HapticForceCalculationAlgorithm

author : wanglin

date: 2023-5-11

Overview

When we use the force feedback device,with Unity application, we often use sofa or other function to calculate force of touch, different from Unity's physics rigid body motion mechanics,touch of force is not provided for calculation.

Therefore, this paper provides a method to calculate the force of touch with a way of penalty.

Different from the traditional spring model to calculate the haptic force, the penalty method provided in this paper calculates the force more accurately.

There are many ways to compute haptic forces, and if that doesn't work well, we can compute them in an optimized constrained way or other better ways, if necessary, and if I have time, I will come up with a better algorithm for this.

Thanks for reading,and suggestions are welcome!

Algorithm Principle

For collision efficiency, we do not use Mesh for force calculation,I choose collider instead. I used a greedy algorithm to fill the Mesh with spheres of different sizes,and before this, I need to voxelize the mesh,the mesh will be filled with voxels directly, or an octree is used to divide the mesh into patches of specified resolution, then, the filling mesh is computed based on these voxels or patches.

Voxelized Mesh and Surface Packing

We need to create a boundingbox arround the complex mesh , and confirm that the mesh is closed. Grid the boudingbox according to the customized specified resolution (box size) ,determine whether the voxel (segmented box) is inside or outside the complex mesh , just like SDF,the outside voxels were eliminated .

We have to calculate a distance map (we could use physical raycast from unity,if not ,BVH instead). we need to find the face which the voxel is closest to the complex mesh,and calculate the

distance and iterate each voxel.

Because the inner sphere rotation is constant, alter the transform only affect the position we use a greedy algorithm.

First,we choose the voxel farthest from the face to fill a largest ball,the radius is the voxel to the face, then,Step by step filling, so,the collision complexity is related to the number of sphere collider.

Because of the loss of significance,there are requirements for the thickness of the grid division and the size of the sphere collider, which need to be balanced.

Finally, we filled the complex mesh with sphere collider large and small.

Voxelized Mesh

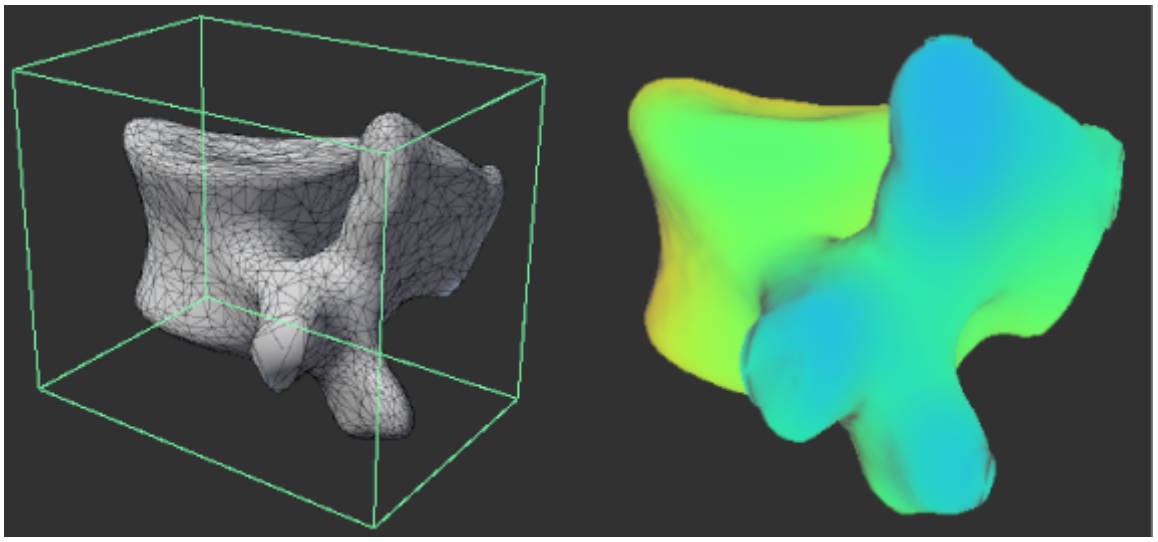
Signed Distance Fields

A Signed Distance Field (SDF) is a 3D texture representation of mesh geometry. To represent the geometry, each texel stores the closest distance value to the surface of the mesh. By convention, this distance is negative inside the mesh and positive outside. This texture representation of the mesh enables you to place a particle at any point on the surface, inside the bounds of the geometry, or at any given distance to it.

I iterate the SDF by ComputeBuffer,create a computeShader and define a computeBuffer to iterate SDF and save the result.I use computeShader.SetBuffer() and .Dispatch() to save and call computeShader. In computeShader, I use thread id and group id to iterate SDF data(make sure that every thread could call on SDF data).Result has been writen in OutputBuffer,and use .Getdata() to write in output buffer and save to disk.

However,there's some exceptions . such as the fileBytes 's length is "(Multiple of 4) + 1",and property (Result) at kernel index (0) is not set ,etc.

So,I 'd like to use another way to implement this func,and when I have been done this thing,I will come back to solve these problems.

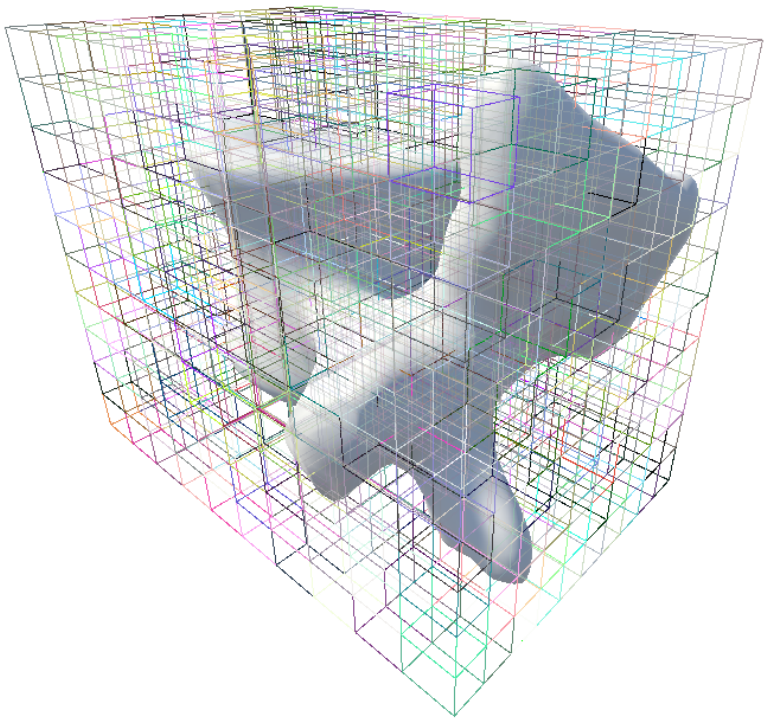


```
var sdfData = ReadSdfDataFromFile(path);
_sdfBuffer = new ComputeBuffer(sdfData.Length, sizeof(float));
_sdfBuffer.SetData(sdfData);
_outputBuffer = new ComputeBuffer(sdfData.Length, sizeof(float));
computeShader.SetBuffer(0,"sdfBuffer",_sdfBuffer);
computeShader.SetBuffer(0,"outputBuffer",_outputBuffer);
computeShader.Dispatch(0,sdfData.Length/64,1,1);
var outputData = new float[sdfData.Length];
_outputBuffer.GetData(outputData);
SaveSdfDataToFile(outpath,outputData);
_sdfBuffer.Release();
_outputBuffer.Release();
```

Voxelization technology based on Octree

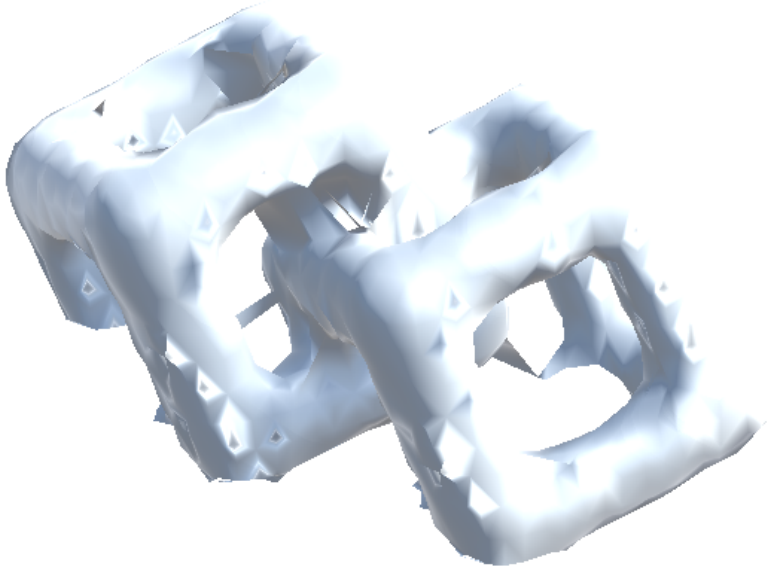
Octree is a hierarchical structure that divides a three-dimensional space into a series of octree nodes, each of which represents a cube volume and is subdivided into smaller child nodes as needed. Using Octree's voxelization technology can improve efficiency while maintaining high quality voxelization.

I'll implement it later.



Native SDF

Evaluate signed-distance-fields with great efficiency using the power of the Unity Job System and the Burst Compiler.



This example is complex and I could not found an API to turn mesh into voxel. However,it provides an efficient way of computing, using the Job system to carry out parallel computing, scheduling multiple cores at the same time. I looked into this technique and planned to use it for SESS3.0 and the rest of this article.

Suface Packing

This module has not yet been processed.

Penalty Calculation Method - volume

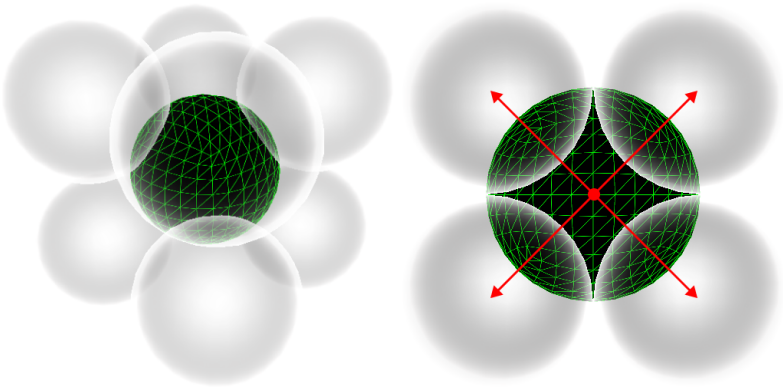
Principle

The direction of the penalty force can be derived from the weighted average of all vectors between the centers of colliding pairs of shperes,weighted by their overlap.

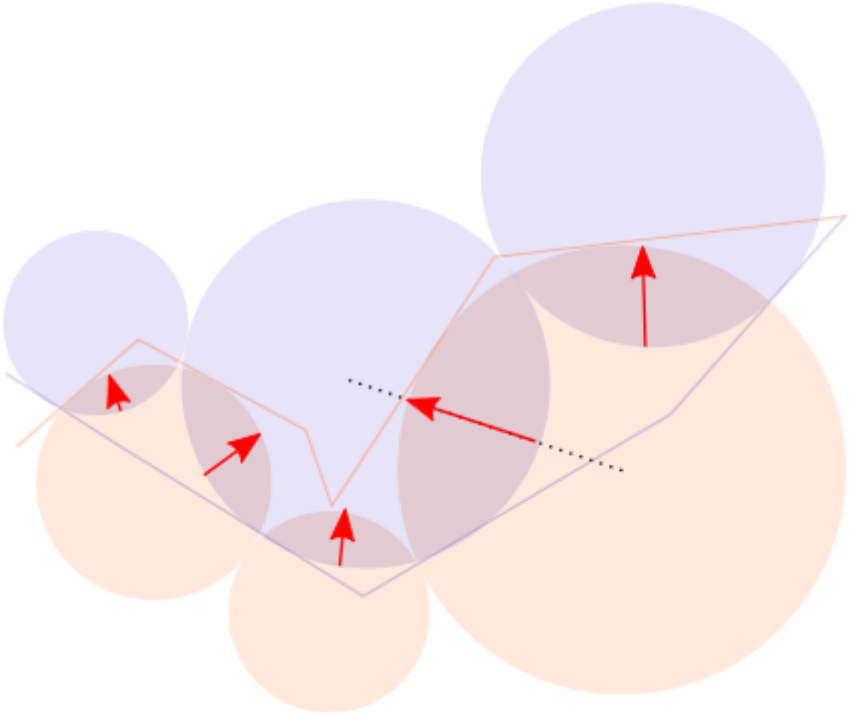
A simple heuristic would be to consider all overlapping pairs of spheres separately. Penetration volume of two spheres with radius r1 and r2 respectively.

```
foreach (var sphere1 in _dynamicList)
{
    foreach (var sphere2 in _staticList)
    {
        if (sphere1.bounds.Intersects(sphere2.bounds))
        {
            var force = Common.PenaltyForce(sphere1, sphere2);
            penaltyForce += force;
        }
    }
}
```

It's relatively easy to understand how to use volume overlap to calculate penalties.The simplest way is to nest over the colliding objects and then calculate the force using a formula.

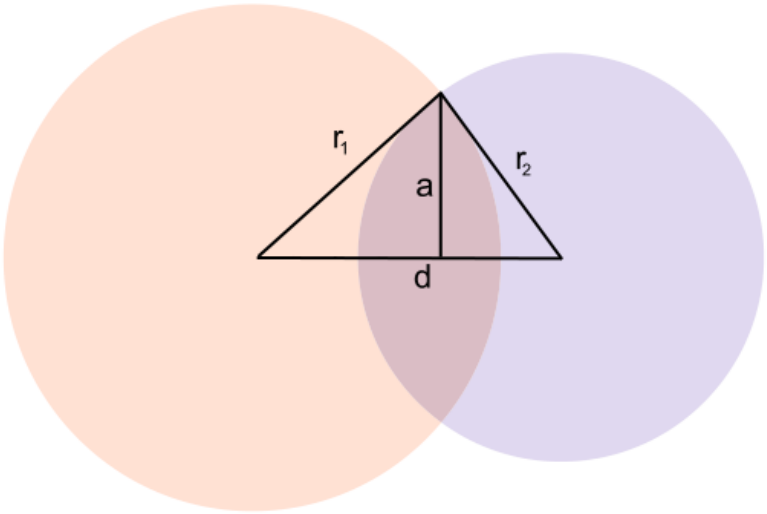


However, this calculation also has a problem, when a collision sphere is wrapped by 8 touched spheres, the force is 0, and the same direction and reverse judgment is needed, of course, the subsequent calculation method also inherits this way.



The direction of the penalty force can be derived from the weighted average of all vectors between the centers of colliding pairs of spheres , weighted by their overlap.

A simple heuristic would be to consider all overlapping pairs of spheres(R_i, S_j) secparately. Let c_i,c_j be their sphere centers and n_{ij} = c_i-c_j.Then we compute the overall direction of the penalty force as the weighted sum n = $\sum_{i,j}$Vol($R_i \cap S_j$)·n_{ij}



Penetration volume of two spheres with radius r_1 and r_2,respectively

Consequently,we get for thr total intersection volume V for two spheres:

$V = V(r_1,h_1)+V(r_2,h_2)$

$= \pi(r_1+r_2-d)^2(d^2+2dr_2- 3r_2^2+2dr_1+6r_1r_2-3r_1^2)/12d$

Summarizing,formula allows us to compute the overlap between a pair ofspheres efficiently during the traversal.

Algorithm and its time-critical derivative return a set of overlapping spheres orpotentially overlapping spheres, respectively. We compute a force for each of thesepairs of spheres (R_i,S_j) by

$\mathbf{f}(R_i) = k_c Vol(R_i \cap S_j)\cdot \mathbf{n}_{R_i}$

where k_c is the contact stiffneess,$Vol(R_i \cap S_j)$ is the overlap volume,and $\mathbf{n}_{R_i}$ is the contact normal.

Summing up all pairwise forces gives the total penalty force:

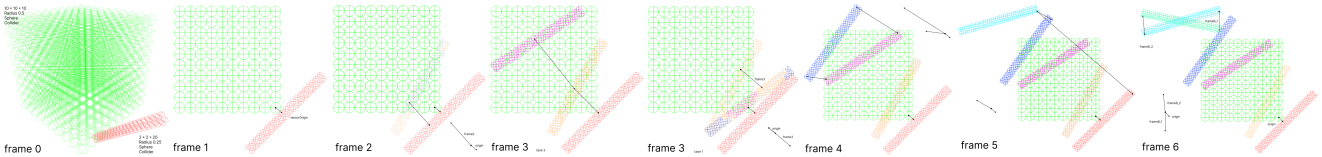
$\mathbf{f}(R) = \sum f(R_i)$

$R_i \cap S_j \neq \emptyset$

Calculation Method - distance

To calculate the penalty force without mesh collide and using lots of sphere colliders ,we need to split the mesh and fill the sphere first, this section does not discuss these and focuses on computing collisions between collections of sphere colliders.

Overview



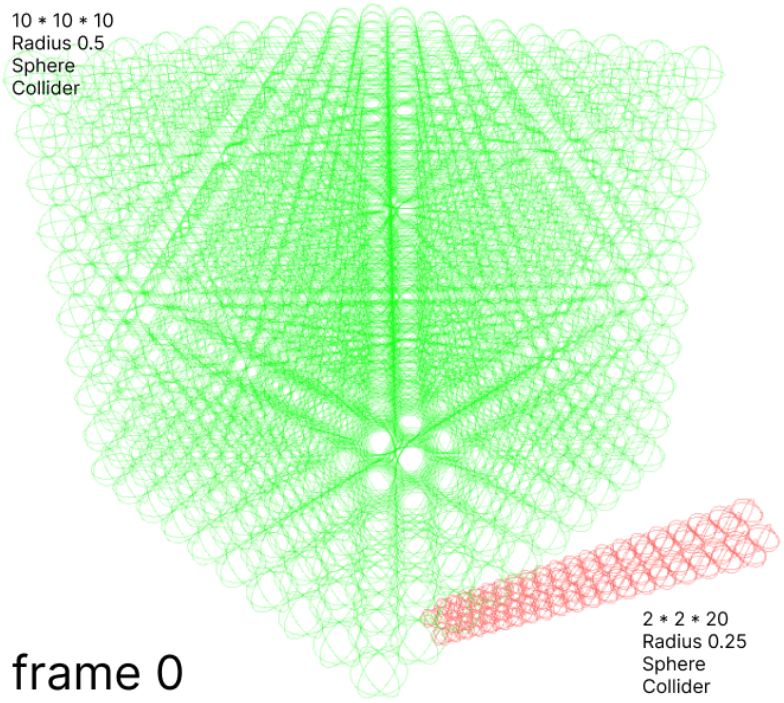
The main principle of this algorithm is as follows:

- Calculate each sphere's distance between its origin position
- Calculate the sum of the distances as penalty force

As for which spheres to calculate, and under which case, I will explain in detail below.

I broke down the force calculation step into several steps,explain this in terms of per-frame computation

Frame 0



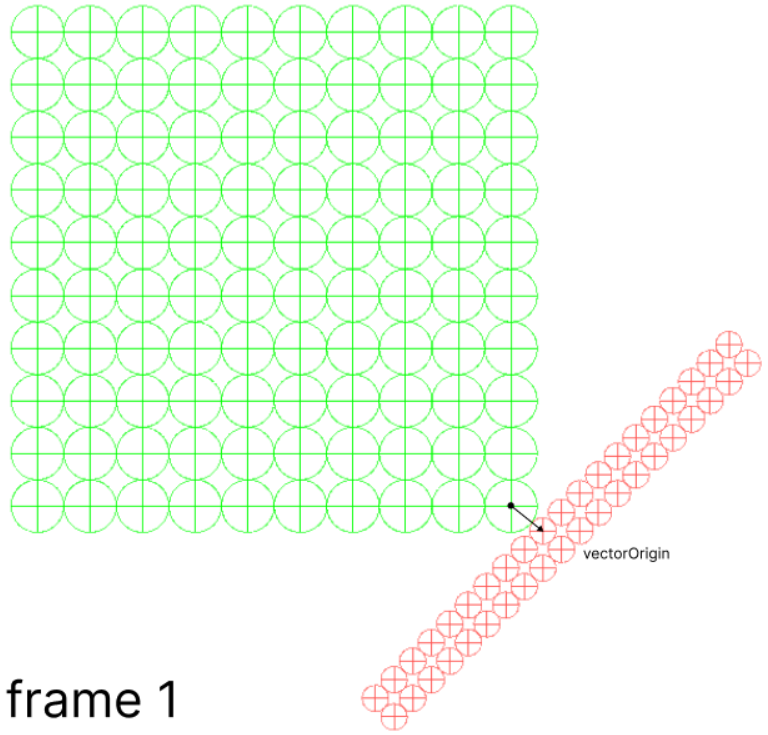
As an illustration, I have divided a cube into 1000 spheres by splitting it 10 * 10 * 10.

In fact, the sphere should not be filled evenly like this,but for convenience, as well as to raise the threshold of the amount of computation,I did that.

Also,I have divided a cylinder as collider tool into 80 spheres by splitting it 2 * 2 * 20. To make the calculation more difficult, they have different sphere radii.

- `Box Collection` : 1000 `SphereColliders` with radius `0.5f` and `Is Trigger` is `on`
- `Cylinder Collection` :80 `SphereColliders` with radius `0.25f` , `Rigid body` is contained and `Is Kinematic` is `on`
- `Box Collection` is static
- `Cylinder Collection` is dynamic

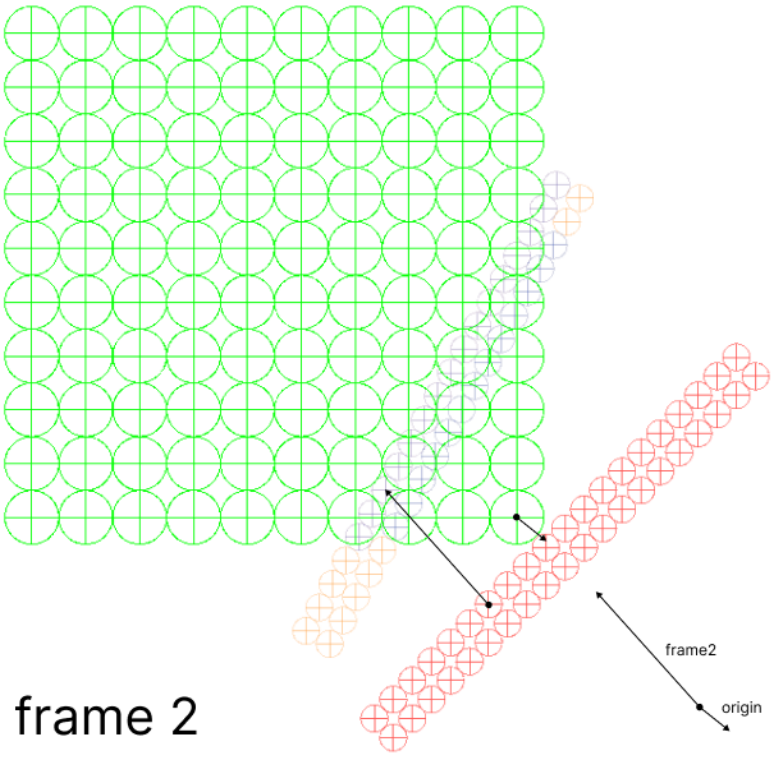
Frame 1



If and only if the `SphereCollider` in `Cylinder Collection` and the `SphereCollider` in `Box Collection` first collide

- Record the position of all `SphereColliders` in the `Cylinder Collection` and write it to the `Dictionary<DynamicCollider, Vector3>`
- Use `IsInitialCollision` to detect whether there are any collisions in the scene
- Record the `OriginVector`

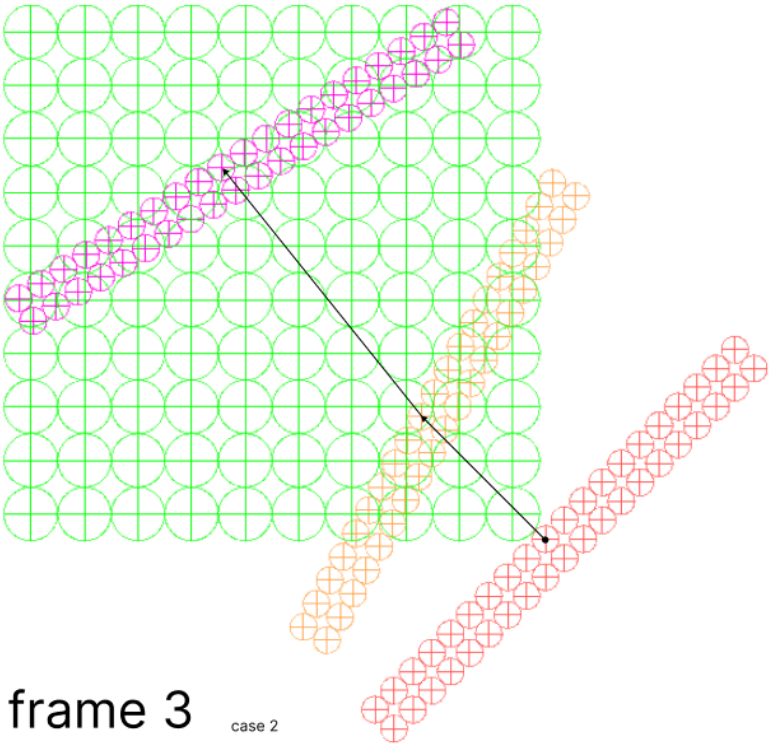
Frame 2



When some `SphereColliders` in `Cylinder Collection` penetrates into `Box Collection`

- I marked them with blue in the figure
- Create a List<DynamicCollider>
- `OnTriggerStay`
  - Check if this exists in List
  - if not , add it
  - record the previous frame postion and current position
- `OnTriggerExit`
  - If the sphere collider leaves the Collection in the same direction as `OriginVector` , it will be removed from the List
- `Update`
  - Iterate the List ,if item as same as someone in Dictionary
  - Calculate the force using its previous pos and current pos
  - Detect whether if there are collisions in scene
  - If there's no collisions in scene,detect if penelty end by calculate some direction

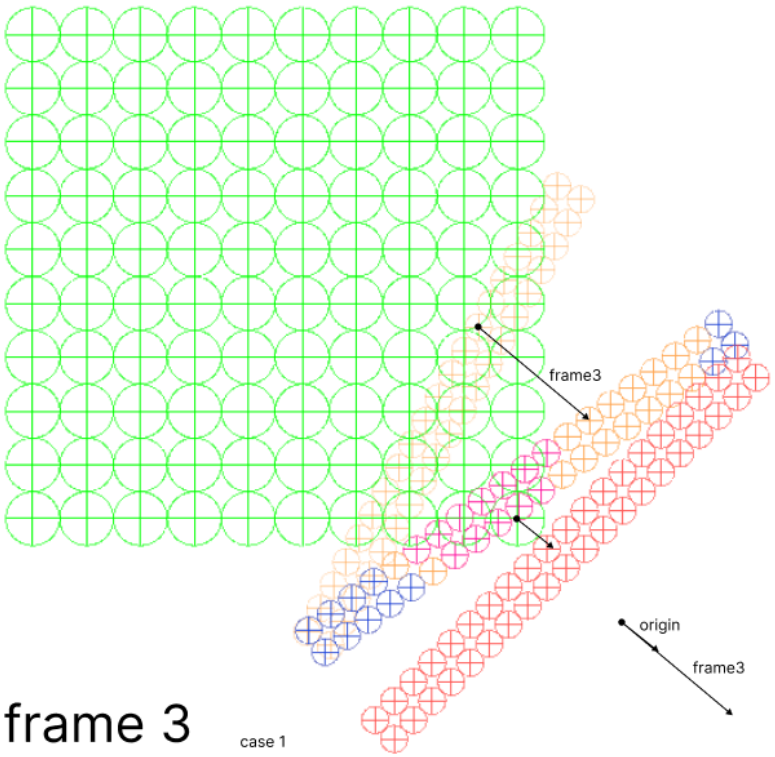
Frame 3 case 1



We can see that in the third frame, the `Cylinder Collection` falls back, and the motion vector of the `sphere collider` identified in the figure at this time is `frame3`

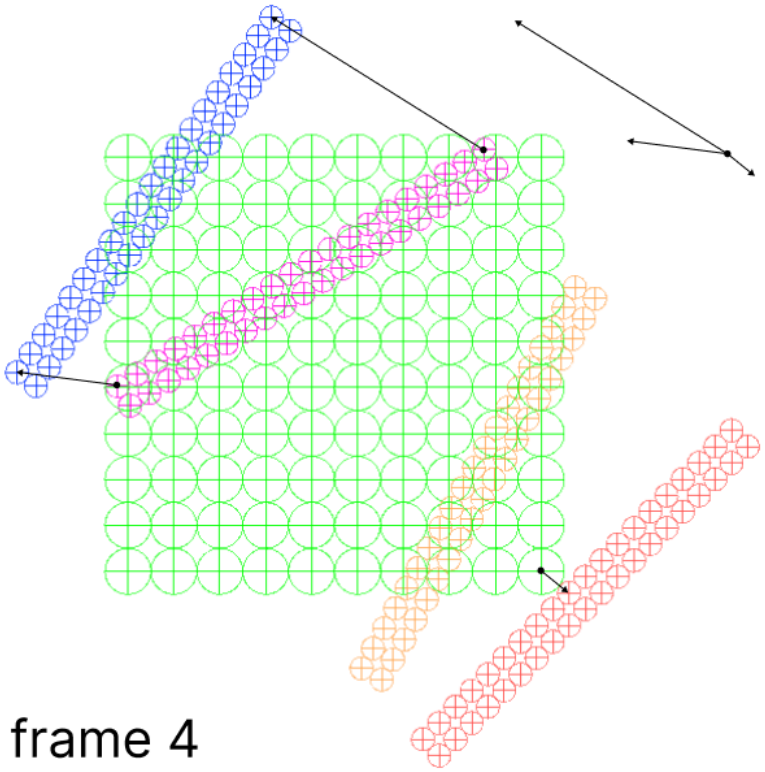
`frame3` and `origin` are in the same direction, so this `sphere collider` is removed from the `collisionList`

Frame 3 case 2



In this case,everything goes easy

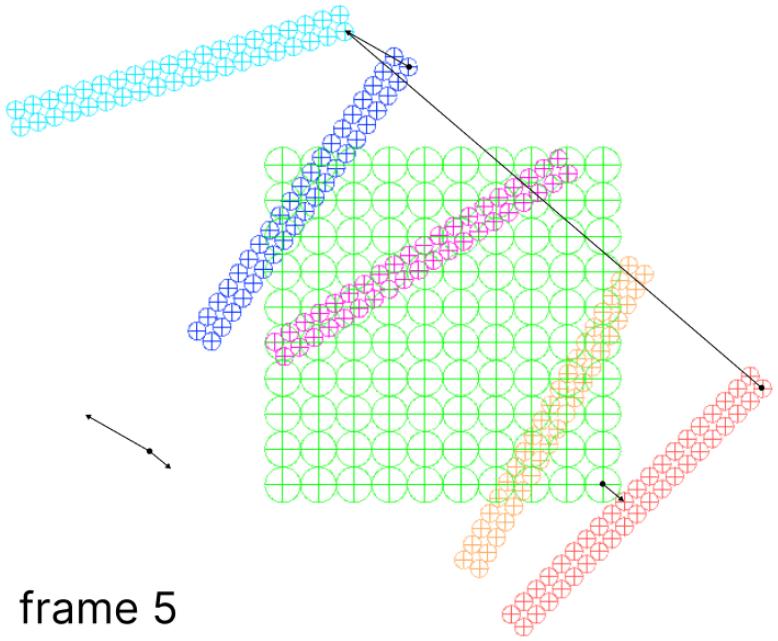
Frame 4



Although some of the `sphere collider` s are out of the `Box Collection` zone

they are still in the `collisionList` and reverse the `origin` , so the force continues to be calculated

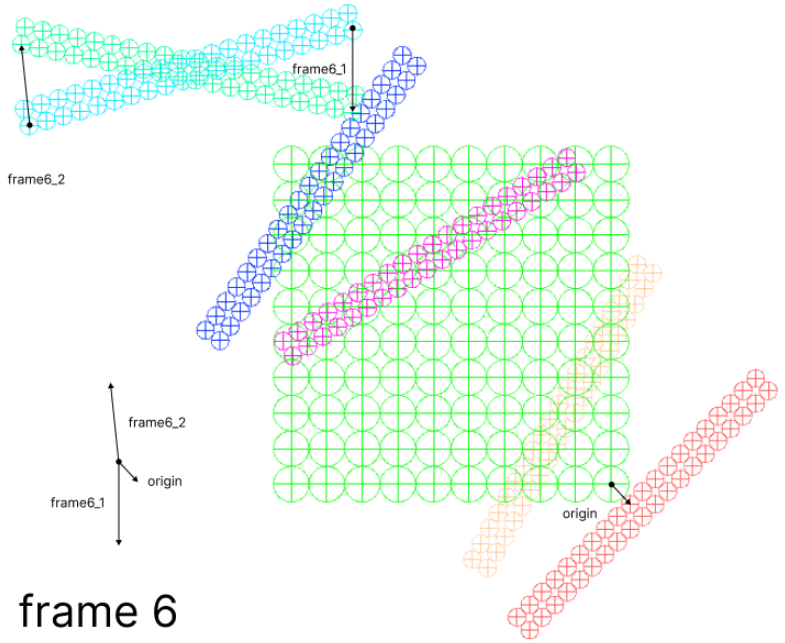
Frame 5



Even though all the `sphere collider` s are out of the `Box Collection` zone

they are still in the `collisionList` and reverse the `origin` , so the force continues to be calculated

Frame 6



In this case , `Cylinder Collection` has been rotated,and some of `sphere collider` s have been gone back,some reversed

- such as,the `sphere collider` which keep vector `frame6_1` ,has the same direciton of `origin`

we could not remove it from the List,because when it rotate next ,we will could not calculate it so , just calculate its distance as usual

the operation `Remove` ,just could be called at the time:

- exit from the collider and go as the same direction with origin

To be solved

- do a research about performance between `trigger` and `collision`

High-Performance

Writing multithreaded code can provide high-performance benefits. These include significant gains in frame rate. Using the Burst compiler with C# jobs gives you improved code generation quality. Implement later~