

Fast collision detection approach to facilitate interactive modular fixture assembly design in a virtual environment

Gaoliang Peng · Xin Hou · Chong Wu · Tianguo Jin ·
Xutang Zhang

Received: 27 May 2008 / Accepted: 21 April 2009 / Published online: 9 May 2009
© Springer-Verlag London Limited 2009

Abstract Collision detection is a fundamental component to simulate realistic and natural object behaviors in virtual reality-based system. In this paper, a hybrid method of space decomposition and bounding volume approach is presented to assist modular fixture assembly design in a virtual environment. Based on characteristics of modular fixture, a novel space decomposition methodology at object level is proposed, which is achieved by automatically partitioning the checking space into cells according to the oriented bounding boxes of assembled elements after the initial approximate collision detection using the intersection checking method based on separation plane-based bounding box. Then the pairs of candidate objects are determined for narrow phase exact polygons overlap tests. Results from several performance tests on modular fixture design system show that an important advantage of this proposed method compared with other universal algorithms is its simple information representation and low preprocessing cost.

Keywords Collision detection · Virtual assembly · Modular fixture · Space decomposition · Bounding volume

1 Introduction

Virtual reality (VR) became a very common mean during the development of the industrial products. The aid provided by VR is noticeable, since the user can interact with the virtual prototype in a very natural way [1–3]. VR holds great potential in manufacturing applications to solve problems before fatal mistakes occur in practical manufacturing so that great costs are prevented. VR applications have gained increasing attention internationally.

Fixture design takes a significant part of the total time (cost) necessary for technical and technological production preparation. The design of a fixture is a highly complex and intuitive process, which requires knowledge and experience [4]. Modular fixtures are one of the important aspects of manufacturing. Proper fixture design is crucial to product quality with regard to the precision, accuracy, and finish of the machined part. Modular fixture is a system of interchangeable and highly standardized components designed to securely and accurately position, hold, and support the workpiece throughout the machining process [5]. Traditionally, fixture designers rely on experiences or use trial-and-error methods to determine an appropriate fixture scheme.

Since the potentially high degree of “reality” experienced in a virtual environment (VE), the VR-based modular fixture design has the advantages of designing a fixture in a natural and instructive manner, providing better match to the working conditions, reducing lead-time, and generally providing a significant enhancement to fixture productivity and economy [6]. In order to achieve this goal, the VR system must be able to simulate realistic and natural object behaviors. First of all, as a basic requirement of fixture design, there should be no collision between fixture, component and machine tool [7, 8]; the objects not

G. Peng (✉) · X. Hou · T. Jin · X. Zhang
School of Mechanical and Electrical Engineering,
Harbin Institute of Technology,
Harbin, China
e-mail: pgl7782@hit.edu.cn

C. Wu
School of Management, Harbin Institute of Technology,
Harbin, China

penetrating into others must be guaranteed. Therefore, a fast interactive collision detection (CD) algorithm is fundamental in such a VR system.

However, collision checking for a complex VE is computationally intensive. Researchers have addressed some “universal” algorithms to reduce the computational costs. But these algorithms often need auxiliary data structures and require intensive preprocessing time cost. In addition, the implementation of such algorithm is very complicated. Therefore, based on the well study of modular fixture characteristics and practical requirements, we develop a “special” CD algorithm to keep the associated costs as low as possible for VR-based modular fixture assembly design.

The paper is organized as follows. A review of related work of the existing CD algorithms is presented in Section 2. Section 3 gives an overview of our proposed algorithm. In Section 4, we describe the space subdivision model used in our algorithm. Section 5 provides the details about the broad phase of our proposed algorithm, in which irrelevant objects are discarded and a set of objects that can possibly collide are determined. The narrow phase for exact polygon based overlap tests is described in Section 6. Section 7 presents some experimental results of our algorithm, and finally, in Section 8, we give concluding remarks and outline directions for future extensions of this work.

2 Related work

During the past few years, a great deal of effort has been made to solve the CD problem for various types of interactive 3D graphics and scenarios. For a workspace filled with n objects, the most obvious problem is the $O(n^2)$ problem of detecting collisions between all objects, which is time consuming and not bearable if the number n is large. Thus, some necessary techniques are needed to reduce the computational costs. Generally, a CD algorithm consists of two main steps, namely broad phase and narrow phase [9]. The first phase aims to filter out pairs of objects which are impossible to interact and determine which objects in the entire workspace potentially interact. The second phase is to perform a more accurate test to identify collision between those selected object parts in the first phase, moreover if necessary, to find the pairs of contacting primitive geometric elements (polygons), and to calculate the overlapping distance.

For a CD algorithm, it is critical to reduce the number of pairs of objects or primitives that need to be checked. Therefore, a number of different techniques have been used to make coarse grain detection, among which space decomposition and bounding volumes is most popular.

In space decomposition methods, the environment is subdivided into space grids using hierarchical space subdivision. Objects in the environment are clustered hierarchically according to the regions that they fall into. These objects are then checked for intersection by testing for overlapping grid cells exploiting spatial partitioning methods like Octrees [10, 11], BSP-trees [12], k -d trees [13], etc. Using such decompositions in a hierarchical manner can further speed up the collision detection process but leads to extremely high storage requirements.

Bounding volume (BV) approach is used in previous computer graphics algorithms to speed up computation and rendering process. The BV of a geometric object is a simple volume enclosing the object. Typically, BV types are axis-aligned boxes (AABBs) [14], spheres [15], and oriented bounding boxes (OBB) [16].

Since AABBs method is simple to compute and allows efficient overlap queries, it is often used in hierarchy, but it also may be a particularly poor approximation of the set that they bound, leaving large “empty corners.” The systems utilizing AABBs include I-COLLIDE [17], Q-COLLIDE [18], and SOLID [19], etc.

Bounding sphere is another natural choice to approximate an object as it is particularly simple to test pairs for overlap, and the update for a moving object is trivial. However, spheres are similar to AABBs as they can be poor approximations to the convex hull of contained objects.

In comparison, an OBB is a rectangular bounding box at arbitrary orientations in 3D space. In an ideal case, the OBB can be repositioned such that it is able to enclose an object as tightly as possible. In other words, the OBB is the smallest possible bounding box of arbitrary orientation that can enclose the geometry in question. This approach is very good at performing fast rejection tests. A system called RAPID [20] for interference detection based on OBB has been built, which approximates geometry better than AABBs. The shortcomings of OBB-tree against sphere tree lie in its slowness to update and orientation sensitive [9].

Most CD-related researches are involved in “universal” algorithms, and few literatures are found to develop CD approach in a special application like virtual assembly. Actually, a fast and interactive collision detection algorithm is fundamental to a virtual assembly environment, which allows designers to move parts or components to perform assembly and disassembly operations.

Figueiredo [21] presented a faster algorithm for the broad and narrow phases of the collision detection algorithm of determining precise collisions between surfaces of 3D assembly models in virtual prototype environments. The algorithm used the overlapping AABB and the R-tree data structure to improve performance in both the broad and narrow phases of the collision detection. This approach is for such a VE with objects dispersed in the

space. In addition, the R-tree data structure is very memory intensive.

Stephane [22] worked on continuous collision detection methods and constraints to deal with rigid polyhedral objects for desktop virtual prototyping. Whereas such a 4D method is only useful for handling the path of known moving objects. Especially, the algorithm is so computationally intensive that it has to run on high-end computers.

Collision detection is a critical problem in multi-axis numerical control (NC) machining with complex machining environments. There has been much previous work on interference detection and avoidance in NC machining simulation. Wang [23] developed a graphics-assisted collision detection approach for multi-axis NC machining. In this method, a combination of machining environment culling and a two-phase collision detection strategy was used.

Researches surveyed above provided various efficient techniques to carry out collision detection for polygonal models. However, these popular algorithms aimed at general polygonal models, most of which need expensive pretreatments or large system memory or both of them in order to improve the performance and meet real-time requirements. Therefore, when these algorithms are utilized in desktop VR application system such as modular fixture design, the requirement of real time cannot be well guaranteed.

Few CD researches can be found in the area of computer-aided fixture design. Hu [24] presented an algorithm of fast interference checking between the machining tool and fixture units, as well as between fixture units, to replace the visually checked method. Moreover, in Kumar's work [25], in order to automate interference-free modular fixture assembly design, the machining interference detection is accomplished through the use of cutter-swept solid based on cutter-swept volume approach. However, these algorithms are only capable of static interference checking and applied in CAD software packages.

The research presented in this paper makes a solution to these issues by addressing a “special” collision detection algorithm for VR-based modular fixture design. The proposed algorithm uses the hybrid approach of space decomposition and bounding volume method to get high performance.

3 Algorithm overview

3.1 Requirements for proposed algorithm

We aimed to develop a desktop VR-based modular fixture assembly design system, in which the designer can select suitable fixture elements and put them together to generate

a fixture structure, like “building blocks.” Without physical fixture elements, he/she can test different structure schemes and finally design a feasible fixture configuration that meets the fixturing function requirements. In order to retain high degree of “reality” in engineering application, there are three main requirements for a CD algorithm to perform modular fixture configuration design:

1. *Precise and fast*: During the simulation of assembly and disassembly operations, finding precise collisions is an important task for achieving realistic behavior [26]. When the user interactively assembles a part or a component, the “flying” object may collide with static models, thus the system must find out the “colliding” event immediately. The interval between two checking points should be near enough to achieve better performance. Otherwise, when objects move very fast, they may appear before checking, which will reduce the immersive feelings. Therefore, the proposed system carries out a CD checking task in each rendering loop of VE. In addition, in modular fixture assembly design process, the designer selects elements and assembles them to right position or disassembles them to change the fixture configuration. Once an element is assembled or disassembled, the “static” environment models are updated. Accordingly, the CD checking model needs restructure. So the preprocess should not take too long; otherwise, the performance of proposed system will be impaired severely for certain “smooth feel” cannot be achieved.
2. *Low system requirements*: Finding collisions in a 3D environment is time-consuming. In some cases, it can easily consume up to 50% of the total run time [21]. However, in modular fixture design workspace, there are some other time-consuming tasks, such as design process control and reasoning, automatic geometric constraints recognition and solving, etc. In spite of the complexity of the 3D virtual prototypes due to thousands of polygons, the designed CD checking procedure must be done in real time with relatively low system resource demands.
3. *Low hardware cost*: In order to achieve wider engineering applications, the proposed modular fixture assembly system is designed to run on common PC like popular CAD commercial software. Although much research has engaged in developing hardware-accelerated CD algorithms, which utilize special graphic hardware, like graphics processing unit, to deal with the computing collisions, thus the system's CPU can be freed. Nevertheless, we did not plan to adopt this kind of method and optimize performance only from software implementation. The objective of this research is to develop a CD algorithm

Taking into account all above requirements, unfortunately, these objectives usually are in conflict. To meet the precise demand, we must increase checking frequency which will enormously increase the computational complexity and the memory bandwidth requirement. So, how can a balance be reached with regard to these? In other words, how can the utilization of system resources be minimized yet the performance optimized without the help of extra hardware? It is the start point of our algorithm.

3.2 Modular fixture analysis

The objective of this research is to develop a CD algorithm for assisting in modular fixture assembly design operations in VE. To simplify the algorithm and to gain high performance, the characteristics of modular fixture should be well studied.

1. *Process of modular fixture assembly design*: The tasks of modular fixture assembly design are to select the proper fixture elements and assemble them to a configuration one by one according to the designed fixturing plan. Thus, the CD problem in VR-based modular fixture assembly design can be stated as: the intersection checking between one moving object (assembling element or unit) with the static environment objects (assembled elements) at discrete time.
2. *Fixture element shape*: Modular fixture elements with regular shape can be classified into three types, namely, block, cylinder, and block-cylinder [27]. Other complicated assembly units can be regarded as compositions of these three meta-elements. It is well known that the OBB is tighter than the AABB and sphere. Moreover, when an object changes its position and orientation in VE, its OBB does not need to rebuild. Therefore, we can construct OBBs of modular fixture elements off-line and store them as attributes of element models.

During the assembly design process, such attributes can be retrieved directly; thus, complex work for constructing bounding volume in run time can be avoided.

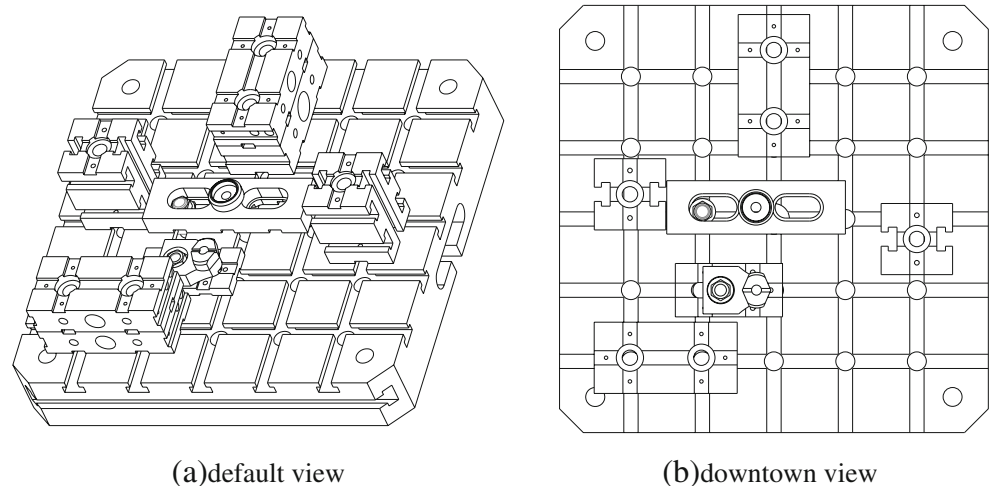
3. *Fixture element layout*: A modular fixture system often consists of supporting units, locating units, and clamping units. These units lie out on the base plate and provide corresponding functions at certain positions. As Fig. 1 shows, in the projection view parallel to the base plate, the units are arranged in some kind of “regions.” In addition, to meet the height requirement of fixturing point, a unit often utilizes a number of supporting elements severed as blocking up objects. Therefore, at the direction perpendicular to the baseplate, the elements lay out hierarchically. Accordingly, we can decompose the space with regard to elements layout feature.

3.3 Algorithm flowchart

According to the above characteristics of modular fixture, the proposed algorithm is designed to decrease the complexity and meet the requirements of VR-based modular fixture assembly design. As Fig. 2 shows, at the preprocessing stage, once an element or component is assembled or disassembled, the Layer-based Projection Model (LPM) is established in terms of OBBs of those assembled elements. Such an LPM is used for the CD checking when a new object is assembled.

Just like the traditional CD method, proposed algorithm consists of two steps, namely, broad phase and narrow phase. The broad phase is responsible for filtering pairs of objects that cannot intersect. At this stage, it determines pairs of objects in the same subspace, whose silhouettes in LPM overlap and their OBBs intersect. These pairs of objects are candidates for exact polygon-based collision tests in the next narrow phase. During the broad phase, the

Fig. 1 Modular fixture structure



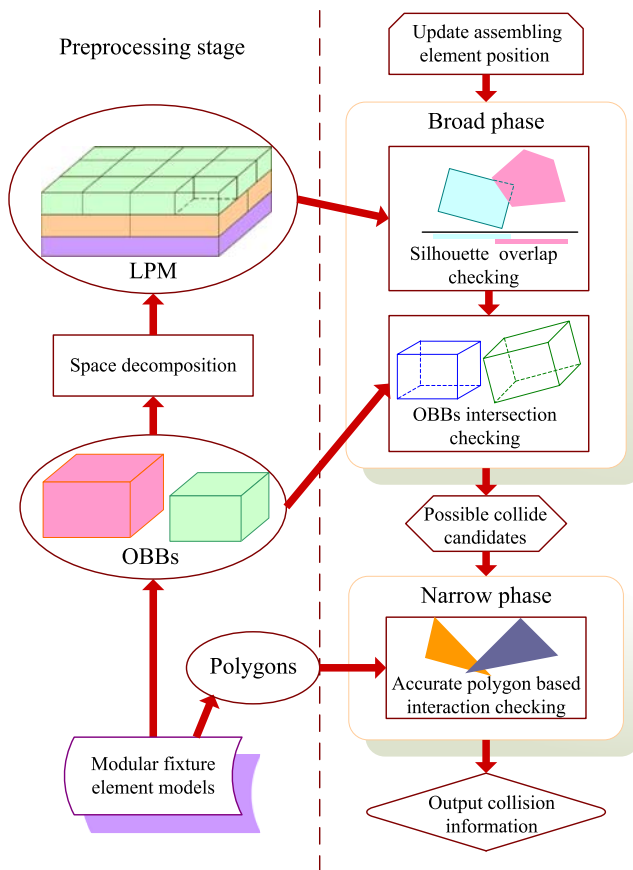


Fig. 2 Overview of collision detection algorithm

test may cease at any time if no intersection is found, which helps to reject many noncollision or trivial collision cases. In the narrow phase, the collision detection algorithm will calculate detailed intersection between geometrical meshes of the objects. If no intersection polygons are found, the collide will not occur, and the active object can keep on moving. Otherwise, whenever overlaps are detected, related reactions (for proposed system, it highlights objects and does back-tracking) may arise.

4 Space decomposition for identifying neighboring objects

Considering the fact that most regions of the “universe” are occupied by only a few objects or left empty, it means that collision only happens among objects that are close enough. So we can use this phenomenon to filter out most of “far-away” objects. Space decomposition is the common approach to be used for this intention. It first splits the “universe” into cells and then does further collision tests for objects in the same cell. In order to keep generality, most of existing space subdivision approaches are based on a set of polygons. Such a “polygon-oriented” approach is so

computationally intensive to deal with large number of polygons. Since standard components are almost with relatively regular shapes, we plan to develop an “object-oriented” space decomposition method.

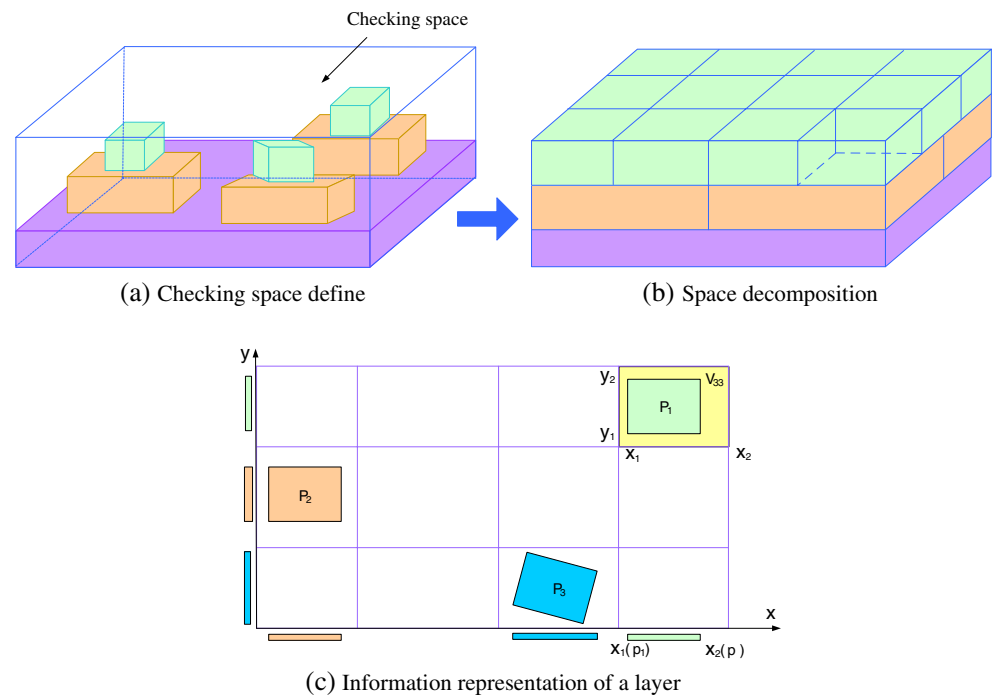
4.1 Space decomposition model

After the baseplate is arranged, the remaining work is to assemble the fixture elements or units onto the baseplate. As the assembling elements or units move to the assembled position, collisions may happen between active object and the assembled elements that have been fixed in the space around the baseplate. Hence, the CD checking process needs start-up only after the active object enters into this space. Firstly, as Fig. 3a shows, we define a valid collision space noted as Ω , which is a cuboid whose bottom face is decided by the baseplate, and its height would change along with the assembling operation. The top of Ω is determined by the vertex coordinates of OBBs. Ω is defined to guarantee that all the assembled elements are inside.

After the checking space is identified, we need to decompose the space into a number of cells. How can we organize these cells into proper structure and represent the relevant information to facilitate interaction checking? In literature, some kinds of hierarchical data structure, like R-tree structure [21], have been used to help find neighbors. In complex environment with lots of dispersed objects, this complicated data structure is useful. For modular fixture assembly design, the element models are relatively centralized, and the number of objects is not so much. Consequently, the complex data structure is not needed as constructing such a model is time-consuming.

We propose a novel data model to represent partition of the checking space. The model gets the advantages of easy intersection tests and simple information representation. As Fig. 3b shows, the checking space is decomposed into several layers along the axis vertical to the baseplate. Each layer can be represented as a 4-tuple $L_i = \{h_1, h_2, V, B\}$, where h_1 is the start height, h_2 is the end height, V is the grid information of this layer, and B describes the projection of elements' OBBs belonging to this layer.

For each layer, the stored information is illustrated in Fig. 3c. Easy to overlap checking, we orthogonally project the bounding box onto the x, y axis (convenient for illustration and does not lose universality). Then, with these projections, intervals are formed on each axis for each object. We construct one list for each axis. Each list contains the coordinate value of the endpoints of the interval on the corresponding axis. By comparing the endpoints, the corresponding pair of objects that are in contact may be determined. If the intervals do not overlap, the corresponding two objects are not in contact and can be discarded.

Fig. 3 LPM representation

4.2 Space decomposition model construction

The above section gives the representation of our space decomposition model; this section will discuss how to construct and reconstruct this model. Most existing space partition methods decompose an entire space into cells in terms of primitive geometric elements (polygons), by computing the position of each polygon and assigning them into corresponding cells, which are often organized into a hierarchical data structure. Despite the data structure remarkably speeding up the CD checking procedure, the establishment of such structure is a complex process and time-consuming. In a situation that the objects in an environment are uniform or the environment models do not change frequently, the cost for preprocessing may be bearable. However, during the modular fixture virtual assembly process, objects within the checking space are changed with time. Therefore, the space composition model must be rebuilt once a fixture element is assembled.

To reduce preprocessing time, we propose an “element”-based space decomposition method instead of calculation based on scattered “polygons.” That is, the space model is constructed with regard to fixture element models. The detailed process of construction follows three main steps:

1. Layers division

Layers division is to find several separation planes orthogonal with z axis to divide the checking space into a set of layers. As a result, the assembled fixture elements would distribute into different layers. During the interaction

checking, only the elements of the same layer with the assembling element will be taken for consideration.

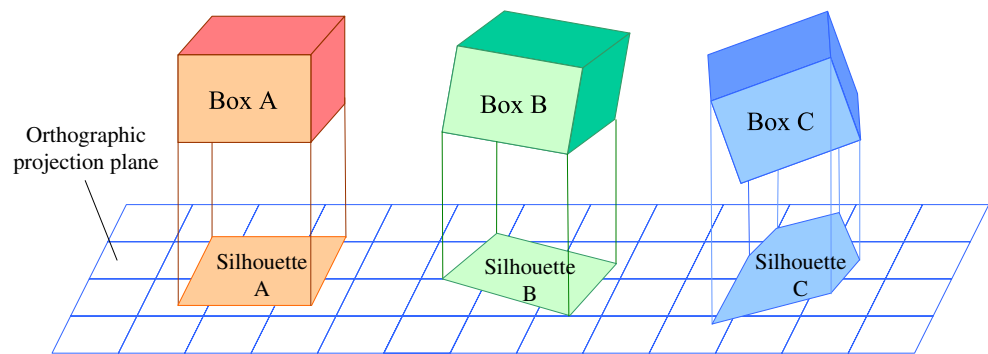
It is a good idea to divide space without cutting objects, but in general, this cannot be avoided. Nevertheless, we find the separation planes among those across the top and bottom face of OBBs of assembled elements. Moreover, the height of each layer should be taller than the height of assembling element’s OBB. The orthographic projections of all OBBs of assembled elements yield rectangles or hexagons. If these hexagons project onto z axis, it would generate a number of single intervals. We record and sort their endpoints and determine which point is the “separation point.” A simple program can realize this function. The principle is to try to make one layer to contain objects as much as possible without cutting them.

2. OBBs projection

According to the results of layers division, in the orthographic view of z axis, we get the orthographic projections of OBBs in each layer (Fig. 4). As Fig. 5 shows, these may yield three types of silhouettes. Because most of the assembled elements are located perpendicular to the baseplate, most of these silhouettes are rectangles (as box A and B). But for assembling element, because its position and orientation will change timely along with the virtual hand, its silhouette is a polygon with at most six edges (as box C).

After box projection, each generated silhouette will project onto two axes of the projection plane. Under this projection, each silhouette forms two intervals, each one on

Fig. 4 The silhouettes of OBB in orthographic projection plane



each axis. The endpoints of these intervals are recorded in LPM.

3. Grids subdivision

After the projection of each OBB is finished, the next step is dividing the projection plane into grids. We try to guarantee that each OBB's silhouette is inside one grid. And the width and length of each grid are larger than that of the assembling element. Figure 5 shows the process of grid division.

Through the above procedure, the division on OX axis is finished. Then we can take the OY axis division with the above algorithm. Finally, grid division of a projection plane of a layer is completed.

By repeating the procedures of OBBs projection and grid subdivision for all layers, the space composition model is finally established. From the above discussion, we can see

that the process is very simple and the information can be stored for model rebuilding. When a new object is assembled, update of such a model just needs to add the projection of new OBB and to repeat the division procedure. Moreover, in some situation, especially when most elements are assembled, the rebuilding of space decomposition model can be done by adding the new OBB information in it.

5 Broad phase: potential collide candidates finding

This section discusses the broad phase of the proposed CD algorithm, which is responsible for finding out potential candidates for further exact checking. Such a filtering procedure consists of two steps: silhouettes overlap checking and OBBs intersection test. During the test procedure, the collision detection may cease any time if no intersection is found. This procedure can be considered as an initial collision detection, which helps to reject many noncollision or trivial collision cases.

5.1 Approximate tests by silhouettes overlap checking

Once the preprocessing stage is completed and the LPM data structure is constructed, we initiate checking procedure to identify which objects are potentially interacting with the active assembling model (Fig. 6). When the active object is entering into the checking space, at each sample time, the position and orientation of active object's OBB are updated and projected onto the LPM. We first determine what layer the active object has been entered, in terms of the vertex of its OBB. Then we project its OBB, the assembled objects to construct the LPM, onto the corresponding axes. So we can do overlap checking in terms of the intervals on the axes. Figure 7 illustrates how to judge if the two projection silhouettes are interacting.

Situations (a) and (b) can be treated easily by comparing the endpoints of intervals respectively, while situation (c) needs further complex calculation in terms of the edges of the two silhouettes. Because at this stage, we only carry out

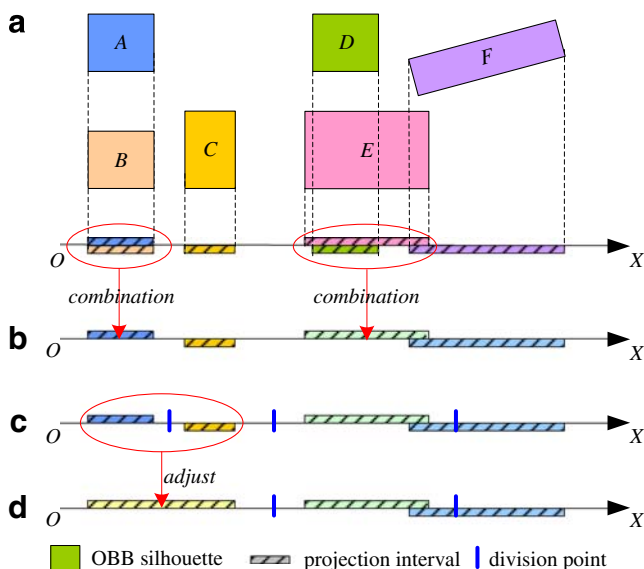


Fig. 5 Division of x axis of projection plane: (a) project the OBB silhouettes to axis OX and generate a set of intervals, (b) combine the generated intervals according to their endpoints, (c) subdivide the axis OX by adding some division points between the endpoints of left intervals, (d) adjust the subdivision zone by combining the short intervals in terms of the longest axis of OBB of assembling element

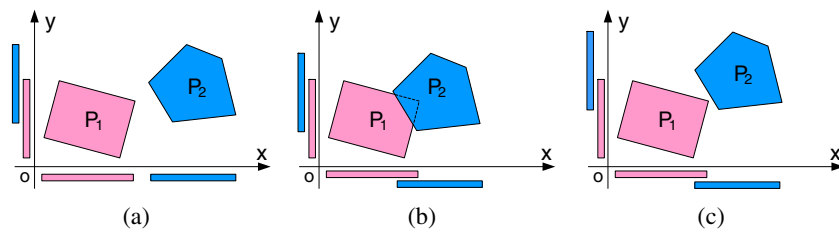


Fig. 6 Three situations for two silhouettes overlap checking: **a** The two silhouettes are not interacting if their intervals on at least one axis are not overlapping. **b** The two silhouettes are interacting when their

intervals on both two axes are overlapping. **c** The two silhouettes are not intersecting but their intervals on both two axes are overlapping

approximate tests to identify the inferring contact objects, to simplify the process, only the first two situations are considered. If the intervals on the two axes are all overlapped, we regard the two projection silhouettes are overlapped. Further calculation about situation (c) is carried out in the next stage.

5.2 OBBs intersection test

At this stage, we further filter out pairs of candidate objects that cannot intersect. It determines pairs of objects identified in the first step, whose bounding volumes overlap. These pairs of objects are candidates for exact collision detection in the narrow phase.

Generally, the interference algorithm typically spends most of its time testing pairs of OBBs for overlap. A simple algorithm for testing the overlap status for two OBBs performs 144 edge-face tests. In practice, it is an expensive test. To perform the test, Gottschalk [16] presented a method to find a “separating axis” for two boxes. Their strategy is to project the centers of the boxes onto the axis and compute the radii of the intervals. If the distance between the box centers projected onto the axis is greater than the sum of the radii, the intervals (and the boxes as well) are disjointed. Although it provides a faster way for the OBB intersection checking, such a “universal” method has to test 15 potential separating axes (three faces from one box, three faces from the other box, and nine pair-wise combinations of edges) for determining overlap status of two OBBs when the two boxes are exactly overlap.

As mentioned earlier, the assembled fixture elements are regularly assigned onto the baseplate, which means most of

OBBs are parallel to the baseplate. Therefore, to achieve a better performance, the algorithm proposed here uses the separation plane-based method for OBB intersection checking. Comparing to the separation axis-based method, our approach is simple to implement since such a separation plane may easily be found just by testing vertex coordinate. Once such a plane is found, the boxes are surely to be disjointed, so the remainder of tests is unnecessary.

First of all, we introduce the principles of separation plane-based method [28]. In general, for any two non-colliding convex polyhedrons, a separation plane exists between them which are either:

1. Parallel to a face of the first polyhedron
2. Parallel to a face of the second polyhedron
3. Parallel to an edge of the first polyhedron and to an edge of the second one

Based on the principle of separation plane, we propose a method for checking whether a separation plane is one of the existing three types. The first two types of plane can be checked by simply comparing vertexes; to filter irrelevant cases early and to avoid complex computation, type 1 and type 2 separation plane checking is carried out firstly, and then the type 3 plane. The algorithm is described in detail as follows:

1. Type 1 and type 2 separation plane checking

As shown in Fig. 7, let C_0 be the World Coordinate System (WCS). Let C_1 and C_2 be the local WCS attached to bounding box B_1 and B_2 , respectively. And boxes B_1 and B_2 are axis-aligned relative to its local CS. Let faces set of each box be $\{f_i | i = 1, 2, \dots, 6\}$, $f_i = (k_i, L_i)$, where $\vec{k}_i \in \{\pm x, \pm y, \pm z\}$ is the vector of f_i , L_i is the distance between f_i and the origin of local WCS. Let the vertex set be $\{v_j | j = 1, 2, \dots, 8\}$. The transform matrix of C_0 to C_1 can be represented as $M(C_0 \rightarrow C_1)$. Then the detail procedure is as follows:

Step1: Transform all vertex of B_2 into C_1 according to Eq. 1.

$$v_j(x', y', z') = v_j(x, y, z) \times M(C_2 \rightarrow C_0) \times M^{-1}(C_1 \rightarrow C_0) \quad (1)$$

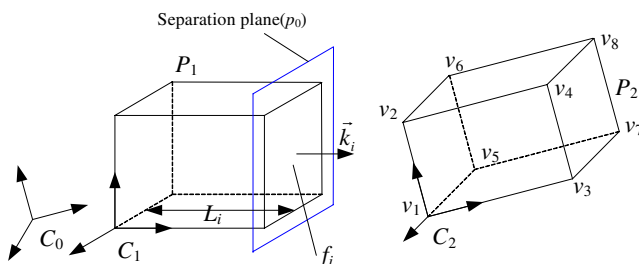


Fig. 7 Type 1 separation plane

Step2: To each face of B_1 , compare each vertex in terms of Eq. 2; if all vertexes satisfy this condition, it indicates that f_i is the separation plane, and jump to step 4. Otherwise, proceed to the next step.

$$\begin{cases} v_j(\vec{k}_j) > L_i & \vec{k}_j \in \{+x, +y, +z\} \\ v_j(\vec{k}_j) < L_i & \vec{k}_j \in \{-x, -y, -z\} \end{cases} \quad (2)$$

Where $v_j(\vec{k}_j)$ is the value of vertex v_j with the coordinate axis parallel to \vec{k}_j .

Step3: Exchange the roles of B_1 and B_2 and repeat the two steps above.

Step4: End the algorithm and output the checking results.

2. Type 3 separation plane checking

As shown in Fig. 8, let C_0 be the world CS. Let C_1 and C_2 be the local CS attached to bounding box B_1 and B_2 , respectively. And boxes B_1 and B_2 are axis-aligned relative to its local CS. Let edges set of each box be $\{b_i | i = 1, 2, \dots, 12\}$, in local CS, if the projection plane is vertical to b_i , its projection becomes a point, marked as $p_i = P_i(\vec{k})$ (\vec{k} is the project direction). Let the vertex set be $\{v_j | j = 1, 2, \dots, 8\}$. Then the detailed procedure is shown as follows:

Step1: Transform all vertexes of B_2 into C_1 according to Eq. 1.

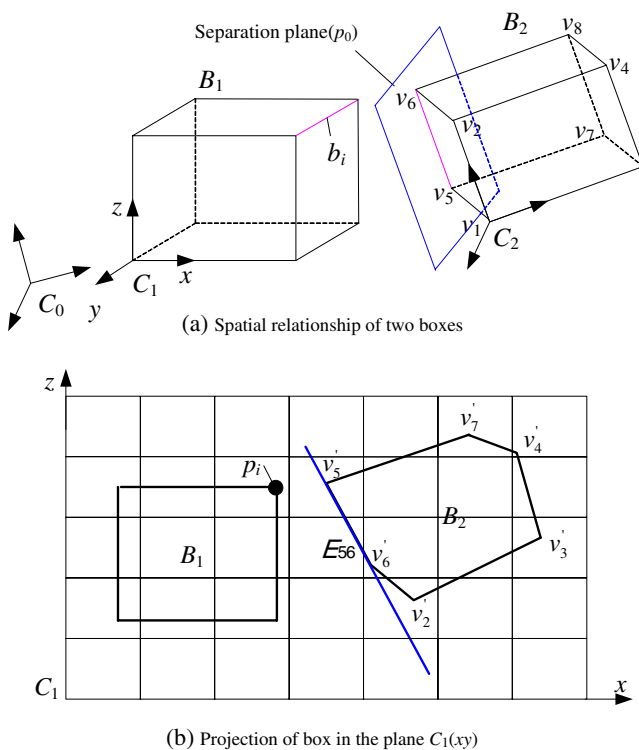


Fig. 8 Type 2 separation plane

Step2: In C_1 , project B_1 and B_2 along \vec{k} , $\vec{k} \in \{x, y, z\}$. In each projection view plane, the projection of B_1 is a rectangle, its vertexes set is $V = \{v_i | i = 1, 2, \dots, 4\}$. And the projection of B_2 is a polygon, which may have more than four edges; let the edges set be $E = \{b_j | j = 1, 2, \dots, n\}$, where $(\max(n)=6)$.

Step3: For $\forall b_j \in E$, its vertexes are $v_1(\alpha_1, \beta_1)$ and $v_2(\alpha_2, \beta_2)$. Establish the linear equation of b_j according Eq. 3.

$$\begin{cases} au + bv + \eta = 0 \\ a = \alpha_2 - \alpha_1 \\ b = \beta_1 - \beta_2 \\ \eta = -(a\alpha_1 + b\beta_1) \end{cases} \quad (3)$$

Step4: For $\forall v_i \in V$, if v_i are all outside of b_j , it indicates that there has a separation plane between box A and box B , and proceed to the next step. Otherwise, go back to step 3.

Step5: End the algorithm and output the checking results.




For every pair of object candidates for collision, their OBB overlapping test is processed by using the above separation plane-finding method. Once any type of separation planes is found, the two OBBs will definitely not intersect. Thus, candidate pairs of objects whose OBBs do not overlap cannot intersect and are discarded.

6 Narrow phase: exact polygon intersection checking

At the broad phase, irrelevant objects are discarded, and a set of objects that possibly collide are determined. In this section, the narrow phase of the hybrid algorithm is responsible for determining exactly where the collision occurred with precise polygon intersect checking. For every pair of object candidates for collision, in order to precisely find whether two objects intersect, checking for collision of every pair of polygons of the corresponding objects is needed. For two objects which have m and n polygons, respectively, it requires $m \times n$ times polygon intersection checking. Intersecting two polygons is an expensive computational operation. In spite of most of the polygons filtering out at the broad phase, in order to gain faster speed, we make great effort to reduce the number of checking operations in the process for filtering polygons.

Through analyzing of the modular fixture, it can be found that the fixture elements are often densely assigned by many locating holes and screw holes, which are used for positioning and fixing when assembling. In our proposed system, the fixture element models are constructed in CAD software and then meshed. For those models, most of the polygons belong to the nonplane surfaces. As shown in Table 1, the holes contain most of polygons for modular fixture element models.

Table 1 Comparison of the number of composited polygons of several fixture elements

Element	Total Polygons	Polygons of planes	Polygons of holes	Others
	14604	7692	6912	0
	1116	558	558	0
	700	257	391	52

In the process of modular fixture assembly, besides the axis-hole assembly, the collision often occurs between outside faces of relevant objects. In our proposed system, axis-hole assembly adopts a kind of geometric constraint-based approach. That means the collision between axis and holes are not considered. So the polygons of holes are not the intersecting candidates. Thus, the number of polygon-checking operations is sharply decreased.

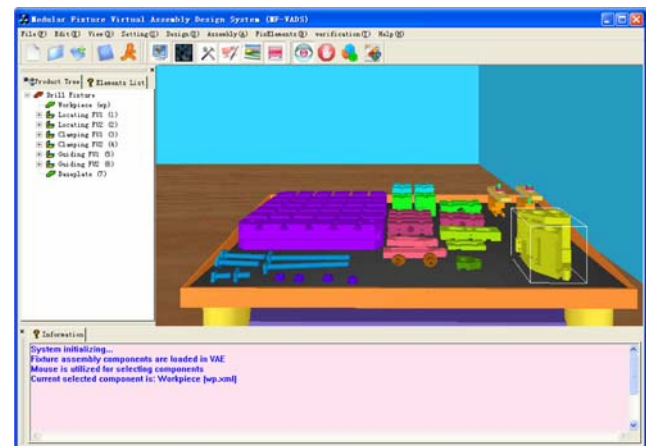
The intersection checking of two polygons is a common topic and has been solved by different methods. Most VR developing toolkits often provide this function, i.e., the WorldToolKit by sense 8 offers the function *Wtpoly_intersectpolygon*, which can test whether the given two polygons intersect. In application development, we just utilize this function directly, and the key point is to determine what polygons need to be checked. So the main work of this research is involved in the broad phase to filter out thousands of irrelevant polygons and to find out just a small number of pairs of candidate polygons for testing.

7 Case study with the implemented software

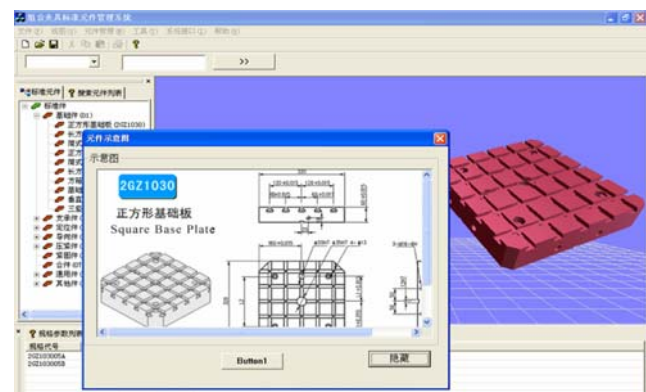
In this section, we explain the implementation issues of proposed CD algorithm and analyze its performance.

7.1 Implemented software

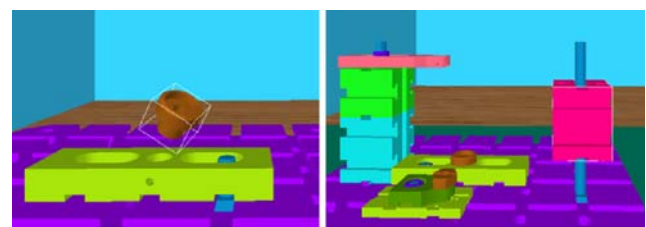
A useful system, named as Modular Fixture Virtual Assembly Design System (MF-VADS), has been implemented on a 2.4 GHz Pentium-IV PC, with 512 MB of RAM. The system is developed with Visual C++ 6.0 and WorldToolKit and run using hardware such as: Intergraph graphics station, 5DT dataglove, Flock of Birds. The GUI of MF-VADS is designed with the reference to the interface layouts of popular CAD software, which are widely used by the industry, as shown in Fig. 9. This system has the advantages of making the fixture design in a natural and instructive manner, providing better match to the working conditions, reducing lead time, and generally providing a significant enhancement of fixture productivity and economy. In VE, the designer can select suitable fixture elements



(a)



(b)



(c)

Fig. 9 Snapshots of the proposed MF-VADS system: **a** the main interface of proposed system, **b** management of modular fixture element database, **c** fixture elements assembly in virtual environment

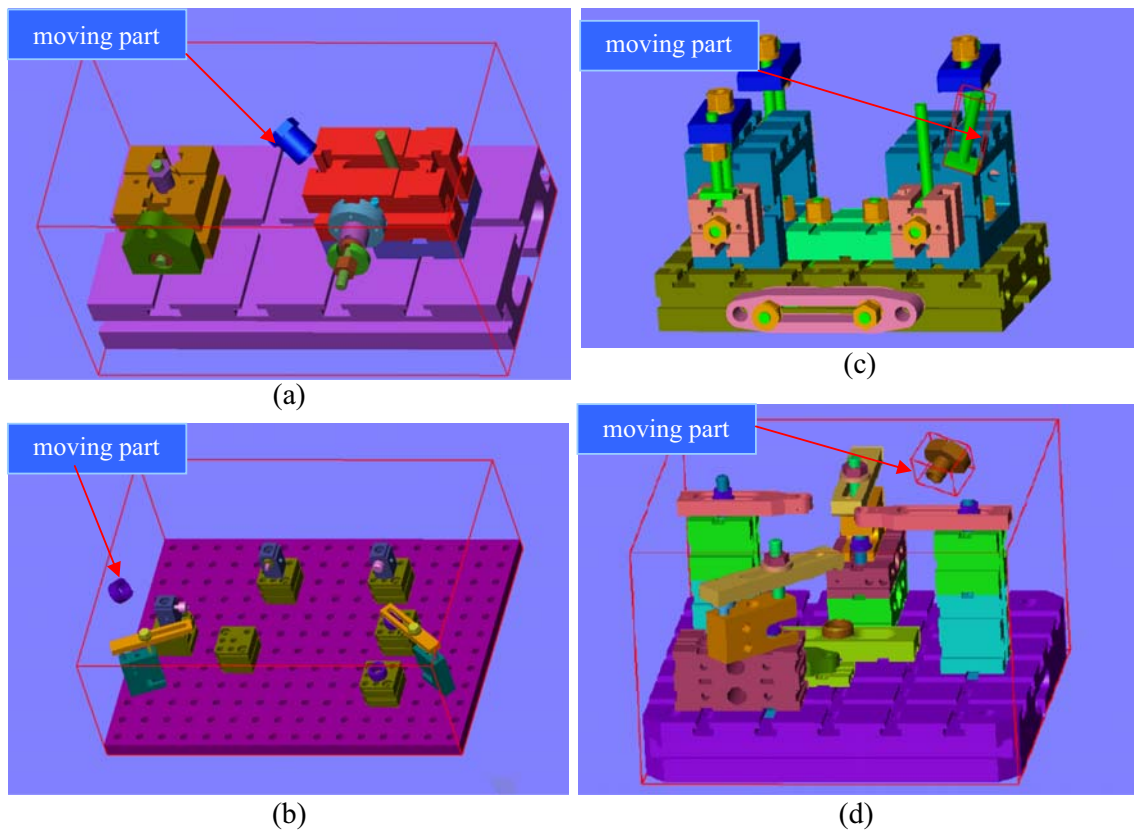


Fig. 10 Several CD test experiments: **a** a slot fixture system with static environment model containing 11,316 triangles and a moving nut containing 215 triangles, **b** a hole fixture system with static environment model containing 31,296 triangles and a moving locator containing 948 triangles, **c** a slot fixture system with static

environment model containing 18,004 triangles and a moving bolt containing 220 triangles, **d** a slot fixture system with static environment model containing 29,452 triangles and a moving rhombus pin containing 236 triangles

and put them together to generate a fixture structure just as “building block” process. Without physical fixture elements, he/she can try on different structure schemes and finally design a feasible fixture configuration to meet the fixturing function requirements.

7.2 Test cases and performance analysis

As shown in Fig. 10, four testing scenarios have been created to test our algorithm in comparison with other related algorithms like SOLID [29] and RAPID [20]. The

time cost for collision detection at each iteration and the average time cost for collision detections among 1,000 iterations are recorded.

Table 2 shows the performance statistics of our algorithm in comparison with SOLID and RAPID. For each test scenario, we recorded the number of the checked overlap bounding box pairs and triangle pairs during the checking process, respectively. We can see that when the actual colliding seldom occurs, our algorithm achieves better performance (can filter most potential number of bounding boxes and triangle pairs overlap checking at the broad

Table 2 Cost compare of collision detection algorithms

Actual collides	Scenario (a)		Scenario (b)		Scenario (c)		Scenario (d)	
	324		89		845		24	
	N_b	N_p	N_b	N_p	N_b	N_p	N_b	N_p
SOLID	5,273	1,271	115,273	40,271	107,412	33,715	67,412	13,715
RAPID	4,327	1,914	64,327	18,314	45,513	14,348	25,513	3,348
Ours	1,282	896	1,182	1,096	1,013	6,543	113	1,543

N_b is the number of bounding box overlap tests, N_p is the number of actual colliding triangle pairs

phase). Generally, in interactive virtual assembly application, often a few collisions may happen due to careless operations. According to the recorded overlap checking elements, we also calculated the average collision detection time for each experiment. The comparison chart of the CD time is shown in Fig. 11. We can reduce that the cost of proposed method does not increase remarkably in terms of the polygon count, while the other two algorithms increased a lot. Moreover, in scenario c (hole-based modular fixture system), we have observed more than three times performance improvement of our method over other two algorithms.

From the result of above two experiments, we can summarize the performance of proposed approach.

1. *Low memory requirements:* Actually, the hierarchical algorithms exploit spatial coherence by divide-and-conquer techniques to find all neighbor objects to speed up the collision detection procedure. Unlimitedly, these methods need very large auxiliary data structures to organize and store the related CD information. For instance, in hierarchical BV method, the data structure associated with hierarchical CD is bounding box trees. Each node in such a tree is associated with a subset of the primitives of the object, together with a BV that encloses this subset. So, in order to provide “optimal” BV trees, except the type, the size of single BV should be small enough to enclose their associated set of polygons as tightly as possible. In most universal algorithms, the hierarchies are built on polygon level. As a result, for a VE containing large numbers of polygons, the tree is often very complicated with thousands of nodes. For example, an OBB-node stores (at least) 18 floats. Therefore, even though these

methods can speed up the collision detection process, they have extremely high storage requirements.

In our approach, the LPM is built upon the object level. In modular fixture assembly design process, only dozens of elements are involved, so very little information need to be stored in LPM. Therefore, the memory is saved to a great extent comparing to the other memory-intensive universal algorithms.

2. *Low preprocessing time cost:* As mentioned before, most universal CD algorithms need auxiliary data structures to speed up checking procedure. However, construction of these data structures is time consuming. For some “static” scenarios (the environment models are fixed and the active objects are unchangeable), the data structures can be constructed in a preprocess stage; thus, the precomputation can be ignored.

But for virtual assembly application, once an element is assembled or disassembled, the “static” environment models are updated. Accordingly, the CD checking model needs to be restructured. Fortunately, in our approach, the LPM is simple and built on a few OBBs which are obtained directly from element models. In addition, due to the specialties of modular fixture, at most situations, the restructure of LPM is simple and fast by projecting the new added OBB onto corresponding axes. It can be concluded that the preprocessing of proposed approach is time-saving. Comparing to the universal algorithm, it is more feasible for practical virtual assembly application.

3. *Easily interfacing with VR applications:* The algorithm is fairly simple to implement and easy to integrate with popular VR-development tools. As we know, a scenegraph is the most common data structure used to represent objects in a VE. Objects are represented as nodes in the scenegraph, and the relationship between the objects and the environment is described by the edges. So the objects and their composite polygons are well represented and stored for VR visualization and interaction. Most universal algorithms need additional collision detection model to facilitate CD checking procedure. All polygons in the scenarios are assigned and recorded in such models. As a result, the polygons are stored twice, which is memory wasted.

In our proposed approach, the simple LPM is based on object level; that is, all polygons in VE are not reassigned and restored. The pretreatment phase and the broad filtering phase are independent of polygons. At the narrow phase, very potential candidates are selected for exact polygon–polygon intersection tests, which is the fundamental function provided by VR tools. So our approach can easily interface with any VR development tools.

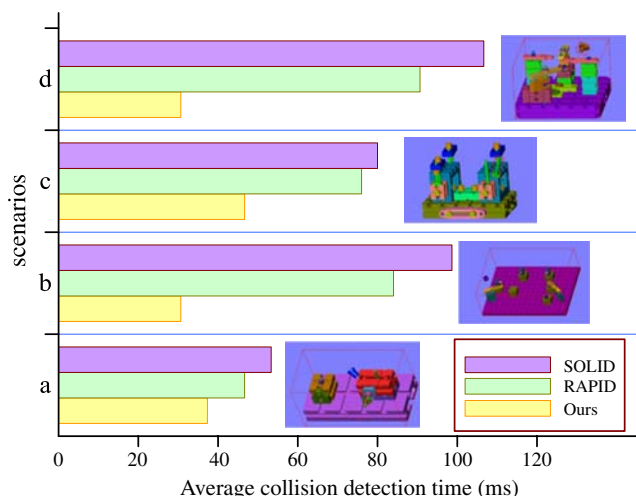


Fig. 11 The comparison chart of the CD time for test scenarios with different algorithms

8 Conclusions

This paper presents a “special” collision detection strategy for interactive modular fixture assembly design in a virtual environment. The strategy works by detecting collisions between two parts: a dynamic assembling element moving in the virtual environment and a static object defined as a collection of all other assembled objects in the space. Based on the characteristics of modular fixture and practical requirements, a new object level space partition methodology, achieved by automatically partitioning the checking space into cells in terms of OBBs of assembled elements, was presented. Such a preprocessing work has the advantages of simple information representation and can easily be constructed compared to other universal algorithms which often need large auxiliary data structures to organize and store the related information. In proposed CD method, a strategy of using two-phase collision detection is developed in order to make the approach more efficient. The broad phase is responsible for finding potential candidates for further exact polygons based collision tests. In this stage, those pairs of objects that cannot intersect are filtered out by silhouette overlap checking and further OBB intersect. In the narrow phase, the collision detection algorithm will calculate detailed intersection between geometrical meshes of the objects. From the results of several performance experiments, the above method was found to provide real-time collide detection with good performance at the expense of time cost and storage requirements.

This proposed method has been implemented in our VR-based modular fixture assembly design system and works well. Although this method is just applied in modular fixture design, we believe that our potential strategy is appropriate for coping with the CD problem of interactive graphics in the mechanical engineering applications. Because mechanical products often have regular shape compared to other scenarios, our future effort will concentrate on extending this method to other mechanical applications.

Acknowledgements We would like to acknowledge the support by the Research Fund of Chinese Postdoctoral under Grant (No.20070420870) and New Century Excellent Talents in University (No.NCET-08-0171).

References

- Bruno F, Caruso F, Li K et al (2008) Dynamic simulation of virtual prototypes in immersive environment. *Int J Adv Manuf Technol*. doi:10.1007/s00170-008-1736-6
- Yao YX, Xia PJ, Liu JS et al (2006) A pragmatic system to support interactive assembly planning and training in an immersive virtual environment (I-VAPTS). *Int J Adv Manuf Technol* 30:959–967. doi:10.1007/s00170-005-0069-y
- Liang JS, Pan WW (2006) Conceptual design system in a Web-based virtual interactive environment for product development. *Int J Adv Manuf Technol* 30:1010–1020. doi:10.1007/s00170-005-0151-5
- Vukelic D, Zuperl U, Hodolic J (2009) Complex system for fixture selection, modification, and design. *Int J Adv Manuf Technol*. doi:10.1007/s00170-009-2014-y
- Jiang-Liang H, Trappey AJC (2001) Computer-aided fixture design system for comprehensive modular fixtures. *Int J Prod Res* 39(16):3703–3725. doi:10.1080/00207540110060851
- Gaoliang P, Wenjian L (2006) A novel modular fixture design and assembly system based on VR. *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Beijing, China*. pp 2650–2655
- Kow TS, Kumar AS, Fuh JYH (2000) An integrated approach to collision-free computer-aided modular fixture design. *Int J Adv Manuf Technol* 16:233–242. doi:10.1007/s001700050151
- Wang Y, Wang Z, Gindy N (2009) Collision-free machining fixture space design based on parametric tool space for five-axis grinding. *Int J Adv Manuf Technol*. doi:10.1007/s00170-009-1939-5
- Fares C, Hamam Y (2005) Collision detection for rigid bodies: a state of the art review. *International Conference Graphicon 2005, Novosibirsk Akademgorodok, Russia*
- Moore M, Wilhelms J (1988) Collision detection and response for computer animation. In *Comput. Graphics (SIGGRAPH '88 Proc.)*, 22:289–298
- Noborio H, Fukuda S, Arimoto S (1989) Fast interference check method using octree representation. *Adv Robot* 3(3):193–212
- Naylor B, Amatodes JA, Thibault W (1990) Merging BSP trees yields polyhedral set operations. In *Comput. Graphics (SIGGRAPH '90 Proc.)*, Dallas, TX, USA, 24:115–124
- Klosowski JT, Held M, Mitchell JSB et al (1998) Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Trans Vis Comput Graph* 4(1):21–36. doi:10.1109/2945.675649
- Van den Bergen G (1997) Efficient collision detection of complex deformable models using AABB trees. *J Graphics Tools* 2(4):1–14
- Palmer IJ, Grimsdale RL (1995) Collision detection for animation using sphere-trees. *Comput Graph Forum* 14(2):105–116. doi:10.1111/1467-8659.1420105
- Gottschalk S, Lin MC, Manocha D (1996) OBB-tree: a hierarchical structure for rapid interference detection. *Proc. of SIGGRAPH '96*. pp. 171–180
- Ponamgi M, Manocha D, Lin M (1997) Incremental algorithms for collision detection between polygonal models. *IEEE Trans Vis Comput Graph* 3(1):51–75. doi:10.1109/2945.582346
- Lin M, Manocha D (1995) Fast interference detection between geometric models. *Vis Comput* 11(10):542–561. doi:10.1007/BF02434040
- Chung K, Wang W (1996) Quick collision detection of polytopes in virtual environments. *ACM Symposium on Virtual Reality Software and Technology*. pp. 1–4
- S. Gottschalk RAPID Robust and Accurate Polygon Interference Detection System. <http://www.cs.unc.edu/geom/OBB/OBBT.html>
- Figueiredo M, Fernando T (2003) An unified framework to solve the broad and narrow phases of the collision detection problem in virtual prototype environments. *Proceedings of the International Conference on Geometric Modeling and Graphics (GMAG'03) IEEE Computer Society*. pp 130–136
- Redon S (2004) Fast continuous collision detection and handling for desktop virtual prototyping. *Virtual Real* 8(1):63–70. doi:10.1007/s10055-004-0138-9

23. Wang QH, Li JR, Zhou RR (2006) Graphics-assisted approach to rapid collision detection for multi-axis machining. *Int J Adv Manuf Technol* 30:853–863. doi:[10.1007/s00170-005-0127-5](https://doi.org/10.1007/s00170-005-0127-5)
24. Hu W, Rong Y (2000) A fast interference checking algorithm for automated fixture design verification. *Int J Adv Manuf Technol* 16:571–581. doi:[10.1007/s001700070047](https://doi.org/10.1007/s001700070047)
25. Senthil Kumar A, Fuh JYH, Kow TS (2000) An automated design and assembly of interference-free modular fixture setup. *Computer-Aided Des* 32:583–596. doi:[10.1016/S0010-4485\(00\)00032-4](https://doi.org/10.1016/S0010-4485(00)00032-4)
26. Zachmann G (2000) Virtual reality in assembly simulation—collision detection, simulation algorithms, and interaction techniques. PhD dissertation, Darmstadt University of Technology, Germany
27. Ma W, Lei Z, Rong Y (1998) FIX-DES: a computer-aided modular fixture configuration design system. *Int J Adv Manuf Technol* 14:21–32. doi:[10.1007/BF01179413](https://doi.org/10.1007/BF01179413)
28. Greene N (1994) Detecting intersection of a rectangular solid and a convex polyhedron. In: Heckbert PS (ed) *Graphics Gems IV*, Academic Press, pp 74–82
29. FreeSOLID Software Library for Interference Detection. <http://www.win.tue.nl/~gino/solid/>