YAŞAR UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING

COMP4910 Senior Design Project 1, Fall 2019
Supervisor: Dr. Gizem Kayar

POF: Performance Optimized Fluid System

Final Report

By:

Baran Budak -15070001012

Cihanser Çalışkan -16070001020

İsmail Mekan -15070001048

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This report was written by the group members and in our own words, except for quotations from published and unpublished sources which are clearly indicated and acknowledged as such.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 The source of any picture, graph, map or other illustration is also indicated, as is the source, published or unpublished, of any material not resulting from our own experimentation, observation or specimen collecting.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		27.12.19
Cihanser Çalışkan	16070001020		27.12.19
İsmail Mekan	15070001048		27.12.19

Project Supervisors:

Name, Last name	Department	Signature	Date
Gizem Kayar	Computer Engineering		27.12.19

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.
Color 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 liquid simulations. Firstly, algorithms that are necessary for the system were decided as a result of researches that had been made. Then, it was agreed to gather these algorithms. Secondly, the system was divided into a structure that has various algorithms and a control panel(controller) which administers these algorithms. While these substructures respectively, gets faster access to data; it provides memory efficiency. It determines the identification of surface and areas which to be visualized, then it visualizes. This project is research-based. So, it is possible for the structures we explain here to change and trade places with different structures. For this reason, these algorithms and structures may change later.

ÖZET

POF sistemi parçacık temelli sıvı simülasyonlarında yüzey tanımlamasını ve görselleştirmesini daha optimize ve hızlı bir şekilde sunmayı amaçlar. İlk olarak yapılan araştırmalar sonucunda sistem için gerekli algoritmalar belirlenmiştir ve bu algoritmaların bir araya getirilerek, anlaşılması sağlanmıştır. İkinci olarak sistem çeşitli algoritmalara ve bu algoritmaları yöneten ana bir kontrol paneline sahip bir yapıya bölünmüştür. Bu alt yapılar verilere daha hızlı erişim kazanırken hafıza verimliliği sağlar. Bu altyapılar sırasıyla uzaysal verilere ulaşımın hızlanması (hash) algoritması. Yüzeyin tanımlanması ve görselleştirilecek alanların belirlenip ve ekrana çizilmesidir. Projenin araştırma temelli olmasından dolayı anlattığımız yapıların değişmesi ve yerlerine daha farklı yapıların gelmesi muhtemeldir. Bu yüzden ilerleyen zamanlarda bu algoritmalar ve yapılar değişebilir.

TABLE OF CONTENTS

Table 2: 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.
GPU	Graphic Processing Unit.
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
MVC	Stands for Model View Controller. MVC is an application design model comprised of three interconnected parts (Model, View, Controller).

Table 3: List of acronyms/abbreviations

1. INTRODUCTION

This section gives main points about 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. There are millions of particles in a small number of liquids such as water. A particle is a rigid body sphere. Simulation applies physics to particles and these particles act as a liquid. Simulation having difficulties in calculations predicated on a surplus of particles. Indirectly, time and memory complexity increasing.

1.2. Project Goal

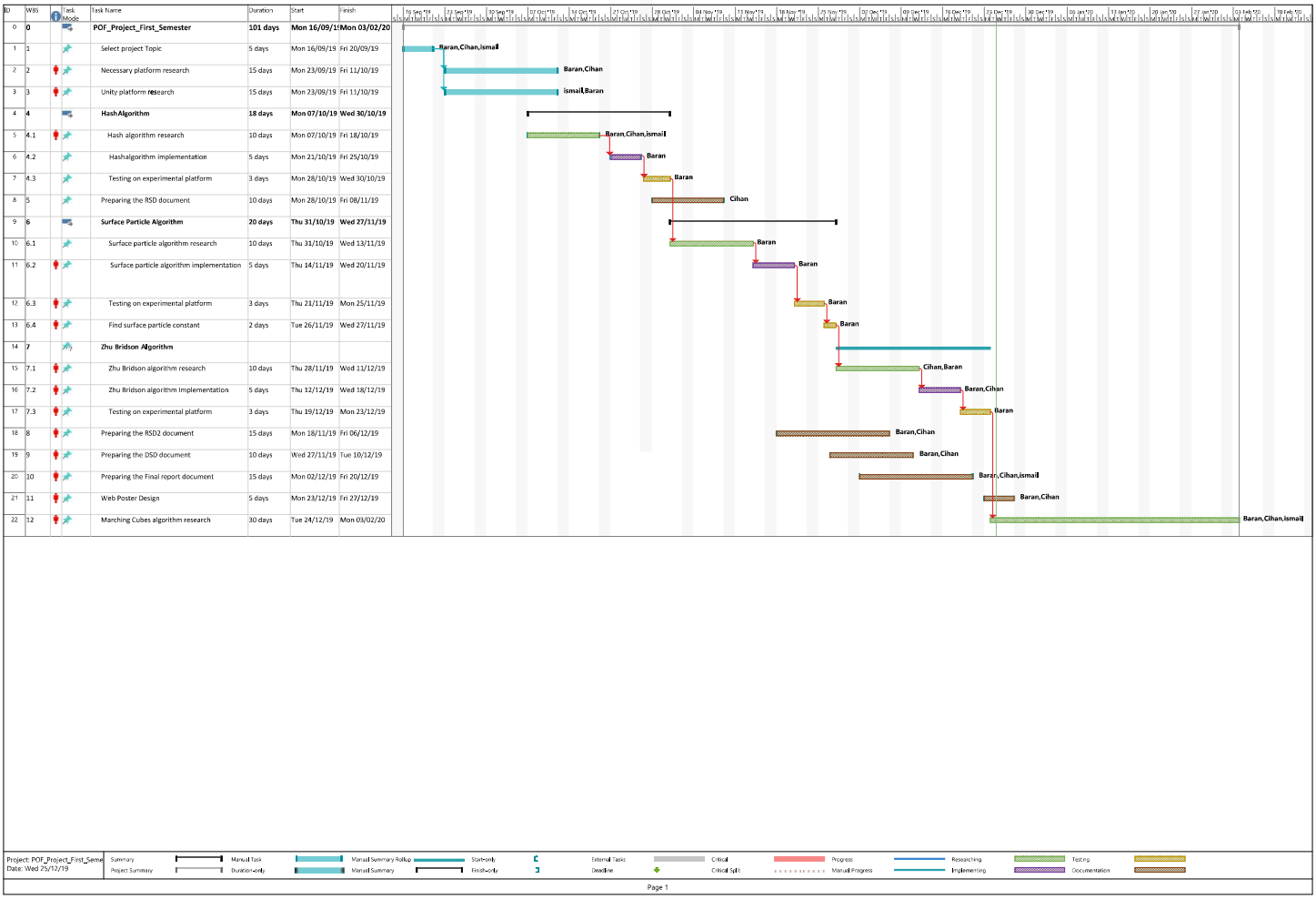
The main goal of the project researches whether there is a way to enhance fluid simulation. Increasing the efficiency and performance of an existing particle-based fluid simulation is a major goal. We aim to achieve these goals by implementing a variety of methods to the POF system such as using special structures to find store particles and visualize it by using various methods like the Marching cubes. In our project, there is no certain way because it is a research and development project and new more effective ways can be found during the project. Various methods and techniques will be researched and implemented while the project is in the development process.

1.3. Project Output

- Better performance.
- Better memory efficiency.
- Fluid-like appearance and behaviour.
- Testing of different algorithms for performance and efficiency.
- Higher frame rates per second.

1.4. Project Schedule

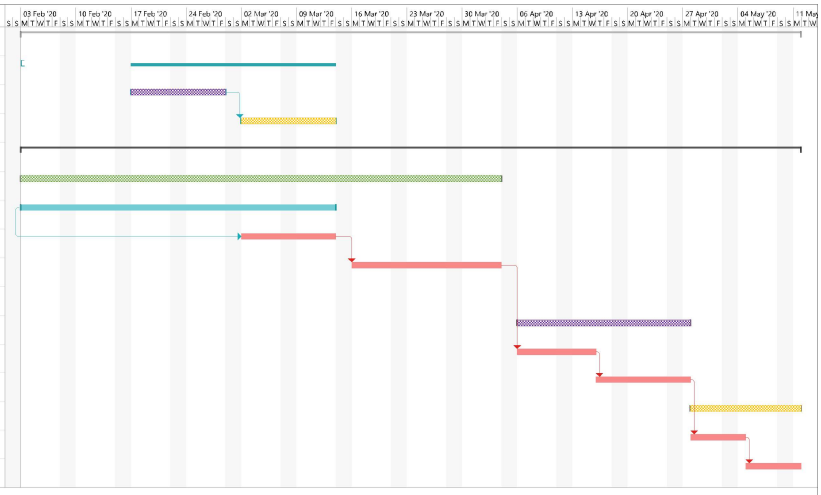
1.4.1 First Semester Gantt Chart



ID	WBS	Task Mode	Task Name	Duration	Start	Finish
0	**0**		**POF_Project_First_Semester**	**101 days**	**Mon 16/09/19**	**Mon 03/02/20**
1	1		Select project Topic	5 days	Mon 16/09/19	Fri 20/09/19
2	2		Necessary platform research	15 days	Mon 23/09/19	Fri 11/10/19
3	3		Unity platform research	15 days	Mon 23/09/19	Fri 11/10/19
4	**4**		**Hash Algorithm**	**18 days**	**Mon 07/10/19**	**Wed 30/10/19**
5	4.1		Hash algorithm research	10 days	Mon 07/10/19	Fri 18/10/19
6	4.2		Hashalgorithm implementation	5 days	Mon 21/10/19	Fri 25/10/19
7	4.3		Testing on experimental platform	3 days	Mon 28/10/19	Wed 30/10/19
8	5		Preparing the RSD document	10 days	Mon 28/10/19	Fri 08/11/19
9	**6**		**Surface Particle Algorithm**	**20 days**	**Thu 31/10/19**	**Wed 27/11/19**
10	6.1		Surface particle algorithm research	10 days	Thu 31/10/19	Wed 13/11/19
11	6.2		Surface particle algorithm implementation	5 days	Thu 14/11/19	Wed 20/11/19
12	6.3		Testing on experimental platform	3 days	Thu 21/11/19	Mon 25/11/19
13	6.4		Find surface particle constant	2 days	Tue 26/11/19	Wed 27/11/19
14	**7**		**Zhu Bridson Algorithm**			
15	7.1		Zhu Bridson algorithm research	10 days	Thu 28/11/19	Wed 11/12/19
16	7.2		Zhu Bridson algorithm Implementation	5 days	Thu 12/12/19	Wed 18/12/19
17	7.3		Testing on experimental platform	3 days	Thu 19/12/19	Mon 23/12/19
18	8		Preparing the RSD2 document	15 days	Mon 18/11/19	Fri 06/12/19
19	9		Preparing the DSD document	10 days	Wed 27/11/19	Tue 10/12/19
20	10		Preparing the Final report document	15 days	Mon 02/12/19	Fri 20/12/19
21	11		Web Poster Design	5 days	Mon 23/12/19	Fri 27/12/19
22	12		Marching Cubes algorithm research	30 days	Tue 24/12/19	Mon 03/02/20

1.4.2 Second Semester GanntChart

ID	Task Mode	WBS	Task Name	Duration	Start	Finish
0		**0**	**POF Project second semester**	**71 days**	**Mon 03/02/20**	**Mon 11/05/20**
1		**1**	**Marching Cubes**		**Mon 03/02/20**	
2		1.1	Implementing Marching Cubes	10 days	Mon 17/02/20	Fri 28/02/20
3		1.2	Testing Marching Cubes	10 days	Mon 02/03/20	Fri 13/03/20
4		**2**	**Algorithms**	**71 days**	**Mon 03/02/20**	**Mon 11/05/20**
5		**2.1**	**Researching Algorithms**	**45 days**	**Mon 03/02/20**	**Fri 03/04/20**
6		2.1.1	Algorithm research for efficiency and performance	30 days	Mon 03/02/20	Fri 13/03/20
7		2.1.2	Performance testing of system	10 days	Mon 02/03/20	Fri 13/03/20
8		2.1.3	Development of algorithms	15 days	Mon 16/03/20	Fri 03/04/20
9		2.1.4	Determine algorithms	0 days		
10		**2.2**	**Implementing Algorithms**	**16 days**	**Mon 06/04/20**	**Mon 27/04/20**
11		2.2.1	Choosen algorithm implementation	8 days	Mon 06/04/20	Wed 15/04/20
12		2.2.2	Visualization changes	8 days	Thu 16/04/20	Mon 27/04/20
13		**2.3**	**Testing**	**10 days**	**Tue 28/04/20**	**Mon 11/05/20**
14		2.3.1	Testing of choosen algorithm	5 days	Tue 28/04/20	Mon 04/05/20
15		2.3.2	Testing of visualization	5 days	Tue 05/05/20	Mon 11/05/20



Project: POF Project second sem
Date: Wed 25/12/19

Task · Summary · Project Summary · Manual Task · Duration-only · Manual Summary Rollup · Manual Summary · Start-only · Finish-only · Critical · Critical Split · Progress · Manual Progress · Researching · implementing · Testing

Page 1

2. DESIGN

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

2.1. High Level Design

2.1.1. Package Diagram

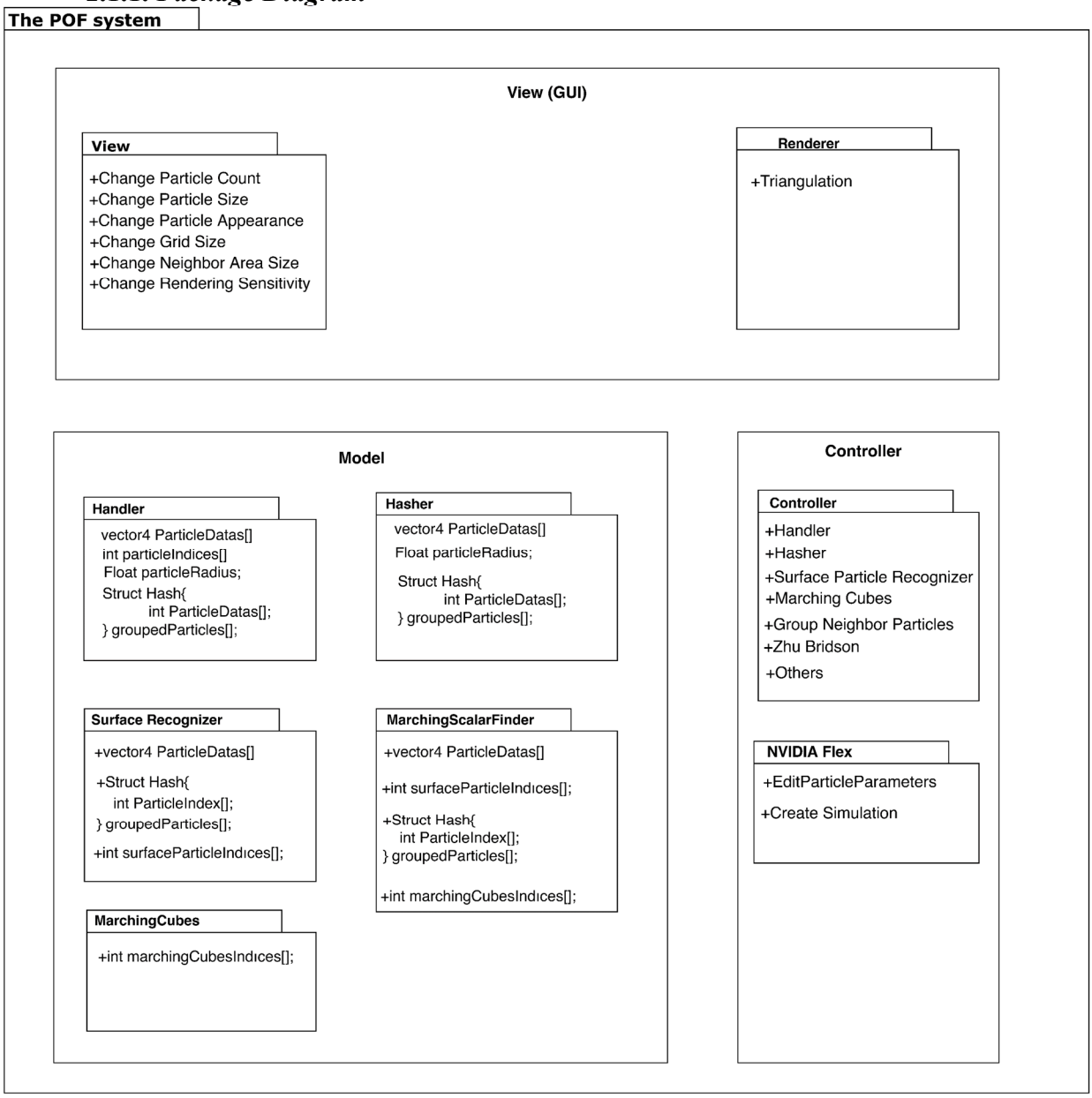


Fig 1: Package Diagram

Package diagram explains classes and their attributes along with relations. NIVIDA flex and Handler classes are the main parts of the system. Handler communicates to transmit data to relevant classes.
This section is extensively explained in design specification document [3] and it will be elaborated in future.

2.2. Detailed Design

Detailed design part is mentioned in design specification document. In DSD, [3] detailed design is explained with activity diagrams of the POF system. This section will be elaborated in future works.

2.3. Realistic Restrictions and Conditions in the Design

We had to neglect some aspects of the project to implement the project in a year. The security issue is ignored because the project aims to help everybody who has interested fluid simulations and contribute to science. We assumed that users of the POF system have the necessary equipment and software and know to how to use them.

3. IMPLEMENTATION and TESTS

This section describes implementation stages of the POF system. Code parts are given

3.1. Implementation of the System

3.1.1. Research Papers and System Structure

This project based on these two research papers, surface reconstruction algorithm that implemented by Zhu et al [ZB05] and Marching cubes algorithm [WH87]. However, some problems occur when system structure creation stage is started on visualization. Our main problem was performance and memory efficiency on that point, we added two algorithms to our project for passing over on performance and memory issues.

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).

Table 4: Problem & Solution

After these solutions, we have started implement our system structure by creating classes, but we realize complexity getting higher due to interconnection between classes, so we create a handler class for control every classes in the one class.

3.1.2 Implementation of Hash algorithm

We use spatial hash algorithm to access particle position easily. Hash algorithm simplifies the three dimensions of float particle positions to integer id numbers in specific order [TH03].

Cube numbering is 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 subtract 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 backward to forward in z dimension.

```
int xId = (int)Math.Ceiling((particle.x - _bounds.min.x) / _length);
int yId = (int)Math.Ceiling((_bounds.max.y - particle.y) / _length);
int zId = (int)Math.Ceiling((particle.z - _bounds.min.z) / _length);
```

Fig 2: cell id numbering

We can reach cell id by position without storing it.

```
public struct vertexIndex
    {
        public int[] pointIndice;
        public vertexIndex(int[] pointIndice)
        {
           pointIndice = new int[1] { -1 };
           this.pointIndice = pointIndice;
        }
    }
```

Fig 3: Group struct

```
this._intervalx = (int)Math.Ceiling((_bounds.max.x - _bounds.min.x) / (_radius * 4));
this._intervaly = (int)Math.Ceiling((_bounds.max.y - _bounds.min.y) / (_radius * 4));
this._intervalz = (int)Math.Ceiling((_bounds.max.z - _bounds.min.z) / (_radius * 4));
groups = new vertexIndex[this._intervalx * this._intervaly * this._intervalz];
```

Fig 4: Hash size

Vertex index struct represents cells by creating an array on this struct. We created cell ids as indices. By finding how many grids in each dimension we find our hash table size represented as fig.4. We find particles by looking cells. Each cell represented by integer id number as seen in fig.3.

We have realized during the implementation when particle centre on intersection point of cell boundaries. We solve this problem by calculating subtract particle x/y/z from boundary max x/max y/max z and divide our grid size. If the remainder is equals to zero is on the boundary.

```
if ((_bounds.max.y - particle.y) % _radius == 0) {
    cubeID = (xId) + (this._intervalx * (yId--)) + (this._intervalx * this._intervaly * zId);

    checkS(cubeID-1, _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 linear search of the particles. Ceiling is implemented to derive integer numbers because particles are float and they are derived according to the AABB size.

3.1.3 Particle neighbour algorithm

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

We need four corner cells to solve the problem of finding neighbour particles [AIA12].

```
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) + 1,
int ty = ((bottomLeftBackward- topLeftBackward) /_intervalx) + 1,
int tz = ((topLeftForward-topLeftBackward) /(_intervalx*_intervaly))+1;
```

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 [AIA12].

```
int[] areaNums = Enumerable.Repeat(-1, (9+((ty*tx*tz)))).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));
}
```

Fig 8: Finding cell numbers in that area.

This project is not finished. Therefore, this section will be completed in future works.

3.2. Tests and Results of Tests

3.2.1 Availability of the Necessary Environment

Before finding NVIDIA Flex, we have tested three particle-based fluid simulation and we disqualify for these reasons.

uFlex	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	Obifluid is eliminated 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 in three thousand particles we have 3fps.
Screen Space Fluids Pro	Like uFlex we recognize small bugs and errors in the code, and we fixed it, but performance was very low on higher particle count.

4. CONCLUSIONS

In this section, the cost table of workers is given and explained. The cost of software and hardware is given with details and benefits of the projects are explained. This part of the final report summarizes our project and gives a cost analysis for the project. Future works mentioned.

4.1. Summary

The massive amounts of particles can be a computational hardship for the computer.
We implement various methods to get better results by making a research. Our project focuses on catalysing computational difficulties by increasing the performance and efficiency. Project should make easier to simulate with higher quantities of particles or getting better results with the same number of particles.

4.2. Cost Analysis

4.2.1 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 5: Cost Analysis 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 in 10 December 2019.

4.2.2 Cost of Software

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

Table 6: Cost of software

4.2.3 Cost of Hardware

4.2.3.1 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 7: Component costs of PC1

4.2.3.2 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 8: Component costs of PC2

4.3. Benefits of the Project

Our project can benefit in all areas where liquid simulation is available.

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

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

- ***4.3.3 Games:*** Some games need a fluid simulation system to make more realistic games. The POF system can be a good factor for the makes realistic games. For instance, in sailing simulator game is a perfect match for our system.

- ***4.3.4 Construction:*** The construction and Architecture sector 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.

4.4. Future Work

We will develop our project in order to achieve performance and efficiency goals. The functionality of the project will remain the same. However, small changes in the calculations will be changed to get better results.

References

1. Requirement Specification Document revision 1.0 (RSD 1.0)
2. Requirement Specification Document revision 2.0 (RSD 2.0)
3. Design Specification Document revision 1.0 (DSD 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. [AIA12] Akinci, G., Ihmsen, M., Akinci, N. and Teschner, M. (2012). Parallel Surface Reconstruction for Particle-Based Fluids. Computer Graphics Forum, 31, 1797-1809.
7. [TH03] Teschner, M., Heidelberger, B., Müller, M., Pomerantes, D., and Gross, M.H. (2003). Optimized Spatial Hashing for Collision Detection of Deformable Objects. VMV.

APPENDICES

APPENDIX A: REQUIREMENTS SPECIFICATION DOCUMENT

COMP4910 Senior Design Project 1, Fall 2019

Advisor: Gizem Kayar

POF: Performance Optimized Fluid Requirements Specification Document

4.12.2019

Revision 2.0

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

Contents

1.0 Introduction

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 simulation utilizing running it faster while occupying less memory of the computer. 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.

1.2. *Scope*

The POF system shall help to increase performance for simulating fluids. The system reduces the necessary computation operation for particles during the simulation. Initially, the POF system runs the NVIDIA flex because the POF system needs particle position data. The POF system works with the Unity engine for visualization but another program can be used for simulation.

The POF system computes the color field quantity of each particle and marks all the surface particles. Surface particles are calculated and marked for the 2r distance which is the two times of a particle radius and each particle is the same. Marked vertices control by the handler in our code. For every surface vertex, we compute every small cell in the 4 times of radius area in the axis-aligned bounding box (AABB). After we find the cells, we find the particles in those cells and calculate how every particle affects the other particles as a scalar value of the vertex which the method is defined in [ZB05]. Lastly, for all the vertex data, we draw a cube and check those eight vertices of a cube and if the cell is on the surface, we pass the information of the vertices for the triangulation stage.

1.3. *Overview*

This document describes the POF system. Requirements specification document involves diagrams that define user roles in the system and more importantly, explaining how the system operates in the background. The document mostly focuses on the specified requirements. System functions are defined by expressing 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

2.1 *Use case diagram*

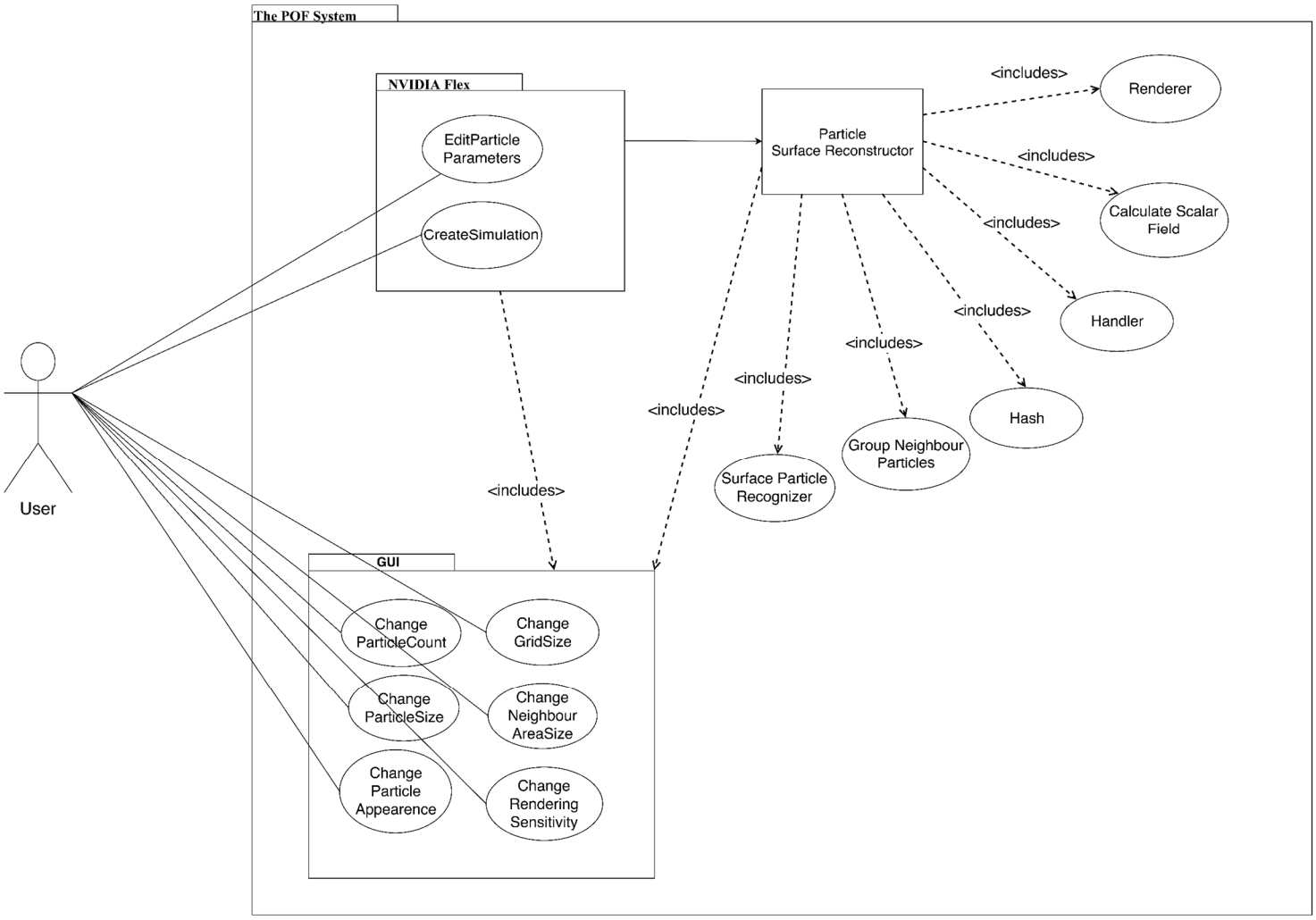


Fig 1: Use Case Diagram

2.2 *Sequence diagram*

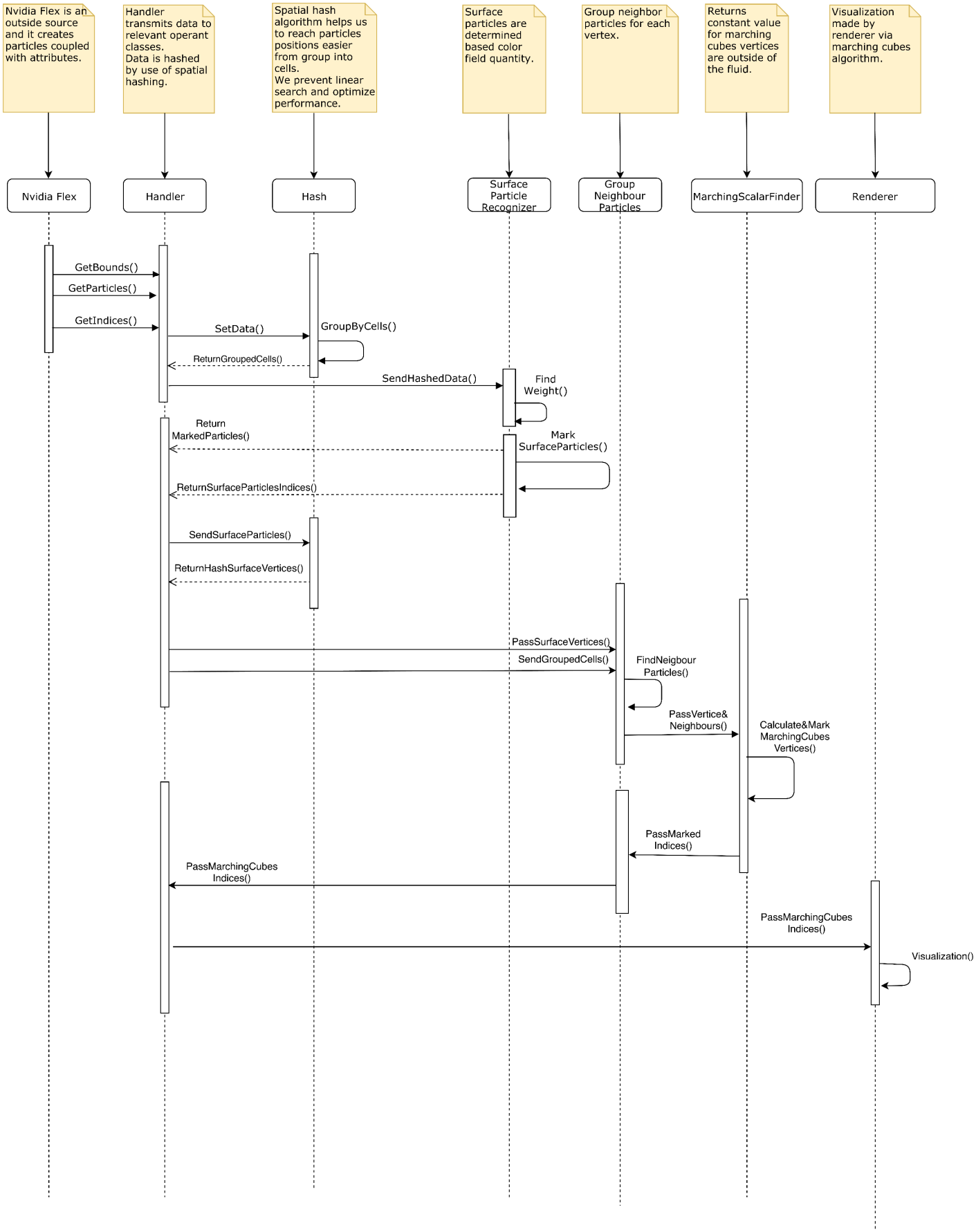


Fig 2: Sequence Diagram

3.0 General Description

The requirement specification document is used to provide a high-level description of the POF system. The documentation describes the mechanism and explain their 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 color 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 [ZB05]. The system visualizes the particles by using the Marching cubes algorithm [WH87].

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 simulation enhancement. We will use NVIDIA flex for creating particles and using particle data to process it for our algorithm. Besides, it is unnecessary to strive with particle physics for our project because it is aimed that enhancing the performance of the already existed particle-based fluid system by surface reconstruction and it is not aimed to create a fluid simulation system from scratch.

4 Functional Requirements

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

4.1 Retrieve the 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 [WH87] 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 [ZB05].

4.2.1 Zhu and Bridson

The research paper written by Zhu and Bridson [ZB05] 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 [ZB05] calculate a scalar value of vertices that outside of the fluid to send marching cubes to visualize.

4.2.2 Mathematical Equations

$$\phi(x) = |x - x_0| - r_0 \quad \textbf{(1)}$$

$$\phi(x) = |x - \bar{x}| - \bar{r} \quad \textbf{(2)}$$

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

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

k is a kernel function; k(s) = max $(0\ , (1 - s^2)^3)$.

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 Importance of particle classification for memory efficiency

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

4.3.1 Color field quantity

Color 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.

$$c_s(r) = \sum_J m_J \frac{1}{p_J} W(r - r_J, h) \qquad \textbf{(5)}$$

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:

$$k(s) = \max(0, (1 - s^2)^3) \qquad \textbf{(6)}$$

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

When the cells are found, afterward we find the particles in those cells and calculate how every particle affects the other particles as a scalar value of the vertex in which the method is defined.

4.4 Marching cubes

The algorithm is used for extracting a polygonal mesh of an isosurface from a three-dimensional discrete scalar field. In this project, the marching cubes algorithm is used with the [ZB05] algorithm. [ZB05] algorithm is used in the marching cubes algorithm to get better visual outputs.

4.5 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 can be listed in three topics such as efficiency, performance, and usability. These requirements are the main goals of the POF system. Some requirements may not be satisfied because they are not promised to realize.

Non-Functional Requirements	**Description**
Efficiency	The aim of the POF system is efficient memory usage.
Performance	The system's performance should be increased after the application POF to the 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 in OpenGL or other various platforms. However, our project will be deployed into the Unity game engine which is supported on Windows and macOS.

Table 1: Non-Functional Requirements

6.0 Glossary

Term	Description
API	Acronym for 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.
Color field quantity	It is a 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.
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	An Acronym stands for the performance-optimized fluid system.
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.
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.

In order 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

9.0 References

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

[BP94] Paul Bourke 1994, Marching Cubes, viewed 1 December 2019, <**http://paulbourke.net/geometry/polygonise/**>

[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.

[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.

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

[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.

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

APPENDIX B: DESIGN SPECIFICATION DOCUMENT

COMP4910 Senior Design Project 1, Fall 2019
Advisor: Gizem Kayar

POF: Performance Optimized Fluids

High Level Design
Design Specifications Document

Revision 1.0
8.12.2019

By:

Baran Budak -15070001012

Cihanser Çalışkan -16070001020

İsmail Mekan -15070001048

Revision History

Revision	Date	Explanation
1.0	12.12.2019	Initial high level design

Table of Contents

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 simulate it. Various methods will be implemented and tested during the research and development of this project. The main goal is making research and sharing our observations of the project results. One of the major project objectives is to reach a more efficient and better performance fluid simulation system but it is not promised because there is no certain way to achieve it and as mentioned, the project is mainly research-based.

The design is based on The POF system Requirements Specification Document, Revision 2.0 [1]
This design process conforms to the Requirements Specification Document and its diagrams. The project conforms to UML diagrams. Diagrams are describing the project to understand mainly operations of the POF system. Imperceptible parts of the POF system can be changed but the main functioning of the 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 are 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 given later in the third section of this document.

2. POF System High Level Design

This section describes the POF system with high level design. High level design section mentions about the POF system architecture and structure. Besides, the system environment is included. High level design part will be elaborated in future work.

2.1. POF System Architecture

The POF system architecture works with NVIDIA Flex as an outsource asset. NVIDIA flex is mandatory because particle positions and AABB data are necessary. The system has a handler between the NVIDIA flex and the POF system. Initially, Flex starts the simulation and creates the particles and AABB. The handler passes data for relevant classes. We have a hash function that uses a hash algorithm to make easier and faster data access. Surface particle recognizer function determines the particles that are on the surface by calculating colour field quantity. Surface particles and their vertices grouped for a specific radius, which is made by group neighbour particle function. Afterward, grouped neighbour particle data send to the marching cubes algorithm [WH87] and it determines which vertices should be drawn. The last part is triangulation and drawing the particles by the renderer section.

2.2. POF System Structure

In this section POF system structure is described with UML diagrams. Diagrams are such as use case, sequence and package diagram.

2.2.1 Use Case Diagram

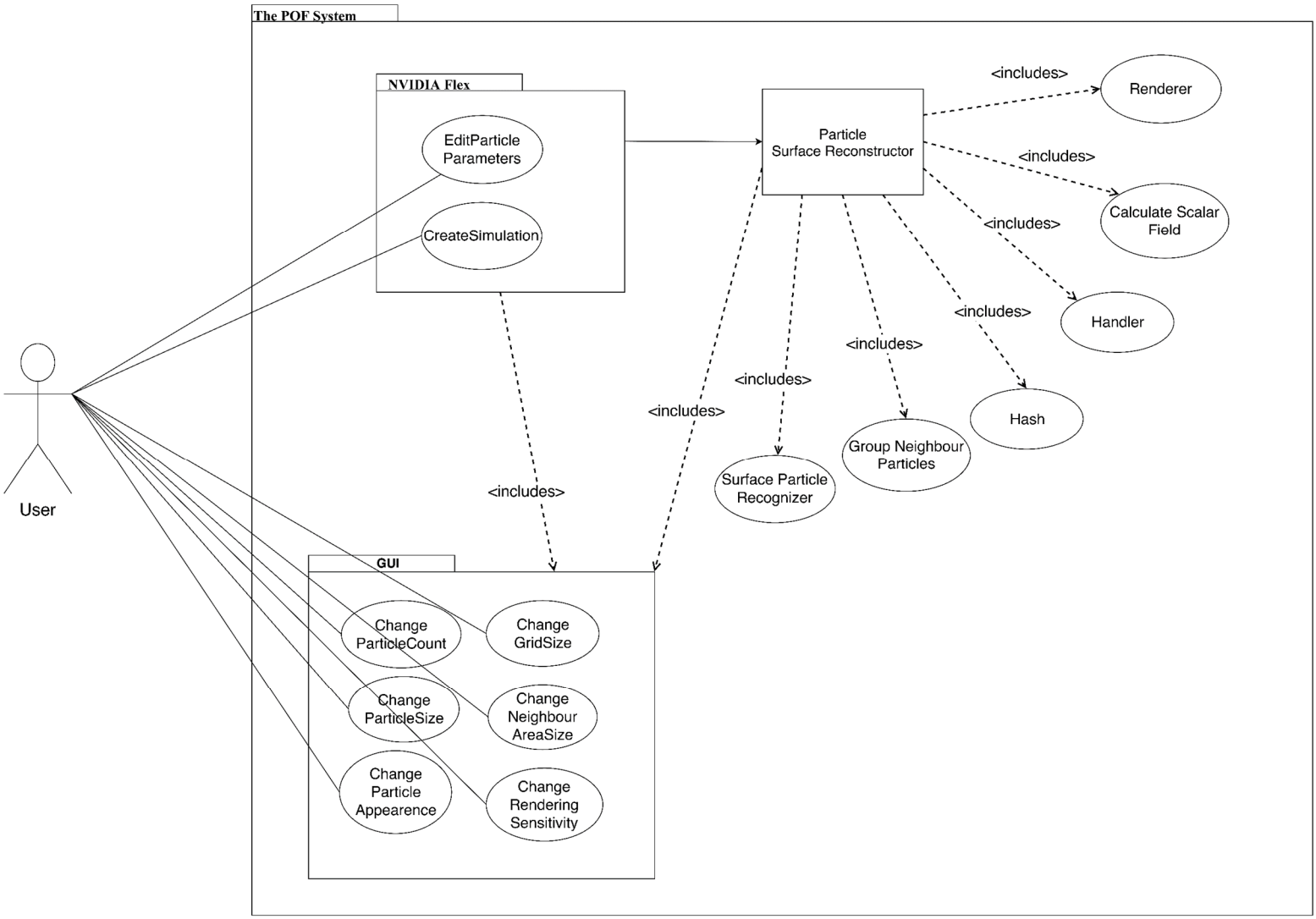


Fig 1: Use case diagram

Title	Description
Calculate Scalar Field	Calculates a constant value of a particle in given range.
Change grid size	Interval of grid size of axis aligned bounding box can be changed with this function.
Change neighbour area size	Neighbour particle range of volume can be changed which 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 count	Particle number in the scene can be edited.
Change particle size	Radius of the particle can be changed with this function. Increase in the particle size will result slow performance compared to less particle size.
Change rendering sensitivity	Rendering sensitivity can be changed with this function. If sensitivity increase, fluid visualization will be more precise however processing time will increase.
Create Simulation	NVIDIA Flex simulation initialize when this function called.
Edit particle parameters	Particle attributes can be edited by user from GUI. Parameters can be maximum particle number, particle size, friction, adhesion etc.
Group Neighbour Particles	Neighbour particles are grouped by looking a specific range.
Handler	Handler transmits data between layers and relevant classes. Handler manage data transmission.
Hash	Hashes the particle and cell position.
Renderer	Visualize fluid by drawing the given polygons.
Surface Particle Recognizer	Surface particles marked and processed for the necessary calculations in this function.
User	Users can be anyone who has access to the program.

Table 1: Description of the use case diagram

2.2.2 Sequence Diagram

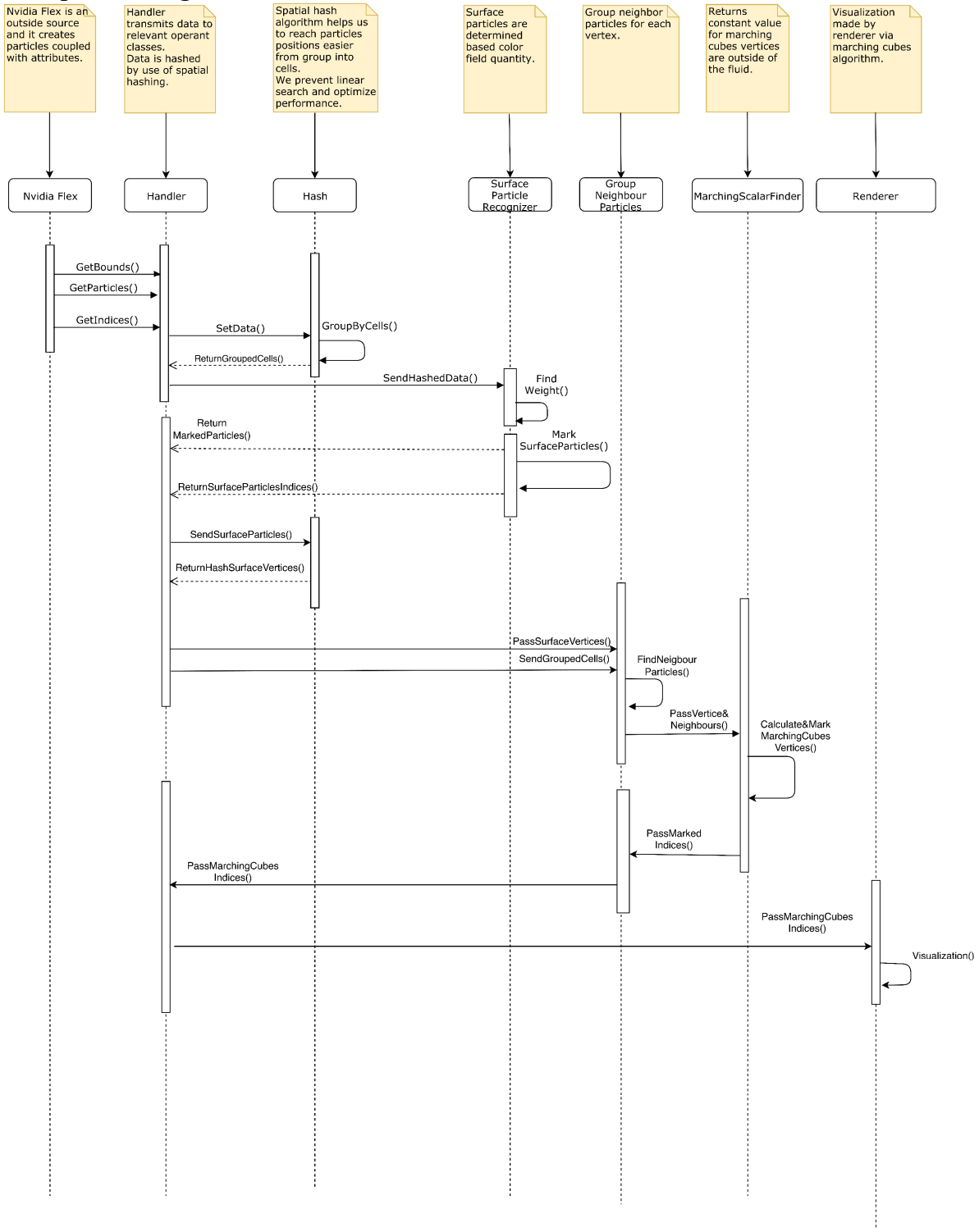


Fig 2: Sequence diagram

Description:

Handler manages data transfer between other sections. If any data has to transmit to another class, Handler executes this operation. NVIDIA flex is an already existed particle-based fluid system that is outsourced and initializes the fluid simulation. Hash applies a special algorithm and makes it easier and faster store and reach it to particle and cell data. Surface particle recognizer finds the surface particles by computing colour field quantity value. Group neighbour particles function finds a particle's neighbours in a specific range. Marching scalar finder function computes a constant value for the particle vertices that outside of the fluid. Renderer makes the triangulation of the specified vertices and draws the fluid for each frame.

2.2.3 Package Diagram

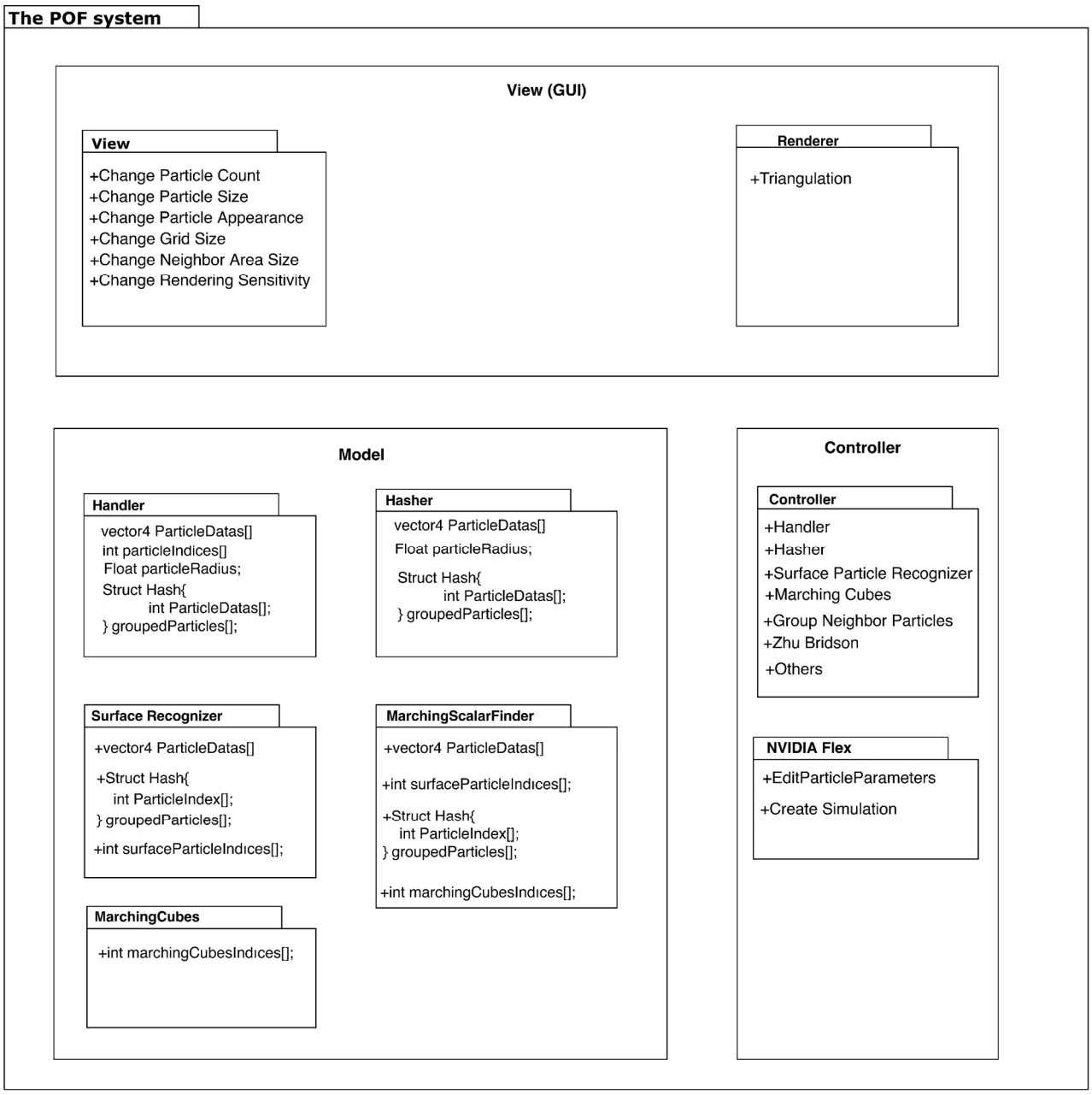


Fig 3: Package diagram

Description:

The package diagram is based on the MVC (model view controller) system. The model section consists of NVIDIA flex and surface particle recognizer. NVIDIA flex creates a simulation and can change hydrodynamic attributes of the particles. A surface particle recognizer is another package in the model section. Hash calculates cell id based on the boundaries of cells. The function of calculating the scalar field returns a constant value of the particle. Group neighbour particles compute the weight of a particle by checking nearby particles in a specific range. The marching cubes algorithm determines which vertices will be triangulated. The implementation of [ZB05] is needed for reconstructing the surface. The controller part has a handler that controls data transmission and communication between sections. View section can change particle attributes such as particle count, size and appearance. Change grid size affects cell sizes. Change neighbour area size can affect the particle rendering. Changing the rendering sensitivity of the rendering affects the appearance of the POF fluid system directly.

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

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

This system has low performance on this project because it can handle very small number of particles. The optimal system should be workstation defined in final report [3].

3. POF System Detailed Design

This section describes the small parts of the POF system. Functionality of these small parts are explained on activity diagrams.

3.1 Activity Diagram of Zhu & Bridson

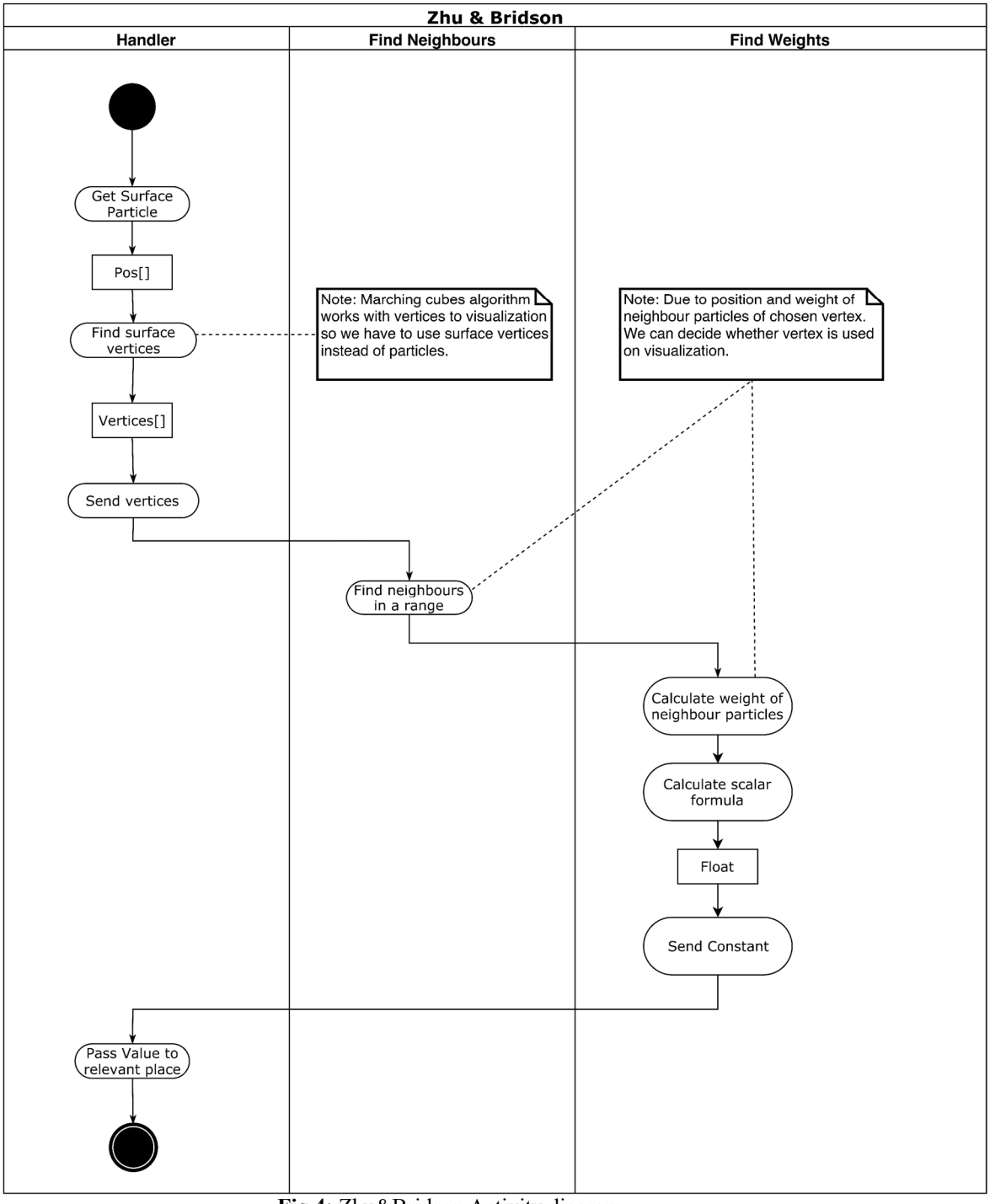


Fig 4: Zhu&Bridson Activity diagram

Description: Handler gets the surface particle from relevant sections. The handler receives the surface vertices and sends it to find neighbours section which is responsible to find neighbours of the particle in a specific range. Find neighbours pass neighbour particle data to find weight section. Find weight calculates the weight of the neighbour particles of a specific particle [ZB05]. Weight is used for calculating a scalar value. Find weight send constant value to the handler. Handler knows where to send specific data.

3.2 Activity Diagram of Surface Recognizer

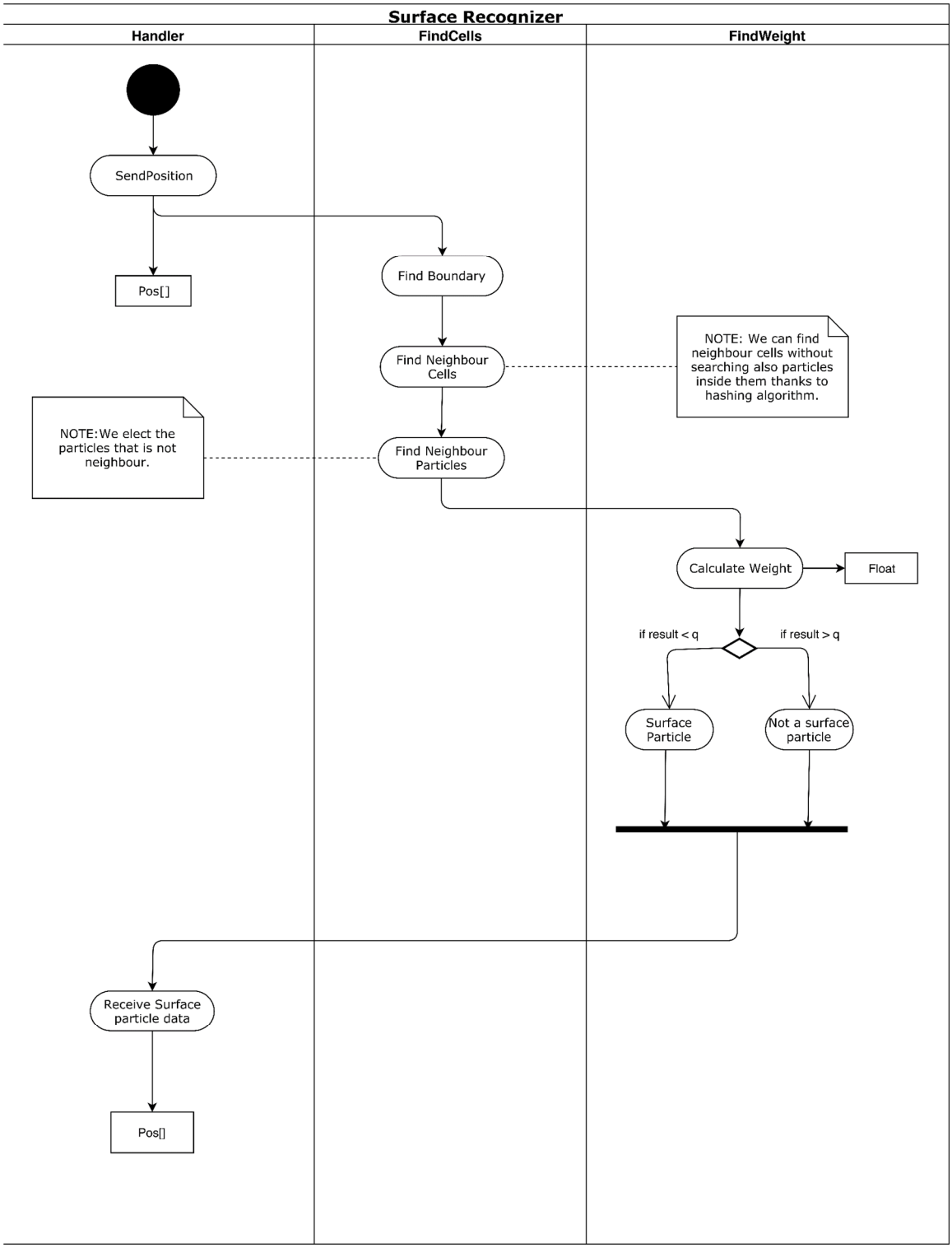


Fig 5: Surface recognizer activity diagram

Description: Handler sends position data of the particle to find cell function. Find neighbour cells section find boundaries and neighbour cells. After that, find cells computes the neighbour particles and elect the particles that are not neighbour with each other and passes the data to find weight section. Find weight section calculates the weight and if weight is smaller than specific constant value 'q'[AIA12], (This constant called kernel can be changed for better results in the future

implementations of the project.) particle marked as a surface particle. If weight is bigger than a constant value, the particle is not a surface particle. Marked particles are sent to the Handler

3.3 Activity Diagram of Marching Cubes

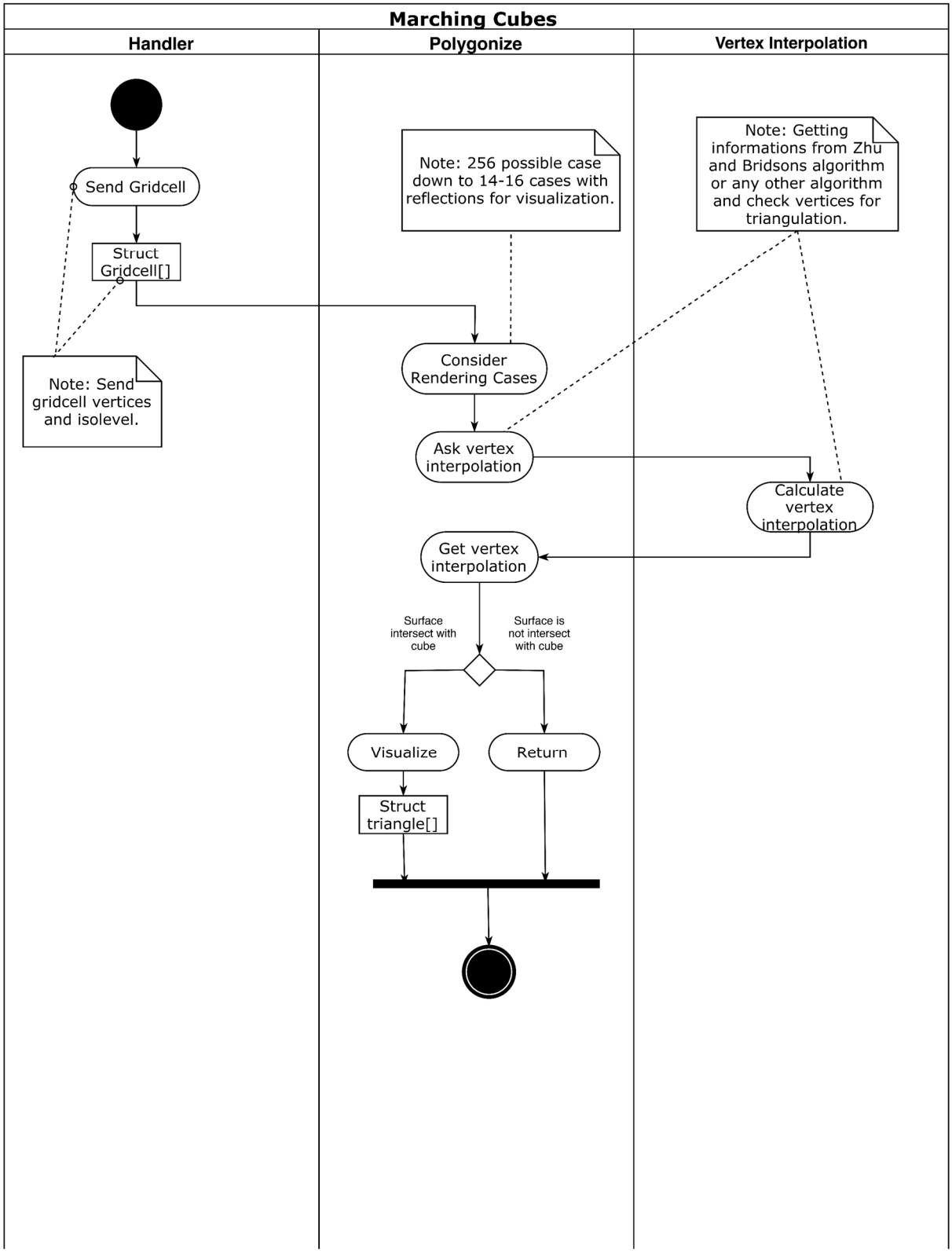


Fig 6: Marching cubes activity diagram

Description: Handler sends grid cell (Cell is a structure consisting of vertices and iso-level) to polygonise function [WH87]. Polygonise consider rendering by looking cases (There are 16 cases predefined). After that, vertex interpolation calculates a value and returns to polygonise function

[ZB05]. If the surface intersects with the cube, the vertex will be visualized. If the surface does not intersect with the cube, the vertex will not be drawn.

3.4 Activity Diagram of Hasher

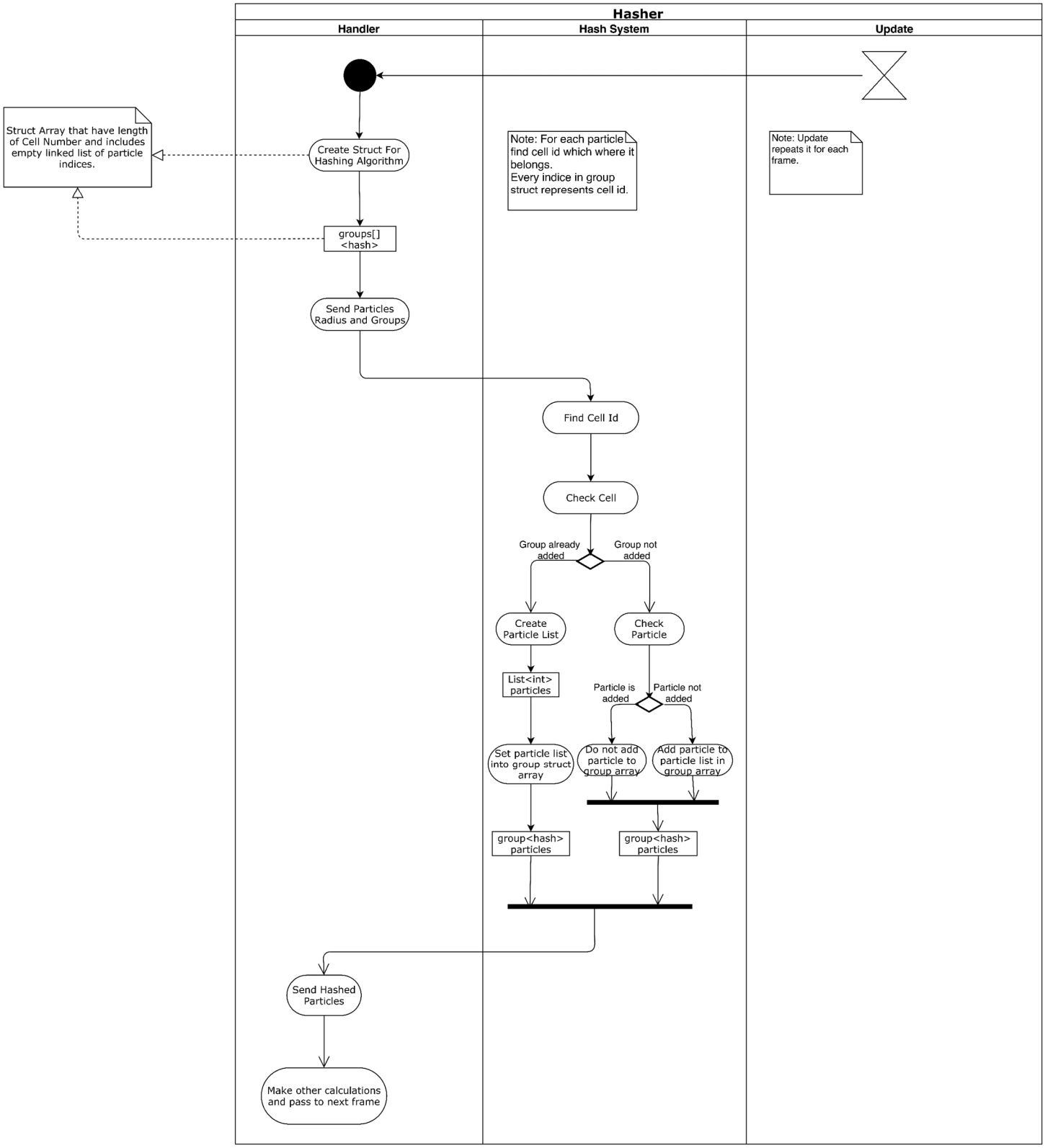


Fig 7: Hasher activity diagram

Description: Handler sends particle radius and groups to the hash system. Hash system finds the cell that particle includes and checks the cell [TH03]. If the group already added, the hash system creates

a particle list and sets it into the group array and sends hashed particles to the handler. If the group is not added, add the particle to particle list in group array and send hashed particles to the handler.

4. Testing Design

Since our project is heavily research-based, the testing design is not handled smoothly in the earlier stages of our project. We will implement various methods mentioned in research papers on our project. The Unity 3D game engine will be used to test results. The best methods will be determined according to the test results. The main standards are performance and efficiency. Memory and CPU usage is important. NVIDIA flex communicates with the GPU for the efficiency of the simulation. Detailed comparison tables will be sketched and elaborated in the future.

Table of figures

References

1. Final Report revision 1.0
2. Requirement Specification Document revision 2.0 (RSD 2.0)
3. Use case and sequence diagrams in RSD 2.0
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. [AIA12] Akinci, G., Ihmsen, M., Akinci, N. and Teschner, M. (2012). Parallel Surface Reconstruction for Particle-Based Fluids. Computer Graphics Forum, 31, 1797-1809.
7. [TH03] Teschner, M., Heidelberger, B., Müller, M., Pomerantes, D., and Gross, M.H. (2003). Optimized Spatial Hashing for Collision Detection of Deformable Objects. VMV.