**A picture containing clipart

Description automatically generated**

**COMP4910 Senior Design Project 1, Fall 2019**

**Advisor: Gizem Kayar**

**POF: Performance Optimized Fluids**

**High Level Design**

**Design Specifications Document**

**Revision 2.0**

**4.3.2020**

**By:**

**Baran Budak -15070001012**

**Cihanser Çalışkan -16070001020**

**İsmail Mekan -15070001048**

# Revision History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Explanation** |
| 1.0 | 12.12.2019 | Initial high level design |
| 2.0 | 4.03.2020 | Extended high level design |

# Table of Contents

[**Revision History 2**](#_Toc501993332)

[**Table of Contents 3**](#_Toc501993333)

[**1. Introduction 4**](#_Toc501993334)

[**2. POF System High Level Design 4**](#_Toc501993335)

[**2.1. POF System Architecture 4**](#_Toc501993336)

[**2.2. POF System Structure 5**](#_Toc501993337)

[**2.2.1 Use Case Diagram 5**](#_Toc501993337)

[**2.2.2 Sequence Diagram 7**](#_Toc501993337)

[**2.2.3 Package Diagram 8**](#_Toc501993337)

[**2.3 POF System Environment 9**](#_Toc501993338)

[**3. POF System Detailed Design 10**](#_Toc501993339)

[**3.1 Activity Diagram of Zhu & Bridson 10**](#_Toc501993339)

[**3.2 Activity Diagram of Surface Recognizer 11**](#_Toc501993339)

[**3.3 Activity Diagram of Marching Cubes 13**](#_Toc501993339)

[**3.4 Activity Diagram of Hasher 13**](#_Toc501993339)

[**4. Testing Design 15**](#_Toc501993340)

[**Table of Figures 15**](#_Toc501993340)

[**References 15**](#_Toc501993341)

**List of Figures**

[**Fig 1:** Use case diagram 5](#_Toc501993332)

[**Fig 2:** Sequence diagram 7](#_Toc501993333)

[**Fig 3:** Package diagram 8](#_Toc501993332)

[**Fig 4:** Activity diagram of Zhu&Bridson 10](#_Toc501993333)

[**Fig 5:** Activity diagram of surface recognizer 11](#_Toc501993332)

[**Fig 6:** Activity diagram of marching cubes 12](#_Toc501993333)

[**Fig 7:** Activity diagram of hasher 13](#_Toc501993332)

**List of Tables**

[**Table 1:** Description of the use case diagram 5](#_Toc501993332)

# 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. However, indicated features are not guaranteed to improve performance. The project goal is visualizing the particle-based fluid system differently by benefiting from certain algorithms and the system is expected to work more efficiently as a result of the implementation of the algorithms in the research papers. The project is heavily research and development 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

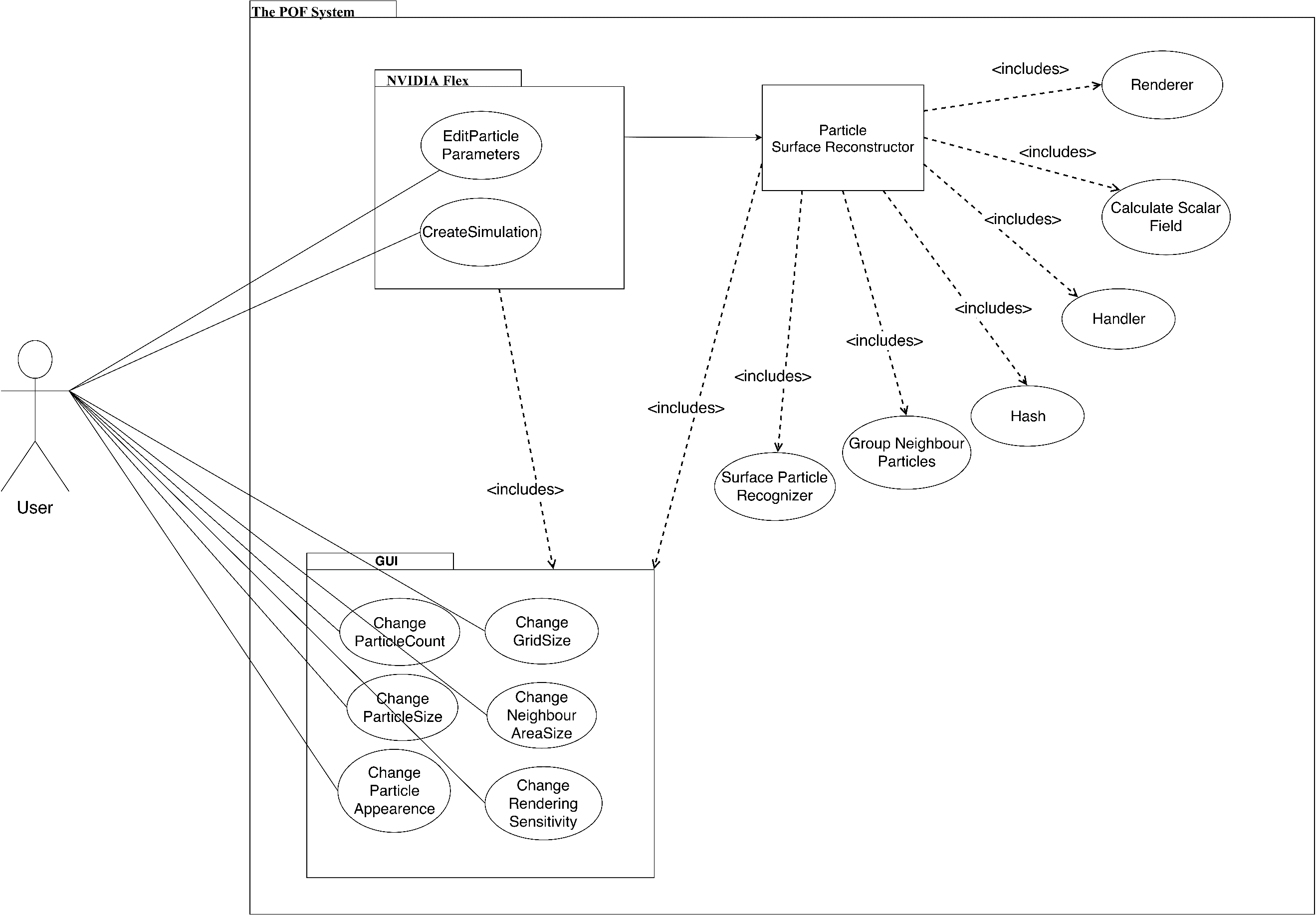
This section describes the POF system with high-level design. The high-level design section mentions about the POF system architecture and structure. Besides, the system environment is included. The high-level design part will be elaborated in future work.

# *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 color field quantity. Surface particles and their vertices grouped for a specific radius, which is made by group neighbor particle function. Afterward, grouped neighbor 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*

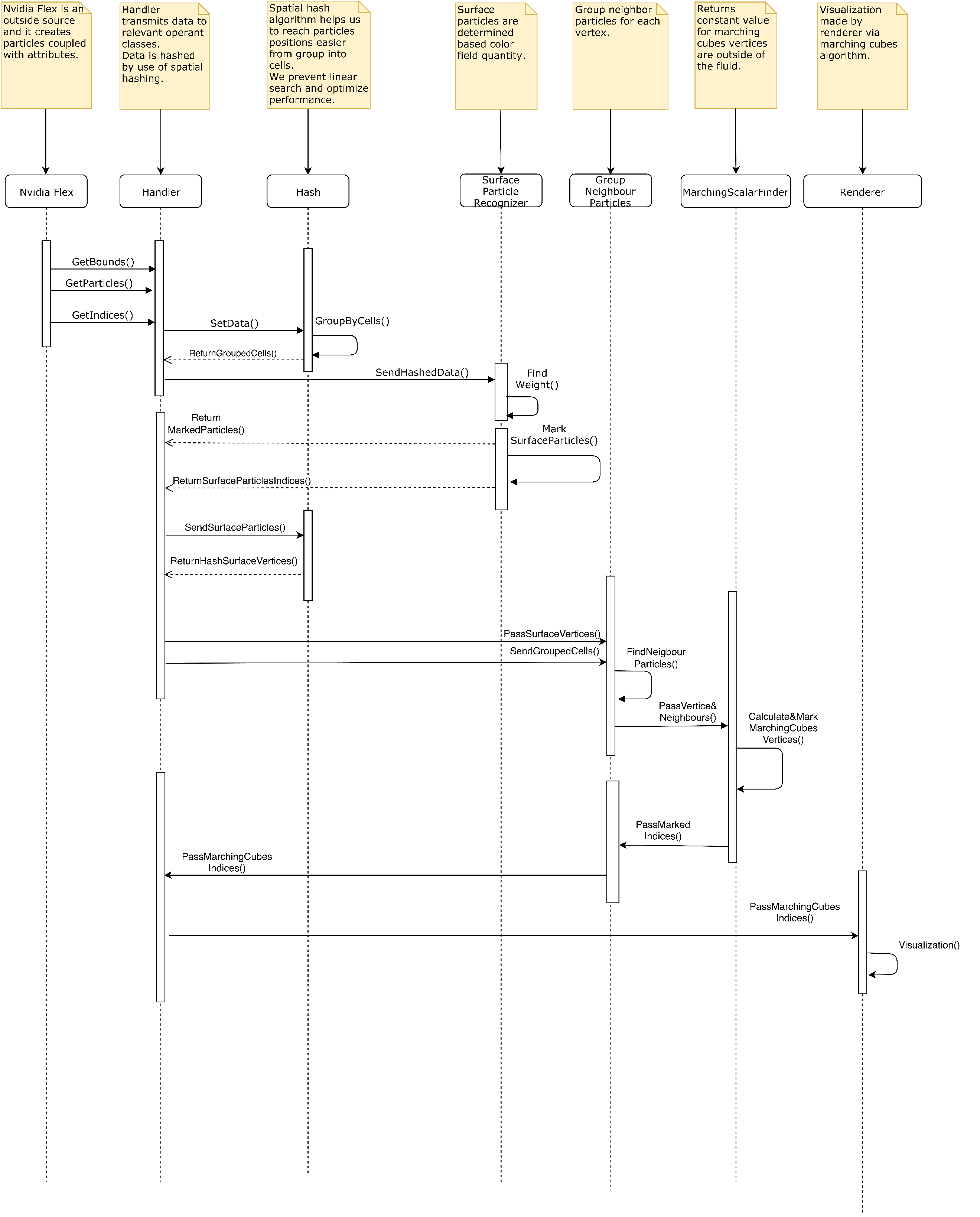
In this section POF system structure is described with UML diagrams. Diagrams are such as use case, sequence and package diagram.

***2.2.1 Use Case Diagram***

**Fig 1:** Use case diagram

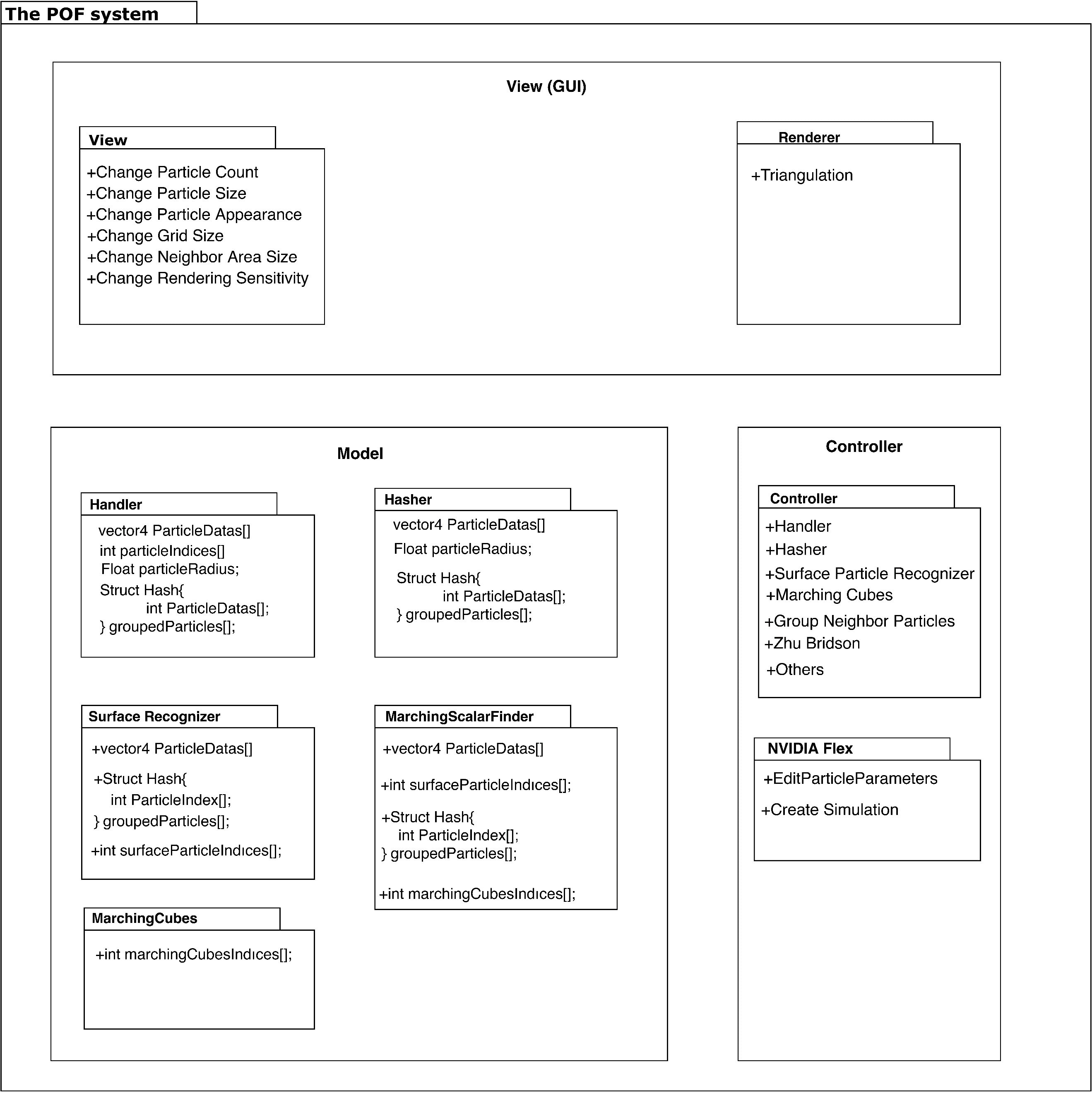
|  |  |
| --- | --- |
| **Title** | **Description** |
| **Calculate Scalar Field** | Calculates a constant value of a particle in given range. |
| **Change grid size** | Interval of grid size of axis aligned bounding box can be changed with this function. |
| **Change neighbour area size** | Neighbour 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 canbe edited. |
| **Change particle size** | Radiusof 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 sensitivitycan 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 andprocessed 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 of the POF system

**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 to store and reach it to particle and cell data. Surface particle recognizer finds the surface particles by computing color field quantity value. Group neighbor particles function finds a particle's neighbors 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 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 neighbor 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 neighbor 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 |

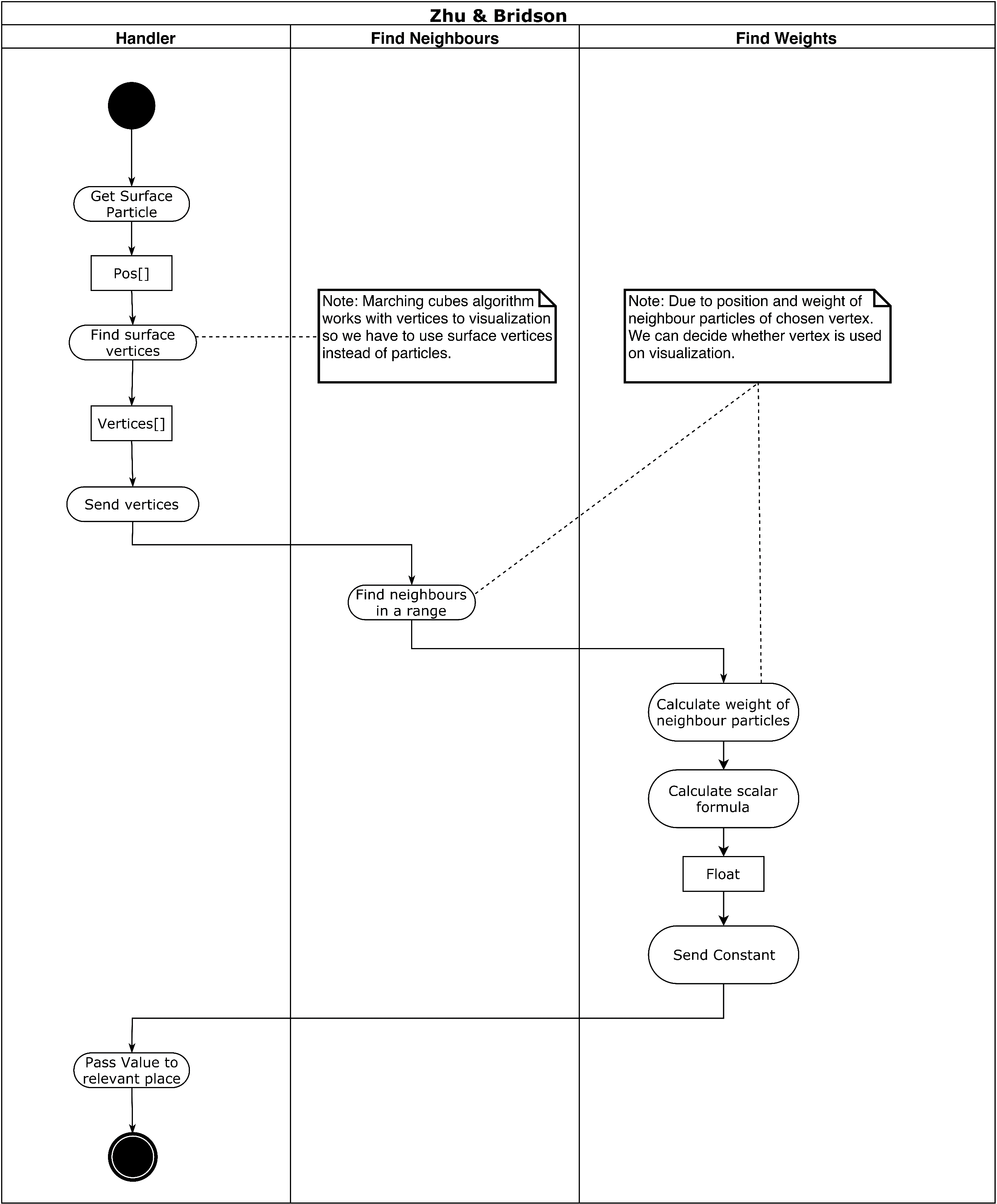
The main project made on the system:

|  |  |
| --- | --- |
| 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 computer system has satisfying performance only 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].

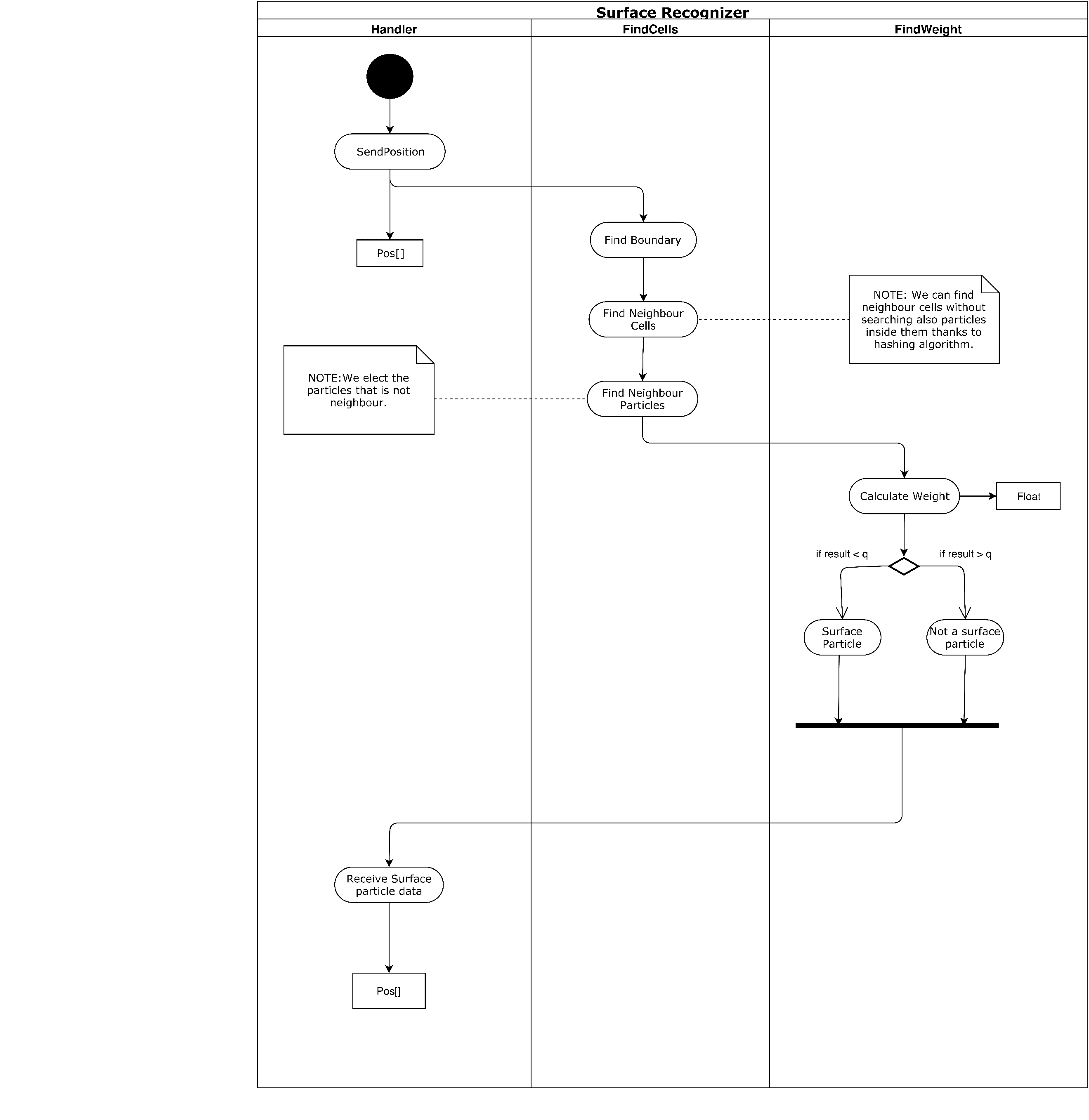
# 3. POF System Detailed Design

This section describes the small parts of the POF system. The functionality of these small parts is explained in activity diagrams.

***3.1 Activity Diagram of Zhu & Bridson***

**Fig 4:** Activity diagram of Zhu&Bridson

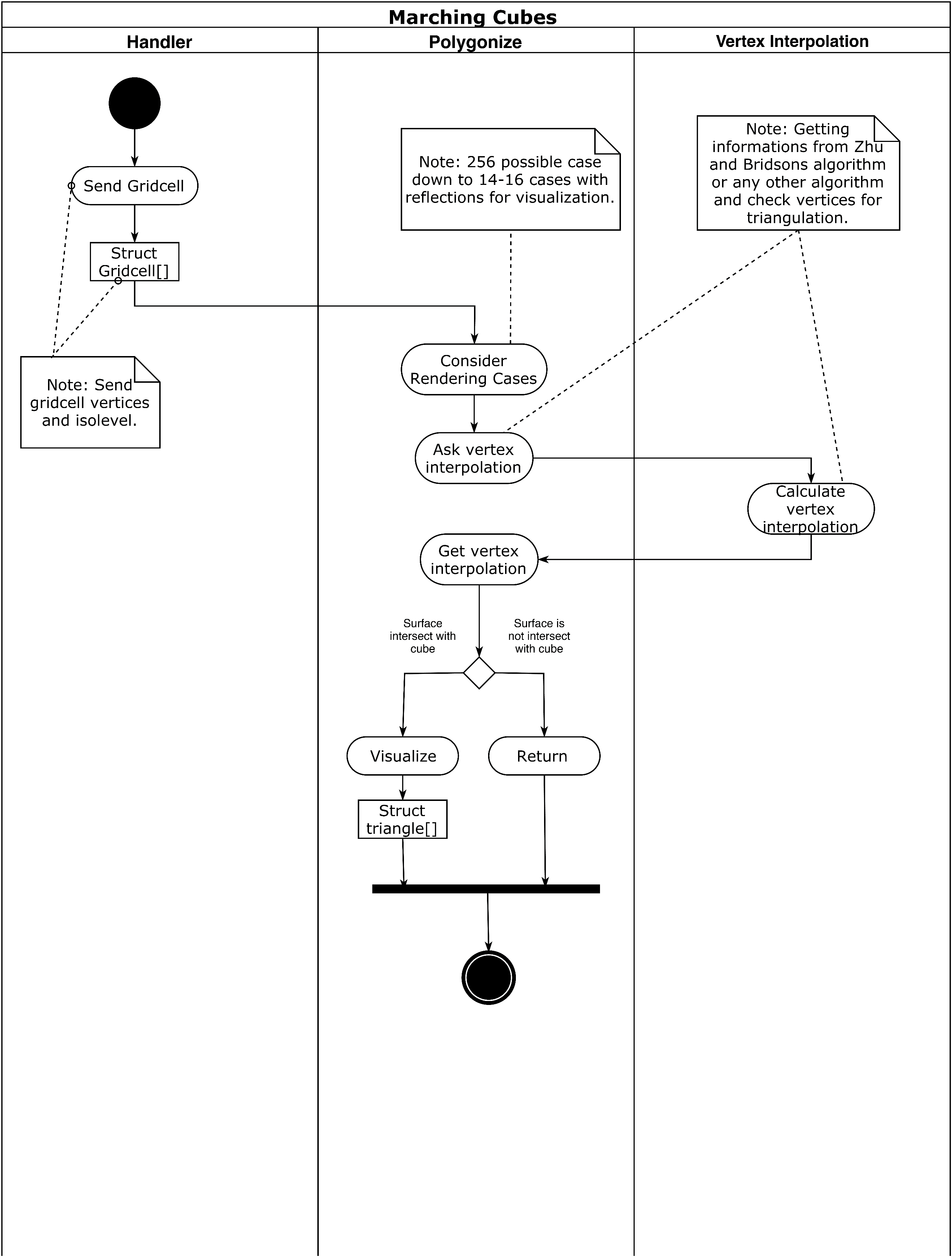
**Description:**Handler gets the surface particle from relevant sections. The handler receives the surface vertices and sends it to find neighbors section which is responsible to find neighbors of the particle in a specific range. Find neighbors section transmit neighbor particle data to find weight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

***3.2 Activity Diagram of Surface Recognizer***

**Fig 5:** Activity diagram of surface recognizer

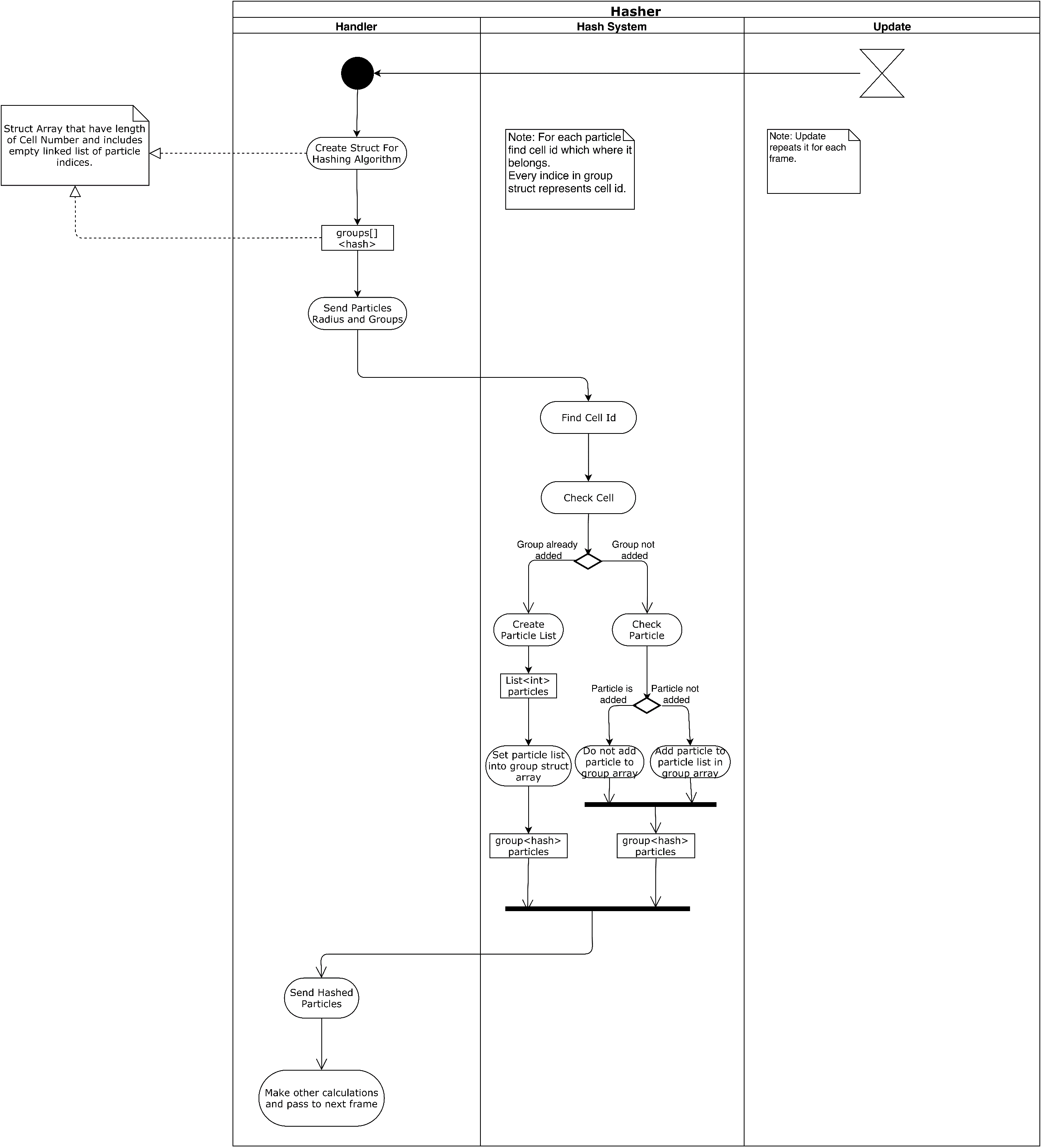
**Description:** Handlersendsposition data of the particle to find cell function. Find the neighbor cells section to find boundaries and neighbor cells. After that, find cells computes the neighbor particles and disqualify the particles that are not neighbor with each other and passes the data to find 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 much 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:** Activity diagram of marching cubes

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

**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

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 mentioned about various improvements on the POF system. The pros and cons of the algorithms are discussed in this section. The main criteria are performance and efficiency. Nevertheless, visualization quality is important along with memory and CPU usage is important. Different methods have been tried to get better visuals and to get performance improvements. This section aims to satisfy the discussion of the advantages and disadvantages of the methods and reasons why specific methods have selected will be explained.

***4.1 Marching Cubes***

Marching Cubes is an algorithm for rendering iso-surfaces in volumetric data. The basic notion is that we can define a voxel (cube) by the pixel values at the eight corners of the cube. If one or more pixels of a cube have values less than the user-specified iso-value, and one or more have values greater than this value, we know the voxel must contribute some component of the iso-surface.

***4.1.1 Linear Interpolation***

Linear interpolation is basically an estimation between two points, it is useful for creating new data points from discrete two data points.

The position that it cuts the edge will be linearly interpolated, the ratio of the length between the two vertices will be the same as the ratio of the iso-surface value to the values at the vertices of the grid cell.

***4.1.2 Calculating Normal***

While the triangulation part continuing, additionally we can compute the surface normal of every triangle. A surface normal is basically for each vertex, you can compute an average surface normal at that point by averaging the normal of all triangles sharing that vertex. At each vertex, you can use that normal to compute the lighting at that point. You can interpolate for every point between to determine how bright the surface is at that point. The result is the appearance of edges looks blurry without changing the actual geometry of the object.

***4.1.3 Mesh Limit***

The triangulation section creates many triangles and holds it as a single mesh. However, a single mesh can only consist of 65,000 triangles. In our project, we can easily exceed this limit. We have added a small bug fix part of the mesh part of the code. If mesh triangles exceed the limit, we create another mesh and fills it until all triangles finish.

***4.1.4 Grid Resolution***

One very desirable control when polygonising a field where the values are known or can be interpolated anywhere in space is the resolution of the sampling grid. This allows course or fine approximation to the iso-surface to be generated depending on the smoothness required and the processing power available to display the surface.

***4.1.5 Solenthaler Addition***

***4.2 Calculating Weight function (Zhu Bridson vs other papers.)***

// Zhu and bridson niye daha tutarlı diğer kağıtlarla karşılaştır.

***4.3 Hash***

// Hash ve hashsiz farkını anlat kişe celleri aabb ile nasıl sınırladık onu anlat.

The comparison tables with details will be added in final report.

# 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. **[BP94]** Paul Bourke 1994, Marching Cubes, viewed 4 March 2020, **<**[**http://paulbourke.net/geometry/polygonise/**](http://paulbourke.net/geometry/polygonise/)**>**
7. **///// \*\*\*\*\*\*ADD Solenthaler \*\*\*\*\*\***