision detection. In *Proc. IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993, pp.24–31.

[9] Bradshaw G, O'Sullivan C. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics*, 2004, 23(1): 1–26.

[10] van den Bergen G. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 1997, 2(4): 1–13.

[11] Klosowski J, Held M, Mitchell J, Sowizral H, Zikan K. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 1998, 4(1): 21–36.

[12] Tang M, Manocha D, Kim Y J. Hierarchical and controlled advancement for continuous collision detection of rigid and articulated models. *IEEE Transactions on Visualization and Computer Graphics*, 2014, 20(5): 755–766.

[13] Tang M, Curtis S, Yoon S E, Manocha D. ICCD: Interactive continuous collision detection between deformable models using connectivity-based culling. *IEEE Transactions on Visualization and Computer Graphics*, 2009, 15(4): 544–557.

[14] Schvartzman S C, Pérez Á G, Otaduy M A. Star-contours for efficient hierarchical self-collision detection. *ACM Transactions on Graphics*, 2010, 29(4): 80:1–80:8.

[15] Barbič J, James D L. Subspace self-collision culling. *ACM Transactions on Graphics*, 2010, 29(4): 81:1–81:9.

[16] Zheng C, James D L. Energy-based self-collision culling for arbitrary mesh deformations. *ACM Transactions on Graphics*, 2012, 31(4): 98:1–98:12.

[17] Tang M, Manocha D, Yoon S E, Du P, Heo J P, Tong R F. VolCCD: Fast continuous collision culling between deforming volume meshes. *ACM Transactions on Graphics*, 2011, 30(5): 111:1–111:15.

[18] Tang M, Manocha D, Tong R F. Fast continuous collision detection using deforming non-penetration filters. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, February 2010, pp.7–14.

[19] Tang M, Manocha D, Tong R F. MCCD: Multi-core collision detection between deformable models using front-based decomposition. *Graphical Models*, 2010, 72(2): 7–23.

[20] Tang C, Li S, Wang G. Fast continuous collision detection using parallel filter in subspace. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, February 2011, pp.71–80.

[21] Du P, Tang M, Tong R F. Fast continuous collision culling with deforming non-collinear filters. *Computer Animation and Virtual Worlds*, 2012, 23(3/4): 375–383.

[22] Curtis S, Tamstorf R, Manocha D. Fast collision detection for deformable models using representative-triangles. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, February 2008, pp.61–69.

[23] Tang M, Zhao J Y, Tong R F, Manocha D. GPU accelerated convex hull computation. *Computers & Graphics*, 2012, 36(5): 498–506.

[24] Zhao J Y, Tang M, Tong R F. Connectivity-based segmentation for GPU-accelerated mesh decompression. *Journal of Computer Science and Technology*, 2012, 27(6): 1110–1118.

[25] Zhang X Y, Kim Y J. Scalable collision detection using *p*-partition fronts on many-core processors. *IEEE Transactions on Visualization and Computer Graphics*, 2014, 20(3): 447–456.

[26] Wong T H, Leach G, Zambetta F. An adaptive octree grid for GPU-based collision detection of deformable objects. *The Visual Computer*, 2014, 30(6/7/8): 729–738.

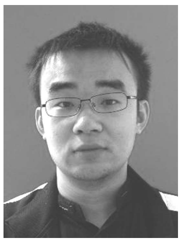
[27] Kim D, Heo J P, Huh J, Kim J, Yoon S E. HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs. *Computer Graphics Forum*, 2009, 28(7): 1791-1800.

[28] Pabst S, Koch A, Straβer W. Fast and scalable CPU/GPU collision detection for rigid and deformable surfaces. *Computer Graphics Forum*, 2010, 29(5): 1605–1612.

[29] Tang M, Manocha D, Lin J, Tong R F. Collision-streams: Fast GPU-based collision detection for deformable models. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, February 2011, pp.63–70.

[30] Devillers O, Guigue P. Faster triangle-triangle intersection tests. Research Report, RR-4488, INRIA, 2002. https://hal.inria.fr/inria-0072100/file/RR-4488.pdf, Mar. 2015.

Peng Du received his Ph.D. degree in computer science and technology from Zhejiang University, Hangzhou, in 2013. Currently, he is a postdoctoral researcher in Korea Advanced Institute of Science and Technology (KAIST), as well as a lecturer in Hangzhou Dianzi University. His research interests include computer graphics, global illumination, parallel computation and collision detection.



Jie-Yi Zhao received his Ph.D. degree in computer science and technology from Zhejiang University, Hangzhou, in 2013. Currently, he is a postdoctoral researcher in the University of Texas at Houston. His research interests include computer graphics and GPU computing.

Wan-Bin Pan is a Ph.D. candidate in the State Key Lab of CAD & CG of Zhejiang University, as well as a lecturer in Hangzhou Dianzi University. His research interests include 3D CAD model reuse and virtual assembly.

Yi-Gang Wang received his Ph.D. degree in applied mathematics from Zhejiang University, Hangzhou, in 1999. Currently, he is a professor in Hangzhou Dianzi University. His research interests include virtual reality, computer graphics, and image process.