AUTOMATIC GENERATION ALGORITHM OF UNITY GAME OBJECT BOUNDING BOX BASED ON MORTON CODE

Huawei MEI[1], Chenyao FAN[2], Ronghua ZHANG[3]*

Aiming at the problem that bounding boxes need to be defined manually in Unity real-time interactive program, an automatic generation algorithm of hierarchical bounding boxes is proposed. Firstly, the advantages and feasibility of hierarchical bounding box automatic generation algorithm in static model in interactive application emphasizing real-time are analyzed theoretically; Then, sphere bounding box and AABB bounding box are selected, and the hierarchical bounding box tree is constructed in parallel through GPU through the top-down construction strategy and the maximum difference bit division method based on Morton code; Finally, the leaf node is determined and the AABB bounding box is filled into the hierarchical bounding box tree through post order traversal. This method is implemented as a Unity plug-in. The experimental results show that compared with the collider manually defined by Unity, this method reduces the tedious work of manually defining the bounding box during modeling and realizes the real-time automatic generation of the bounding box; Compared with the existing automatic bounding box generation algorithm, this method can more accurately describe the shape features of 3D model and has higher dynamic performance.

Keywords: hierarchical bounding box; Morton code; GPU parallel processing; maximum difference bit division; automatic generation algorithm

1. Introduction

As a key step in the preprocessing stage of the collision process, the automatic bounding box generation algorithm is widely used in 3D games, cloth simulation, virtual fitting, and other scenes to quickly build the collider and enhance the fidelity and immersive experience of the scene. Hierarchical bounding box technology can improve the robustness of the collision effect from the perspective of strategy and algorithm. With the development of virtual reality technology, higher requirements are put forward for the real-time and fidelity of collision effect. In Unity, to achieve fast and effective collision, first manually define a set of sphere or capsule colliders around a complex static model. A

[1] Prof., School of Control and Computer Engineering, North China Electric Power University, China, e-mail: fn123@ncepu.edu.cn
[2] M.S., School of Control and Computer Engineering, North China Electric Power University, China, e-mail: fancy_chenyao@163.com
[3] Eng., School of Control and Computer Engineering, North China Electric Power University, China, e-mail: zronghua88@aliyun.com

collider cannot accurately describe the state of static model collision. For finer processing, multiple colliders can be manually placed on a 3D model. However, placing multiple colliders on a complex 3D model is a typically time-consuming and cumbersome task. The definition of colliders by manual has some problems, such as low efficiency and being prone to errors.

To solve the above problems, this paper designs an automatic generation algorithm of hierarchical bounding box based on Morton code. This method uses the top-down method based on Morton code to construct the BVH tree and then processes each non-leaf node in parallel to further improve the performance of the algorithm. It can generate a set of hierarchical bounding boxes in real-time, and can realistically describe the shape of the model.

At present, the recognized bounding box generation algorithms use hierarchical bounding box technology [1] to encapsulate and store object information. Hierarchical bounding box technology (BVH) is used to combine bounding boxes into a tree structure, and the time complexity is reduced from $O(n^2)$ to $O(\log n)$ [2]. Hierarchical bounding box technology is convenient to improve the efficiency of collision elimination and improve the performance of the algorithm. Commonly used bounding boxes include sphere, AABB (axis-aligned bounding box), OBB (oriented bounding box), and k-DOPs (discrete orientation polymers) [3]. There are three construction strategies of hierarchical bounding box [4]: top-down construction strategy, bottom-up construction strategy, and incremental construction strategy. Compared with the other two construction strategies, the top-down method adopted in this paper has higher execution efficiency. The top-down method is the earliest construction method used. Lauterbach et al. [5] proposed a hierarchical bounding box construction method (LBVH) based on Morton code, which changed the hierarchical bounding box construction problem into a sorting problem and realized high-quality bounding boxes at a very low cost. Pantaleoni et al. [6] extended this algorithm and improved it by using the spatial correlation of hierarchical grid decomposition (HLBVH), reducing the computational overhead and memory usage. These algorithms improve the construction and traversal efficiency of BVH to a certain extent. This paper draws lessons from the core idea of the Morton code construction method. The bottom-up method mainly highlights the global optimal segmentation plane. This method starts from the leaf node and combines the elements in pairs according to different combination methods to form the internal nodes of the hierarchical bounding box until the final root node is completed. Hu et al. [7] proposed a method using local density clustering, which constructs a hierarchical bounding box on the GPU by analyzing the distribution of entity density. Many scholars have studied the generation algorithm of a single bounding box [8-11], but in a complex environment, the model shapes are different, and the single bounding box algorithm cannot meet the requirements. Therefore, many scholars have studied the hybrid bounding box [12-14] algorithm. Liu Xiaoping et al. [15] proposed a sphere OBB hybrid hierarchical bounding box structure. This method can shorten the update time of the bounding box and improve the detection efficiency, but it is not suitable for complex structure models. Sun et al. [16] proposed a sphere-k-dops hybrid hierarchical bounding box structure, which has strong real-time performance. Guo et al. [17] proposed a sphere-k-dops hybrid hierarchical bounding box structure based on it, but these two methods affect the automatic generation efficiency of hybrid hierarchical bounding boxes due to the slow update of k-DOPs. The bottom-up method can often construct high-quality hierarchical bounding boxes, but it is not as fast as the top-down method. To obtain higher quality bounding box, an incremental method is proposed based on the first two. Its main research direction also focuses on the optimization strategy. Meister and Bitner [18] completed the incremental construction method of GPU on this basis and improved the construction speed. For dynamically changing scenes, the incremental method will be better, but the cost is lower execution speed, so it is difficult to achieve real-time. In 2019, Kim et al. [19] designed and implemented the automatic generation algorithm of the static model using Unity plug-in but did not realize GPU parallel processing in the automatic generation process, and the automatic generation efficiency of the collider is not ideal.

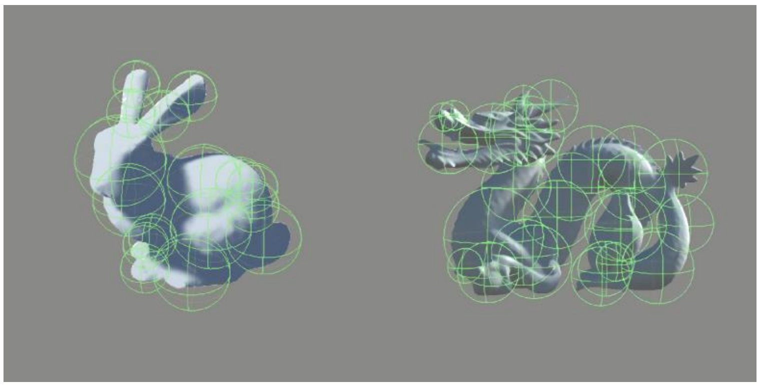


Fig. 1 The effect of a bounding box defined manually by Unity

This paper presents a fast and efficient algorithm for the automatic generation of hierarchical bounding boxes. A set of BVH structure colliders are automatically generated for three-dimensional objects through the algorithm, and the BVH tree is generated based on Morton Code. For non-leaf nodes, the AABB bounding box is used to quickly eliminate the area without collision and improve the elimination efficiency; The leaf node adopts a sphere bounding box. And develop it as a plug-in of Unity, which is widely used in real-time game development. Fig. 1 shows the model effect of manually defining 27 sphere bounding boxes for a simple static model in 4 hours. There is still a situation that the accuracy of the bounding box cannot accurately describe the shape of the model. This paper gives the performance improvement of this method by comparing it with the method of manually defining bounding boxes by Unity and literature [19].

2. Material and Methods

Compared with most methods of constructing bounding boxes for each triangular surface of the model, this paper proposes a method to simplify the model by constructing a group of BVH structural bounding boxes. Firstly, an AABB bounding box is defined for the whole model as the root node of the BVH tree, and then the whole is divided into multiple voxels according to the defined grid structure, and a sphere bounding box is defined for each voxel. The sphere bounding box is used as the leaf node of the BVH tree, and each non-leaf node is filled with an AABB bounding box through the upward index of the tree structure for collision elimination.

2.1 Selection of bounding box

Table 1 shows the functional comparison of common bounding boxes. The sphere bounding box has a simple structure, but its tightness is poor. It is suitable for scenes with a large amount of rotation. OBB is much stronger in compact than AABB and sphere, but complex construction methods will reduce the real-time performance. K-DOPs are very important for the selection of K values. It needs many tests to select the best K value, which is more troublesome. Although the tightness of AABB is not as good as OBB, the construction method is simple and can better meet the real-time requirements in interactive programs. Therefore, this paper chooses to use AABB and sphere bounding box to construct BVH. Build AABB bounding box for non-leaf nodes and sphere bounding box for leaf nodes, as shown in Fig. 2.

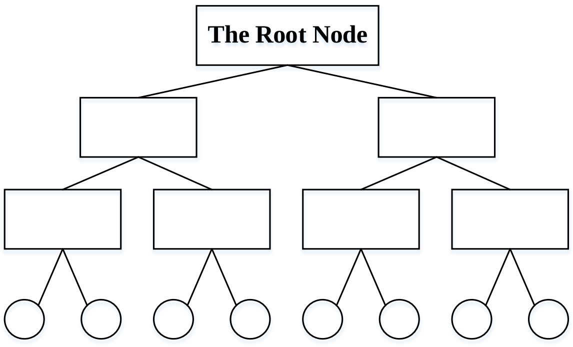


Fig. 2 AABB-Sphere Hierarchical bounding box

Table 1

Comparison of common bounding box performance

Type of bounding box	Difficulty of construction	Tightness	Efficiency of renewal
Sphere	easy	medium	no need to update

AABB	easy	medium	fast
OBB	complex	closely	slow
k-DOPs	medium	closely	slow

2.2 Constructing bounding box based on Morton code

a. Degree of tree

The common choices of hierarchical bounding box trees include binary tree, trigeminal tree, quadtree, and octree. With the increase of the degree of the tree, the depth of the tree becomes smaller, and the overall iteration times of constructing the hierarchical bounding box tree will also become smaller, which can reduce the construction time overhead of the hierarchical bounding box tree to a certain extent [20]. However, the degree of hierarchical bounding box tree becomes larger, which often leads to more intersections between bounding boxes. As shown in Fig. 3, A and B represent two AABB hierarchical bounding box trees of objects m and N respectively. Region D of M and region a of N intersect. The intersection test process of the binary tree only needs 9 intersection tests, but quadtree needs 17 intersection tests. It is not that the greater the degree of the tree, the better. The existing algorithms generally use binary tree structure and dynamic multi-tree structure. In this paper, the binary tree structure is adopted. The expression of the tree structure is simple, which is convenient for the intersection test in the process of collision. When the top-down construction method is adopted, only one segmentation surface is needed to realize the division of parent nodes.

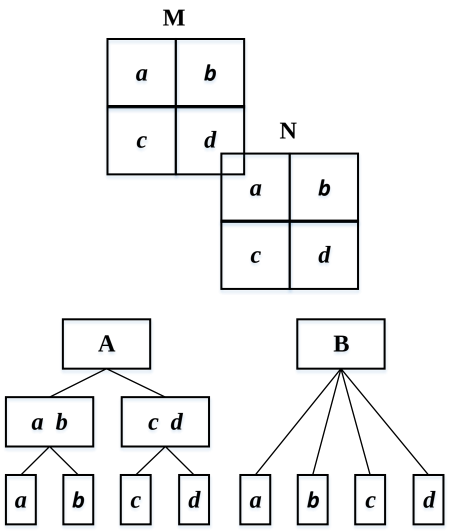


Fig. 3 The effect of degree of tree on efficiency

b. Top-down construction strategy

The top-down method is the most widely used hierarchical bounding box construction method. This method is simple and easy to implement. The steps are as follows: first, construct the bounding box for the whole static model as the root node, subdivide the static model into small geometry according to certain rules, and construct the bounding box for the geometry, which can also be subdivided into smaller geometry. The construction method of subdividing in this order is similar to the tree structure, and the leaf nodes constitute the smallest geometric element of the deformable body. The key to this method is how to correctly divide a given set into two subsets. In this paper, the maximum difference bit division method based on Morton code is used to construct a binary tree. The pseudo-code tree node of the top-down construction strategy is shown in Algorithm 1. The Morton codes are quickly sorted, and then the elements are placed in a spatially coherent manner. Then the BVH is constructed by recursively dividing the range of the current root node.

```
Algorithm 1: TreeNode
Input: Sphere[], AABB
Output: root
Begin
   Map all spherical coordinates to a cube of length 1 with the smallest point at the origin
   Converts spherical coordinates in space to Morton codes
   Quick sort of Morton code
   The position of each leaf node was determined according to the maximum difference bit
division of Morton code
   Get a hierarchical bounding box tree with only leaf nodes
   The AABB bounding box was filled by post-order traversal
   return root
End
```

c. Maximum difference bit division based on Morton code

For the Morton code at a given point, its position is defined as $X_0Y_0Z_0X_1Y_1\cdots$ in the unit cube. Where the X coordinate of the point is $0.X_0X_1X_2\ldots$, the Y coordinate is $0.Y_0Y_1Y_2\cdots$ and the Z coordinate is $0.Z_0Z_1Z_2\ldots$. The generation method of Morton code is shown in Fig. 4. According to the above method, first, calculate the Morton code according to the center point of each entity. The generation algorithm of Morton code is shown in Algorithm 2. Morton3D method first converts the x, y, and z components of the coordinates to be processed into integers, then expands the 10-bit integers to 30 bits by ExpandBits method, and finally performs an operation to obtain the required Morton code. Specifically, for the current root node of the coverage range [i, j], the maximum difference between Morton codes in the range [i, j] is found, which is recorded as γ.

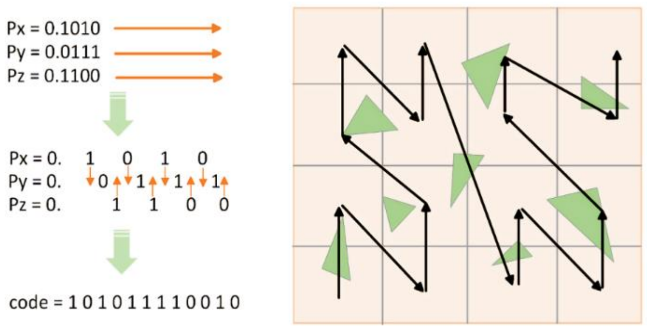


Fig. 4 Schematic diagram of Morton code generation method

Then the left child node of the current node is overwritten [i, γ], Right child node overlay [γ+1, j]. This step needs to be performed recursively until the current node contains only one element, and then the construction process of BVH is completed. The schematic diagram of this construction method is shown in Fig. 5.

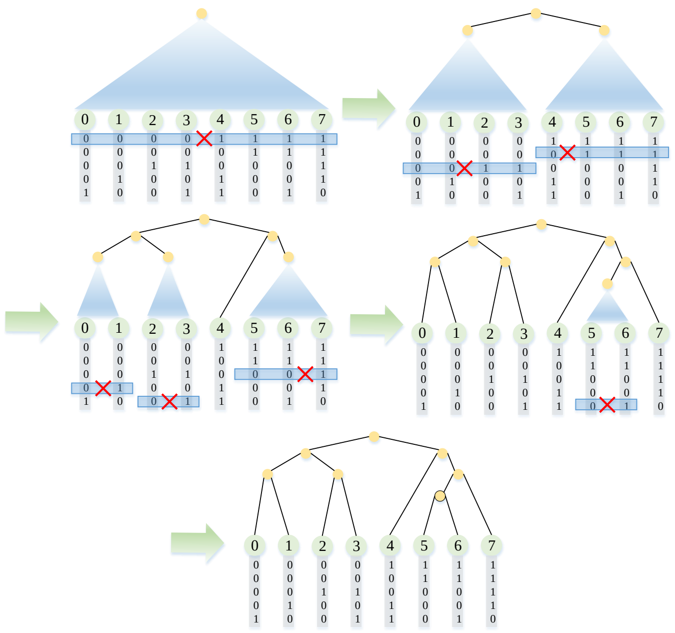


Fig. 5 The diagram is divided according to the maximum difference bits between Morton codes

```
Algorithm 2: Morton3D
Input: px, py, pz(A 10-bit unsigned integer)
Output: MortonCode(30 bit unsigned integer)
Begin
    x = Mathf.Min(Mathf.Max(x * 1024.0f, 0.0f), 1023.0f);
    y = Mathf.Min(Mathf.Max(y * 1024.0f, 0.0f), 1023.0f);
    z = Mathf.Min(Mathf.Max(z * 1024.0f, 0.0f), 1023.0f);
    uint xx = ExpandBits((uint)x);
    uint yy = ExpandBits((uint)y);
    uint zz = ExpandBits((uint)z);
    return xx * 4 + yy * 2 + zz;
End
```

The code for calculating the maximum difference is shown in Algorithm 3. First, take the minimum and the maximum number of Morton code groups to be segmented, perform XOR operation on them, and then find the longest common bit. Then conduct a binary search to find the element whose first bit is different from the previous element.

```
Algorithm 3: FindSplit
Input: sortedMortonCodes, first, last
Output: split(Point of division)
Begin
uint firstCode = sortedMortonCodes[first];
uint lastCode = sortedMortonCodes[last];
if (firstCode == lastCode)
    return (first + last) >> 1;
int commonPrefix = LeadingZeros(firstCode ^ lastCode);
int split = first;
int step = last - first;
do{
    step = (step + 1) >> 1; // exponential decrease
    int newSplit = split + step; // proposed new position
    if (newSplit < last) {
        uint splitCode = sortedMortonCodes[newSplit];
        int splitPrefix = LeadingZeros(firstCode ^ splitCode);
        if (splitPrefix > commonPrefix)
            split = newSplit; // accept proposal
    }}
while (step > 1);
return split;
End
```

d. GPU parallel processing

The basic idea of using GPU parallel processing [21] has been described in Fig. 4. The parallel construction of BVH includes four parts: assigning a Morton code to each entity; Sort Morton codes; Establishing a binary base tree and assigning an AABB bounding box to each non-leaf node. The BVH constructed in this paper is a full binary tree structure, that is, if there are n leaf nodes, there will be n-1 non-leaf nodes. For each non-leaf node, parallel processing is performed on the GPU. First, determine the child nodes of each non-leaf node. When dividing according to the maximum difference of Morton code, because the same ancestor has the same prefix, different ancestors have different prefixes. With this idea, the direction of leaf nodes can be determined quickly. After finding the direction of leaf nodes, the prefix property of descendant nodes can still be used to further determine the range of non-leaf nodes, and then leaf nodes can be determined. Then, according to the determined leaf node, find the prefix of the leaf node, and use the binary search to find the position where the last bit changes for the first time as the segmentation point. Finally, the whole parallel construction process is completed.

3. Results

3.1 Experimental environment

The automatic generation algorithm of the hierarchical bounding box proposed in this paper is implemented as a Unity plug-in, and the automatic generation test of the bounding box for various 3D objects is carried out. The experiment was carried out in the computing environment shown in Table 2. The method of literature [19] is implemented in the current experimental environment to ensure that the practice environment is consistent. The experimental model used in this paper is shown in
Table *3*.

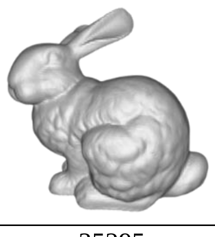
Table 2

Table of experimental environment parameters

Components	Instructions
OS	Windows 10 Family Chinese version
Unity	Unity 2019 2.3 f1
CPU	Intel Core i5 10400F
RAM	16GB
GPU	NVIDIA GeForce GTX1650
VRAM	8GB

Table 3

The experimental model

Model names	Stanford bunny model	Dragon model
Model figure		
The number of vertices	35295	104872
The number of faces	70580	209227

3.2 Effect demonstration

Fig. 6 shows the automatic generation effect of the bounding box of the experimental model with an accuracy of 1 * 1 * 1, 3 * 3 * 3, and 5 * 5 * 5. Fig. 7 shows the effect diagram of a group of bounding boxes automatically generated after customizing the voxel accuracy for other static models with different structures. The original static model is on the far left, the complete effect of the

AABB sphere bounding box is in the middle, and the sphere bounding box effect used to realize collision and describe the characteristics of the model is on the right.

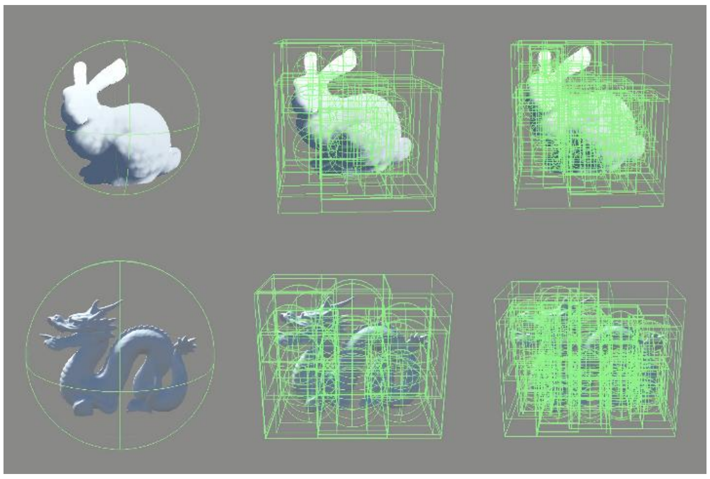


Fig. 6 The experimental model automatically generates the effect pictures of bounding boxes with different precision

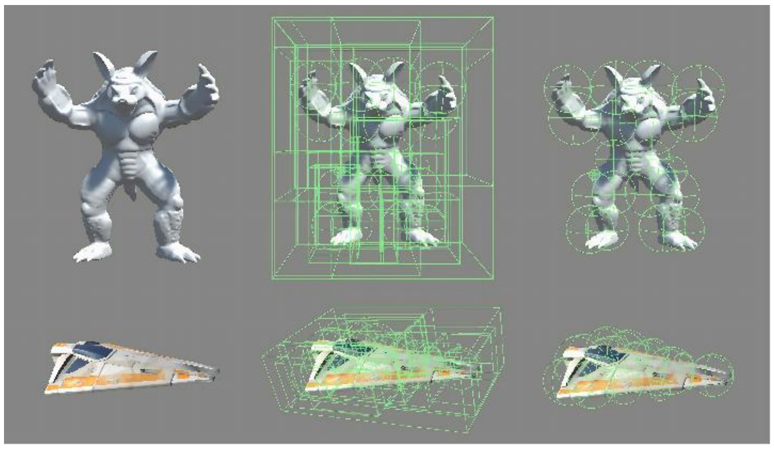


Fig. 7 Other models automatically generate bounding box renderings

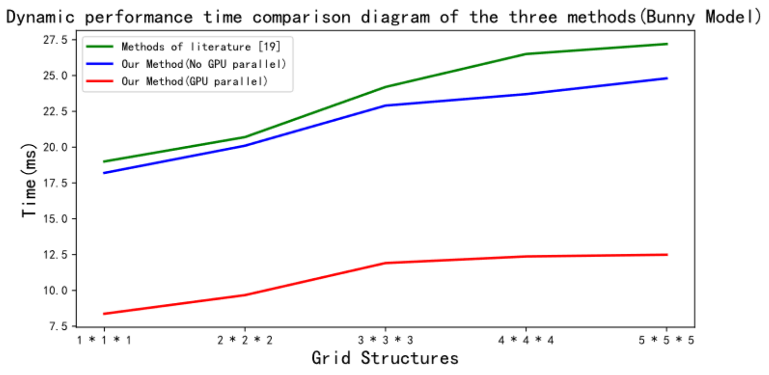
The experimental results show that the bounding box automatically generated by this algorithm is suitable for static models with different structures, and with the improvement of accuracy, it can describe the characteristics of the model more accurately. When there is only one bounding box, there is only one leaf node, so there is no AABB bounding box, only one bounding sphere; In the 3 * 3 * 3 grid, there are 27 leaf nodes and 26 non-leaf nodes, that is, there are 27 bounding spheres and 26 AABB bounding boxes. This paper compares the dynamic performance time of the manually added Collider in Unity, the method of reference [19], the automatic generation collider without GPU parallel processing, and after GPU parallel processing. Table 4 shows the comparison of dynamic performance time of the four methods under different accuracy.

Table 4

Comparison of test model performance of collider with different structure

Methods	Grid structure	Model name	
		Bunny model	Dragon model
Unity defines bounding boxes manually	3 * 3 * 3	0.211s	0.247s
Methods of literature [19]	1 * 1 * 1	0.019s	0.021s
	3 * 3 * 3	0.021s	0.022s
	5 * 5 * 5	0.027s	0.027s
Our methods (No GPU parallel processing)	1 * 1 * 1	0.018s	0.018s
	3 * 3 * 3	0.023s	0.025s
	5 * 5 * 5	0.025s	0.027s
Our methods (GPU parallel processing)	1 * 1 * 1	0.008s	0.013s
	3 * 3 * 3	0.012s	0.013s
	5 * 5 * 5	0.012s	0.015s

Fig. 8 shows the dynamic performance time comparison between the method in this paper and the method in reference [19] under the same network structure.



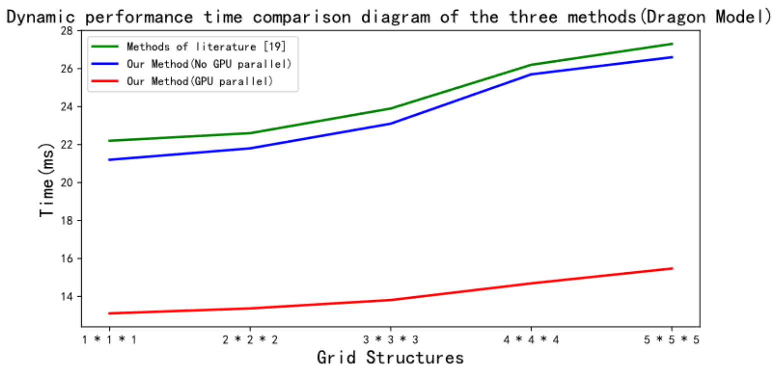
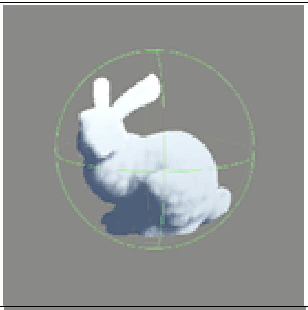
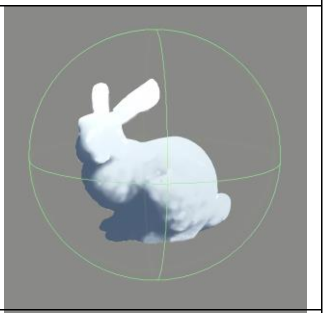
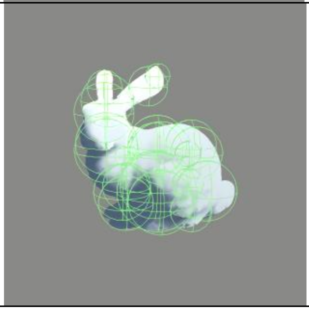
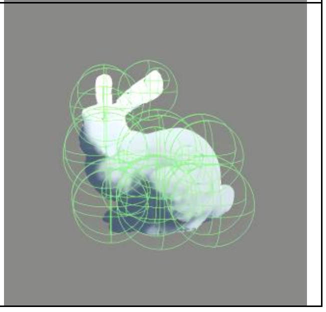


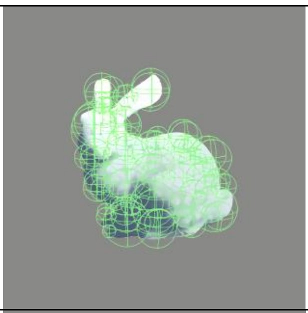
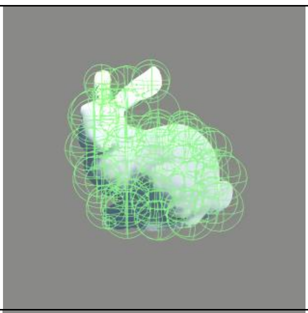
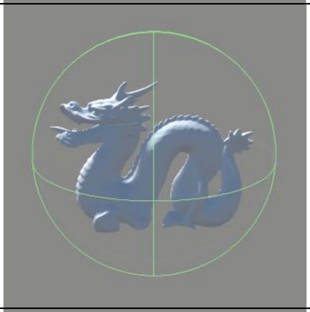
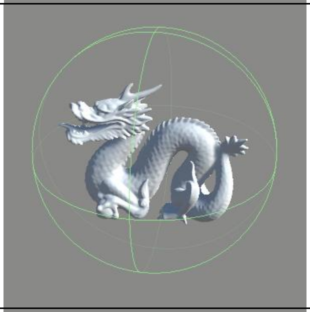
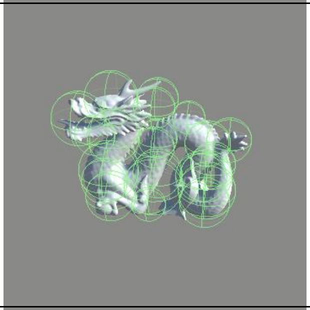
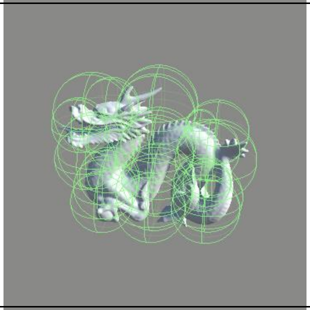
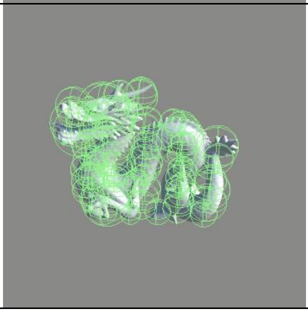
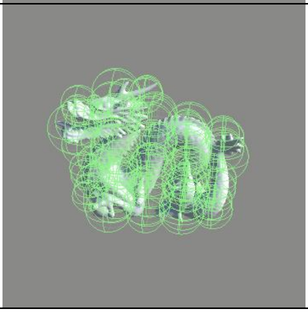
Fig. 8 Dynamic performance time comparison diagram of the three methods

The experimental results show that the method proposed in this paper can quickly realize the automatic generation of a variety of colliders, and with the improvement of accuracy, it can more accurately describe the three-dimensional features of the model. The efficiency of manually defining the bounding box is the lowest. Under the condition of GPU parallel processing, the efficiency of this method is significantly higher than that of the method in literature [19] and without GPU parallel processing. When the grid structure is more complex, the dynamic performance time of the method in this paper is only increased by 0.004s at most, which is still far less than that of Unity adding collider manually.
Manually defining the bounding box requires a lot of time to calculate the center coordinates and radius of the sphere bounding box, which is inefficient and error-prone. This method generates the bounding box results in real-time after setting the accuracy and ensures that the generated bounding box can accurately describe the model structure and avoid manual errors.
Table *5* shows the comparison between the method in this paper and the method in reference [19] in the effect of automatically generating bounding boxes under the same model and the same accuracy.

Table 5

Comparison between our method and the method in literature [19] on the generation effect of bounding box

Model names	Grid structure	Our methods	Methods of literature [19]
Stanford bunny model	1 * 1 * 1 grid-based		
	3 * 3 * 3 grid-based		

	5 * 5 * 5 grid-based		
Dragon model	1 * 1 * 1 grid-based		
	3 * 3 * 3 grid-based		
	5 * 5 * 5 grid-based		

The generation effect shown in this method hides the AABB bounding box used to eliminate collision, and only shows the sphere bounding box used to depict the shape of the model. Compared with the method in reference [19], the bounding box generation quality of this method is higher and can describe the model features more accurately. In the experiment, with the increase of voxel accuracy, the collection of colliders is closer, and the shape features of the model are described more specifically.

Table 6

Simulation results of the experimental model

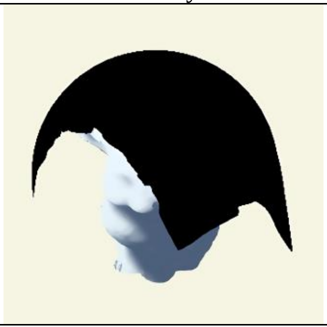
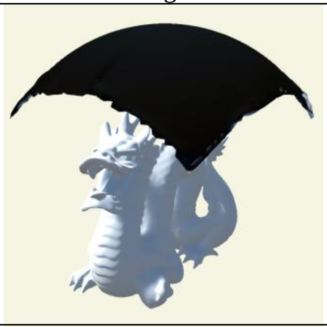
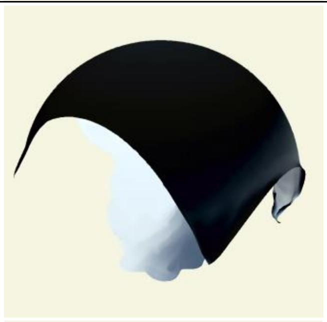
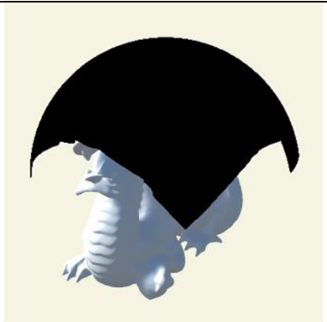
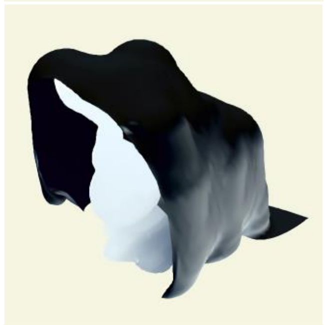
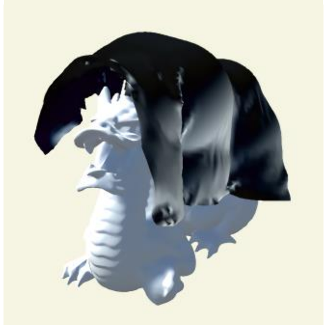
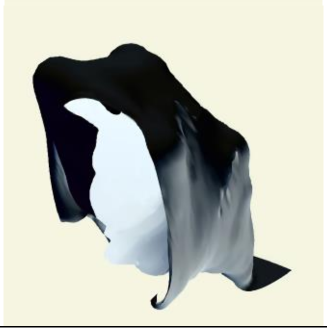
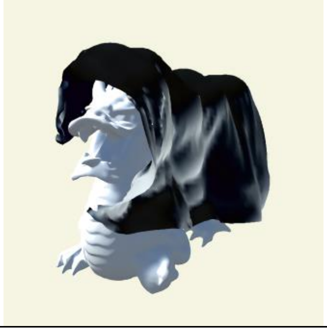
Method	Grid structure	64 * 64 cloth model	
		Bunny	Dragon
Unity cloth & one collider	1*1*1		
Our methods	1*1*1		
	3*3*3		
	5*5*5		

Table 6 intuitively shows the impact simulation effect of the static model of cloth component and the method in this paper with flexible cloth in the simulation scene with collision. This paper is implemented in the form of Unity plug-in. Our 5*5*5 precision method can more realistically display the simulation effect of objects in a shorter time than unity's collision processing based on a single Collider and cloth component.

4. Conclusions

Based on the Unity game engine, this paper designs and implements an automatic generation algorithm of hierarchical bounding box based on Morton code, which eliminates the redundant work of manually defining bounding box and improves operational efficiency. This paper compares the two cases of using GPU parallel processing and without GPU parallel processing. GPU parallel processing reduces the dynamic performance time by about 50% under the condition of ensuring the same effect of automatic generation of the bounding box, improves the performance of the algorithm, and makes the method in this paper more real-time. At the same time, this paper compares the effect and performance with the bounding box manually defined in Unity and the method in reference [19]. Experimental results show that this method reduces cumbersome human tasks, automatically generates higher quality hierarchical bounding boxes with shorter dynamic performance time under the condition of higher accuracy, and accurately describes the characteristics of the static model. At present, only the automatic generation algorithm of the static model is considered, and the automatic generation algorithm of the dynamic model is considered to be explored in the future. At the same time, different construction methods are used to evaluate the performance of the automatic generation algorithm, and the latest collision processing algorithm [22-24] is considered to improve the realistic effect. The algorithm is applied to cloth simulation and other scenes with high real-time requirements, so as to provide a more immersive simulation effect and more realistic visual feedback.

R E F E R E N C E S

[1]. *Meister D, Ogaki S, Benthin C, et al*. A Survey on Bounding Volume Hierarchies for Ray Tracing. Computer Graphics Forum, 2021, 40(2): 683-712.

[2]. *Wodniok D, Goesele M*. Recursive SAH-based Bounding Volume Hierarchy Construction//Graphics Interface. 2016: 101-107.

[3]. *Yang Fan*. AABB bounding box collision detection Algorithm based on B+ tree storage. Computer science, 48(6A): 331-333.

[4]. *Qu Huiyan*. Research on fast collision detection technology in complex virtual Environment. Jilin University, 2020.

[5]. *Lauterbach C, Garland M, Sengupta S, et al*. Fast BVH Construction on GPUs. Computer Graphics Forum, 2009, 28(2): 375-384.

[6]. *Pantaleoni J, Luebke D*. HLBVH: Hierarchical LBVH construction for real-time ray tracing of dynamic geometry// Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on High Performance Graphics 2010, Saarbrücken, Germany, June 25-27, 2010. ACM, 2010.

[7]. *Hu Y, Wang W, Li D, et al*. Parallel BVH Construction Using Locally Density Clustering. IEEE Access, 2019: 105827-105839.

[8]. *Xu-Sheng S, Li-Hong Q, Zuo-Wei Z*. Algorithm of collision detection based on improved oriented bounding box. J. Hunan Univ.(Natural Sci.), 2014, 41(5): 26-31.

[9]. *Cheng Shijun, Feng Yueping*. Fast collision detection algorithm based on bus bar of cylindrical bounding box. Journal of Jilin University (Natural Science), 2015, 53(02): 291-296. DOI:10.13413/j.cnki.jdxblxb.2015.02.26.

[10]. *Erxi Z, Min X, Yuanjun H*. A collision detection algorithm using delaunay triangulation. Journal of Graphics, 2015, 36(4): 516.

[11]. *Xing-Xing Z, Ming-Hong X, Ya-Yun Z*. Fast collision detection algorithm based on uniform spatial subdivision and linear programming. Comput. Eng. Appl., 2017, 53(23): 236-240.

[12]. *Wang Chao, Zhang Zhili, LONG Yong, Wang Shao-di*. Research on improved Hybrid Bounding Box Collision Detection Algorithm. Journal of System Simulation, 2018, 30(11): 4236-4243.DOI:10.16182/j.issn1004731x.joss.201811023.

[13]. *Jian L I, Ming-Yue W, Ru-Jing Y, et al*. Optimization of collision detection algorithm based on hybrid hierarchical bounding box under background of big data. J. Jilin Univ.(Sci. Ed.), 2017, 55(3): 673-678.

[14]. *Xie Weichao*. Research on collision detection algorithm of hybrid bounding box in complex environment. Jiangxi University of Science and Technology,2018.

[15]. *Liu Xiaoping, Zhang Yingkai, Xie Wenjun, et al*. Sphere-obb bounding box Fast collision detection algorithm for Character Animation. Journal of System Simulation, 2014, 26(7): 1535-1540.

[16]. *Sun Hanqin*. Research and implementation of key technology of Automobile virtual Assembly System. University of north, 2017.

[17]. *Guo X, Zhang Y, Liu R, et al*. Efficient collision detection with a deformable model of an abdominal aorta//2016 IEEE International Conference on Information and Automation (ICIA). IEEE, 2016: 927-932.

[18]. *Meister D, Bittner J*. Parallel Reinsertion for Bounding Volume Hierarchy Optimization. Computer Graphics Forum, 2018, 37(2): 463-473.

[19]. *Minsang Kim, Nak-Jun Sung, Sang-Joon Kim, Yoo-Joo Choi, Min Hong*. Parallel cloth simulation with effective collision detection for interactive AR application. Springer US, 2019, 78(4).

[20]. *Chitalu F M, Dubach C, Komura T*. Binary Ostensibly-Implicit Trees for Fast Collision Detection//Computer Graphics Forum. 2020, 39(2): 509-521.

[21]. *Karras T*. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees//Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics. 2012: 33-37.

[22]. *Min Tang, Tongtong Wang, Zhongyuan Liu, Ruofeng Tong, and Dinesh Manocha*. I-Cloth: Incremental Collision Handling for GPU-Based Interactive Cloth Simulation, ACM Transactions on Graphics, 37(6), Article 204 (November 2018), 10 pages (Proc. of ACM SIGGRAPH Asia), 2018.

[23]. *Tang M, Liu Z, Tong R, et al*. PSCC: Parallel self-collision culling with spatial hashing on GPUs. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2018, 1(1): 1-18.

[24]. *Huamin Wang*. 2021. GPU-Based Simulation of Cloth Wrinkles at Submillimeter Levels. ACM Transactions on Graphics (SIGGRAPH), vol. 40, no. 4, pp. 169: 1-16.