**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 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

[Revision History 2](#_Toc501993332)

[Table of Contents 3](#_Toc501993333)

[1. Introduction 4](#_Toc501993334)

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

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

[2.2. POF System Structure 4](#_Toc501993337)

[2.3. POF Environment 4](#_Toc501993338)

[3. POF System Detailed Design: 4](#_Toc501993339)

[4. Testing Design 4](#_Toc501993340)

[References 5](#_Toc501993341)

# 1. Introduction

The purpose of the 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.

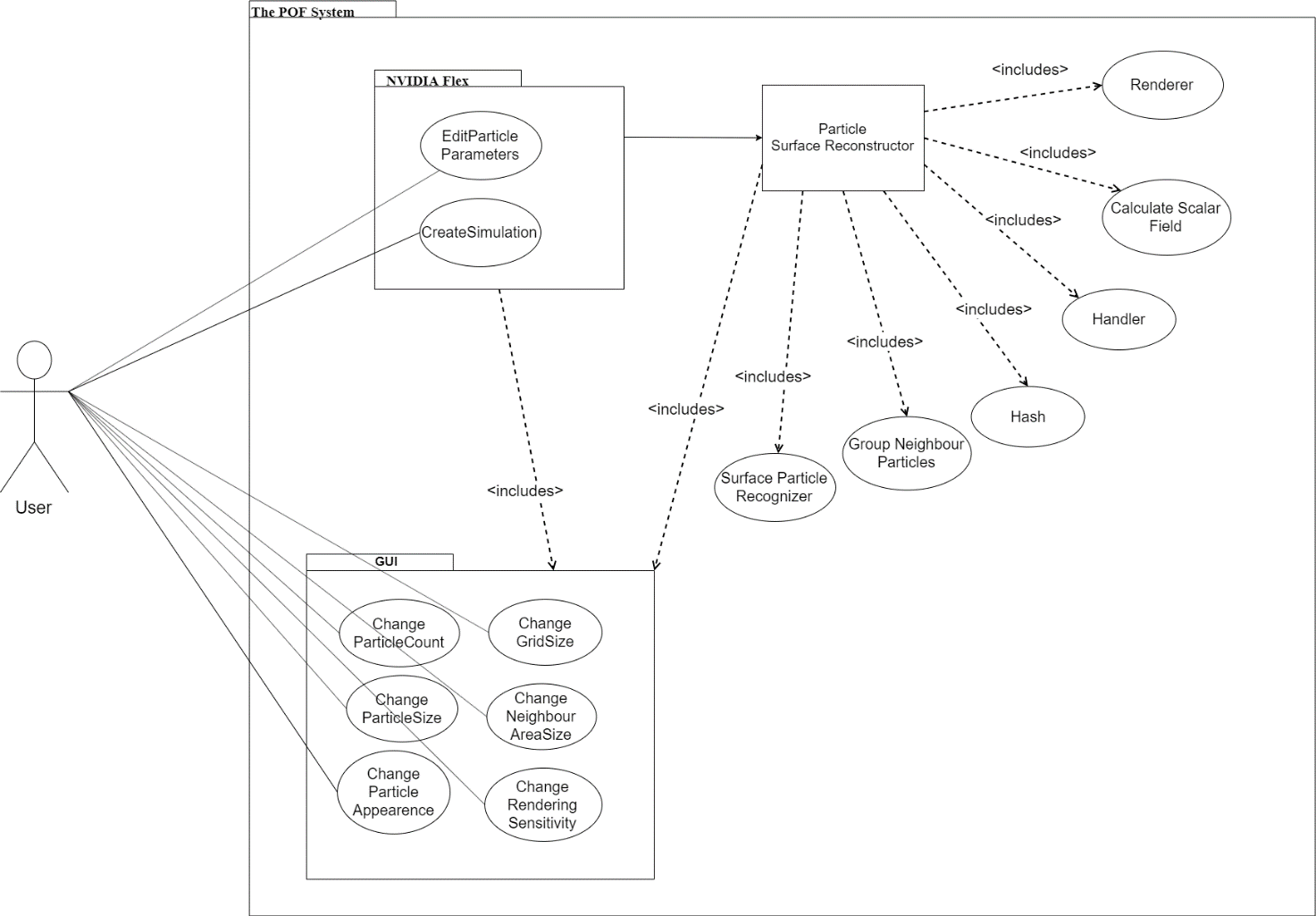
# 2. POF Computer System High Level Design

# 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 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

**2.2.1 Use Case Diagram**

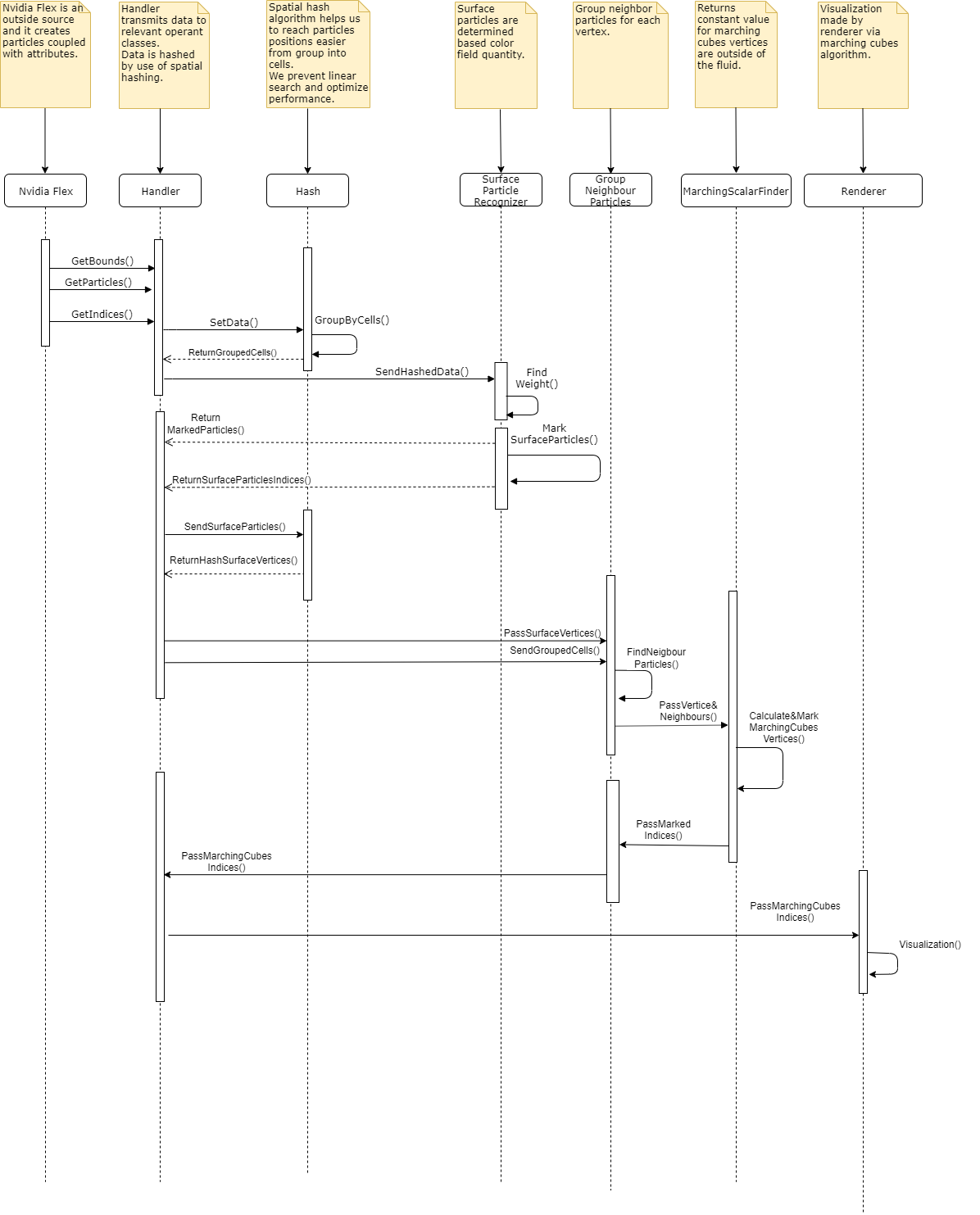


**Fig 1:** Use case diagram

|  |  |
| --- | --- |
| **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 neighbor area size** | Neighbor particle range of volume can be changed which affects the visualization of particles and changes the shape. |
| **Change particle appearance** | Particle color, 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**

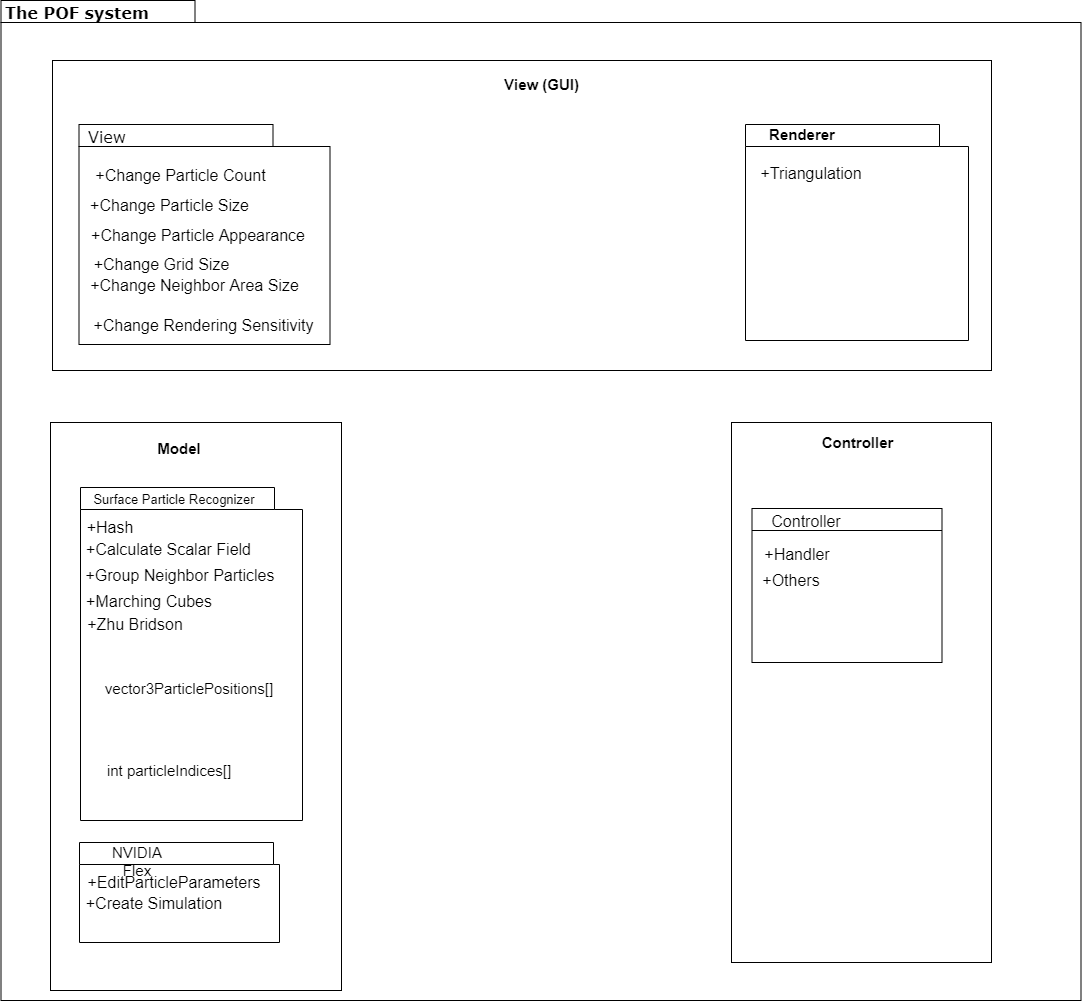
****

**Fig 2:** Sequence diagram

**Description:**

Handler manages data transfer between other sections. If any data have 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 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 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 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. [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

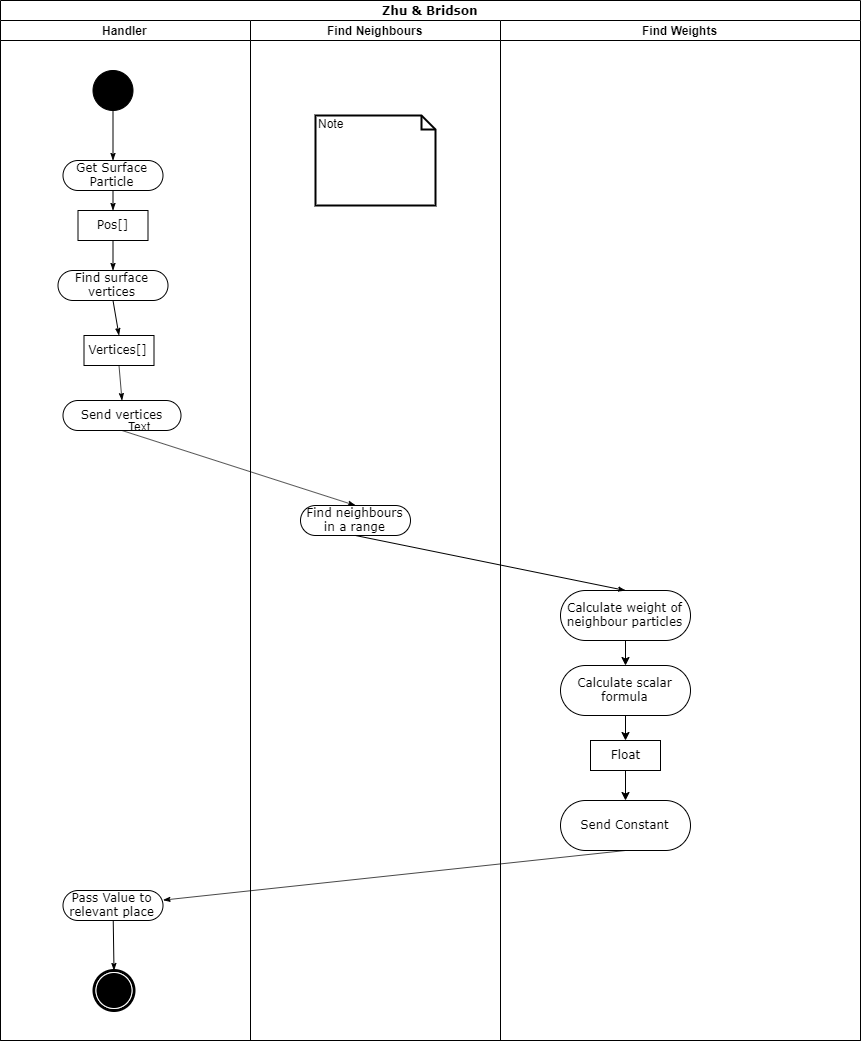
The main project made on the system:

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

# 3. POF System Detailed Design:

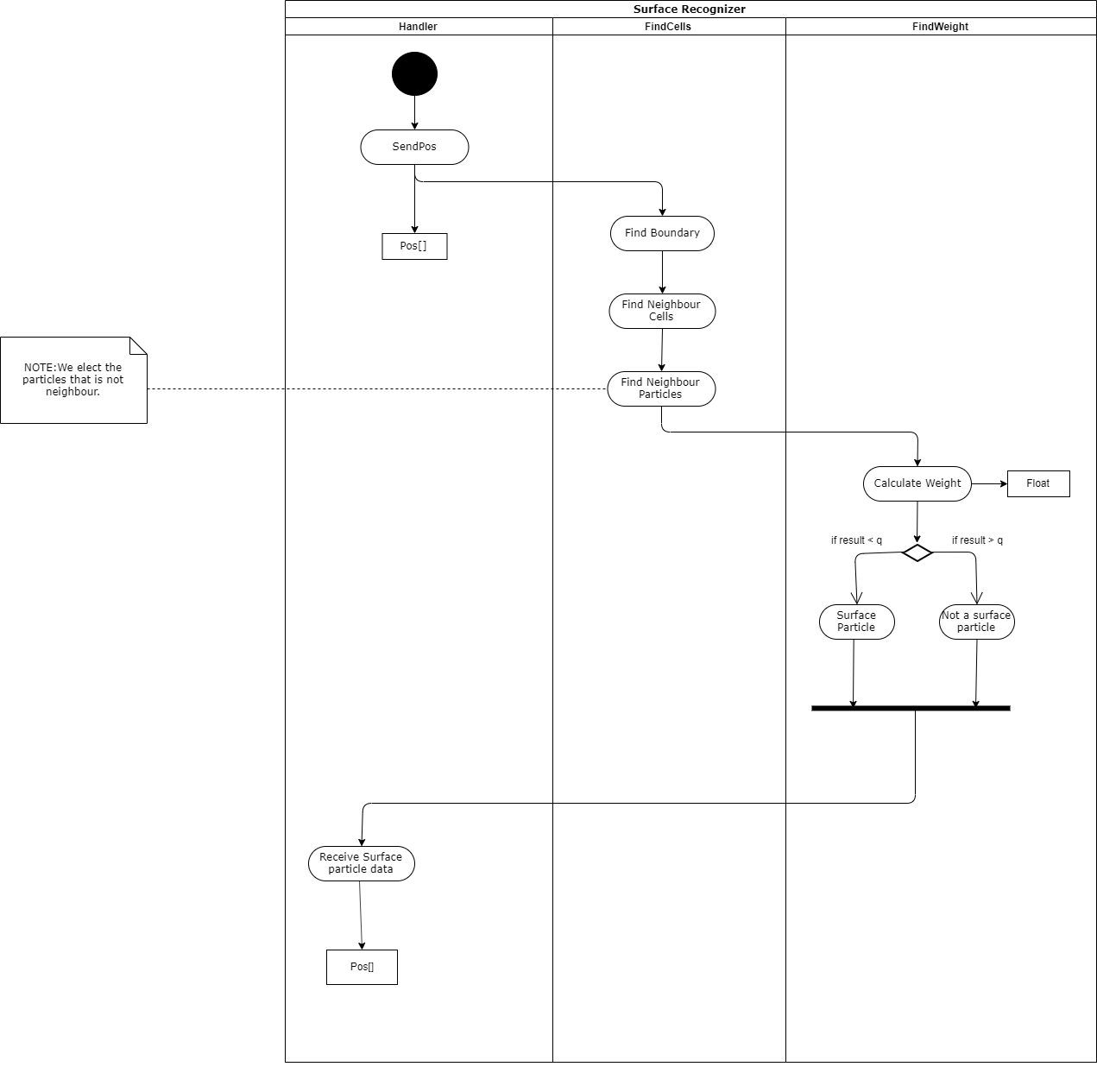
**3.1 Activity Diagram of Zhu & Bridson**

****

**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. Weight is used for calculating a scalar value. Find weight send constant value to the handler. Handler knows where to send specific data.

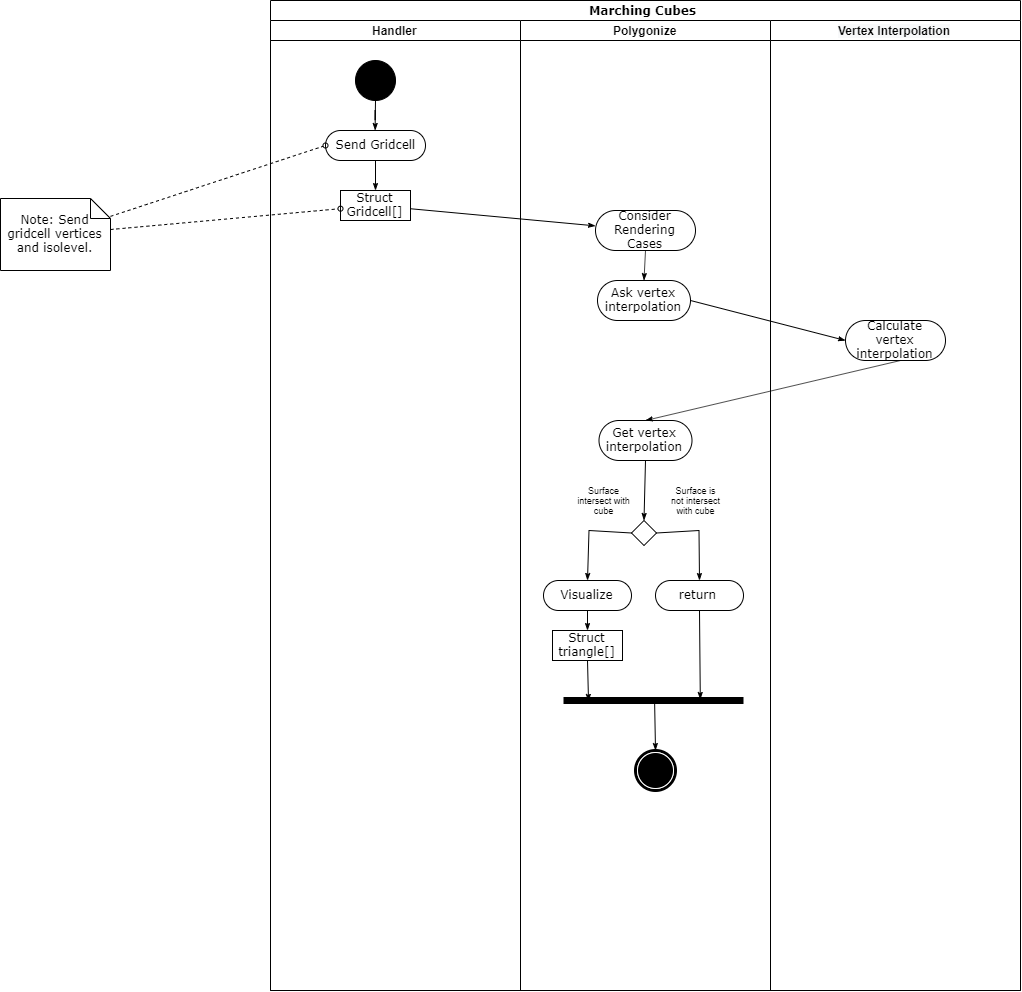
**3.2 Activity Diagram of Surface Recognizer**

****

**Fig 5:** Surface recognizer activity diagram

**Description:** Handlersendsposition data of the particle to find cell function. Find neighbour cells 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’, (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.

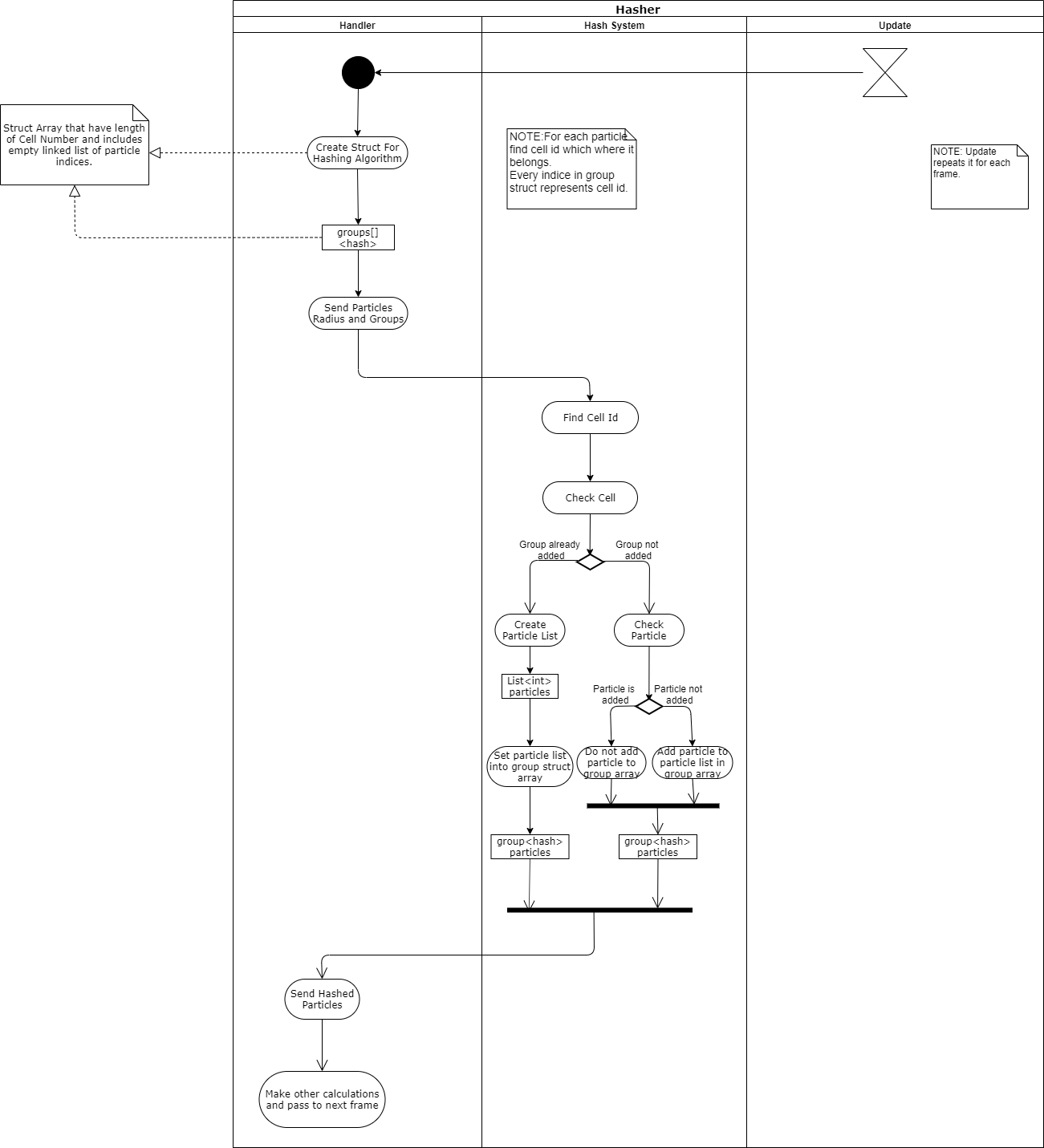
**3.3 Activity Diagram of Marching Cubes**

****

**Fig 6:** Marching cubes activity diagram

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

**Table of figures**

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

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

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

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

[**Fig 5:** Surface recognizer activity diagram 11](#_Toc501993332)

[**Fig 6:** Marching cubes activity diagram 12](#_Toc501993333)

[**Fig 7:** Hasher activity diagram 13](#_Toc501993332)

# References

1. Requirement Specification Document revision 2.0 (RSD 2.0)
2. Use case and sequence diagrams in RSD 2.0
3. Final Report revision 1.0
4. **[ZB05]** Zhu, Y., & Bridson, R. (2005). Animating sand as a fluid. (New York, NY, USA, 2005) ACM Trans. Graph., 24, 965-972.

/////////////////////////////////////////////////////////////

Other references to additional documents, like other internal organizational documents,software project management documents, software design tool documents, etc.

References to additional bibliographic sources, like professional books, textbooks, handbooks, patents, standards, technical reports, journal/conference papers, etc.