Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Controlling the contact levels of details for fast and precise haptic collision detection*

A Ram CHOI, Sung Min KIM, Mee Young SUNG‡

(*Department of Computer Science and Engineering, Incheon National University, Incheon, Korea*)

E-mail: choiaram11@inu.ac.kr; ksm3656@gmail.com; mysung@inu.ac.kr

Received Dec. 31, 2015; Revision accepted May 30, 2016; Crosschecked July 6, 2017

Abstract: For accurate and stable haptic rendering, collision detection for interactive haptic applications has to be done by filling in or covering target objects as tightly as possible with bounding volumes (spheres, axis-aligned bounding boxes, oriented bounding boxes, or polytopes). In this paper, we propose a method for creating bounding spheres with respect to the contact levels of details (CLOD), which can fit objects while maintaining the balance between high speed and precision of collision detection. Our method is composed mainly of two parts: bounding sphere formation and two-level collision detection. To specify further, bounding sphere formation can be divided into two steps: creating spheres and clustering spheres. Two-level collision detection has two stages as well: fast detection of spheres and precise detection in spheres. First, bounding spheres are created for initial fast probing to detect collisions of spheres. Once a collision is probed, a more precise detection is executed by examining the distance between a haptic pointer and each mesh inside the colliding boundaries. To achieve this refined level of detection, a special data structure of a bounding volume needs to be defined to include all mesh information in the sphere. After performing a number of experiments to examine the usefulness and performance of our method, we have concluded that our algorithm is fast and precise enough for haptic simulations. The high speed detection is achieved through the clustering of spheres, while detection precision is realized by voxel-based direct collision detection. Our method retains its originality through the CLOD by distance-based clustering.

Key words: Collision detection; Haptic rendering; Bounding sphere; Clustering; Contact levels of details (CLOD)
http://dx.doi.org/10.1631/FITEE.1500498 **CLC number:** TP391

1 Introduction

Collision detection is a crucial issue that arises whenever we interact with virtual objects. If there is no collision detection, virtual objects penetrate each other and we will not be able to move them. Many algorithms have been proposed in decades to accelerate collision detection. However, there are still some open challenges such as the extremely high updating frequencies required for haptic rendering.

The core technology of the haptic virtual reality system is haptic rendering. By rendering, we refer to the process in which the desired sensory stimuli are imposed on the user to convey information about a virtual haptic object (Salisbury *et al*., 2004). The precise sensory feedback necessitates precise collision detection between a haptic device and a virtual haptic object. In addition, an ideal haptic rendering requires an update rate of 1000 Hz.

Recently, haptic rendering technology has been rapidly developing, and many experimental systems are being created. The main challenge is to construct a bounding volume that can fit (enclose or pack) the underlying geometry as tightly as possible.

Among many collision detection methods, we are interested in methods based on bounding spheres. They are often used to accelerate the computations in computer graphics, haptics, robotics, and computational geometry, because they are effective in collision detection. Consequently, making efficient

‡ Corresponding author

* Project supported by Incheon National University Research, Korea (No. 20120238)

ORCID: Mee Young SUNG, http://orcid.org/0000-0001-8078-624X

computations to form boundaries of virtual objects has become a very significant topic.

Collision detection involves identifying the interpenetration of objects, which can be classified into three cases in haptic simulations: (1) collisions of objects, (2) haptic probing between an object and a haptic interface, and (3) self-detection of collision such as cloth-folding.

In this study, we are interested only in the second case, and the objective of this study is to develop an efficient collision detection method for haptic probing between an object and a haptic interface. The method that we present is to effectively construct optimal bounding spheres through clustering, which can fit the target virtual object as tightly as possible. This work also aims to fulfill the following criteria (Weller, 2013): (1) tight fit of the underlying geometry; (2) provision of fast collision tests; (3) no use of too much memory; (4) possibility of being built automatically; (5) provision of fast enough collision enough for haptic rendering.

2 Related works

Collision detection refers to the computational aspect of determining whether two objects have collided; collision response refers to the simulation of what happens when a collision is detected. So far, many important methods have been developed for collision detection (Moore and Wilhelms, 1988; Quinlan, 1994; Hubbard, 1995; 1996; Gottschalk *et al*., 1996; Ruspini *et al*., 1997; van den Bergen, 1997; Klosowski *et al*., 1998; Larsson and Akenine-Möller, 2001; Bridson *et al*., 2002; Bradshaw and O'Sullivan, 2004; James and Pai, 2004; Teschner *et al*., 2005; Barbič and James, 2008; Weller and Zachmann, 2009; Zeng and Zheng, 2012; Vlasov *et al*., 2013).

2.1 Bounding volumes

To quickly detect collision between an object and a haptic pointer, a bounding volume of the object is necessary. In computational geometry and computer graphics, a bounding volume for a set of objects is a closed volume that can completely contain the union of objects in the set. Since simpler volumes normally permit simpler ways to test for overlap, the method of using easier figures is used to improve the efficiency of geometrical operations.

To obtain bounding volumes of complex objects, a common way is to break the objects down using bounding volume hierarchies (BVHs). Bounding volumes for building hierarchies include bounding spheres (BSs) (Quinlan, 1994; Hubbard, 1995), axis-aligned bounding boxes (AABBs) (van den Bergen, 1997), oriented bounding boxes (OBBs) (Gottschalk *et al*., 1996), and discrete orientation polytopes (k-DOPs) (Klosowski *et al*., 1998). The basic idea behind these hierarchies is to organize a complex object in a tree-like structure where the root comprises the whole object and each leaf contains a smaller subpart. Collision detection is then carried out corresponding to the hierarchies of the bounding volumes.

Wrapping objects in bounding volumes and performing collision tests on them before testing the target geometry itself simplify the tests and can significantly improve performance. By arranging the bounding volumes into a hierarchy, the time complexity (the number of tests performed) can be reduced to logarithms of the number of objects. With such a structure in bounding volumes, children in the hierarchy do not have to be examined for collisions if their parent volumes have not collided. Bounding volumes are typically based on space decomposition by tree data structures, such as binary space partition (BSP) trees (Thibalt and Naylor, 1987), *k*-d trees (Bentley, 1975), octrees (Samet, 1984), R*-trees (Beckmann *et al*., 1990), and grids (Garcia-Alonso *et al*., 1994).

Octrees are often used to partition a 3D space into eight octants via recursive subdivision. However, a major limitation of such an approach is that the leaf objects cannot usually be tightly enclosed because the number of leaf nodes is always fixed to eight. One method to resolve this problem is to divide the space using the medial axis (topological skeleton). The motivation for this approach is from Blum's medial axis (Blum, 1967), which can be interpreted as a 'skeleton' of a 2D object. The medial axis or Voronoi skeleton of a polygon is the set of two or more neighboring points on the polygon's boundary (Fig. 1a). A more technical definition involves the locus of points equidistant from two sides of the object. The 3D version is called the medial axis surface. This structure contains surfaces rather than lines, but

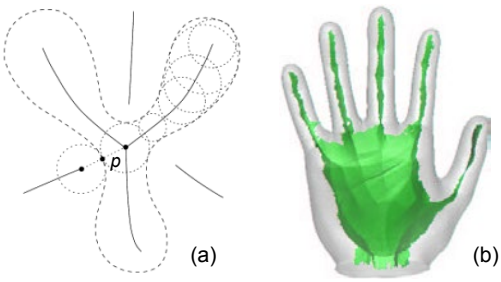

Fig. 1 Medial axis (surface) in two (three) dimensions and a skeletal representation of a simple geometrical 3D structure: (a) a curve (dashed), some medial balls (dotted), and the medial axis (solid) in 2D space; (b) an example of a medial axis surface in 3D space (Dey and Zhao, 2004)

it remains analogous to a skeleton (Hubbard, 1996). Our study is based on this medial axis concept.

2.2 Bounding spheres

There are several fast and simple algorithms for constructing a bounding sphere volume with high practical value in real-time computer graphics applications. Since a sphere can be represented with its center and radius, it can be tested for collision with each other very quickly by simple checks on the radii. As two spheres intersect, the distance between their centers should not exceed the sum of their radii. This property makes bounding spheres applicable to objects in any dimension.

Ruspini *et al*. (1997) generated hierarchical bounding spheres using a balanced binary tree and a multilevel approach. In their study, 'high-level detection' corresponds to the 'fast detection' in our proposed method, while 'low-level detection' corresponds to 'precise detection' in ours. The experiments in Ruspini *et al*. (1997) were conducted with a top-down algorithm, whereas ours were conducted with a bottom-up algorithm for sphere formation. The bottom-up approach can generate more precise and tighter bounding spheres because it starts from the terminal polygons.

Alternatively, Weller and Zachmann (2009) proposed a top-down approach, whereas ours is a bottom-up one. Their study uses a voxel-based algorithm and generates inner spheres, whereas ours generates outer spheres. The approach of wrapping with inner spheres often fails to envelop all the boundary polygons, but our outer spheres approach can do so successfully.

Most of these methods can be directly applied to deformable objects. However, the major drawback of hierarchical collision detection algorithms is that the hierarchy has to be updated after every deformation. Consequently, much attention in the field has been devoted to developing hierarchies for easy computations.

3 Methods

3.1 Overview

Our work uses the bounding spheres model because of its simplicity and speed for intersection detection. There are two different strategies to build bounding volumes: top-down node-split approach (Roussopoulos and Leifker, 1985; Ruspini *et al*., 1997; Weller and Zachmann, 2009) and bottom-up node-grouping scheme (Goldsmith and Salmon, 1987). The top-down construction is to recursively split a set of object primitives until a threshold is reached. The user can split the objects in his/her own way that can produce adequate bounding volume hierarchies. The bottom-up approach starts with primitives that represent the objects and then fits a bounding volume to the given number of primitives. It groups primitives and bounding volumes recursively and stops when there is a single bounding volume at the hierarchy level.

The top-down strategy is used mostly for collision detection, but it has limitations on automatic construction of bounding volumes. Therefore, we use the bottom-up approach with some complementary measures, which are the clustering of spheres and voxel-based direct collision detection. Thanks to the recent advancement of computer technologies such as general-purpose graphics processing units (GPGPU) and multicore processing, some voxel-based detection techniques are now partially possible in common computing environments (McNeely *et al*., 1999; Aragón and Molinari, 2014).

Before describing our method, some terms are to be defined: 'centroid' is the center of gravity of a triangle mesh; 'central axis' is a set of midpoints of the line that connects the opposite centroids of the triangles (similar to the medial axis); 'center (of the sphere)' is the center of a bounding sphere; 'clustering cube' is a cube whose side length is the clustering

distance in three dimensions.

Our fast and precise collision detection consists of two methodologies:

1. Bounding spheres formation

First, spheres are created according to the following three steps:

(1) Find centroids of all triangular polygons of the target object.

(2) Generate the central axis (skeleton) of the target object by generating the midpoints of all the connecting lines between the centroids of opposite meshes (the set of all midpoints forms a skeletal structure and becomes the central axis of an object).

(3) After bisecting the connecting lines with the midpoints, repeat the bisecting process on the right-hand and left-hand segments of the lines. Then generate two new centers of spheres at the right and left midpoints (i.e., one-fourth and three-fourths points of the original connecting line).

Second, the created spheres are clustered according to the following two steps:

(4) Discard meaningless centers that contribute not much to skeleton formation.

(5) Cluster centers with contact levels of details (CLOD) through single-link clustering with variable distance (measured with the side length of a clustering cube whose volume is defined according to the CLOD).

2. Two-level collision detection

First, sphere are fast detected:

(1) Calculate the distance vector between the centers of two spheres.

(2) Check for the overlap condition between the haptic pointer (considered as a very small sphere) and each bounding sphere.

Then, the spheres are detected precisely:

(3) Project the haptic pointer onto the mesh.

(4) Take the outer (cross) and inner (dot) products of vectors (one is the projection of the haptic pointer and the other is the normal vector of the mesh) to check if they are on the same triangular mesh plane.

(5) Check the distance between the haptic pointer (considered as a very small sphere) and each bounding sphere.

3.2 Bounding spheres formation

A bounding sphere volume is constructed by two steps: creating spheres and clustering spheres. Before creating spheres, preprocessing with the medial axis is necessary to represent an object in skeletal form. The medial axis guides the optimization process for matching the spheres to the object shape.

3.2.1 Creating spheres

Among the four types of triangle centers (centroid, circumcenter, orthocenter, and incenter), we use the centroid (center of gravity) of a triangular polygon to represent its center. The centroid of a triangle is the intersection of the lines connecting the midpoints of each side of the triangle to the opposite vertices. The line segments created to find the centroid are called the medians, which are shown in Fig. 2 with dotted lines. Note that regardless of the shape of the triangle, the centroid will always be inside the triangle.

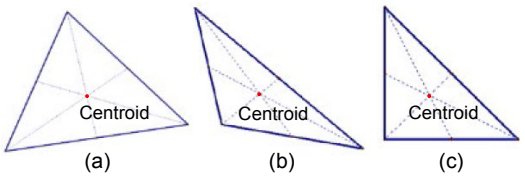

Fig. 2 Centroids of different types of triangles: (a) acute triangle; (b) obtuse triangle; (c) right triangle

Initially, the centroid of each polygon mesh is generated. Then the ray tracing method is applied to find the opposite mesh based on the fact that the angle between the normal vectors of the opposite and target meshes is π. The inner product (cosine product) of the two vectors should give −1 (recall that cos π=−1), so the opposite mesh can be obtained by finding whose normal vector gives the smallest inner product value (≈−1) with that of the originally selected mesh. The threshold for the initial search is set to −0.99. If no opposite mesh is found, repeat the process with the threshold increased by 0.01 (i.e., −0.98), and so forth.

In the Euclidean space, the inner product (or dot product) of two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is defined as

$$\boldsymbol{a}\cdot\boldsymbol{b}=\|\boldsymbol{a}\|\|\boldsymbol{b}\|\cos\theta,$$

where θ is the angle between vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ within the plane containing them (i.e., $0\le\theta\le\pi$), and $\|\boldsymbol{a}\|$ and $\|\boldsymbol{b}\|$ are the magnitudes of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively. In addition, the outer product (or cross product) is defined by

$$\boldsymbol{a}\times\boldsymbol{b}=\|\boldsymbol{a}\|\|\boldsymbol{b}\|\sin\theta\,\boldsymbol{n},$$

where $\boldsymbol{n}$ is a unit normal vector orthogonal to the plane containing $\boldsymbol{a}$ and $\boldsymbol{b}$.

Afterward, the midpoints of the line segments connecting the centroids of opposite meshes are generated. The set of these midpoints forms the central axis of the object. Then, two new centers of bounding spheres are created at the middle of the right and left segments of the initial connecting line segment between centroids, respectively (i.e., repeat the generating process by bisecting the remaining halves of the line). The reason of bisecting once again is to avoid generating an excessively large sphere, which can get partially out of the underlying model (Fig. 3).

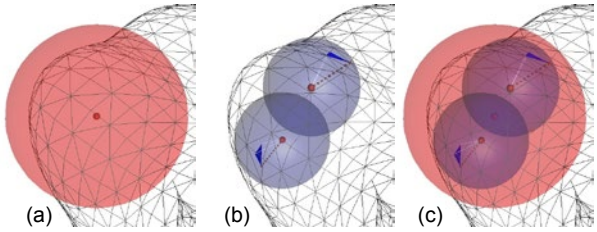

Fig. 3 Example of generating two bounding spheres by bisecting the line segment between centroids twice: (a) an excessively large bounding sphere generated from the initial midpoint between two opposite meshes; (b) two bounding spheres generated from the two newly placed centers on the bisected central axis, fitting the model more tightly; (c) simultaneous representation of (a) and (b) (References to color refer to the online version of this figure)

Once the two centers of bounding spheres are positioned on each line segment, the bounding spheres can be determined by calculating their radii. The distance between the center of a sphere and the furthest vertex of the nearest polygon (indicated by dotted lines in Fig. 4) becomes the radius of the sphere. Fig. 4 illustrates an example of the determination of the radius.

3.2.2 Clustering spheres

Grouping nearby points helps the extraction of meaningful centers. We use the single-link cluster (nearest-neighbor) method (Sibson, 1973), which is one among the simple and efficient methods of agglomerative hierarchical clustering.

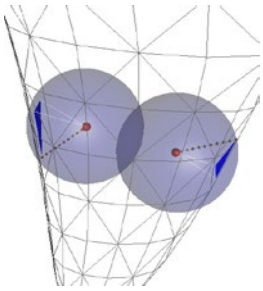

Fig. 4 Determination of the radius: a line segment between the center of the sphere and the most distant vertex of the nearest polygon (indicated by dotted lines)

The generated centers belong to each bounding sphere. To avoid unnecessary calculations, we group all centers of spheres whose distances are less than the chosen cubic distance (e.g., 0.02) (Fig. 5). In other words, we find the centers of spheres lying within a cube, whose side length is the clustering distance in three dimensions.

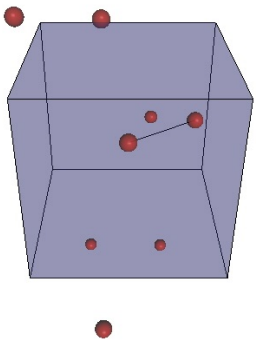

Fig. 5 Grouping centers within a clustering cube

Discarding, which removes some meaningless points on the model, had been used at the initial stages of the computation. However, we avoid this process in this study because of the calculation overhead.

Figs. 6a and 6b compare the results of bounding spheres with or without ‘discarding’, respectively. ‘Discarding’ induces some errors that cannot be ignored. As shown in Fig. 6a, some segments are polygons, which are not correctly included in bounding spheres. In addition, the additional computation time for ‘discarding’ is relatively high, whereas the time needed for sphere generation without ‘discarding’ is trivial. For example, for the tooth model without clustering in Fig. 6b, the computation time for ‘discarding’ was 4600 ms and the time needed for sphere generation was 130 ms with ‘discarding’, while it was

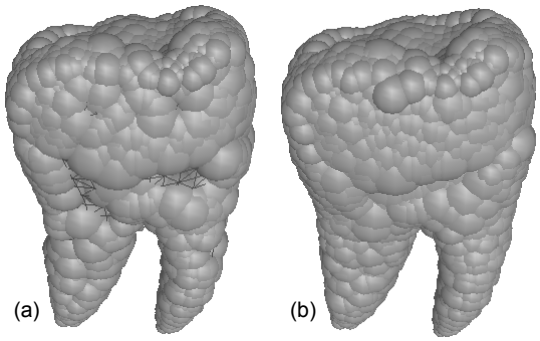

Fig. 6 Sphere generation with discarding with polygons (2510 spheres) (a) and sphere generation without discarding (3072 spheres) (b)

170 ms without 'discarding'. For these reasons, 'discarding' is omitted in our study.

Mathematically, the distance $D(X, Y)$ between clusters X and Y with single-link clustering is

$$D(X,Y)=\min d(x,y),\ x\in X,\ y\in Y,$$

where X and Y are any pair of sets that include elements identified as clusters, and $d(x, y)$ denotes the distance between elements x and y ($x\in X$, $y\in Y$).

In the beginning of the clustering process, each polygonal element is in a cluster of its own. The clusters are then sequentially combined into larger clusters until all elements end up being in the same cluster. At each clustering step, the two nearest clusters are combined. In single-link clustering, the link between two clusters is determined by a single closest pair of elements contained in two different clusters. The shortest of these links combines small clusters in which the pair is included into a larger cluster. An example of clustering of nearby centers of spheres is shown in Fig. 7.

Fig. 8 presents an example of bounding spheres formation for a tooth model. Fig. 8a marks the selected centers in red points; Fig. 8b shows the clustered centers (clustering distance=0.02) in red points; Fig. 8c presents the spheres transparently; Fig. 8d demonstrates the spheres in opaque. Algorithm 1 outlines the construction of bounding spheres.

3.3 Two-level collision detection

Our two-level collision detection contains two steps: fast detection of spheres and precise detection in spheres.

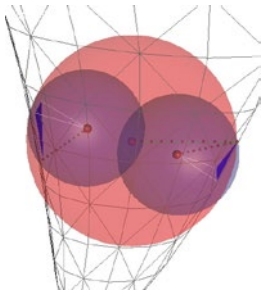

Fig. 7 Clustering of two neighboring centers of spheres

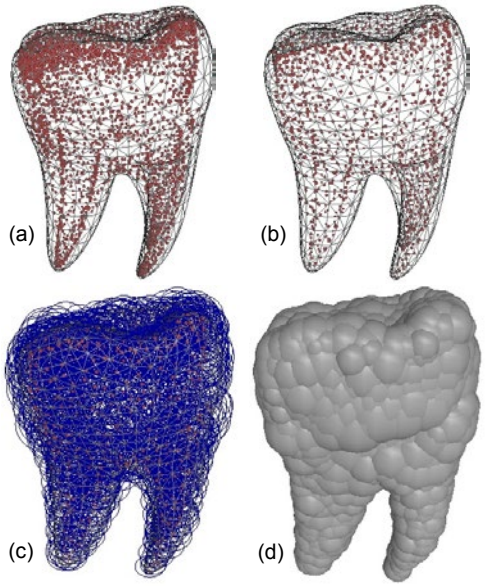

Fig. 8 Example of clustered centers for a tooth model: (a) selected centers in red points; (b) clustered centers (clustering distance=0.02) in red points; (c) transparent spheres; (d) spheres in opaque (References to color refer to the online version of this figure)

3.3.1 Fast detection of spheres

Note that the haptic pointer can be considered as a very small sphere. Therefore, collision detection between the pointer and the bounding sphere is equivalent to detecting the collision of two spheres.

To detect the collision, we use a simple calculation for detecting the overlap between two spheres, which takes the sum of the radii, and compare them with the distance between the centers of the two spheres. If the distance is smaller than the sum, a collision is detected.

3.3.2 Precise detection in spheres

A more refined collision detection scheme is required because the constructed structure of the

Algorithm 1 Constructing bounding spheres

```
for each triangle_i do
    GPU: // calculate the centroid of triangle_j
    triangle_j.centroid.x=(triangle_j.vertex0.x
        +triangle_j.vertex1.x+triangle_j.vertex2.x)/3
    triangle_j.centroid.y=(triangle_j.vertex0.y
        +triangle_j.vertex1.y+triangle_j.vertex2.y)/3
    triangle_j.centroid.z=(triangle_j.vertex0.z
        +triangle_j.vertex1.z+triangle_j.vertex2.z)/3
end for
for each triangle_i do
    GPU: compute the distance between the centroids of
        triangle_i and triangle_j
    GPU: compute the cosine of the angles between the
        normal vectors of triangle_i and triangle_j
    GPU: find the opposite triangle_j (whose cosine value
        is closest to −1)
    Calculate the central axis and the double central axis
    Create two centers of bounding spheres
end for
for each triangle_i do
    for each triangle_j do
        if the distance from center_i to center_j is smaller
            than the clustering distance then
            Generate clustered_center_i using single-
            link clustering
        end if
    end for
end for
for each clustered_center_i do
    Create sphere_i for each clustered_center_i with radius_i
    equal to the distance from clustered_center_i to the
    most distant vertex of the nearest triangle_i
end for
```

bounding spheres is not the exact shape of the object. Once a collision between the haptic pointer and a bounding sphere is detected, precise detection in spheres is performed between the pointer and each polygon mesh point inside the colliding sphere.

Our spheres are represented in the following data structure:

```
typedef struct axisInfo {
    vector3D axispoint;
    float dist;
    int index[MAX_NUM_POINTS];};
```

Our bounding spheres are generated to wrap all polygon meshes of the base geometry, so that the data structure includes an integer array, whose elements correspond to each polygon mesh point in the sphere.

In Fig. 9, the large circle in blue indicates the bounding sphere, and the small circle in gray represents the haptic pointer. The red circle corresponds to the center of the blue circle. The refined mesh detection within the sphere is carried out in the following steps:

1. Projection: the haptic pointer is projected onto one of the meshes in the sphere, and a mesh detection process is conducted on every mesh for discerning which one contains the projected point. To exclude possible errors, the projected point is checked as to whether it is inside the mesh or not. It is always possible that the pointer can be projected to a point outside the mesh, which is not a collision.

2. Vector products: the inclusion can be examined by taking the product of vectors. If two vectors are on the same triangular mesh plane, the results of both the outer product and inner product of the two vectors are positive. Fig. 10 explains this with an example. Among the three vectors in the figure, **at** is a vector that starts from a vertex of a triangle (A) to the projected point of the haptic pointer (T). **ab** and **ac** are vectors from vertex A to the other two vertices (B and C). Recall that the sum of the angles within a triangle is 180°. Then for a given vertex A, we can take two different cross products (**ab**×**at** and **at**×**ac**) with each vector, and these results will be positive if the pair is on the same triangle. If all four points are on the same triangle, then the inner product will also be positive.

3. Distance check: if the projected mesh is found, it is probable that the haptic pointer collides with that mesh. In this case, the calculation for determining the overlap condition between the sphere and the pointer is simply done by checking whether the distance

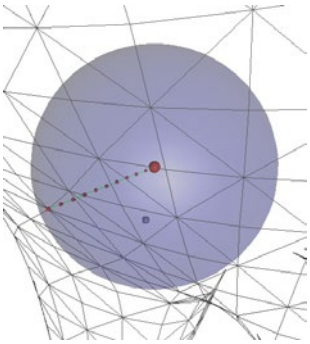

Fig. 9 Example of precise collision detection in a sphere (References to color refer to the online version of this figure)

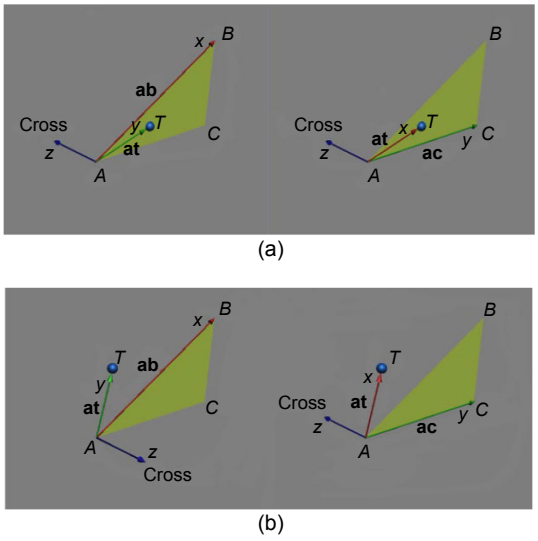

Fig. 10 Illustration of vector products: (a) the cross product lies on the same triangular plane if the projected point of haptic pointer is inside of the triangular mesh; (b) the cross product lies on a different plane if the projected point is outside of the triangular mesh

between the center and the pointer is smaller than the radius. Our two-level collision detection method is summarized in Algorithm 2.

4 Implementation

Phantom Desktop 3.0 (Fig. 11a) was used as a haptic device. It provides three-degree-of-freedom force feedback on the directions of x, y, and z and six-degree-of-freedom position and perception, which are three positions in x, y, and z coordinates and three rotational inertias (pitch, roll, and yaw with ±3% linearity potentiometers, Fig. 11b). The computing environment for our experiments is shown in Table 1.

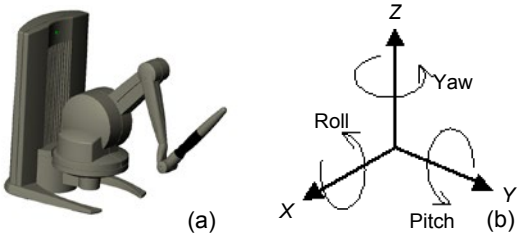

Fig. 11 Phantom Desktop 3.0 (a) and six degrees of freedom (b)

Algorithm 2 Two-level collision detection

```
GPU: for each sphere_i do
    difference=hapticCursor.pos−sphere_i.pos
    distance=difference.length
    sumOfRadius=hapticCursor.radius+sphere_i.radius
    if distance<sumOfRadius then
        collision detected
    end if
end for
T=the coordinates of the center point of the haptic cursor
    projected to triangle_i in the sphere
index=−1  // index of triangle_i in the sphere
isPositive=false  // variable for checking whether all three
// vertices of triangle_i are on the same mesh plane or not.
// vertex_a, vertex_b, and vertex_c indicate three vertices of
// triangle_i
for each triangle_i do
    for each triangle_j do
        isPositive=true
        vector_1=(vertex_a, vertex_b)
        vector_2=(vertex_a, vertex_c)
        vector_3=(vertex_a, T)
        crossResult_1=vector_1×vector_3
        crossResult_2=vector_3×vector_2
        if (crossResult_1 · crossResult_2)<0 then
            isPositive=false
            break
        end if
    end for
    if isPositive then
        index=i
        break
    end if
end for
distance=the distance between T and the center of the
    haptic cursor
if distance<the radius of the cursor then
    Colliding to triangle_i occurs
end if
```

Table 1 Computing environment for experiments

Component	Specification
Operating system	Windows 7 ultimate 32-bit
CPU	Intel Xeon 5160 Dual Core 3.0 GHz
RAM	3 GB
GPU	Nvidia GeForce GT 520
Haptic device	Phantom Desktop 3.0 • Force feedback: x, y, and z directions • Position: x, y, and z • Perception: pitch, roll, and yaw (±3% linearity potentiometers)

The algorithm is implemented using a GPGPU. GPUs are data stream processors that can run one kernel function in parallel on a set of data streams, and our GPU model can be summarized as follows:

1. GPU stream processors are made of 32 blocks. Each block consists of 512 threads at maximum.

2. A block is assigned to process each sphere and operates in parallel. A maximum of 30 blocks can be executed in parallel.

3. The 511th thread of a block (Fig. 12) is used to examine whether the haptic pointer is inside the assigned sphere or not. All other threads in the block also conduct checks in parallel on all of the polygon mesh points of that sphere.

4. Finally, precise detection in spheres is performed on the polygon meshes where the projection vector of the haptic pointer is included.

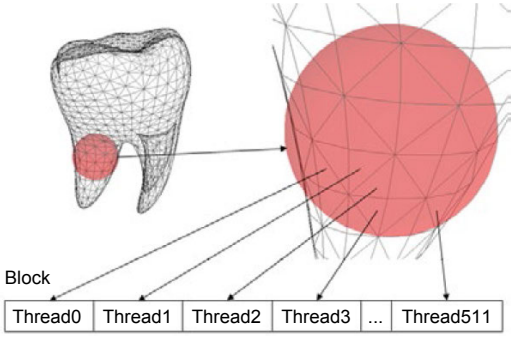

Fig. 12 Thread model for a GPU block

5 Results

Experiments were conducted to examine the usefulness of our method on three different graphic models (i.e., tooth, clover, and hat models). The performance was analyzed by investigating how many spheres can be reduced while maintaining the tightness of the boundary for each model.

To validate our algorithm, we introduce two performance criteria:

Criterion 1 The bounding sphere reduction ratio is the ratio of the number of bounding spheres after clustering the spheres to that before clustering.

Criterion 2 The collision detection time is the total time taken for checking the distances between the haptic pointer and all the spheres.

The update rate should be higher than 1000 Hz under various contact scenarios (i.e., either a single contact or a multi-point contact case). Real-time graphics applications have display update frequency requirements somewhere between 20 and 30 frames/s. On the other hand, the update rate of haptic simulations must be as high as 1000 updates/s to maintain a stable system. This rate varies with the spatial frequency and stiffness of displayed forces and with the speed of motion of the user. In addition, because human tactile sensation is sensitive to vibrations with frequencies higher than 500 Hz, changes in force at even relatively high frequencies are also detectable. This value is important to decide the optimal value of distance for clustering. (Colgate and Brown, 1994; Gregory *et al*., 2005).

5.1 Contact levels of details

For single-link clustering, variable distances (d=0–0.08) are applied for the CLOD (Otaduy and Lin, 2003) and the selection of the distance depends on the complexity of base geometry. Taking the tooth model as an example, Fig. 13 presents the constructed bounding spheres corresponding to each level of CLOD, and Table 2 shows an example of CLOD with variable distances (0–0.08).

5.2 Optimum clustering distance

The optimum distance can be determined according to numerous criteria, e.g., the characteristics of models, application requirements, and computing power. The analyses of the reduction ratios of centroids with different clustering distances are presented in Table 2 and Fig. 14 for the tooth model, Table 3 and Fig. 15 for the clover model, and Table 4 and Fig. 16 for the hat model. The optimum clustering distance can be determined by considering performance criterion 2 concerning the collision detection time; i.e., the update rate should be higher than 1000 Hz.

5.3 Comparison with other algorithms

The octree and Hubbard (Hubbard, 1995; 1996) algorithms, along with the sphere tree construction toolkit (Bradshaw, 2003), were chosen to be compared with our algorithm. Figs. 17–19 show the results for the tooth, clover, and hat models, respectively. As shown, the octree method shows a very

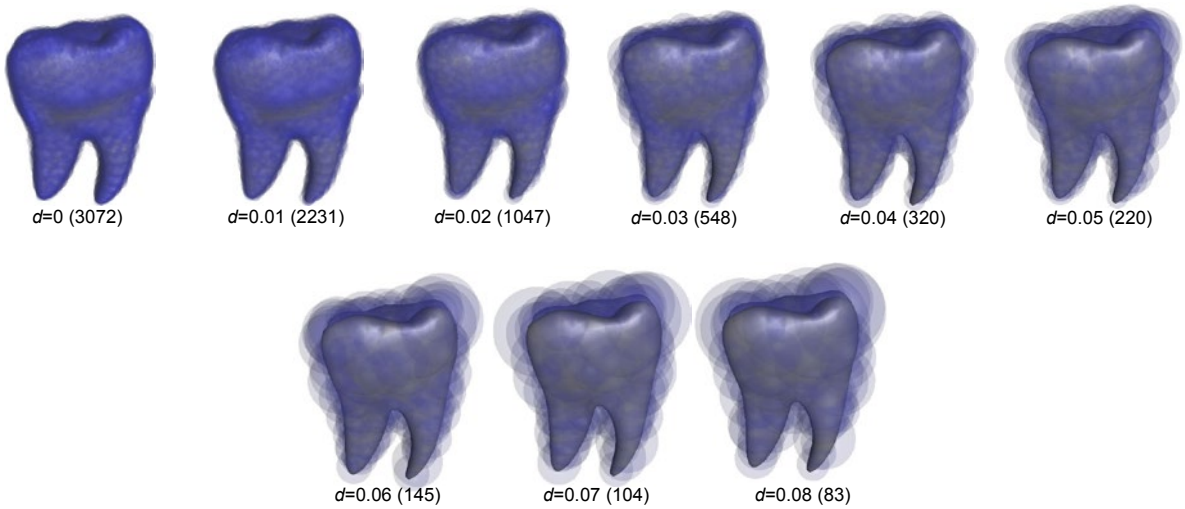

Fig. 13 Contact levels of details controlled by variable distances (*d*=0–0.08) of single-link clustering, where *d* indicates the side length of the clustering cube, and the figures in the parentheses denote the numbers of spheres

Table 2 Analysis of single-link clustering with different clustering distances for the tooth model

Clustering distance	Number of centers	Step reduction ratio of centers	Total reduction ratio of centers	Collision detection time (ms)			Update rate (Hz)
				Fast detection	Precise detection	Total	
0	3072	0%	0%	2.037	0.118	2.155	491
0.01	2231	27%	27%	1.521	0.123	1.644	631
0.02	1047	53%	66%	0.677	0.133	0.810	1476
0.03	548	48%	82%	0.336	0.160	0.496	2789
0.04	320	42%	90%	0.213	0.169	0.382	4887
0.05	220	31%	93%	0.139	0.227	0.366	7097
0.06	145	34%	95%	0.102	0.247	0.349	9762
0.07	104	28%	97%	0.071	0.378	0.449	14 509
0.08	83	20%	97%	0.062	0.425	0.487	16 483

Table 3 Analysis of single-link clustering with different clustering distances for the clover model

Clustering distance	Number of centers	Step reduction ratio of centers	Total reduction ratio of centers	Collision detection time (ms)			Update rate (Hz)
				Fast detection	Precise detection	Total	
0	5288	0%	0%	3.320	0.122	3.442	340
0.01	2176	59%	59%	1.412	0.131	1.543	757
0.02	858	61%	84%	0.510	0.151	0.661	1732
0.03	514	40%	90%	0.326	0.155	0.481	2880
0.04	384	25%	93%	0.240	0.205	0.445	4325
0.05	224	42%	96%	0.142	0.220	0.362	6850
0.06	153	32%	97%	0.097	0.259	0.356	9463
0.07	113	26%	98%	0.075	0.414	0.489	14 237
0.08	84	26%	98%	0.058	0.609	0.667	17 325

Table 4 Analysis of single-link clustering with different clustering distances for the hat model

Clustering distance	Number of centers	Step reduction ratio of centers	Total reduction ratio of centers	Collision detection time (ms)			Update rate (Hz)
				Fast detection	Precise detection	Total	
0	20 164	0%	0%	12.908	0.118	13.026	76
0.01	5322	74%	74%	3.365	0.200	3.565	295
0.02	1790	66%	91%	1.146	0.273	1.419	1031
0.03	1078	40%	95%	0.609	0.322	0.931	1637
0.04	641	41%	97%	0.420	0.432	0.852	2439
0.05	421	34%	98%	0.280	0.539	0.819	3671
0.06	297	29%	98%	0.202	0.612	0.814	5245
0.07	226	24%	99%	0.150	0.799	0.949	6754
0.08	185	18%	99%	0.120	0.937	1.057	8216

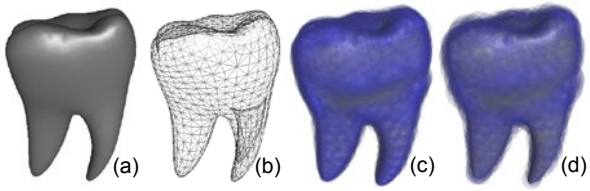

Fig. 14 Depiction of a tooth: (a) model; (b) polygons; (c) bounding spheres without clustering; (d) bounding spheres with the optimal clustering distance of 0.02

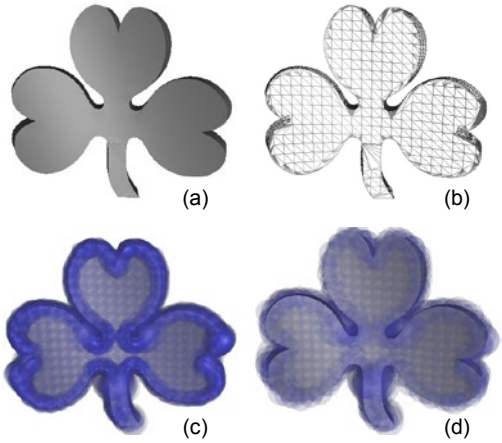

Fig. 15 Depiction of a clover: (a) model; (b) polygons; (c) bounding spheres without clustering; (d) bounding spheres with the optimal clustering distance of 0.02

limited performance, while the Hubbard method and our method present better tightness for the same number of bounding spheres. However, the Hubbard method fails to entirely wrap all the polygons of the model, leaving the lines of polygons, as shown by the blue lines in Figs. 17d, 18d, and 19d. Our method successfully includes all polygon meshes of the model within the generated bounding spheres.

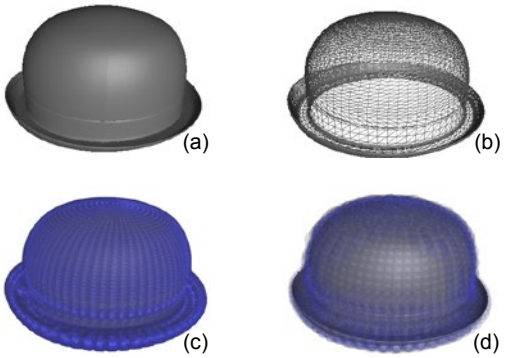

Fig. 16 Depiction of a hat: (a) model; (b) polygons; (c) bounding spheres without clustering; (d) bounding spheres with the optimal clustering distance of 0.02

5.4 Results depending on the number of polygons

From the experiments, we find some interesting results regarding the relationship between the bounding sphere reduction ratio and the number of polygons. As shown in Table 5, the results with a clustering distance of 0.02 demonstrate that, after center clustering, the bounding sphere reduction ratio increases dramatically due to the increase of the number of model polygons (66% for 3072 polygons, 84% for 5288 polygons, and 91% for 20 164 polygons).

The collision detection speed reduction ratio also dramatically increases as the original number of polygons increases (such as 62% for 3072 polygons, 81% for 5288 polygons, and 89% for 20 164 polygons). Fig. 20a compares the number of spheres without and with clustering, and Fig. 20b compares the collision detection time taken without and with clustering.

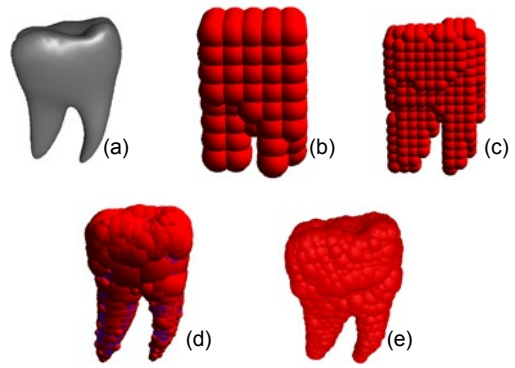

Fig. 17 Comparison with other methods for a tooth: (a) model; (b) octree level 3 (512 spheres); (c) octree level 4 (4096 spheres); (d) Hubbard (1047 spheres); (e) our method (1047 spheres)

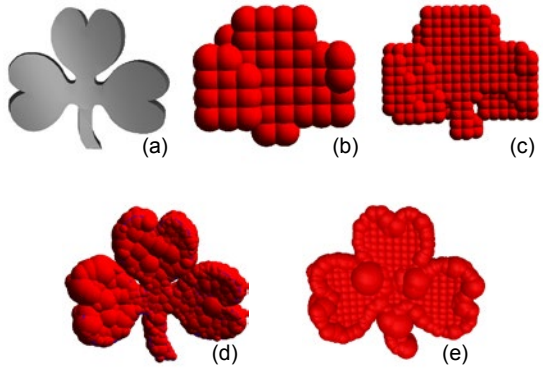

Fig. 18 Comparison with other methods for a clover: (a) model; (b) octree level 3 (512 spheres); (c) octree level 4 (4096 spheres); (d) Hubbard (858 spheres); (e) our method (858 spheres)

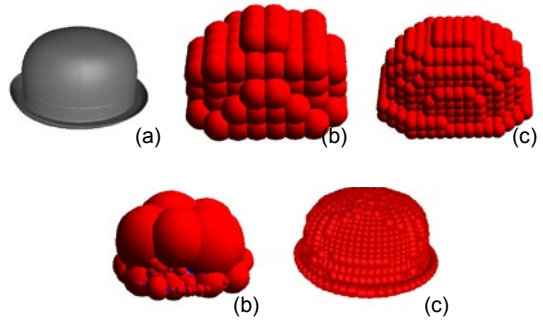

Fig. 19 Comparison with other methods for a hat: (a) model; (b) octree level 3 (512 spheres); (c) octree level 4 (4096 spheres); (d) Hubbard (1790 spheres); (e) our method (1790 spheres)

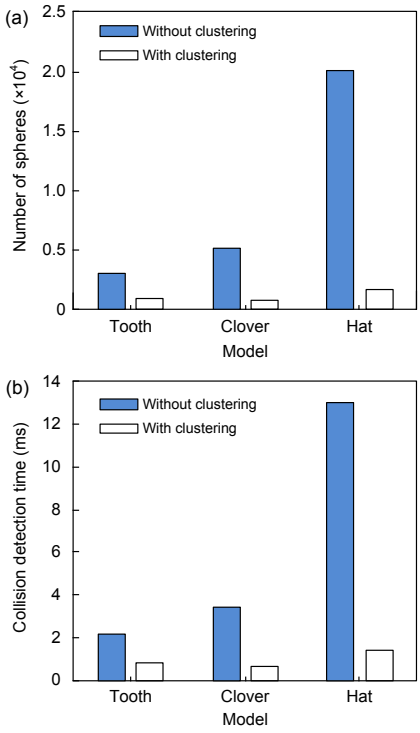

Fig. 20 Performance comparison: (a) number of spheres without or with clustering (clustering distance=0.02); (b) collision detection time without or with clustering (clustering distance=0.02)

6 Discussion and conclusions

Collision detection is a fundamental issue in many fields of computer science, especially in computer graphics and haptic rendering. A fast and precise algorithm for bounding sphere formation, which can generate object boundaries that can fill up 3D models as tightly as possible, has been proposed to accommodate stable haptic collision detection. In addition, our method allows easy control of contact levels of details (CLOD). A number of experiments demonstrate that CLOD can be done through single-link clustering with variable distances (e.g., 0–0.08). The selection of the clustering distance depends on the update rate and the complexity of the base geometry. Through this CLOD with distance-based clustering, our research holds its originality.

Our algorithm reduced the average radius of spheres to enable a more precise detection of the object by implementing the centroid of a triangle, instead of its circumcenter for calculating the center

Table 5 Performance comparison of the bounding spheres reduction ratio and collision detection speed reduction ratio without and with clustering*

Model	Number of polygons	Number of spheres with clustering	Bounding spheres reduction ratio	Total detection time without clustering (ms)	Total detection time with clustering (ms)	Collision detection time reduction ratio
Tooth	3072	1047	66%	2.155	0.810	62%
Clover	5288	858	84%	3.442	0.661	81%
Hat	20 164	1790	91%	13.026	1.419	89%

* Clustering distance=0.02

of a sphere. The accuracy and speed of collision detection were examined through numerous experiments, and the results demonstrated that the algorithm is fast indeed and precise enough to be applied to haptic simulations.

This work may contribute to basic studies in the field of haptic simulations. The advantages of our method based on clustered bounding spheres can be summarized as follows: (1) fast enough for haptic rendering, using spherical structure for bounding volume; (2) precise enough to the voxel level; (3) variable clustering distances for controllable CLOD; (4) less overhead for maintaining bounding sphere hierarchy; (5) smaller radii of bounding spheres using centroids of triangular polygons.

In future work, we will focus on improving our algorithm, applying our method to deformable objects effectively, and optimizing the usage of GPGPU in our system.

Acknowledgements

The authors would like to express their sincere thanks to Prof. Ji Bum KIM and Mr. Doug Jae YOO for numerous enlightening discussions and helpful comments.

References

Aragón, A.M., Molinari, J.F., 2014. A hierarchical detection framework for computational contact mechanics. *Comput. Meth. Appl. Mech. Eng.*, **268**:574-588. https://doi.org/10.1016/j.cma.2013.10.001

Barbič, J., James, D.L., 2008. Six-DoF haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Trans. Hapt.*, **1**(1):39-52. https://doi.org/10.1109/TOH.2008.1

Beckmann, N., Kriegel, H.P., Schneider, R., *et al*., 1990. The R*-tree: an efficient and robust access method for points and rectangles. Proc. ACM SIGMOD Int. Conf. on Management of Data, p.322-331. https://doi.org/10.1145/93597.98741

Bentley, J.L., 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM*, **18**(9):509-517. https://doi.org/10.1145/361002.361007

Blum, H.A., 1967. A transformation for extracting new descriptors of shape. *In*: Wathen-Dunn, W. (Ed.), Models for the Perception of Speech and Visual Form. MIT Press, Cambridge, p.362-380.

Bradshaw, G., 2003. Sphere-tree Construction Toolkit. http://isg.cs.tcd.ie/spheretree

Bradshaw, G., O'Sullivan, C., 2004. Adaptive medial axis approximation for sphere-tree construction. *ACM Trans. Graph.*, **23**(1):1-26. https://doi.org/10.1145/966131.966132

Bridson, R., Fedkiw, R., Anderson, J., 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.*, **21**(3):594-603. https://doi.org/10.1145/566570.566623

Colgate, J.E., Brown, J.M., 1994. Factors affecting the Z-width of haptic display. Proc. IEEE Int. Conf. on Robotics and Automation, p.3205-3210. https://doi.org/10.1109/robot.1994.351077

Dey, T.K., Zhao, W.L., 2004. Approximating the medial axis from the Voronoi diagram with a convergence guarantee. *Algorithmica*, **38**(1):179-200. https://doi.org/10.1007/s00453-003-1049-y

Garcia-Alonso, A., Serrano, N., Flaquer, J., 1994. Solving the collision detection problem. *IEEE Comput. Graph. Appl.*, **14**(3):36-43. https://doi.org/10.1109/38.279041

Goldsmith, J., Salmon, J., 1987. Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl.*, **7**(5):14-20. https://doi.org/10.1109/MCG.1987.276983

Gottschalk, S., Lin, M.C., Manocha, D., 1996. OBB-tree: a hierarchical structure for rapid interference detection. Proc. 23rd Annual Conf. on Computer Graphics and Interactive Techniques, p.171-180. https://doi.org/10.1145/237170.237244

Gregory, A., Lin, M.C., Gottschalk, S., *et al*., 2005. A framework for fast and accurate collision detection for haptic interaction. Proc. ACM SIGGRAPH 2005 Courses, Article 34. https://doi.org/10.1145/1198555.1198604

Hubbard, P.M., 1995. Collision detection for interactive graphics applications. *IEEE Trans. Visual. Comput. Graph.*, **1**(3):218-230. https://doi.org/10.1109/2945.466717

Hubbard, P.M., 1996. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph.*, **15**(3):179-210. https://doi.org/10.1145/231731.231732

James, D.L., Pai, D.K., 2004. BD-tree: output-sensitive collision detection for reduced deformable model. *ACM Trans. Graph.*, **23**(3):393-398. https://doi.org/10.1145/1015706.1015735

Klosowski, J.T., Held, M., Mitchell, J.S.B., *et al*., 1998. Efficient collision detection using bounding volume hierarchies of *k*-DOPs. *IEEE Trans. Visual. Comput. Graph.*, **4**(1):21-36. https://doi.org/10.1109/2945.675649

Larsson, T., Akenine-Möller, T., 2001. Collision detection for continuously deforming bodies. Proc. Eurographics, p.325-333. https://doi.org/10.2312/egs.20011005

McNeely, W.A., Puterbaugh, K.D., Troy, J.J., 1999. Six degree-of-freedom haptic rendering using voxel sampling. Proc. 26th Annual Conf. on Computer Graphics and Interactive Techniques, p.401-408. https://doi.org/10.1145/311535.311600

Moore, M., Wilhelms, J., 1988. Collision detection and response for computer animation. Proc. 15th Annual Conf. on Computer Graphics and Interactive Techniques, p.289-298. https://doi.org/10.1145/378456.378528

Otaduy, M.A., Lin, M.C., 2003. CLODs: dual hierarchies for multiresolution collision detection. Proc. Eurographics/ ACM SIGGRAPH Symp. on Geometry Processing, p.94-101. https://doi.org/10.2312/SGP/SGP03/094-101

Quinlan, S., 1994. Efficient distance computation between non-convex objects. Proc. IEEE Int. Conf. on Robotics and Automation, p.3324-3329. https://doi.org/10.1109/ROBOT.1994.351059

Roussopoulos, N., Leifker, D., 1985. Direct spatial search on pictorial databases using packed R-trees. Proc. ACM SIGMOD Int. Conf. on Management of Data, p.17-31. https://doi.org/10.1145/318898.318900

Ruspini, D.C., Kolarov, K., Khatib, O., 1997. The haptic display of complex graphical environments. Proc. 24th Annual Conf. on Computer Graphics and Interactive Techniques, p.345-352. https://doi.org/10.1145/258734.258878

Salisbury, L., Conti, F., Barbagli, F., 2004. Haptic rendering: introductory concepts. *Comput. Graph. Appl.*, **24**(2): 24-32. https://doi.org/10.1109/MCG.2004.1274058

Samet, H., 1984. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, **16**(2):187-260. https://doi.org/10.1145/356924.356930

Sibson, R., 1973. SLINK: an optimally efficient algorithm for the single-link cluster method. *Comput. J.*, **16**(1):30-34. https://doi.org/10.1093/comjnl/16.1.30

Teschner, M., Kimmerle, S., Heidelberger, B., *et al*., 2005. Collision detection for deformable objects. *Comput. Graph. For.*, **24**(1):61-81. https://doi.org/10.1111/j.1467-8659.2005.00829.x

Thibalt, W.C., Naylor, B.F., 1987. Set operations on polyhedra using binary space partitioning trees. Proc. 14th Annual Conf. on Computer Graphics and Interactive Techniques, p.153-162. https://doi.org/10.1145/37401.37421

van den Bergen, G., 1997. Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools*, **2**(4):1-13. https://doi.org/10.1080/10867651.1997.10487480

Vlasov, R., Friese, K.I., Wolter, F.E., 2013. Haptic rendering of volume data with collision detection guarantee using path finding. *LNCS*, **7848**:212-231. https://doi.org/10.1007/978-3-642-38803-3_12

Weller, R., 2013. New Geometric Data Structures for Collision Detection and Haptics. Springer International Publishing, Switzerland. https://doi.org/10.1007/978-3-319-01020-5

Weller, R., Zachmann, G., 2009. A unified approach for physically-based simulations and haptic rendering. Proc. ACM SIGGRAPH Symp. on Video Games, p.151-159. https://doi.org/10.1145/1581073.1581097

Zeng, D., Zheng, D.Y., 2012. Three-dimensional deformation in curl vector field. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **13**(8):565-572. https://doi.org/10.1631/jzus.C1200004