****

**YAŞAR UNIVERSITY**

**FACULTY OF ENGINEERING**

**DEPARTMENT OF COMPUTER ENGINEERING**

**COMP4920 Senior Design Project II, Spring 2020**

**Advisor: Gizem Kayar**

**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

[Revision History 2](#_Toc36726022)

[Table of Contents 3](#_Toc36726023)

[1. Introduction 4](#_Toc36726024)

[2. POF Software Subsystem Implementation 4](#_Toc36726025)

[2.1. Source Code and Executable Organization 4](#_Toc36726027)

[2.2. Software DevelopmentTools 4](#_Toc36726028)

[2.2.1. Unity 4](#_Toc36726027)

[2.2.2. Visual Studio 2017 4](#_Toc36726027)

[2.2.3. Github 5](#_Toc36726027)

[2.2.4. Gitkraken 5](#_Toc36726027)

[2.2.5. NVIDIA FleX 5](#_Toc36726027)

[2.3. Hardware and System Software Platform 6](#_Toc36726029)

[3. POF Software Testing 6](#_Toc36726030)

[3.1. Testing of Hash System 7](#_Toc36726030)

[3.2. Testing of Surface Recognizer 9](#_Toc36726030)

[4. POF system Installation, Configuration and Operation 10](#_Toc36726031)

[4.1. NVIDIA Flex installation and set up 10](#_Toc36726031)

[4.2. POF installation and set up 17](#_Toc36726031)

[4.3. Learning the basics of POF! 19](#_Toc36726031)

[4.3.1. NVIDIA Flex inspector settings 19](#_Toc36726031)

[4.3.1. POF inspector settings 22](#_Toc36726031)

[References 23](#_Toc36726032)

**List of Figures**

Fig 1: Particle and its neighbours in the cell. 8

Fig 2: Tracking particle in a cell 9

Fig 3: Tracking particle and its neighbours in a cell. 10

[Fig 4: Nvidia Flex in Unity asset store](#_Toc501993332) 11

[Fig 5: Import Unity package window](#_Toc501993332) 12

[Fig 6: Creating Flex container in Unity](#_Toc501993332) 12

[Fig 7: Flex container table](#_Toc501993332) 13

[Fig 8: Assigning Flex container to the Flex Array Actor table](#_Toc501993332) 14

[Fig 9: Make right click and select create](#_Toc501993332) 14

[Fig 10: Select Flex Array Asset](#_Toc501993332) 14

[Fig 11: Select Boundary Mesh](#_Toc501993332) 15

[Fig 12: Creating Flex Source Asset](#_Toc501993332) 15

[Fig 13: Selecting surface mesh of flex source asset](#_Toc501993332) 16

[Fig 14: Surface mesh type menu](#_Toc501993332) 16

[Fig 15: Creating Flex Source Actor and adding component](#_Toc501993332) 17

[Fig 16: NVIDIA sidebar in Unity](#_Toc501993332) 17

[Fig 17: Select custom package in Unity](#_Toc501993332) 18

[Fig 18: Find POF unity package in file explorer](#_Toc501993332) 19

[Fig 19: Import package file window](#_Toc501993332) 19

[Fig 20: POF system set up on your unity project 20](#_Toc501993332)

[Fig 21: Description of Flex Array Asset parameters](#_Toc501993332) 21

[Fig 22: Description of Flex Container parameters. 22](#_Toc501993332)

[Fig 23: Description of POF inspector parameters 23](#_Toc501993332)

**List of Tables**

[Table 1: Minimum Requirements of software Platform](#_Toc501993332) 6

**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

***2.1.1 Handler Organization***

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.

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.

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***

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.

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.

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.

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.

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

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.

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.

FindDimensionalIntervalNum function calculates how many cells exist in every dimension.

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.

In FindBoundary 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.

In FindNeighbourParticles function is calls neighbour cells function and adds these cell’s particles to the dynamic list

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.

This function(?) 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.

In findSurfaceParticles function, we check if particles around a specific particle are out of the range of volume, if particle out of this range, we assign it back in the range limit again. We do this operation for every three-dimension.

In findNeighbourArea function, we determine a specific volume range to find neighbour particles.

***2.1.5 Marching Scalar Value Organization***

Zhu Bridson [5] 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:

(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 () (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 cart
* 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 [1].

# 3. POF Software Testing

This section describes the POF system testing stages with screenshots as a 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.

tablo, pasta, oturma, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Figure 1:** 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.

**bilgisayar içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Figure 2:** 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 an 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.

**tablo, farklı, grup, geniş içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Figure 3:** 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.

A picture containing box, table, computer

Description automatically generated

**Figure 4:** Insideblueparticles vs all particles.

# 4. POF system Installation, Configuration and Operation

This section describes how user installs, configures and operates the POF system. Firstly, the user must have the required system mentioned before.

***4.1 NVIDIA Flex installation and set up***

We assume that a normal user has the POF system digital copy. User should open an empty 3D project from Unity. You can download Unity from the official website [6]. Firstly, you should import the NVIDIA Flex asset, you can download from the Unity asset store as the figure below. Access the NVIDIA Flex asset page in the Unity asset store. Click download.

When download finished, import button will appear. Click import. Also, you can import the POF system, so you do not necessarily need to import NVIDIA Flex from the Unity asset store. However, importing from the Unity asset store is shown here.

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu  **Figure 4:** Nvidia Flex in Unity asset store

Another window will appear, and you select all files and click import again.

ekran görüntüsü, elektronik eşyalar, bilgisayar içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Figure 5:** Import Unity package window

NVIDIA Flex is imported, the next step is making ready our scene by setting up the Flex. Make right-click on the project files which positioned below of the screen. Select create, from the new window select Flex and then select flex container.

A screenshot of a computer

Description automatically generated

**Figure 6:** Creating Flex container in Unity

A screenshot of a cell phone

Description automatically generatedWe have created flex container and a new table of flex container should appear as the figure below.

**Figure 7:** Flex container table

Next step is making a right-click in the hierarchy window and select create empty and Game object is created. Make right-click and change its name to Flex array actor. Click the Flex array actor and at the inspector menu click add component. Write Flex Array Actor to the search bar and press enter. On the new open menu, we need to specify flex container which is not assigned right now. Drag and drop Flex Container to space across Container of the Container title in the Flex Array Actor inspector menu. It should look like this in the figure below.

A screenshot of a cell phone screen with text

Description automatically generated**Figure 8:** Assigning Flex container to the Flex Array Actor table

A screenshot of a cell phone

Description automatically generatedMake right-click and select flex and from there select flex array asset. We create Flex array asset from the same place that we created a flex container. This step is similar to the previous one.

**Figure 9:** Make right click and select create

A screenshot of a cell phone

Description automatically generated

**Figure 10:** Select Flex Array Asset

Click the small button to select the boundary mesh as shown in the figure. You can select different boundary meshes. However, it is preferred to select the cube mesh.

A screenshot of a computer

Description automatically generated

**Figure 11:** Select Boundary Mesh

Next, we click our flex array actor and assign flex array asset by drag and drop to the array asset row. Enable the Fluid checkbox so our particles can act as a fluid. This is a method of creating particles to our scene. There are particles which are stored like an array in somewhere. When you play the scene, particle-based fluid simulation behaves as the NVIDIA Flex operates with the CUDA supported GPU in the back end. Another way is creating a Flex source asset. Flex source asset operates a kind of tap or fountain. Here is how you can set up Flex source asset into your scene.

Right-click in the project file window and create a flex source asset as shown in the figure.A screenshot of a cell phone

Description automatically generated

**Figure 12:** Creating Flex Source Asset

Change the value of mesh tessellation to 1 and then click flex source actor and select surface mesh as shown in the figure.

A screenshot of a cell phone

Description automatically generated

**Figure 13:** Selecting surface mesh of flex source asset

Select mesh from the newly opened menu. We prefer quad mesh however you can prefer different meshes such as a cylinder. It will change the way of the release of the particles.

A screenshot of a cell phone

Description automatically generated

**Figure 14:** Surface mesh type menu

Create an empty game object in scene hierarchy and rename it as flex source actor. Then, click add component and add Flex source actor script to our game object. Flex source actor script has not assigned parts so we should assign Flex container to the container row by drag and drop. Assign the flex source asset to the relevant row in flex source actor script as shown in the figure with a red square and an arrow.

A screenshot of a cell phone

Description automatically generated

**Figure 15:** Creating Flex Source Actor and adding component

Click flex array actor and click add component. Write flex fluid renderer and press enter. Do this part again for the flex source actor. Particles look like water or fluid-like because of the fluid renderer.

Another way to access NVIDIA Flex components is by using the new NVIDIA button on the top of the window between component and window buttons as shown in the figure. You can access to flex assets by using this way alternatively such as flex container, flex array asset etc.

A screenshot of a cell phone

Description automatically generated

**Figure 16:** NVIDIA sidebar in Unity

***4.2 POF installation and set up***

**ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu**Now, you can import the POF system. Click assets, from the opening menu please click import package. Click custom package as shown in the figure.

**Figure 17:** Select custom package in Unity.

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

A screenshot of a social media post

Description automatically generated

**Figure 18:** Find POF unity package in file explorer.

A new window will open as a next step. You can select which files to be imported on your unity project because of this window. Please select all and click import.

A screenshot of a cell phone

Description automatically generated

**Figure 19:** Import package file window.

When the process finished, you can see the POF package is set up to your unity project. As you can see, all components are included, NVIDIA Flex is one of them. However, you can import Flex asset from both Unity asset store and just import from our project. Next, you prepare the NVIDIA Flex because POF can work when Flex is executable properly. Marching cubes assets provides different performance effective visualization method. The Simu System covers most of the project. The Marching Cubes makes visual output completely different so we think that user can change the main setting from Marching cubes folder.

A screenshot of a computer

Description automatically generated

**Figure 20:** POF system set up on your unity project.

***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.

***A screenshot of a cell phone

Description automatically generated***

**Figure 21:** Description ofFlex 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.

***A screenshot of a cell phone

Description automatically generated***

**Figure 22:** Description ofFlex 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.

A screenshot of a cell phone

Description automatically generated

**Figure 23:** 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 do 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

<<https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/flex/manual.html#manual>> (Last review: 10.4.2020)

1. Unity 2018.3 manual, viewed 10 April 2020, <<https://docs.unity3d.com/2018.3/Documentation/Manual/index.html>>
2. NVIDIA Flex Set up tutorial video playlist , viewed 10 April 2020, <<https://www.youtube.com/watch?v=Fp1SMb3SWoo&list=PL4FII4B-zM0dMI-GgR3KsfJwm100MH3TT>>
3. [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.
4. [ZB05] Zhu, Y., & Bridson, R. (2005). Animating sand as a fluid. (New York, NY, USA, 2005) ACM Trans. Graph., 24, 965-972.