Haptic Collision Detection on Highly Complex Medical Data Structures

Niels Pirotte[1], Casper Vranken[1], Wout Swinkels[1], Luc Claesen[1], Yi Sun[2], and Constantinus Politis[2]

[1]Faculty of Engineering Technology, Hasselt University - UHasselt, Diepenbeek, Belgium
[2]OMFS IMPATH Research Group, Department of Imaging and Pathology, Faculty of Medicine, KU Leuven, and Oral and Maxillofacial Surgery, University Hospitals Leuven, Leuven, Belgium

***Abstract*—Nowadays, the planning procedure for orthognathic surgery consists of a manual workflow which relies on cost and time consuming tasks. The burden that this procedure has on the surgeon and the medical staff can be reduced by substituting the current procedure with a digital workflow. In the novel workflow the surgeon uses a haptic feedback device to mimic the haptic information perceived from the manual procedure. However, highly complex 3D medical scan models of the upper and lower jaw are needed to reproduce a realistic feeling. These complex models stress the need for an efficient collision detection algorithm to obtain the necessary update rate of at least 1 kHz for haptic feedback devices. In this paper the potential of the Inner Sphere Tree (IST) data structure is analyzed for application in the orthognathic surgery digital planning workflow. An open-source C++ program is developed on the CHAI3D platform for the implementation and evaluation of the IST. For the evaluation, the detection speed, but also the accuracy of the collision detection, in terms of the error introduced by the proximity of the minimum distance between bounding volume hierarchies (BVHs), are taken into consideration. Various tree traversal algorithms, distance and backtracking, are implemented and evaluated. Finally, a multi-point tree traversal algorithm is developed to find multiple contact-points between two ISTs. Due to the added optimizations and by using these tree traversal algorithms, the required update speed is reached.**

***Index Terms*—Orthognathic surgery, Digital workflow, Collision detection, Haptic rates, Inner sphere trees**

I. Introduction

Orthognathic surgery is the medical field specialized in the repositioning of the mandible (i.e. lower jaw) and/or the maxilla (i.e. upper jaw). Nowadays, the planning to perform such surgical corrections consists of a manual workflow. This procedure starts with a wax-bite taken from the patients teeth of which two plaster casts are derived. The surgeon uses the plaster casts to find the right occlusion between the teeth based on an optimal fit. Once the right occlusion is found the plaster casts are sticked together and a wafer is produced in a dental lab. This wafer contains the dental imprint of the desired occlusion and is used during the operation as depicted in Fig. 1.

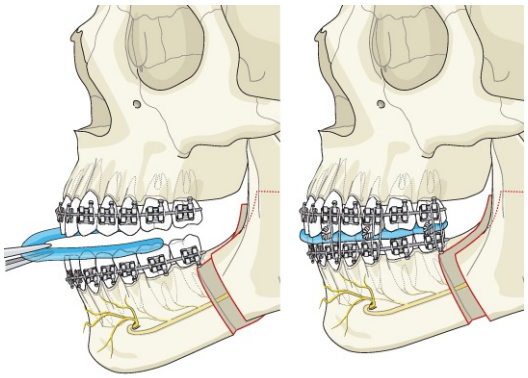

Fig. 1. Mandible repositioning [1]

The manual procedure is costly and time consuming. A digital workflow where 3D models represent the upper and lower jaw in combination with haptic feedback devices could provide a solution. However, these haptic feedback devices are constrained to an update rate of at least 1 kHz to mimic a realistic feedback. Due to this constraint there is a need for time-efficient and accurate data structures. In this case a bounding volume hierarchy (BVH) can be used as data structure. A BVH is a tree structure where every node is made of a simple geometric object (e.g. a cube, a box, a sphere, ...) to approximate the 3D model. There are several methods for the construction of BVHs. The most used are the top-down and the bottom-up methods. The top-down method starts with one simple geometric object that covers the entire model, which is called the root node. Then the model is divided into subregions. Each subregion is again represented as one simple geometric object which covers the part of the 3D model in that region. The deeper the tree is traversed the better the object is approximated by the simplified objects, where the leafs are the closest representation of the actual model. The bottom-up method starts from the leaf nodes and combines them to create the internal nodes. This process is recursively repeated until the root node is reached. The advantage of the tree structure is its spatial coherence, which allows fast exclusion of certain object regions for collision detection. In case there is an intersection between the root nodes of both models, the algorithm will search for an intersection in the next level of the tree. These nodes are already a better

approximation of the object. When there is no collision the algorithm stops traversing the tree. On the other hand when there is a collision, the algorithm moves on to the next level. This procedure is repeated until the leaf nodes are reached. If the leaf nodes collide it is assumed that the two models are also colliding. Research towards different kind of geometric primitives has already been conducted and includes the implementation of Axis Aligned Bounding Boxes (AABB) [2], Oriented Bounding Boxes (OBB) [3], Discrete Oriented Polytopes (k-DOP) [4], Spheres [5], ... In this paper the use of spheres as geometric representation is examined. Sphere trees can be constructed according to various principles. Hubbard [5] builds tight fitting sphere trees for the object that needs to be approximated. To achieve a tight fit, the object's Voronoi diagram is derived and used as the medial-axis approximation. This medial-axis is then used to construct the leaf spheres. The bottom-up approach is applied to build the sphere tree. An adaptation on this approach is made by Bradshaw et al. [6]. The concept of implicit sphere trees is introduced by Ruffaldi et al. [7]. A sphere hierarchy is constructed from octree nodes during the traversal of the octree for collision detection. Finally, Weller et al. [7] propose the use of inner spheres to approximate the volume of a 3D model. The inner spheres are non-overlapping spheres and serve as the leafs of the sphere tree. The sphere tree is constructed by applying the top-down technique, starting from a root sphere enclosing all inner spheres. Next the leaf spheres are partitioned and the spheres covering these partitions form a next layer in the tree. This process is repeated until a desired depth is reached.

In this paper the Inner Sphere Tree (IST) algorithm is examined for collision detection between two different high-poly 3D medical scan models of the upper and the lower jaw. The principles behind the IST are explained in section II. Different tree traversal algorithms, including a novel multipoint tree traversal algorithm, are described in more detail in section III. Section IV covers the experimental results regarding the speed and the obtained accuracy for collision detection. Finally, section V concludes the paper and gives some suggestions for further improvements.

II. Inner Sphere Trees

A. Voxelizer

To create inner spheres a voxelizer is needed, which fills the model with voxels. A voxel is a finite interval in a n-dimensional space, in this case a 3D-space, that contains the position and the shortest distance to the surface of the model. The bounding box of a model is filled with potential voxels of which the final set of voxels, contained in the model, is derived. Next, a ray is casted out from the position of a potential voxel to the infinite x-direction, however this could be any direction. If the ray intersects the surface of the model an odd number of times, the voxel is contained within the model. Otherwise it is located outside the model and the voxel will be discarded. After repeating this procedure for every potential voxel, only the voxels positioned inside the model will remain. To check for collisions between the ray and the model, the Möller-Trumbore intersection algorithm is used as described in [8]. This algorithm checks for collisions between a ray and a triangle. To check for intersections with the model, every triangle of the model needs to be verified. However, optimizations, to speed up this process, are made by implementing a tree structure. A binary tree of axis-aligned bounding boxes is constructed where every leaf contains exactly one triangle of the model. If the ray intersects the root bounding box, it is checked if the ray collides with the two children of this bounding box. This process repeats itself until a leaf is reached. When it intersects the leaf, the Möller-Trumbore intersection algorithm is applied. Finally the voxelizer uses the distance map generation algorithm as described in [9]. This algorithm calculates a largest distance to a set of triangles (i.e. the model) for a set of points (i.e. the voxels) using a BVH. In other words a distance mapping is created between the model and the voxels.

B. Inner Spheres

The inner spheres, which become the leafs of the sphere tree, are constructed from the voxels obtained after the voxelization step. The voxel that has the largest distance towards the surface of the model is used to create the first inner sphere with a radius equal to this distance and a center equal to the voxel position. This means that the created sphere touches the surface of the model but does not penetrate it. Next, all other voxels are iterated. If one of the other voxels is contained within the created sphere, it will be discarded. On the other hand if it is not contained within the sphere but the distance to the sphere is shorter than the distance to the surface of the model, the largest distance of the voxel is replaced with the distance to the sphere surface. Otherwise, the voxel remains unchanged. This step is repeated for every voxel. An example of the creation of the first 2 inner spheres is depicted in Fig. 2.

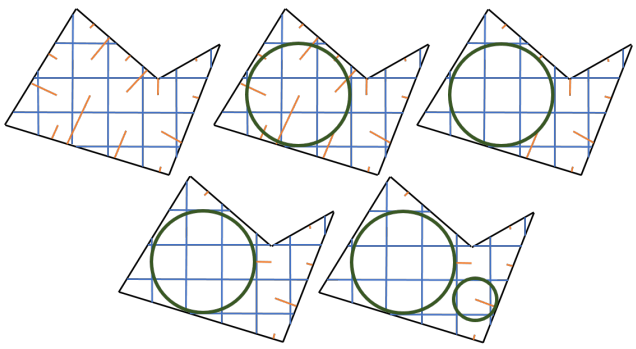

Fig. 2. The creation of the first 2 inner spheres. The orange line is the shortest distance to the surface or to a new sphere.

The distance between the voxels influences directly the number of inner spheres that will be created. If the voxels are closer together, more spheres are created. If, however, the

voxels are separated rather far apart from each other, less spheres are created. The number of spheres influences also the accuracy of the model approximation. The more spheres there are used to represent the model, the better the accuracy. In Fig. 3 a 2D-representation of the Stanford bunny with a completed sphere-packing is displayed.

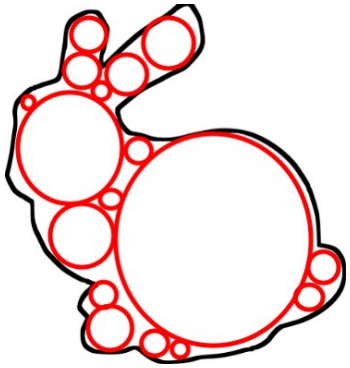

Fig. 3. The inner circles of the Stanford bunny model. This is a 2D-equivalent of a 3D sphere packing.

C. Batch Neural Gas algorithm

After the creation of the tight sphere packing, the IST is created recursively. The algorithm performing the IST construction process greatly influences the efficiency and thus the speed of the tree traversal. In [10] the BNG clustering algorithm is proposed to create the tree structure. This BNG algorithm is used for clustering data-points with similar characteristics and is applied in AI applications. In this case the spatial characteristics of the object are taken into consideration to provide efficient partitioning of the leaf spheres. First, all leaf spheres are enclosed by a root sphere. Repeatedly the leaf spheres are partitioned by the BNG algorithm, creating a number of disjoint groups. After each partitioning step the grouped leaf spheres are enclosed creating a number of new spheres representing nodes on a lower level in the tree. The process stops when the desired tree depth is reached.

The BNG algorithm provides a fast and robust clustering method independent of initialization and with guaranteed convergence by iteratively minimizing a cost function. The algorithm provides n new prototypes w_i for $i \in \{1..n\}$, in our case midpoints of new nodes. During each iteration step the prototypes converge further by assigning weights to each prototype and leaf sphere pair. For each midpoint these weights represent how close the midpoint is to each prototype. Next the prototypes are adjusted considering the weights, the positions and the volume of the leaf spheres as proposed in [10]. The maximum iteration number and the minimum movement of all prototypes are implemented as stop criteria. These values have a great influence on the final tree structure and thus the collision detection speed.

The adaptations during each iteration step are:

$$k_{ij} := |\{w_i : d(x_j; w_k) < d(x_j; w_i)\}| \in \{1..n\} \quad (1)$$

$$w_i := \frac{\sum_{j=1}^{m} h_\lambda(k_{ij}) x_j V(x_j)}{\sum_{j=1}^{m} h_\lambda(k_{ij}) V(x_j)} \quad (2)$$

The new prototypes are calculated based on a monotonous decreasing Gaussian shaped curve h_λ. This curve ensures convergence and decreases during each iteration step. A thorough mathematical foundation of the algorithm is given in [11].

Fig. 4 shows the grouping in a sphere packing applied on a mandible (i.e. lower jaw) model. Next each partition is enclosed by a sphere which hereby creates four new nodes in the tree structure. Finally, the BNG algorithm is applied recursively to each partition in order to create more tree levels.

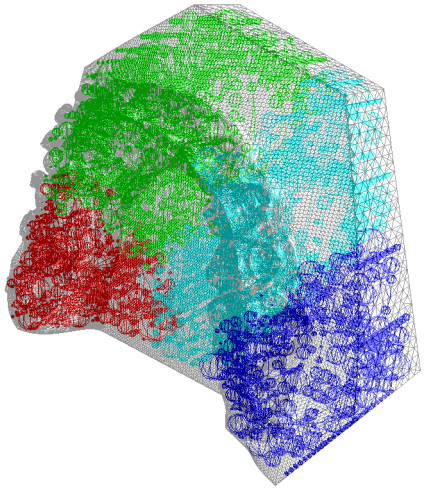

Fig. 4. Four partitions in a mandible sphere packing created by the BNG algorithm to obtain four new nodes in the tree structure.

III. Tree Traversal

To traverse the IST, several algorithms are implemented and optimized to speed up the collision detection. The first optimization is the broad-phase collision detection which checks if the root nodes are intersecting before starting to traverse the ISTs of both jaw models. Another optimization is that the models are only verified for collision if one of the objects has moved further than a predefined distance. When a collision check has been performed but the objects did not collide, an approximated distance between the objects is calculated. This approximation is always smaller than the actual minimal distance between the two objects. A collision check is only performed if the models have moved a certain distance, relative towards each other, which is greater than this approximated distance.

A. Distance traversal

The first tree traversal algorithm that will be discussed is the distance traversal algorithm. This algorithm performs a

query returning a minimum distance between two ISTs and–if one exists–a contact-point. This traversal algorithm starts at the root sphere and descends the tree as long as the distance computation between two spheres indicates that there is an intersection. When the level of the leaf spheres is reached and both spheres are still intersecting then the collision position is set. When there is no longer a collision, while descending the tree, the tree traversal will stop and the minimum distance calculated between two spheres so far is returned.

B. Distance backtracking traversal

The second algorithm is a backtracking traversal. The first collision is found according to the principles of the distance traversal algorithm. When the first collision is known and the traversal starts again, due to changing positions of the 3D models, the tree traversing starts at the parent sphere of the previous collision. Once the child nodes of the parent are visited and there is no collision, the algorithm starts searching for a collision one level higher. Now the children of this parent sphere are examined to find the collision. When traversing the tree starting from this sphere the subtree of the previous parent is ignored to avoid checking this branch multiple times. In the worst case this algorithm needs to track back to the root and traverse the tree again. However, the assumption is made that this is a very rare condition because of the small positional changes of the 3D models during each iteration. Algorithm 1 and 2 show the corresponding pseudocode.

Algorithm 1: checkDistanceB(A,B,minDist)

```
Input  : A,B = spheres in the inner sphere tree
Output: Boolean indicating if there is a collision or not
In/Out : minDist = overall minimum distance seen so far

if A or B == nullptr then
    // root sphere
    checkDistance(rootA, rootB, minDist)

if !checkDistanceR(parentA, parentB, minDist) then
    return checkDistanceB(parentA, parentB, minDist)

return true
```

C. Multipoint traversal

The final algorithm is the multipoint traversal, which is based on the backtracking algorithm. This algorithm tries to find a number of collisions in contrast to the previous algorithms, which only search for one collision. The first time the traversal starts at the root node. If there is a collision found then the parent of this sphere located at a certain split depth is excluded for further examination. The split depth is thus an indicator of how close different colliding points can lie together. The next search starts again from the root node and excludes the branch of the parent spheres on the split depth from the previous colliding points. The algorithm stops traversing the tree when there are no more collisions or when a specified number of collisions are found.

Algorithm 2: checkDistanceR(A,B,minDist)

```
Input  : A,B = spheres in the inner sphere tree
Output: Boolean indicating if there is a collision or not
In/Out : minDist = overall minimum distance seen so far

if distance(A,B) >minDist then
    return false
else
    minDist = distance(A, B)

if A and B == leaf and distance(A,B) == 0 then
    pos = position of the collision
    stop iteration
    return true

if distance(A,B) >0 then
    // When no leaf but minDist is adjusted
    return false

forall the children a[i] of A do
    if node a[i] already visited then
        Skip this node
    forall the children b[j] of B do
        if node b[j] already visited then
            Skip this node
        checkDistanceR(a[i], b[j], minDist)
        if stop iteration then
            return true

return false
```

The next time the tree is traversed, the previous collisions are checked first. The colliding points that are no longer valid are now the starting points for the backtracking. The parent spheres at the split depth of the other points, that are still valid, are set. These branches are no longer examined. If a new collision is found then the parent at the split depth of this sphere is also set. In case a sphere that is the starting point for backtracking has now the same parent at the split depth because of the new colliding point, then the parent of this split depth parent is taken as the starting point.

In case the split depth is not chosen properly it can prevent the algorithm to find collisions.

IV. Results

For the calculations and the algorithm implementations a Windows 10 laptop with an Intel i5 2.6GHz dual core CPU and 4GB of RAM is used. Collisions are tested between two highly-complex jaw models, one with 119k vertices and the other with 106k vertices.

To test the accuracy and speed, an identical hand recorded motion is used for the evaluation of all detection algorithms. The predefined distance optimization, discussed in section

III, is removed and no backtracking is implemented. This is because data regarding the inner sphere tree structure is needed, and not about the optimization algorithms.

A. Sphere packing

The sphere packing algorithm provides a tight packing for the test models. Three different accuracies are generated by the program. Two of those three accuracies are also tested with multiple depth levels in the tree. The tight packing is illustrated in Fig. 5 for the Stanford dragon model. In this picture it is clear that the big spheres in the body of the dragon are surrounded by smaller spheres to fill up the tiny spaces. This is as expected.

Fig. 5. The sphere packing algorithm applied to the Stanford dragon.

B. Detection speed

The speed depends on the tree depth, the number of leaf nodes within the tree and the settings of the BNG algorithm. Fig. 6 shows the collision detection speed with respect to an increasing number of voxels and thus an increasing number of leaf spheres. If the quality of the voxelization rises, the speed decreases. Fig. 7 and 8 depict the effect of an increasing tree depth on the average detection speed. The speed rises if there are more levels in the tree until a certain point, then the speed starts to decrease again. Results also show that slightly different settings of the BNG algorithm, such as stop criteria or prototype start values, can result in a higher or lower update rate.

C. Accuracy

The accuracy depends only on the amount of leaf spheres in the IST which on its turn depends on the quality of the voxelization. As depicted in Fig. 9, an increase in the voxelization quality results in a better accuracy. The results are expressed as a relative error percentage. A slow but very accurate PQP library has been used to calculate the exact distances between the models. The relative error percentage is calculated by dividing the absolute error by the distance the models are removed from each other.

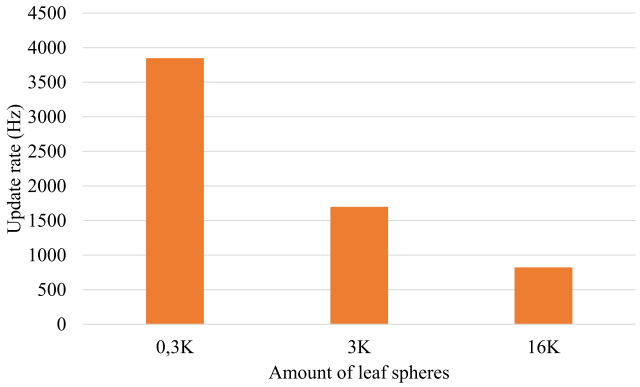

Fig. 6. The average collision detection speed for multiple accuracies.

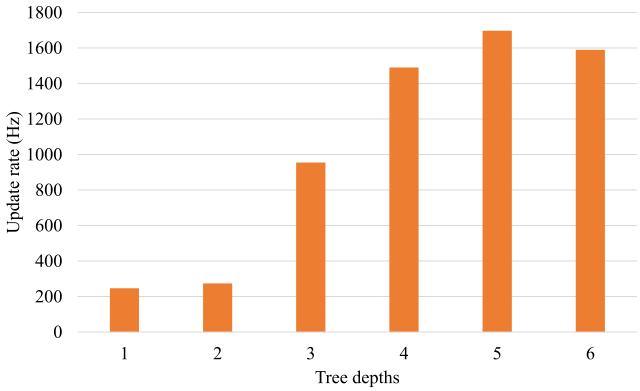

Fig. 7. The average collision detection speed for multiple depth levels of a tree with 3K leaf spheres.

D. Backtracking

In certain cases, the backtracking algorithm can result in far higher detection speeds compared to the distance algorithm. When the backtracking algorithm searches its first contact point or if there is no contact point, it is equally fast to the distance algorithm. However, if the two jaw models are already colliding, the backtracking algorithm can speed up the detection 50 times compared to the distance detection speed. It is presumed that small changes happen in the movement of the jaw models, which is true in normal cases.

E. Multipoint

The multipoint algorithm can find multiple contact points by using the backtracking algorithm multiple times within one collision check. When there is no contact between the models, it is as fast as the distance algorithm. For the multipoint algorithm a predefined number of possible collisions is set. Then, the algorithm tries to find this number of collisions. If there are more contact points than this number, it can increase the detection speed up to 50 times. But when there is only one contact point, this algorithm slows down significantly. This is because the first point will be found quickly using the backtracking algorithm, but for the next point the algorithm

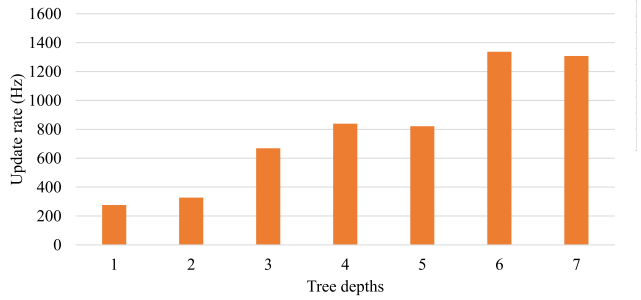

Fig. 8. The average collision detection speed for multiple depth levels of a tree with 16K leaf spheres.

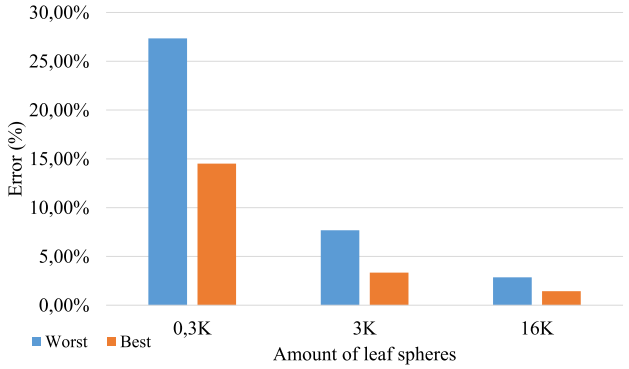

Fig. 9. The error on the collision detection algorithms decreases when more leaf nodes are added to the IST.

will have to traverse the entire tree because it will not be able to find a collision point.

V. Conclusion

An open-source C++ program is created on the CHAI3D platform [12] to build the IST of 3D models. The packing of inner spheres is performed properly and verified by the Stanford dragon model and two jaw models which resemble the same result as described in [10]. The implemented BNG algorithm cannot be compared to the one used in [10]. Although it has the same basic functionality, the settings cannot be optimized to construct the perfect tree. The depths of the tree are crucial with respect to the update rate. The speed reaches a maximum at a certain depth, therefor further expanding the tree is non-beneficial. The accuracy depends on the amount of leaf spheres in the tree. This is also directly related to the quality of the voxelization. Thus the higher the quality of the voxelization, the higher the collision detection accuracy. However, the voxelization quality has also an effect on the speed of the collision detection. A trade-off between the speed and accuracy is necessary and should carefully be made for every model.

After implementing tree traversal optimizations haptic update rates are reached. The first optimization is the backtracking algorithm, which improves the detection speed when the two models are colliding. If the two models are not colliding, the algorithm calculates a certain distance that the models are allowed to move relative towards each other before it is necessary to check their relative distance again. This is the second optimization. Combining these optimizations, results in haptic update rates for every scenario. In the future, parallelization could be added to the program. This will result in higher speeds and could provide haptic rates without any other optimization.

Future research includes the implementation of a collision response algorithm to model the appropriate forces once a collision is detected. A second aspect that needs further investigation is that currently it is still possible that some small areas inside the model are not covered. This is due to the fact that non-overlapping spheres cannot cover the complete volume of an arbitrary object because of their geometry.

The open-source code can be found at the following link: https://github.com/caspervranken/BachelorThesis_Pirotte_Vranken.

Acknowledgment

The research in this paper was sponsored in part by the Belgian FWO (Flemish Research Council) and the Chinese MOST (Ministry of Science and Technology) bilateral cooperation project number G.0524.13.

References

[1] A. S. Reference, "Positioning of the tooth bearing segment," 2017, [Online; accessed July 6, 2017].

[2] H. Samet, *An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 51–68.

[3] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 171–180.

[4] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, Jan-Mar 1998.

[5] P. M. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Trans. Graph.*, vol. 15, no. 3, pp. 179–210, Jul. 1996.

[6] G. Bradshaw and C. O'Sullivan, "Adaptive medial-axis approximation for sphere-tree construction," *ACM Trans. Graph.*, vol. 23, no. 1, pp. 1–26, Jan. 2004.

[7] E. Ruffaldi, D. Morris, F. Barbagli, K. Salisbury, and M. Bergamasco, "Voxel-based haptic rendering using implicit sphere trees," in *2008*.

[8] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *J. Graph. Tools*, vol. 2, no. 1, pp. 21–28, Oct. 1997.

[9] D. Morris, "Algorithms and data structures for haptic rendering: Curve constraints, distance maps, and data logging." Stanford University, June 2006.

[10] R. Weller and G. Zachmann, "Inner sphere trees for proximity and penetration queries," in *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[11] M. Cottrell, B. Hammer, A. Hasenfu, and T. Villmann, "Batch and median neural gas," *Neural Networks*, vol. 19, no. 6, pp. 762 – 771, 2006.

[12] F. Conti, F. Barbagli, R. Balaniuk, M. Halg, C. Lu, D. Morris, L. Sentis, J. Warren, O. Khatib, and K. Salisbury, "The chai libraries," in *Proceedings of Eurohaptics 2003*, Dublin, Ireland, 2003, pp. 496–500.