Collision detection based on octree for virtual surgery system

Yanrong Hu, Zhiyuan Yan\textsuperscript{2*}, Zhenlun Yin\textsuperscript{3}, Zhijiang Du\textsuperscript{4}

\textsuperscript{1}State Key Laboratory of Robotics and System, Harbin Institute of Technology, Harbin, China
\textsuperscript{2}State Key Laboratory of Robotics and System, Harbin Institute of Technology, Harbin, China
\textsuperscript{3}School of Naval Architecture and Ocean Engineering, Harbin Institute of Technology, Harbin, China
\textsuperscript{4}State Key Laboratory of Robotics and System, Harbin Institute of Technology, Harbin, China

*Corresponding author e-mail: zhiyuanyan@163.com

Abstract. In the virtual surgery system, the organ is always non-convex and deforms over time which makes the collision detection problem more difficult to handle. In this case, we put forward an innovative collision detection algorithm based on octree. This algorithm can perfectly apply to deal with non-convex organs and organ deformation scenario. In addition, it can achieve high run-time efficiency.

1. Introduction

With the rapid development of robotics and surgical technology, robot-assisted minimally invasive surgery has become a significant trend since the 1990s\textsuperscript{[1]}. The technology includes a small camera and some custom-made surgical instruments. The surgeon performs surgical operations by inserting the camera and instruments into patient body without making large openings. Consequently, for patients, minimally invasive operations are less pain and basically no scars, shorter recovery times and less discomfort during rehabilitation over traditional open surgery. Although minimally invasive surgery possesses these advantages, its clinical application still faces tremendous problems including appropriate training methods. Virtual surgical training system is such a platform that can provide training for doctors. The higher immersion of the surgical training system, the better the training effect.

A more immersive virtual surgery training system not only need to interact in real-time with the virtual organs through a force-feedback device but also need to perform a real-time visualization of the deformations\textsuperscript{[2]}. Whereas, an efficient collision detection is the key to realize the higher immersion of the training system, since it will make a preparation for subsequent calculation of organ deformations and response force. And the rest of this paper focuses on the this specific problem. Because of its wide applications, collision detection between objects has been studied for years in various fields such as robotics, game, simulation etc. The solutions vary based on the geometric representation of the colliding objects and the type of query the algorithm could support\textsuperscript{[2]}. In the background of virtual surgery training system, we mainly focus on the methods that could detect collision between polygonal models. Most of the algorithms in collision detection between polygonal objects are suitable for convex polyhedra\textsuperscript{[3, 4]}. These algorithms determine the closest features of a pair of polyhedra based on specific data structures. They complexity of time is low and can achieve high run-time efficiency. However, they are not applicable in the case of surgery simulator, since organs are...
Generally non-convex. Among the collision detection techniques that are applicable to non-convex polygonal objects, almost all of them rely on a pre-computed hierarchy of bounding volumes. Solutions range from axis-aligned box trees, oriented bounding box trees, sphere trees to more specific data structures\textsuperscript{5, 6, 7}. Nevertheless, all these pre-computing algorithms do not seem adequate, since the organs deform over time and this computation has to be redone at each time step. Therefore, when organ deformation occurs, these algorithms may considerably slow down.

From the above analysis, an efficient collision detection algorithm applied to virtual surgery training system not only could be applicable in the case of non-convex organs but also should be suitable for organ deformation scenario. In this paper, we propose an octree-based collision detection algorithm. This algorithm could be applicable to more general polygonal models and also could deal with the organ deformation. Compared to other pre-computing algorithms, it could achieve high run-time efficiency.

The remainder of this paper develops as follows: Section 2 explains the methodology of this algorithm and analyzes the performance of this algorithm. Section 3 presents our experiments and results. In the end, we make a summary in Section 4.

2. Methodology

In the virtual surgery training system, organ usually has a complex shape but represented using a collection of simpler geometric primitives such as vertices, edges, and triangles. Whereas surgical tool has a very simple shape typically represented by thin and long cylinder. When collision between organ and surgical tool occurs, it means that the distance between the tip of the surgical tool and one of the vertices of the organ is approximately equal to zero, i.e.

\[
|T - X_i| < \xi
\]  

(1)

\( T \) is the position of the surgical tool’s tip, \( X_i \) is the \( i \) th vertex’s position of the organ, \( \xi \) is a very small positive number and is approximately equal to zero.

Therefore, in this case, collision detection problem is converted to calculating the distance between the tip and these vertices of organ. To optimize the calculation, we adopt a novel approach instead of brute-force way calculating each distance between tip and vertices. The algorithm develops as follows:

1. Set up an octree in the vertices’s space and initialize the octree according to the vertices’ position.
2. Tell whether the tip is in the octree’s space or not, if yes, search the leaf node of the octree, otherwise, return to step (2).
3. Calculate each distance between the tip and the leaf node’s vertices, if satisfy the formula (1), then collision occurs, otherwise, return to step (2).
4. Find the leaf nodes in the deformable space according to the deformable region.
5. Update the octree in the deformable region and return to step (2).

In this algorithm, step (1) is a pre-computing step, whereas step (2) (3) (4) (5) have to compute in every render frame to guarantee the real-time performance. Considering the surgical tool’s velocity is very slow, this algorithm would always be valid and it is be verify in the following paper. Next, we explain the octree initialization, collision detection, octree updating process in detail.

2.1. Initializing the octree

As figure 1 shows, an octree is an axis-aligned hierarchical data structure that is generated by recursively subdividing the axis-aligned bounding box into eight equally-sized cells as necessary. The aim we use octree is to find the tip nearby vertices as quick as possible. Thus, we only need to calculate very few distances to optimize the collision detection process. Before we initialize the octree, we have to define octree data structure and determine the subdividing rule.
2.1.1. Octree data structure. Generally speaking, the octree data structure must contain the cube boundary, the points in this cube, the points’ amount and eight children node. Whereas, in our application scenario, the points’ position are stored in the organ 3D model. Therefore, we just need the points’ index instead of the points’ position to save more memory. To simplify the octree data structure, we define a cube data structure to represent the octree boundary. The octree data structure and the cube data structure show in figure 2 and figure 3 respectively.

```
public class Octree
{
    public Cube boundary;
    public int count;
    public List<int> pointIndexes;
    public Octree northEastBack;
    public Octree northWestBack;
    public Octree southEastBack;
    public Octree southWestBack;
    public Octree northEastFront;
    public Octree northWestFront;
    public Octree southEastFront;
    public Octree southWestFront;
}
```

```
public class Cube
{
    public float centerX;
    public float centerY;
    public float centerZ;
    public float width;
    public float height;
    public float depth;
}
```

Figure 2. Octree data structure written in C#.
Figure 3. Cube data structure written in C#.

2.1.2. Subdividing process. To optimize the collision detection computation, we need to subdivide the octree. First of all, we set up an octree root node and determine its cube boundary according to all of the vertices’ position. Afterwards, the choice of whether to subdivide an octree node or not depends on the the desity of the vertices. If the vertices amount is greater than leaf node maximum capacity, then we subdivide the octree.

2.2. Collision detection

Collision detection is to determine whether the tip of surgical tool is in contact with one of the organ vertices or not. After the octree being initialized, we adopt the greedy search strategy to find the leaf node of the octree according to the tip’s position. If we find the leaf node, it means that the tip is likely to contact with one of the leaf node vertices. Afterwards, we calculate the distance between the tip and all of the leaf node vertices, if satisfy the formula (1), then collision occurs.

To guarantee the collision detection correctness, $\xi$ in the formula (1) has to greater than 3D modeling precision. As figure 4 shows, two red rectangles represent two octree leaf nodes and the points labelled with number are vertices in these two leaf nodes. When the tip lies in the red point, the octree leaf node that algorithm finds is the left one. Obviously, the tip is approaching the point 4 but
point 4 belongs to the right leaf node. Thus, $\xi$ in the formula (1) has to greater than $p$ which is 3D modeling precision. But if we want a high position accuracy, the $\xi$ best value is $p$.

The above analysis is static, but actually, the collision detection process is dynamic because the tip is always moving during simulation. As figure 5 shows, the tip is moving from $A$ to $A'$ during a time interval. Thus, in this case, we have to ensure $\xi$ is greater than $\eta$. According to the geometric relationship,

$$\eta = (p^2 + 0.25\Delta x^2)^{1/2}$$  \hspace{1cm} (2)

$\Delta x$ is the distance that the tip moves during a time interval and it can be calculated by the following formula (3).

$$\Delta x = v\Delta t$$  \hspace{1cm} (3)

$v$ is the velocity of the tip and the value is normally 5~10mm/s, $\Delta t$ is the time interval between two simulation steps and its value is normally less than 1/30 s. Whereas, the value of $p$ is roughly equal to 1~2 mm. Therefore, in the formula (2), compared to the value of $p$, the value of $\Delta x$ could be neglected. From above analysis, when $\xi$ equals to $p$ could guarantee the validity of this algorithm.

**Figure 4.** Static collision detection analysis

**Figure 5.** Dynamic collision detection analysis

### 2.3. Updating the octree

As the collision between surgical tool and organ occurs, the organ generally would deform and the deformation field dies off rapidly with increase in distance from the surgical tool tip\(^8\). Therefore, we need a local neighborhood search algorithm to confirm the region influenced by the surgical tool tip. Because it helps update the octree and significantly improves the run-time efficiency.

The deformation field is determined by the characteristic of the organ which is not the research topic of this paper. For updating the octree, we need to ascertain which leaf nodes are influenced by deformation. Thus, we take points by uniformly-spaced in three dimensional direction in this deformation area as equation (4) displays.

$$(x + kl_x, y + kl_y, z + kl_z) \quad k = \pm 0, 1, 2 \cdots$$  \hspace{1cm} (4)

$x, y, z$ represents the contact point position, $l_x, l_y, l_z$ represent the octree leaf node width, height and depth respectively.

Afterwards, we adopt the greedy search strategy to find the leaf nodes of the octree according to these points. If the vertices in this leaf nodes change, we replace these vertices by the new one. At this point, we finish the octree updating work.

### 2.4. The time complexity of this algorithm

In the virtual surgery training system, we have to make sure it can run in real time. Therefore, the time complexity of this collision detection algorithm is the factor that we have to take into account. Step (1) is a pre-computing step and thus it does not affect the real-time performance of this algorithm. From above analysis, step(2) and (3) belong to the collision detection process and step (4) and (5)
belong to the octree updating process. In the collision detection process, we need to find the octree leaf node where the tip locates and calculate a few distance between the tip and the vertices in this leaf node. Therefore, the time complexity in this process can be denoted as:

\[ T_1 = \log_8 N + n_1 M \]  

And in the octree updating process, we need to find a few octree leaf nodes and replace a few vertices. Therefore, the time complexity in this process can be denoted as:

\[ T_2 = n_2 \log_8 N + n_2 K \]  

The whole time complexity of this algorithm can be calculated by:

\[ T = T_1 + T_2 = (n_2 + 1) \log_8 N + n_1 M + n_2 K \]  

N represents the whole vertices amount in the organ. M represents the time used to calculate the distance between two point and it is almost constant. K represents the time used to replace one vertex and it is almost constant. \( n_1 \) represents the vertices amount in the leaf node. \( n_2 \) represents the vertices amount influenced by deformation area. In practice, N is always a big number whereas \( n_1 \) and \( n_2 \) is very small number. Therefore, the equation (7) can be simplified as:

\[ T = O(\log_8 N) \]  

3. Experiments and results

We test our algorithm in Unity 3D. The organ are obtained by marching cubes algorithm dealing with the patient CT images and it has 3996 vertices. The surgical tool is modelled as a thin and long cylinder. Figure 6 and figure 7 show the organ model and the surgical tool model respectively. Afterwards, We test the performance of our algorithm from four aspects: octree initialization, collision detection, octree updating and the performance of this algorithm.

![Figure 6. Organ model](image1)

![Figure 7. Surgical tool model](image2)

3.1. Octree initialization

We let the octree leaf node maximum capacity equal to 10. We draw the octree leaf nodes’ boundary in the program and the octree subdividing result is shown in figure 8. From this figure, octree leaf nodes are small and concentrated in the region where vertices are dense. On the contrary, octree leaf nodes are large and scarce in the region where vertices are sparse. The result totally meets our expectation because we subdivide the octree according to the amount of the vertices.

![Figure 8. Octree subdividing result](image3)
3.2. Collision detection

As the surgical tool tip approach the organ, the algorithm would find the octree leaf node where the tip locates and display the leaf node vertices in white color just as figure 9 shows. When distance between the tip and one of these vertices satisfies equation (1), the program would draw a red ray from the vertex just as figure 10 shows. These phenomena are in line with our expectations.

3.3. Octree updating

For the ease of research work, we use a red cube to represent the deformation area just as figure 11 shows. According to the approach mentioned in chapter 2.3, we search all the octree leaf nodes around the deformation area and display all the vertices in these nodes in white color. After the organ deformation, from the figure 12, we could observe that some vertices displaying in white color have changed their position and this phenomenon suggests that we update the octree successfully.

3.4. The performance of this algorithm

To demonstrate that our algorithm could significantly improve the run-time efficiency, we conduct collision detection experiments based on octree and brute-force way respectively. A brute-force way means testing all the possible pairs of vertices whose time complexity is $O(N^2)$. From figure 13 and figure 14, we could obviously observe that octree-based way 263.8 FPS (i.e., frames per second) is much larger than brute-force way 0.9 FPS and it shows that our algorithm runs roughly 300 times faster than the brute-force way in virtual surgery system.
4. Conclusion

Aiming at the virtual surgery training system, our collision detection algorithm can apply to deal with the non-convex organs. In addition, it also can apply to the organ deformation scenario. Most importantly, its time complexity is $O(\log N)$ and can achieve high run-time efficiency. However, from chapter 2.2 analysis, the precision of this algorithm totally depends on 3D modeling precision. If we want to improve the precision of this algorithm, we have to improve the 3D modeling precision.

5. Acknowledgments

This research was supported by the project of Study on The Key Technology of Remote Bone Trauma Surgery Robot(Grant No. 2017YFB1304202).

References

[1] Basdogan, Cagatay, et al. "VR-based simulators for training in minimally invasive surgery." IEEE Computer Graphics and Applications, 2007, 27.2: 54-66.
[2] Lombardo, J-C., M-P. Cani, and Fabrice Neyret. "Real-time collision detection for virtual surgery." Proceedings Computer Animation. 1999. IEEE.
[3] Baraff, David. "Curved surfaces and coherence for non-penetrating rigid body simulation." ACM SIGGRAPH Computer Graphics, 1990, Vol. 24. No. 4. ACM.
[4] M. Lin and J. Canny. Efficient collision detection for animation. In Third Eurographics Worship on Animation and Simulation, 1992, Cambridge, England.
[5] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. Computer Graphics, Proceedings of SIGGRAPH'96, 1996, pages 171-180.
[6] P. Hubbard. (1996) Approximating polyhedra with spheres for time-critical collision detection. ACM Transactions on Graphics, 1996, 15(3):179-210.
[7] S. Quinlan. Efficient distance computation between nonconvex objects. In International Conference of Robotics and Automation, 1994, pages 3324-3329.
[8] De, Suvranu, et al. "Physically realistic virtual surgery using the point-associated finite field (PAFF) approach." Presence: Teleoperators and Virtual Environments, 2006, 15.3: 294-308.