COMP4920 Senior Design Project II, Spring 2020
Advisor: Mehmet Ufuk Çağlayan

POF: Performance Optimized Fluids

High Level Design
Design Specifications Document

Revision 3.0
1.5.2020

By:

Baran Budak -15070001012

Cihanser Çalışkan -16070001020

İsmail Mekan -15070001048

Revision History

Revision	**Date**	**Explanation**
1.0	12.12.2019	Initial high-level design.
1.1	2.2.2020	The introduction part has been revised. POF system architecture part has been revised and sentences were made more descriptive.
1.2	9.2.2020	Sequence, use case, package diagrams have been updated.
1.3	15.2.2020	Activity diagrams have been updated.
1.4	21.2.2020	High-level design section elaborated.
1.5	26.2.2020	Descriptions have updated for the new diagrams.
1.6	5.3.2020	Table of Contents, list of figures and list of tables updated.
1.7	9.3.2020	Integration testing of the hash system added.
1.8	11.3.2020	Integration testing of marching cubes added.
2.0	12.3.2020	Testing Design section elaborated.
2.1	25.3.2020	Warning added.
2.2	5.4.2020	Table of contents upgraded.
2.3	14.4.2020	Spelling errors fixed.
2.4	20.4.2020	Writing types changed in some parts.
2.5	21.4.2020	Warning page updated.
2.6	24.4.2020	Surface recognition testing section added.
2.7	25.4.2020	Marching cubes testing section discarded.
3.0	1.5.2020	Final version of Design Specifications Document.

Table of Contents

List of Figures

List of Tables

WARNING!

Important Note: POF project has hardware-based requirements. Your GPU must have CUDA 8.0.44 or better version and D3D11 support. If you do not have the required components, POF will not work.
We were using the Yaşar university computer lab in the first semester. Since Yaşar University is closed because of the COVID-19, we cannot access the computer laboratory. Therefore, we cannot make any progress in visualization.
The %75 of the project is finished. Implementation of the Marching Cubes algorithm which is the last step about the visualization part of our project could not be completed (We have a working marching cubes code as a prototype. However, we did not implement to the POF system.). For this reason, we have restated our project requirements and goals which will be clarified detailed in the Final Report and Requirements Specifications Document. In brief, the implementation and testing of the surface recognition system is the new goal of our project and some of the requirements are discarded such as Marching Cubes.

1. Introduction

The purpose of the Performance Optimized Fluid (POF) system is to research and apply existed methods to simulate fluids and looking for a better way to represent it. Numerous algorithms will be implemented and tested during the research and development of this project. The main goal is to research and discuss the advantages and disadvantages of methods. One of the project objectives is to reach a more efficient and better performance fluid simulation system. However, indicated features are not guaranteed to improve performance. The project goal is visualizing the particle-based fluid system differently by benefiting from specific algorithms. The system expected to work more efficiently because of the implementation of the algorithms in the research papers. The project is exceedingly research and development based.

The design is based on The POF system Requirements Specification Document, Revision 3.0 [1]
This design process conforms to the Requirements Specification Document and its diagrams. Diagrams are describing the project to understand mainly operations of the POF system. Imperceptible parts of the POF system can be changed. However, the general operands of the POF system will remain the same as before. If any change occurs during the development of the POF system, this document and diagrams will be changed.

The system architecture and overall high-level structure of the POF system have given in the second section. Detailed design of all system functions and the user interface in terms of are methods of all classes will be explained later in the third section of this document.

2. POF System High Level Design

This section describes the POF system with high-level design. The high-level design section mentions about the POF system architecture and structure in different headings. Besides, the system environment explains the system constraints for the execution of the POF system. In the high-level design, activity diagrams testing design section have elaborated.

2.1. POF System Architecture

The POF system architecture works with NVIDIA Flex which works as an outsource asset. Utilization of NVIDIA flex is mandatory because particle positions and AABB data are necessary to initialize. Initialization of fluid simulation data cannot be randomized. The system has a handler which acts as a communicator between NVIDIA flex and the POF system. Initially, NVIDIA Flex starts the fluid simulation and creates the particles and AABB. The Handler transmits necessary data to relevant classes. We have a hash system section that uses a hash algorithm to reach data quicker which keeps away us from linear search. Surface particle recognizer section determines the particles that are on the surface by calculating colour field quantity. Surface particles and its vertices grouped for a specific radius which is made by group neighbour particle function. Afterwards, grouped neighbour particle data send to the marching cubes algorithm [4]. Marching cubes section determines which vertices must be drawn. Lastly, the Marching cubes section intercommunicates with triangulation section which draws the given vertices.

2.2. POF System Structure

In this section, the POF system structure has described with UML diagrams. Use case and sequence diagrams have drawn in this section.

2.2.1 Use Case Diagram

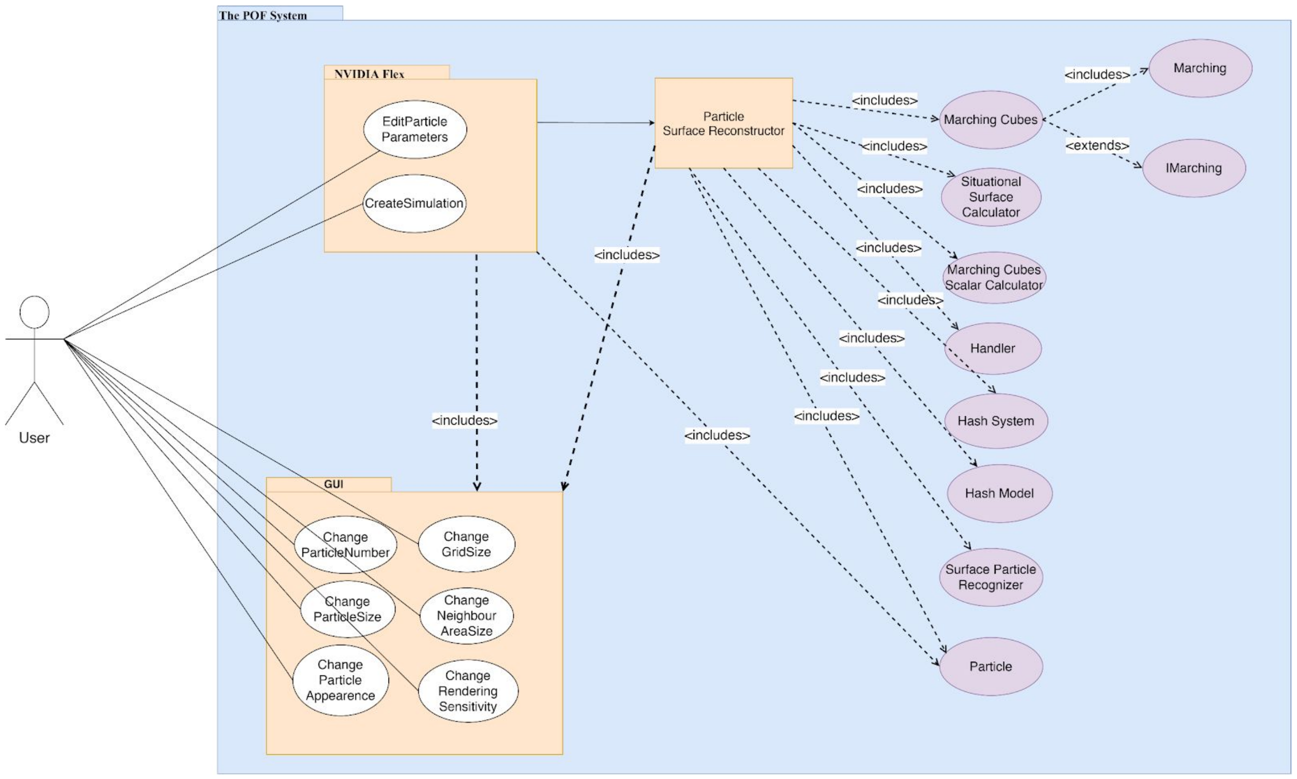


Fig 1: Use case diagram

Title	**Description**
Calculate Scalar Field	Calculates a constant value of a particle in a given range.
Change grid size	Interval of grid size of axis-aligned bounding box can change with this function.
Change neighbour area size	Neighbour particle range of volume can be changed. It affects the visualization of particles and changes the shape.
Change particle appearance	Particle colour, texture and light settings can be changed with this function.
Change particle number	Changes the particle number in the scene.
Change particle size	This function changes the radius of the particle. Excessively disproportionate sizes compared to the scene can result in bugs and anomalies in physics behaviour of the particles
Change rendering sensitivity	User can change the rendering sensitivity with this function. If the sensitivity increase, fluid visualization will be more precise as a result. However, the processing time will increase.
Create Simulation	NVIDIA Flex simulation initialize when this function called.

Edit particle parameters	Particle attributes can edit by a user from GUI. Parameters can be maximum particle number, particle size, friction, adhesion etc.
Group Neighbour Particles	Neighbour particles are grouped by looking at a specific range.
Handler	Handler transmits data between layers and relevant classes. The Handler manages data transmission.
Hash Model	Structure class for the Hash System.
Hash System	This function hashes the particle's position and cell position that particle belongs to.
IMarching	Interface class for Marching and Marching Cubes classes.
Marching	Analyze the cube for vertices and convert edges to triangles from the prewritten table.
Marching Cubes	Applies the Marching cubes algorithm on a single cube.
Marching Cubes Scalar Calculator	It is an application of Zhu et al. [5].
Particle	It represents the particles which used by both NVIDIA Flex and Particle Surface Reconstructor.
Renderer	Visualize fluid by drawing the given polygons.
Situational Surface Calculator	This section calculates and returns the value.
Surface Particle Recognizer	Surface particles marked and processed for the necessary calculations in this function.
User	User can be anyone who has access to the program.

Table 1: Description of the use case diagram

2.2.2 Sequence Diagram

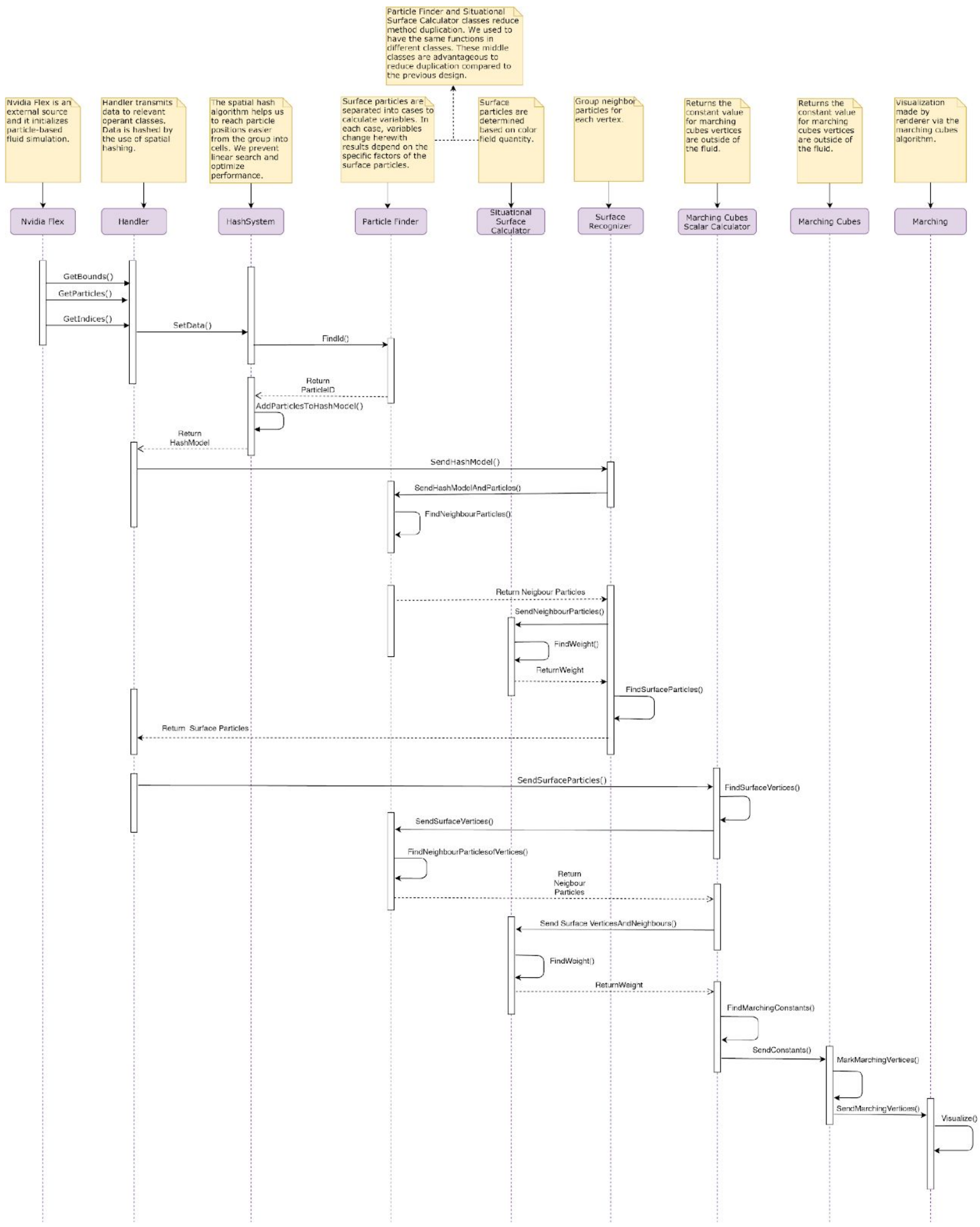

Particle Finder and Situational Surface Calculator classes reduce method duplication. We used to have the same functions in different classes. These middle classes are advantageous to reduce duplication compared to the previous design.
Nvidia Flex is an external source and it initializes particle-based fluid simulation.
Handler transmits data to relevant operant classes. Data is hashed by the use of spatial hashing.
The spatial hash algorithm helps us to reach particle positions easier from the group into cells. We prevent linear search and optimize performance.
Surface particles are separated into cases to calculate variables. In each case, variables change herewith results depend on the specific factors of the surface particles.
Surface particles are determined based on color field quantity.
Group neighbor particles for each vertex.
Returns the constant value for marching cubes vertices are outside of the fluid.
Returns the constant value for marching cubes vertices are outside of the fluid.
Visualization made by renderer via the marching cubes algorithm.
Nvidia Flex
Handler
HashSystem
Particle Finder
Situational Surface Calculator
Surface Recognizer
Marching Cubes Scalar Calculator
Marching Cubes
Marching
GetBounds()
GetParticles()
GetIndices()
SetData()
FindId()
Return ParticleID
AddParticlesToHashModel()
Return HashModel
SendHashModel()
SendHashModelAndParticles()
FindNeighbourParticles()
Return Neigbour Particles
SendNeighbourParticles()
FindWeight()
ReturnWeight
FindSurfaceParticles()
Return Surface Particles
SendSurfaceParticles()
FindSurfaceVertices()
SendSurfaceVertices()
FindNeighbourParticlesofVertices()
Return Neigbour Particles
Send Surface VerticesAndNeighbours()
FindWeight()
ReturnWeight
FindMarchingConstants()
SendConstants()
MarkMarchingVertices()
SendMarchingVertices()
Visualize()

Fig 2: Sequence diagram

Description: Handler manages data transfer between other sections. If any data must transmit to another class, Handler executes this operation. The handler receives the bounds, particles, and indices from NVIDIA flex. Handler sends this information to the hash system. Hash systems ask Particle Finder to find the id of the particle and adds to the hash model and acknowledges the handler. Handler sends the hash model to the surface recognizer. Surface recognizer processes the data and asks to the particle finder to find neighbour particles of a specific particle. Surface recognizer sends the data to the situational surface calculator which is responsible to calculate and return weight. The handler receives the neighbour particles and sends it to the marching cubes scalar calculator which finds the surface vertices. Particle finder receives the surface vertices and finds the vertices of the neighbour particles. Marching cubes scalar calculator receives the data from particle finder and sends to the situational surface calculator and it calculates the weight and returns the value. Marching cubes receives the marching cubes constant value which is calculated and send from the marching cubes scalar calculator. Marching cubes mark the vertices to be drawn. The Marching class starts to visualize by triangles.

2.3. POF System Environment

The POF system environment constraints:

D3D11 capable graphics card
NVIDIA: GeForce Game Ready Driver 372.90 or above.
AMD: Radeon Software Version 16.9.1 or above.
Microsoft Visual Studio 2013 or above.
G++ 4.6.3 or higher
CUDA 8.0.44 or higher
DirectX 11/12 SDK
Windows 7 (64-bit) or higher.
Unity 3D 2017.3 version or higher

Table 2: POF system environment constraints

The main project made on the system:

Operating System	Windows 10 (64-bit)
Processor	Intel Core i7-4700 HQ CPU
Memory	16 GB RAM – DDR3L-1600 MHz
GPU	NVIDIA GeForce GTX850M 4GB DDR3

Table 3: System that POF is operated

This computer system has satisfying performance on only a small number of particles on this project because it can handle a very small number of particles. The optimal fluid simulation computer system mentioned in the final report [3].

3. POF System Detailed Design

This section describes the components of the POF system with activity diagrams. Detailed design has a class diagram for the overall structure. The functionality of these small parts is explained in activity diagrams.

3.1 POF system class diagram

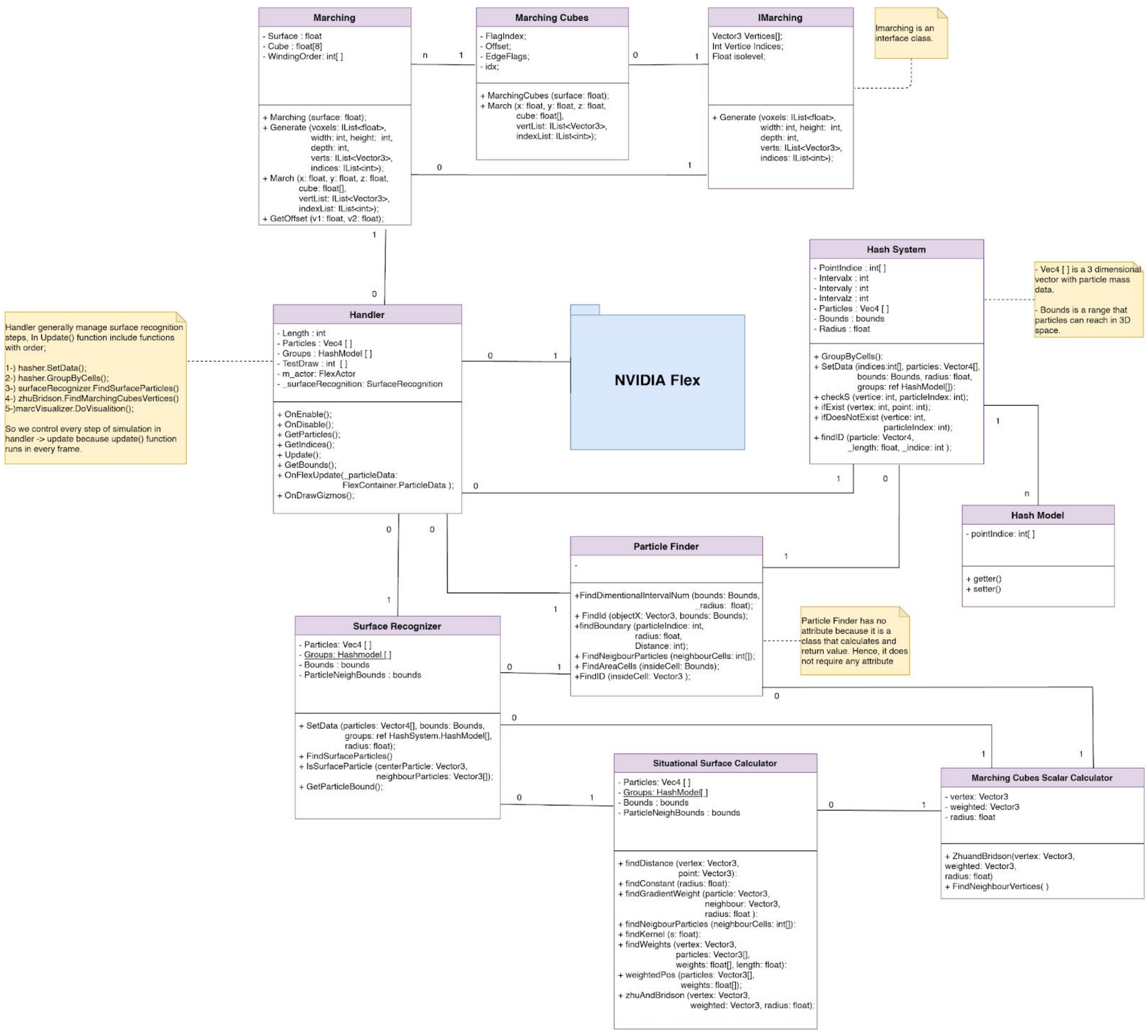

Marching
- Surface : float
- Cube : float[8]
- WindingOrder: int[ ]
+ Marching (surface: float);
+ Generate (voxels: IList<float>, width: int, height: int, depth: int, verts: IList<Vector3>, indices: IList<int>);
+ March (x: float, y: float, z: float, cube: float[], vertList: IList<Vector3>, indexList: IList<int>);
+ GetOffset (v1: float, v2: float);
Marching Cubes
- FlagIndex;
- Offset;
- EdgeFlags;
- idx;
+ MarchingCubes (surface: float);
+ March (x: float, y: float, z: float, cube: float[], vertList: IList<Vector3>, indexList: IList<int>);
IMarching
Vector3 Vertices[];
Int Vertice Indices;
Float isolevel;
+ Generate (voxels: IList<float>, width: int, height: int, depth: int, verts: IList<Vector3>, indices: IList<int>);
Imarching is an interface class.
Hash System
- PointIndice : int[ ]
- Intervalx : int
- Intervaly : int
- Intervalz : int
- Particles : Vec4 [ ]
- Bounds : bounds
- Radius : float
+ GroupByCells();
+ SetData (indices:int[], particles: Vector4[], bounds: Bounds, radius: float, groups: ref HashModel[]);
+ checkS (vertice: int, particleIndex: int);
+ ifExist (vertex: int, point: int);
+ ifDoesNotExist (vertice: int, particleIndex: int);
+ findID (particle: Vector4, _length: float, _indice: int);
- Vec4 [ ] is a 3 dimensional vector with particle mass data.
- Bounds is a range that particles can reach in 3D space.
Handler generally manage surface recognition steps, In Update() function include functions with order;
1-) hasher.SetData();
2-) hasher.GroupByCells();
3-) surfaceRecognizer.FindSurfaceParticles()
4-) zhuBridson.FindMarchingCubesVertices()
5-)marcVisualizer.DoVisualition();
So we control every step of simulation in handler -> update because update() function runs in every frame.
Handler
- Length : int
- Particles : Vec4 [ ]
- Groups : HashModel [ ]
- TestDraw : int [ ]
- m_actor: FlexActor
- _surfaceRecognition: SurfaceRecognition
+ OnEnable();
+ OnDisable();
+ GetParticles();
+ GetIndices();
+ Update();
+ GetBounds();
+ OnFlexUpdate(_particleData: FlexContainer.ParticleData);
+ OnDrawGizmos();
NVIDIA Flex
Hash Model
- pointIndice: int[ ]
+ getter()
+ setter()
Particle Finder
-
+FindDimentionalIntervalNum (bounds: Bounds, _radius: float);
+ FindId (objectX: Vector3, bounds: Bounds);
+findBoundary (particleIndice: int, radius: float, Distance: int);
+ FindNeigbourParticles (neighbourCells: int[]);
+ FindAreaCells (insideCell: Bounds);
+FindID (insideCell: Vector3);
Particle Finder has no attribute because it is a class that calculates and return value. Hence, it does not require any attribute
Surface Recognizer
- Particles: Vec4 [ ]
- Groups: Hashmodel [ ]
- Bounds : bounds
- ParticleNeighBounds : bounds
+ SetData (particles: Vector4[], bounds: Bounds, groups: ref HashSystem.HashModel[], radius: float);
+ FindSurfaceParticles()
+ IsSurfaceParticle (centerParticle: Vector3, neighbourParticles: Vector3[]);
+ GetParticleBound();
Situational Surface Calculator
- Particles: Vec4 [ ]
- Groups: HashModel[ ]
- Bounds : bounds
- ParticleNeighBounds : bounds
+ findDistance (vertex: Vector3, point: Vector3):
+ findConstant (radius: float):
+ findGradientWeight (particle: Vector3, neighbour: Vector3, radius: float):
+ findNeigbourParticles (neighbourCells: int[]):
+ findKernel (s: float):
+ findWeights (vertex: Vector3, particles: Vector3[], weights: float[], length: float):
+ weightedPos (particles: Vector3[], weights: float[]);
+ zhuAndBridson (vertex: Vector3, weighted: Vector3, radius: float):
Marching Cubes Scalar Calculator
- vertex: Vector3
- weighted: Vector3
- radius: float
+ ZhuandBridson(vertex: Vector3, weighted: Vector3, radius: float)
+ FindNeighbourVertices( )

Fig 3: Class diagram

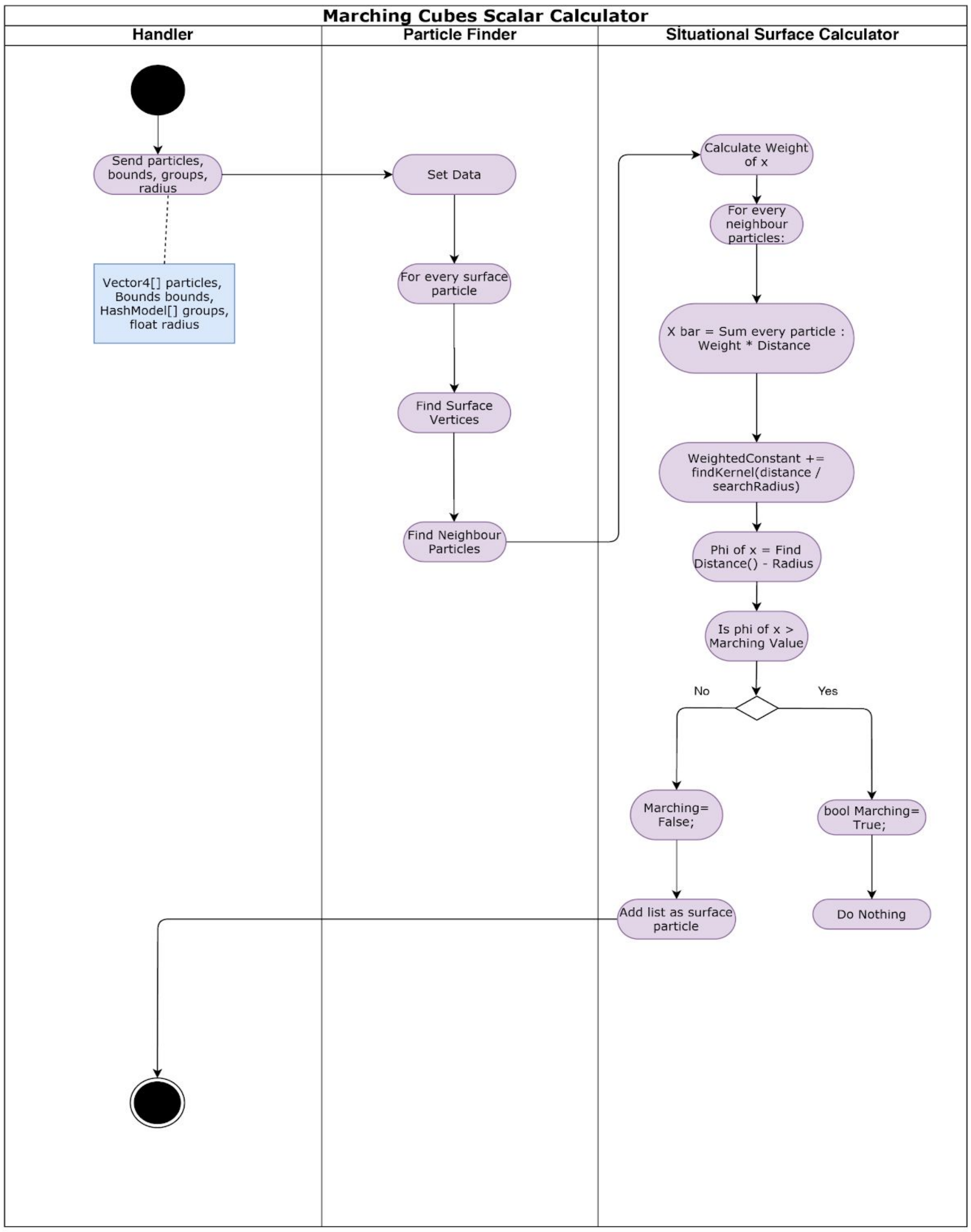
Description: NVIDIA flex is an external package which it is accessed by Handler. Handler makes the communication and organizes the data transmission as you can see from the relations. Consecutive actions are described in the sequence diagram. Visualization part is consisting of Marching, Marching cubes and IMarching classes. IMarching is an interface class for the other two class. Marching cubes analyse the cube situation according to the Marching cubes algorithm [4]. Marching class draws the triangles and applies the algorithm for a cube. The situational surface calculator and particle finder is a class that made calculations and returns values for other classes.

Marching cubes scalar calculator is an application of Zhu et al. [5]. It calculates and returns a scalar value for being used in the marching cubes algorithm. Surface recognizer receives the calculated valued from other classes and decides the surface particles.

3.2 Subclasses of the POF system

This section includes subclasses of activity diagrams.

3.2.1 Activity Diagram of Marching Cubes Scalar Calculator



Marching Cubes Scalar Calculator
Handler
Particle Finder
Situational Surface Calculator
Send particles, bounds, groups, radius
Vector4[] particles, Bounds bounds, HashModel[] groups, float radius
Set Data
For every surface particle
Find Surface Vertices
Find Neighbour Particles
Calculate Weight of x
For every neighbour particles:
X bar = Sum every particle : Weight * Distance
WeightedConstant += findKernel(distance / searchRadius)
Phi of x = Find Distance() - Radius
Is phi of x > Marching Value
No
Yes
Marching= False;
bool Marching= True;
Add list as surface particle
Do Nothing

Fig 4: Activity diagram of Marching Cubes Scalar Calculator

Description: The handler sends the particles, bounds, groups, and radius. Particle finder class receives and set data. Particle finder class is responsible for find neighbours of the particle in a specific radius. Particle finder marks for the surface vertices in every surface particle. Neighbour particles are sent to the situational surface calculator. Weight is calculated and the vertices that will be drawn are marked in according to the algorithm [5]. Surface Particles are added to the list and return to the Handler.

3.2.2 Activity Diagram of Surface Recognizer

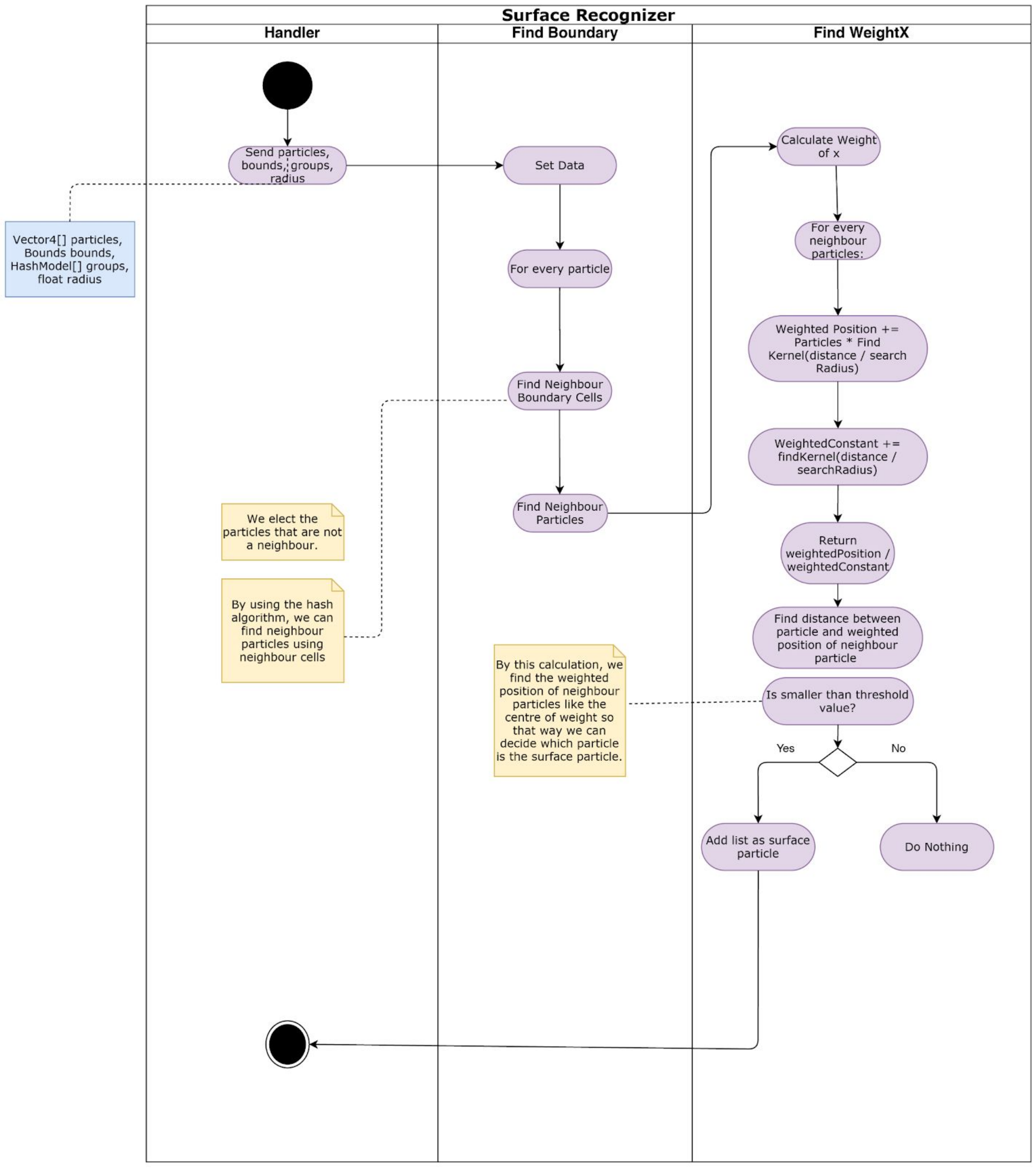

Surface Recognizer
Handler
Find Boundary
Find WeightX
Send particles, bounds, groups, radius
Vector4[] particles, Bounds bounds, HashModel[] groups, float radius
Set Data
For every particle
Find Neighbour Boundary Cells
Find Neighbour Particles
We elect the particles that are not a neighbour.
By using the hash algorithm, we can find neighbour particles using neighbour cells
Calculate Weight of x
For every neighbour particles:
Weighted Position += Particles * Find Kernel(distance / search Radius)
WeightedConstant += findKernel(distance / searchRadius)
Return weightedPosition / weightedConstant
Find distance between particle and weighted position of neighbour particle
Is smaller than threshold value?
By this calculation, we find the weighted position of neighbour particles like the centre of weight so that way we can decide which particle is the surface particle.
Yes
No
Add list as surface particle
Do Nothing

Fig 5: Activity diagram of Surface Recognizer

Description: Handler sends data of the particle, bounds, groups, and radius to find boundary function. Find boundary searches neighbour boundary cells for every particle and then find neighbour particles process starts. Find weightx function is calculates the weight and return a value. Find weightx section calculates the weight and if weight is smaller than specific constant value ‘q’ [8], Under favour of this calculation, we can decide the surface particles. If weight is bigger than a constant value, the particle is not a surface particle. Marked particles are sent to the Handler.

3.2.3 Activity Diagram of Marching Cubes

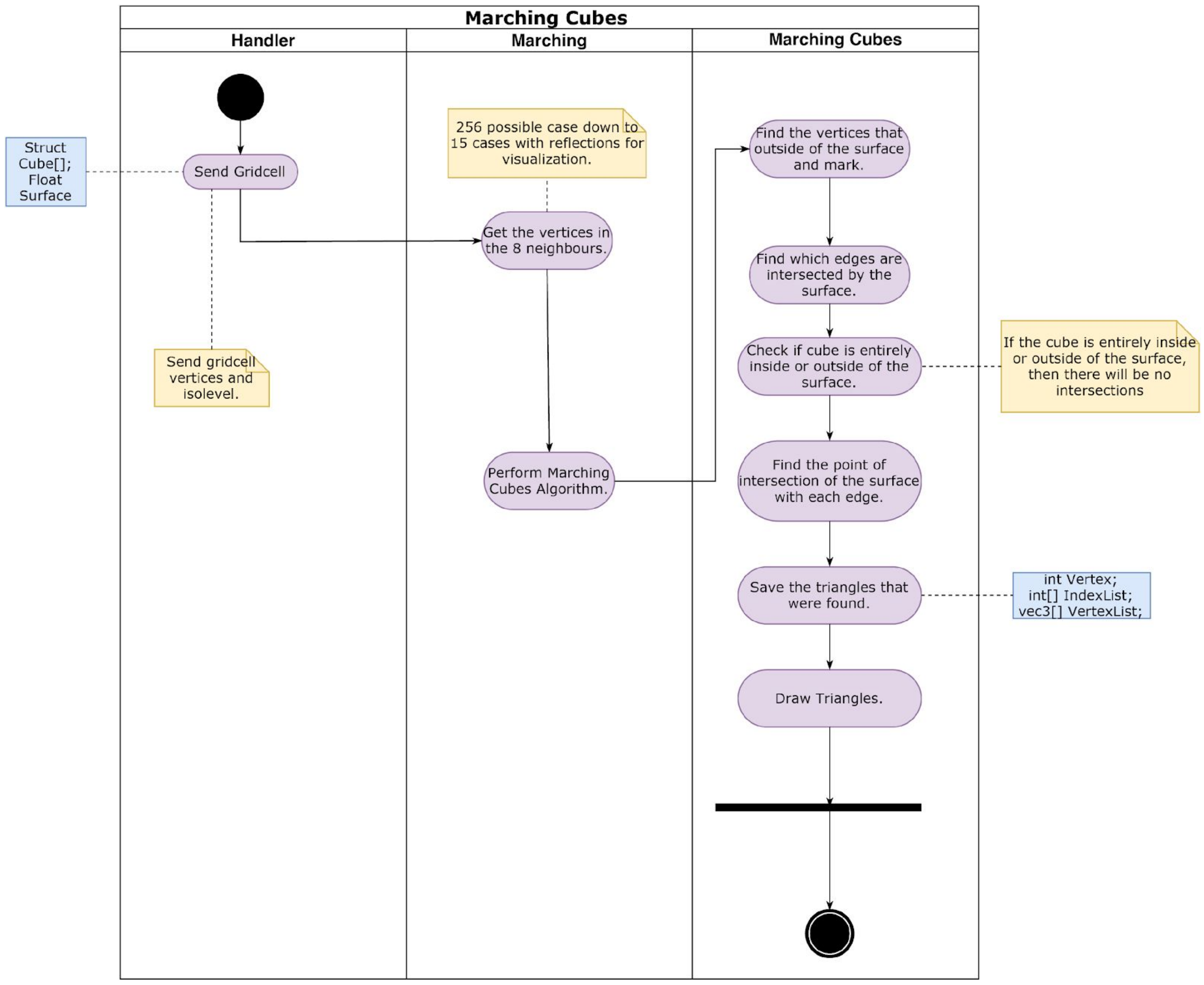
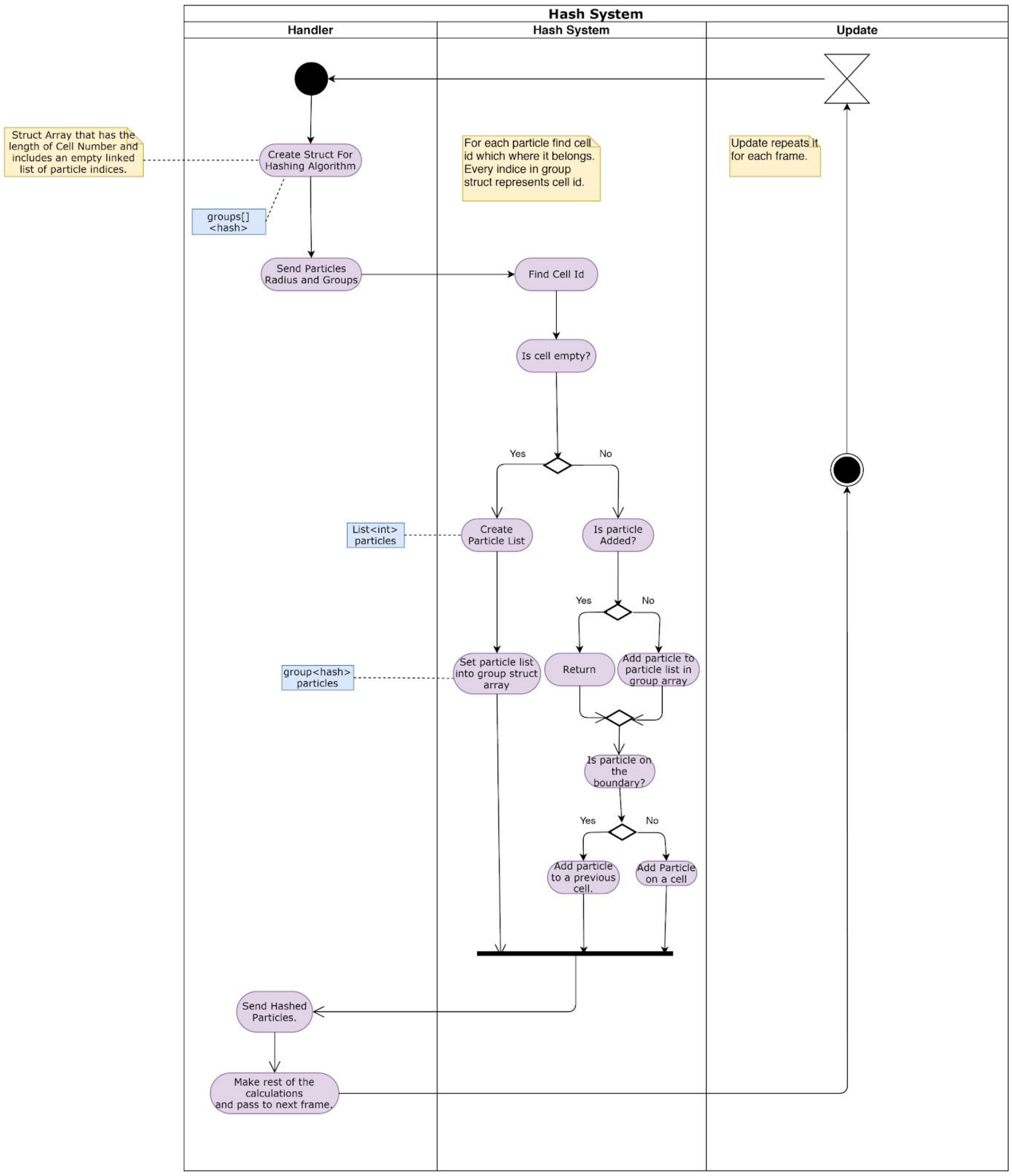

Marching Cubes
Handler
Marching
Marching Cubes
Struct Cube[]; Float Surface
Send Gridcell
Send gridcell vertices and isolevel.
256 possible case down to 15 cases with reflections for visualization.
Get the vertices in the 8 neighbours.
Perform Marching Cubes Algorithm.
Find the vertices that outside of the surface and mark.
Find which edges are intersected by the surface.
Check if cube is entirely inside or outside of the surface.
If the cube is entirely inside or outside of the surface, then there will be no intersections
Find the point of intersection of the surface with each edge.
Save the triangles that were found.
int Vertex; int[] IndexList; vec3[] VertexList;
Draw Triangles.

Fig 6: Activity diagram of Marching Cubes

Description: Handler sends a grid cell (Cell is a structure consisting of vertices and iso-level) to polygonise function [4]. Marching analyses the cube situation by looking cases in [4]. There are 256 cases, but it is downgraded to 15 configurations predefined. After that, Marching Cubes finds the surface vertices and marks it. Then, edges that intersected by the surface is found. If the surface does not intersect with the cube, the vertex will not be drawn. Triangles are created by gathering three edges. Finally, the drawing process starts, and triangles are visualized.

3.2.4 Activity Diagram of Hash System



Hash System
Handler
Hash System
Update
Struct Array that has the length of Cell Number and includes an empty linked list of particle indices.
Create Struct For Hashing Algorithm
groups[] <hash>
Send Particles Radius and Groups
For each particle find cell id which where it belongs. Every indice in group struct represents cell id.
Update repeats it for each frame.
Find Cell Id
Is cell empty?
Yes
No
List<int> particles
Create Particle List
Is particle Added?
Yes
No
group<hash> particles
Set particle list into group struct array
Return
Add particle to particle list in group array
Is particle on the boundary?
Yes
No
Add particle to a previous cell.
Add Particle on a cell
Send Hashed Particles.
Make rest of the calculations and pass to next frame.

Fig 7: Activity diagram of Hash System

Description: Handler sends particles, radius, and groups to the hash system. Hash system finds the cell id and checks if it is empty. If empty, particle list is created, and the particle list has added. If not empty, checks if particle added. If the particle is not added, the particle is added to the particle list array. Final check occurs for the particles that are on the cell boundary. If the particle is on the boundary, the particle is added to the previous cell. If not, particle remains in the same cell. Hashed cells are sent to the handler and process ends.

4. Testing Design

Considering our project is predominantly research and development based, scientific papers and algorithm methods research takes a lot of time. Because of these reasons testing and design changes deferred to the later stages of our project. In this section, we have made the integration test design and prove that our code generally works.

4.1 Hash System Testing

The first difficulty of the POF system was to solve the hash system to reach particles faster and increase performance. However, developing on Unity platform has its difficulties. Debugging was a problem in unity game mode. When a bug occurs, unity stops running and close immediately. You cannot test components even for the little bit off the scale values because unity program freezes and terminate the operation. Therefore, we have tried to colour in scene mode. We have made our test by paint the selected particle to blue colour on gizmos. Neighbour particles painted to red and cell that particle in is drawn as a red wire cube. The other particles are white means neutral and offset the colour of the particles. The outmost yellow wire cube is AABB. Besides, the cell system is an imaginary mathematical structure that does not exist in the POF system. However, it can be computed with position data of a particle by a function immediately.

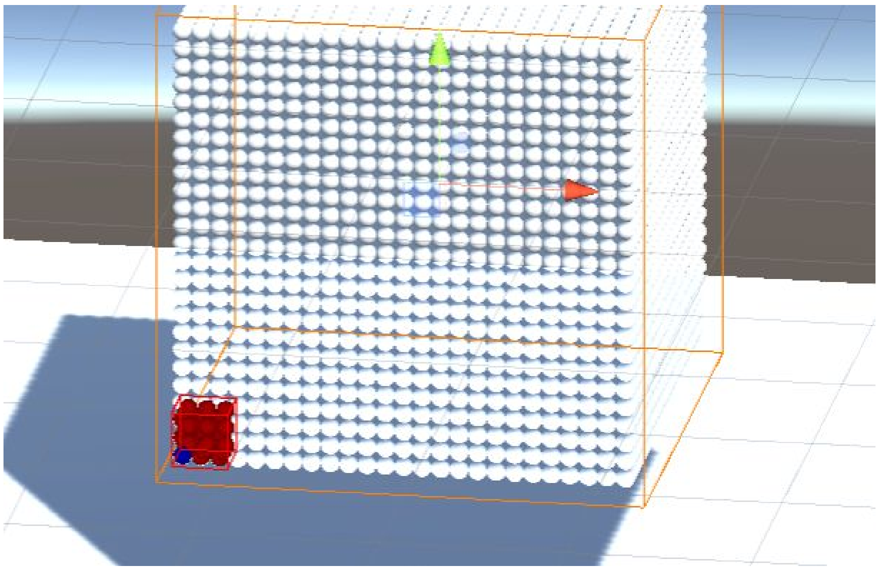


Fig 8: Particle and its neighbours in the cell.

In game mode, the particles start to move, and we must track the selected particle and its neighbours including the cell that particle belongs to. From the figure below, you can see that the objective is achieved.

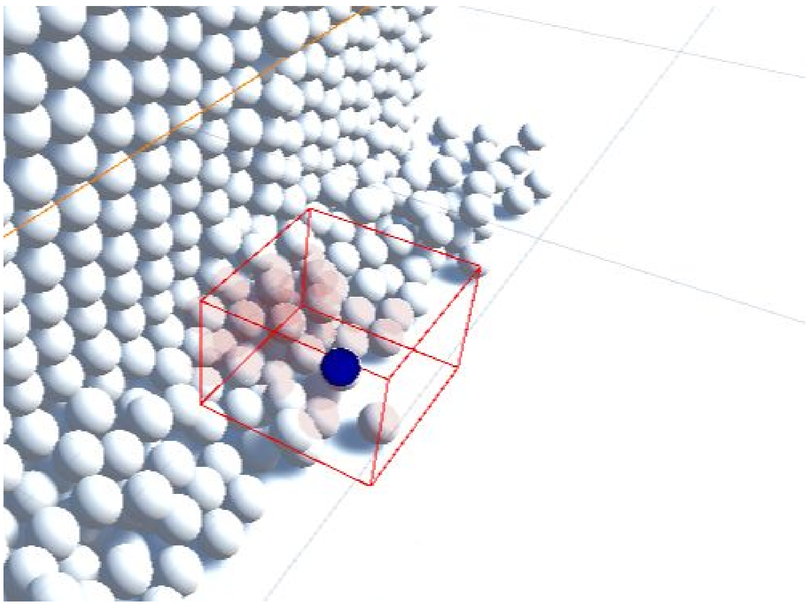


Fig 9: Tracking particle in a cell.

Firstly, we managed to track the particle that we want by changing the particle id which proves that our hash system works correctly. We have made the proof by calculating manually particle order in the AABB afterwards. Tracking the cell boundaries and neighbour particles is as important as tracking the particle itself. Images are taken while unity is playing on game mode.

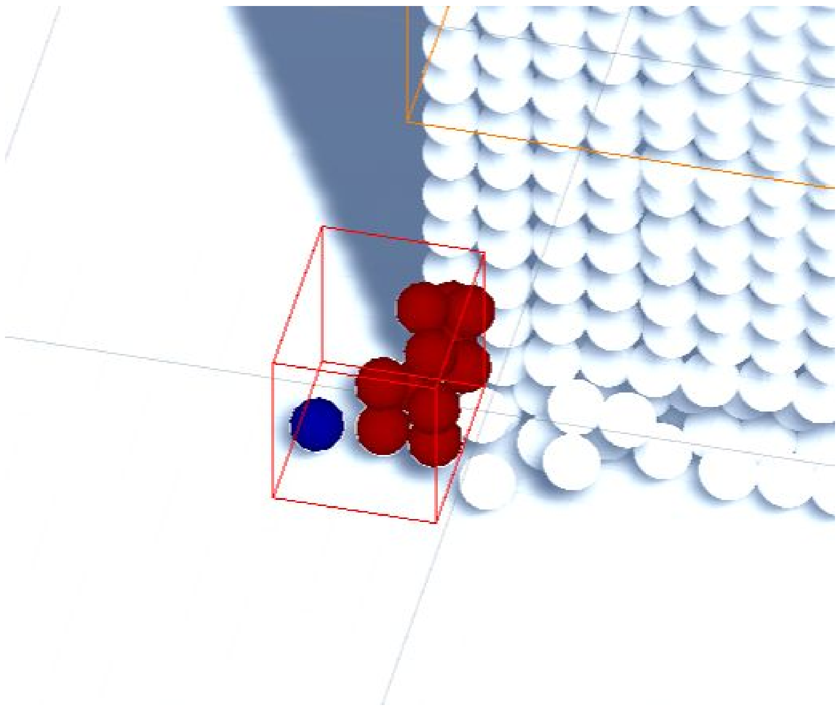


Fig 10: Tracking particle with neighbours in a cell.

4.2 Surface Recognition Testing

We implemented surface recognition system however testing stage was difficult because of the deficiency of the simulation computer. Even though we had small computational power, we could have managed to finish surface recognition testing. Surface recognition testing has a longer testing phase because of particle's three-dimensional complexity in the scene varies to the crashes, speeds, or other various factors.

We applied the same methods when we tested hash system. Surface particles are painted to blue and in some scenes inner particles are painted to blue colour.

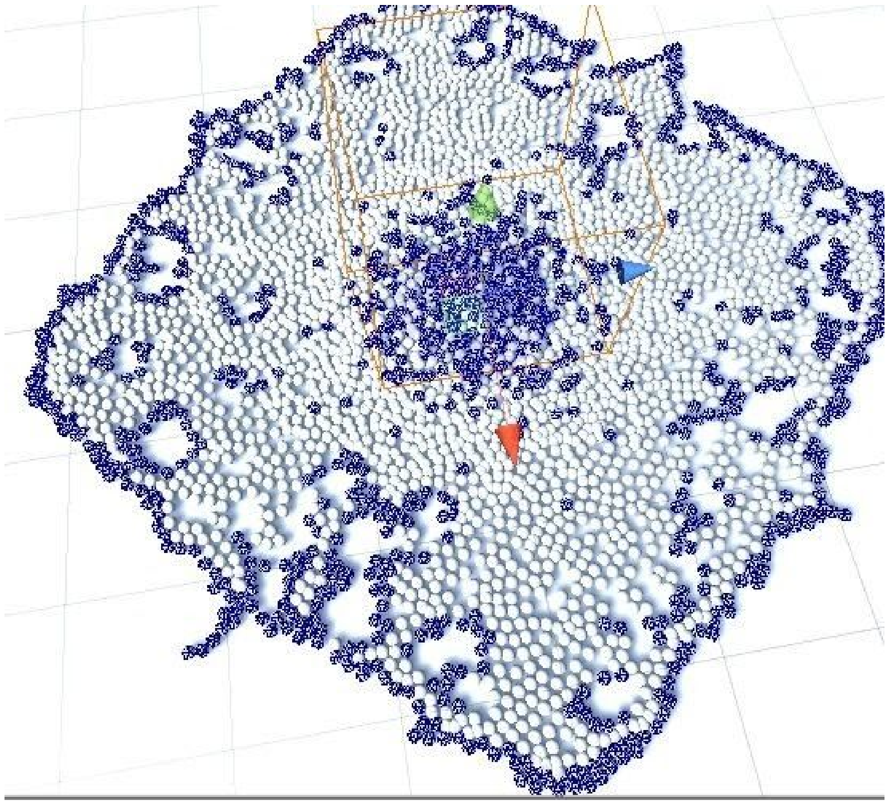


Fig 11: Surface particles are painted to blue colour.

As you can see in the figure above, we can track the surface particles. Surface recognition system determines if a particle is a surface particle and mark with blue colour.

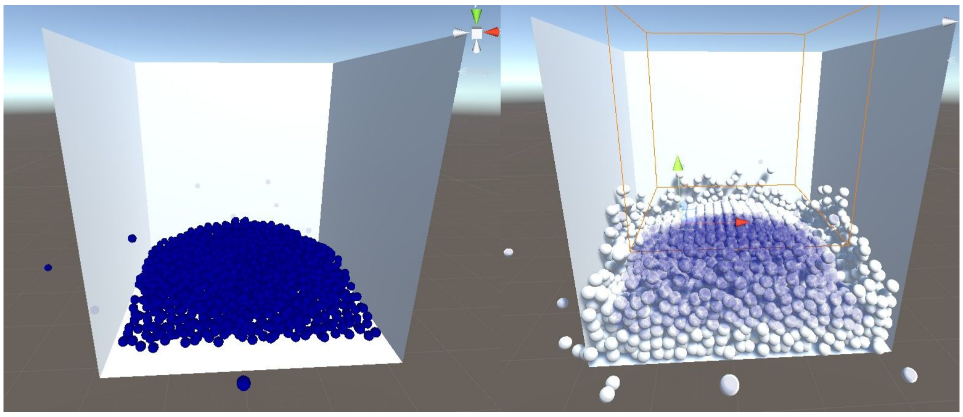


Fig 12: Inner particles are painted to blue and surface particles are painted to white.

References

1. Final Report revision 1.0

2. Requirement Specification Document revision 3.0 (RSD 3.0)

3. Use case and sequence diagrams in RSD 3.0

4. **[WH87]** William E. Lorensen and Harvey E. Cline. (1987). Marching cubes: A high resolution 3D surface construction algorithm. ACM SIGGRAPH Computer Graphics. 21, 163-169.
5. **[ZB05]** Zhu, Y., & Bridson, R. (2005). Animating sand as a fluid. (New York, NY, USA, 2005) *ACM Trans. Graph., 24*, 965-972.

6. **[BP94]** Paul Bourke 1994, Marching Cubes, viewed 4 March 2020, <http://paulbourke.net/geometry/polygonise/>

7. **[SSP07]** Solenthaler, Barbara & Schläfli, Jürg & Pajarola, Renato. (2007). A unified particle model for fluid-solid interactions. Journal of Visualization and Computer Animation. 18. 69-82. 10.1002/cav.162.

8. **[AIA12]** Akinci, Gizem & Ihmsen, Markus & Akinci, Nadir & Teschner, Matthias. (2012). Parallel Surface Reconstruction for Particle-Based Fluids. CGF. 31. 1797-1809. 10.1111/j.1467-8659.2012.02096.x.

9. **[TH03]** Teschner, M., Heidelberger, B., Müller, M., Pomerantes, D., and Gross, M.H. (2003). Optimized Spatial Hashing for Collision Detection of Deformable Objects. VMV.