YAŞAR UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING

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

POF: Performance Optimized Fluid System

Final Report

(Bachelor of Science Thesis)

By:

Baran Budak -15070001012

Cihanser Çalışkan -16070001020

İsmail Mekan -15070001048

Revision History

Revision	Date	Explanation
1.0	8.12.2019	Initial Final Report.
1.1	10.2.2020	- Abstract and Özet part renewed. - Revision history added.
1.2	17.2.2020	Warning page added.
1.3	2.3.2020	Introduction section writings updated.
1.4	7.3.2020	2. semester Gantt chart updated.
1.5	15.2.2020	Requirements section added.
1.6	5.3.2020	Class diagram and description is updated.
1.7	18.3.2020	Code parts and explanations updated in implementation section.
1.8	30.3.2020	Performance test and results added.
1.9	14.4.2020	- Cost analysis of second semester added. - Plagiarism statement updated.
2.0	27.4.2020	- Appendices of product manual added. - Table of contents updated. - List of figures updated. - List of tables updated.

PLAGIARISM STATEMENT

This report was written by the group members and in our own words, except for quotations from published and unpublished sources which are clearly indicated and acknowledged as such. We are conscious that the incorporation of material from other works or a paraphrase of such material without acknowledgement will be treated as plagiarism according to the University Regulations. The source of any picture, graph, map, or other illustration is also indicated, as is the source, published or unpublished, of any material not resulting from our own experimentation, observation or specimen collecting.

Project Group Members:

Name, Last name	**Student Number**	**Signature**	**Date**
Baran Budak	15070001012		24.05.20
Cihanser Çalışkan	16070001020		24.05.2020
İsmail Mekan	15070001048		24.05.20

Project Supervisors:

Name, Last name	**Department**	**Signature**	**Date**
Mehmet Ufuk Çağlayan	Computer Engineering		

ACKNOWLEDGEMENTS

We would like to thank Gizem Kayar for discussions and suggestions in the development of the project. This project is supported by Yasar University Computer Engineering Department.

KEYWORDS

Term	Description
Cell	Axis aligned bounding box is divided into small identical cubes.
Colour field quantity	It is a function that calculates how each particle is affected by all the other particles.
Gradient	The directional derivative of a scalar field gives a vector field directed towards where the increment is most, and its magnitude is equal to the greatest value of the change.
Grid	Series of vertical and horizontal lines that are used to subdivide AABB vertically and horizontally into cells in three-dimensional space.
Iso-surface	An isosurface is a 3D surface representation of points with equal values in a 3D data distribution which is the 3D equivalent of a contour line.
Marching Cubes	Marching cubes is a computer graphics algorithm, published in 1987 for extracting a polygonal mesh of an isosurface from a three-dimensional discrete scalar field.
NVIDIA Flex	NVIDIA Flex is a particle-based simulation technique for real-time visual effects created by NVIDIA company.
Polygonal Mesh	A polygon mesh is the collection of vertices, edges, and faces that make up a 3D object.
Unity 3D	Unity is a cross-platform game engine developed by Unity Technologies. Unity is used for developing video games and simulations for consoles and mobile devices.
Spatial Hashing	Spatial hashing is a technique in which objects in a 2D or 3D domain space are projected into a 1D hash table allowing for very fast queries on objects in the domain space.

Table 1: Keywords

ABSTRACT

POF system aims at providing more optimized and faster surface identification and visualization on particle-based fluid simulations.

This project is research-based. It is possible for the small parts of the structures can change during the project. We research possible solutions for the problem and examined a lot of research papers for the algorithms. We discussed the pros and cons of various methods and decided to use specific algorithms for the mentioned reasons.
The POF system divided into a structure that has various algorithms. A control panel (controller or handler) administers these algorithms that placed as substructures in the POF system.
We can list these substructures as Hash System, Surface Particle Recognizer and Visualization parts. Hash System is an imaginary structure and serves to search for data easily. Surface particle recognizer distinguishes the surface particles. Visualization part draws the surface particles vertices which is an implementation of the Marching Cubes algorithm [4].

ÖZET

POF sistemi, partikül bazlı sıvı simülasyonlarında daha optimize ve daha hızlı yüzey tanımlama ve görüntüleme sağlamayı amaçlamaktadır.

Bu proje araştırmaya dayalıdır. Proje sırasında yapıların küçük bölümlerinin değişmesi mümkündür. Sorun için olası çözümleri araştırdık ve algoritmalar için birçok araştırma makalesini inceledik. Çeşitli yöntemlerin artılarını ve eksilerini tartıştık ve belirtilen nedenlerden dolayı spesifik algoritmalar kullanmaya karar verdik.
POF sistemi, çeşitli algoritmalara sahip bir yapıya ayrılmıştır. Sistemin arayüzü olan bir kontrol paneli, (denetleyici veya işleyici) POF sistemindeki alt bileşenleri yönetir.
Bu alt yapıları karma sistemi (hash system), yüzey partikül tanıyıcı ve görselleştirme parçaları olarak listeleyebiliriz. Karma (hash) sistemi hayali bir yapıdır ve verileri kolayca aramaya yarar. Yüzey partikül tanıyıcı yüzey partiküllerini ayırt etmemize yarar. Görselleştirme bölümü, Yürüyen Küpler algoritmasının [4] bir uygulaması olan yüzey parçacıklarının köşelerini çizmemizi sağlar.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

AABB	Axis Aligned Bounding Box. Bounding volume for a set of objects is a closed volume that completely contains the union of the objects in the set.
API	Application Programming Interface.
CPU	Central Processing Unit.
CUDA	Compute Unified Device Architecture. CUDA is a parallel computing platform and application programming interface (API) model created by NVIDIA.
D3D11	Direct3D 11. Is to create 3-D graphics for games and scientific and desktop applications.
GPU	Graphic Processing Unit.
MVC	Stands for Model View Controller. MVC is an application design model comprised of three interconnected parts (Model, View, Controller).
NVIDIA	NVIDIA corporation is a company designs GPUs.
OPENGL	Open Graphics Library is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics.
POF	Performance Optimized Fluid.
SSF	Screen Space Fluids Pro.

Table 3: List of acronyms/abbreviations

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

This section explained in three main titles: problem description, project goal and project output.

1.1. Description of the Problem

The main problem of the particle-based fluid simulation system is excessive numbers of the particles. A particle is a rigid body sphere.
There are millions of particles in a small number of liquids. Simulation control particles by physics-based calculations to obtain fluid behaviours. Simulation having difficulties in calculations dependent on a surplus of particles and time and memory complexity increases indirectly. Visualizing millions of particles on a scene are a tedious job.

1.2. Project Goal

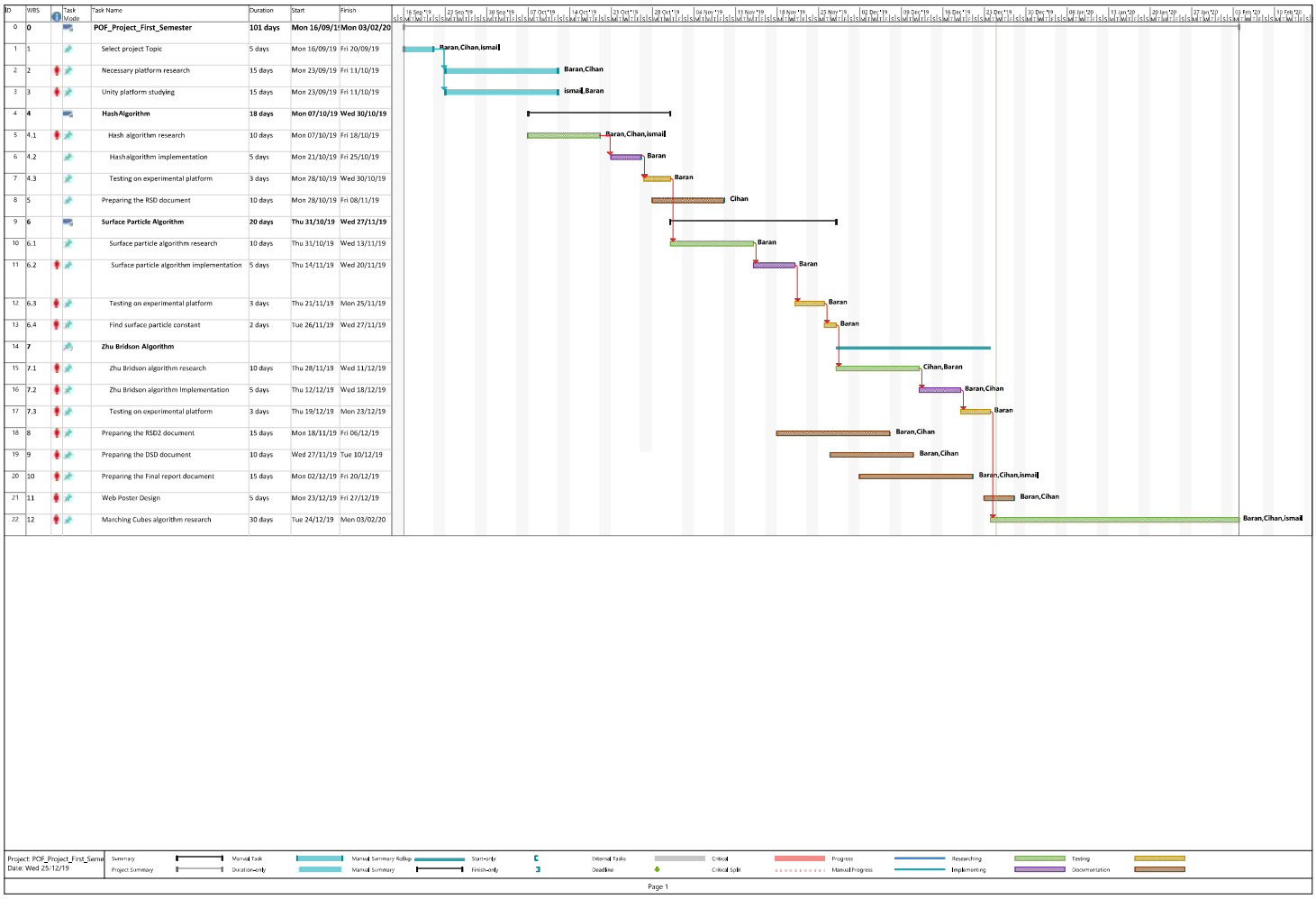
During the POF project, we were researching ways of enhancing the performance and efficiency of particle-based fluid simulation. Creating a suitable and stable platform for executing a particle-based fluid simulation is one of our primary project goals. This platform is Unity for our project. Constructing a more user-friendly platform for testing and comparing various algorithms for scientific research is another project goal.
We aim to achieve these goals by reaching particles faster by constructing a spatial hash sub-system. To detect surface particles, we must implement surface recognition sub-system.
Our project has no predetermined method because POF is a research and development based. We can research and implement new methods during the project.

1.3. Project Output

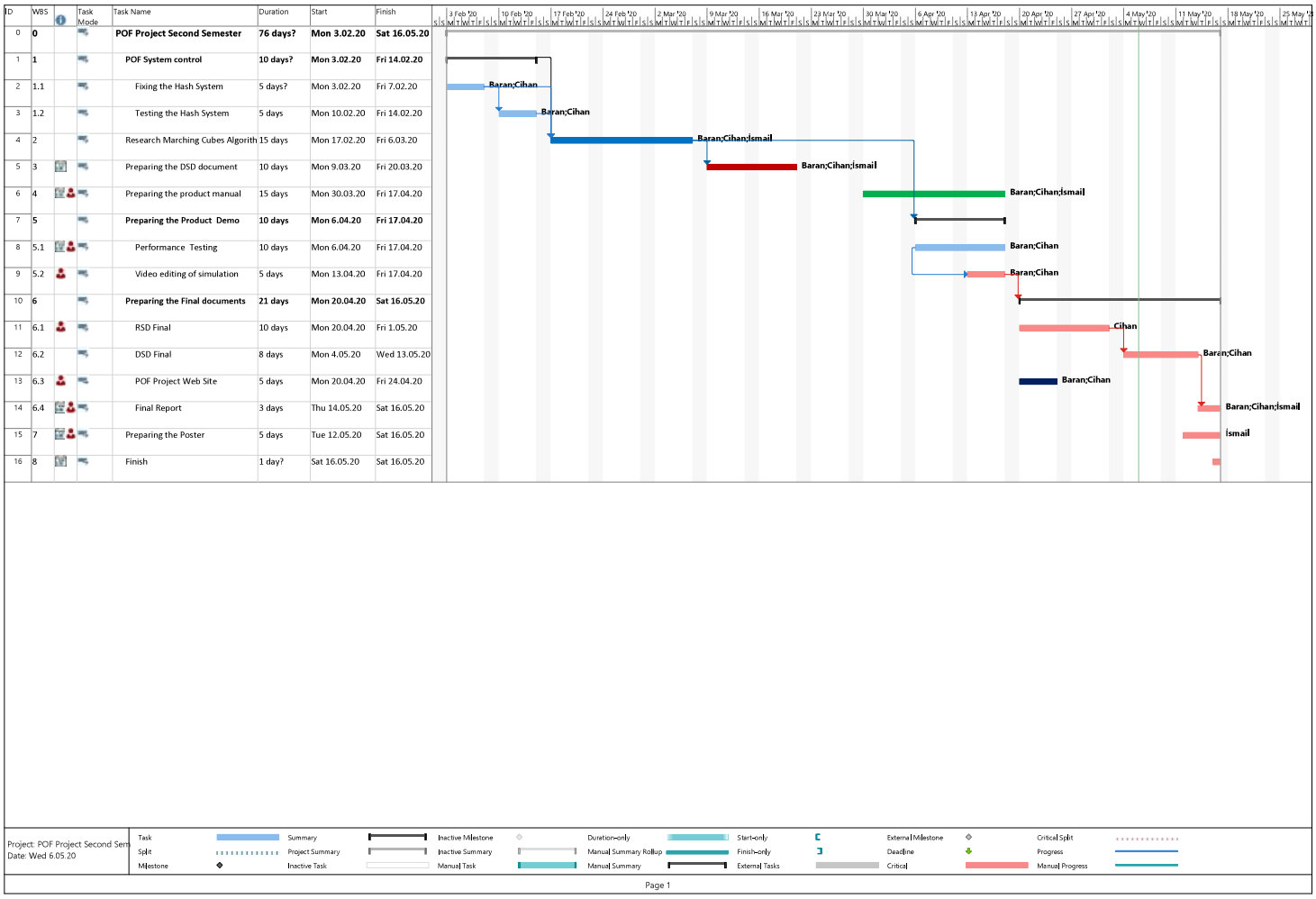
- Better performance.
- Better memory efficiency.
- Fluid-like appearance and behaviour.
- Different algorithms testing for performance and efficiency.
- Surface particles detection.

1.4. Project Activities and Schedule

1.4.1 First Semester Gantt Chart



1.4.2 Second Semester GanntChart



2. REQUIREMENTS

We discarded the last step of our project due to our access has inhibited to the computer laboratory in the university because of the COVID-19. We discussed with our project advisor and agreed on discarding the implementation of Marching Cubes algorithm requirement.
The final requirements of the project are provided in Appendix A: Requirements Specifications Document, revision 3.0 (RSD 3.0). All requirements are on the table below.

Requirement Type	**Requirement Name**
Functional Requirement	Retrieve Particle Data
Functional Requirement	Divide into Cells
Functional Requirement	Surface Recognition
Functional Requirement	Marching Cubes
Non-Functional Requirement	Efficiency
Non-Functional Requirement	Performance
Non-Functional Requirement	Usability
Non-Functional Requirement	Testability

Table 3: Requirements list.

3. DESIGN

This section describes about design of the POF system. High level and detailed designs are explained. Restrictions and conditions mentioned in this section.

3.1. High Level Design

High level design is explained in detail and a sequence diagram is included in DSD 2.0.

3.1.1. Class Diagram

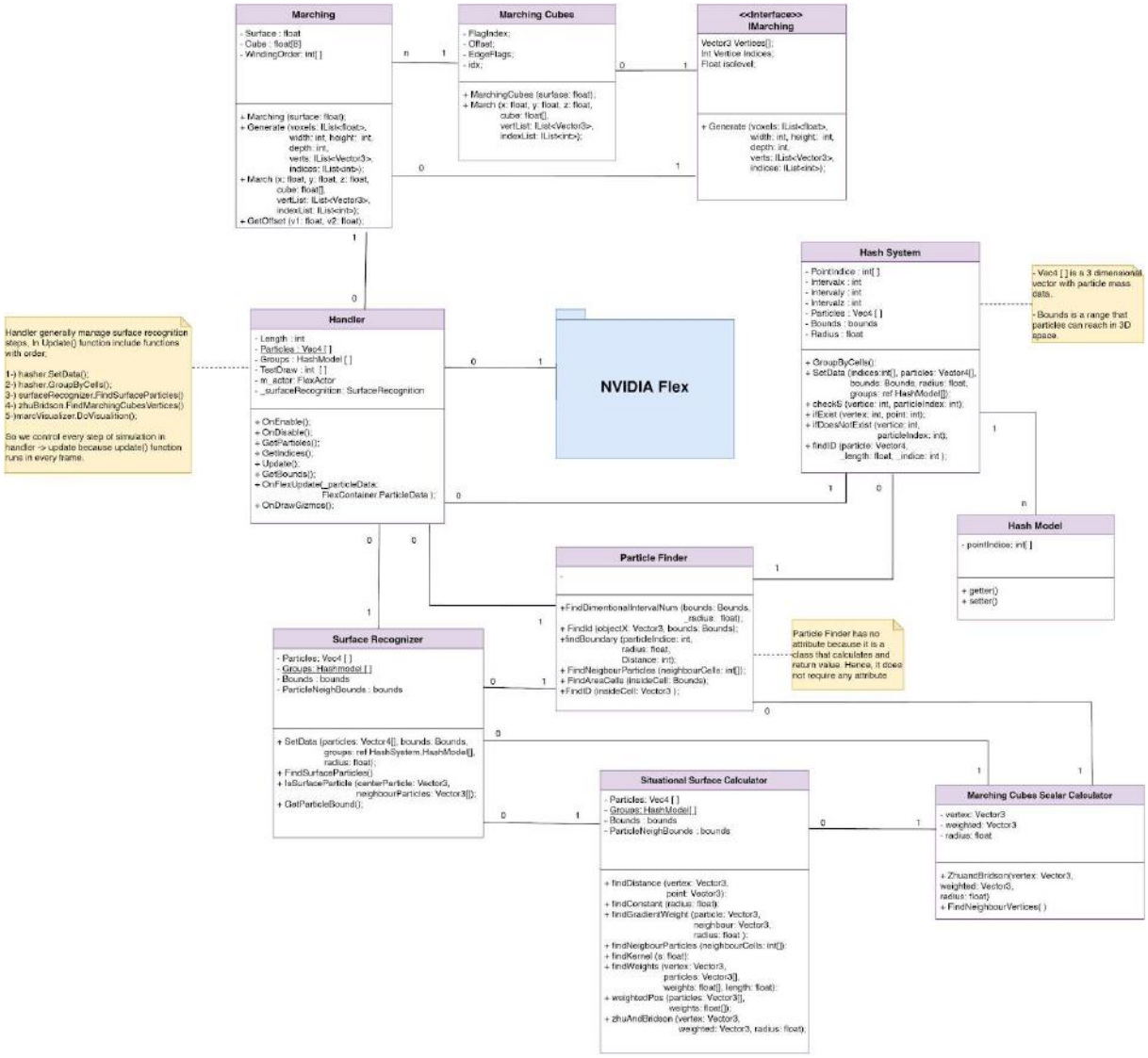


Fig 1: Class Diagram

The class diagram above explains classes and their attributes along with relations. NVIDIA flex is obligatory and an external source asset. Handler class is the main controller of the system. Handler communicates with classes transmit data to relevant classes.

Hash System class is responsible for the classification and finding particles. Particle Finder class returns the particle data such as position, particle id or particle's cell. Surface recognizer marks the surface particles when called.
Three classes on the top are related with Marching Cubes algorithm which is responsible for visualizing the particles in a different way. We wrote the Marching Cubes algorithm as a C# script in our project. However, we could not implement the Marching Cubes method into the POF system for obvious reasons (Please read warning page!). You can visit the design specifications document revision 3.0 (DSD 3.0) [2] for detailed explanations of class diagrams.

3.2. Detailed Design

The detailed design part is mentioned in the DSD 3.0. In DSD 3.0 [2], detailed design is explained with activity diagrams of the POF system.

We explained those substructures as four parts of activity diagrams. First activity diagram is Marching Cubes Scalar Calculator which calculates the scalar value for the marching cubes algorithm. Second activity diagram is Surface Recognizer which determines the surface particles. Third activity diagram is Marching Cubes which analyses the surface vertices and decides which configurations of triangles will be sent to draw. Last activity diagram is Hash System which finds cell of a particle, cell id, particle id etc. Hash system is helping to find data briefly.

3.3. Realistic Restrictions and Conditions in the Design

We neglected some aspects of the project on purpose to finish the project on time. The security problems are ignored because the project aims to help everybody who has interested fluid simulations and contribute to science. Normal computers are not capable of running the simulation fluently. User can not increase particle number without sacrificing the performance with normal computers. Simulation computer is required for this job. We assumed that users of the POF system have the necessary equipment and software and know-how to use them.

4. IMPLEMENTATION and TESTS

This section describes the tests and implementation stages of the POF system. Test results are discussed in the performance tests and results title. In the implementation of the POF system title, we shared important code snippets and explained the logic behind it.

4.1. Implementation of the POF System

4.1.1. Problems and Solutions

Firstly, we had to recognize elements of NVIDIA flex, and then we discussed how to classify the particles and wrote the hash system. After, we research methods of identifying surface particles and discussed how to apply it. Last part we wrote a prototype code of the Marching Cubes algorithm to implement into POF system.
We explained how implementations made and code explanations in the product manual 1.0.

Main Problem	**Solution of Problem**
Searching particle data linearly due to 3D space positions and vector3 to integer translation.	Spatial hashing algorithm provides reaching particles by put them into cell data.
Too many particles appear in simulations and handling all of them occurs performance problems.	Do not put into calculations inactive and unnecessary particle on visualization (surface particle finding algorithm).
Duplication of functions and similar methods.	We created a situational surface calculator class and particle finder classes to remove code duplication and increase the functionality of our code.
Increasing inter-community and data transmission complexity.	We constructed the Handler class which is responsible for all data transmission and user controls interface.

Table 4: Problems and Solutions

4.1.2 Implementation of Hash Algorithm

We use a spatial hash algorithm to access the particle position easily. Hash algorithm simplifies the three dimensions of float particle positions to integer id numbers in a specific order.

Cube numbering starts from top left and from left to right and then top to bottom. Then it implies the same operation for the third dimension. On x dimension, we found cell id by subtracting minimum boundary area x from particle x, so it means that numbering increases from left to right. Same logic implies on y dimension, cell id number

increase from top to bottom and similarly cell id number increases from backwards to forward in the z dimension.

We subtracted one from the xid, yid and zid when we are assigning an integer number because we initialize id numbering from zero. Cubex represents how many particles can be fit into the cell. We apply these calculations for all dimensions. We can reach cell id by position without storing it.

```
int xId = (int)Math.Ceiling((partcle.x - _bounds.min.x) / _length) - 1;
int yId = (int)Math.Ceiling((_bounds.max.y - particle.y) / _length) - 1;
int zId = (int)Math.Ceiling((particle.z - _bounds.min.z) / _length) - 1;
float cubeX = (particle.x - _bounds.min.x) % _radius ;
float cubeY = (_bounds.max.y - particle.y) % _radius;
float cubeZ = (particle.z - _bounds.min.z) % _radius;
```

Fig 2: Cell ID Numbering

Vertex index struct represents cells by creating an array on this struct. We keep each vertex as 3d vector struct. We hold index point data in integer list.

```
public struct vertexIndex
{
    public Vector3 vertexs;
    public List<int> pointIndice;
}
```

Fig 3: Group Struct

```
public void FindDimentionalIntervalNum(Bounds bounds, float radius)
{
    this._intervalx = (int)Math.Ceiling((_bounds.max.x - _bounds.min.x) / _ra-
dius); // Calculate the small cube number in x-axis.
    this._intervaly = (int)Math.Ceiling((_bounds.max.y - _bounds.min.y) / _ra-
dius); // Calculate the small cube number in y-axis.
    this._intervalz = (int)Math.Ceiling((_bounds.max.z - _bounds.min.z) / _ra-
dius);
    // Calculate the small cube number in z-axis.
}
```

Fig 4: Hash Size

We created cell ids as indices. By finding how many grids in each dimension we find our hash table size represented as figure 4. We find particles by looking at cells.

We have realized a problem occurs in this section during the implementation. If particle centre intersects with cell boundaries, we must decide to assign which particle will be in which cell. We solve this problem by logically assigning particle closest previous cell. If you imagine many neighbour cubes such as small cubes constructing a bigger cube such as 64 cubes, in the centre intersection point there are 8 possible cubic cells you can assign the intersecting particle. These if cells are looking for all intersecting possibilities for all dimensions. There are 2 to the power of 3 cases because of the spatial space we are dealing with. We did not share the whole code but give a snippet to make more understandable. If there is an intersection, we assign a previous cell. To give an example, let us assume a particle intersects with 2 cells. These cells are called cell id 1 and cell id 2. We assign particle to smaller cell id which is cell id 1.

```
// Five errors in here for fill to fix
        if (cubeX == 0 && cubeY == 0 && cubeZ == 0)
        {
            if (xId > 0 && yId > 0 && zId > 0)
            {
                cubeID = (xId--) + (this._intervalx * (yId--)) + (this._inter-
valx * this._intervaly * (zId--));
                checkS(cubeID, _indice);
            }
            if (xId > 0 && yId > 0)
            {
                cubeID = (xId--) + (this._intervalx * (yId--)) + (this._inter-
valx * this._intervaly * (zId));
                checkS(cubeID, _indice);
            }
```

Fig 5: Intersection Boundaries Check.

For example, in fig. 5. we found that particle on intersection boundaries and we found its second cell on y dimension

So that way we can find the particles by looking in cells instead of a linear search of the particles. The ceiling is implemented to derive integer numbers because particles are float and they are derived according to the AABB size.

4.1.3 Particle Neighbour Algorithm

We must find the effect of the particles in a range on a specific particle and calculate it for each particle. To think mathematically consistent and coherent with the particles, the shape of the volume that we are checking should be spherical. However, the required time and finish the project according to the given time interval has compelled project team to search cubical volumes. We must look at the hashed cell ids in the volume that we are searching to find neighbour particles.
We need four corner cells to solve the problem of finding neighbour particles.

```
int topLeftBackward  = FindID(new Vector3(insideCell.min.x, insideCell.max.y,
insideCell.min.z));
int topLeftForward = FindID(new Vector3(insideCell.min.x, insideCell.max.y,
insideCell.max.z));
int topRightBackward = FindID(new Vector3(insideCell.max.x, insideCell.max.y,
insideCell.min.z));
int bottomLeftBackward  = FindID(new Vector3(insideCell.min.x, insideCell.min.y,
insideCell.min.z));
```

Fig 6: Finding Corner Cells.

These three cell borders are called as tx, ty and tz.

```
int tx = (topRightBackward - topLeftBackward),
    ty = ((bottomLeftBackward - topLeftBackward) / _intervalx),
    tz = ((topLeftForward - topLeftBackward) / (_intervalx * _intervaly));
```

Fig 7: Finding Dimensional Cell Count

We find grid intervals that cells are going towards in every dimension. So that way we can find all cells in this volume just by looking at one cell that we have grouped neighbour particles in that cell.

```
int[] areaNums = Enumerable.Repeat(-1, ( (ty+1) * (tx+1) * (tz+1)) ).ToAr-
ray();
int i = 0, tempNum = topLeftBackward;

for (int k = 0; k < ty; k++)
{
    for (int j = 0; j < tz; j++)
    {
        for (int m = 0; m < tx; m++)
        {
            areaNums[i] = tempNum + m;
            i++;
        }
        tempNum += (_intervalx * _intervaly);
    }
    tempNum = areaNums[0] + (_intervalx * (k + 1));
}
return areaNums;
```

Fig 8: Finding Cell Numbers in Specific area.

4.2 Performance Tests and Results

These performance tests are made in the Unity platform to test our POF system and the performance of its subsystems. We have created a corridor with two obstacles (a capsule and a sphere objects) and released the particles (4096 particles) above a ramp. During the particles are moving in the test scene, we recorded surface recognition performance execution time frame by frame. We have recorded 100 frames and the particles go into a stable position rest of the frames in the scene. That is why we did not record after the particles are in halt position because it would not be correct measuring. The time per frame values could be wrong.

In the test scene, we aim to change the position complexity of the particles. So that why we could create a different combination of position data cases between particles and calculation parameters in the surface recognition system.

We divided the test scene into five phases. The initial position of the test scene is called perfect cube phase as you can see in the figure below. When the user pressed the play button, the particle set will be released, and particles will scatter. Inner particles are painted to blue.

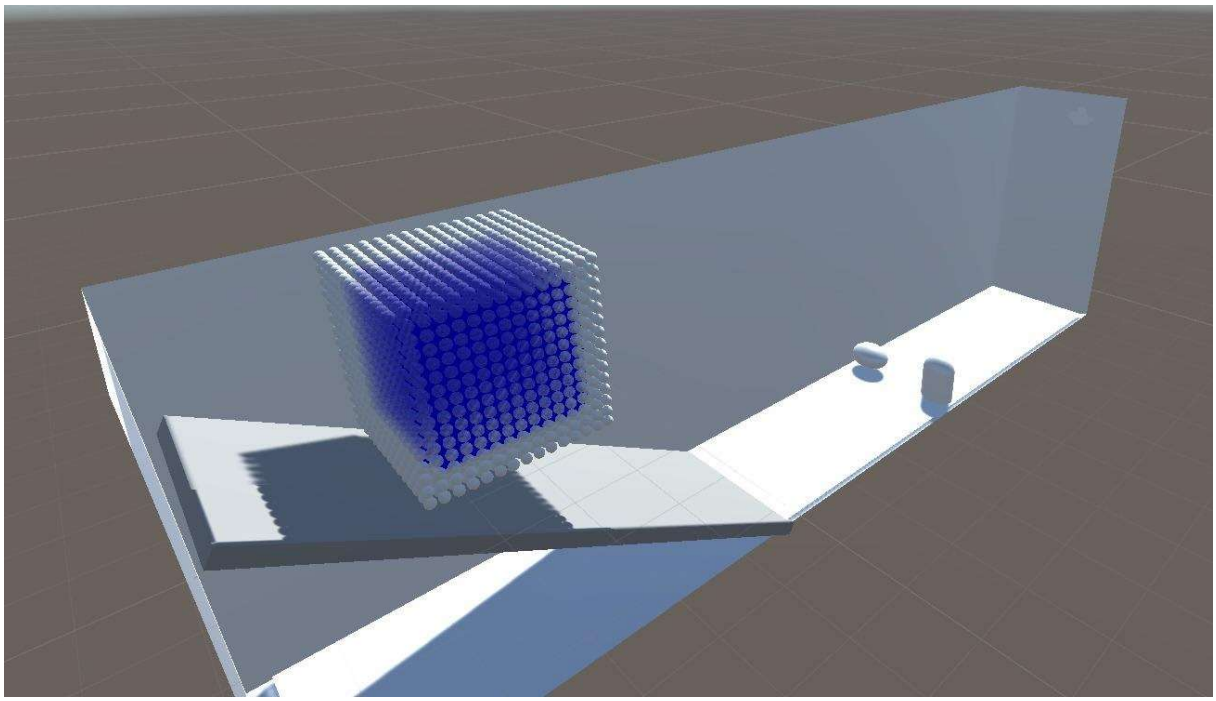


Fig 9: Particles in Perfect Cube Form.

The particles released (4096 particles) and moving downwards through the ramp and complexity of particles is increasing. The particles have approached one another and particle number in a cell is increased. That means more calculation when calculating the weight of a particle and it increases the execution time of the surface recognition.

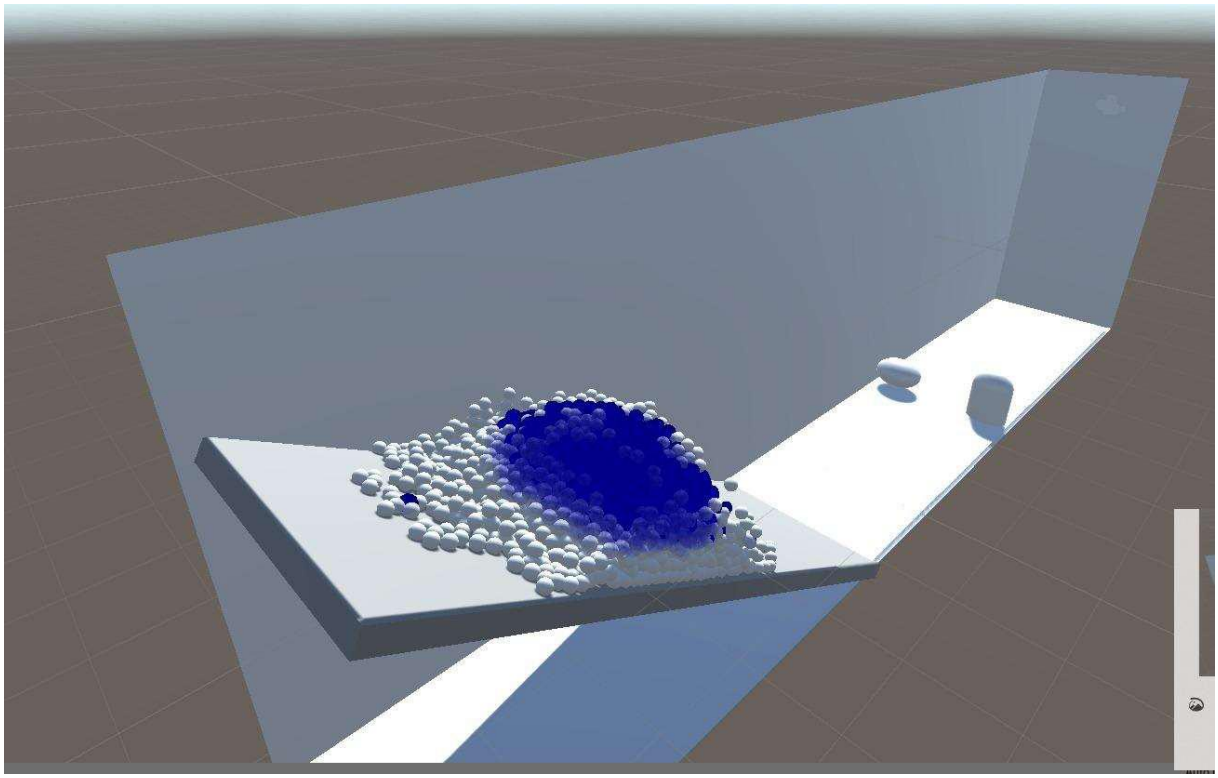


Fig 10: Particles on Ramp.

Particle complexity is decreasing between the ramp and obstacles because particles are just moving forward. When the particles crashed to the obstacles, complexity is increasing. Especially the particles that accumulate in front of the obstacles. These particles are approaching one another but the rest of the particles are going from the middle of the corridor floor.

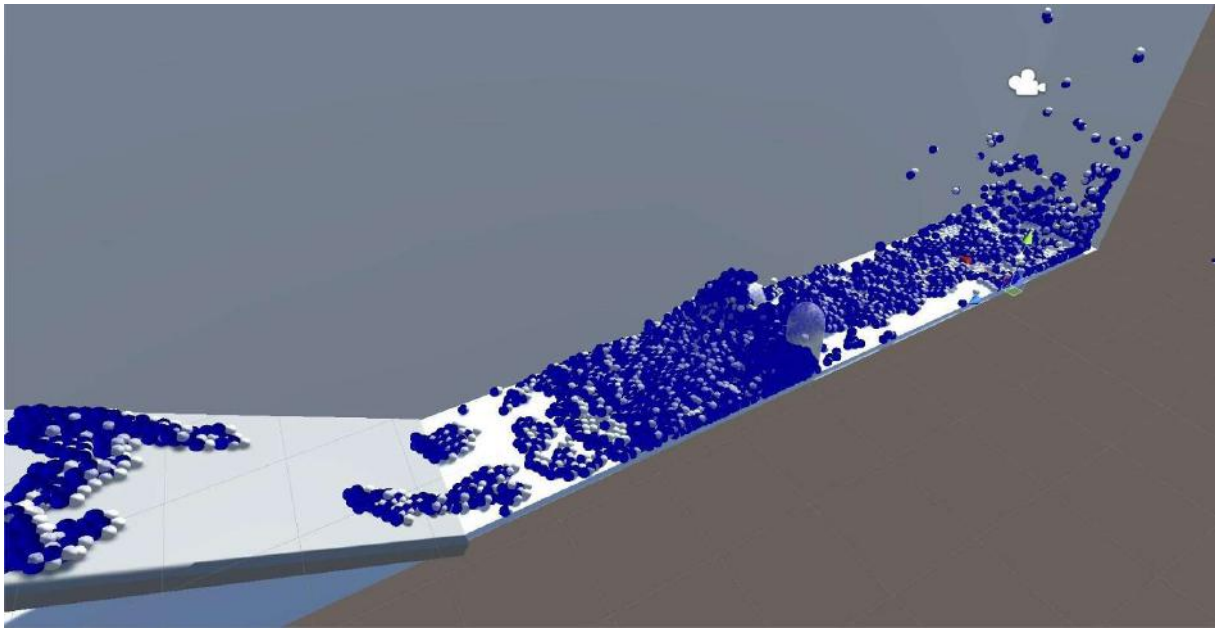


Fig 11: Particles are Crashing to Obstacles.

After the crashing of the particles to the sphere obstacles, particle speed is decreasing, and the other particles position is getting stabilized. That means complexity is decreasing again and surface execution takes less time because of the less computational load.

We deduced the first hundred frame of the scene that we explained, and this is the chart of the surface recognition time per frame chart. We divided a hundred frames of test scene into five phases. As you can see from the chart, the initial execution time is about 700 milliseconds and increasing when moving on the ramp. When the particles pass the ramp and moves through the sphere and capsule obstacles, complexity is decreasing exponentially. At the moment of particles collide with obstacles, particle complexity is increasing, and execution take more time. After the particles pass obstacles in the last phase, particle complexity is decreasing, and surface execution is getting faster.

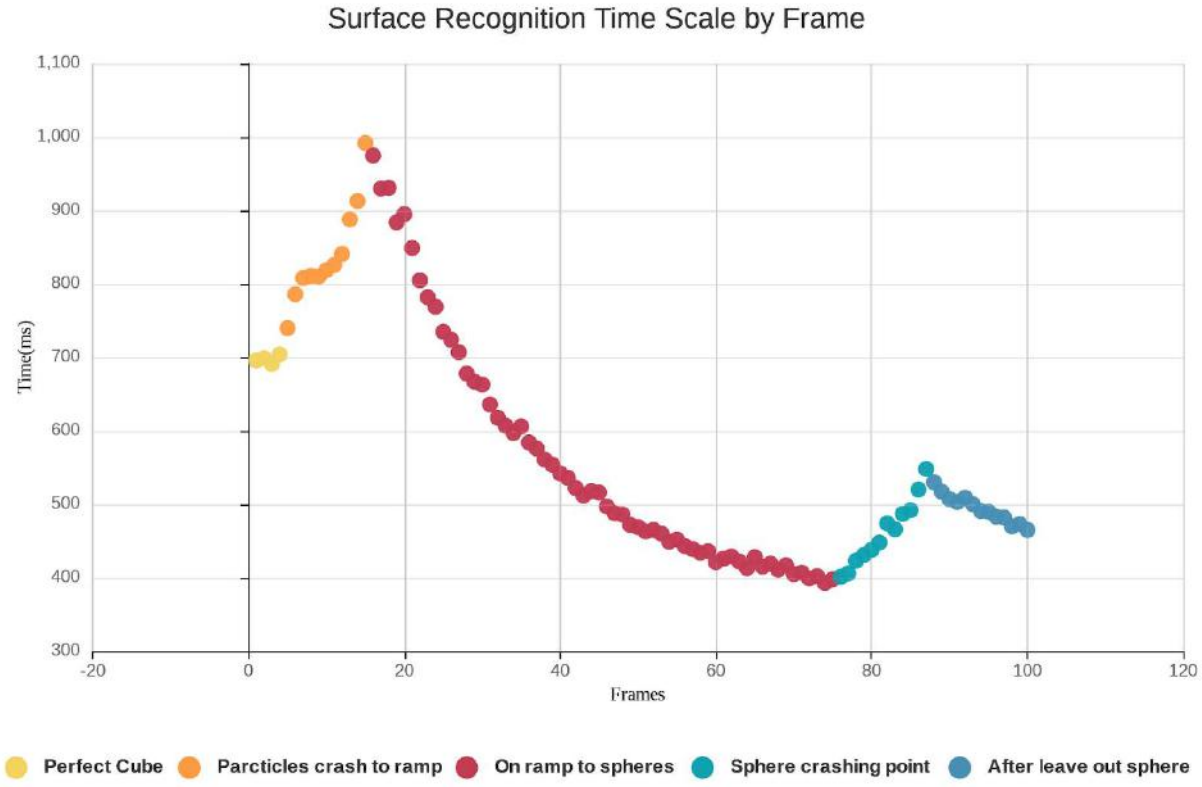


Fig 12: Surface Recognition Time Scale by Frame Chart

We prepared another test measuring scale as particle number. We could not increase the particle numbers more than four thousand due to lack of simulation computer. So, we decided four test cases: One thousand, two thousand, three thousand and four thousand and ninety-six particles.

We decided that measuring the first sixty frames of every particle cases is enough for the testing. We did not change the test scene so we can expect the same results of the previous chart with 4096 particles. The other lines of the different particle numbers are very similar to the yellow line that represents 4096 particles. However, in this test scene we measured 60 frame and that means we did not include particles are colliding to the obstacles phase so that little spike at the right of the chart is not exist compared with the previous chart.

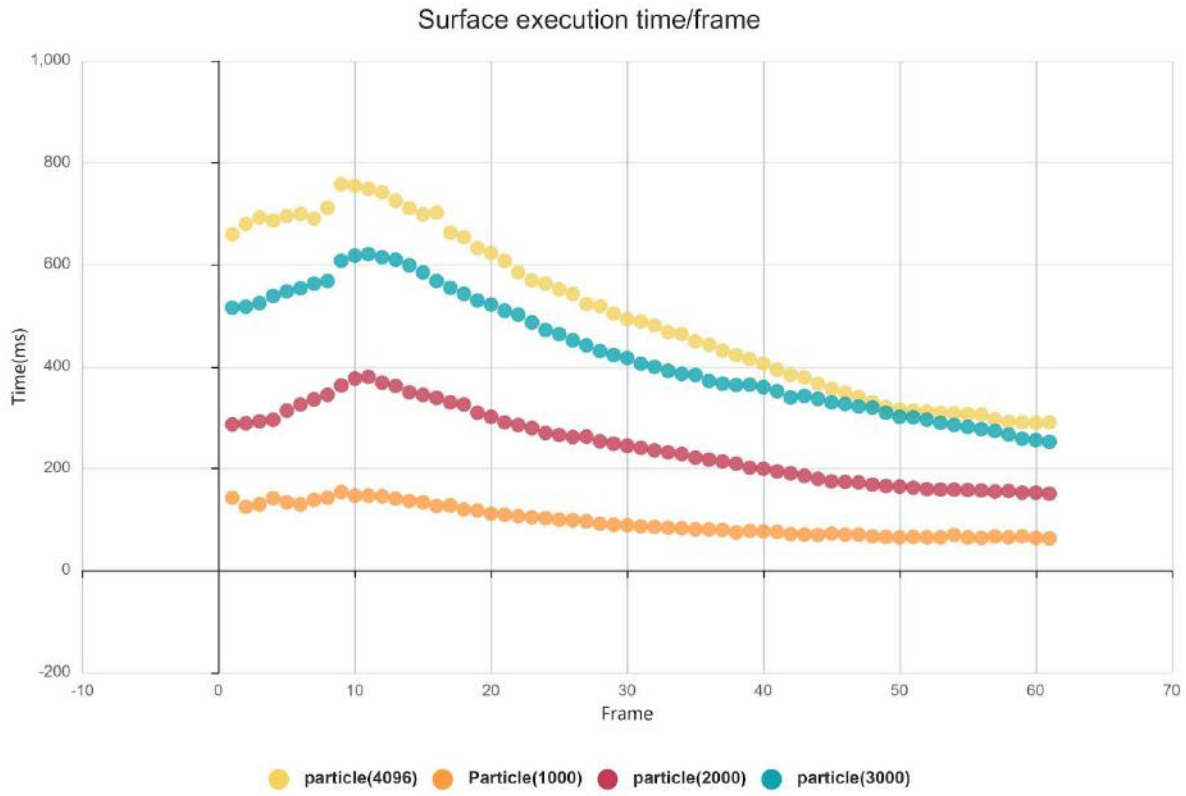


Fig 13: Surface Recognition by particle number comparison chart.

4.3. Environment Testing

In this section, we explained why we decided to use NVIDIA Flex and compared to other particle-based fluid simulations.

4.3.1 Availability of the Necessary Environment

We have tested three particle-based fluid simulation before deciding to use NVIDIA Flex and we disqualified other assets. We stated the reasons for the table below.

Name	**Description**
uFlex	We had small bugs and errors in the code, even though we have fixed minor bugs, the particles were not recognizing the collider of the objects. Collider of the simple primitive objects was not recognized by the Uflex and particles were penetrating the objects. The only plane object was being recognized by the uFlex. The problem could not be solved, and we have changed the fluid simulation.
Obifluid	We eliminated the Obifluid because of performance problems. The expected result was not satisfied by the Obifluid compared to other fluid simulations. Our expectation was reaching 30fps with a hundred thousand particles, but we obtained 3 fps in a scene with three thousand particles.

Screen Space Fluids Pro	We encountered bugs and errors in the code like uFlex. Although we fixed bugs, the performance was very low on higher particle count compared to NVIDIA Flex.

Table 5: Environment Availability.

To give an overview of the test results of the product manual, we showed with images that components of the POF systems operates correctly. Hash System and Surface Recognizer components proved that operating without any problems. In DSD 3.0, we mentioned the working prototype of Marching Cubes code however it will not be implemented into our project. Other test results are included in the product manual revision 1.0 (PM 1.0).

5. CONCLUSIONS

In this section, we summarize our project by explaining the project goals and how we achieved it. We give cost analysis of workers, software, and hardware. We described the benefits of the POF system and fields of usage. Lastly, we reviewed future works as further developments on the project.

5.1. Summary

We stated our requirements in RSD 3.0 [1]. Generally, our requirements were increasing performance and efficiency. Functional requirements except for the last part which is the implementation of Marching Cubes algorithm. The excuse is explained in the warning title page 8. The non-functional requirements are not completely met because we could not finish the project. If we could have finished the last step. There would be tangible proof that all requirements fully met. The marching cubes algorithm would vastly decrease the workload on a GPU. Because drawing and rendering millions of sphere particles instead of drawing triangles to covering the surface vertices take so much time.

5.2. Cost Analysis

In this section we separated project schedule as first and second semester.

5.2.1 Cost Analysis of First Semester

First-semester cost analysis involves the cost of workers, software, and hardware. Besides, we shared the ideal hardware components of the simulation computer after the table of the system that we used.

5.2.1.1 Cost of Workers

Members	Day/Hour	Week/Hour	Semester/Hour	Salary/Hour	Salary/Monthly	TOTAL
Member	8	40	560	30 TL	4800 TL	16800 TL

Table 6: 1. Semester Cost of Workers.

As shown in the cost analysis table, three people works in the project. Every people work equally as workload. Therefore, only one member is represented on the cost table.
Every member works 8 hours a day and 5 days a week. A semester consists of 14 weeks and salary is 30 Turkish Lira per hour. Each member costs 4800 TL per month and costs 16800 in a semester. The salary costs of all three members are 50400 TL per semester. The equivalent of 16800 TL is $2894, 67. Currency translation has made from Dollar / Turkish Lira = 1 / 5.80 on 10 December 2019.

5.2.1.2 Cost of Software

Title of Software	Cost
uFlex	$30
Obi Fluid	$30
SSF	$7
Total Cost	**$67**

Table 7: Cost of software

5.2.1.3 Cost of Hardware

5.2.1.3.1 PC components that used in Project

Total cost = Total employee cost + Total software cost + Total Hardware cost (PC1)

PC 1 components that used in Project	Description
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
Cost of PC 1 per user	$1693, 21
Total cost (for 1 worker)	**$4684, 88**
Total cost (for 3 workers)	**$14054, 64**

Table 8: Cost of PC1

5.2.1.3.2 Optimal Simulation Computer (PC 2)

Total cost = Total employee cost + Total software cost + Total Hardware cost (PC2)

Optimal Simulation Computer (PC 2)	**Description**
Operating System	Windows 10(64-bit Pro)
Processor	8-core Intel i7 5.1 GHz
Memory	32 GB RAM- DDR4- 2666MHz
GPU	NVIDIA Quadro P2200 5GB
Cost of PC 2 per user:	$5017
Total cost (for 1 worker)	**$8008,67**
Total cost (for 3 workers)	**$24026,01**

Table 9: Component costs of PC2

5.2.2 Cost Analysis of Second Semester

In this section cost of workers and hardware tables are given. There is no cost of software and hardware table because we did not spend money on software and hardware on the contrary of the first semester.

5.2.2.1 Cost of workers

In this semester, we could not continue to project code implementations since the university is closed in the middle of March. We spend our working time by completing document work and preparing other materials. Therefore, the cost of workers in April and May is lower than general because we spent fewer hours for obvious reasons. Calculations are made for each month and each member.

Month	**Day/Hour**	**Workday/Month**	**Month/Hour**	**Salary/Hour**	**Salary/Monthly**
February	8	20	160	30 TL	4.800 TL
March	8	22	176	30 TL	5.280 TL
April	4	21	84	30 TL	2.520 TL
May	4	14	56	30 TL	1.680 TL
				Total Salary (for 1 member) = 14.280 TL	
				Total Salary = 42.840 TL	

Table 10: 2. Semester Cost of Workers

5.3. Benefits of the Project

In this section, we listed the areas that we can think about. It could be more areas that our project can benefit.

5.3.1 ***Animations and Movies:*** The POF system can be used in any movies, animations that used fluids.

5.3.2 ***Scientific work:*** Our project benefits scientific areas the most because the project is heavily research and development based on the research papers about the particle-based fluid simulations. Scientist and researchers can use the POF system for their scientific research.

5.3.3 ***Games:*** Games can necessitate a fluid simulation system to make more realistic games. The POF system can be used to obtain more realistic games. For instance, a sailing simulator game is a viable option for our system.

5.3.4 ***Construction:*** The construction and Architecture areas can benefit from our system because the simulation is physics-based which means the POF system is almost realistic. The POF system neglects some imperceptible elastic deformations. For instance, a civil engineer can build a barrage and want to test endurance, on the computer simulation. Therefore, our system can be used for construction and architecture testing.

5.4. Future Work

The first thing to do is implementing the Marching Cubes algorithm and later research algorithms to make the visual output more fluid-like. More research papers can be implemented to the POF system and discuss the results for future work.

References

1. Requirement Specifications Document revision 3.0 (RSD 3.0).

2. Design Specifications Document revision 3.0 (DSD 3.0).

3. Product Manual revision 1.0 (PM 1.0).

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. **[KBSS01]** Kobbelt, Leif & Botsch, Mario & Schwanecke, Ulrich & Seidel, Hans-Peter. (2001). Feature sensitive surface extraction from volume data. Proceedings of the ACM SIGGRAPH Conference on Computer Graphics. 10.1145/383259.383265.

7. **[AIAT81]** G. Akinci, M. Ihmsen, N. Akinci, and M. Teschner. 2012. Parallel Surface Reconstruction for Particle-Based Fluids. Comput. Graph. Forum 31, 6 (September 2012), 1797–1809. DOI:https://doi.org/10.1111/j.1467-8659.2012.02096.x

APPENDICES

APPENDIX A: REQUIREMENTS SPECIFICATIONS DOCUMENT

COMP4920 Senior Design Project II, Spring 2020

Advisor: Mehmet Ufuk Çağlayan

POF: Performance Optimized Fluid Requirements Specifications Document

1.5.2020

Revision 3.0

By:

Baran Budak-15070001012

Cihanser Çalışkan-16070001020

İsmail Mekan-15070001048

Revision History

Revision	Date	Explanation
1.0	03.11.2019	Initial requirements.
2.0	19.12.2019	Requirements model.
2.1	20.4.2020	Grammar errors fixed.
2.2	21.4.2020	Diagrams updated.
2.3	22.4.2020	Explanations added to the mathematical equations.
2.4	22.4.2020	Warning page added.
2.5	23.4.2020	Introduction section writings extended.
2.6	25.4.2020	Surface recognition formulas with explanations added.
3.0	1.5.2020	Non-functional requirements extended.

Table of Contents

List of Tables

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.

1.0 Introduction

Introduction part consists of three parts: Purpose of the POF system, the scope of the POF system and lastly, we give an overview.

1.1. *Purpose*

The purpose of the performance-optimized fluid (POF) system is to research and apply surface reconstruction methods to create a more efficient particle-based simulation system.

The POF system should increase the efficiency of the simulation. We obtain that by analysing situations of particles of each frame and finding the surface particles by using the hash system. Hash system utilizes to lowering the memory consumption.

POF system aims to develop an alternative way to analyze and visualize particles in each frame. We endeavour to research and development in smoothed-particle hydrodynamics (SPH).

In detail, the POF system is reconstructing the surface particles by benefiting from various research papers mentioned. The POF system approaches particles as a continuum and inspects the fluid as a whole object. Herewith, system approaches to fluids as there are no separate particles but rather the fluid is a continuous material. POF system analyses particles in each frame and applies specific algorithms.

1.2. *Scope*

POF system helps to increase performance for particle-based fluid simulation. POF system work as dependent on the NVIDIA Flex and Unity game engine. POF reduces computations for visualization of particles. We increase the computation in the background slightly, but we increase the performance much better.

Initially, there must be a handler class to manage all communications between the sectors of the POF system. When the simulation starts, NVIDIA Flex creates the particles and particle position data is transmitted to the POF system. Hash system receives position data and retrieves the particle id or the cell id of a particle when asked. We created cells in the axis-aligned boundary box (AABB) which is useful to find and analyze particles. POF system must find surface particles by calculating weight function and applying it as mentioned in the research papers. We calculate these functions for a specific range and these values are predetermined intuitionally. POF system computes the colour field quantity of each particle and marks all the surface particles. We find the particles in a cell and calculate how every particle affects the other particles as a scalar value. This is an implementation of Zhu and Bridson method [7]. POF construct the surface vertices as mesh triangles and particles reach their last appearance. However, this last part of the POF system could not be implemented due to the reasons that we have mentioned in warning page.

1.3. *Overview*

This document describes the POF system. Requirements specification document included with user case and sequence diagrams that define user roles in the system and more importantly, explaining how the system operates in the background. Further information exists on the Design Specifications Document revision 3.0 (DSD 3.0) [8].

The document focuses on describing the POF system project requirements. System functions are examined in two separate sections as functional and non-functional requirements. The function of NVIDIA Flex and how it is used in the POF system is described. User characteristics and constraints specify how the POF system can work under which circumstances.

2.0 Diagrams

This section gives use case and sequence diagrams which are design part of our project. Detailed information and explanations are in DSD 3.0 [8].

2.1 *Use Case Diagram*

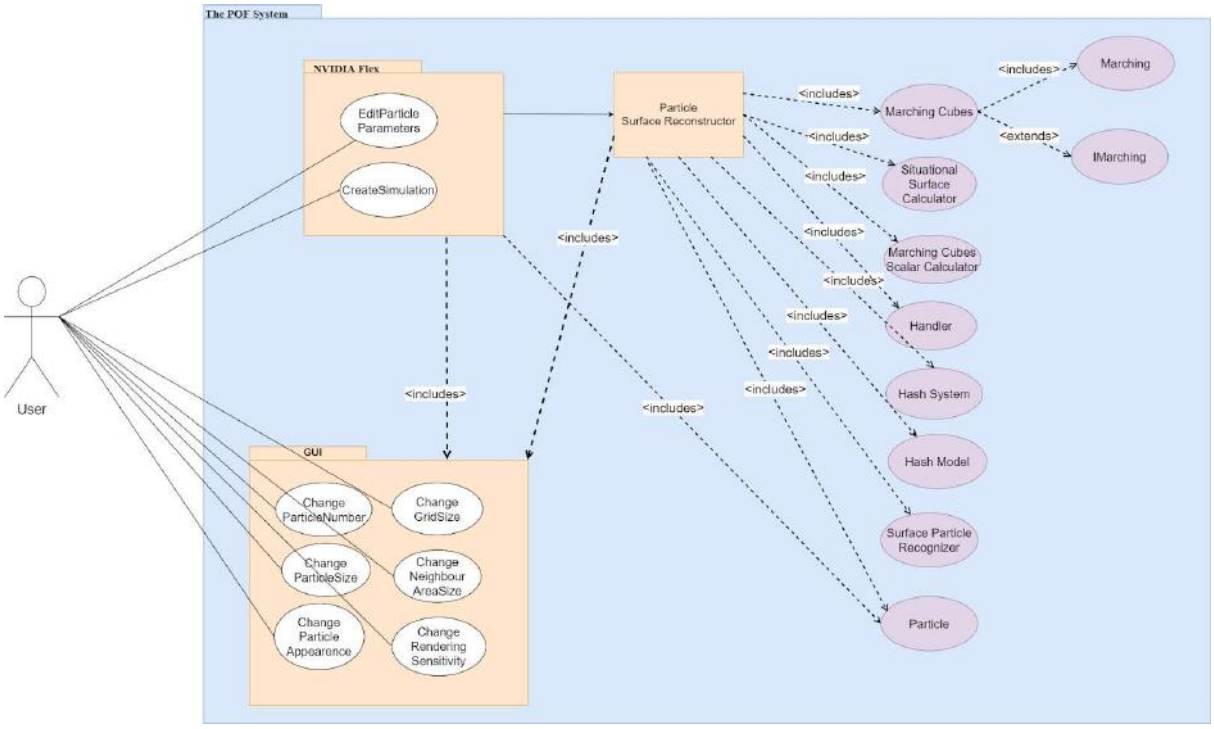


Fig 1: Use Case Diagram

2.2 *Sequence Diagram*

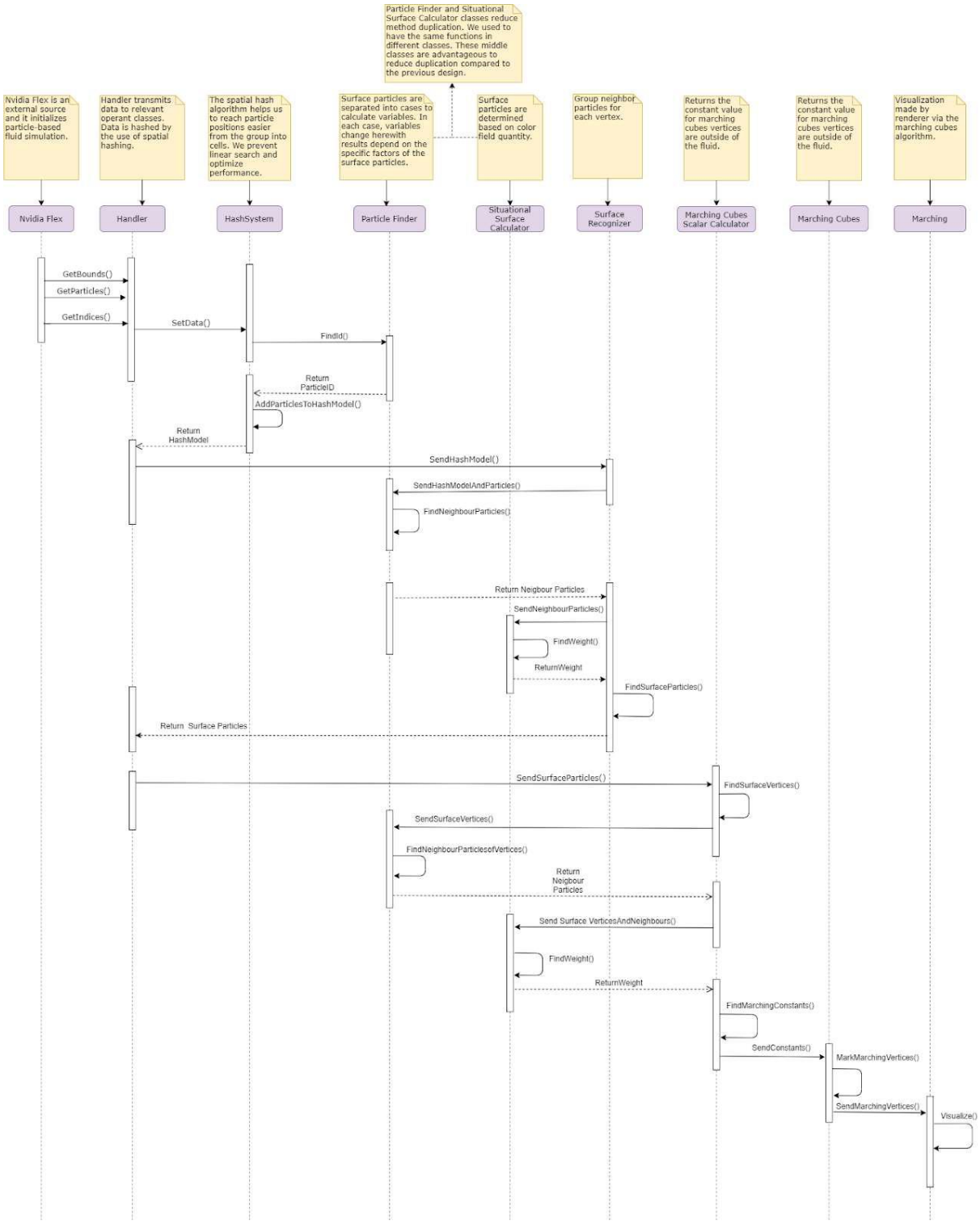


Fig 2: Sequence Diagram

3.0 General Description

The requirement specification document provides a high-level description of the requirements of the POF system. The documentation describes the operation of sections in the POF system and explains its roles.

3.1. System Functions

The POF system shall retrieve the position data of the particles and AABB which is created by the NVIDIA flex particle-based fluid simulation system. The POF system computes the colour field quantity of each particle and marks the surface particles and restore their vertices. The POF system calculates the cell id for each particle and calculates the scalar value of how the particles affect each other by using [7].

3.2. NVIDIA Flex

NVIDIA Flex is a particle-based simulation technique for real-time visual effects. It is an outside source tool for our project. NVIDIA Flex is used for creating particles data to process in our algorithm. NVIDIA Flex is an obligatory external tool since we cannot create fluid simulation from scratch, and it is not our project aim either. The aim of the POF system is enhancing the performance of the already existed particle-based fluid system.

4.0 Functional Requirements

This section describes the functional requirements and stages of the POF system.

4.1 *Retrieve Particle Data*

The POF system takes the particles from NVIDIA Flex which creates the particles and particle attributes such as radius adhesion, damping, and restitution. NVIDIA flex creates an axis-aligned boundary box by looking at the coordinates of particles. AABB is a necessary preliminary step for dividing into cells.

4.2 *Divide into Cells*

The reason for using AABB is to make the search algorithm is more efficient. Axis aligned bounding box is divided into cubic cells to analyze the situation of the particle. Cells are divided by the ratio of one-eight times of radius for the Marching cubes algorithm [6] initialization. Cubes are an easy way to reach vertex information. Instead of holding eight vertex data, the system holds a cube position and it is a memory-efficient

way. The POF system uses these cells to calculate the scalar values of the particles inside the cells by using [7].

4.2.1 Zhu and Bridson

The research paper written by Zhu and Bridson [7] offers an alternative to simulate liquids. The paper mentions surface reconstruction from particle and gives the functions and formulas to apply the method. Zhu and Bridson [7] calculate a scalar value of vertices that outside of the fluid to send marching cubes to visualize.

4.2.2 Mathematical Equations

Mathematical equations given below are belongs to the Zhu and Bridson research paper. We must implement these equations as code script in our POF system. Brief explanations are given below for each equation.

$$\varphi(x) = \left|x - x_0\right| - r_0 \qquad \textbf{(1)}$$

This equation finds signed distance value of a particle according to its neighbours. X0 is all neighbours average position as one dimension. X is a single particle position. The radius of a particle is subtracted because X values represents the centre of the particles. is how much close to their neighbours.

$$\varphi(x) = |x - \bar{x}| - \bar{r} \qquad \textbf{(2)}$$

This is the previous same formula with a small change. X bar in abstract represents average position data of the neighbour particles of a particle in a specific range. R bar represents the average radii of all particles. In our system all particles have the same value so we can replace it with a constant value.

$$\bar{x} = \sum_i w_i x_i \qquad \textbf{(3)}$$

This equation is used to calculate x bar. For each particle, position data is multiplied with weight data and the sum of these values are equals to the x bar.

$$w_i = \frac{k(|x - x_i|/R)}{\sum_j K(|x - x_j|/R)} \qquad \textbf{(4)}$$

We know the variables in the absolute value. K is a kernel function;

k(s) = max (0 , $(1 - s^2)^3$). R is the radius of the volume around the particle which is x as the previous equations. In Zhu and Bridson [7] writers recommend the R value as twice of the particle spacing. In the denominator, we find the distance as we mentioned in other relevant equations but this time, we add these values. It explains the particle weight which the effect of a particle is compared to other neighbour particles.

4.2.3 What is Kernel Function?

The parameters of a kernel function can be anything such as two integers or two vectors, trees whatever provided that the kernel function knows how to compare them.

4.2.4 What is Weight?

Weight function calculates a single particle how much affected by the summation of every other particle in a specific range in space. Represented as 'w' in mathematical equations.

4.2.5 Particle Classification

Particle classification is one of the main reasons why the POF system offers a better alternative. Grouping particles and by searching and calculating from these bigger parts of the fluid volumes make the system more efficient and faster in terms of computation.

4.3 *Surface Recognition*

The surface recognizer algorithm detects surface particles and their cells so we can discard inactive cells and focus on the surface particles. This method makes the system more efficient and results with better performance by discarding unnecessary cells. The POF system finds each particle.

$$\varphi(x) = |x - \bar{x}| - \bar{r}f \qquad \textbf{(5)}$$

This equation helps us to decide whether the particle is a surface particle. X is the particle. X bar is the neighbour particles of x in a specific range. If φ(x) = 0, we can say that particle is on the surface. r is the radius. f is a scalar value and there are other

calculations mentioned to find f. However, we do not consider the f value while we are implementing in this equation because of the project deadline.

$$\bar{x} = \frac{\Sigma_j x_j k(|x - x_j|/R)}{\Sigma_j k(|x - x_j|/R)} \quad \textbf{(6)}$$

This equation finds the effect of a particle as a weight in a system. Equation finds a particle distance and divides to the other neighbour particles average distance. R is the specific range that we look into neighbours of x. xj is the particles that stay in the range of R. k is the kernel function which is explained in section 4.3.2 Kernel Function.

4.3.1 Colour Field Quantity

Colour field quantity is a mathematical function that calculates a particle that is affected by the other particles. Because of this function, the POF system determines whether a particle is a surface particle.

$$c_s(r) = \sum_J m_J \frac{1}{p_J} W(r - r_J, h) \quad \textbf{(7)}$$

4.3.2 Kernel Function

The kernel function is necessary for kernel and particle approximation of a field function and its derivatives. Kernel function formula:

$$\mathrm{k(s)} = \max(0,\ (1 - s^2)^3) \quad \textbf{(8)}$$

4.3.3 Weight Function

A gradient is a vector-valued function that computes a particle verge which direction.

4.3.4 Marking Cells and Vertices

We must find the particles in cells and calculate how every particle affects the other neighbour particles. Vertices of these particles are marked as well.

4.4 *Performance*

This requirement can be accepted as both kinds of requirement types. The project does not give this requirement as mandatory, but to achieve performance has significant importance. This requirement explained in non-functional requirements.

5.0 Non-Functional Requirements

Non-Functional requirements are listed in the table below. These requirements are the main goals of the POF system. Some requirements may not be satisfied because they are not promised to realize since our project is research and development based.

Non-Functional Requirements	Description
Efficiency	The aim of the POF system is efficient memory usage. For instance, because of the hash system, we divide 3D space into cells and instead of searching a particle linearly, we only search the particles a cell which is a more efficient way to search particles.
Performance	The system's performance should be increased with the POF system. Due to the POF system, particle simulation has a higher fps rate, or it can be run at lower-end devices. The existed methods will be checked whether it can be developed or not.
Usability	Similar fluid systems are developed on many platforms. However, our project can execute in the Unity game engine which is supported on Windows and macOS. User's Computer must be met with specified software and hardware requirements.
Testability	The POF system is testable with different conditions such as different particle attributes or different scenes. Controllability is not possible fully because NVIDIA Flex inter-communicates with the GPU by using parallel programming methods.

Table 1: Non-Functional Requirements

6.0 Glossary

Term	Description
API	Application Programming Interface.
AABB	Acronym for Axis Aligned Boundary Box. Bounding volume for a set of objects is a closed volume that completely contains the union of the objects in the set.

Cell	Axis aligned bounding box is divided into small identical cubes.
Colour field quantity	Function that calculates how each particle is affected by all the other particles.
CPU	Central Processing Unit.
GPU	Graphic Processing Unit.
Gradient	The directional derivative of a scalar field gives a vector field directed towards where the increment is most, and its magnitude is equal to the greatest value of the change.
Grid	Series of vertical and horizontal lines that are used to subdivide AABB vertically and horizontally into cells in three-dimensional space.
Iso-surface	An isosurface is a 3D surface representation of points with equal values in a 3D data distribution which is the 3D equivalent of a contour line.
Marching Cubes	Marching cubes is a computer graphics algorithm, published in 1987 for extracting a polygonal mesh of an isosurface from a three-dimensional discrete scalar field.
Mesh	A mesh consists of triangles arranged in 3D space to create the impression of a solid object.
NVIDIA Flex	NVIDIA Flex is a particle-based simulation technique for real-time visual effects.
OPENGL	Open Graphics Library is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics.
POF	Performance-Optimized fluids.
Polygonal Mesh	A polygon mesh is the collection of vertices, edges, and faces that make up a 3D object.
SPH	Smoothed-Particle Hydrodynamics
Unity 3D	Unity is a cross-platform game engine developed by Unity Technologies. Unity is used for developing video games and simulations for consoles and mobile devices.
Visual Studio	Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft.

Table 2: Glossary

7.0 User Characteristics

The performance-optimized fluid system can be used by anyone who has an interest in particle-based fluid simulation.

8.0 General Constraints

A D3D11 capable graphics card with the following driver versions:

NVIDIA: GeForce Game Ready Driver 372.90 or above.

AMD: Radeon Software Version 16.9.1 or above.

To build the demo at least one of the following is required:

Microsoft Visual Studio 2013 or above.

G++ 4.6.3 or higher

CUDA 8.0.44 or higher

DirectX 11/12 SDK

References

1. **[AIA12]** Akinci, G., Ihmsen, M., Akinci, N. and Teschner, M. (2012). Parallel Surface Reconstruction for Particle-Based Fluids. Computer Graphics Forum, 31, 1797-1809.

2. **[BP94]** Paul Bourke 1994, Marching Cubes, viewed 1 May 2020, <**http://paulbourke.net/geometry/polygonise/**>

3. **[MCG03]** M. Müller, D. Charypar, and M. Gross (2003). Particle-based fluid simulation for interactive applications. In Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '03), 154–159.

4. **[PTB03]** Premžoe, S. , Tasdizen, T. , Bigler, J. , Lefohn, A. and Whitaker, R. T. (2003). Particle-Based Simulation of Fluids. Computer Graphics Forum, 22, 401-410.

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

6. **[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.

7. **[ZB05]** Zhu, Y., & Bridson, R. (2005). Animating sand as a fluid. (New York, NY, USA, 2005) *ACM Trans. Graph., 24*, 965-972.

8. Design Specification Document revision 3.0 (DSD 3.0)

APPENDIX B: DESIGN SPECIFICATIONS DOCUMENT

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

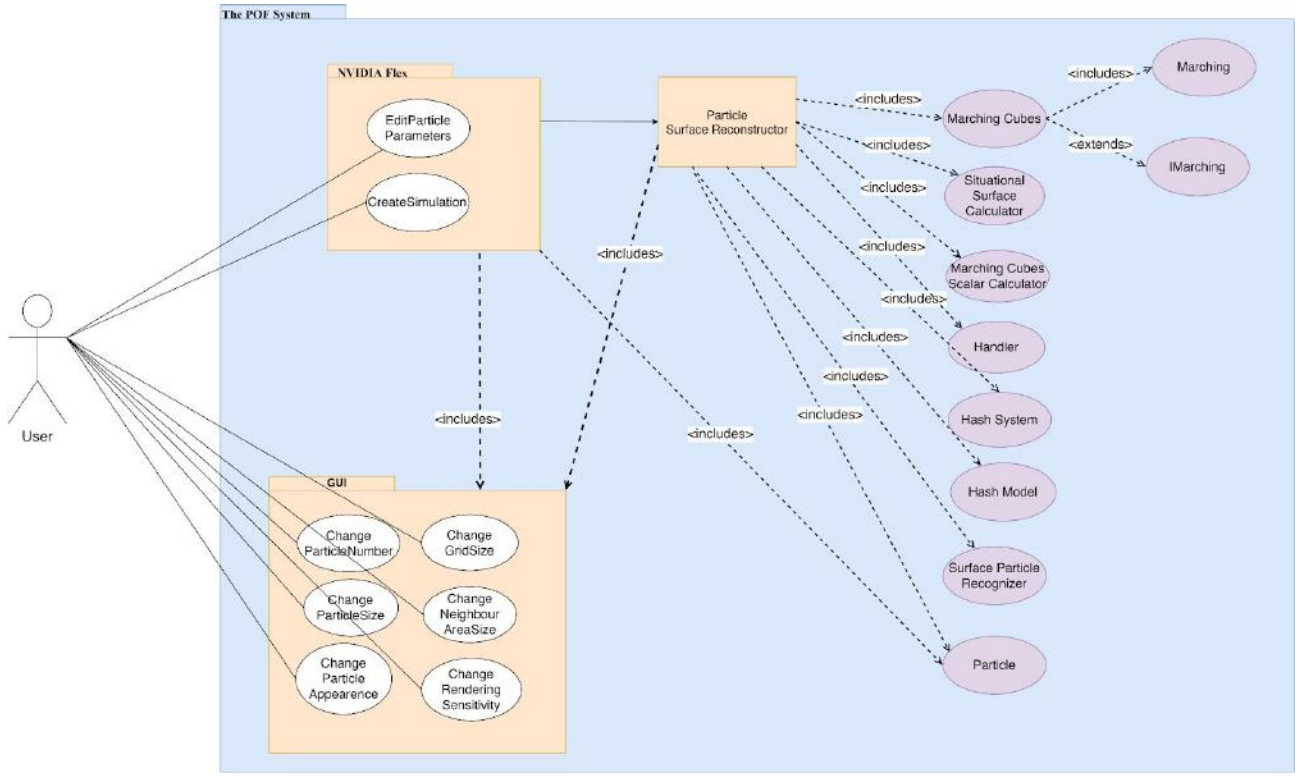


Fig 1: Use case diagram

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

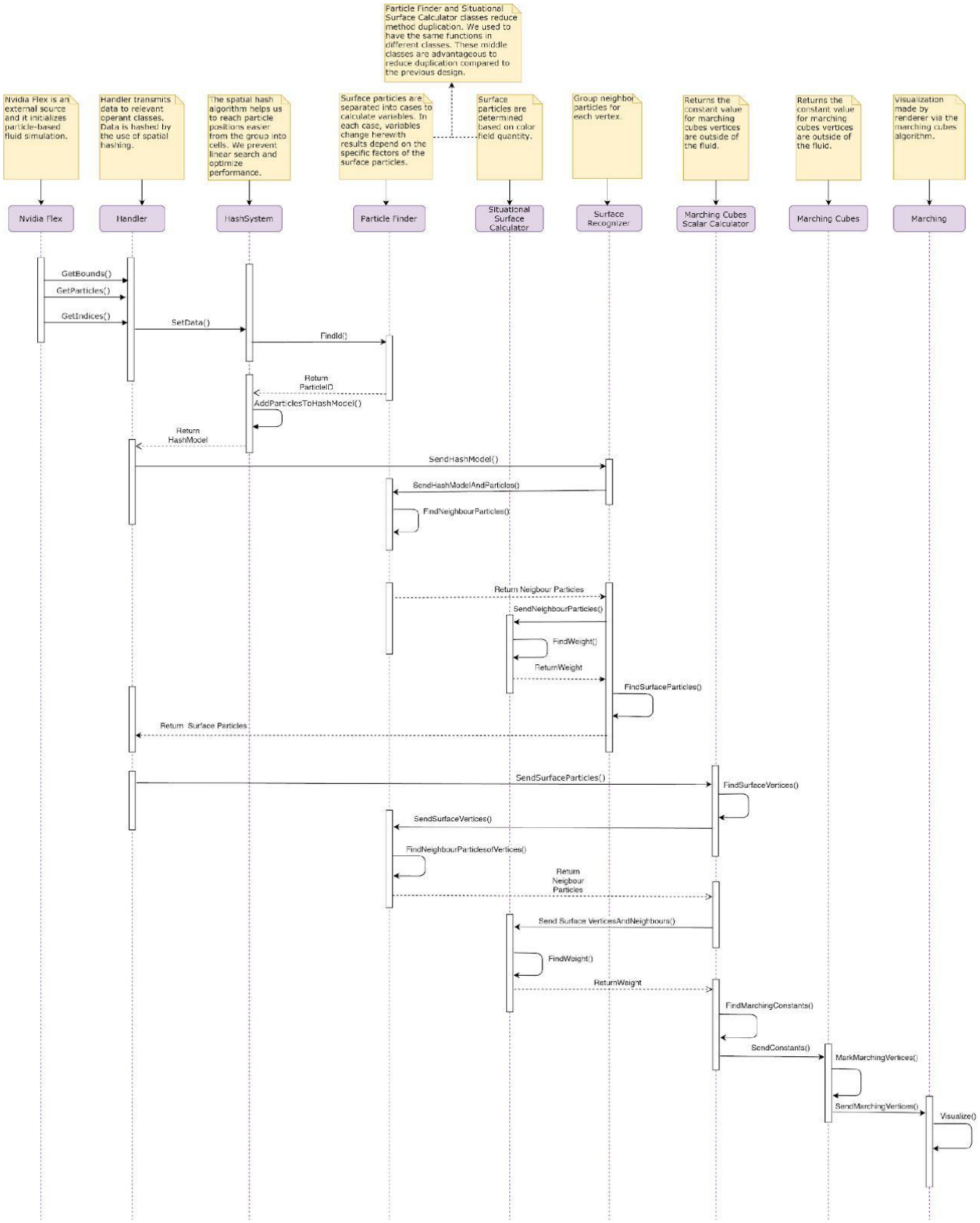


Fig 2: Sequence diagram

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

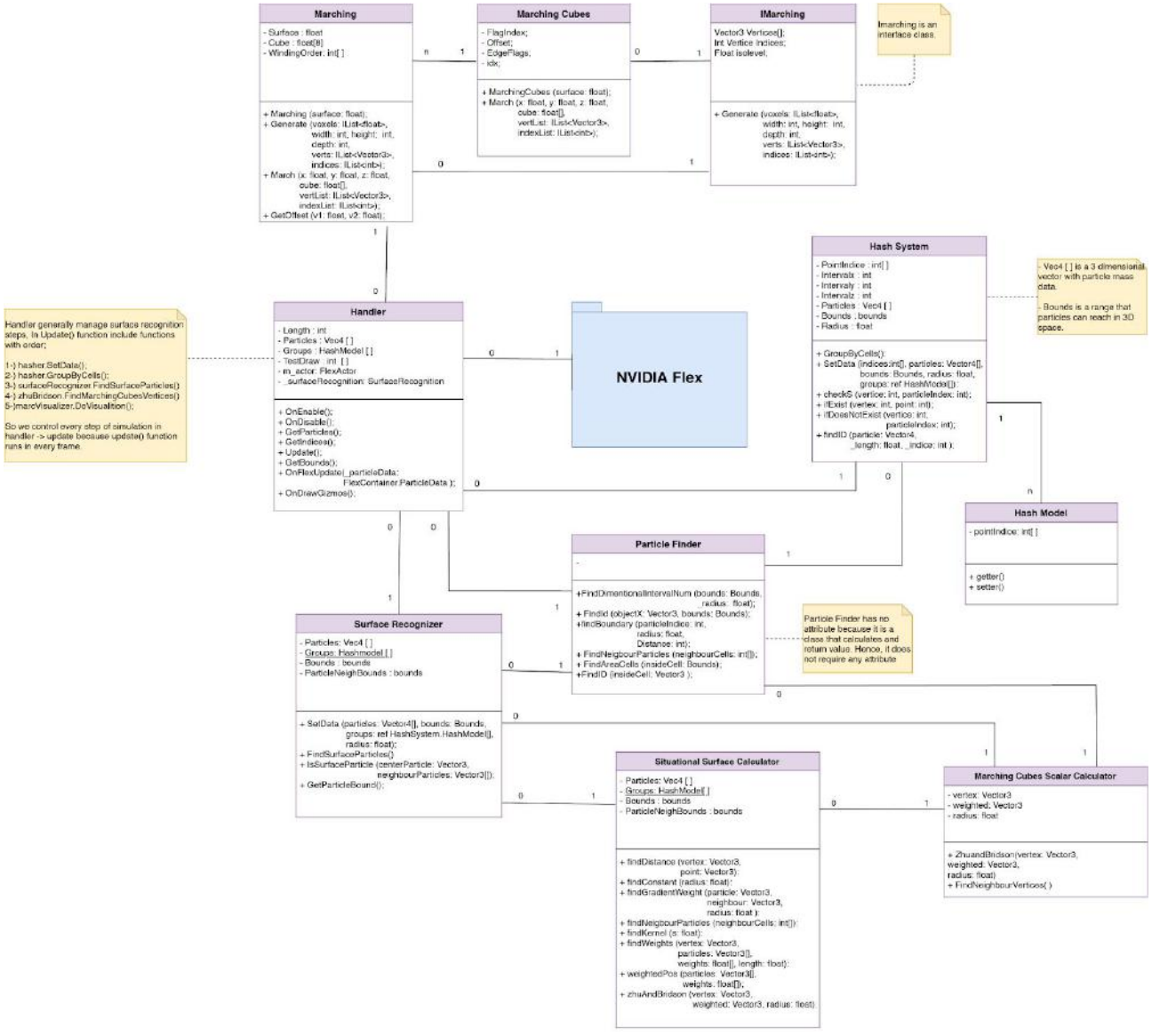


Fig 3: Class diagram

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

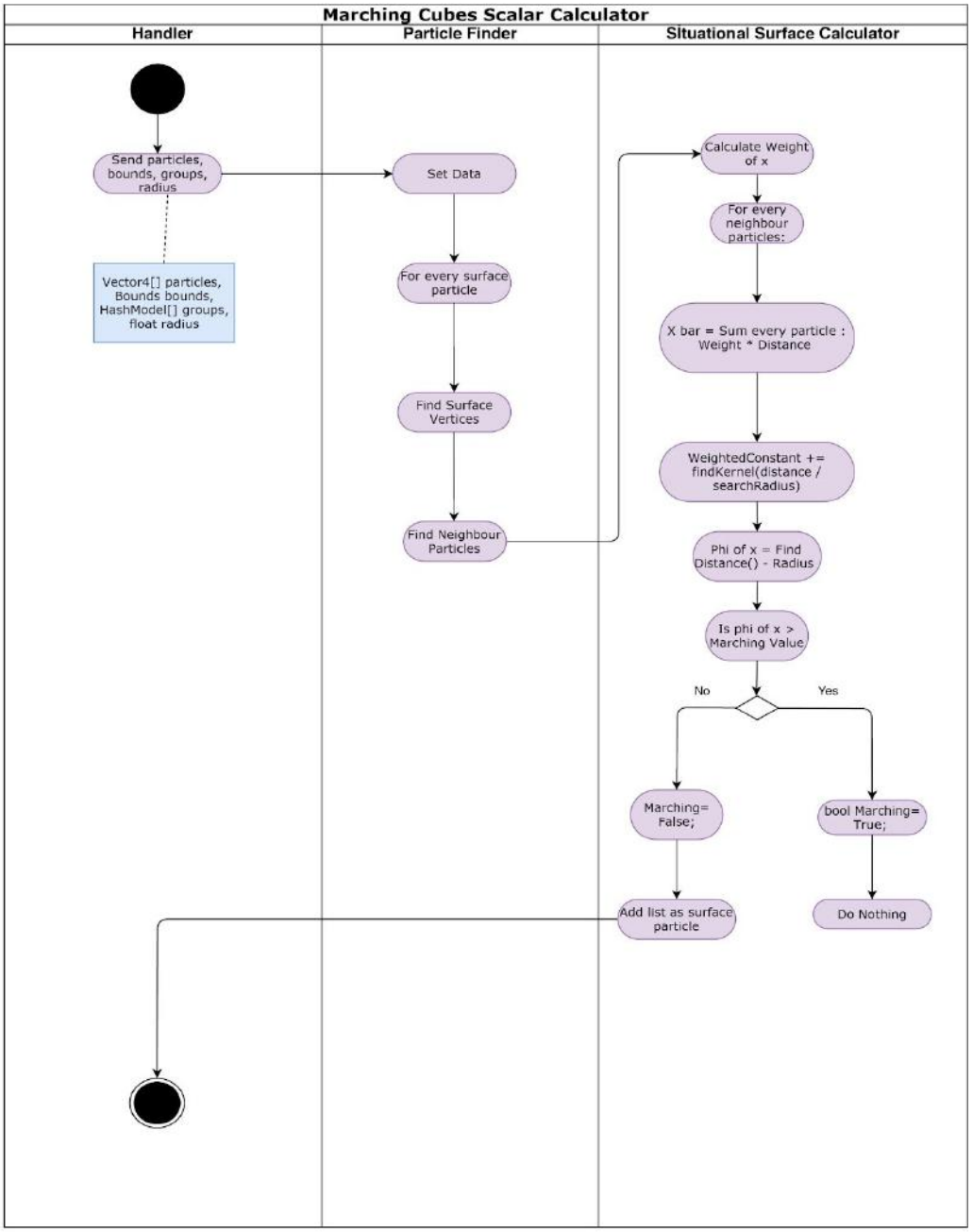


Fig 4: Activity diagram of Marching Cubes Scalar Calculator

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

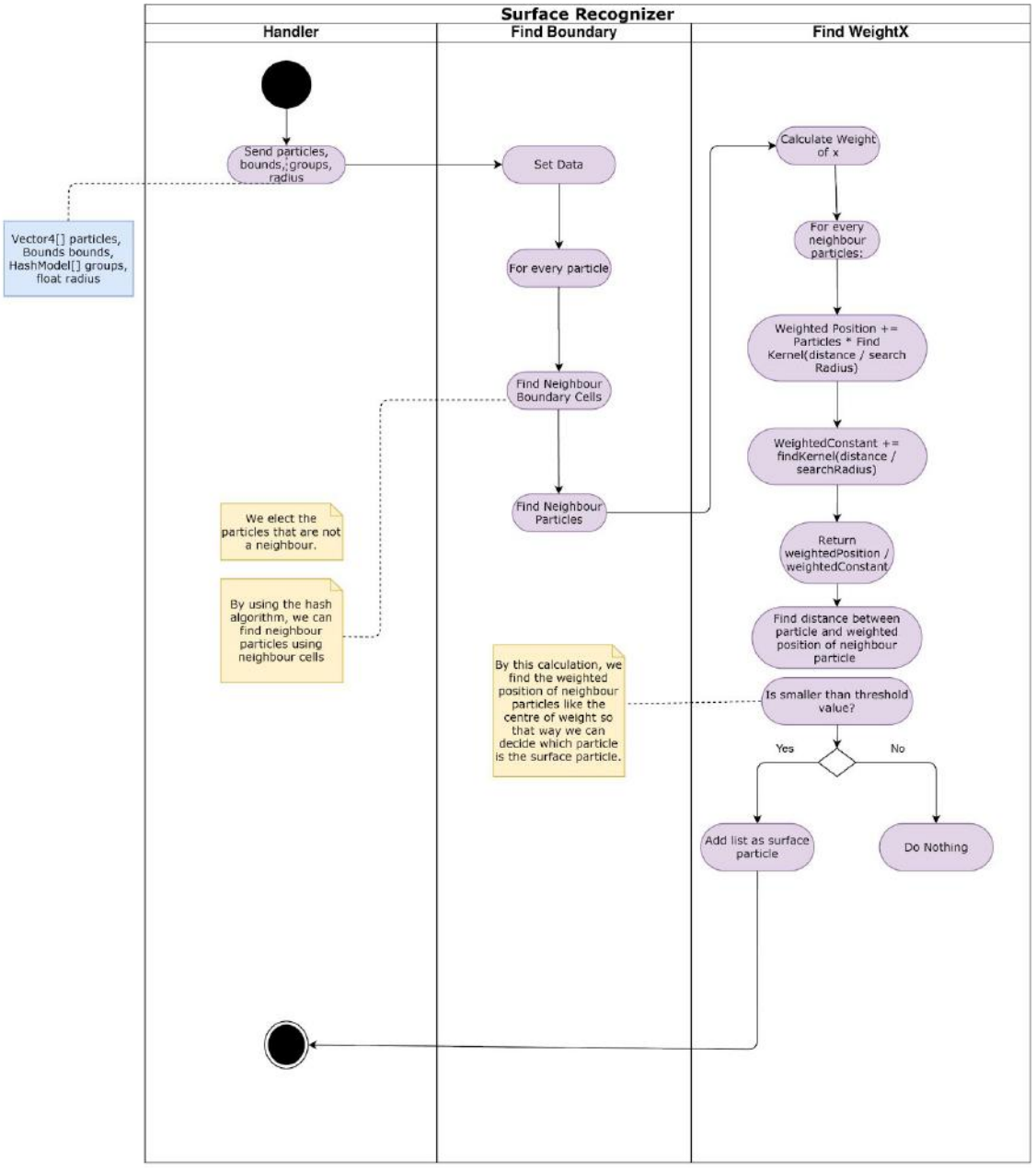


Fig 5: Activity diagram of Surface Recognizer

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

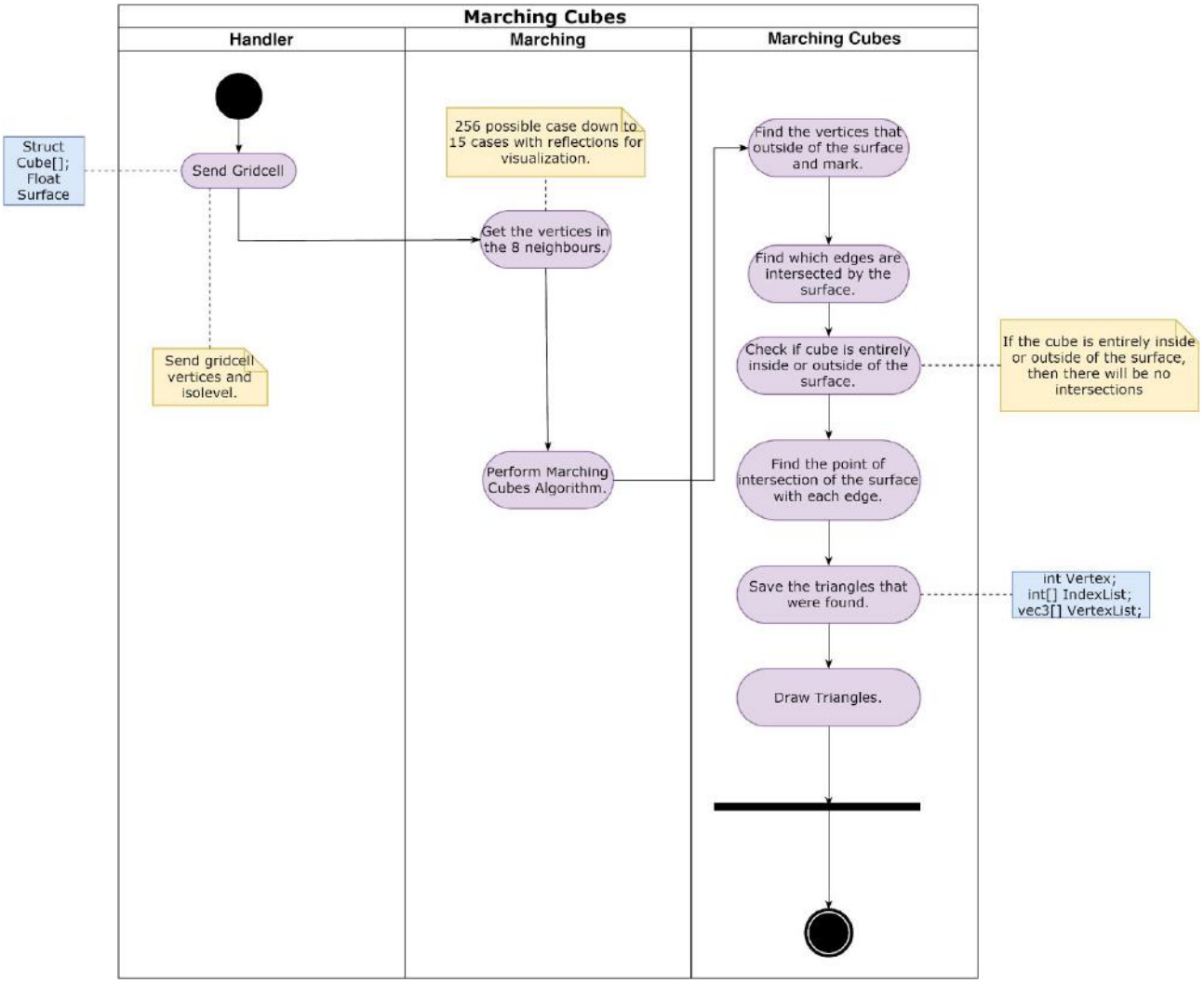


Fig 6: Activity diagram of Marching Cubes

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

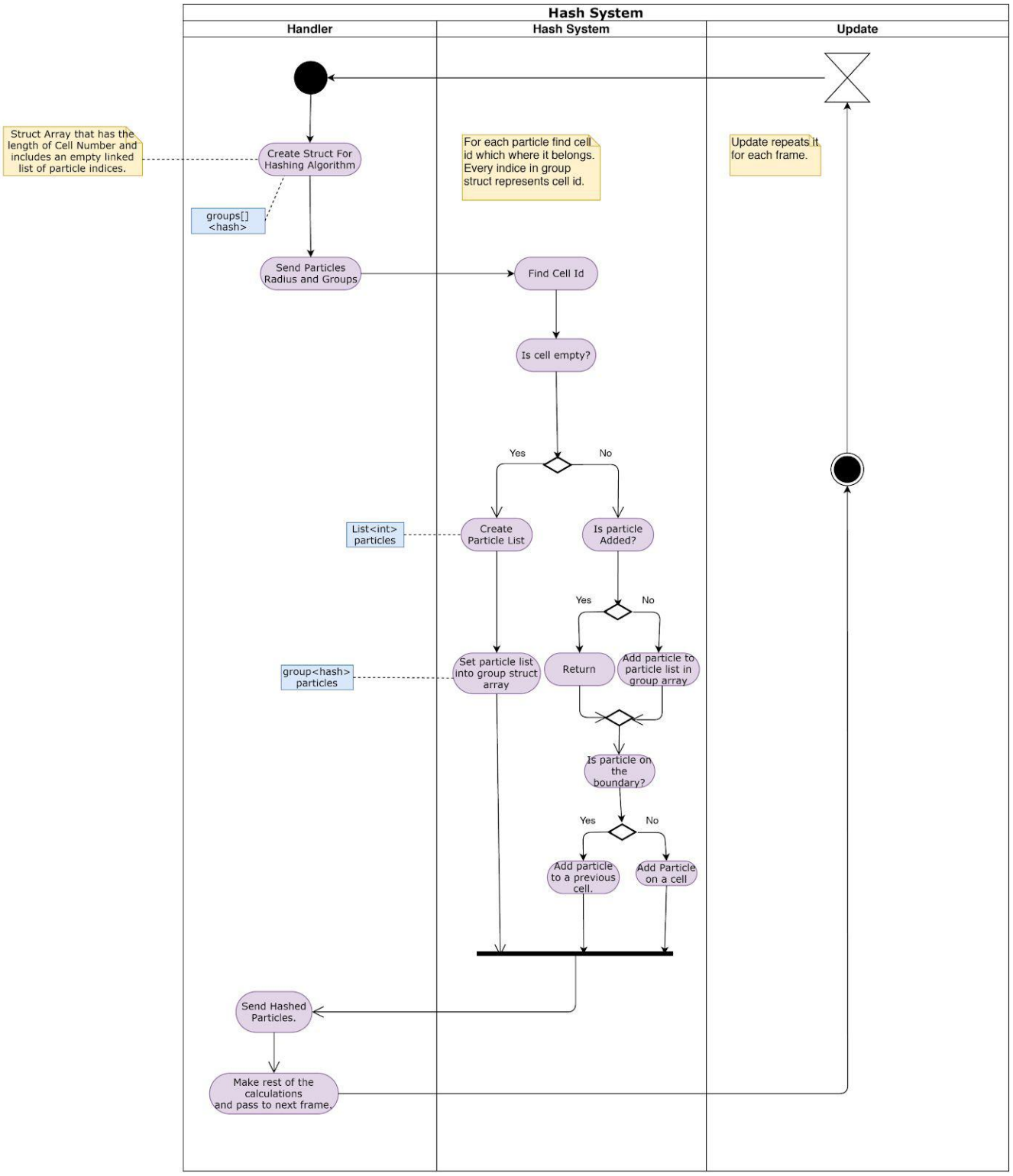


Fig 7: Activity diagram of Hash System

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.

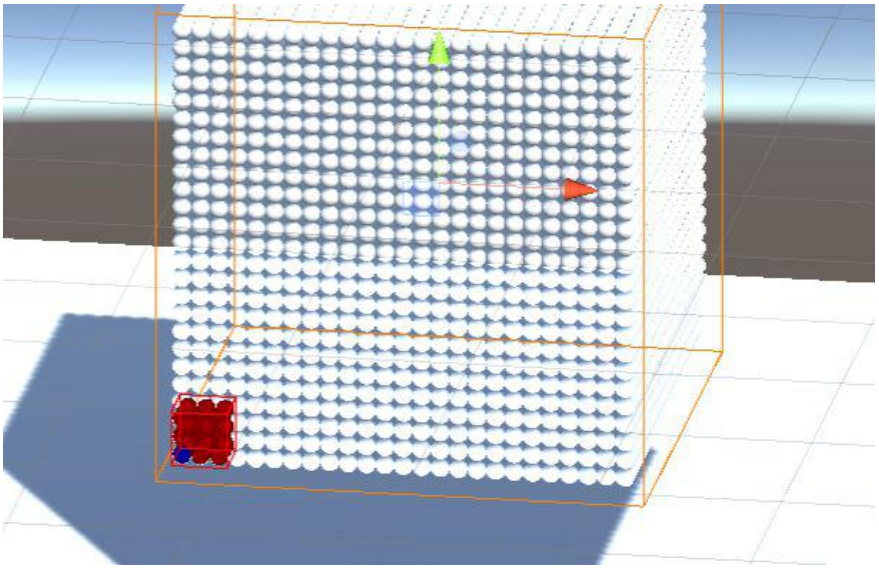


Fig 8: Particle and its neighbours in the cell.

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.

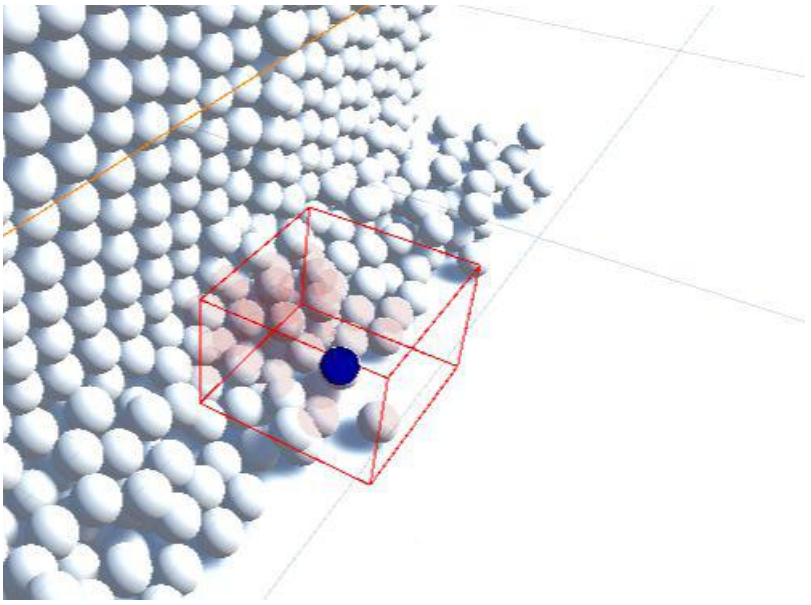


Fig 9: Tracking particle in a cell.

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.

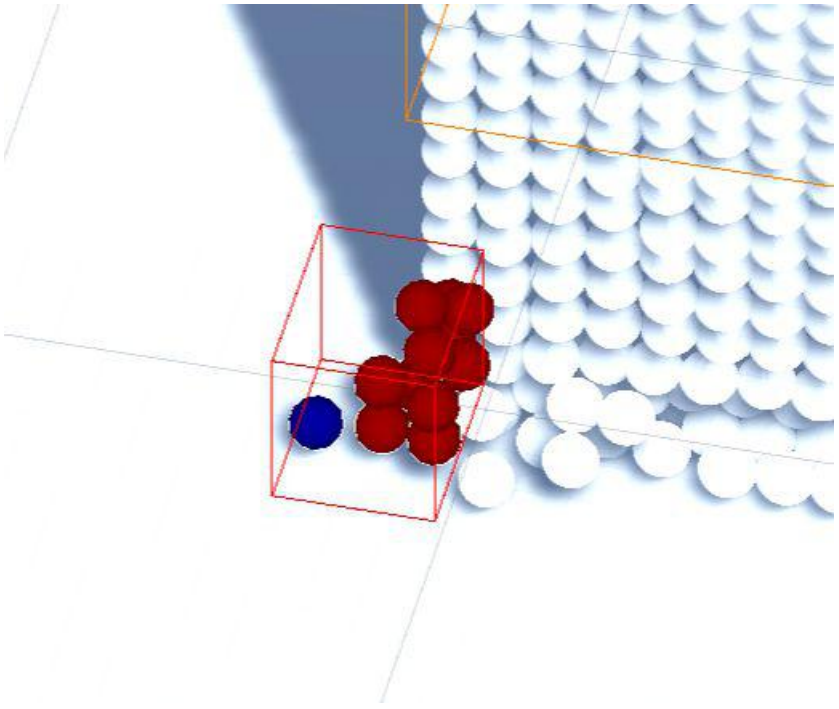


Fig 10: Tracking particle with neighbours in a cell.

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.

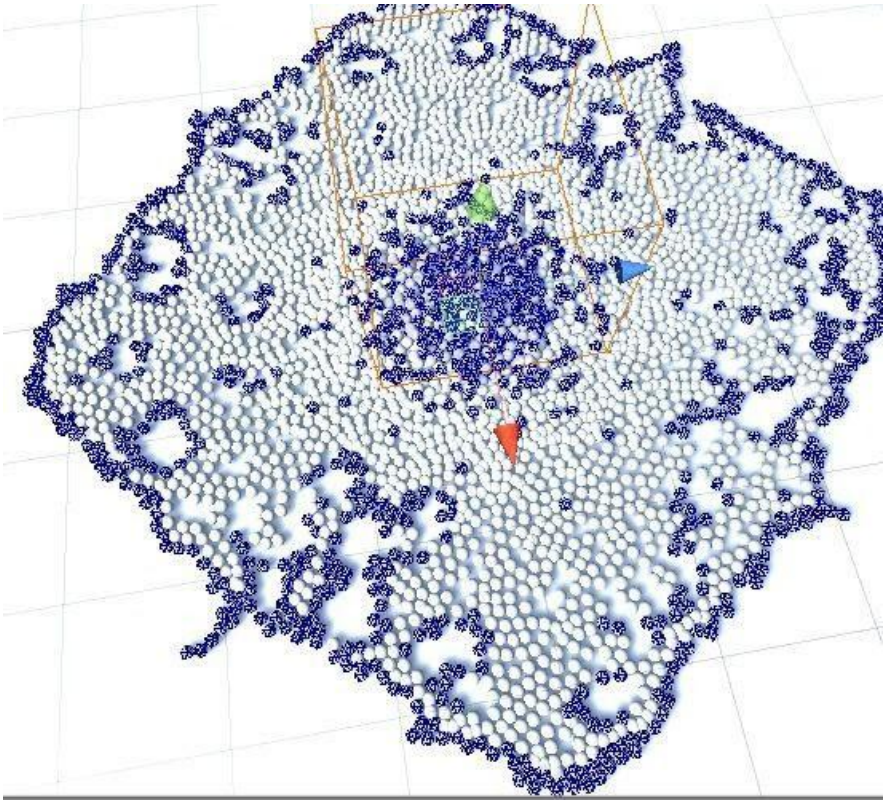


Fig 11: Surface particles are painted to blue colour.

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.

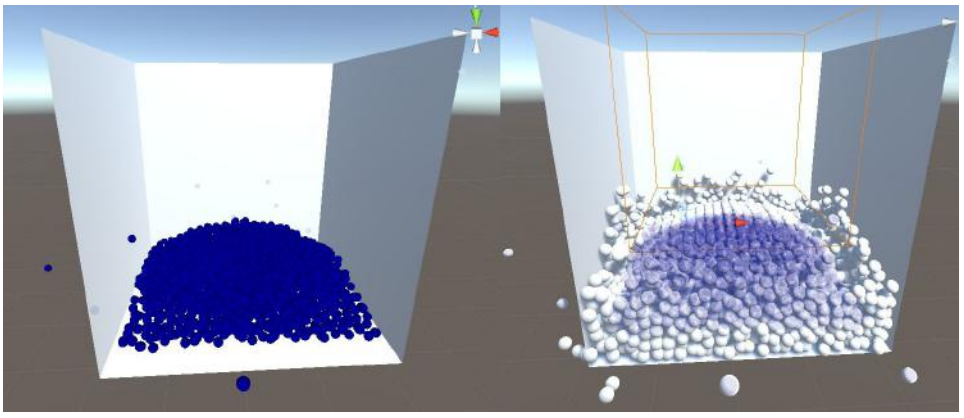


Fig 12: Inner particles are painted to blue and surface particles are painted to white.

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.

APPENDIX C: PRODUCT MANUAL

YAŞAR UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING

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

POF: Performance Optimized Fluid System

Product Manual

Revision 1.0
13.04.2020

By:

Baran Budak, Student ID: 15070001012

Cihanser Çalışkan, Student ID: 16070001020

İsmail Mekan, Student ID: 15070001048

Revision History

Revision	Date	Explanation
1.0	13.04.2020	Initial Product Manual

Table of Contents

List of Tables

List of Figures

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 this product manual is to document the implementation, testing, installation, and operation of the POF system as a software product.

The POF system is implemented and tested as it is described in Design Specification Document, Revision 2.0 [3], satisfying the requirements in POF system Requirements Specification Document, Revision 1.0.

Implementation, testing, and operation details are given in the following sections of this document.

2. POF Software Subsystem Implementation

This section describes the implementation of the POF system and its subsystems.

2.1. Source Code and Executable Organization

In this section we described our POF system organization and how it works by giving code snippets.

2.1.1 Handler Organization

In the Handler class, we have OnFlexUpdate function which is responsible for executing Flex. This function executed in each frame. Checks if flex actor and flex container assets are in the scene and working. Retrieving the particle data of simulation.

```
/// Get particle data when nFlex updated
void OnFlexUpdate(FlexContainer.ParticleData _particleData)
{
	if (m_actor && m_actor.container)
	{
		length = m_actor.indexCount;
		_particleData.GetParticles(m_actor.indices[0], 4096, _particles);

	}

 }
```

Figure 1: OnFlexUpdate function in Handler class.

The execution process of the POF system starts with the handler. NVIDIA Flex must be already initialized and various data such as particle position and AABB boundaries received from Flex. The handler transmits these data to relevant other classes. Hence, we can say that handler is a member of the controller part of the model-view-controller (MVC) system.

```
// Decide AABB
public Bounds GetBounds()
{
	Bounds b = new Bounds();
	Vector3 min = m_actor.bounds.min;
	Vector3 max = m_actor.bounds.max;

	min.x -= (m_actor.container.radius / 3);
	min.z -= (m_actor.container.radius / 3);
	min.y -= (m_actor.container.radius / 3);
	max.x += (m_actor.container.radius / 3);
	max.z += (m_actor.container.radius / 3);
	max.y += (m_actor.container.radius / 3);
	b.SetMinMax(min, max);
	return b;
}
```

Figure 2: GetBounds function in Handler class.

In the handler system, we use GetBounds function which we determine if AABB can interact with the environment. The function checks the three times of the AAABB radius of every dimension and decides that if AABB can interact with nearby cells.

```
// Control simulation in order
void Update()
{

	_vertexSystem.SetData(GetIndices(), GetParticles(), GetBounds(),
	(m_actor.container.radius*2) / 3, ref groups);

	_vertexSystem.GroupByCells();

	_surfaceRecognition.SetData(_particles, GetBounds(), ref groups,
	(m_actor.container.radius * 2) / 3);

	int[] particles = new int[1] { 0 };
	this.testDraw =  _surfaceRecognition.findBoundary();
	// _marchingCubes.StartSimu(testDraw)
}
```

Figure 3: Update function in Handler class.

The void update is a classic unity function when you open an empty script in the Unity project. Update functions operate in each frame repeatedly. We used this function to call our POF software and control it for each frame. To get into detail, update function receives data and retrieve it to the hash system by using the SetData function.

2.1.2 Hash System Organization

```
public void SetData(int[] indices, Vector4[] particles, Bounds bounds, float radius,
ref HashModel[] groups)
{
        // it comes from simuData class
       _indices = indices;
        _particles = particles;
        _bounds = bounds;
        _radius = radius;

        this._intervalx = (int)Math.Ceiling((_bounds.max.x - _bounds.min.x) /
       _radius); // x size
        this._intervaly = (int)Math.Ceiling((_bounds.max.y - _bounds.min.y) /
       _radius); // y size
        this._intervalz = (int)Math.Ceiling((_bounds.max.z - _bounds.min.z) /
       _radius);// z size

        groups = new HashModel[this._intervalx * this._intervaly * this._intervalz];
// decide hash size
        this._vertices = groups;
}
```

Figure 4: SetData function in Hash System class.

We created a hash system to analyze and find a particle in 3D space. Simply, it is 3D mapping to find particles and cells. FindID function does that operation in the hash system class. FindID function finds out the id of the cell that keeps specific particle inside.

In the SetData function, we specify the hash size by using grid size and AABB boundaries that how many cells can fit into AABB. That operation is only for one dimension. We must repeat calculations for three dimensions and set it.
However, it is not a smooth process as we discuss here because some particles are coinciding with the cell boundary and this is very normal. However, we cannot ignore coinciding particles by not considering calculations. Because it affects the computation of weight function and other important processes.

```
    // Find particle in which cells
    void findID(Vector4 particle, float _length, int _indice)
    {
        int cubeID;
        // 0 - 1 // 1 - 2 // 2 - 3
        int xId = (int)Math.Ceiling((particle.x - _bounds.min.x) / _length) - 1;
        int yId = (int)Math.Ceiling((_bounds.max.y - particle.y) / _length) - 1;
        int zId = (int)Math.Ceiling((particle.z - _bounds.min.z) / _length) - 1;
        float cubeX = (particle.x - _bounds.min.x) % _radius ;
        float cubeY = (_bounds.max.y - particle.y) % _radius;
        float cubeZ = (particle.z - _bounds.min.z) % _radius;

        // Five errors in here for fill to fix
        if (cubeX == 0 && cubeY == 0 && cubeZ == 0)
        {
            if (xId > 0 && yId > 0 && zId > 0)
            {
                cubeID = (xId--) + (this._intervalx * (yId--)) + (this._intervalx *
this._intervaly * (zId--));
                checkS(cubeID, _indice);
            }
            if (xId > 0 && yId > 0)
            {
                cubeID = (xId--) + (this._intervalx * (yId--)) + (this._intervalx *
this._intervaly * (zId));
                checkS(cubeID, _indice);
            }
            if (xId > 0 && zId > 0)
            {
                cubeID = (xId--) + (this._intervalx * (yId)) + (this._intervalx *
this._intervaly * (zId--));
                checkS(cubeID, _indice);
            }
            if (xId > 0)
            {
                cubeID = (xId--) + (this._intervalx * (yId)) + (this._intervalx *
this._intervaly * (zId));
                checkS(cubeID, _indice);
            }

            if (yId > 0 && zId > 0)
            {
                cubeID = (xId) + (this._intervalx * (yId--)) + (this._intervalx *
this._intervaly * (zId--));
                checkS(cubeID, _indice);
            }

            if (yId > 0)
            {
                cubeID = (xId) + (this._intervalx * (yId--)) + (this._intervalx *
this._intervaly * (zId));
                checkS(cubeID, _indice);
            }

            if (zId > 0)
            {
                cubeID = (xId) + (this._intervalx * (yId)) + (this._intervalx *
this._intervaly * (zId--));
                checkS(cubeID, _indice);
            }
```

Figure 5: FindID function in Hash System class.

Therefore, we used if-else checks in the hash system to find these special cases and apply a special solution to these particles. If particle coincides with a dimension of a cell boundary, we assign these particles to a previous cell id. Let's assume we have two neighbour cubes called cube number 1 and cube number2. There is only one particle coincides with the z dimension of boundaries for both cubes. Function assign the particle to the cube number 1. Additionally, an exceptional case, if cube id is zero we cannot assign a previous cell because it is already the beginning of an AABB so we just do not interfere with the cube id and include the particle into the first cube. The same situation is valid for the last cell of the AABB.

2.1.3 Computations Organization

In situational surface calculator class, we have operations that used in the hash system and surface recognizer classes and marching cubes scalar value calculator classes. This class is a subclass for calculating with given data and returns to that class. So, this class helps reduce method duplication. This is another performance improvement for the POF system compared to the first semester.

```
public float findDistance(Vector3 vertex, Vector3 point)
 {
     float xDimension = vertex.x - point.x;
     float yDimension = vertex.y - point.y;
     float zDimension = vertex.z - point.z;

     xDimension = (float)Math.Pow(xDimension, 2);
     yDimension = (float)Math.Pow(yDimension, 2);
     zDimension = (float)Math.Pow(zDimension, 2);


     return (float)Math.Sqrt(xDimension + yDimension + zDimension);
 }
```

Figure 6: FindDistance function in Situational Surface Calculator class.

In this class, we have FindDistance function that finds the distance between a point and a vertex. We assign the neighbour cells which gives us neighbour particles. This step required to apply kernel function which is used for determining if a particle is a surface particle.

```
public float findGradientWeight(Vector3 particle, Vector3 neighbour, float radius)
    {
        float statcons = findConstant(radius);
        float q = findDistance(particle, neighbour) / radius;
        float gradient = 0;
        if (0 <= q && q < 0.5)
        {
            gradient = (-2 * q) + (3 * q * q / 2);
        }
        else if (0.5 <= q && q <= 1)
        {
            gradient = (-1 / 2) * (float)Math.Pow((2 - q), 2);
        }
        else if (q > 1)
        {
            gradient = 0;
        }
        gradient *= (statcons / (float)Math.Pow(radius, 3));
        return gradient;
    }
```

Figure 7: FindGradientWeight function in Situational Surface Calculator class.

FindGradientWeight function is used for measuring the effect of the change on a particle by considering other particle's distances. If particles are too close, it affects more, the effect is getting weaker and after a certain distance, it does not affect.

```
    public float findKernel(float s)
    {
        float q=s;
        q = (float)Math.Pow(1 - Math.Pow(q , 2) , 3);
        return Math.Max(0 , q);
    }
```

Figure 8: FindKernel function in Situational Surface Calculator class.

FindKernel function is the implementation of a research paper, we decided the best suitable kernel function is this that we use now.

```
    public float[] findWeights(Vector3 vertex, Vector3[] particles, float[] weights, float
length)
    {
        float sum = 0;
        for (int j = 0; j < particles.Length; j++)
        {
            sum += findKernel(findDistance(vertex, particles[j]) / length);
        }
        for (int i = 0; i < particles.Length; i++)
        {
            weights[i] = findKernel(findDistance(vertex, particles[i]) / length) / sum;
        }
        return weights;
    }
```

Figure 9: FindWeights function in Situational Surface Calculator class.

FindWeights function calculates the weight for every particle. It calls the kernel function and sends distance data and divide to length then sum ups all values. After we calculate kernel

again the same way as we did before, but we divide this kernel value to the value that we find out by sum up. The function finds weights of particles with this method.

```
public Vector3 weightedPos(Vector3[] particles, float[] weights)
{
    Vector3 ret = new Vector3(0, 0, 0);
    for (int i = 0; i < particles.Length; i++)
    {
        ret += particles[i] * weights[i];
    }
    return ret;
}
```

Figure 10: WeightedPos function in Situational Surface Calculator class.

WeightedPos function multiplicates the particles with the weight and sums up then returns it.

In particle finder class, we have computation functions as well just like in the situational surface calculator.

```
public int FindID(Vector3 insideCell) // Particle finder'a koy.
{
    int cubeID;
    int _intervalx = (int)Math.Ceiling((_bounds.max.x - _bounds.min.x) / (_radius));
    int _intervaly = (int)Math.Ceiling((_bounds.max.y - _bounds.min.y) / (_radius));

    int xId = (int)Math.Ceiling((insideCell.x - _bounds.min.x) / (_radius));
    int yId = (int)Math.Ceiling((_bounds.max.y - insideCell.y) / (_radius));
    int zId = (int)Math.Ceiling((insideCell.z - _bounds.min.z) / (_radius));
    cubeID = (xId) + (_intervalx * (yId)) + (_intervalx * _intervaly * (zId));

    return cubeID;
}
```

Figure 11: FindID function in Particle Finder class.

We have FindID function in this class such as hash system, but it is used for a different purpose. It calculates the cell id of a particle.

```
public Bounds FindNeighbourArea(Vector4 particle)
    {
        Bounds insideCell = new Bounds();
        float xMax = particle.x + _radius * 4;
        if (xMax > _bounds.max.x)
        {
            xMax = _bounds.max.x;
        }

        float xMin = particle.x - _radius * 4;
        if (xMin < _bounds.min.x)
        {
            xMin = _bounds.min.x;
        }

        float yMax = particle.y + _radius * 4;
        if (yMax > _bounds.max.y)
        {
            yMax = _bounds.max.y;
        }

        float yMin = particle.y - _radius * 4;
        if (yMin < _bounds.min.y)
        {
            yMin = _bounds.min.y;
        }

        float zMax = particle.z + _radius * 4;
        if (zMax > _bounds.max.z)
        {
            zMax = _bounds.max.z;
        }

        float zMin = particle.z - _radius * 4;
        if (zMin < _bounds.min.z)
        {
            zMin = _bounds.min.z;
        }

        insideCell.SetMinMax(new Vector3(xMin, yMin, zMin), new Vector3(xMax, yMax,
zMax));

        return insideCell;
    }
```

Figure 12: FindNeighbourArea function in Particle Finder class.

In FindNeighbourArea function, we find the particle's cell and return with the boundaries. However, this process has exceptional cases. If particles are out of the volume of the AABB we assign these particles to the closest point of the AABB. If the particle's position in y dimension is smaller from the minimum boundary of the AABB, we assign particle to the minimum boundary of AABB. We do the same thing for the maximum boundaries. Same operation repeats for every dimension.

2.1.4 Surface Recognizer Organization

In surface recognizer class, we call the mathematical operations from the situational surface calculator class, so we do not need to explain these functions again. But while describing the main operation and organization system, we explain the logic and operations behind it.

```
    // Get surface particles and return it
    public int[] findNeighbourParticles()
    {

        int[] neigbourCells = { -1 };
        List<int> surfaceParticles = new List<int>();
        Bounds insideCell = new Bounds();
        for (int i = 0; i < 4096; i++)
        {

            insideCell = FindNeighbourArea(_particles[i]);

            neigbourCells = FindAreaCells(insideCell);
            bool isSurface = isSurfaceParticle(_particles[i],
FindNeigbourParticles(neigbourCells).ToArray());
            if (isSurface)
            {
                surfaceParticles.Add(i);
            }
        }
        this._ParticleNeighbound = insideCell;
        return surfaceParticles.ToArray();
        //return neigbourCells;
    }
```

Figure 13: FindID function in Surface Recognizer class.

FindNeighbourParticles function calls FindAreaCells function and adds these cell's particles to the list.

```
    public int[] FindAreaCells(Bounds insideCell)
    {
        int _intervalx = (int)Math.Ceiling((_bounds.max.x - _bounds.min.x) / _radius);
        int _intervaly = (int)Math.Ceiling((_bounds.max.y - _bounds.min.y) / _radius);
        int topLeftBackward = FindID(new Vector3(insideCell.min.x, insideCell.max.y,
insideCell.min.z));
        int topLeftForward = FindID(new Vector3(insideCell.min.x, insideCell.max.y,
insideCell.max.z));
        int topRightBackward = FindID(new Vector3(insideCell.max.x, insideCell.max.y,
insideCell.min.z));
        int bottomLeftBackward = FindID(new Vector3(insideCell.min.x,
insideCell.min.y, insideCell.min.z));

        int tx = (topRightBackward - topLeftBackward),
            ty = ((bottomLeftBackward - topLeftBackward) / _intervalx),
            tz = ((topLeftForward - topLeftBackward) / (_intervalx * _intervaly));

        int[] areaNums = Enumerable.Repeat(-1, ( (ty+1) * (tx+1) * (tz+1))
).ToArray();
        int i = 0, tempNum = topLeftBackward;

        for (int k = 0; k < ty; k++)
        {
            for (int j = 0; j < tz; j++)
            {
                for (int m = 0; m < tx; m++)
                {
                    areaNums[i] = tempNum + m;
                    i++;
                }
                tempNum += (_intervalx * _intervaly);
            }
            tempNum = areaNums[0] + (_intervalx * (k + 1));
        }
        return areaNums;
    }
```

Figure 14: FindAreaCells function in Surface Recognizer class.

In FindAreaCells function, we find all neighbour cells of a cell in a specific range by using cell grid value and position value. So that way by using spatial hashing we obtain particles through the cells. It resembles an encrypt-decrypt system.

```
    public bool isSurfaceParticle(Vector3 centerParticle, Vector3[] neighbourParticles)
    {
        float returnVal = FindDistance(centerParticle, FindWeigtedX(centerParticle,
neighbourParticles, _radius * 4));

        if (returnVal < 0.035)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

Figure 15: isSurfaceParticle function in Surface Recognizer class.

The isSurfaceParticle finds the if a particle is a surface particle or not. Issurfaceparticle function calls the findDistance which we already know its operation and it calls another function named findWeightedX. It is an implementation of a specific formula which is explained in a research paper. This findWeightedX function gives us the centre of mass that a particle creates with its neighbours as a vector.

2.1.5 Marching Scalar Value Organization

Zhu Bridson [8] is a research paper named "Animating Sand as a Fluid". Zhu and Bridson offer a different approach to calculate the scalar value of vertices outside of the fluid. In other words, vertices of surface particles. This value is used in marching cubes for visualizing step. The formula is given in the paper and we applied it to our POF software.
Formula implementations look similar compared to surface methods. However, small sub calculations and kernel functions are the same, but the main mathematical operations are different. It is a different structure but with the same pieces. This implementation changes the perception of the surface. The surface becomes smoother.

The weight function is below:

$$w_i = \frac{k(|x - x_i|/R)}{\sum_j K(|x - x_j|/R)} \qquad (1)$$

2.1.6 Explanation of Kernels

Kernel function: A kernel function is a comparison function that given two values. These values can be in any form.
In our project, a kernel function is obligatory for detecting the value approximation precisely.

Kernel function formula:

$$k(s) = \max(0, (1 - s^2)^3) \qquad (2)$$

2.2. Software Development Tools

In this following section, we describe software tools that we have used in the POF system project.

2.2.1 Unity

Unity game engine is used in our project as a visual tool for testing and implementation. Various programs can be used such as Unreal Engine 4. We used Unity because learning speed is faster compared to the Unreal Engine (Unity GUI is relatively easier for us). Also, because some of the members in our team have experience with Unity engine, our advisor suggested us to use the Unity engine. Conceivable reasons, we determined to use Unity in our project.

Details: Unity 3D version 2018.3.11 (29 March 2019), Unity Technologies.

2.2.2 Visual Studio 2017

Visual Studio is an integrated development environment (ide). We write our code in C# by using Visual Studio. The reason we use Visual Studio is we are developing project in Windows operating system and Unity has Visual Studio support. You can import Unity library to Visual Studio.

Details: Visual Studio 2017 v15.9.15 (13 August 2019), Microsoft.

2.2.3 Github

GitHub, Inc. is a company that provides hosting for software development version control using Git. We used Github in our project for storing safely. Keeping our data in local is not an efficient way and it confuses version order. Besides, the importance of tools as Github vastly shows its importance in telecommuting.

Details: GitHub Inc., Subsidiary to Microsoft.

2.2.4 Gitkraken

GitKraken is another Git GUI client is used from developers to increase productivity. It has the same operation as Github. However, Gitkraken has a reasonable advantage when it comes to code handling. Gitkraken shows the changing parts of the code and it makes easier to reduce confusions and accelerates the project speed.

Details: Gitkraken, Axosoft.

2.2.5 NVIDIA FleX

We used NfleX as a third-party software which serves us to the purpose of having and initialization of particle-based fluid simulation. As we emphasized in the final report [1], since creating a particle-based fluid simulation is another complex thesis topic,
we aim to improve both visualization and performance as much as we can in a research paper implementation manner. We do it by using already existed particle-based fluid simulation.

Flex used in as an asset for Unity. The software operates on Windows or Linux, but it operates on windows in our project. It can be executed on Unity or Unreal Engine 4 platforms, but we use unity for the reasons that we mentioned before.

NVIDIA FleX Requirements:

- Windows 7 (64-bit) or newer.
- DX11 or CUDA capable graphics card
- Unity 2017.3 or later version

Details: NVIDIA FleX v1.0 (19 July 2018), NVIDIA company.

2.3. Hardware and System Software Platform

The minimum specification requirements are listed below:

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 1: Minimum Requirements of software Platform

You can see the system that we used while developing the POF system in Final Report revision 1.0 [1].

3. POF Software Testing

This section describes the POF system testing stages with screenshots as proof of that testing results are correct. We also explained in Design Specifications Document revision 2.0 (DSD 2.0) [3] as more detailed.
Since this section aims to describe how testing is designed and confirmed that the POF system works properly, we do not necessarily need to narrate how we designed and implemented the hash system but you can learn more details from Final Report revision 1.0 [1] or you can check on Design Specifications Document revision 2.0 [3].

We used the bottom-up integration test technique, which has a gradual structure in the testing process of our project. As a result, the POF software works correctly. However, we assumed as security issues and unity restrictions are excluded for obvious reasons such as project deadline. Expediently to a bottom-up testing approach, we tested smaller components are tested first and build the project for the bigger components step by step. The testing process repeats until the component at the top of the hierarchy is tested. Otherwise, POF project testing would be too difficult to apply since our project has multiple large components. Only NVIDIA Flex library is huge by its own.

3.1 Testing of Hash System

The very first step of the POF system testing is checking three-dimensional hashing functionality. We first tested whether the Hash System was working properly. The purpose of the Hashing system is to increase performance by reaching the particles faster.
We tested on the Unity platform our software. We used unity library to handling testing processes. Our project is heavily research-based for that matter, it involves abstract concepts.
We tried visual elements to see test results in the scene mode of the Unity platform. We coloured specific particles or draw certain shapes or printed various things as calculations or particle attributes to the unity log console.

We selected a random particle index number in our particle set and painted the selected particle to blue colour on gizmos.
We coloured neighbour particles to red colour and draw a red wire cube called a cell in our project. White particles mean they are in neutral form and offset the colour of the particles. You can see the description mentioned colouring and red wire cube from the figure down below.

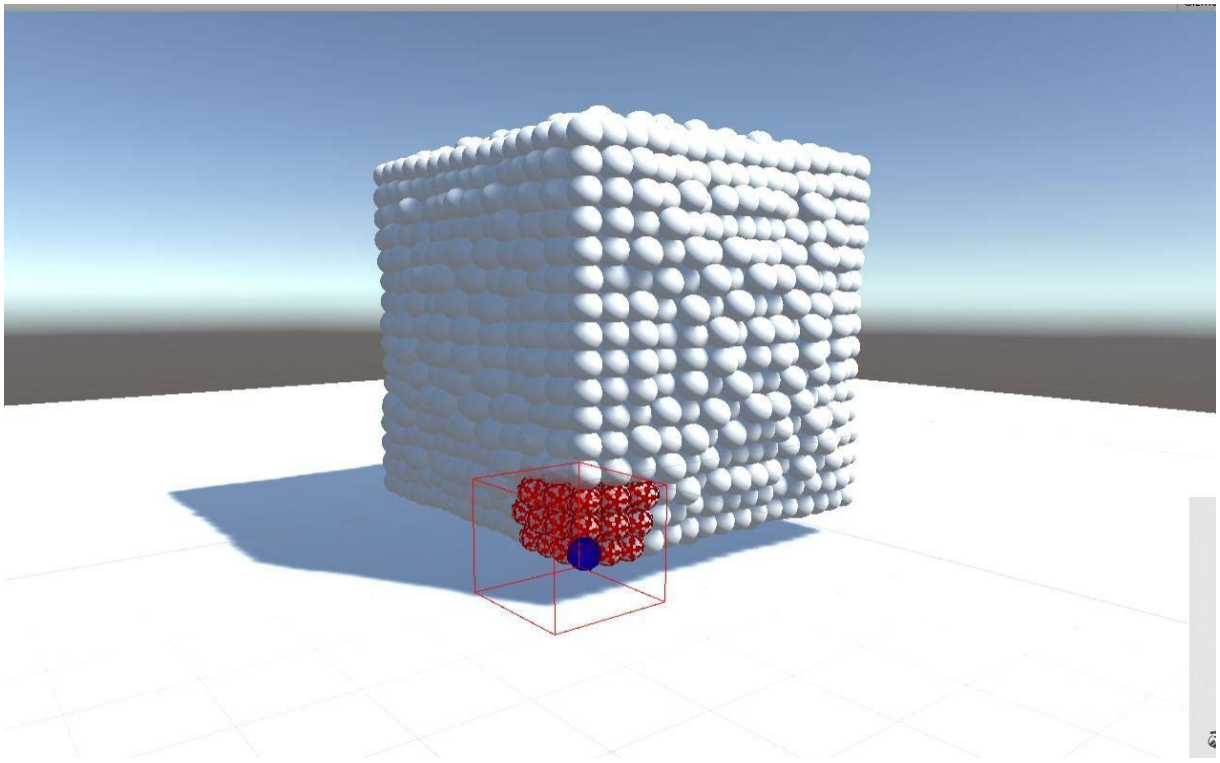


Figure 16: Particle and its neighbours in the cell.

Particles start to move when pressed to play in game mode. It is obliged to track the selected particle and neighbour particles in a range. We find neighbour particles from the particle's cell which is drawn in a red cube that edges drawn only as shown in the figure below. As frames pass, particles travel in the scene. Selected particle colour does not change, and we can print its location from the console in each frame.

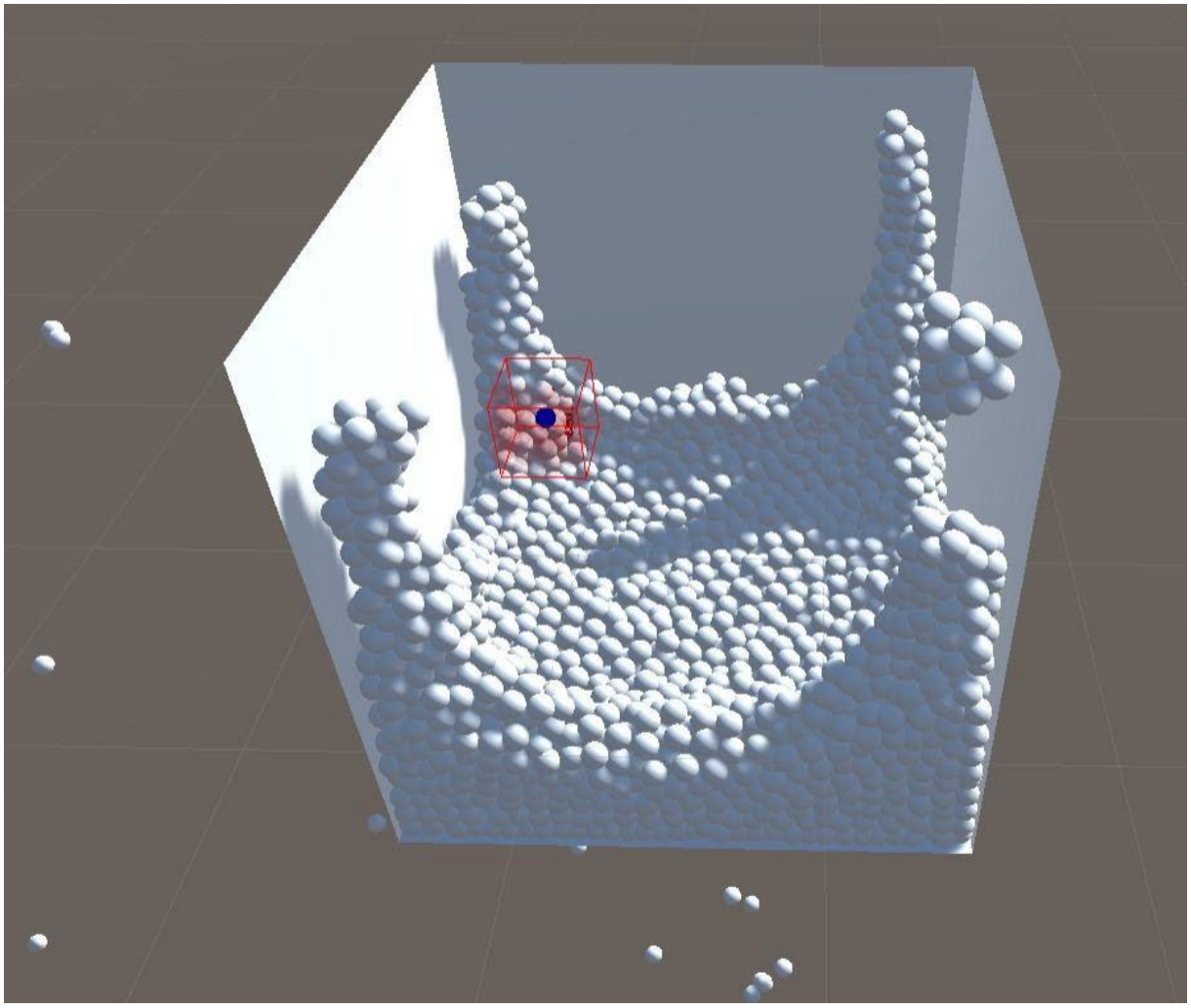


Figure 17: Tracking particle in a cell.

We managed to track any particle in the scene. We just need a particle index. Tracking the neighbour particles and its cell is as important as tracking particle itself. In essence, all we do is finding the particle and analysing the environment for every particle. We also proved the hash system works correctly by calculating the results ourselves by hand.

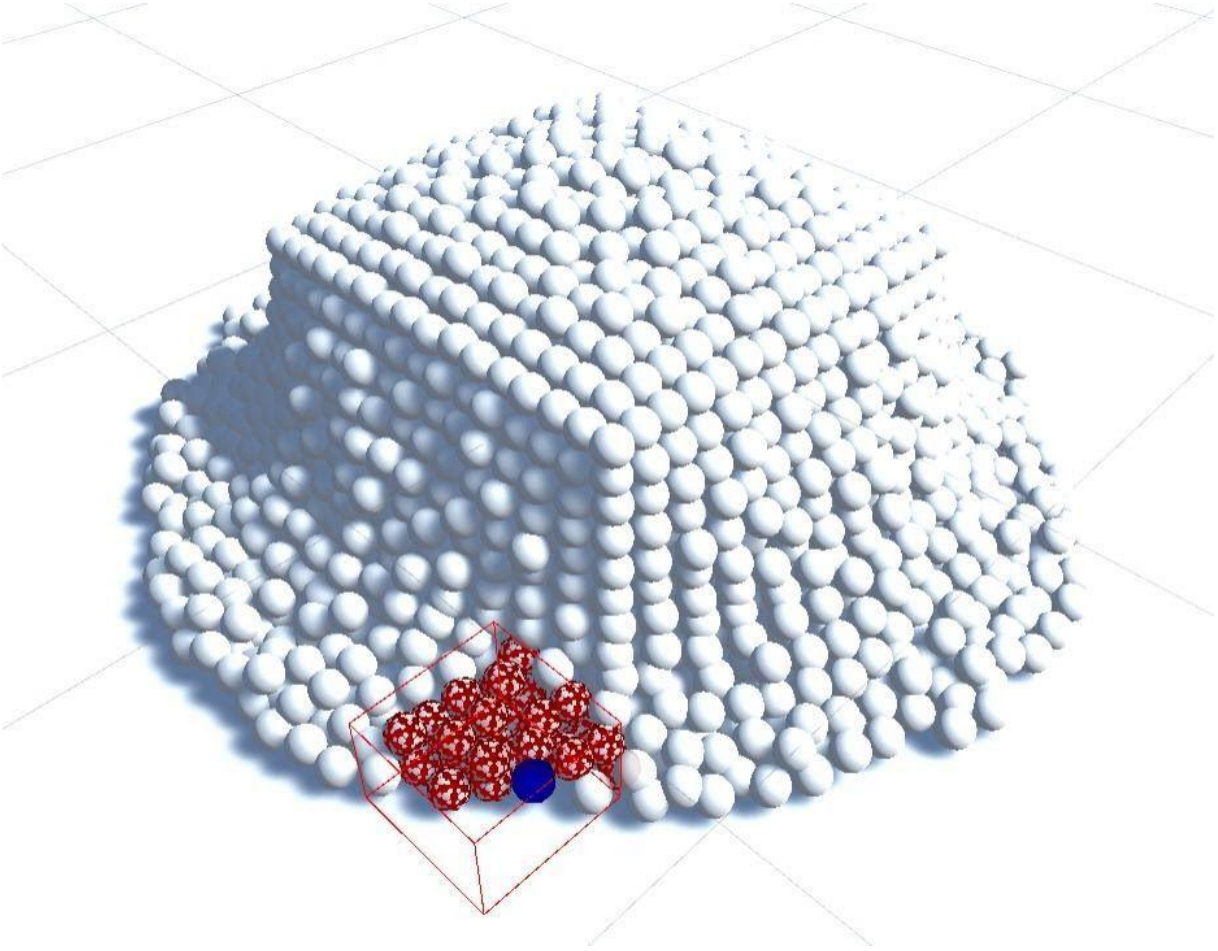


Figure 18: Tracking particle and its neighbours in a cell.

3.2 Testing of Surface Recognizer

The objective of surface recognizer is to find surface particles and other relevant information such as neighbour particles and neighbour cells. To apply the next step algorithms, we must find surface particles by using surface recognizer. Instead of dealing with and controlling all particles one by one, we only make operations on the surface particles because surface particles play a crucial role in our POF software. Therefore, makes the POF system faster and efficient. We can determine the surface particle by calculating weight with mathematical operations. We find certain scalar value. If the weight value is less than 0.035, we can say that the particle is on the surface. Otherwise, the weight is bigger than the value and we can understand that particle is on the surface.

We implemented certain algorithms in the mentioned research papers. We can find any particle and our hash system works very well. The next step is finding and tracking surface particles. We implemented the script code in our project, and we can find the surface particles. We coloured to blue inner particles that is not a surface particle. As you can see from the figure below, we highlighted inner particles with blue and compared with all particles. White particles are surface particles. Surface particles can be tracked in each frame.

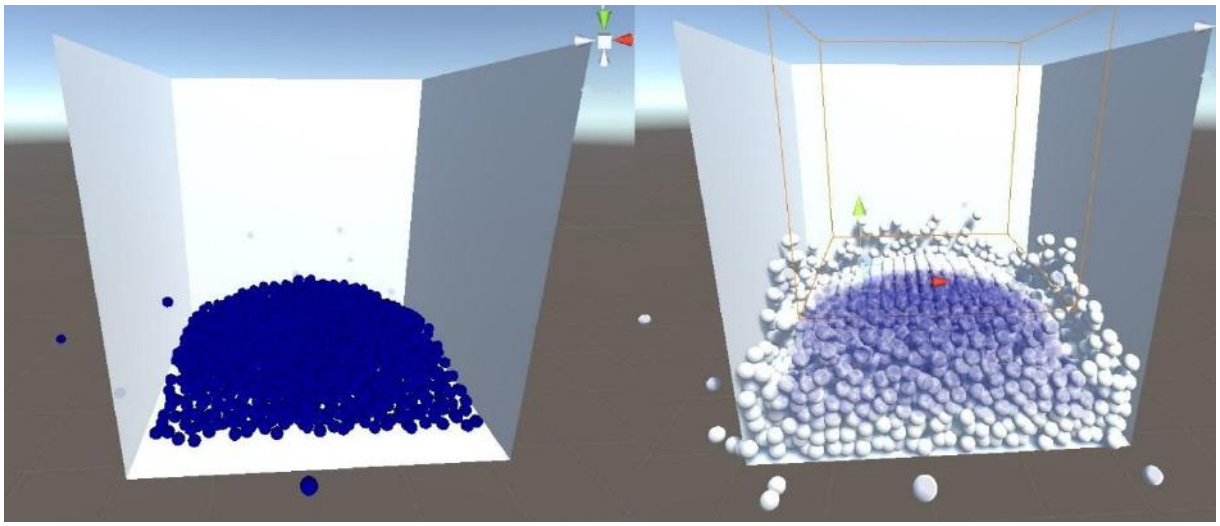


Figure 19: Inside blue particles vs all particles.

4. POF System Installation, Configuration and Operation

This section describes how user installs, configures, and operates the POF system with its environment. Firstly, the user must have the required system mentioned before. Secondly, the user should install and set up the Unity Platform. Thirdly, the user imports the POF system as a custom package and starts the sample scene. Lastly, the user can learn description attributes of the system in the last section.

4.1 Unity installation and set up

We assume that a normal user has the POF system digital copy and the user meets all requirements mentioned. User should download unity hub by clicking the button from the unity website [9] as shown in the figure.

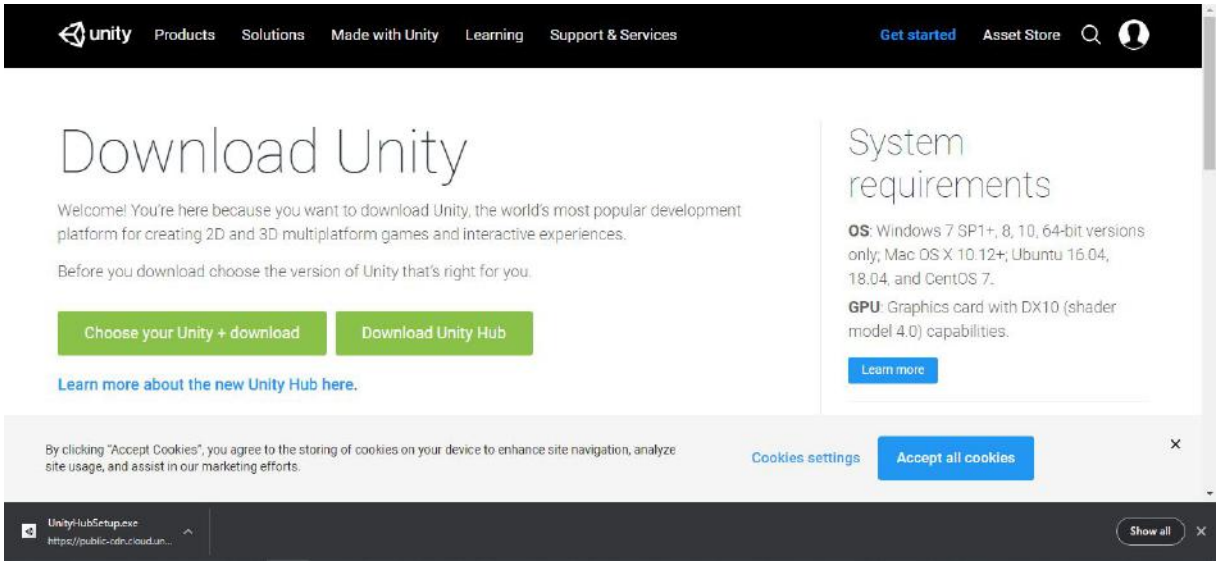


Figure 20: Download Unity Hub from website.

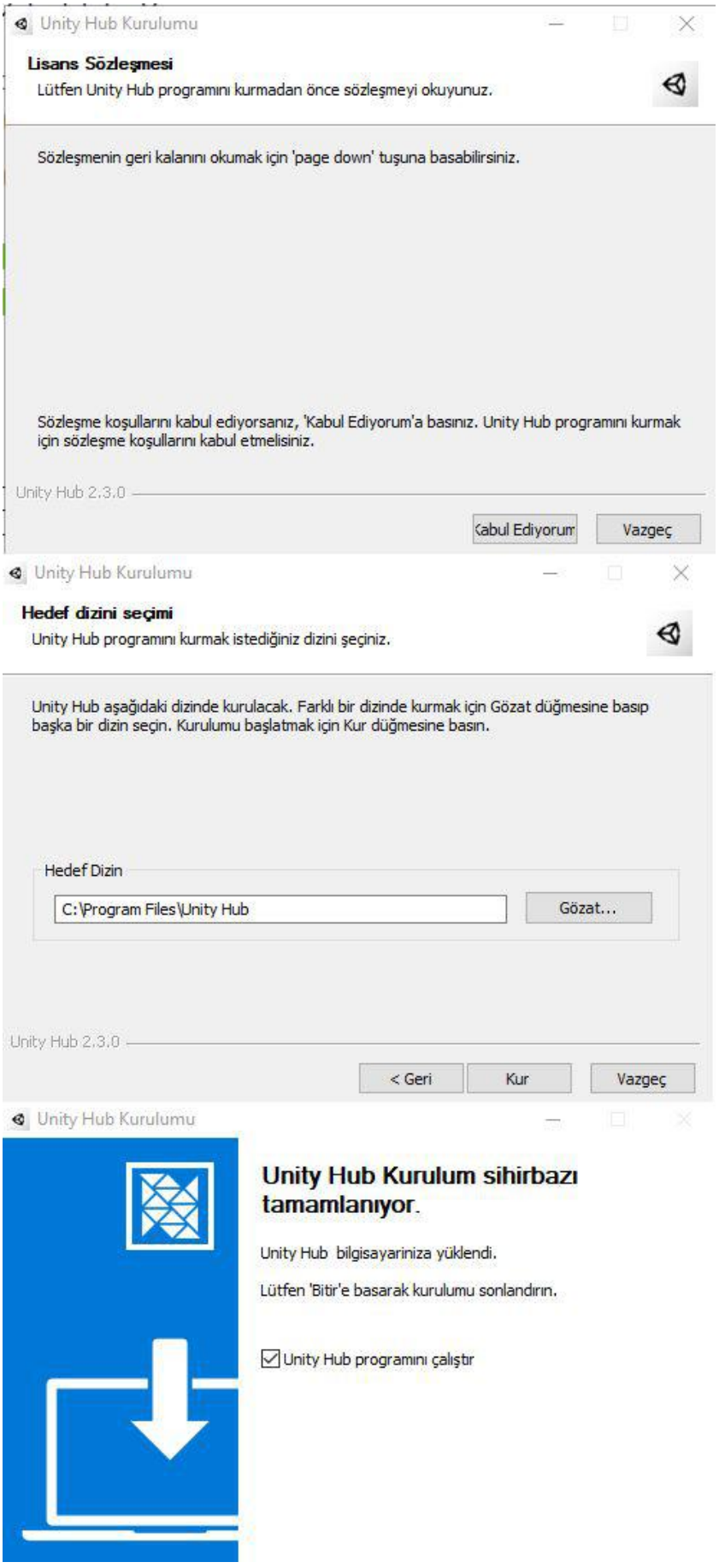


Figure 21: Install Unity Hub program.

Run Exe file when the .exe file downloaded to your computer; installation window will appear as shown in the figure above. Click I agree and select where unity hub will be installed and click set up. Wait a couple of minutes for the installation to finish and then start unity hub program.

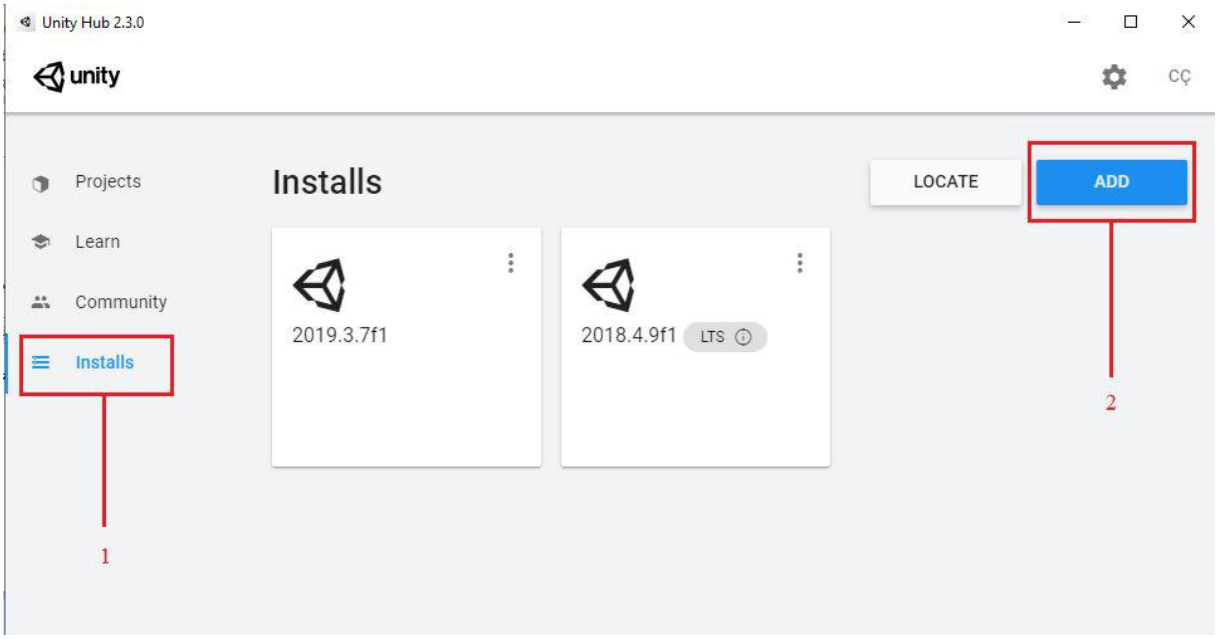


Figure 22: Go to installations window in Unity Hub.

When Unity Hub opened, click installs from the bar on the left side on the window which is shown in a red square with marked as number step number one. Your installed file should be empty if you never installed unity before. Also, it gives the information of which versions of the Unity is installed. If you do not have the minimum required Unity version, you should click add button as shown in the figure with a red square as step number two.

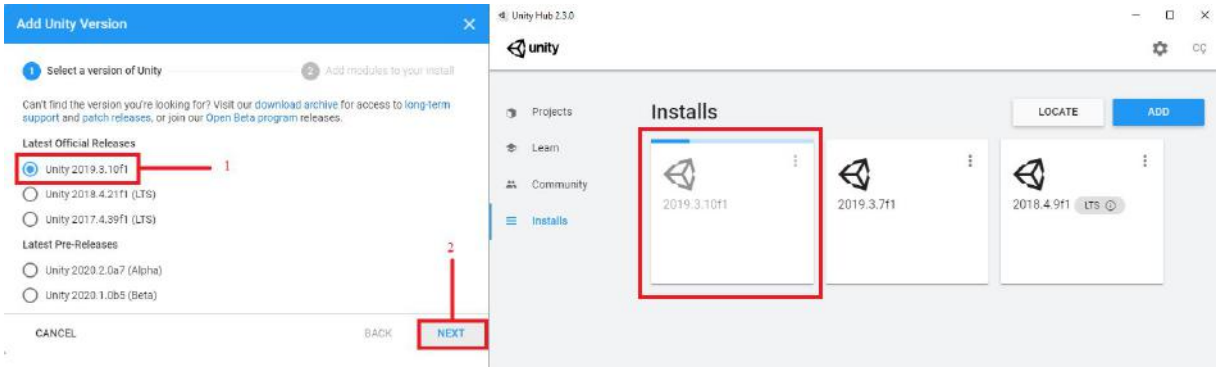


Figure 23: Select Unity version and install.

Select the Unity version as shown in the figure with red square number one. Click next, in the second picture you see that Unity version is installing. When the download has finished, the user must create a new empty 3D project.

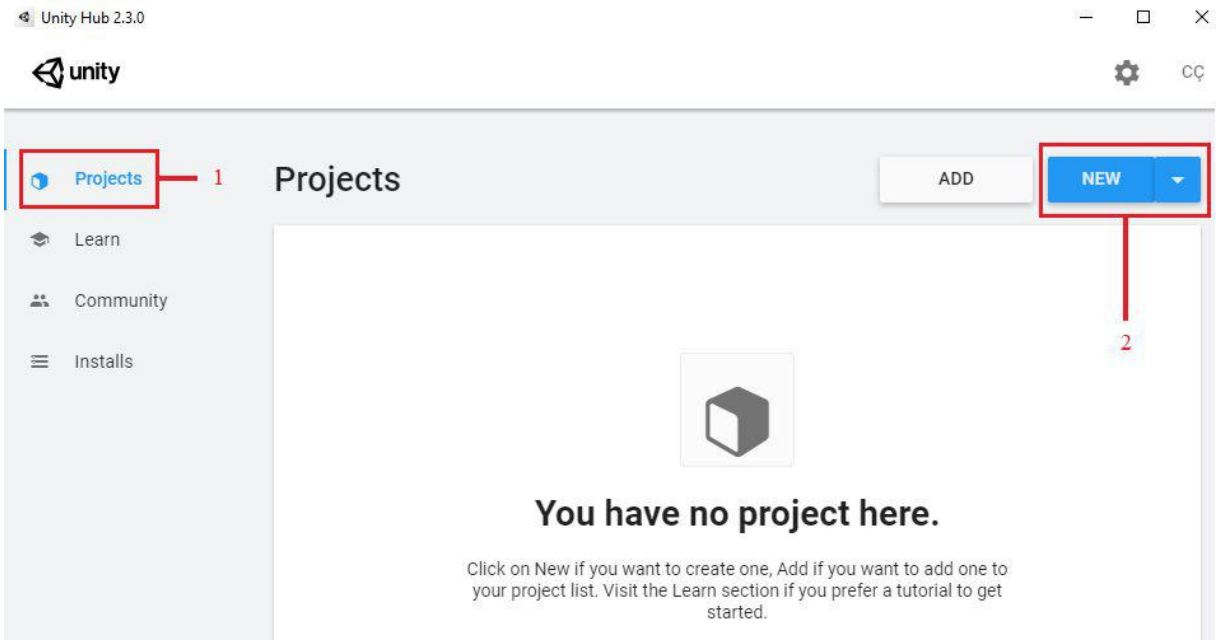


Figure 24: Create a new empty project in Unity.

Click the projects button which is highlighted with red square number one. Then, click the new button. A new window will open.

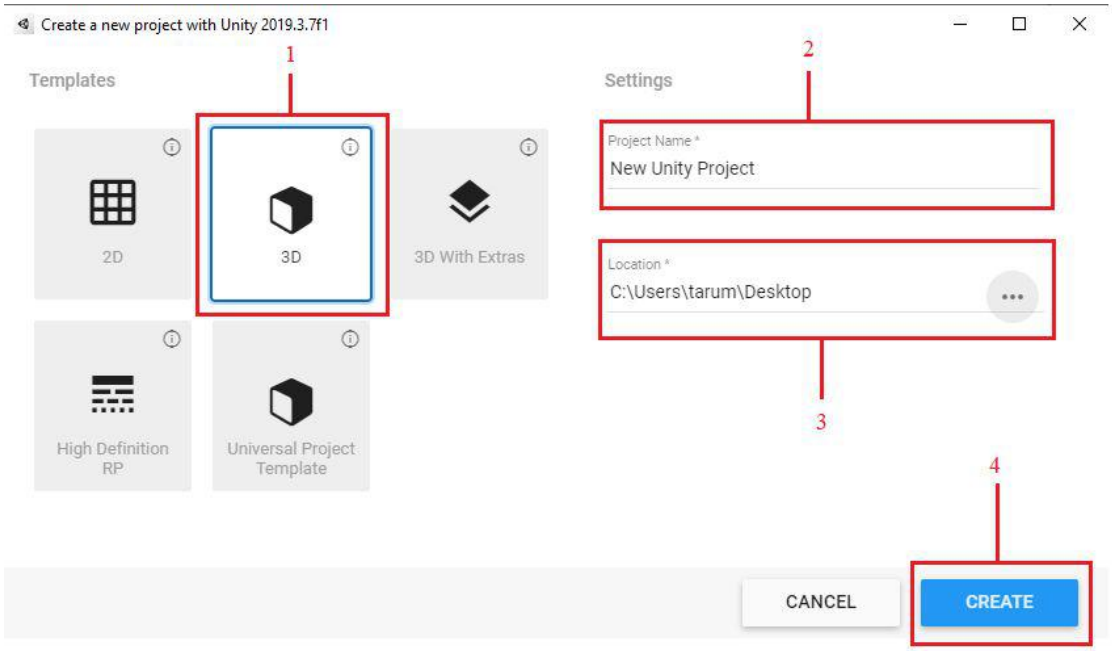


Figure 25: New Unity project configurations.

In this window, firstly select 3D from the template menu. Then edit the project name and edit the project location. After you have finished, click create. New empty unity project will be open.

4.2 POF installation and set up

Now, you can import the POF system. Click assets from the opening menu and please click import package. Click the custom package as shown in the figure.

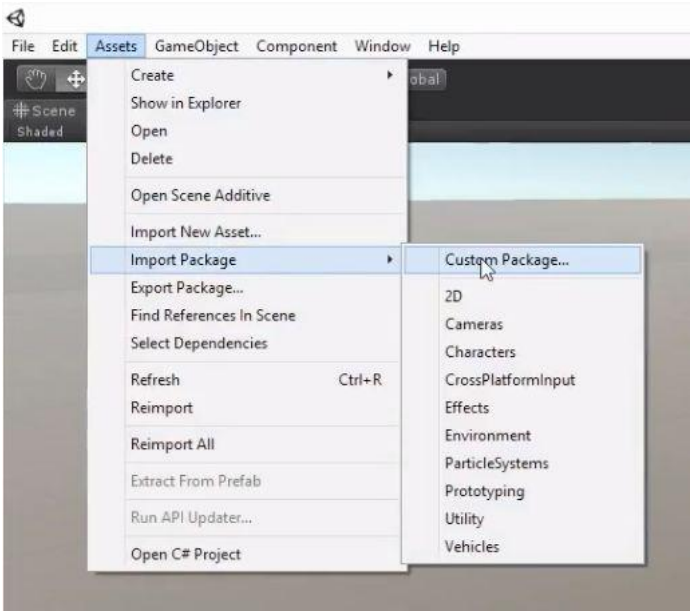


Figure 26: Select the custom package in Unity.

Find POF unity package file location in your computer and select.

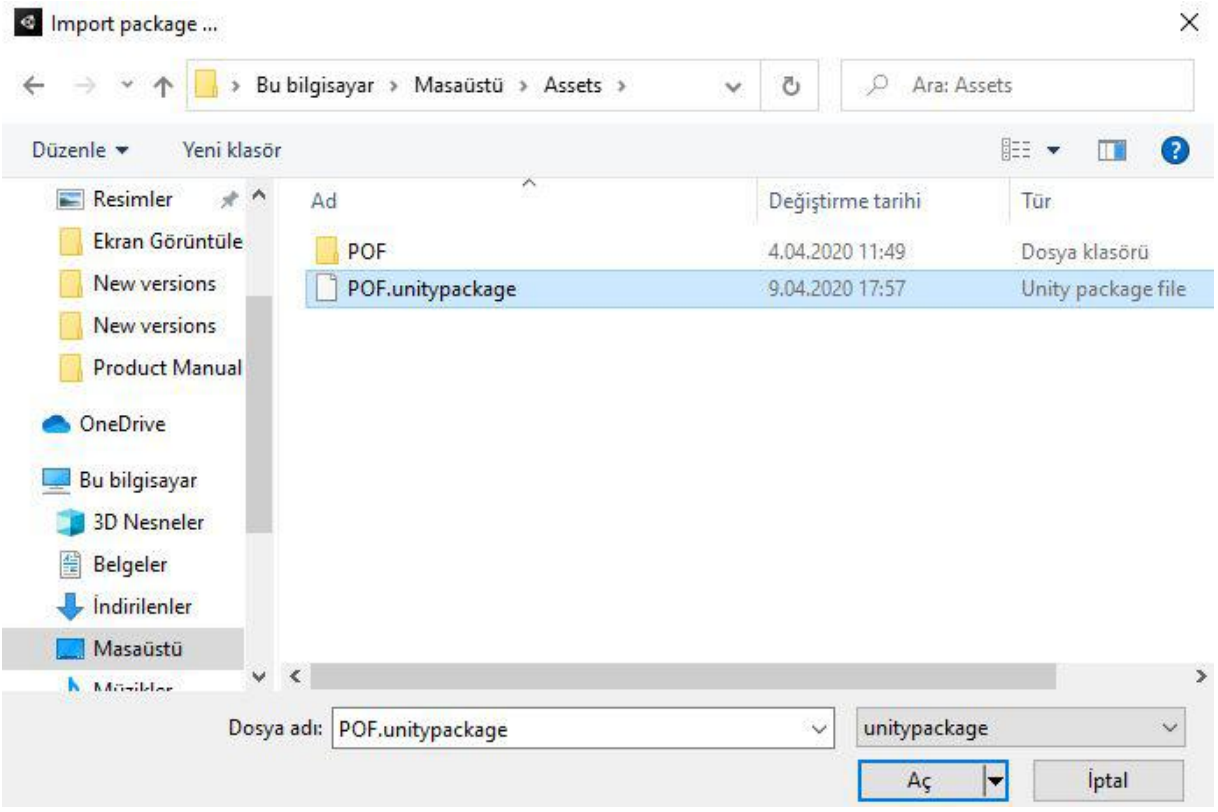


Figure 27: Find POF unity package in file explorer.

A new window will open as a next step. You must not change the selected files to prevent encountering any problem. Please click import. You must see folders as shown in the figure below.

Figure 28: Imported Package files in assets folder.

Double click to sample scene and press play button to start the simulation. When you have entered the scene, your screen should look like as shown in the figure below. You can rotate the camera in the scene by holding right click of the mouse. W button goes forward. S button goes backwards. A button goes left, and the D button goes right direction in the scene.

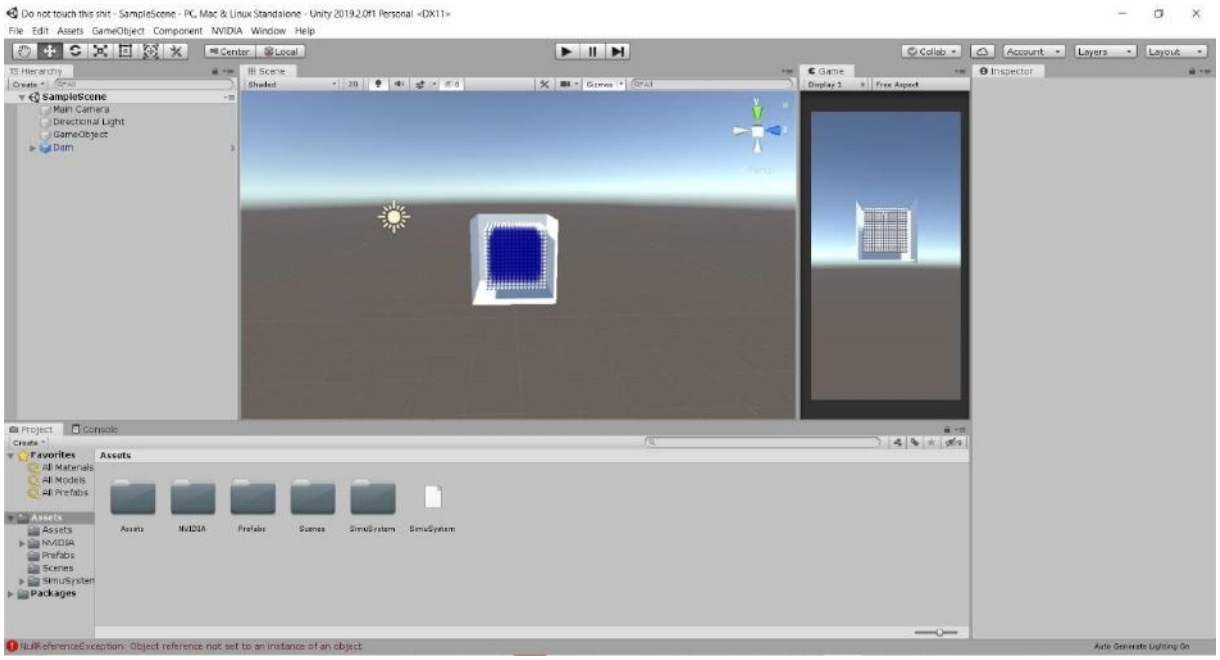


Figure 29: Sample scene in the Unity.

In case you can not run POF software on your computer, we have shared more screenshots of the POF software while it is working on a computer.

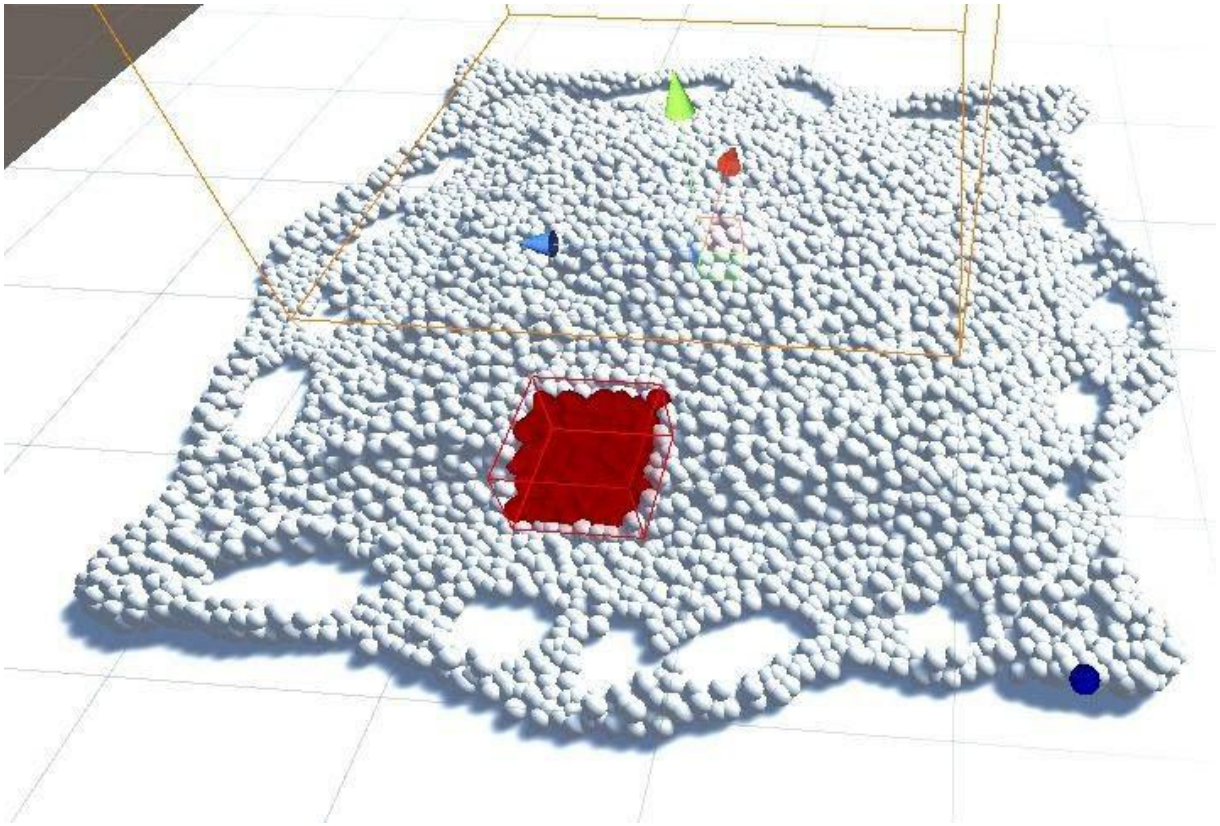


Figure 30: Screenshots of the scene mode while simulation is working-1.

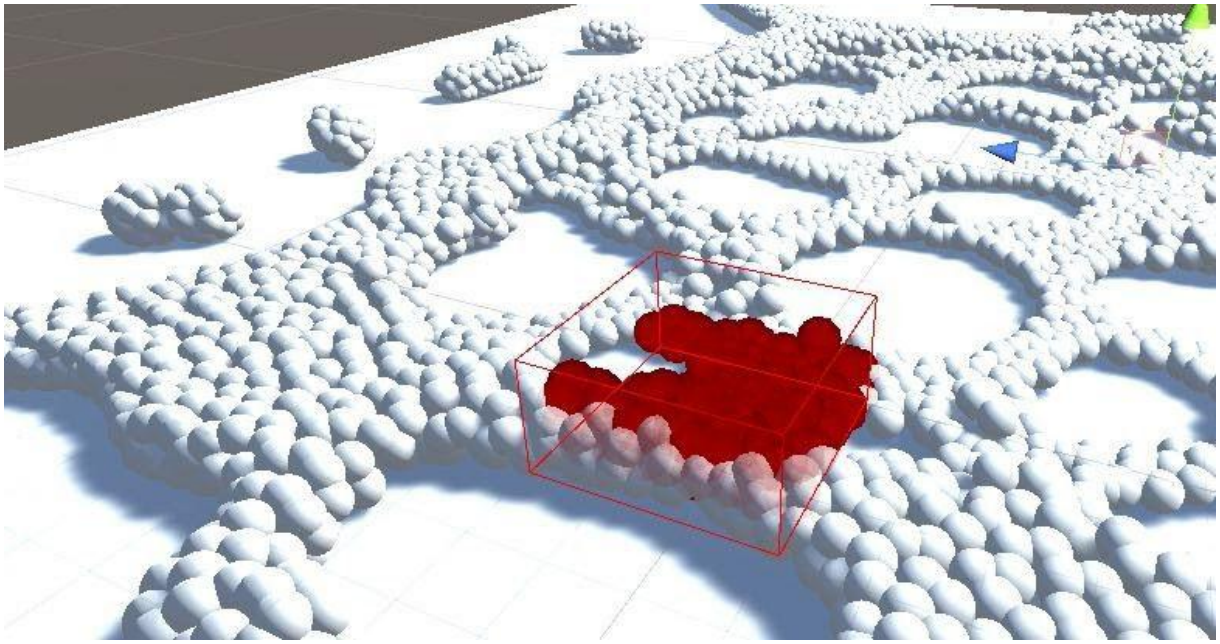


Figure 31: Screenshots of the scene mode while the simulation is working-2.

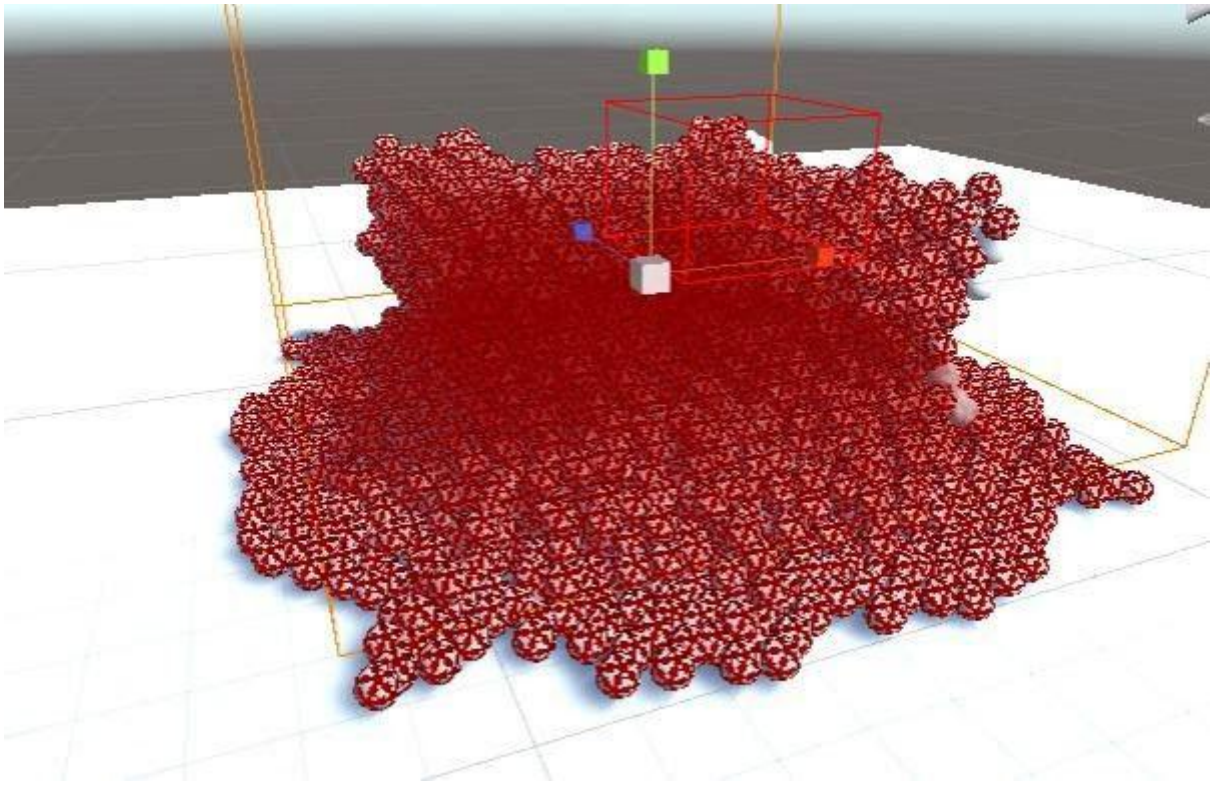


Figure 32: Screenshots of the scene mode while the simulation is working-3.

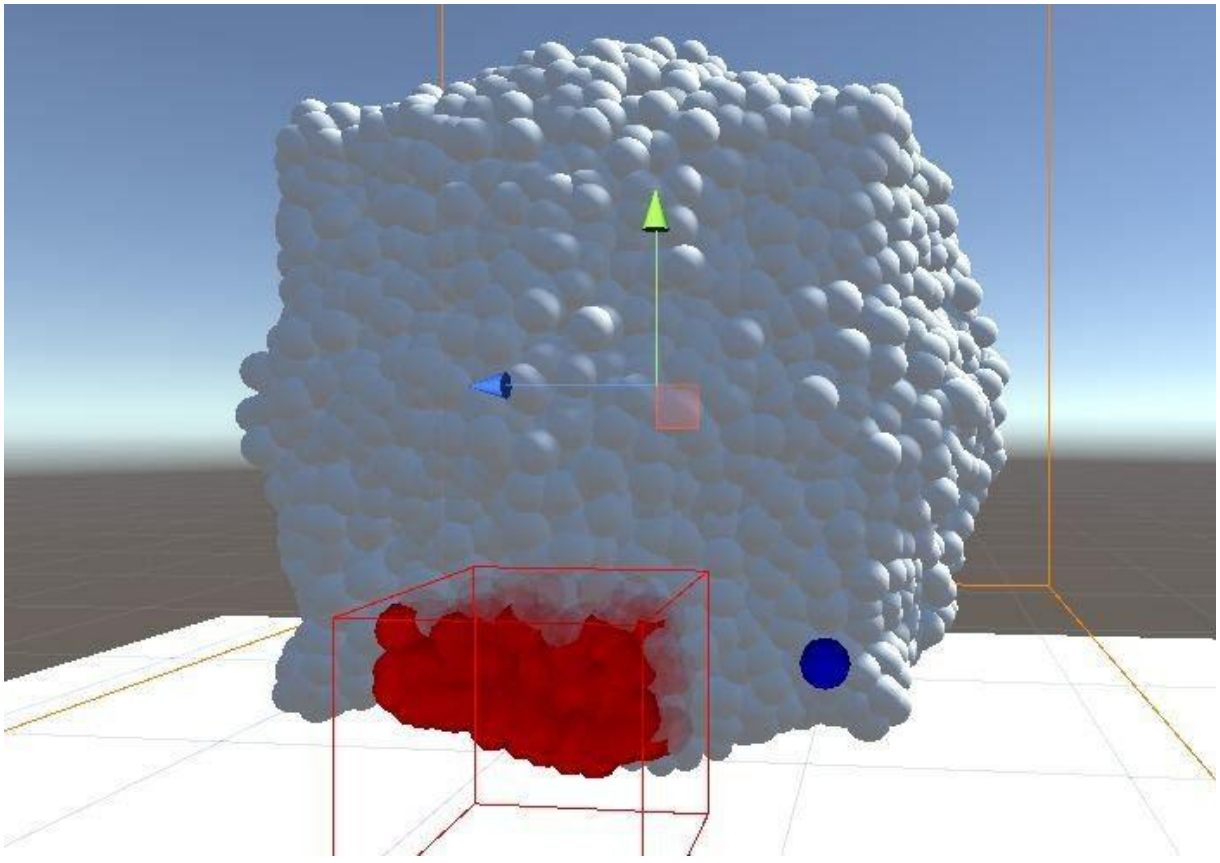


Figure 33: Screenshots of the scene mode while the simulation is working-4.

4.3 Learning the basics of POF

Welcome to the POF system! In this section, we describe the parameters and configurations that the user can change from the inspector menu. The other unity settings such as lighting etc. are irrelevant and depends on the user.

4.3.1 NVIDIA Flex inspector settings

In this section, we explained how to change the settings of Flex from the inspector menu. Inspector menu is a GUI that makes easier to reach attributes in the C# script.

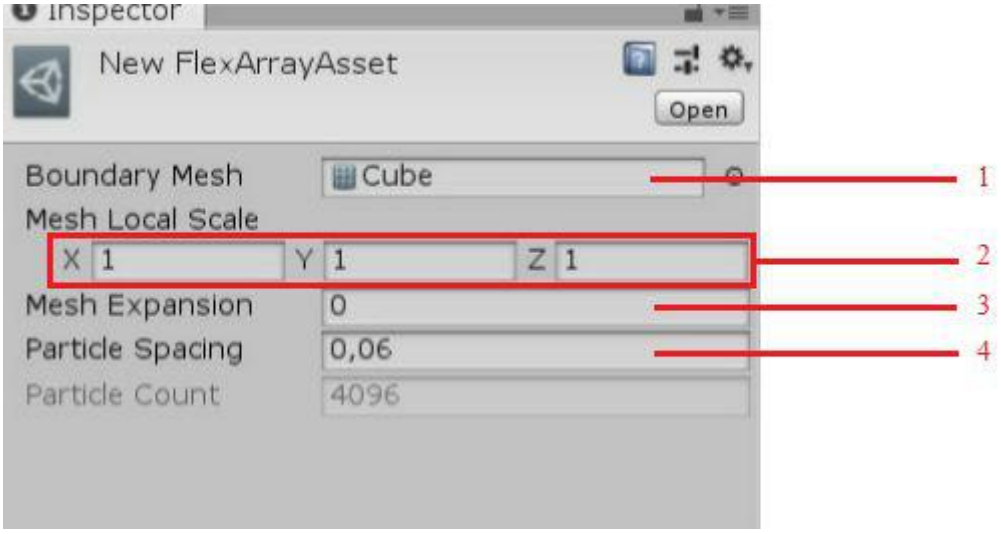


Figure 34: Description of Flex array asset parameters.

1. Boundary Mesh: You can select the boundary mesh from this row. If you click the image circle with the dot, a new window opens, and you can assign the boundary mesh. Boundary mesh restricts the particles. It is a kind of storage space for the particles.
2. Mesh Local Scale: This button scales the boundary mesh. You can scale it for every dimension separately.
3. Mesh Expansion: Click this value to change the AABB mesh boundary.
4. Particle Spacing: This row determines the distance between particles. It aligns the particles by considering this spacing distance. Affects the particle number indirectly.

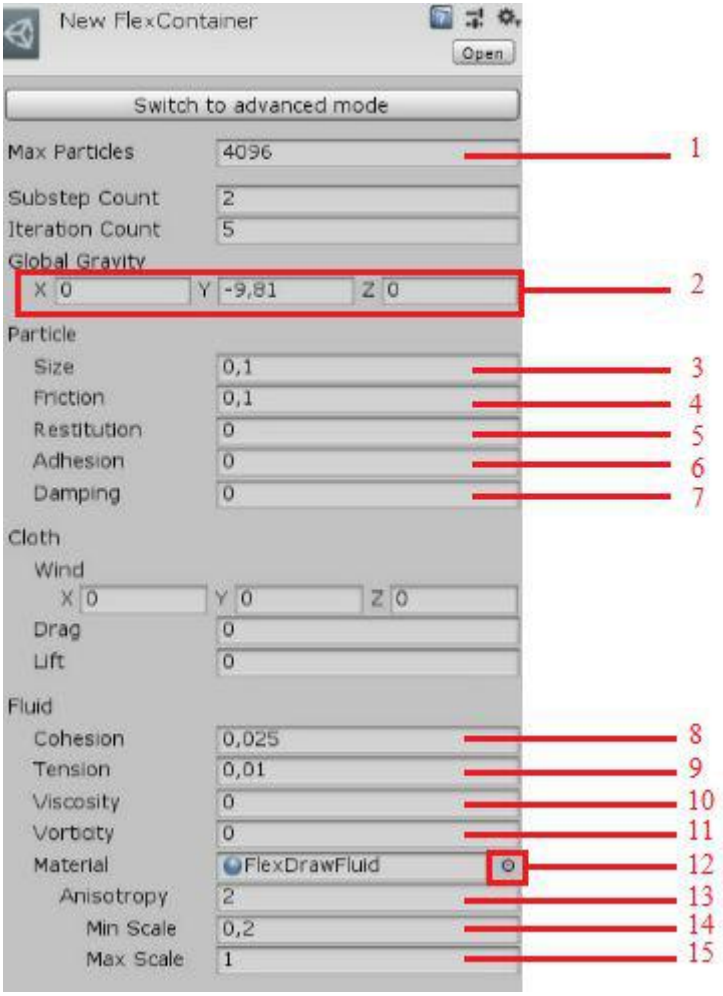


Figure 35: Description of Flex Container parameters.

1. Max Particles: You can select the maximum particle number in your scene.
2. Global Gravity: You can change gravity direction or magnitude.
3. Particle Size: Click this value to change particle size.
4. Particle Friction: Click this value to change friction between the particle and another surface.
5. Particle Restitution: You can change particle restitution by clicking this value.
6. Particle Adhesion: You can change particle adhesion by clicking this value.
7. Particle Damping: Click this value to change damping of particles.
8. Fluid Cohesion: You can change fluid cohesion value from this row.
9. Fluid Tension: Click this value to change fluid tension force.
10. Fluid Viscosity: Click this value to change the viscosity of a fluid.
11. Fluid Vorticity: You can change fluid vorticity by clicking this row.
12. Fluid Material: Click this button to select fluid material from a new window.
13. Anisotropy: You can change the anisotropy by clicking this value.
14. Min Scale: You can change the minimum scale of the anisotropy value.
15. Max Scale: You can change the maximum scale of the anisotropy value.

4.3.2 POF inspector settings

In this section, we explain POF settings by using an inspector.

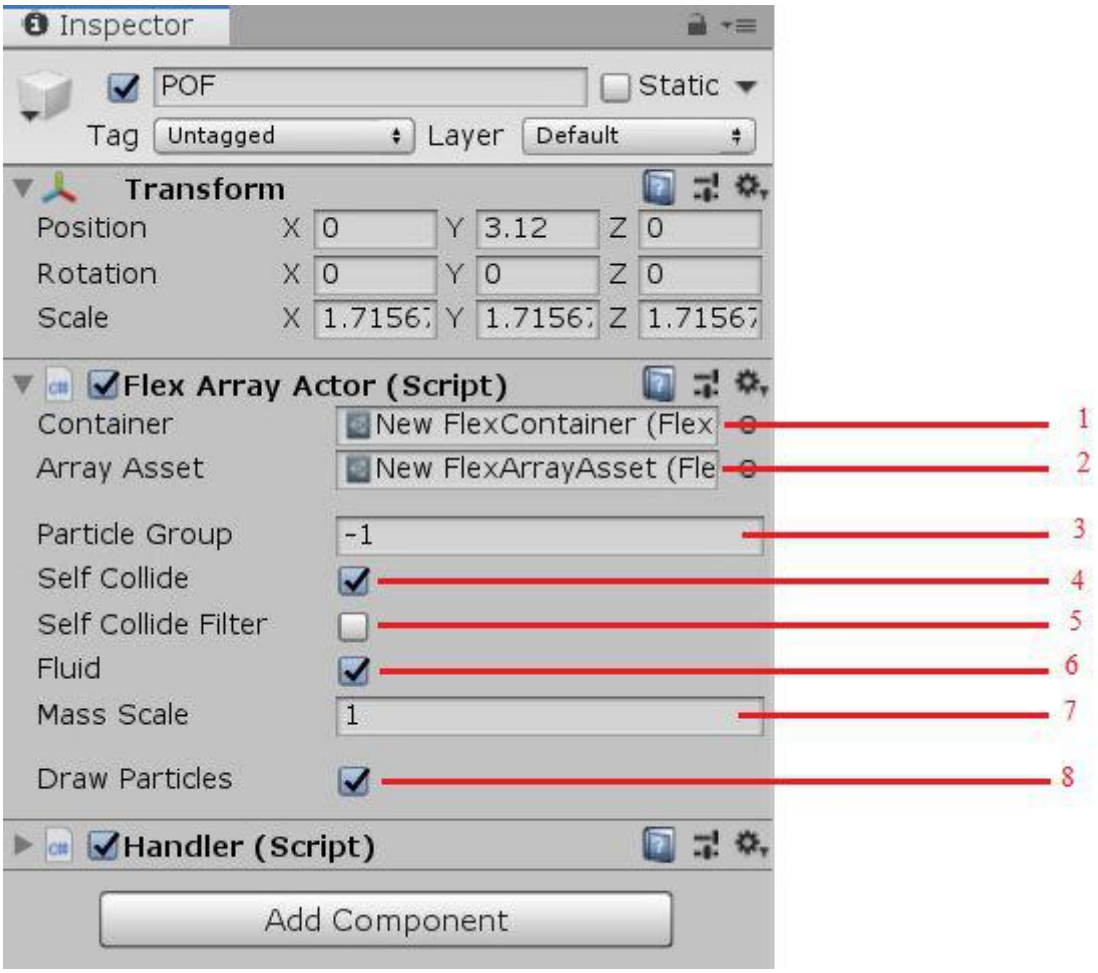


Figure 36: Description of POF inspector parameters.

1. Container: You can change the container by clicking this value. It is not recommended to change this row.
2. Array Asset: You can change the array asset by clicking this value. It is not recommended to change this row.
3. Particle Group: You can change particle grouping. -1 means grouping is not allowed between particles.
4. Self-Collide: Click this checkbox to change self-collision.
5. Self-Collide Filter: Collision only recognizes the particles. Other objects mesh does not collide with particle mesh.
6. Fluid: Click this checkbox to change particle form to fluid or vice versa.
7. Mass Scale:
8. Draw Particles: Click this checkbox to draw particles or vice versa.

References

1. Final Report revision 1.0

2. Requirement Specifications Document revision 2.0 (RSD 2.0)

3. Design Specifications Document revision 2.0 (DSD 2.0)

4. NVIDIA FleX manual, viewed 10 April 2020, <https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/flex/manual.html#manual>

5. Unity 2018.3 manual, viewed 10 April 2020, <https://docs.unity3d.com/2018.3/Documentation/Manual/index.html>

6. NVIDIA Flex Set up tutorial video playlist , viewed 10 April 2020, <https://www.youtube.com/watch?v=Fp1SMb3SWoo&list=PL4FII4B-zM0dMI-GgR3KsfJwm100MH3TT>

7. [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.

8. [ZB05] Zhu, Y., & Bridson, R. (2005). Animating sand as a fluid. (New York, NY, USA, 2005) ACM Trans. Graph., 24, 965-972.

9. Unity Hub download website, viewed 19 April 2020, <https://unity3d.com/get-unity/download>