

A Collision Detection and Cloth Simulation Method Based on Parallel BVH

Huawei Mei¹, Chenyao Fan¹, Ronghua Zhang^{1*}, Yonggang Liu²

¹The college of control and Computer Engineering, North China Electric Power University, Baoding, Hebei, China

²Shijiazhuang water supply Co., Ltd, Shijiazhuang, Hebei, China

*Corresponding Author.

Abstract:

To solve the challenges such as low simulation performance, poor simulation effect, and manual definition of a collider in the real-time collision in cloth simulation, a collision detection method based on parallel hierarchical bounding volume hierarchies (BVH) is proposed. Firstly, the advantages and feasibility of using hierarchical bounding box for collision detection in cloth simulation are analyzed theoretically; Then AABB bounding box and the sphere bounding box is selected, and a group of BVH structure bounding boxes is constructed in parallel by using the hierarchical bounding box generation method based on Morton code; Finally, the computational shader is used to execute in parallel so that multiple particles update their positions at the same time and traverse the BVH structure of the model. The experimental results shown in Unity show that this method reduces the tedious work of manually defining the bounding box, speeds up the collision detection rate, accurately depicts the state when the cloth collides with the object and is suitable for real-time interactive applications.

Keywords: Cloth Simulation, Collision Detection, Mass-Spring Model, Hierarchical Bounding Box Technology, GPU Parallel Processing.

I. INTRODUCTION

Collision detection technology is now widely used in 3D games, cloth simulation, virtual fitting, and other scenes to enhance the fidelity and immersive experience of the scene. Collision detection is an essential process in cloth simulation. Cloth is different from a rigid body, with unique material and irregular shape. An efficient collision detection algorithm is needed to improve the real-time and fidelity of cloth simulation. Many interactive applications define the collider manually to achieve the collision effect. However, in complex cases, its operation efficiency is low, and it is not easy to ensure the system's robustness. In cloth simulation, seeking the balance between simulation effect and real-time has become one of the main problems to be solved in the application of cloth simulation and collision detection in interactive programs. Therefore, it is vital for applying cloth simulation in an interactive system to correctly consider the balance between simulation effect and simulation efficiency and improve the real-time performance of the simulation.

At present, the recognized collision detection algorithms use hierarchical bounding box technology [1] to encapsulate and store object information. Hierarchical bounding box technology (BVH) combines bounding boxes into a tree structure, and the time complexity becomes log level [2]. If the upper node does not collide, it is not necessary to judge the following nodes. Standard bounding boxes include Sphere, AABB (axis-aligned Bounding Box), OBB (Oriented Bounding Box), and k-DOPs (Discrete Orientation Polymers) [3]. After selecting the bounding box, the next step is to establish the hierarchical bounding box. There are three construction strategies [4]: top-down construction strategy, bottom-up construction strategy, and incremental construction strategy. Among them, the top-down method has higher execution efficiency. Lauterbach [5] et al. 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 meager cost. Pantaleoni [6] et al. 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 [7] et al. Proposed a method using local density clustering, which constructs a hierarchical bounding box on the GPU by analyzing the distribution of entity density. 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 methods, and its leading research direction also focuses on the optimization strategy. Meister and Bitner [8] completed the incremental construction method of GPU and improved the construction speed. The incremental method will be better, but it is not easy to achieve in real-time at the cost of lower execution speed.

A very efficient dynamic simulation method needs to be used for real-time processing to realize the dynamic simulation between three-dimensional objects. People have tried various methods to represent deformable objects in real-time, and many methods based on GPU have been studied to improve the performance of cloth simulation. Tang et al. [9] proposed a GPU-based streaming algorithm to perform high-resolution and accurate cloth simulation. They map all components of the cloth simulation pipeline to the GPU-based kernel and data structure, including time integration, collision processing, and speed update. Then Tang et al. [10] proposed a parallelizable matrix assembly algorithm for time integration, which uses the coupling of time and space to accelerate. However, this method uses direct time integration and forces tiny time steps, which is prone to the problem of overstretching. In 2017, yarn-level cloth simulation and rendering [11-12] were proposed to simulate more realistic cloth behavior. Tang et al. [13-14] added the GPU based nonlinear solver to the simulation system and constructed the GPU based interactive cloth simulation system I-cloth; The cloth simulation system is accelerated by using the similar characteristics of GPU, and a collision elimination algorithm based on regular cone test and spatial hash is proposed, which is more accessible to GPU parallel. Recently, Tang et al. [15] proposed a collision processing scheme for discrete and continuous collision detection using a spatial hash to accelerate the

simulation system again. In these methods, based on high-performance GPU, the simulation calculation of high-resolution cloth takes several minutes per frame, which can not be executed in real-time. To pursue higher real-time performance, Kim M [16] and others proposed cloth simulation with effective collision detection for interactive AR applications in 2019. The Unity plug-in was used to design and implement a similar cloth simulation method using efficient collision detection technology. However, no collision elimination process slowed down the simulation efficiency. In 2020, Lyu M et al. [17] proposed a collision solution method based on the penalty function, which quickly deals with cloth contact friction and collision. It has improved the problem that some collisions will be ignored in a significant time step and enhanced the robustness of the simulation. However, the time cost is too high due to its simulation calculation based on the implicit Euler integral and Newton method. In 2021, Wang et al. [18] realized the submillimeter cloth fold simulation based on GPU. The simulation effect of this method is more refined, but it also increases the consumption of performance and time.

In this paper, a fast and efficient collision processing method is proposed and applied to cloth simulation. A set of BVH structure colliders are automatically generated for 3D objects by algorithm, and the BVH tree is generated by the method based on Morton code. For non-leaf nodes, the AABB bounding box is used to quickly eliminate the areas without collision and improve the elimination efficiency; The leaf node adopts a sphere bounding box to realize effective collision processing between 3D object and cloth model. It is developed as a Unity plug-in, which is widely used in real-time game development. The performance improvement of this method is given by comparing it with the existing cloth components and literature [16].

II. MATERIALS AND METHODS

The algorithm structure is divided into four modules: preprocessing module, time integration module, parallel processing module, and collision processing module. The preprocessing module preprocesses the rigid body when adding a rigid body to the scene, reads the grid information of the rigid body, and automatically generates a collider according to the grid information to simulate the shape of the object, to simplify the process of collision detection and collision response; In each iteration, the time integration module will time integrate the cloth particles and calculate the distance of the next time step according to the force condition of the particles; The parallel processing module implements the collision processing algorithm of spring particle model in parallel by calculating shaders to improve the parallelism; The collision processing module includes that after the particle passes through the time integration, the position and velocity change, and the model penetration may occur. Find the penetrating particle through collision detection, carry out collision response, and update the position and velocity of the particle.

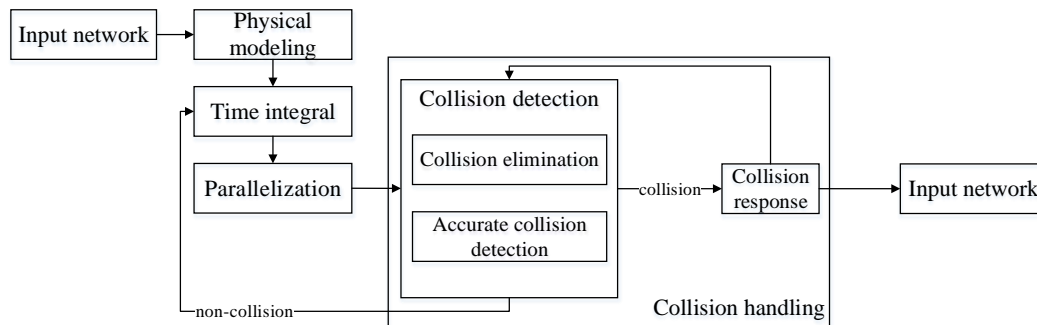


Fig 1: Simulation flow chart

The simulation flow is shown in

Fig 1. Firstly, the force analysis of each particle is carried out according to the particle spring model, and the current resultant force of each particle is calculated. Then, the particle's time integration is carried out using the displayed fourth-order Runge-Kutta integral method. Then the position and velocity of the particle are updated, and then the collision detection is carried out. If the particle collides with a rigid body, the collision response is carried out, and the iterative process of this simulation ends.

2.1 Preprocessing

In this study, the ideal particle spring model is adopted, and the elastic deformation force of the spring can be calculated by Hooke's law. Assuming that the set of adjacent particles of particle P_0 is R , the elastic deformation force of P_0 is represented by formula (1).

$$f_{elast} = \sum_{i \in R} c_e (|\overline{P_0 P_i}| - |\overline{P_i P_0}|_0) \cdot N_{|\overline{P_i P_0}|} \quad (1)$$

Where c_e represents the elastic deformation coefficient of the spring between two points, $|\overline{P_0 P_i}|$ represents the distance between particle P_0 and P_i at time t , $|\overline{P_i P_0}|_0$ represents the initially set distance between particles P_0 and P_i at zero time, $N_{|\overline{P_i P_0}|}$ represents the unit vector of P_i pointing to P_0 . P_0 and P_i determine, which one of the structural spring, shear spring and bending spring is according to the connection direction and the number of intermediate spacing particles, and then calculate the response force.

The change of deformation force may cause the vibration of cloth particles. To prevent the excessive vibration of particles and reduce the authenticity of simulation, the spring damping force f_{damp} is added, which is a function of the speed of particles, as shown in equation (2).

$$f_{damp} = -C_d \frac{\partial X}{\partial t} \quad (2)$$

c_d is the damping coefficient, and the damping force is positively correlated with the particle velocity, which can effectively prevent the untrue stretching and deformation of cloth caused by too fast particle vibration, to improve the fidelity of the simulation.

Based on the above contents, the mechanical equation can be established. In the model, the acceleration, velocity, and displacement of the particle [i, j] with time are expressed as formula (3).

$$\begin{cases} \vec{a}_{i,j(t)} = \frac{1}{m} \vec{F}_{i,j(t)} \\ \vec{v}_{i,j(t+\Delta t)} = \vec{v}_{i,j(t)} + \vec{a}_{i,j(t+\Delta t)} \Delta t \\ \vec{p}_{i,j(t+\Delta t)} = \vec{p}_{i,j(t)} + \vec{v}_{i,j(t+\Delta t)} \Delta t \end{cases} \quad (3)$$

Where Δt is the time step, $\vec{F}_{i,j(t)}$ is the resultant force of the particle [i, j] at time t, $\vec{a}_{i,j(t)}$ is the acceleration of the particle [i, j] at time t, $\vec{v}_{i,j(t)}$ is the velocity of the particle [i, j] at time t, and $\vec{p}_{i,j(t)}$ is the position of the particle [i, j] at time t.

2.2 Time Integral

In this paper, the Runge Kutta method is used to solve the time integral. Compared with the truncation error of the Euler method $o(h^2)$ and the truncation error of median method $o(h^3)$, the Runge Kutta method can obtain a higher truncation error. Its idea is to predict the slopes of multiple points in (x_n, x_{n+1}) , so as to construct a more precise time integration formula. The fourth-order Runge Kutta integral method predicts four slopes in each time step, and its truncation error is $o(h^5)$.

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_2 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \\ k_3 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2) \\ k_4 &= f(x_n + h, y_n + hk_3) \\ y_{n+1} &= y_n + \frac{1}{6}hk_1 + \frac{1}{3}hk_2 + \frac{1}{3}hk_3 + hk_4 \end{aligned} \quad (4)$$

It can be seen from formula (4) that the fourth-order Runge Kutta integral method needs more

operations and occupies more memory space than the Euler method, but its advantage is that it can improve the accuracy and stability of simulation, which is very suitable for simulation with large time step.

2.3 Parallelization

Compute shader is a shader that can be used in any shading stage. It provides parallelism for general tasks. To improve the real-time performance, this paper uses compute shaders to implement the collision processing algorithm of the spring particle model in parallel.

The user-defined calculation space controls the number of calls to the compute shader. The calculation space can be defined as one-dimensional, two-dimensional, or three-dimensional space. The calculation space is divided into several groups, and each group contains many calls to the compute shader. For each different compute shader, different global space and local space can be defined. This paper uses three compute shaders to speed up the simulation process. **TABLE I** shows the name, function, and global and local space each compute shader uses.

TABLE I. Global space and local space used by each compute shader (n is the number of particles)

Shader name	Function description	Global space	Local space
NodeUpdate	Calculate the current stress condition for each node and perform time integration	(n/8,n/8,1)	(8,8,1)
CollisionDecton WithBVH	Collision detection and collision response are performed for each node	(n/8,n/8,1)	(8,8,1)
normalShader	Calculate the normal vector of each face of the cloth for subsequent rendering	(n/8,n/8,1)	(8,8,1)

The compute shader does not have any direct user-defined inputs and outputs. The compute shader obtains data directly from memory through the image access function or the use of shader storage buffer objects (SSBO). A shader buffer object is a particular buffer object that can be written and read in a GLSL shader. Their size is limited by the amount of GPU memory available and up to 16kb. This article created multiple SSBO to facilitate access to information. **TABLE II** shows the SSBO defined for each shader.

TABLE II. SSBO used by each compute shader

Shader name	SSBO name	Stored information
NodeUpdate	Position	Local coordinate information of each particle
	Velocity	Velocity of each particle in local coordinate system
	PositionTemp	World coordinate information of each particle
	prevPosition	World coordinate information of the previous step of each particle
	trsMatrix	A matrix that converts local coordinates into world

		coordinates
	VelocityTemp	The velocity of each particle in the world coordinate system
CollisionDectonWithBVH	PositionTemp	World coordinate information of each particle
	VelocityTemp	Velocity of each particle
	PrevPosition	World coordinate information of the previous step of each particle
	Root	BVH structure of rigid body model
NormalShader	Position	Local coordinate information of each particle
	Normal	Normal vector of each point

The pseudo-code for calculating the shader NodeUpdate is shown in algorithm 1. The number of threads enabled by the shader is the same as the number of soft body particles. Each thread will obtain a global ID and then use this ID to obtain the coordinates, speed, and other information of the processed particles. It also has a matrix that can convert the local coordinate system of the model into the world coordinate system. This shader updates the position and velocity of the particle in the local coordinate system and the world coordinate system. In Algorithm 1, lines 4 and 5 calculate the resultant force generated by the particle's adjacent spring, calculate the particle's acceleration in line 6, update the velocity in lines 7 and 8, and calculate the new local coordinates with the fourth-order Runge Kutta integral. Calculate the coordinates and velocity of the particle in the global coordinate system in lines 9 and 10.

Algorithm 1: NodeUpdate shader

Input: SSBO(Position, Velocity, trsMatrix), nodeMass

Output: SSBO(Position, PositionTemp, prevPosition, Velocity)

1:Begin

2: currentNode=SV_DispatchThreadID;

3: prevPosition= trsMatrix*Position[currentNode];

4: for each neighborNode

5: Calculate the resultant force between the particle and the adjacent node

6: float a =force/nodeMass;

7: Velocity[currentNode] = velocity[currentNode] + a * delta Tim

8: Updating the local coordinates of particles by fourth-order Runge Kutta integral

9: PositionTemp[currentNode] = trsMatrix * Position[currentNode];

10: VelocityTemp[currentNode]= trsMatrix* Velocity [currentNode];

11:End

The pseudo-code for calculating the shader CollisionDectonWithBVH is shown in Algorithm 2. The enabled threads are the same as the shader NodeUpdate, and the number of soft body particles is the same. Each thread first obtains its global ID, then obtains the particle information to be processed according to this global ID, and then uses this particle to traverse the BVH of the model in a non-recursive depth-first method. If the particle collides with the sphere Collider, the collision response is performed to update the

position and velocity of the particle.

Algorithm 2: CollisionDectonWithBVH

Input: SSBO(PositionTemp,VelocityTemp,PrevPosition,Root)

Output: SSBO(PositionTemp,VelocityTemp)

```
1: Begin
2:   currentNode=SV_DispatchThreadID;
3:   Stack.Pust(root)
4:   while(!Stack.empty())
5:     targetNode=Stack.pop();
6:     if(Overlap(targetNode))
7:       if(targetNode==leafNode)
8:         Accurate collision detection and collision response
9:       Else
10:        Stack.Pust(root->rightChild);
11:        Stack.Pust(root->leftChild);
12:End
```

In calculating the pseudo-code of the shader NormalShader, as shown in Algorithm 3, the number of enabled threads is the same as the number of prime points. Each thread first obtains its ID in the global, reads the particle's coordinates, the overall number of rows and columns, and then uses the shader to calculate the normal vectors of all faces connected by the particle, normalizing the vectors calculating the normal of the particle.

Algorithm 3: NormalShader

Input: SSBO(Position), id, nodeRow, nodeCol

Output: SSBO(Normal)

```
1:Begin
2:  if(id.y < nodeRow-1)
3:    if(id.x<nodeCol-1)
4:      Calculate the normal vector of the two faces associated with the lower right of the node
5:    else
6:      Calculate the normal phasor of the two faces associated with the lower left of the node
7:  if(id.y > 0)
8:    if(id.x<nodeCol-1)
9:      Calculate the normal phasor of the two surfaces associated with the upper left of the node
10:  else
11:    Calculate the normal vectors of the two faces associated with the upper right of the node
12:  Unitized vector
13: End
```

2.4 Collision Handling

In soft body simulation, the collision types between objects can be divided into soft body rigid body collision and soft body self-collision. This paper focuses on the collision between soft body and rigid body and focuses on the collision processing algorithm between soft body and rigid body.

2.4.1 Pretreatment of rigid body model

Compared with the previous method of constructing bounding boxes for each model's triangular surface to ensure the simulation process's authenticity, this paper proposes a method to simplify the model by constructing a group of BVH structure colliders. Firstly, an AABB is constructed for the whole model, and then the AABB is divided into multiple voxels according to the needs of the situation, and a sphere collider is obtained for each voxel. The collider can surround all the vertices contained in the voxel. The sphere collider is the leaf node in the BVH structure tree, which represents the actual collision part of the rigid body model. AABB collider is a non-leaf node. The soft body points do not have to collide with each face of the model for collision detection, which is used to eliminate the non-collision area quickly.

2.4.2 Selection of bounding boxes and parallel construction of BVH

A sphere bounding box is simple in structure, but its tightness is very poor. It is suitable for scenes with a large amount of rotation. OBB is much more robust in compact than AABB and sphere, but the complex construction methods will reduce the real-time performance. K-DOP is very important for the selection of K value. 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, so this paper chooses to use AABB and a sphere bounding box to construct BVH. This paper uses the top-down method based on Morton code to construct the BVH tree. The top-down method means that in the collision process, the bounding box is constructed for the whole target collision object as the root node, the target collision object is subdivided into small geometry according to specific rules, and the bounding box is constructed for the geometry, which can also be subdivided into smaller geometry. In this way, the structure subdivided in turn is similar to the tree structure, and the leaf nodes constitute the minimum geometric unit of the target collision.

According to the above method, first, calculate their Morton code according to the center point of each element. Then, the Morton code is used to sort the elements, and then the elements will be placed in a spatially coherent manner. The BVH is constructed by recursively dividing the range of the current root node. For the current root node covering the range $[i, j]$, find the maximum difference between Morton codes in the range $[i, j]$, recorded as γ . Then, the left child node of the current node overrides $[i, \gamma]$, Right child node overlay $[\gamma + 1, j]$. Perform this step recursively until the current node contains only one entity.

The BVH constructed in this paper is a complete binary tree structure; if there are N leaf nodes, there will be N-1 non-leaf nodes. Each non-leaf node is processed in parallel[19], and the bounding box hierarchy is divided according to the maximum difference of Morton code. This method only needs to allocate an array

composed of N-1 non-leaf nodes and then process all non-leaf nodes in parallel. Each thread first needs to determine the object range corresponding to its node and then continue to divide this range in the same way. Finally, the sub-nodes are selected for nodes according to their respective sub-ranges. If there is only one object in the child range, it must be a leaf node. Otherwise, another non-leaf node in the array is referenced.

This construction method is shown in

Fig 2. According to this method, the hierarchical bounding box of the model is generated.

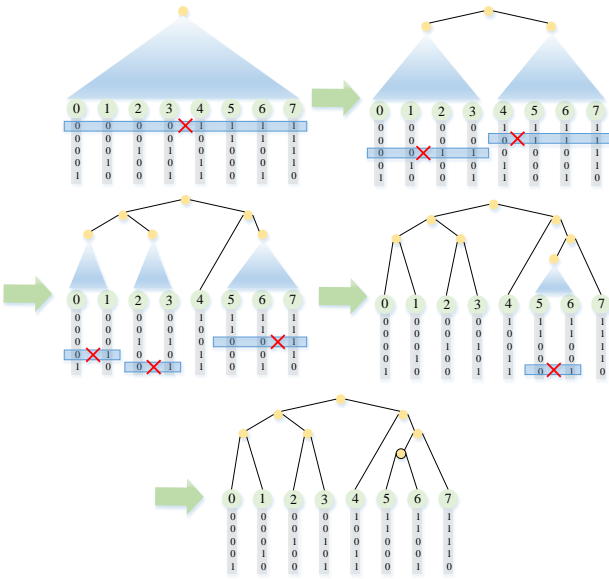


Fig 2: Schematic diagram of division according to the maximum difference position between Morton codes

2.4.3 Traversal of hierarchical bounding box

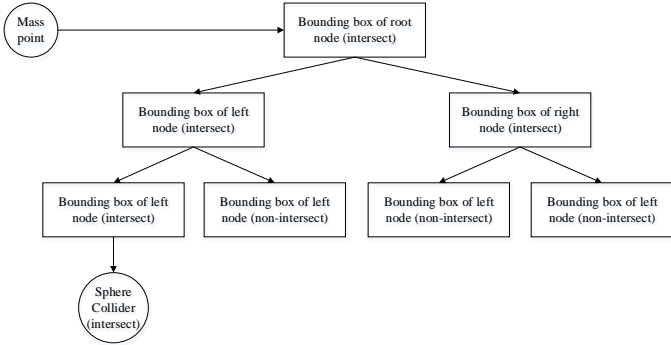


Fig 3: Traversal process of particle and hierarchical bounding box tree

The particle traverses the BVH of the model in a non-recursive depth-first method. When the particle traverses the BVH of the rigid body model, the particle performs an intersection test on all nodes of a layer of BVH. If they do not intersect, the particle does not intersect with the model. The traversal process is shown in

Fig 3.

2.4.4 Collision response

In this study, if a collision is detected between the cloth particle and the sphere, that is, the distance between the particle and the sphere's center is less than the radius of the sphere, the collision response needs to be carried out. You need to adjust the position of the particle to the first point where the particle collides with the sphere along the velocity direction, and correct the velocity of the particle to retain only the tangential velocity with the sphere. As shown in

Fig 4, the position of the particle at t_0 is P_0 and the velocity is V_0 . After a step-by-step time integration, the position at t_1 is P_1 and the velocity is V_1 . At this time, the distance between the particle and the center of the sphere is less than the radius of the sphere, the particle collides with the sphere, the point P'_1 where the particle collides with the sphere is calculated, and the position of the particle is modified to P'_1 , Then the velocity is changed to the tangential velocity of the original velocity along the direction of the spherical shell. Thus, the collision response process is completed.

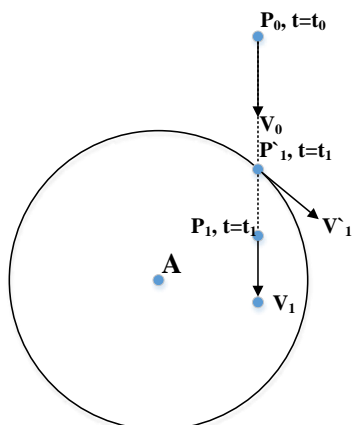


Fig 4: Collision response of particles

III. CONCLUSION

3.1 Experimental Environment



Using the collision detection method proposed in this paper, the cloth model is implemented as a Unity plug-in. The collision test is carried out on various three-dimensional objects. The experiment was carried out in the computing environment shown in TABLE III. This paper uses more and more complex models than the number of vertices and faces in literature [16], as shown in TABLE IV. The performance is compared with the Unity cloth component and the method in the document [16].

TABLE III. Table of experimental environment parameters

Assembly	Explain
----------	---------

Operating system	Windows 10 home Chinese
Unity	Unity 2019 2.3 f1
CPU	Intel Core i5 10400F
RAM	16GB
GPU	NVIDIA GeForce GTX1650
VRAM	8GB

TABLE IV. Test models for collision test with the proposed cloth model

Model name	Bunny	Dragon
		
Number of vertices	35295	104872
Number of triangular faces	70580	209227

3.2 Parameter Initialization

Before the simulation process, the cloth parameters should be initialized. The description of the parameters is shown in **TABLE V** below. After setting the cloth parameters, the simulation process can be started.

TABLE V. Description of cloth parameters

Parameter name	Explain
Vertex Column	Number of particle rows
Vertex Row	Number of particle columns
Cloth Size	Length of cloth
Cloth Mass	Fabric quality
Looper	The number of simulation iterations performed in each frame
SpringK	Elastic coefficient of spring
Damping	Damping coefficient of fabric simulation
Externa Force	Non gravity external force

3.3 Effect Demonstration

Firstly, show the simulation effect of the cloth with two ends fixed, set the coordinates of the two ends of the

cloth, and then the cloth sags under the action of gravity. The simulation effect is shown in Fig 5.

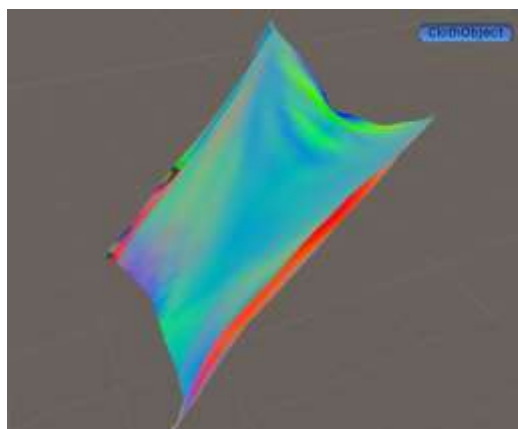


Fig 5: Cloth simulation diagram

TABLE VI. The results of BVH structure collider generation for the test models



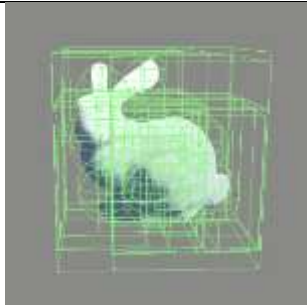
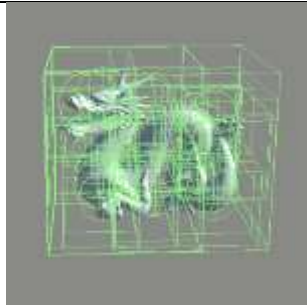
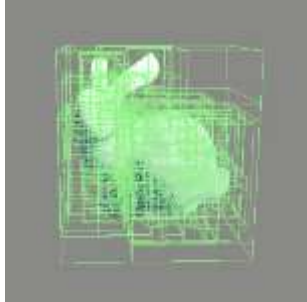
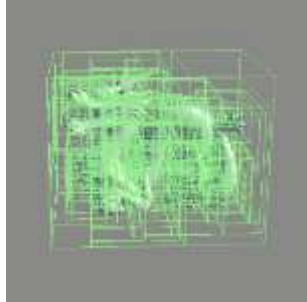
Model name	Bunny	Dragon
1 * 1 * 1 grid		
3 * 3 * 3 grid		
5 * 5 * 5 grid		

TABLE VI shows the process of automatically generating sphere colliders and box colliders according to the accuracy of models and voxels. The precision of voxels used in the experiment is set to $1 * 1 * 1$, $3 * 3 * 3$ and $5 * 5 * 5$. The higher the precision of voxels, the tighter the collection of colliders.

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 three execution times; The collision between the cloth component provided by the Unity game engine and a sphere collider connected to each model. The method in document [16] and the method in the text generate a set of collisions between BVH structure colliders and cloth objects. The cloth model used in each experiment is a $64 * 64$ grid structure compared in the scene with collision.









TABLE VII compares dynamic performance time between the method in this paper and Unity components and literature [16]. The experimental results show that in the collision scene, the execution time of each frame of the method proposed in this paper is shorter, which has higher simulation efficiency than the Unity cloth component and the method in literature [16]. On the premise of obtaining the same simulation effect as literature [16], the dynamic performance time is reduced by 40%. With the improvement of accuracy, this method's maximum dynamic performance time is only increased by 0.004s. **Error! Reference source not found.** intuitively compares the simulation results of the cloth component and the method in this paper. The $5 * 5 * 5$ precision method proposed in this paper can more realistically display the simulation effect of the object than Unity collision processing based on a single Collider and cloth component.

TABLE VII. Comparison of simulation performance of test models used in different numbers and structures of Colliders

Algorithm	Network Structure	64 * 64 cloth model	
		Bunny	Dragon
Unity cloth & one collider	$1 * 1 * 1$	0.211s	0.247s
Method of reference [16]	$1 * 1 * 1$	0.017s	0.020s
	$3 * 3 * 3$	0.019s	0.020s
	$5 * 5 * 5$	0.025s	0.025s
This paper	$1 * 1 * 1$	0.008s	0.013s
	$3 * 3 * 3$	0.012s	0.013s
	$5 * 5 * 5$	0.012s	0.015s

TABLE VIII. Cloth simulation results of the test model

Algorithm	network structure	64 * 64 cloth model
-----------	-------------------	---------------------

		Bunny	Dragon
Unity cloth & one collider	1 * 1 * 1		
This paper	1 * 1 * 1		
	3 * 3 * 3		
	5 * 5 * 5		

IV. CONCLUSIONS

This paper designs and implements a novel cloth simulation method based on similar BVH collision detection technology. In the proposed method, a set of BVH structure colliders are generated for the 3D objects colliding with the cloth model. The collision processing is performed in parallel between each particle and each collider of the cloth model. BVH structure is used to accelerate the collision detection process. The tree structure of BVH can significantly accelerate the collision detection process between particles and rigid bodies and improve collision detection and elimination efficiency. At the same time, the parallel execution process also makes full use of the computing power of GPU. To prove the performance of the proposed method, this paper compares the performance with the cloth component provided by Unity

and other cloth simulation methods. The experimental results show that this method reduces the cumbersome human tasks and can more accurately deal with the collision between 3D objects and cloth in real-time. Improve the execution efficiency and realize real-time collision under the condition of ensuring high-quality collision. In the experimental test, two different test models are used to test the collision of the cloth model. For the 64 * 64 grid, under the condition of higher accuracy, the dynamic performance time is still better than the method in literature and the structure of cloth components with a single collider. At present, only the collision between soft body and rigid body is considered, and it is difficult to deal with friction, penetration, etc. In the future, it is considered to combine the latest collision processing algorithm to improve the realism further, improve the collision response process and the self-collision processing process of cloth, realize the self-collision detection algorithm of cloth, and apply it to the virtual fitting scene combined with MR, to provide users with more immersive cloth simulation effect and more realistic visual feedback.

REFERENCES

- [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 F. AABB bounding box collision detection algorithm based on B + tree storage. Computer science, 2021,48 (S1): 331-333 + 348
- [4] Qu H. 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. Oxford, UK: Blackwell Publishing Ltd, 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 Conference on High Performance Graphics. 2010: 87-95.
- [7] Hu Y, Wang W, Li D, et al. Parallel BVH construction using locally density clustering. IEEE Access, 2019, 7: 105827-105839.
- [8] Meister D, Bittner J. Parallel reinsertion for bounding volume hierarchy optimization//Computer Graphics Forum. 2018, 37(2): 463-473.
- [9] Tang M, Tong R, Narain R, et al. A GPU-based streaming algorithm for high-resolution cloth simulation//Computer Graphics Forum. 2013, 32(7): 21-30.
- [10] Tang M, Wang H, Tang L, et al. CAMA: Contact-aware matrix assembly with unified collision handling for GPU-based cloth simulation//Computer Graphics Forum. 2016, 35(2): 511-521.
- [11] Chen J, Zheng Y, Song Y, et al. Cloth compression using local cylindrical coordinates. The Visual Computer, 2017, 33(6): 801-810.
- [12] Cirio G, Lopez-Moreno J, Otaduy M A. Yarn-level cloth simulation with sliding persistent contacts. IEEE transactions on visualization and computer graphics, 2016, 23(2): 1152-1162.
- [13] Tang M, Wang T, Liu Z, et al. I-Cloth: Incremental collision handling for GPU-based interactive cloth simulation. ACM Transactions on Graphics (TOG), 2018, 37(6): 1-10.
- [14] 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.
- [15] Li C, Tang M, Tong R, et al. P-cloth: interactive complex cloth simulation on multi-GPU systems using dynamic matrix assembly and pipelined implicit integrators. ACM Transactions on Graphics (TOG), 2020, 39(6): 1-15.

- [16] Kim M, Sung N J, Kim S J, et al. Parallel cloth simulation with effective collision detection for interactive AR application. *Multimedia Tools and Applications*, 2019, 78(4): 4851-4868.
- [17] Lyu M, Wan Y, Zhao W, et al. Fast Cloth Collision Detection and Effective Contact Friction Algorithm. *Journal of Computer-Aided Design & Computer Graphics*, 32(3): 392-400.
- [18] Wang H. GPU-based simulation of cloth wrinkles at submillimeter levels. *ACM Transactions on Graphics (TOG)*, 2021, 40(4): 1-14.
- [19] 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.