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.

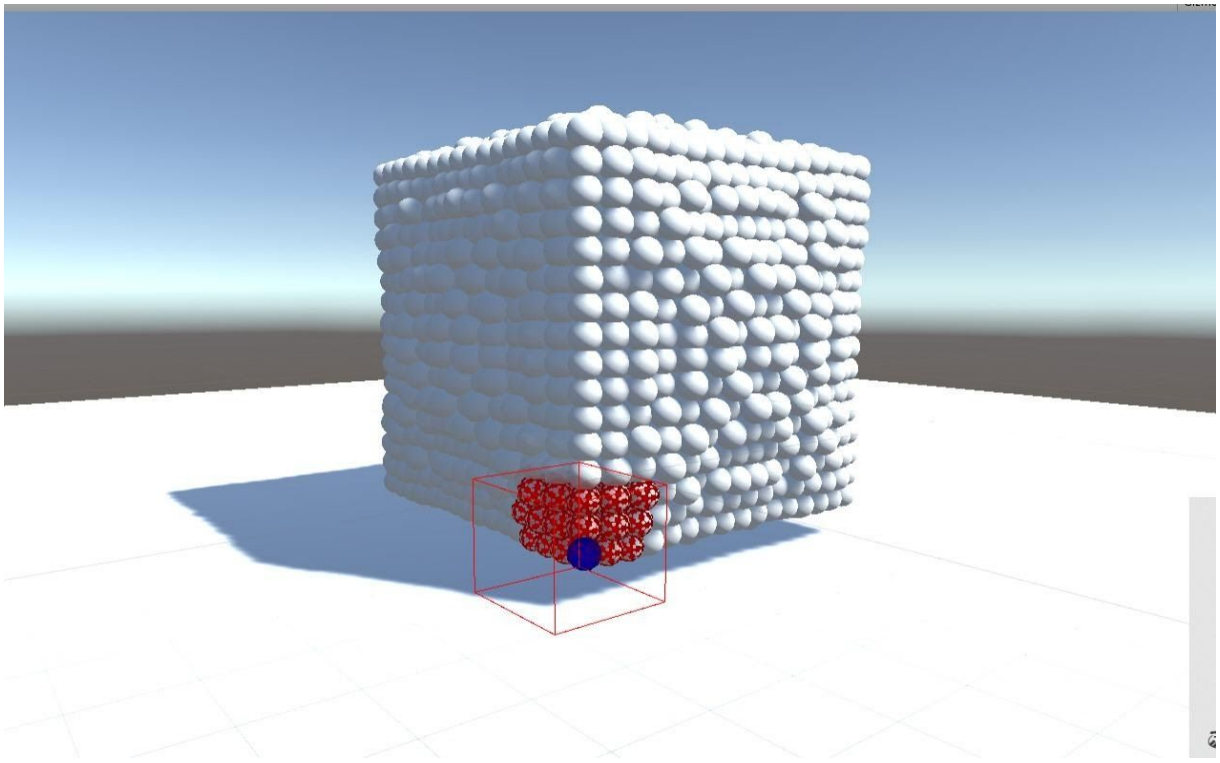

Figure 16: Particle and its neighbours in the cell.

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.

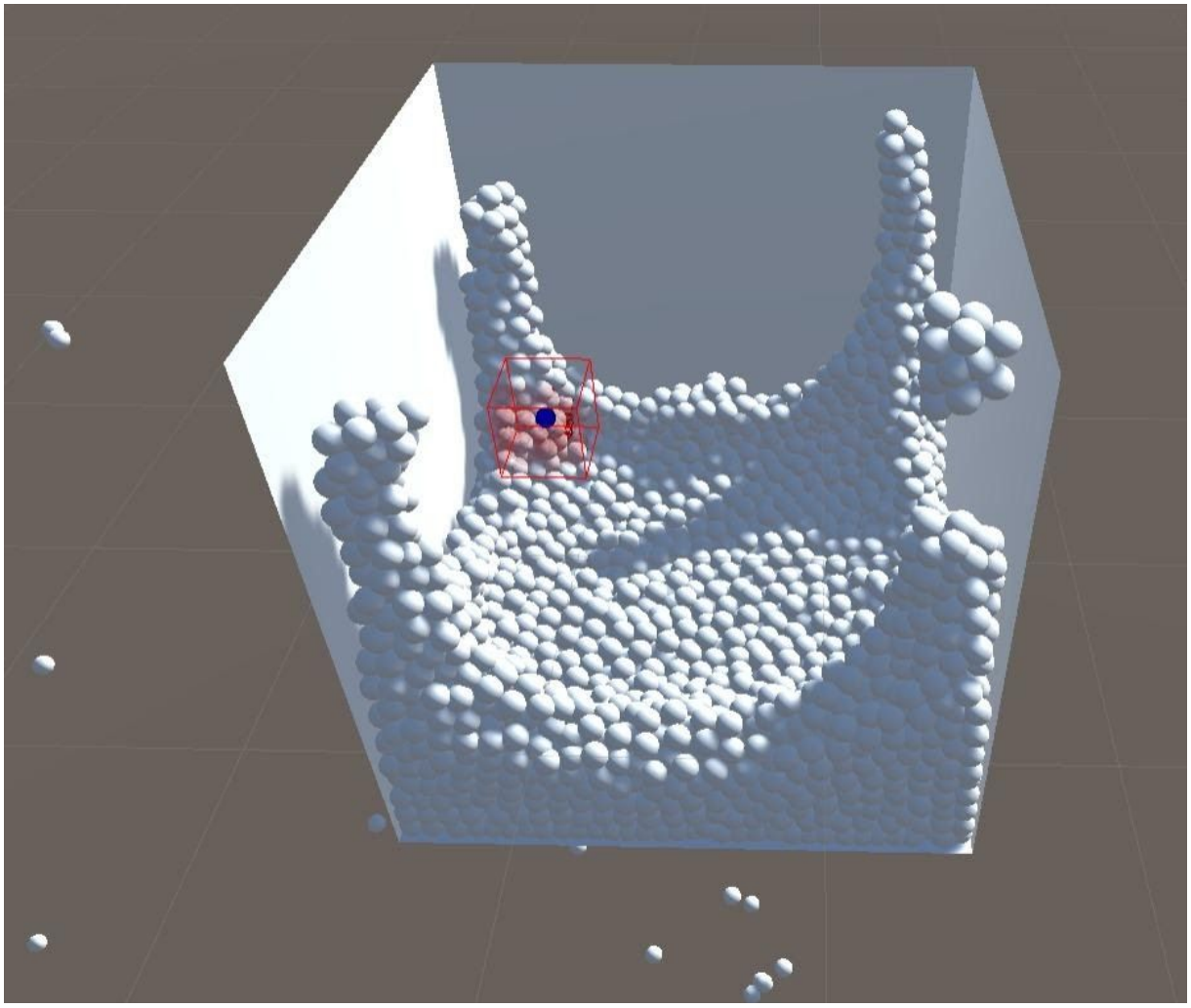

Figure 17: Tracking particle in a cell.

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.

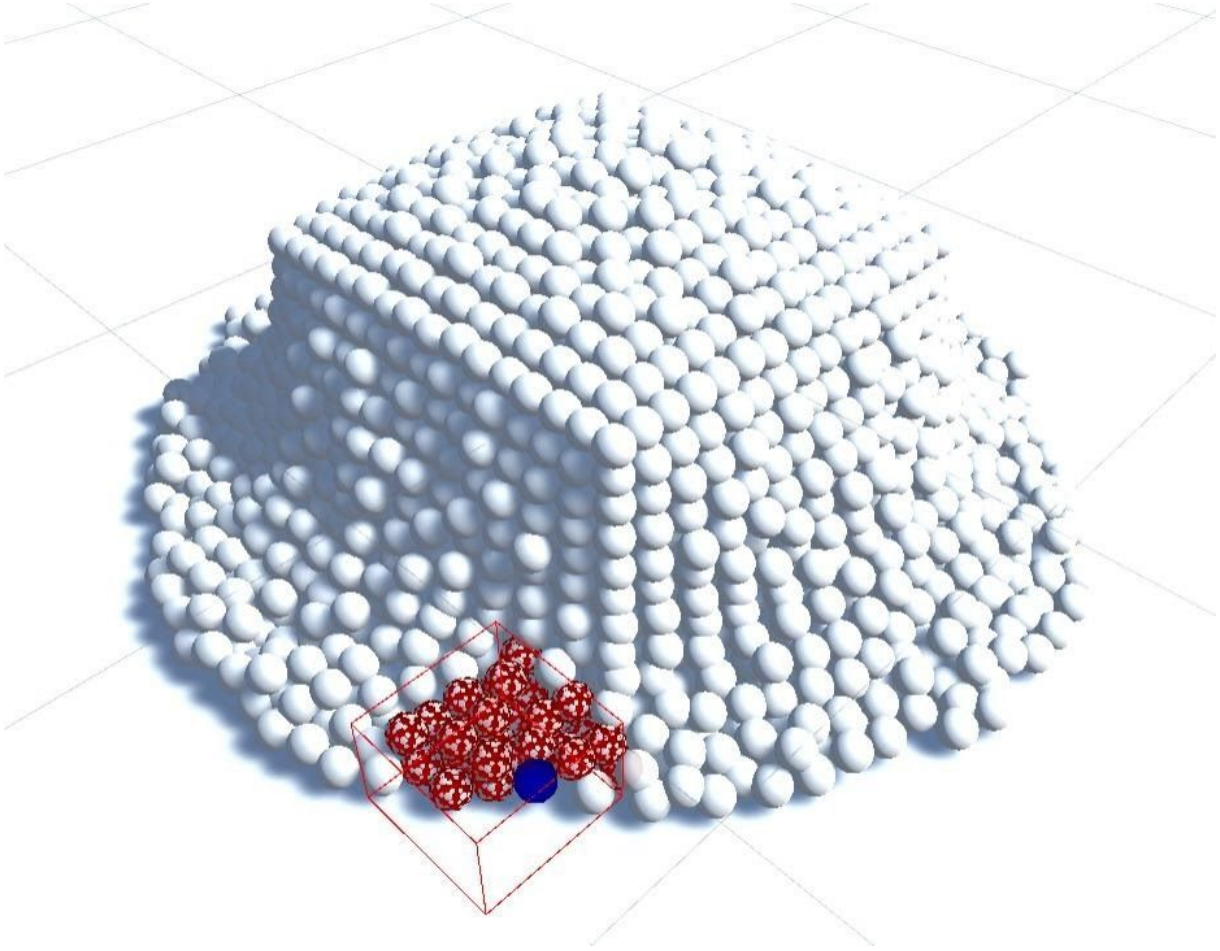

Figure 18: Tracking particle and its neighbours in a cell.

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.

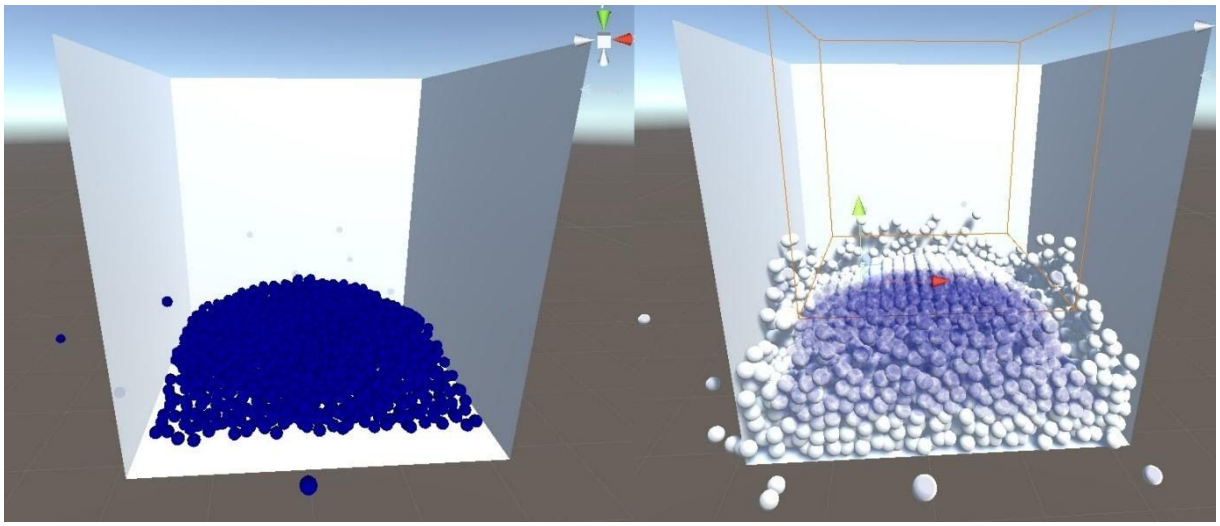

Figure 19: Inside blue particles vs all particles.

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.

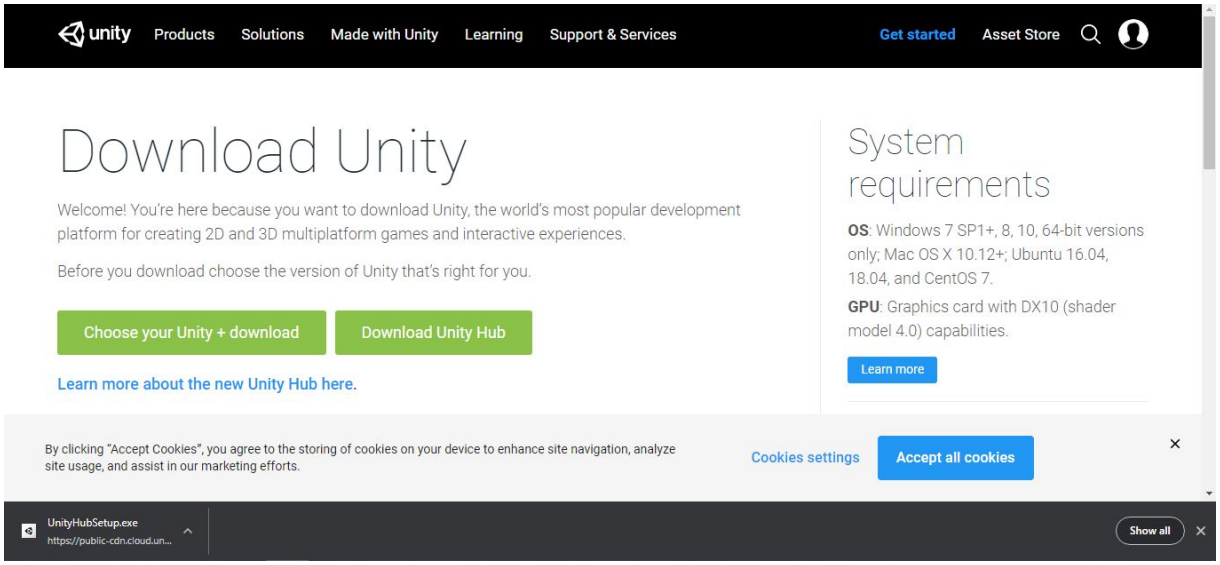

Figure 20: Download Unity Hub from website.

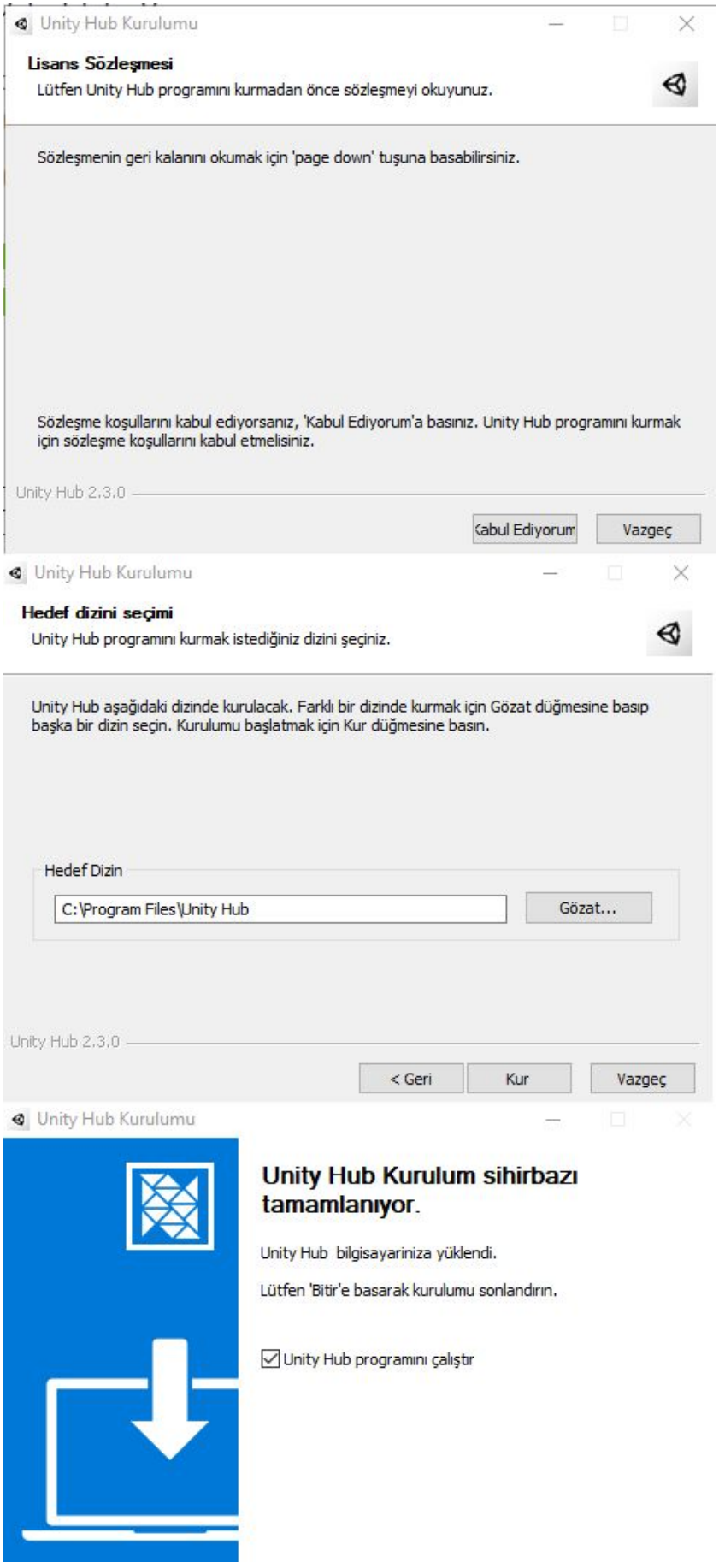

Figure 21: Install Unity Hub program.

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.

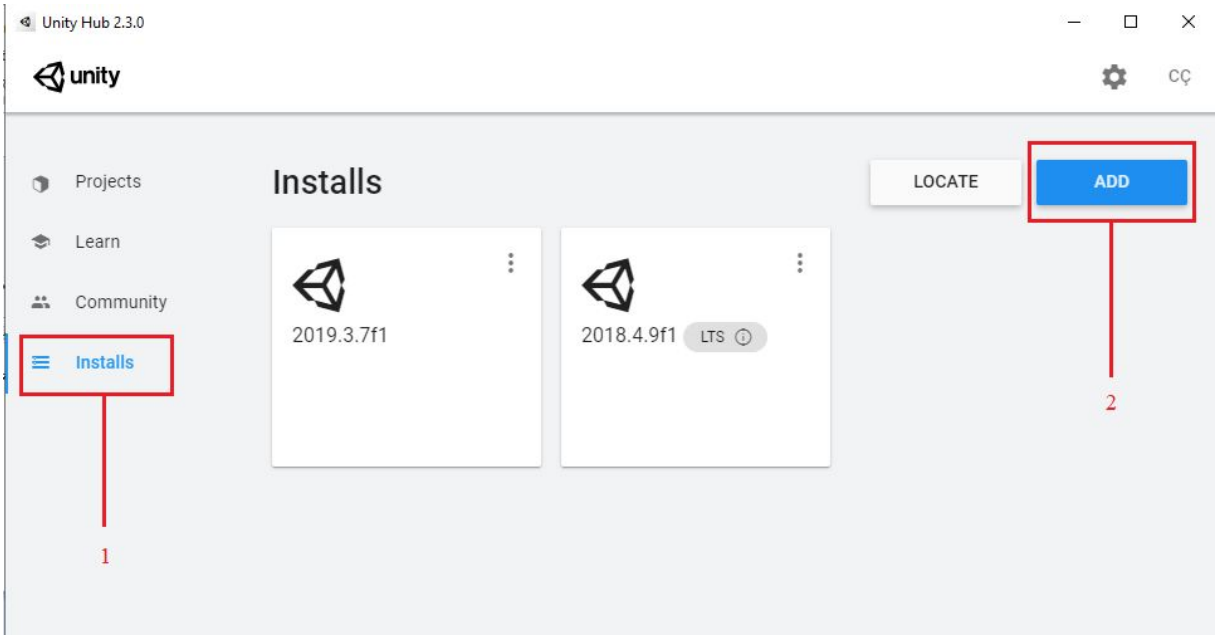

Figure 22: Go to installations window in Unity Hub.

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.

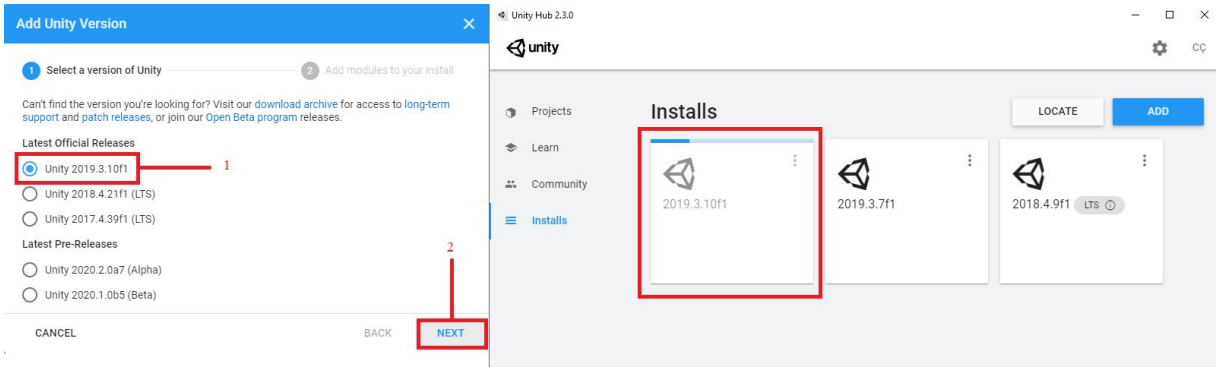

Figure 23: Select Unity version and install.

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.

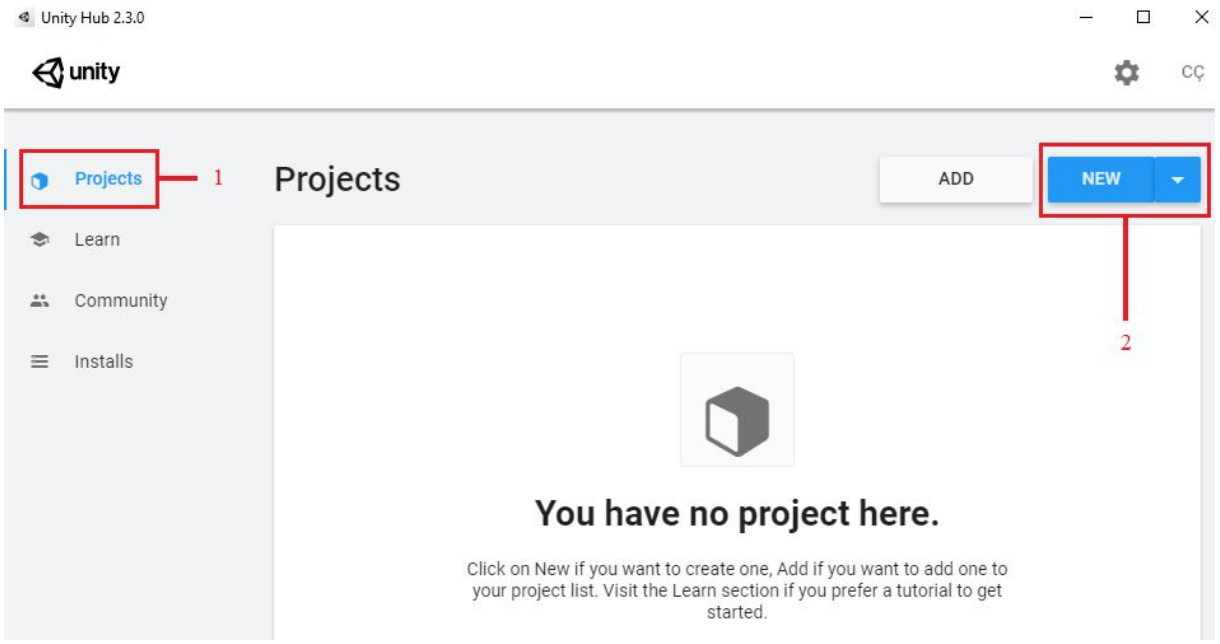

Figure 24: Create a new empty project in Unity.

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

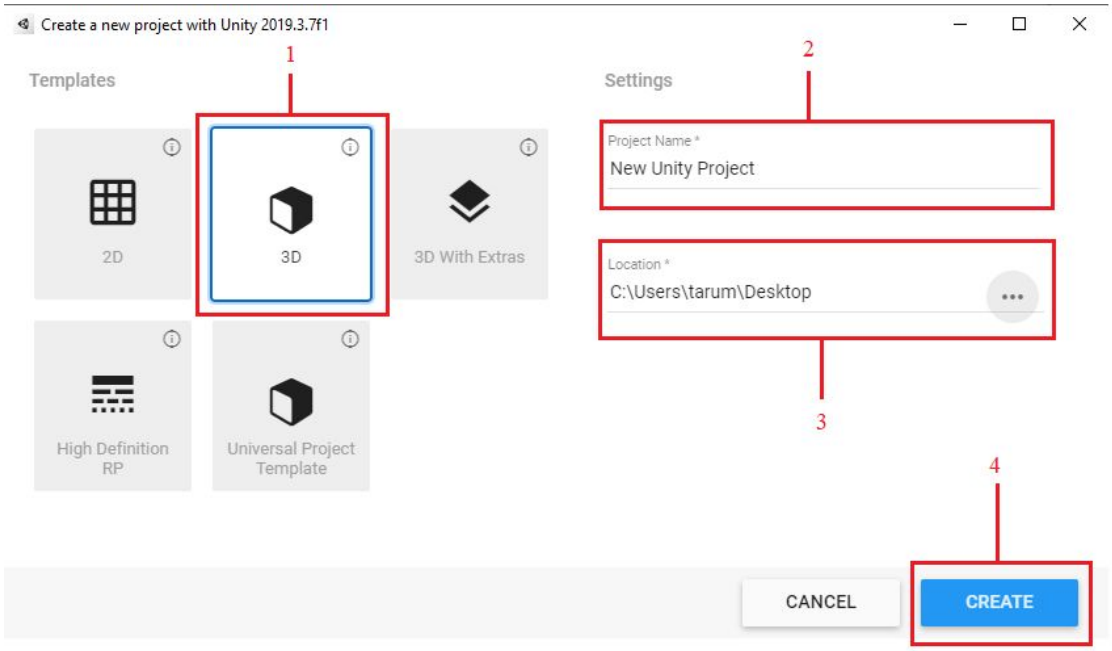

Figure 25: New Unity project configurations.

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.

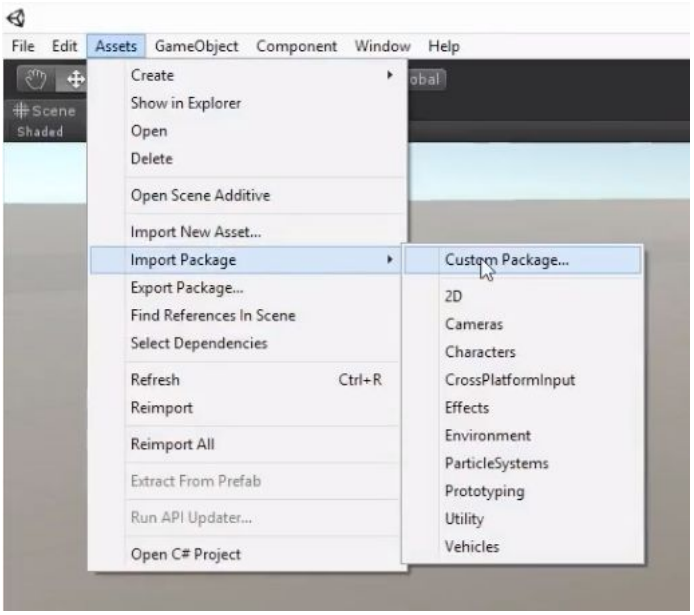

Figure 26: Select the custom package in Unity.

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

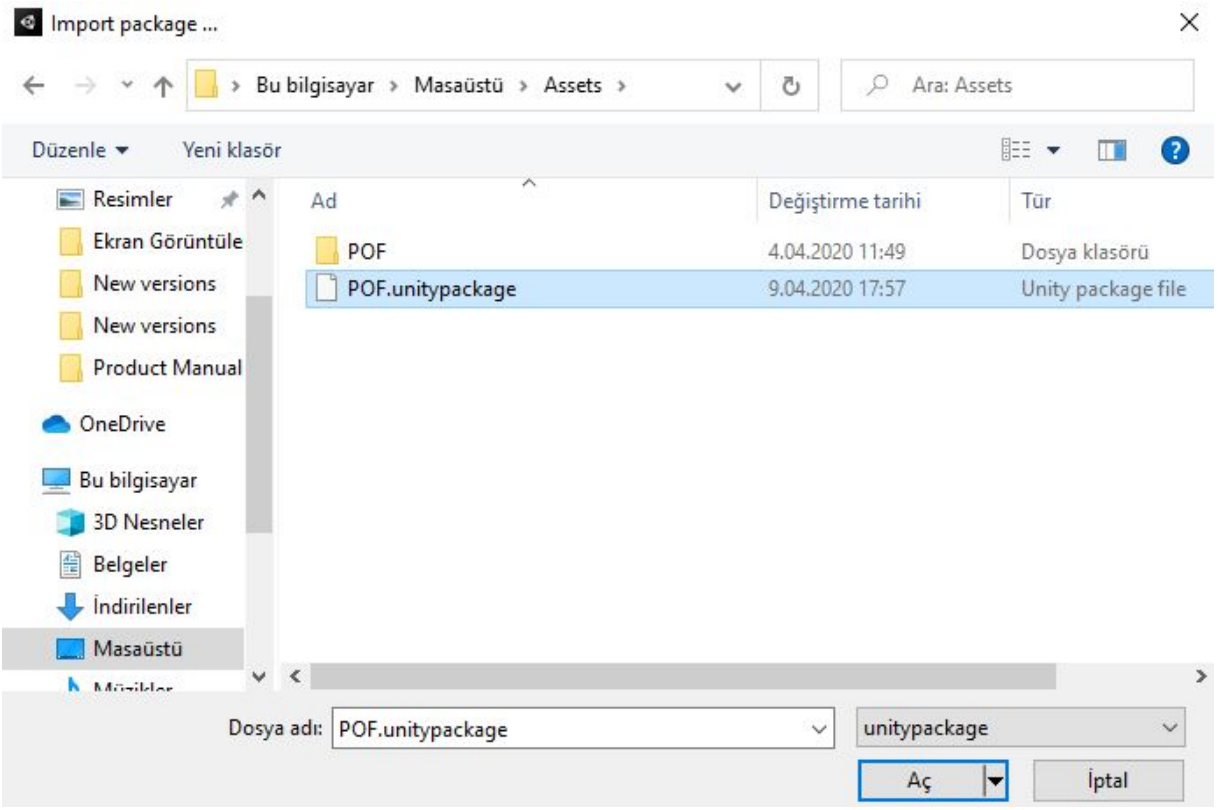

Figure 27: Find POF unity package in file explorer.

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.

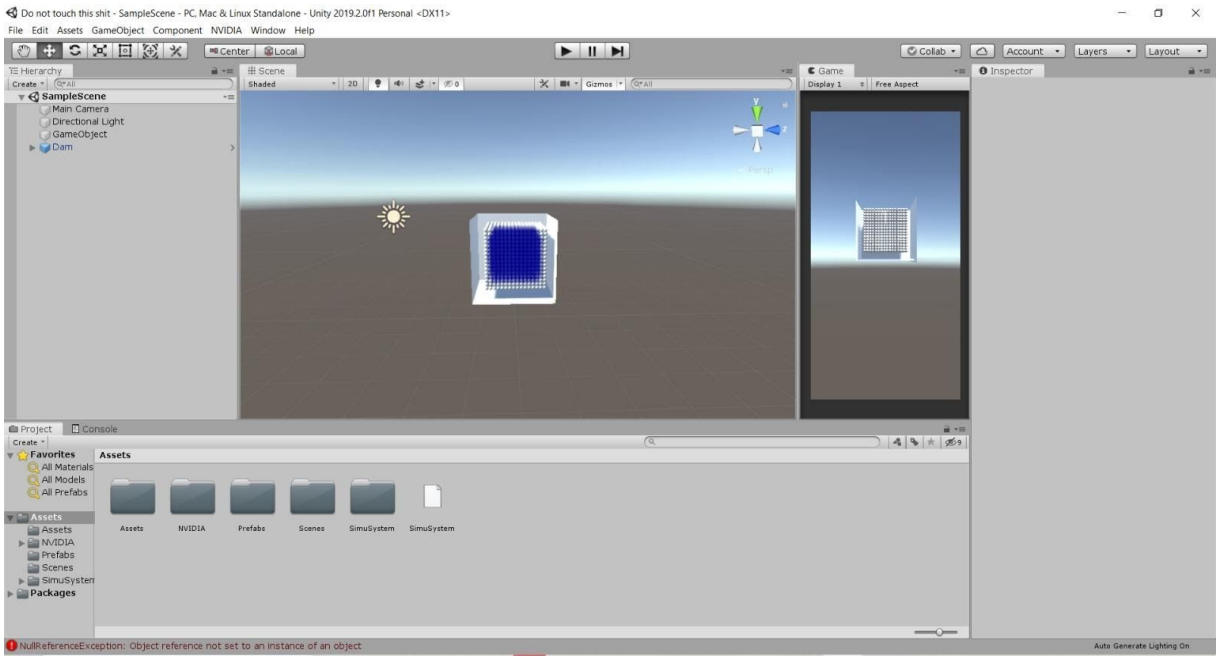

Figure 29: Sample scene in the Unity.

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.

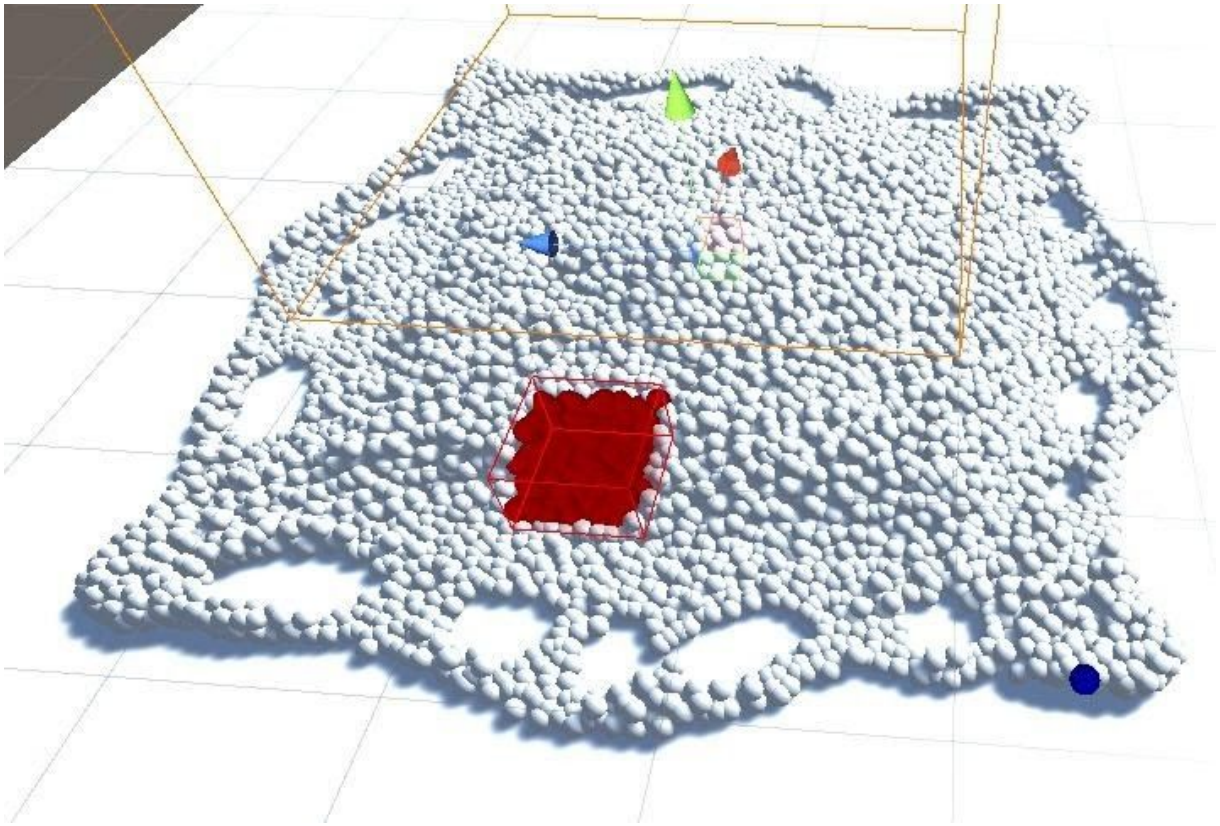

Figure 30: Screenshots of the scene mode while simulation is working-1.

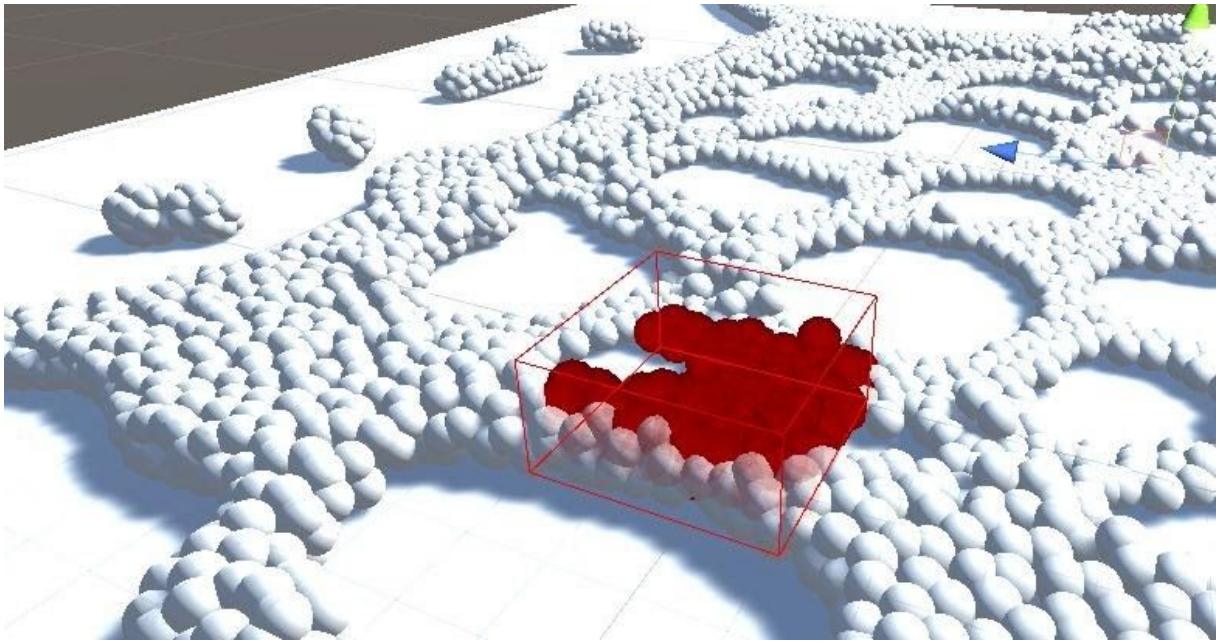

Figure 31: Screenshots of the scene mode while the simulation is working-2.

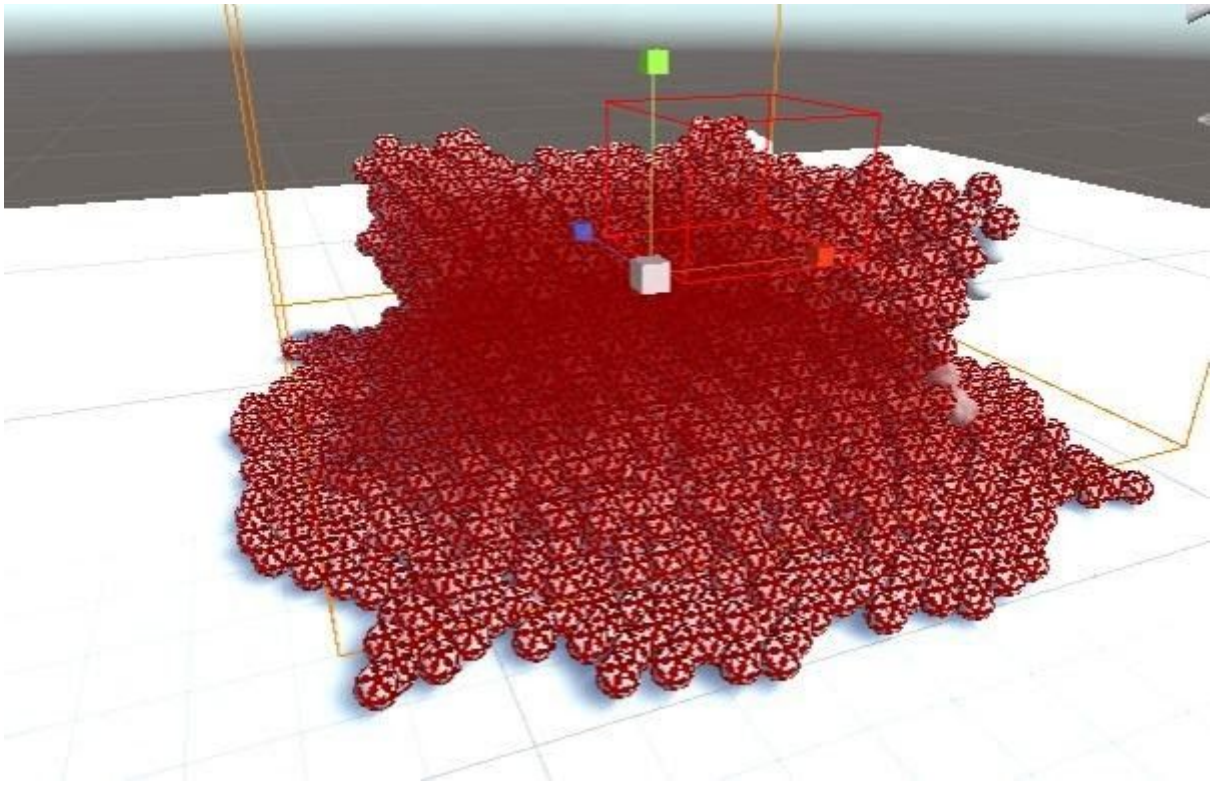

Figure 32: Screenshots of the scene mode while the simulation is working-3.

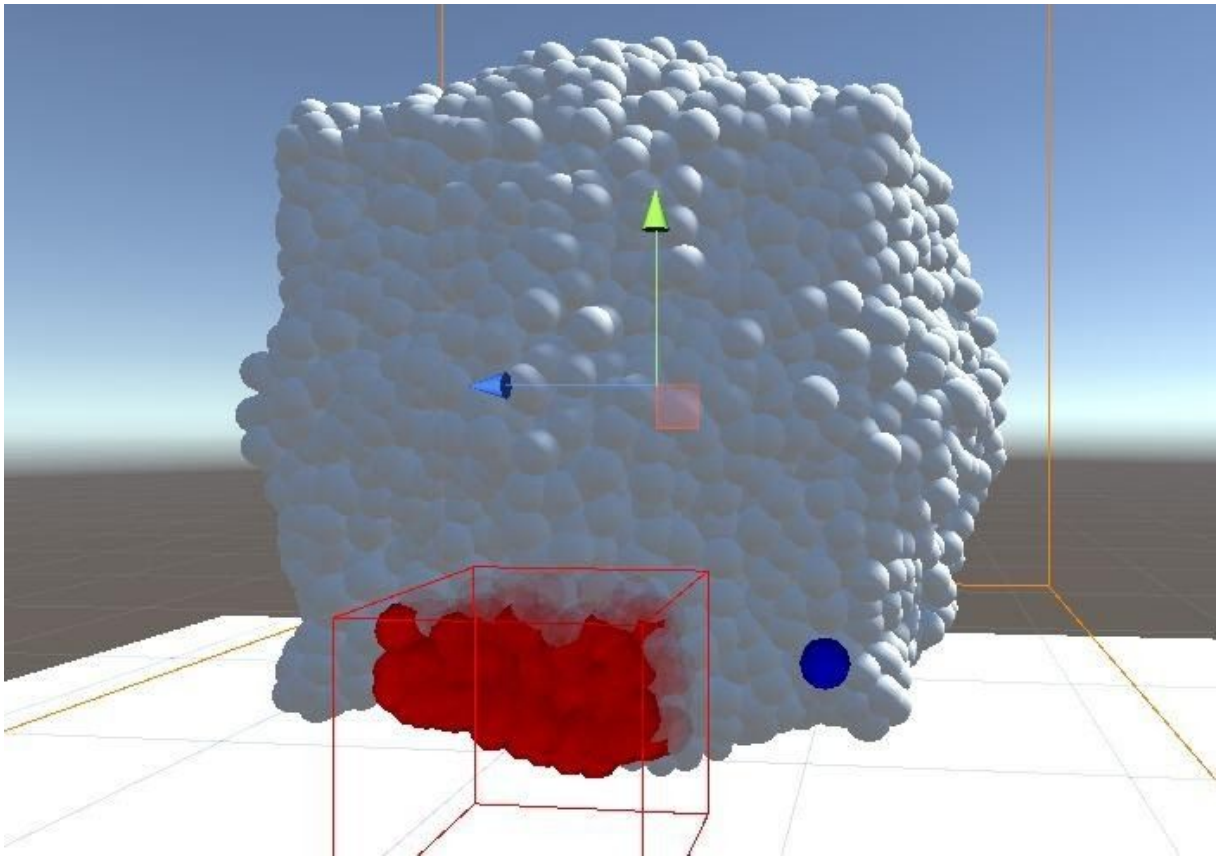

Figure 33: Screenshots of the scene mode while the simulation is working-4.

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.

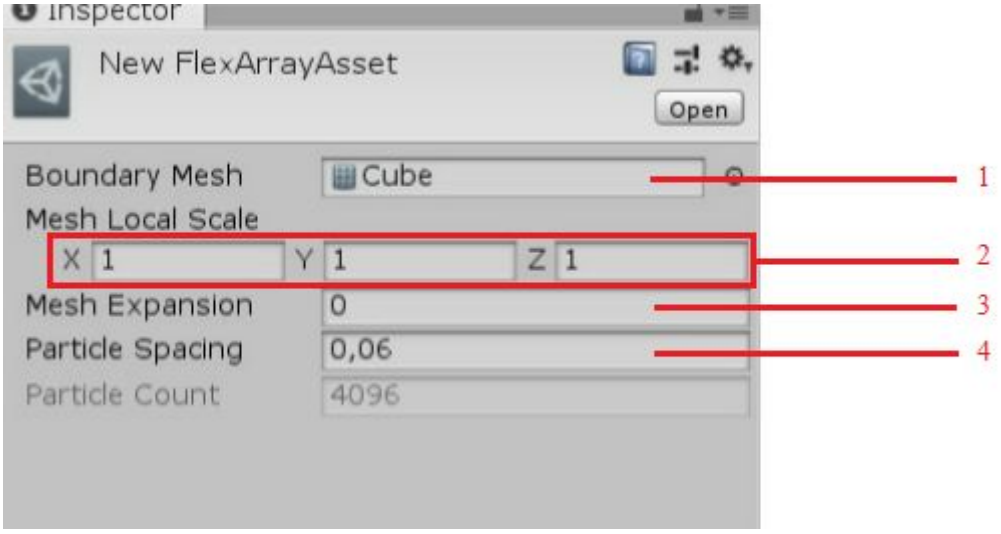

Figure 34: Description of Flex array asset parameters.

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.

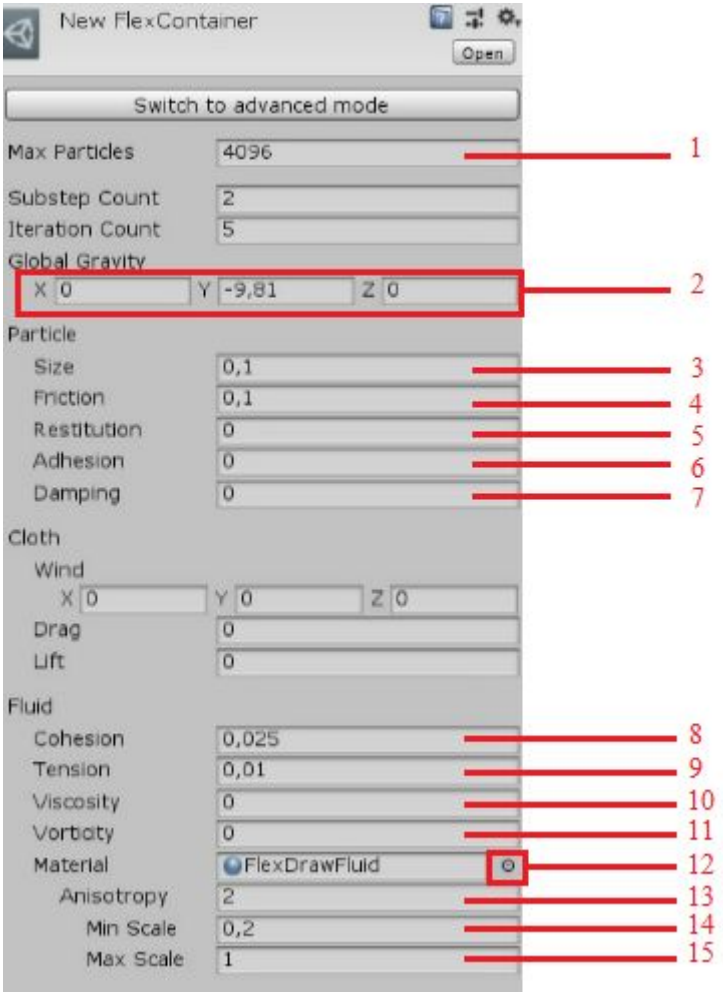

Figure 35: Description of Flex Container parameters.

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.

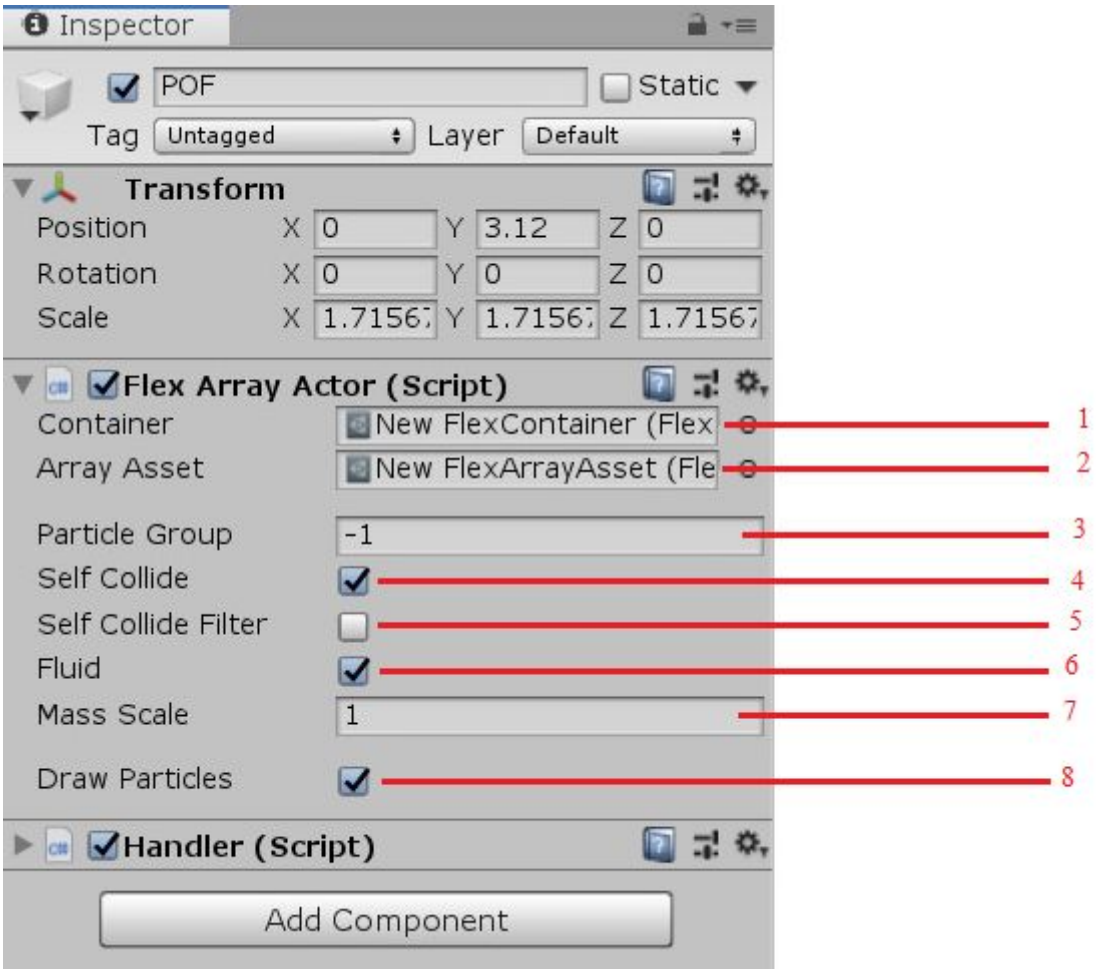

Figure 36: Description of POF inspector parameters.

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>