A real time algorithm for soft mesh deformation and interaction in collaborative, networked virtual reality environments

Lydatakis Nikolaos

Thesis submitted in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Associate Prof. George Papagiannakis

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

Computing Science Department

University of Crete

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

A real time algorithm for soft mesh deformation and interaction in collaborative, networked virtual reality environments

Thesis submitted by
Lydatakis Nikolaos
In partial fulfilment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Nikolaos Lydatakis

Committee approvals: _____
George Papagiannakis
Associate Professor, Thesis Supervisor

_____
Anthony Savidis
Professor, Committee Member

_____
Maria Papadopouli
Associate Professor, Committee Member

Departmental approval: _____
Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, January 2020

Abstract

A real time algorithm for soft mesh deformation and interaction in collaborative, networked virtual reality environments

In the previous years, Virtually Reality has become a useful tool for simulations and computer games. Collaborative-networked Virtual Reality Environments, allow users having suitable Virtual Reality devices, to interact in a shared virtual world with other participants.

Realistic deformation plays an important role in computer graphics, games, simulations and VR environments. Soft body simulations are used to change an object shape, when external forces are applied on it. With the use of term soft body, we mean a mesh, which can change its initial shape when interacting with other physical objects.

In virtual reality environments, the user is able to use hand controllers or other devices to interact with virtual rigid and soft objects. This interaction includes pick up, hold and drop objects. In a shared virtual world, interaction must be synchronized across all the participants.

In this thesis, we propose a soft mesh deformation algorithm suitable for Virtual Reality interaction and synchronization in shared virtual worlds. Our particle-based soft body algorithm provides easy control of the particles as physical objects in which direct interaction can be applied. Our system is capable to simulate interaction and soft particles deformation in shared virtual worlds by using a server/client architecture.

Περίληψη

Αλγόριθμος για παραμόρφωση και αλληλεπίδραση ελαστικών σωμάτων σε συνεργατικά εικονικά περιβάλλοντα

Τα τελευταία χρόνια, η εικονική πραγματικότητα έχει γίνει ένα χρήσιμο εργαλείο για προσομοιώσεις και παιχνίδια στον υπολογιστή. Συνεργατικά περιβάλλοντα εικονικής πραγματικότητας, επιτρέπουν στους χρήστες που διαθέτουν κατάλληλες συσκευές εικονικής πραγματικότητας να αλληλεπιδρούν σε έναν κοινό εικονικό κόσμο με άλλους συμμετέχοντες.

Οι ρεαλιστικές παραμορφώσεις παίζουν σημαντικό ρόλο στα γραφικά, τα παιχνίδια, τις προσομοιώσεις και τα περιβάλλοντα εικονικής πραγματικότητας. Οι προσομοιώσεις ελαστικών σωμάτων χρησιμοποιούνται για την αλλαγή του σχήματος ενός αντικειμένου, όταν εφαρμόζονται εξωτερικές δυνάμεις πάνω σε αυτό. Με τη χρήση του όρου ελαστικό σώμα, εννοούμε ένα πλέγμα, το οποίο μπορεί να αλλάξει το αρχικό του σχήμα όταν αλληλεπιδρά με άλλα φυσικά αντικείμενα.

Σε περιβάλλοντα εικονικής πραγματικότητας, ο χρήστης είναι σε θέση να χρησιμοποιεί χειριστήρια χειρός ή άλλες συσκευές για να αλληλεπιδρά με εικονικά άκαμπτα και ελαστικά αντικείμενα. Αυτή η αλληλεπίδραση περιλαμβάνει σήκωμα, κράτημα και πέταγμα αντικειμένων. Σε έναν κοινό εικονικό κόσμο, η αλληλεπίδραση πρέπει να συγχρονιστεί σε όλους τους συμμετέχοντες.

Σε αυτή την εργασία, προτείνουμε έναν αλγόριθμο παραμόρφωσης ελαστικών αντικειμένων κατάλληλο για αλληλεπίδραση σε εικονική πραγματικότητα και συγχρονισμό σε κοινόχρηστους εικονικούς κόσμους. Ο αλγόριθμος ελαστικών σωματικών βασισμένος σε σωματίδια μας παρέχει τον εύκολο έλεγχο τους ως φυσικά αντικείμενα στα οποία μπορεί να εφαρμοστεί άμεσα η αλληλεπίδραση. Το σύστημά μας είναι ικανό να προσομοιώνει την αλληλεπίδραση και τη ελαστική παραμόρφωση σωμάτων σε συνεργατικές εφαρμογές χρησιμοποιώντας μια αρχιτεκτονική διακομιστή / πελάτη.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this. I certify that this dissertation reports original work by me during my University project except for the following:

- Unity 3D, game engine this project is based on
- Unity 3D networking (UNet) layer
- Steam VR, Unity plugin from Valve Corporation
- Newton VR, Unity plugin from Tomorrow Today Labs

Acknowledgements

I would first like to thank my supervisor George Papagiannakis for his tireless support, inspiration, and for believing in me while still undergraduate to join his team, where I have gained invaluable experience and met people and partners sharing the same passion for knowledge as I do.

I would also like to thank to thank my co-workers and friends Paul Zikas, Stelios Georgiou, Steve Kateros, Mike Kentros, Emanuel Skordalakis, Mixalis Kontopoulos and Efstratios Geronikolakis for their support in stressful moments and for giving me strength to go on. Saridakis Leferis and Flora Fyka for helping me with spelling and syntax checking of this work.

Finally, I would like to express my gratitude to my family for their support, sacrifices, motivation and love.

Do. Or do not. There is no try.

Table of Contents

List of Figures

List of Tables

1 Introduction

In the previous years, Virtually Reality has become a useful tool for simulations and computer games. Collaborative-networked Virtual Reality Environments, allow users having VR devices, to interact in a shared virtual world. Nowadays, popularity of VR constantly grows leading to more demanding applications. Creating an immersive interactive virtual world, not only requires good visuals, but also needs natural interaction with the virtual scene.

Realistic deformations play an important role in computer graphics, games, simulations and VR environments. Soft body simulations are used to change an object shape, when external forces are applied. The computation of physically accurate deformation of objects when VR users uses hands controllers to interact is a liturgy, which requires much computation power. Only a few applications and simulations use soft body deformation due to computation power needed.

Virtual environments can be shared by multiple users across the internet. Currently many VR applications and simulations, have been developed to be shared by many participants in the same virtual world. Another even more challenging area, deformations of a soft mesh in a virtual collaborative environment must be shared across all users in the virtual world.

1.1 Scope and Objectives

The main goal of this project is to implement and evaluate a soft body mesh algorithm for collaborative virtual reality networked environments. In more detail, the proposed system developed to provide a soft body simulation to solve networking issues and VR compatibility. The user will be able to join in a VR networked collaborative environment with other VR users and interact with soft deformable objects, by using a VR headset and a pair of compatible controllers.

1.2 Achievements

In this thesis, we propose a novel soft mesh deformation algorithm suitable for Virtual Reality interaction and collaboration. The soft deformation algorithm in based on shape matching techniques and particle based soft body simulations. Our particle-based soft body algorithm is different from the state of the art because it provides easy control of the particles as physical objects and a center point, which controls the entire soft body position. Velocity based interaction can be applied directly to our particles while as physical objects can interact also with the environment.

Our Virtual Reality interaction system uses velocity base approach providing the ability to pick up, hold and drop objects. Due to our soft body particles nature, this interaction can be applied directly. To connect many participants in shared virtual world we use an authorative server/client network architecture. The VR users will have avatars that will be synchronized and the interaction with the environment and soft mesh deformations will be synchronized by the objects or particles transformations.

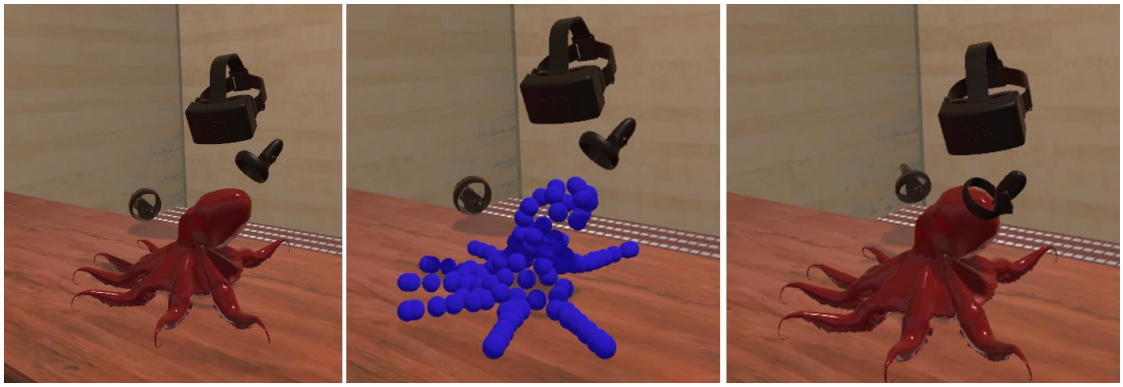


Figure 1.1Soft body mesh particles visualization

1.3 Overview of Dissertation

Over the next chapters, we will analyze the components that contributed to create our system. In more detail:

- On chapter 2 we will present the related work in the thematic areas of soft mesh deformations, Virtual reality networked environments, interaction in Virtual environments, soft mesh deformation in VR environments, and more.
- On chapter 3, details the implementation of our proposed algorithm for soft mesh deformations.
- On chapter 4, we will present the VR user interaction with solid and soft objects.
- Moving on chapter 5, we will propose a networking server/client model, to synchronize our deformation in collaborative VR environments.
- Finally, in chapter 6, we will summarize our results and evaluate our system in three perspectives, a) performance, b) networking and c) user experience. Also discusses the conclusion and future work.

2 State-of-The-Art

This section describes previous work on soft mesh deformations and interaction in Virtual Reality collaborative environments as well as similar projects and publications.

2.1 Existing real time soft mesh deformation methods

In 1987 D. Terzopoulos developed an approach, to model soft deformations based on the elasticity theory [7]. Also pointed the advantages of deformable objects over the rigid ones in computer graphics. Gribson at 1997, Nealen at 2006, Moore and Molloy at 2007 and Zhang at 2017 wrote great surveys on different types of models [8, 9, 10, 11]. Some of these methods are Mass-Spring Systems, Finite Element method and Shape matching techniques. These can be categorized in Geometrically based approaches, and Meshless - Free Form Deformations.

Mass-spring models feature a commonly used method to simulate soft deformable objects due to its simplicity [3, 12]. As Vassilev describes, "A general mass-spring system consists of n mass points, each of them being linked to its neighbors by massless springs of natural length greater than zero" [12]. Finite element soft deformation simulation, provide more accurate visual results [9, 10, 11]. In order, this method to be used in soft mesh deformations, mesh is encountered as connected volumes. The volumes can be obtained by dividing the model into a number of elementary building components called the finite elements, forming a finite element mesh of tetrahedral or hexahedral elements [10]. Another approach used by many real time physics engines as NVIDIA Flex[1] and Havok[2] and Bullet[3], is position-based-dynamics. As Camara describes, "In order to simulate deformable objects, the position-based-dynamics approach relies on a geometrically motivated shape-matching technique" [13].

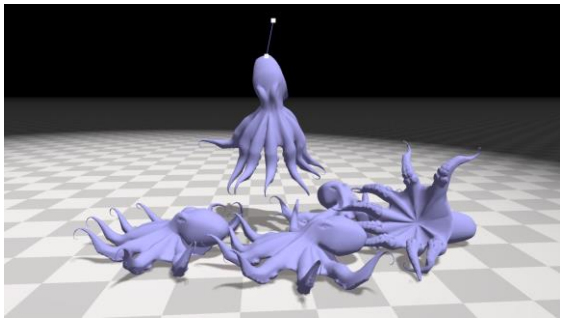


Figure 2.1 NVIDIA Flex[1] Soft Bodies

1. https://developer.nvidia.com/flex
2. https://www.havok.com/
3. http://bulletphysics.org/

Using finite elements method, Junggon K. at 2011 introduce a fast physically based system, for skeleton-driven deformable characters [4]. In this approach, the mesh skeleton and its animation drive the soft deformations. Another approach for computation offloading, introduced by Danevicius at 2018. In this approach, using an intelligent cloud computing technique to simulate soft body deformation to a client by a cloud server [5].

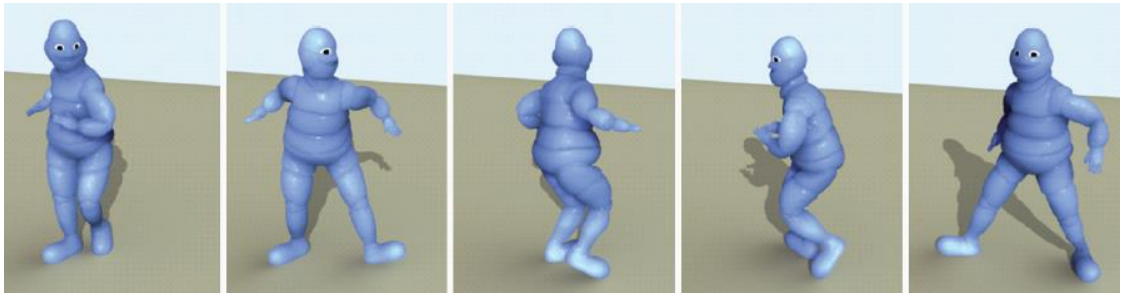


Figure 2.2 A skeleton-driven deformable body character (Fatman): A realistic jiggling behaviour of the dancing character's deformable body can be captured in near real time with our physically based simulation technique. [4]

2.2 Collaborative and Interactive Virtual Reality Environments

It was 1999 when Molet T., introduced a real time a virtual, networked, interactive tennis game simulation [15]. In this game, a general-purpose client/server model used called Virtual Life Network (VLNET). Their goal was to create realistic simulation with VR networked tennis simulation with VR avatars that will also support body deformations.

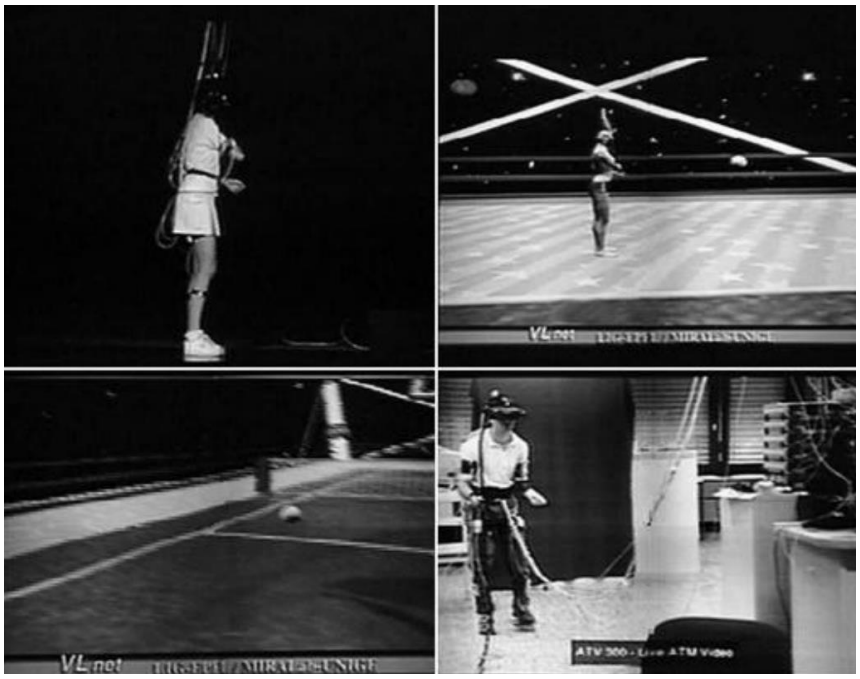


Figure 2.3 Real time tennis game [15]

Jorissen P. proposed a system for collaborative virtual environment with physically realistic interaction and synchronization [22]. They used a dedicated server that will run the simulation. The connected users won't have to run the simulation but only to synchronize by the server. At startup, clients connect to the server with reliable TCP to receive the current state of the virtual world Connected clients send their interaction to the server using UDP protocol (to reduce network load) and synchronize the simulation by the server.

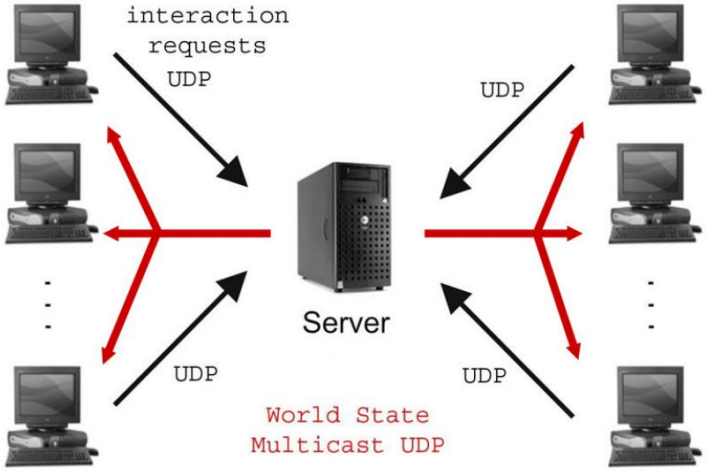


Figure 2.4 Network setup: after initialization using TCP, the clients send their interaction requests using UDP. State updates are distributed by the server using multicast

With the use of term Collaborative Virtual Reality Environments, Benford S. at 2001 described as "Collaborative Virtual Reality Environments are virtual worlds shared by participants across a computer network. Participants are provided with graphical embodiments called avatars that convey their identity, presence, location, and activities to others" [14]. These worlds can be applied in many areas. To mention some of these are areas learning environments and training environments. At the University of Washington, Randolph L and Eileen Fagan explored the potentials of the collaborative VR environments in learning processes [16]. Also, Teresa Monahan provide a study of for collaborative e-learning, showing the effect on the participants [17]. The learning capabilities of VR have great potential from surgical simulations [18] [20] to supplementary material for learning courses [16] [19].

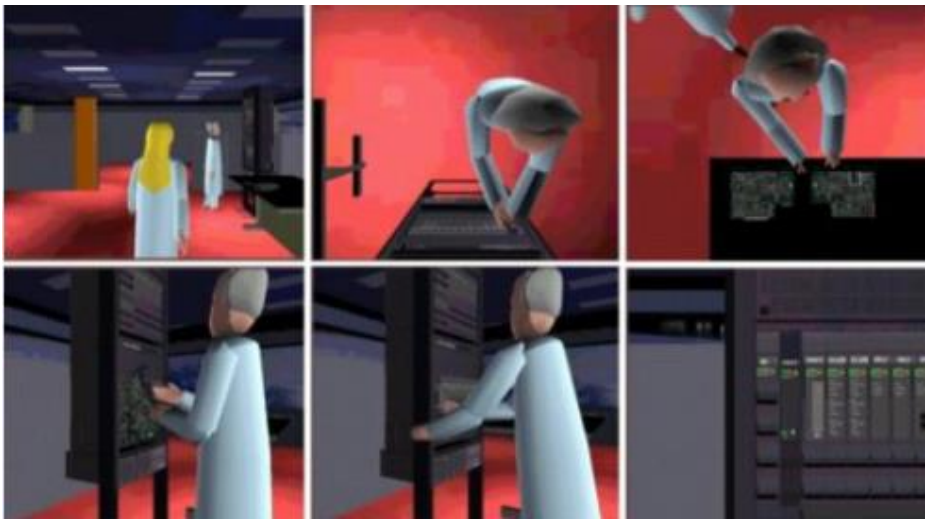


Figure 2.5 Replacing a Faulty Card [19]

Oculus Company provides a SDK[7], allowing rigid body synchronization in VR collaborative environments. This includes also, interaction and synchronization with virtual objects. The company Ready at Dawn, developed a state-of the-art virtual reality adventure game, called Lone Echo[6]. Players can move in space by grabbing or pushing objects in the environment. In interaction with the environment is used Finger IK for natural grasping. Other VR companies also released games and demos that are still considering state-of-the-art for their interactive and ease of use (The Lab[4], Robo Recall[5]).

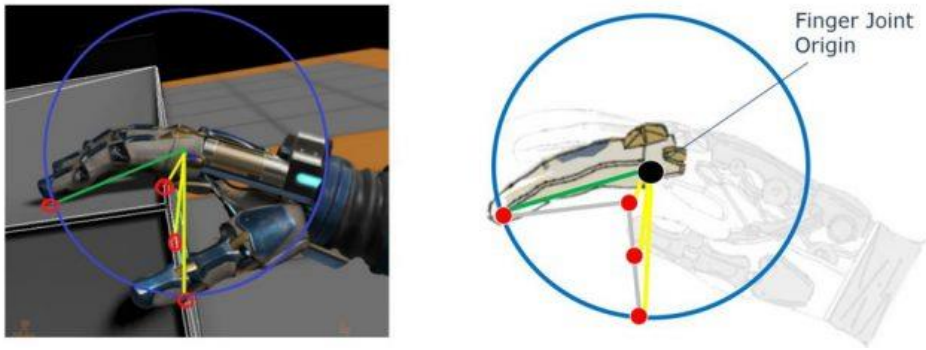


Figure 2.6 Lone Echo finger IK

4. https://store.steampowered.com/app/450390/The Lab/
5. https://www.epicgames.com/roborecall/
6. https://www.oculus.com/lone-echo/
7. https://developer.oculus.com/blog/networked-physics-in-virtual-reality-networking-a-stack-of-cubes-with-unity-and-physx/

Other integrations for interaction with virtual scene came from open source development kits. Keith Bradner and Nick Abel, developed an open source system in Unity3D that allows pick up, hold and throw objects called NewtonVR[8]. This approach, uses a Velocity Based interaction by taking advanced of PhysX physics engine, allows the interaction to be according to laws of physics. Also, provides a general integration of Oculus SDK, SteamVR and Windows Mixed Reality SDK.

At 2018, Elvezio C. from Columbia University at 2018 proposed a low latency interaction system for collaborative VR interaction [23]. They found that the collaboration is effective when the network latency is below 15ms, and it improves further at 3-7ms of latency.

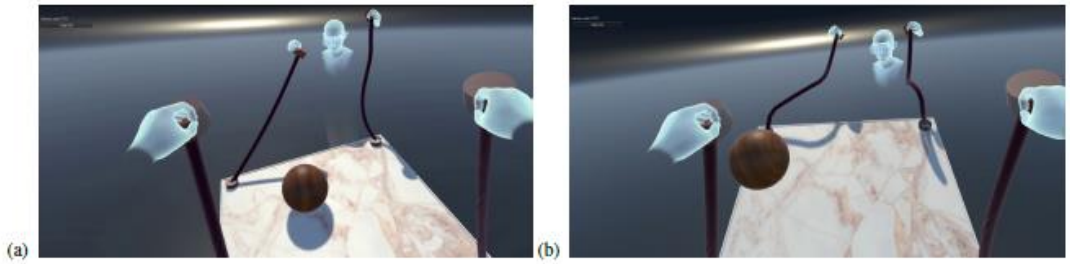


Figure 2.7 (a) Two users manipulate virtual board using four virtual ropes, one in each hand, to control virtual ball. (b) Users throw the ball upwards and try to catch it again. This experience requires low network latency for users to effectively coordinate their actions.

8. http://www.newtonvr.com/

2.3 Soft Mesh Deformation and Interaction in Collaborative Virtual Reality Environments

Sumengen and Tolga E. at 2007 proposed a method for soft mesh in VR collaborative environments based on linear finite-element [24]. Their approach requires a uniform-mesh representation of the simulated structure so they first had to implement a remeshing algorithm that converts it into a uniform triangular mesh. The network model used was based on peer-to-peer architecture with UDP for speed and bandwidth requirements. They use a quad-tree structure domain witch can be easily updated over the network. Their system requires 20 MBits per second without any compression for a surface with 10k vertices.

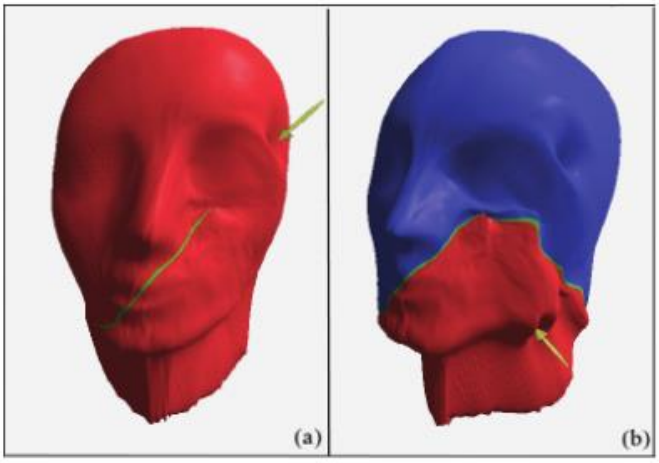


Figure 2.8 (a) One peer and (b) two-peers collaborative network deformation of a sample model having a regular mesh structure

Schiefer R continued the research by proposing a web service that facilitates deformable objects for shared virtual world [26]. This web-based simulation service can be accessed from any web client making the system to be able to run in also low-end devices or can run on the same computer to minimize the latency.

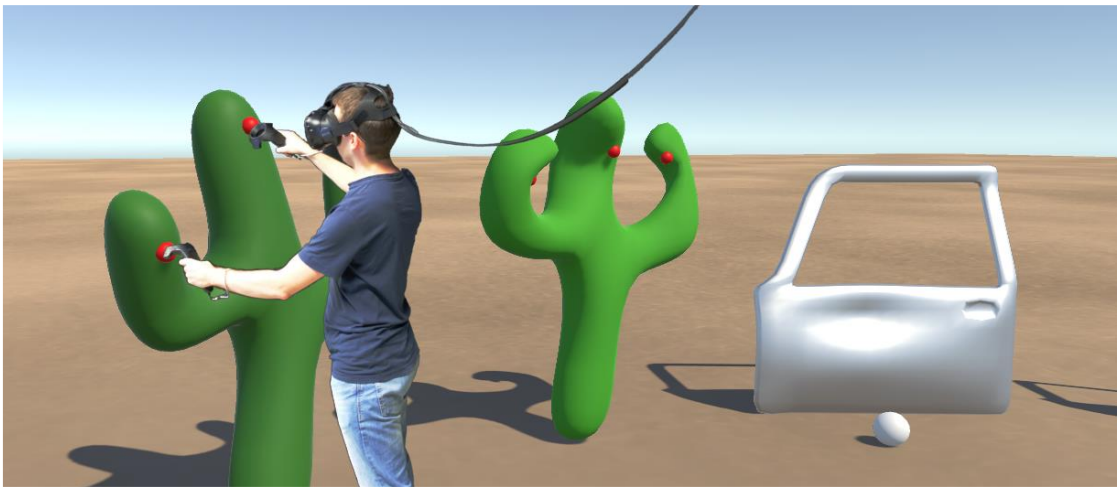


Figure 2.9 Subdivision surfaces can be interactively deformed in an intuitive way using hand controllers and a head mounted display. Users interact directly with the subdivision limit surface and can define constraints, visualized with small red spheres, by pushing and pulling the surface. Additionally, users can also pick up and move rigid body objects, like the white ball in the scene, which can also cause deformations by colliding with surfaces.

Soft deformable objects research is also focused on VR collaborative surgical applications [10] [13]. Qin J. developed a server/client model using TCP data protocol, to simulate

soft-tissue deformation in collaborative VR simulations [28]. They observed that the bandwidth needed increases dramatically when many participants are connected. Chheang V at 2019, propose a system for VR Collaborative Laparoscopic Liver Surgery Training called CollaVRLap. Their system use Unity multiplayer (unet) networking model [27].

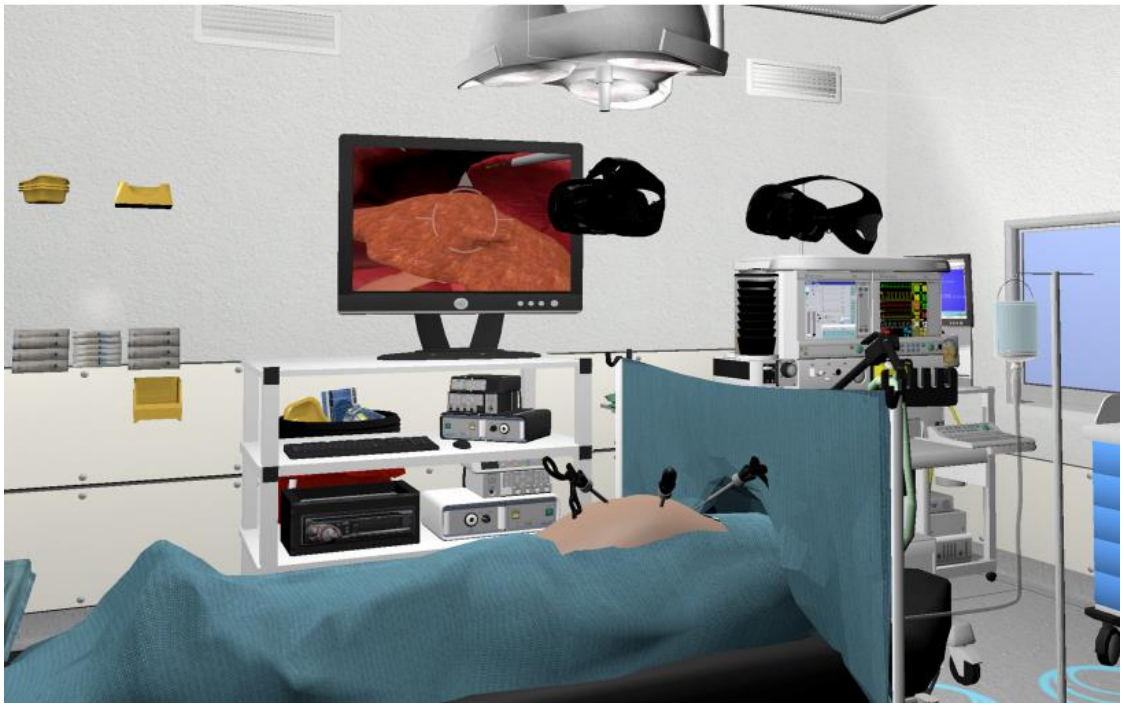


Figure 2.10 Surgeons virtually cooperate in the VR

Other techniques introduce soft mesh deformation collaborative VR with haptic interaction. Tang Z. at haptic-enables collaborative environment that supports soft deformations. In their system server/client model is used but also peer-to-peer architectures are compatible and also uses the UDP protocol for transformations transfer.

2.4 Our publications related to this work

- **Transforming medical education and training with VR using M.A.G.E.S. [20]**

In this work, we proposed a novel VR s/w system aiming to disrupt the healthcare training industry with the first Psychomotor Virtual Reality (VR) Surgical Training solution. Virtual surgeons connected on the same VR collaborative environment, can perform a virtual surgery. We delivered an educational tool for orthopedic surgeries to enhance the learning

procedure with gamification elements, advanced interactability and cooperative features in an immersive VR operating theater.

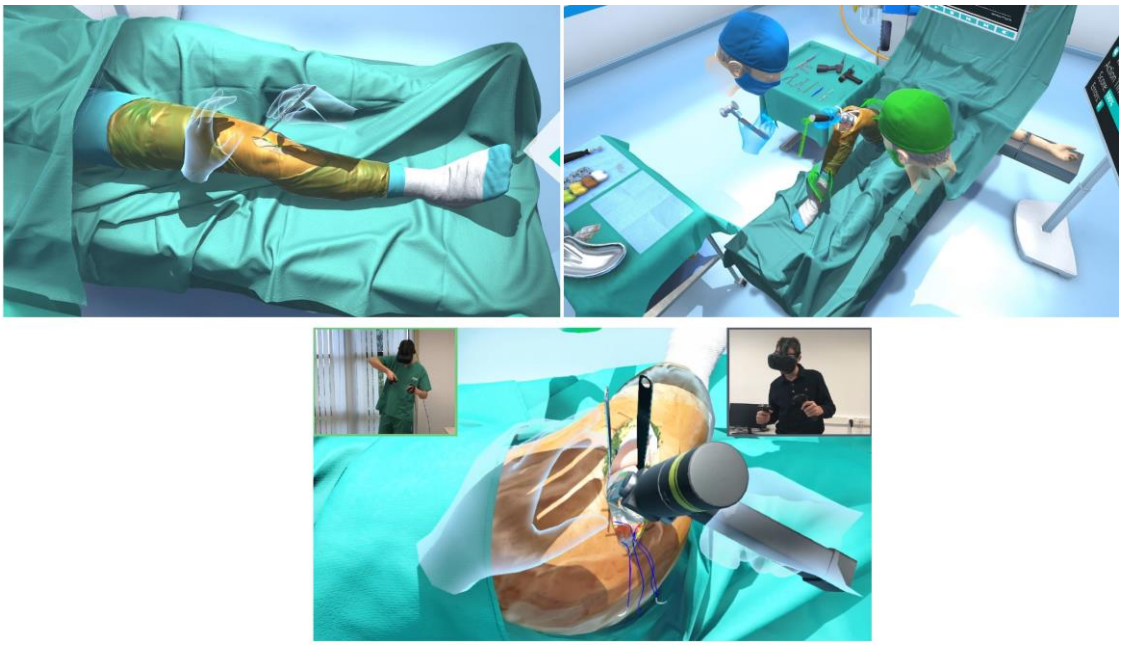


Figure 2.11 Different medical scenarios and functionalities of our system: a) Initial incision in Total Knee Arthroplasty simulation based on the medial parapatellar approach, b) Cooperative Total Knee Arthroplasty in which the main surgeon inserts the femoral implant, c) Femoral Hip resection performed by the main surgeon in Cooperative Total Hip Arthroplasty.

- **Mixed Reality Serious Games and Gamification for smart education [21]**

In this project, we developed two mixed reality serious games featuring the palace of Knossos in Crete. The first, was an AR application using the Meta AR glasses: a holographic-AR, tethered headset, ancestor of Microsoft HoloLens, introduced novel interactive features with gesture manipulation and holographic projection. The second application featured a mobile VR application regarding a virtual exploration of the Knossos Palace. Both applications refer to the same content in a different approach, based on the used Medium (AR & VR).

Figure 2.12 (Left) Bull leaping puzzle, grabbing tiles with both hands. (Middle) Put the columns of North Entrance back in place. (Right) Constructing color dyes to restore the painting.

3 Real time soft mesh deformations

With the use of term soft body, we mean a mesh that can change its initial vertices position when interacting with other physical meshes. Our main idea of creating a soft body physics algorithm is based on three main categories:

1. **Clustering**

   The term clustering describes how each vertex on our mesh being grouped in another object that will be used to calculate its deformation.

2. **How physics are applied**

   For the soft bodies to achieve some deformation, physics should be applied. To apply physics in our soft bodies efficient and with good performance, we apply them to the clusters that we used to group all the vertices.

3. **Mesh deformation**

   Finally, we have to calculate the mesh deformation. The resulting deformation derives from the Clustering transformation after the physics calculation. This transformation will have an effect on our mesh vertices.

3.1 Clustering

The main idea of a soft body's algorithm is to calculate the position of mesh vertices, after physics simulation. A solution could be to assume that each vertex is physical object. That means that we will have a physical object for each vertex in our mesh. A solution like this, cannot be performance efficient. For example, if we have a mesh with 10.000 vertices, we need to have 10.000 physical objects to simulate.

A solution to this problem is to group the mesh vertices together in other objects. We call this set of vertices a **particle**. This particle will have an effect on the mesh vertices when it will be transformed. Utilizing particles, instead of calculating the result of physics simulation for each vertex, we can calculate it for each particle Figure 3.1.

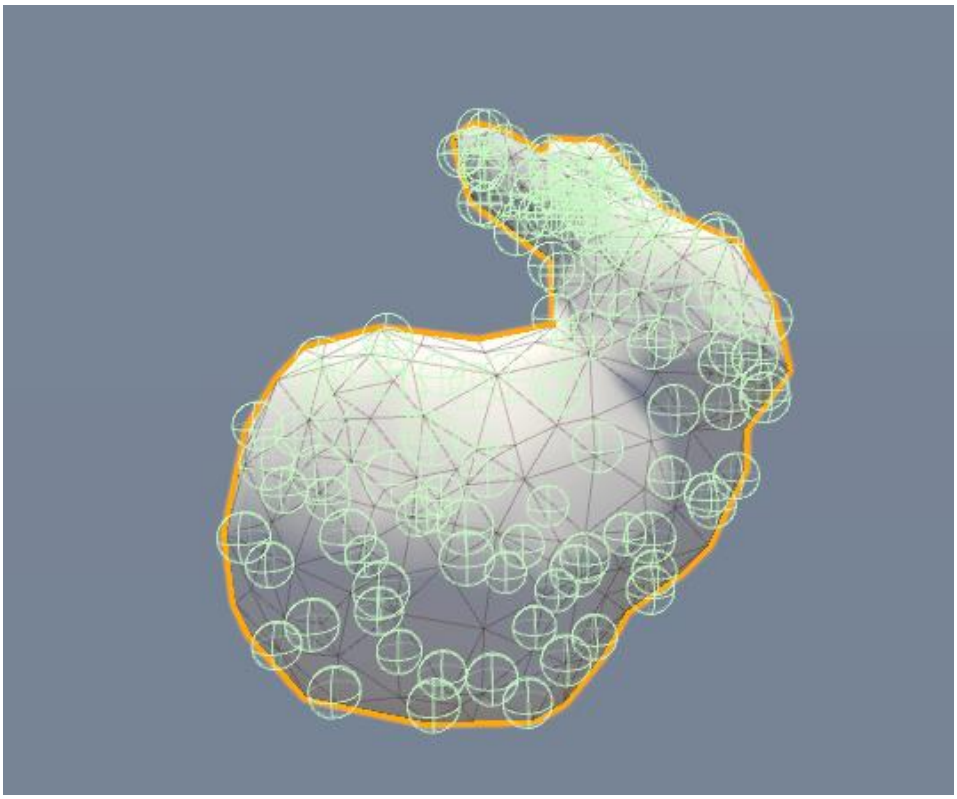


Figure 3.1: Particles in bunny 3d model

To create a cluster we have to find a way to group vertices that are close to each other. Thus, we have to create an algorithm that will take account the vertices distance and calculate as result the particles. If we create the particle based on vertices distance, there is no way we can predict how many vertices will be clustered. That implies we have to take into account how many vertices our mesh have, and how many of them we want to be grouped in our particle.

We implemented three clustering algorithms. The first one, clusters the vertices only by finding the closest N vertices. The second one, picks a vertex and group all the vertices that have the distance N into a particle. The third one, group all vertices based on desired distance but a vertex can be stored also in different particles with weights. The reason for implementing different clustering algorithms is because the vertices topology is different on meshes and only algorithm cannot serve all the different meshes.

3.1.1 Cluster based on closest vertices

The main goal of the first clustering algorithm we implemented, was to create a cluster with the N vertices that was the closest to each other. The number N is the vertices a particle will contain and it is picked by the user. The goal of this clustering algorithm, is to create particles that will contain same number of vertices.

The algorithm first, creates a dictionary. Then in this dictionary we insert all the mesh vertices with key (number) the vertex position and value (boolean) false. This dictionary will be used in our algorithm to mark if a vertex is clustered or not. Then we execute a loop for all vertices in our mesh: in each interaction our goal is to create a particle with N vertices. The first step to check if the current vertex is marked as clustered, if it is already clustered we continue with the next one. Then, we create a particle that will contain the current vertex and mark it as clustered. The next step is to find N-1 vertices that are the closest vertices to current one. To achieve that, we store the dictionary containing the vertices to a new array and sort this array based on the distance with the current vertex Figure 3.2. Then we can pick the N-1 first vertices from the new array. We also have to mark this vertices as clustered so we can ignore them in the next loops.

```
Sort(delegate (KeyValuePair<int, bool> c1, KeyValuePair<int, bool> c2)
{
       return Vector3.Distance
       (_vertices[i], _vertices[c1.Key]).CompareTo
       ((Vector3.Distance(_vertices[i], _vertices[c2.Key])));
});
```

Figure 3.2: Sort vertices by distance

3.1.2 Cluster based on distance

Cluster based on distance algorithm, groups all vertices together that have the desired distance d. The user picks the desired distance d. This algorithm defers from the previous one because there are no limits in how many vertices a particle will contain. The main criterion for clustering is the vertices distance.

This algorithm is a derivative of the previous one described in section 3.1.1. The difference is the vertices selection. After we sort our array that containing all the mesh vertices, instead of picking the N closest vertices, we now pick the vertices that have the desired distance d Figure 3.3, Figure 3.4.

```
for (int j = 0; (j < meshIDs.Count); j++)
     {
         float currParticleDist =
                    Vector3.Distance(_vertices[i],
              _vertices[sortedBydistance.ElementAt(j).Key]);

         if (currParticleDist <= ParticleDistanceLayer)
         {
             ph.AddAffectedVertex(sortedBydistance.ElementAt(j).Key, 1.0f);
             meshIDs[sortedBydistance.ElementAt(j).Key] = true;
             meshIDs.Remove(sortedBydistance.ElementAt(j).Key);
         }
         else if (currParticleDist > ParticleDistanceLayer)
         {
             break;
         }
     }
 }
```

Figure 3.3: Select vertices with the disired distance

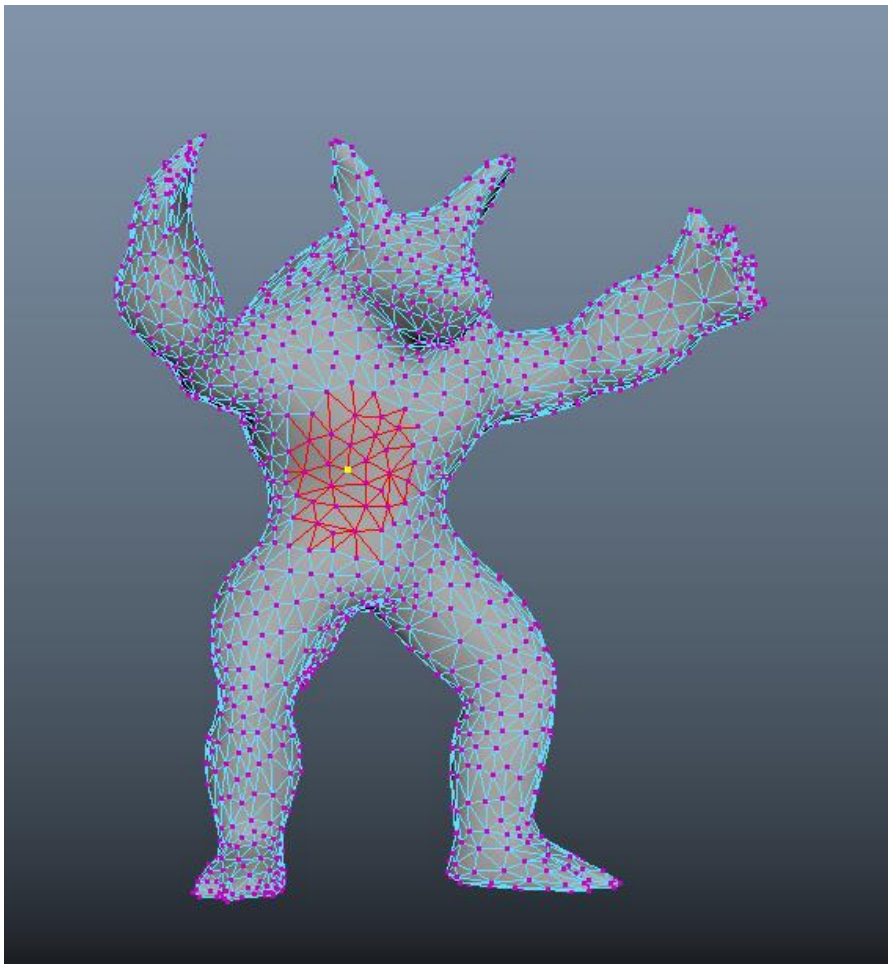


Figure 3.4: Cluster based on distance, particle affect area

3.1.3 Cluster vertices with weights based on distance

Both of the previous algorithm can create good enough clusters, but each can be affected only by one particle. In this clustering algorithm, we will create particles that will affect vertices with weights. Each vertex will be affected by many particles with a different weight value from each one.

In this algorithm, we will extend the previous implementation, as described in section 3.1.2. We will use instead of one, two different layers of distance thresholds Figure 3.5. The first one will be used to add vertices in clusters and mark them clustered and the second will add them to cluster but without marking them as clustered. With this method a vertex can be stored multiple particles. The benefited of this algorithm is that by using two distance threshold, can create a smooth cluster without having many number of particles.

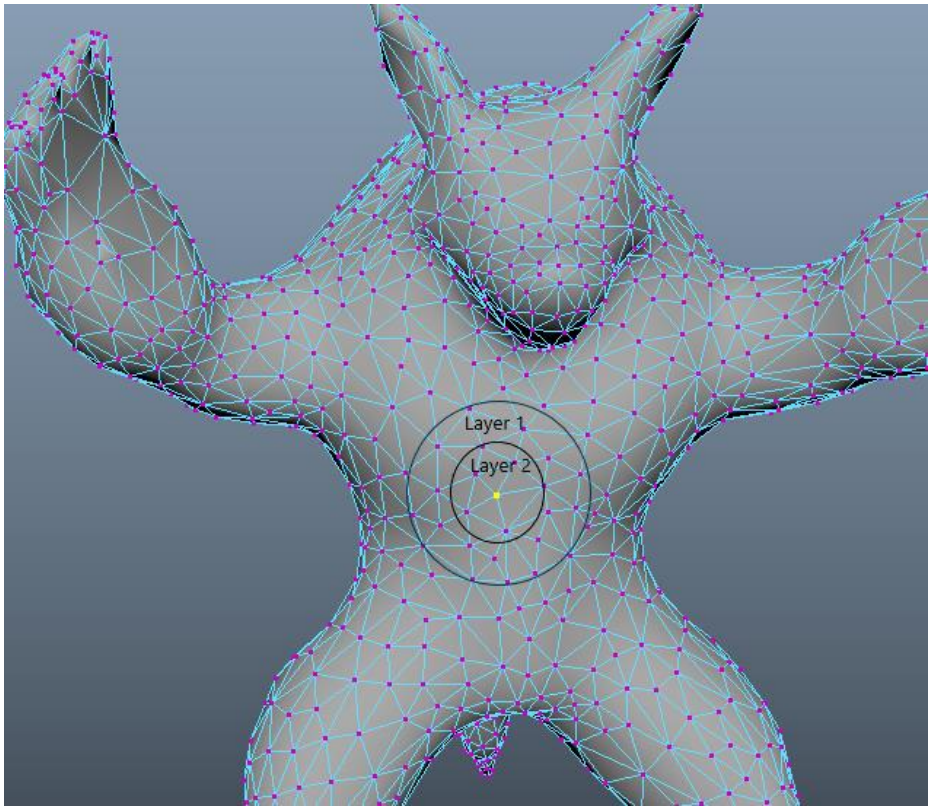


Figure 3.5: Clustering smoothing layers

The next step is to calculate the vertices weights for each particles. This step will be computed after the clustering is done. First, we will find for each vertex the particles that contain it and the total distance for this vertex from all the particles. Then we change the vertex weight for each particle with the inverted vertex distance with particle, divided by the total distance Figure 3.6.

```
for (int i = 0; i < _vertices.Length; i++)
{
    float total_distance = 0.0f;
    int total_afftect = 0;

    //Find the total distance for each vertex
    foreach (ParticleHelper _ph in particlesList)
    {
        if (_ph.affectedVertices.ContainsKey(i))
        {
            total_afftect++;
            float curr_distance = Vector3.Distance(_vertices[i],
_ph.transform.localPosition);
            total_distance += curr_distance;
        }
    }
    float total_distance_inverted = 0.0f;
    //Calculate inverted distance weight
    foreach (ParticleHelper _ph in particlesList)
    {
        if (_ph.affectedVertices.ContainsKey(i))
        {
            float curr_distance = Vector3.Distance(_vertices[i],
_ph.transform.localPosition);
            total_distance_inverted += 1.0f - (curr_distance / total_distance);
        }
    }

    foreach (ParticleHelper _ph in particlesList)
    {
        if (_ph.affectedVertices.ContainsKey(i))
        {
            float curr_distance = Vector3.Distance(_vertices[i],
_ph.transform.localPosition);
            float affected_vertext_value = ((1.0f - (curr_distance / total_distance)) /
total_distance_inverted);
            _ph.ChangeAffectedVertex(i, affected_vertext_value);
        }
    }

}
```

Figure 3.6 Particle weights recalculation

3.2 Particles physics simulation

After clustering our 3D model its time to simulate the particle physics. Since our particles are connected with our mesh vertices, we will be able to deform our mesh based on our particle transformations. To render a convincing soft body physics deformation, our mesh must be affected by collisions, gravity and other forces that applied Figure 3.7.

To simulate physics we use the integration of NVIDIA PhysX physics system in unity3D game engine. This allow us to simulate physics by using a “rigidbody” component to our objects. We can also use the mass model to simulate the mass of our objects, drag, angular drag and define if an object will be affected by gravity or be will kinematic or neither of this.

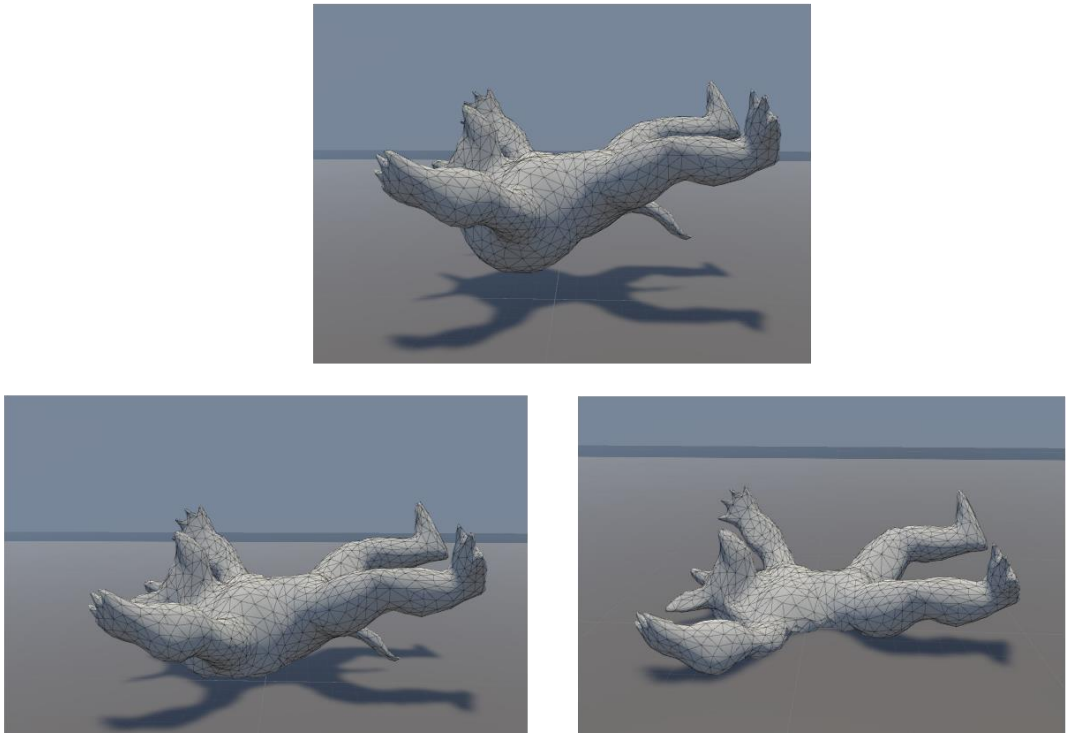


Figure 3.7 Soft body mesh simulation

3.2.1 Soft body and particles initialization

When the clustering is done, we have stored in each particle, the vertices that are affected by this particle with the affected value. The first stage of the initialization is to calculate the center of all the particles. We will use it later, in order to correct the velocities that will be applied to our particle. For the calculation of the particle center, we sum all the particles positions and divide by the number of the particles Figure 3.8.

$$\frac{\sum_{1}^{particles\_count} particle\_position}{particles\_count}$$

Figure 3.8: Soft body center calculation

Our particles current position is also the position of the first vertex we used to create it. We will use the same technique to re-center also our particles. Here we will use the particle

affected vertex positions to re-center it. Then we will save the particle initial position to use it later.

Then in order to have a better interaction when forces are applied to our particles, we created a connection between the particles. By using this connection, when a force is applied to a particle, this particle will apply a small amount of that force to the particles that are closed to it.

This connection needs to be configurable by the user for easier development and customization. This is the reason we added a distance d that the user will set. Then by using this value, we create the connection with the other particles with the following type.

$$Clamp\left(1.0 - \left(\frac{Dis\tan ce(particle1, particle2}{d}\right),0,1\right)$$

3.2.2 Physics behavior

Until now, we grouped all mesh vertices into finite particles. The next step of our algorithm is to simulate physics for these particles. Our first attempt was to let the particles do the physics simulation without making any intervention on it. The results was the each particle was split and the original shape of our model was not maintained.

To keep the original shape of the model, we had do find something to make the particles trying to stay in their original position. To do that, we let the particle do the physics simulation normally, and then we apply some extra velocities on each update to make the particle return in its original position.

To find the velocity target, we use the center of soft body that we calculate on the initialization. We will also use the center to apply our velocities from rotation changes. Using the center and the particle initial position we can find the velocity target that will be:

$$Destination = Initial\_position + Center\_position$$
$$DestinationRotated = (Center\_rotation * (Destination - Center\_position)) + Center\_position$$
$$VelocityDestinaiton = DestinationRotated - current\_position$$

The next step is to use now our particle connection to apply an amount of the velocity to other particles. We store this value to the particle that will be affected multiplied by the value that the particles are connected (3.2.1 above). This velocity is the velocity we use for correction, so we can understand that is targeting by the opposite direction. That is why we will remove it of our base velocity correction later.

We now have to apply this velocity correction to our particle. To apply now this velocity smoothly we want it to be applied by some amount in each game engine update. To solve this issue we multiply it with the time that our game engine need to make one update (delta time). Our application frame rate is target is 90 frame per second. To make the calculations easy for us we will use the expected update time that is 1/90 = 0.1111 as a denominator for the delta time. Finally, we want to ensure that there will not be applied some huge amount of velocity changes at the same time. For that reason, we set a limit at 10 max velocity change per update.

$$VelocityDestinaiton = VelocityDestinaiton - VelocityFromOtherParticles$$

```
expectedDelta = Time.deltaTime / 0.0111112f;
velocity = Vector3.MoveTowards(velocity, VelocityDestinaiton * expectedDelta, MaxVelocityChange);
```

To give our object the filling of softness or stiffness we set a floating value (configured by the user) from range 0.1-100. The lower the value is the more our soft body mesh become softer and higher the value is the soft body mesh become stiffer. This value will be multiplied to our velocity correction.

However, when is the correct time to apply our velocity corrections? As we can see in the Figure 3.9, Unity3d order of execution functions, there are three update functions. The FixedUpdate, the Update and the LateUpdate. The Update is called once per frame to update our game logic. The LateUpdate is called also once per frame but after all Updates are finished. The FixedUpdate may be called more frequently. The FixedUpdate is called on “fixed” times, that means that if our frame rate is low it will be called more often (not binded with the framerate). From this explanation we acquire that if we use the FixedUpdate we can also remove the delta time multiplication because it will be called “fixed” times and will be not depended to our frame rate Figure 3.9.

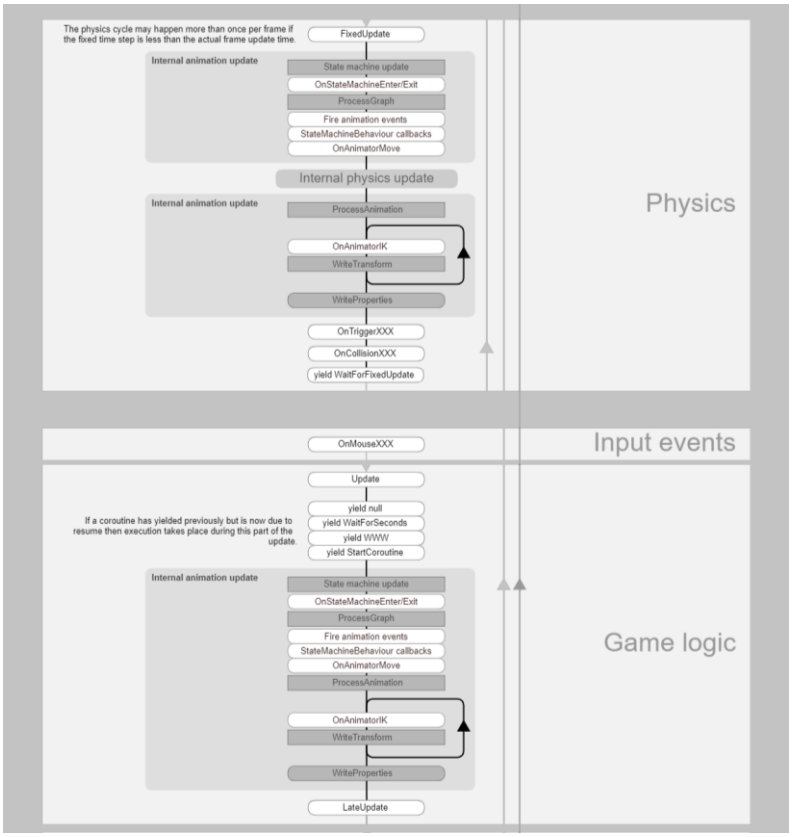


Figure 3.9: Unity3d order of execution for event functions

Image downloaded from:

https://docs.unity3d.com/Manual/ExecutionOrder.html

3.2.3 Collision handling

As we describe above, we use the PhysX physics engine integration in Unity3D game engine. We will let the physics engine to handle the collisions for us. What we will do is to set up each particle as a physical objet by adding the Rigidbody component on it. Then, we will let user decided the mass of each particle. The mass is used in this physics engine to simulate objects collations based on their masses.

With the Rigidbody component we had to choose of two simulation techniques on how physics will be calculated for our mesh. The first one is to simulate with gravity. Each of particle will be simulated with gravity. The problem here is that our algorithm will remove these gravity velocities that are applied on each update. The result is that we jittering because our algorithm and the gravity simulation are trying to outmatch each other. Our second option was to use simulation without gravity for our particles. With this simulation will be applied normally all forces for collations, but gravity will not be applied to our particles. This will solve our jittering problem. Without the gravity simulation model, our soft body mesh will simulate physics in local space. We will provide our solution for that in 3.4.

The next part is to choose which collider our particle will have. Our first thought was to create a collider based on the vertices that our particle have. This would be a mesh collider. The problem with this implementation is that a mesh collider will calculate the collision with all the mesh triangles. This collision detection method is too expensive. We decided instead of this to use Sphere and Box colliders to increase our performance.

3.3 Mesh deformations

In this section, we describe how our particle transformation is applied on our actual mesh. Our mesh vertices are an array of 3d points (x, y, z coordinates) on local space of our mesh. This array will be modified by the particles that are generated for this soft body mesh.

Each particle has stored which vertices will transform with a value 0.0-1.0 that is the effect value 3.1 in an array with a pair value. The first value is the index of the vertex in the vertex array and second value is the percentage of the transformation that will be applied. On each frame, the particles will update vertices from our particle transformation, by adding their movement multiplied by the affected value. We will store this result in temporary array and the end of the update will swap with our mesh vertex array.

$$ParticleDistanceTravelled = Current_{position} - Last_{position}$$
$$Vertex_{position} = Vertex_{position} + ParticleDistanceTravelled * VertexAffectValue$$

```
for(int i = 0; i < affectedVerticesArray.Length; i++)
{

       temp_mesh_verticies[affectedVerticesArray[i].Key] +=
                            movment * affectedVerticesArray[i].Value;

}
```

Our algorithm can group many vertices in one particle. When our object rotates, our particle will find its correct position but our vertices will translate there without any rotation applied. The reason of this problem is that the vertices and particles don't match one by one. Figure 3.10 shows the result of the rotation of 90 degrees in one axis for a triangle.

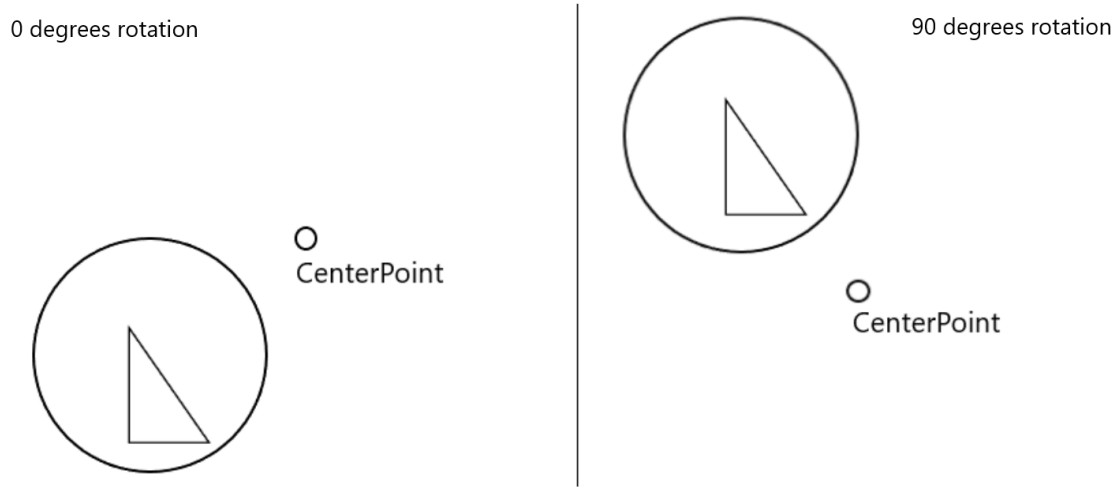


Figure 3.10: Deformation rotation error

Our solution for this problem is to apply the rotation of the center point, to the vertices using the particles as the pivot of the rotation. The correct order to apply the rotation is after the vertices are translated Figure 3.11.

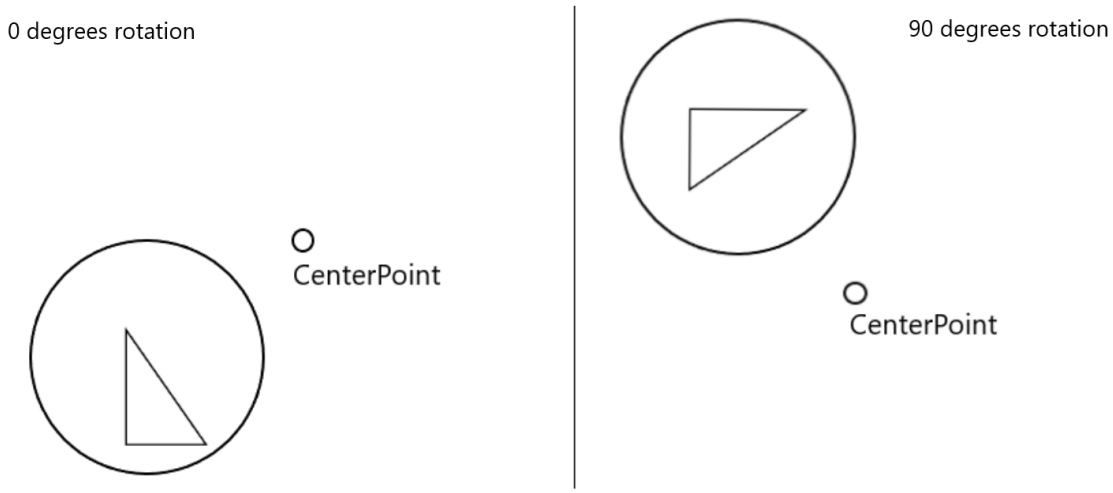


Figure 3.11 Appropriate rotation of a triangle

Our vertices also can be affected by many particles. Which paricle is the correct to use as pivot? Our first try was to use the particle with the highest affect value as pivot. This solution did not have good visual results. We use as pivot for the rotation, the sumarized positions of the vertices, multiplied by their affected value for each vertex to use as center.

3.4 World and local simulation

Currently our algorithm can perform a soft body mesh simulation in local space. If we use gravity on our particles as described on 3.2.3, we will have a jittering effect. We want our algorithm to be able to perform in local and world space. But how we will change our algorithm to be able to perform in world space?

In our soft body algorithm, we have a center that we use to calculate the mesh velocities corrections 3.2.2. If our center moves, our soft body will also move and simulate based on the center. That means that if our center is an object that simulate the gravity physics model, our soft body will do it also. Therefore, if we want to simulate in world space we will just have to simulate gravity for our soft body center Figure 3.12.

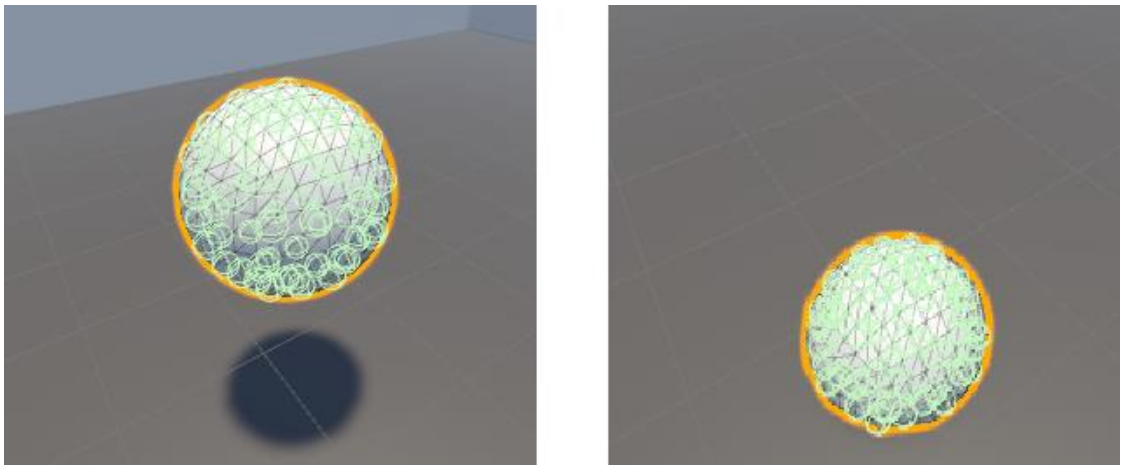


Figure 3.12: Soft body mesh in world space simulation

3.5 Performance Optimization

One of the goals of our algorithm is to run in high frame rate. Virtual reality applications target up to 144Hz (144 frames per second). In order our algorithm to be able to run in this frame rate, we had to do some optimizations. Also, it's important for algorithm to run fast, because virtual reality render time is higher than a normal render.

3.5.1 Remove base mesh properties

When we place soft body mesh in our scene, we first place in its starting position, rotation and scale. These three properties should be computed in our calculation to find the correct velocities for our particles. That means that we have to use these properties in our calculations.

In our algorithm to increase our performance and to simplify our code, we will remove these properties from our mesh. Since our mesh is consists of vertices, normals and tangents

that are affected by this properties, we will modify their initial values to remove this base properties. Then we replace with properties with the zero translation, zero rotation and scale one, so we will have the identity matrix. To achieve that, we will create a transformation matrix with properties and use it remove them. Finally, we have to multiply them with our matrix and them save this result back to our mesh.

```
Matrix4x4 trs = Matrix4x4.TRS(translation, rotation, scale);
Vector3[] vertices = mesh.vertices;
for (int i = 0; i < vertices.Length; i++)
{
    vertices[i] = trs.MultiplyPoint3x4(vertices[i]) ;
}
mesh.vertices = vertices;
```

3.5.2 Restrict soft body mesh in a selected area

In real world, not all objects are completely soft or solid. Some of them are consists of soft and solid parts. In this case, if we contain the whole object vertices in our simulation, it will be completely soft. Also calculating these vertices in our simulation will decrease our performance.

In our implementation, we provide an option to choose which region of our mesh we want to be soft. For simplification, region can be selected by setting up a collider that will contain the area of the mesh we want to be soft Figure 3.13. Then, we check for each vertex if is inside this collider. If it is not we will exclude it from our clustering algorithm.

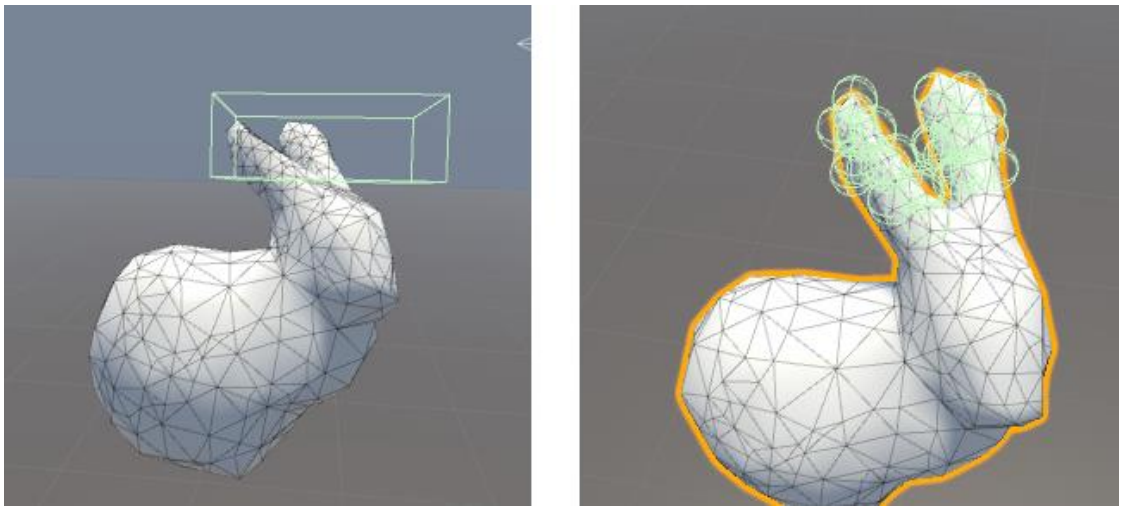


Figure 3.13 Restrict soft mesh in a selected area

3.5.3 Self-collisions

In a physics engine, the more collisions detections, the more time it will need find them. We will provide the option in our implementation to make the particles ignore collisions between its other. The first solution to do that was to use collision layers. We could use them so the game engine will not compute collision if two colliders are in this layer but will compute those if the colliders are in different. In this solution, the user had to create this layer for each soft body mesh. To simplify this task, instead of use this solution, we will directly unregister the collision detection between the particles in a soft body.

```
foreach (ParticleHelper _ph1 in particlesList)
{
    foreach (ParticleHelper _ph2 in particlesList)
    {
        if(_ph1!=_ph2)
        {
            if(_ph1.GetComponent<Collider>()&& _ph2.GetComponent<Collider>())
            {
                        Physics.IgnoreCollision(_ph1.GetComponent<Collider>(),
                        _ph2.GetComponent<Collider>());
            }
        }
    }
    if (_ph1.GetComponent<Collider>() && center.GetComponent<Collider>())
    {
        Physics.IgnoreCollision(_ph1.GetComponent<Collider>(),
                        center.GetComponent<Collider>());
    }

}
```

4 Soft bodies interaction in Virtual Reality

In virtual reality environments, the user can utilize hand controllers or other devices to interact with virtual objects. In this chapter, we will provide a method on how this interaction can be performed based on physics properties. We will focus on how this interaction can be done by pickup and drop an object or physically interact with it. Then we will use this method to interact with our soft body mesh.

- **Pickup objects**
  With the use of VR controller, the user is able to pick an virtual object and move it according the laws of physics. The object will not be able to pass by another object and will normally stop or drag around it or push it.
- **Physical interaction**
  The user will be able to use the VR controllers to interact with object physically. For example, the user will be able to push an object.
- **Interact with soft body**
  To interact with a soft body mesh, we will use both of the above methods. The user will be able to grab a soft body mesh (the entire object or a part of it) or physically interact with it.

4.1 Interacting with virtual hands

Inside a virtual environment, we want the user to be able to interact with virtual objects by using hand controllers. We want this interaction to be done according to laws of physics. The first step for providing this functionality is to know which objects the user virtual hand hovers. Each time user hovers a virtual object we store it on a list with the other objects that user hovers. We will use this list to choose an object that the user will pick up, when will press a button on his hand controller. The object that the user will pick up, will be the one that is in the closest distance with his hand Figure 4.1. We will store then, the position of the user hand that picked up the object.

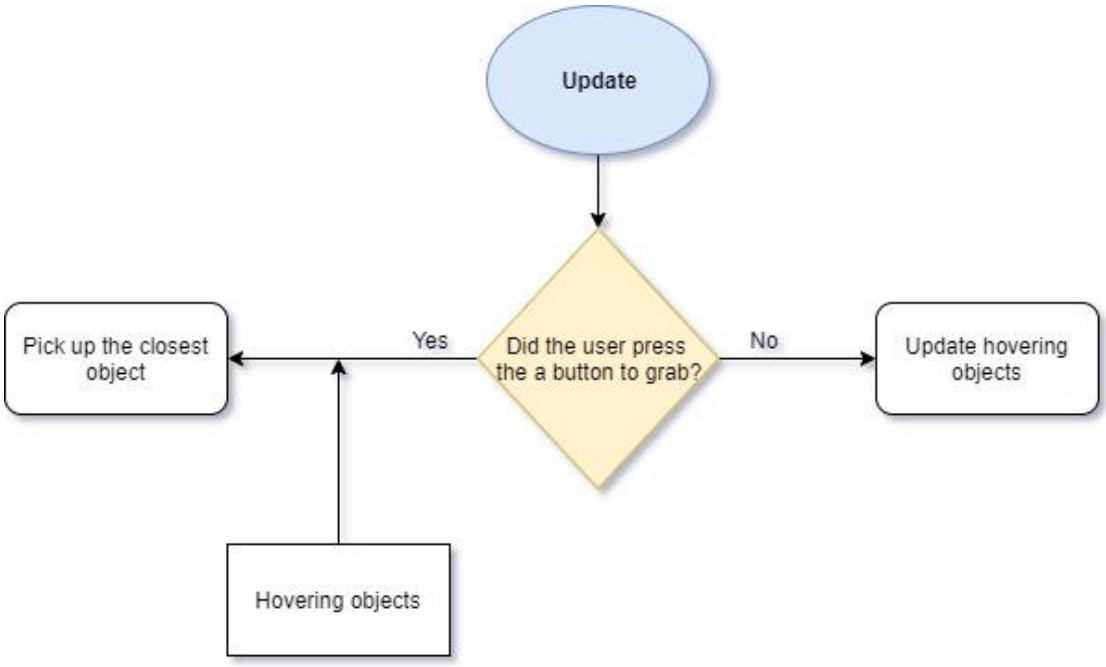


Figure 4.1 Pickup virtual object proses diagram

When the user picks up the object, it will follow his/her virtual hand. We want this object not to be able to pass through other physical objects taking mass into account. To achieve that, in each update the user is holding the button to grab the object, we will change the object velocity of the object to reach the user hand. This way other forces will still be able to affect our object.

By the time the user release the grab button, we want the objects to be dropped. If we just stop updating the velocities the object will be dropped based on the last velocity that was applied resulting in a free fall from the drop point. In order our object to get the hand acceleration when it will be dropped, we will apply the average velocity of the three last positions that the user was holding it Figure 4.2.

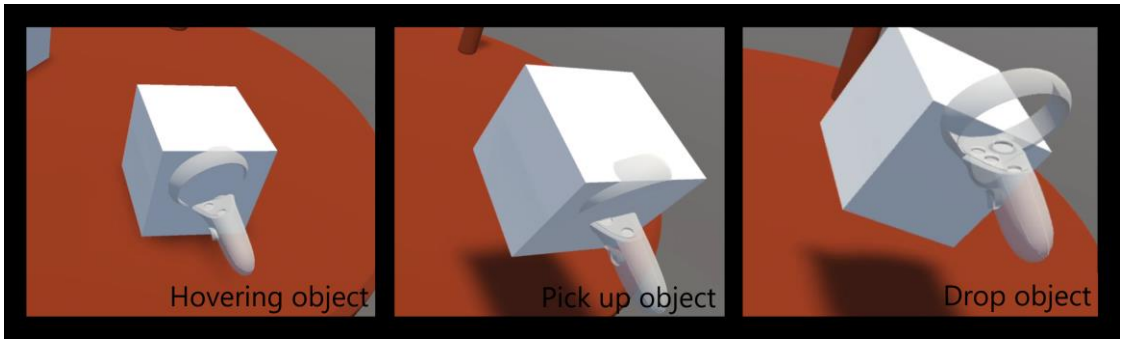


Figure 4.2 Hover, Pick up and drop an object

4.1.1 Interaction with soft body

We want soft body mesh to have this kind of interaction. The first thought, was to set all our soft mesh particles to interact when the user pick up the soft body. What we saw, was that this method was too expensive because we had to update the velocities for all our particles.

Our solution is to use this method in the center object of our soft body mesh. This method works because all our particles simulate around of the center object. When the center moves the particles will also move and simulate around it.

4.1.2 Particle based interaction for soft body mesh

To provide a higher fidelity interaction mechanism, we will provide the functionality of interacting not only the center object of a soft body but with each particle also. Since we interact with non-rigid deformable objects, a more sophisticated method, is to be able to interact with specific areas of model also.

We will also use here the same method but instead of using it on the center of object of a soft body mesh, we will use it on each particle separately. This will give the ability to our particles, to be able to pick up. The user will be able to pick up one particle each time. When the user controller hovers the soft body object, the controller hovers many particles. By the time the user press the grip button, will pick up the particle that is the closest to the controller.

Based on the clustering method that we will choose to create our soft body mesh, described at 3.1, we will have different visual results for the interaction. This also will also be affected by the particles connection. As described in 3.2.1 when a particle change its velocity, will apply an amount of that to the connected particles. As result of this, when the user will pick up an object, the velocity that will be applied to hold it will also affect the connected particles.

Using “Cluster based on closest vertices” or “Cluster based on distance” described at 3.1.1 or 3.1.2 each vertex of our soft body mesh will be affected by only one particle. As a result, when the user will interact with this particle, the visual result will be more solid.

Using “Cluster vertices with weights based on distance” described at [3.1.3] each vertex can be affected by many particles with influence from 0 to 1. The closest the vertex is to our particle, the bugger the weight will be. When our particle will be moved due to interaction, the vertices that are closest to this particle, will be transformed more. This will make the visual result softer.

By the time the grip button is released, the interaction will end. Then, based on the velocity history will be applied some extra velocity to our particle (drop acceleration). Due to our soft body algorithm, the particles will have tend to return on their initial position (based on the center object) [3.2.2], this velocity will be combined.

4.1.3 Physical controller interaction

Until now in this section, we described how the user in a virtual environment can pick up hold and drop objects and soft body objects. In this subsection, we will analyze how to interact with them with physical controller. **With the term "physical controller", we mean that our virtual controller will interact as a physical based objected and will be able to push other objects and soft bodies.**

We will enable this functionality to the user when the grip button is pressed without hovering only other object. Then it will stop when the grip button is released. Our physical controller will be an object same as our controller that be picked up with the same method described in 4.1 and then our actual controller will be hidden. Releasing the grip button will reset the initial virtual controller.

With this functionality, the user will be able to physically interact with other physical objects. As a result, will be able to push other objects and if approach a rigid surface the controller will not pass through it Figure 4.3.

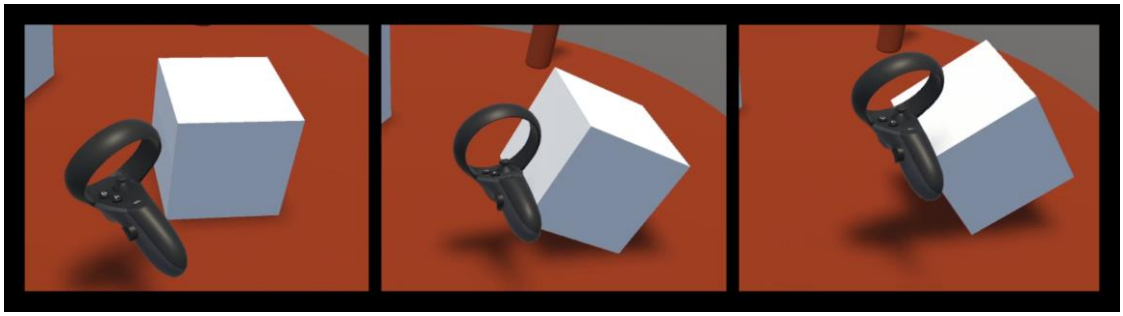


Figure 4.3 Physical controller interaction

In our soft body mesh, this will be used, to push particles or the center object. Since our method described at 3.2, accept external velocities, the interaction with the physical controller will be normally be provided.

4.2 Comparison with parent based interaction

Another method used to interact with object is the parent-based interaction. In this method, the object behaves as a child of the virtual controller when the interaction starts and follows the controller transformation changes. Due to this behavior, it skips the physics update

when the controller position changes. This cause the object to pass through other physical objects.

In our soft body method, when we tried this approach, we noticed that our particles stick in other physical object, for example under the virtual floor. Also, due to the way this method works we are not able to know the acceleration of the object and when the user release the grip button, the object will fall ignoring the acceleration. Trying also to use this method to simulate a physical controller will have the same results, and the controller will not able to interact as a physics object.

	Velocity based interaction	**Parenting interaction**
Pick up	✓	✓
Object throwing	✓	×
Physics simulation	✓	×
Physical controller	✓	×

Table 1: Comparing velocity based interaction with parenting.

5 Interactive soft body simulation in collaborative networked virtual environments

Up to this point, we presented a soft body mesh algorithm for real time mesh deformations and VR interaction with soft bodies. In this section, we will describe how to synchronize this deformation and interaction in a collaborative virtual environment.

Collaborative virtual environments are virtual words shared by participants across a computer network.

Steve Benford, University of Nottingham

2001

In our collaborative virtual environment, users will be able to interact with the environment and soft objects by using a VR headset and a pair of compatible controllers. We want this interaction to be shared in all users by using the lowest bandwidth we can.

5.1 Network model

Our networking layer is made upon Unity3D networking layer UNET. UNET provide us a high level API for building multiplayer applications. We use an authorative server/client model. In our system, there is no dedicated server, we use host model where the server (host) will also be a client at the same time Figure 5.1.

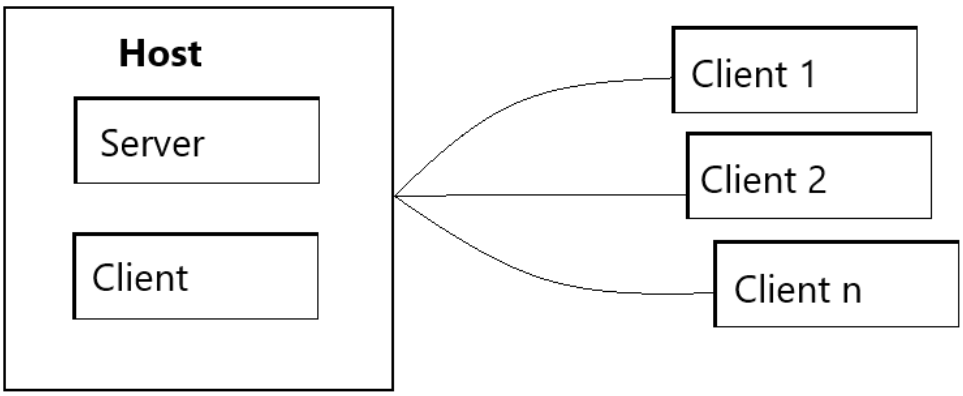


Figure 5.1 Networking host model

At the beginning, the host creates a room using the Unity multiplayer service. Then the clients can connect. Our client has direct connection to the server but only the server is able to know all client connections. When a client want to send a packet, sends it to the server and the server will retransmit that to all other clients Figure 5.2.

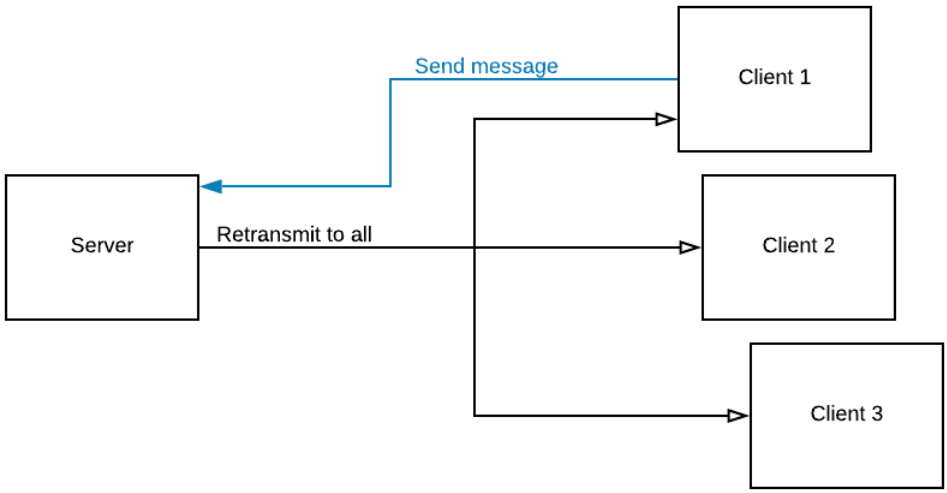


Figure 5.2 Server retransmit message to clients

In our virtual world, there are objects that can be moved or not. There is not a need to synchronize objects that are static. We will place this objects on our virtual world and will not synchronize them. On the other hand, moving objects must be synchronized across all clients. We will spawn all this objects at the beginning of the connection. These objects will be spawned by the host and then will send a message to all clients to also spawn this objects. When an object will be spawned over the network, it will get a unique id. This id will be used as reference to our messages, so we can know when we send a message to synchronize transformation what object we are targeting.

In our network authorative model each network object have an owner. Only the owner of the object can change the object transformation. When the server will spawn the network objects, will also set an owner for each one.

To synchronize our objects from host to all remote clients we need to send them messages over the network. For example to synchronize the transformations. A message to synchronize an object position contains three float values that will describe the position in the virtual environment. In order to have a smooth simulation we will to send many messages every second. In addition, his messages contains an overhead for the packet headers. To eliminate this overhead, we gather all this messages and send them in a single packet every 1ms. The

maximum size of this message will be up 1470 bytes. If the size become more that this, the remaining messages will be packed in another packet.

5.2 Synchronize VR users

To give the presence of other users in a VR collaborative environment, we present them as VR avatars. Since our VR user uses a VR headset and a set of compatibles controllers, to synchronize them in the virtual space we have to synchronize this transformations. Each user, has a “copy” of other user users headsets and controllers that represents the player.

Figure 5.3 VR users avatar representation

In our first experiments, we tried to synchronize the input. Our input in VR is the tracking of the headset and controllers and the controllers buttons. If the input was synchronized on from all clients on the server side and the VR controllers behaved as the actual server controllers, then when the client presses the grip button if an object was hovering, it would be picked up. What we noticed is that with this method, there is a big amount of delay until the user actual pick up an object. The user will press the grip button, then a message will send to the server for this event. Next, the server will update that check if an object is hovering. If it is it will picked up by the client controller. This procedure require three updates (11ms) and a message latency (minimum latency 10ms) until it is done. In our case, the minimum amount

will be 3*11 + 30 = 63ms latency. This would be the minimum response of interaction in our application.

Our solution to solve this, is instead of synchronizing the input to synchronize only the transformations 5.3. Each user will simulate on his side the simulation and update the transformations. With this method, our application response will be insignificant on the user side. For the other users, the response will be same as the network latency, minimum 30ms.

5.3 Synchronizing objects transformation

In order for all users in our collaborative virtual reality environment, to see the objects in the correct position we have to synchronize the objects transformations. Each time an object changes transformation, we will update this transformation from the object owner (the user that has the object's authority) to the host. The host then, retransmits that to the other users. If the host owns the object, the host will only update the transformation to the other clients. When our objects perform physics simulation, their position changes. We want that, to be performed only by the object owner. To achieve that, we will set these objects as static to all users that do not own it.

The transformation we want to update contains the position, rotation and scale. The position and scale is represented by three float number (x,y,z coordinates) and the rotation by a quaternion, four float numbers. In our model, when he objects are spawned, we will update all their values.

Our first try to synchronize the objects transformation, was to update it on each frame. We noticed that with this method we used bandwidth to synchronize the same transformation when the object was not moved. To solve this issue we will update the transformation, only if it is changed.

Sending the objects transformation on each frame had the effect to decrease the performance on the server side. Suppose that we have ten clients that owns one object each. That means each client will sent up to 90(our refresh rate) transformations updates per second. The server will receive 900 messages that will have to retransmit them to the other clients.

To solve that, we will update the transformation 30 or less times per second and interpolate the inbetween frames. In addition, we will send the position and rotation separately. When only the position or rotation change, we will not have to update both of them.

We will also, eliminate tiny movements in order to save more bandwidth. Will set a movement and rotation threshold that transformation smaller that this will not updated. To do that, we

will store the last value (on the objects owner side) that was send to compare each time with the new transformation changes.

5.4 Synchronizing deformable soft objects

In 5.2, we described how we update the objects transformation for rigid objects. To synchronize a soft body mesh over network, we need the soft body particles and the center object to have the same transformation in all user of our shared environment.

Figure 5.4 Soft body mesh in VR shared room

5.4.1 Synchronize soft body by the center object

When the center in our particle changes transformation, also the particles move to reach the destination based on this movement 3.2. Our first thought was to synchronize only the center object of the soft body and let the particle perform normally physics simulation in all users.

In our network model as described in 5.1, we update the transformations less times than the simulation and lerp he inbetween frames. In addition, we have an offset transformation that will be ignored. Resulting that, in our simulation we will have some different visual results in our VR users.

5.4.2 Synchronize soft body particles

Extending the method proposed in 5.4.1, we want also to update the particles changes across the virtual world. Our first thought was to synchronize the velocity changes of each particle and use as corrections to our system. What happened was that even with low latency, the deformation algorithm described in 3.2, was could not produce the same results in all users. This caused because until our algorithm get the corrections, will perform physics simulations and then even with the corrections results will be different. This could happen for example if particle is colliding with a wall.

To overcome this problem, we used the same synchronization method we used for solid objects but in all our particles, we update only the transformations. What is need with this method, is to find a way of sending not too many updated. If send massive updates our bandwidth will be extremely high in case we have many particles. We will have to use a low balance of low send intervals and high ignore transformation threshold.

6 Conclusion

6.1 Summary

The main purpose of this master thesis was to implement an algorithm for soft body mesh deformation and interaction that will be able to run in Virtual Reality collaborative environments. In this section, we will summarize the achievements of the proposed system. What was introduced can be separated in three parts. Those are:

1. **Real soft body deformations**
2. **Soft body interaction in VR environments**
3. **VR soft body interaction in networked environments.**

Each one is a piece to achieve our goal. In the next sections, we will evaluate each of them in the perspective of user experience and performance.

6.2 Evaluation

To evaluate correctly our proposed system, we will evaluate from three perspectives. Those perspectives are:

1. Performance
2. Network model evaluation
3. User experience

6.2.1 Performance evaluation

We execute performance measurements to examine our soft body deformation algorithm performance. In more detail, we measure the initialization time and the simulation time. In each of them, we simulate it with different number of vertices and particles. The workstation used to perform our timing measurements runs on Intel Core i5-8500 3.00GHz and 16.0 GB RAM 2666MHz. Each time measurement that is presented, is the average of 10 times simulations.

We begin with the evaluation of the soft body initialization time. In the following tables, the initialization time with each clustering algorithm, is measured in ms. From Table 2, Table 3 and Table 4, we saw that when we increases the particles number and the mesh vertices, the initialization time also will be increased. Also, our algorithm in 3.1.1 upper bound is

$O(n^2\log(n)/m)$ (where m is the number of vertices that each particle will contain) and in 3.1.2 and 3.1.3 the upper bound is $O(n^2\log(n))$.

Table 2 Cluster based on closest vertices initialization performance

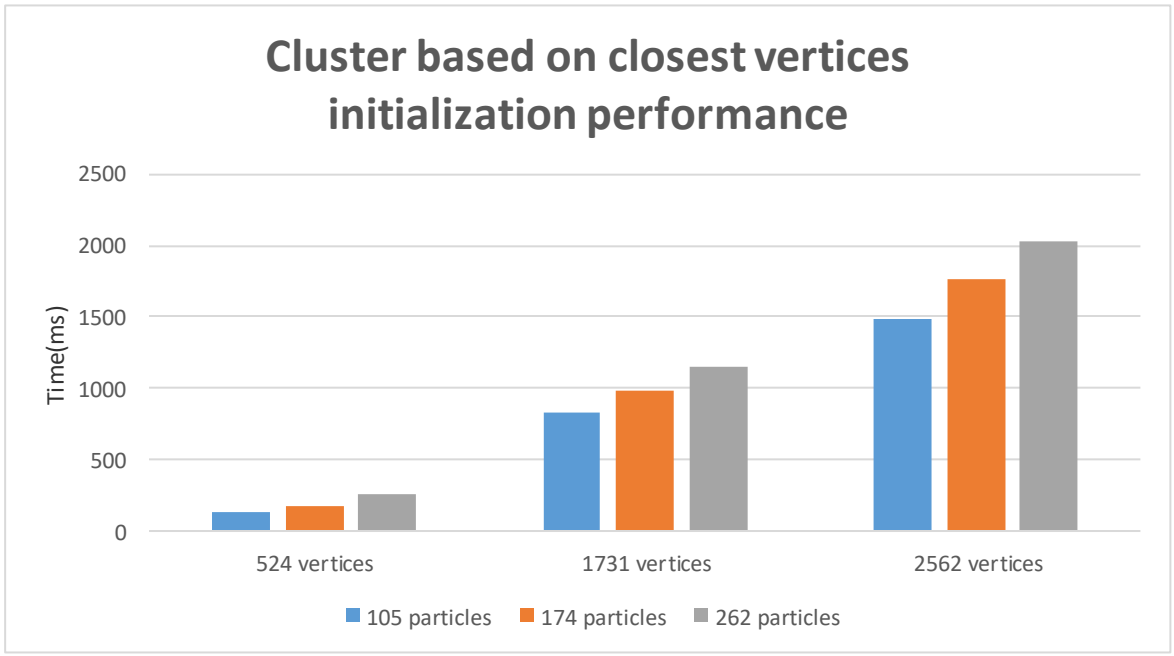


Table 3 Cluster based on distance initialization performance

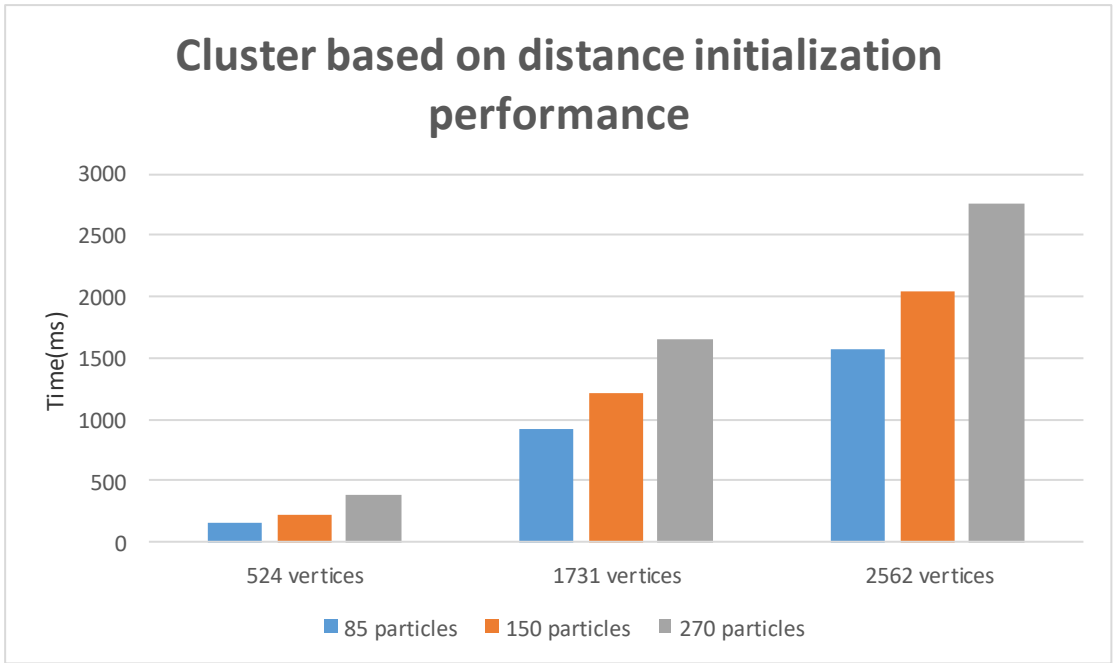
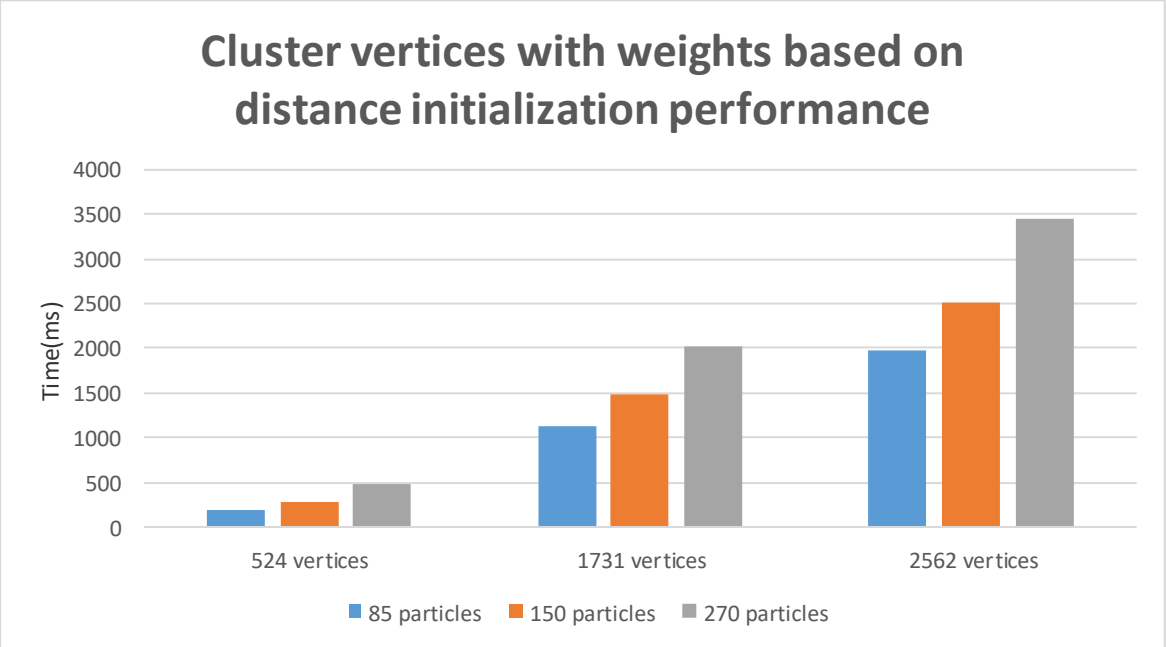


Table 4 Cluster vertices with weights based on distance initialization performance



We continue with the simulation performance measurements. We simulate each method with 3 different meshes with different vertices and on each one we will use a variety number of particles. With clustering method 3.1.1 and 3.1.2, we have the same results because it runs the same exact simulation and differs only in clustering. We recorded the minimum and maximum simulation time. From Table 5, Table 6, Table 7 and Table 8, we found out that if increase the number of particles or the vertices our simulation time also will increase. Also, because in our soft clustering a vertex can be transformed in more than our particle, the simulation time is higher.

Table 5 Simulation time measurement with 1731 vertices

Particles number	*min time(ms)*	*max time(ms)*
248	1.1	2.9
433	1.2	3.2
865	1.5	5.1

Table 6 Simulation time measurement with 1731 vertices and soft clustering

Particles number	*min time(ms)*	*max time(ms)*
215	1.2	3.5
445	1.3	4.1
872	1.6	6.3

Table 7 Simulation time measurement with 2562 vertices

Particles number	*min time(ms)*	*max time(ms)*
257	1.2	3.2
427	1.3	3.6
854	1.6	5.6

Table 8 Simulation time measurement with 2562 vertices and soft clustering

Particles number	*min time(ms)*	*max time(ms)*
224	1.3	3.5
452	1.4	3.9
863	1.7	6.2

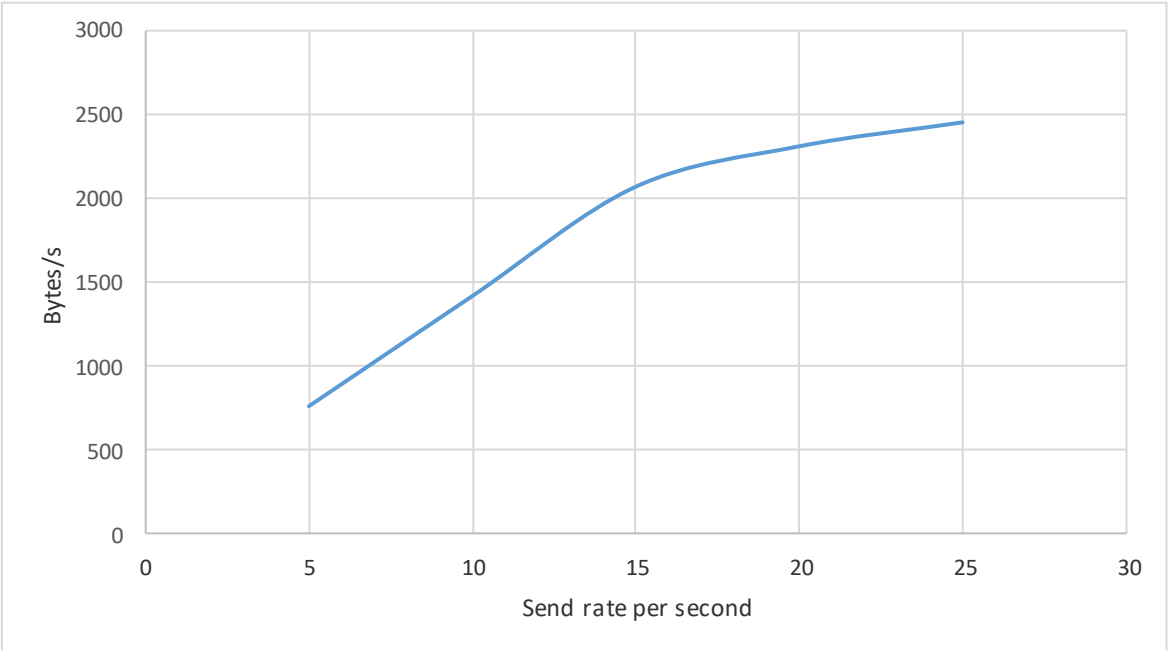
6.2.2 Network evaluation

In a collaborative VR environment the time any transformation or deformation is synchronized, is very important. Having high latency, will probably break the VR immersion. In the following tables, we will examine how the synchronizing of all transformations and deformations in our VR environment affects the bandwidth and the latency.

We begin with the bandwidth measurement of VR avatar and controllers. In Table 9, we can see the measurements of bandwidth needed per second to synchronize VR avatars and controllers with different send transform rates and low transformation ignore rate. We can observe that as we increase the send rate, that average bandwidth is not increased at the same rate.

Table 9 Synchronization measurements of VR avatar and controllers with different send rates

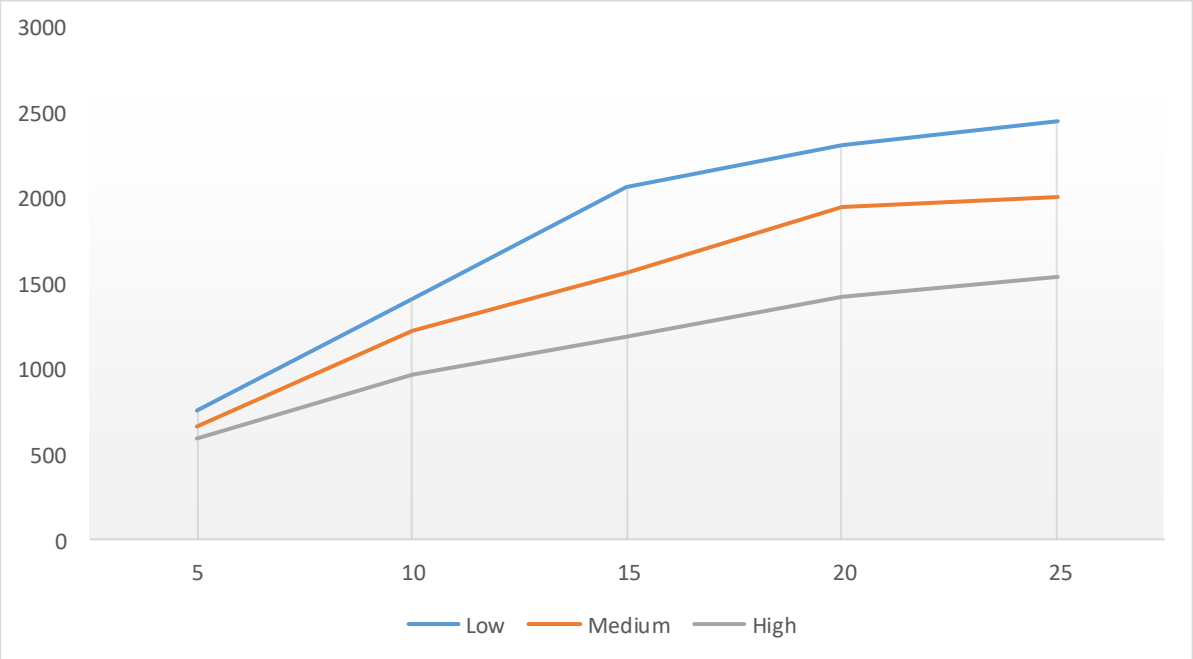
Synchronization rate per second	*Max bandwidth (bytes/s)*	*Average bandwidth (bytes/s)*
5	824	762
10	1582	1419
15	2436	2071
20	3217	2312
25	3978	2455



We continued our measurements for the VR avatars with different send rates and different ignore transformation thresholds. In Table 10, we present the average bandwidth in bytes per second with different synchronization send rate and transformation ignore threshold. We use low ignore transformation threshold 0.005 units, medium 0.015 units and high 0.025 units.

Table 10 Synchronization measurements of VR avatar and controllers with different send rates and transformation ignore threshold

Synchronization rate per second	*Average bandwidth (bytes/s) Low ignore transformation threshold*	*Average bandwidth (bytes/s) Medium ignore transfor-mation threshold*	*Average bandwidth (bytes/s) High ignore transformation threshold*
5	762	662	596
10	1419	1222	971
15	2071	1560	1196
20	2312	1951	1425
25	2455	2015	1537



Continuing, we will present the measurements of the particles synchronization. Our particles transformations rotation and scale are used only for the physics simulation and are not synchronized over the network. That means we synchronize only the position. Each message will contain the id of the soft body (4 bytes), the id of the particle (4 bytes) and the position (24 bytes). Total 32 bytes per synchronization message.

To measure the bandwidth needed we run a simulation with two users. Only one of them interacts with the soft body and the other one only observes. Also, in our measurements will include only the time that the user interact. In Table 11, we present these measurements with with different synchronization send rate and transformation ignore threshold.

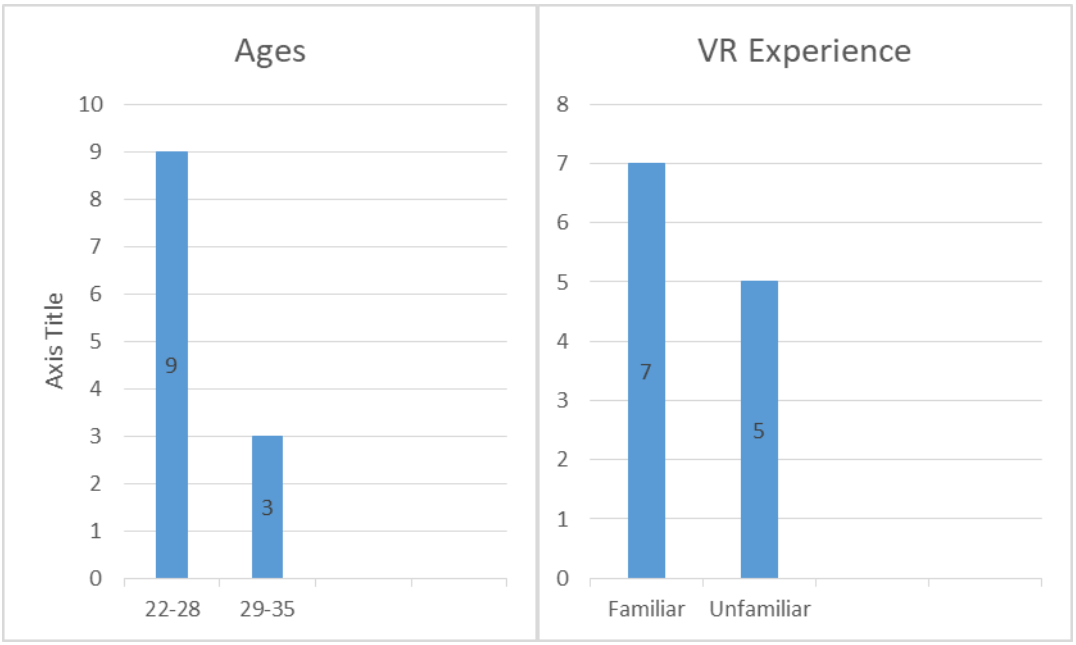
Table 11 Synchronization measurements for a soft body with 60 particles

Synchronization rate per second	*Average bandwidth (bytes/s) Low ignore transformation threshold*	*Average bandwidth (bytes/s) Medium ignore transformation threshold*	*Average bandwidth (bytes/s) High ignore transformation threshold*
5	3974	3155	2785
10	7567	5785	4917
15	11459	7687	6813
20	14895	10984	8773

6.2.3 User experience evaluation

User experience is not simple. Users comes from different backgrounds and experience. For this evaluation, we use a virtual room where the users will have to complete some scenarios. We will evaluate the user experience in the collaborative and personal environment VR environment. The application was tested from 12 participants. More specifically, 9 were between 22 and 29 years old, 7 of them were familiar with VR and 5 not.

Table 12 Participants age and VR experience



The first step was to create a standardized questionnaire for each of our scenarios for all participants. For each question, the users asked to answer their satisfaction rate from scale 1-10. Each questionnaire filled in after the end of each scenario.

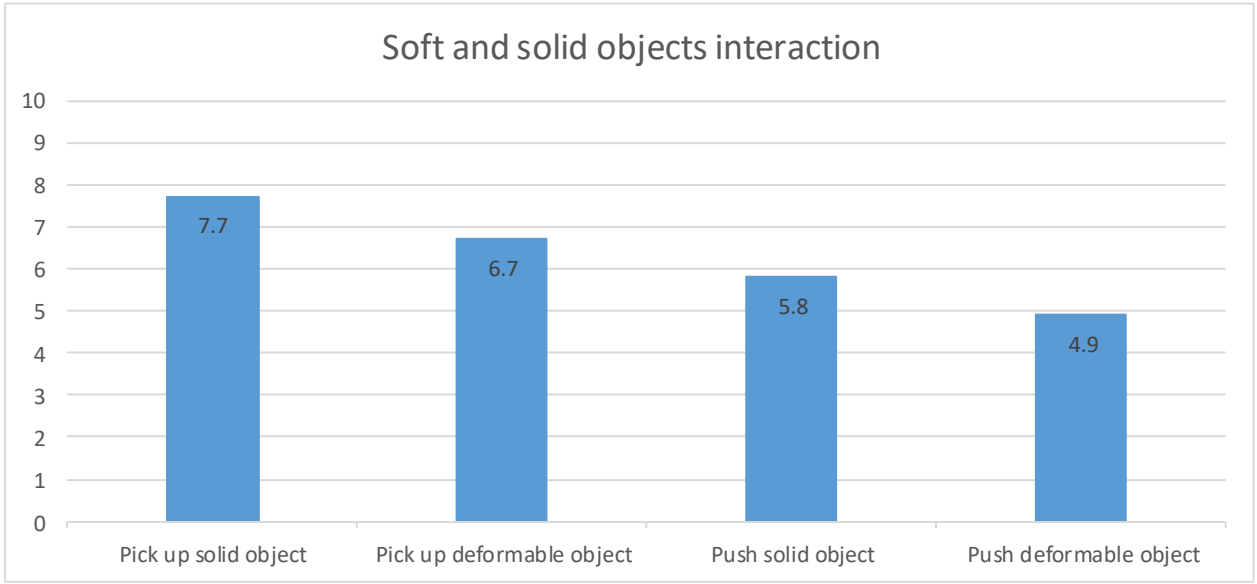
The first scenario starts with the user being in a non-collaborative virtual room. The participant had in front of him a table with deformable objects and a plate. Before the scenario starts, we explained each participants the application controls. The user asked to complete four tasks.

1. Interact with the deformable objects
2. Push the deformable objects with the virtual controllers
3. Place the plate close to the objects and push it
4. Pick up the deformable objects and put inside the plate

Then the users asked to interact with three different types of deformable objects. This procedure is done three times with different number of particles each time. We will use the same mesh with low medium and high particles (low particles = 40, medium = 60 and high = 100). Each time we used a different clustering algorithm to evaluate also our visual result of the deformable objects.

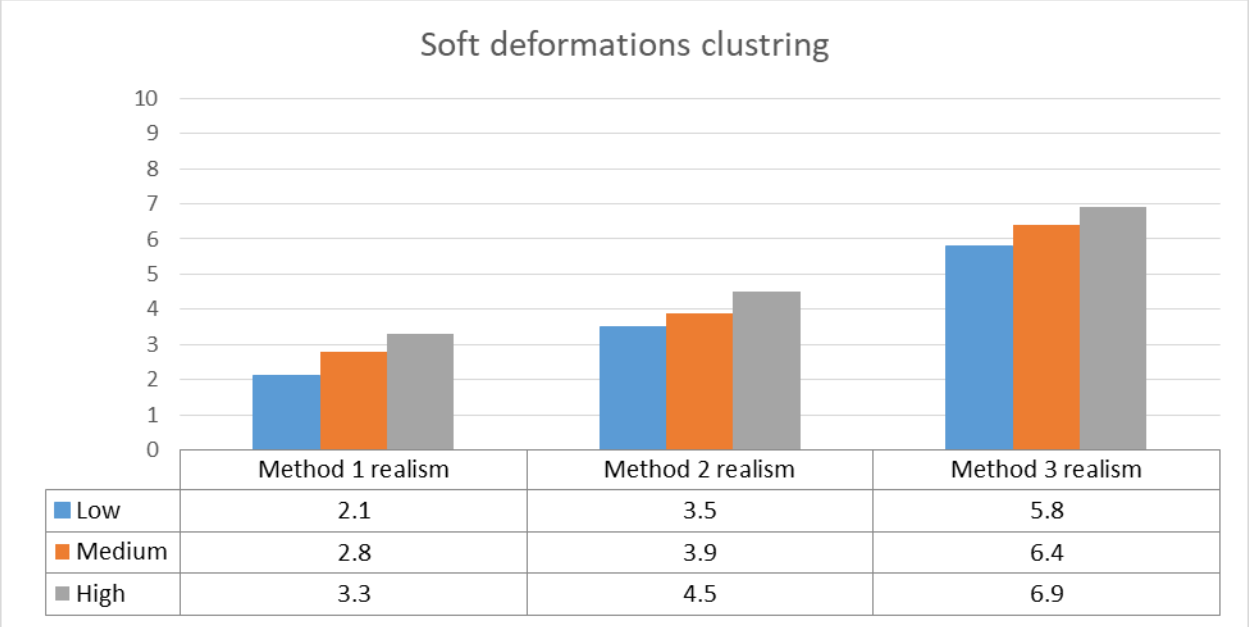
In Table 13, results are shown user satisfaction charts (1 low – 10 high). We found out that the participants found it easier to pick up or push rigid objects rather than deformables. As we noticed, that happened because when the users try to pick up they picked up the objects particles. Same on pushing objects, when the users saw resistance on the soft objects, they stopped pushing. Also, we saw that the participants when was asked to push objects they couldn't remember to press the grip button before reach the object, as a result they picked it up instead.

Table 13 Soft deformations evaluation



In results are shown user satisfaction charts (1 low – 10 high, results are shown user satisfaction charts (1 low – 10 high) about the visual result realism of each our clustering method. Where method 1 is Cluster based on closest vertices, 2 is Cluster based on distance and 3 is Cluster vertices with weights based on distance. As we can observe Method 3 has much better rating than the others do. Also we noticed that when we increase the number of particles in our simulation, the users find our soft deformation methods, more realistic.

Table 14 Soft deformation clustering method evaluation with different number of particles



	Method 1 realism	Method 2 realism	Method 3 realism
Low	2.1	3.5	5.8
Medium	2.8	3.9	6.4
High	3.3	4.5	6.9

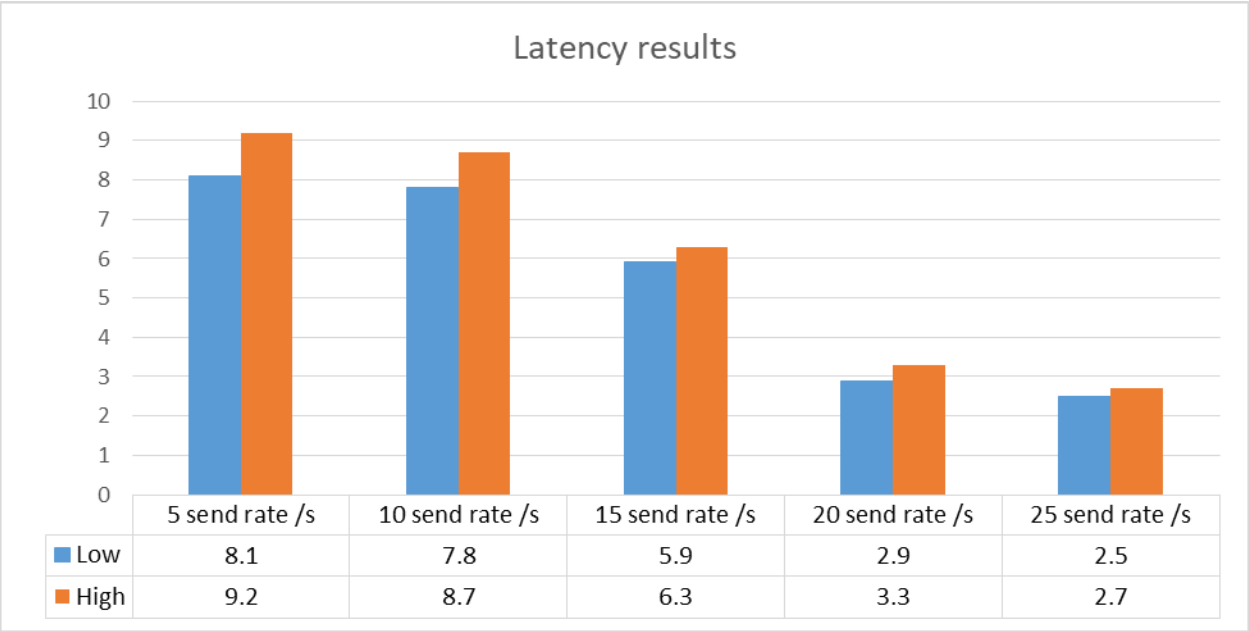
The second scenario purpose is to evaluate our method in a VR collaborative environment. In this scenario, we execute the same steps as in non-collaborative environment (above), but with many participants. On each evaluation was 2-4 users connected. Each time, one of them had to execute the scenario steps and the others was observing him. Then the next one execute the steps.

This procedure was executed 10 times with different synchronization rate (5, 10, 15, 20 and 25 times per second) and ignore threshold. We use low ignore transformation threshold 0.005 units and high 0.025 units. After each procedure was asked 3 questions about his satisfaction. The first one was if the user experienced latency, the second one if other user avatars was natural synchronized and if the soft deformations when the other users was interacting with them was natural.

Table 15 shows user evaluation about latency in the application in scale of 1-10 (1 not latency at all). Some interesting results we can extract are that as we increase the transformations send rate, users feels less latency. What is interesting is that after 20 send rate/s, there

is a very small difference in participants latency feeling. Also, with lower transformation ignore threshold, users feels less latency.

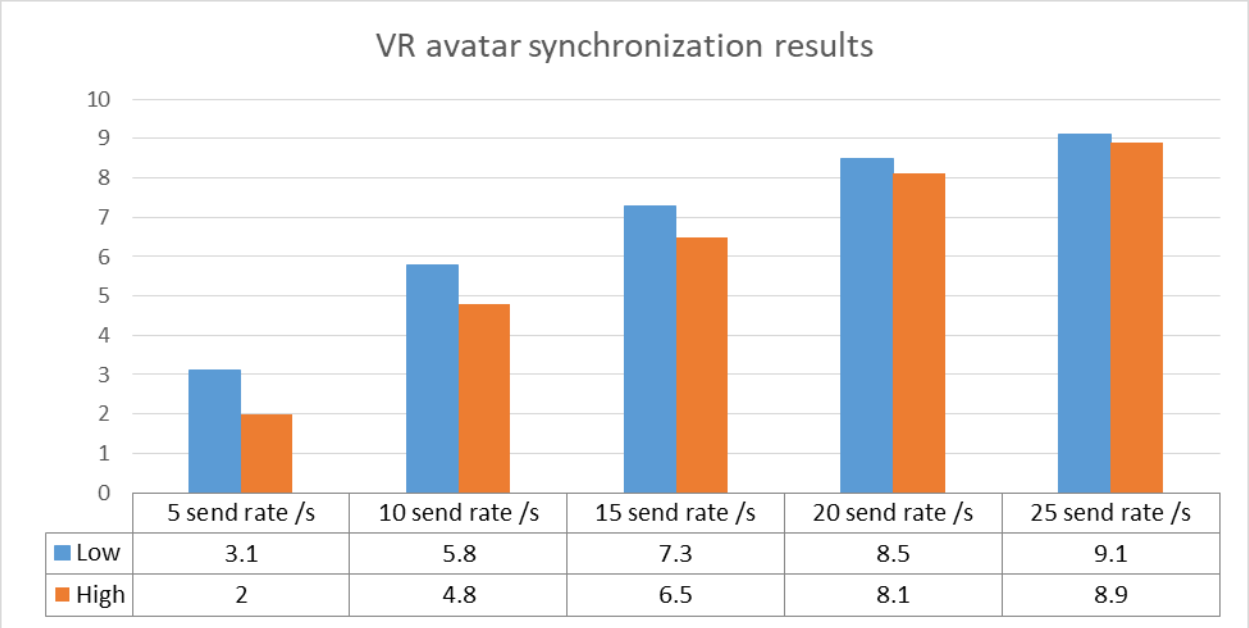
Table 15 Latency user evaluation



	5 send rate /s	10 send rate /s	15 send rate /s	20 send rate /s	25 send rate /s
Low	8.1	7.8	5.9	2.9	2.5
High	9.2	8.7	6.3	3.3	2.7

Table 16, shows user satisfaction on how natural the VR avatars are synchronized (1 very natural, 10 not natural at all). As the above, the information we can extract is that as we increase the send rate /s the user see transformations more natural.

Table 16 VR avatar synchronization evaluation

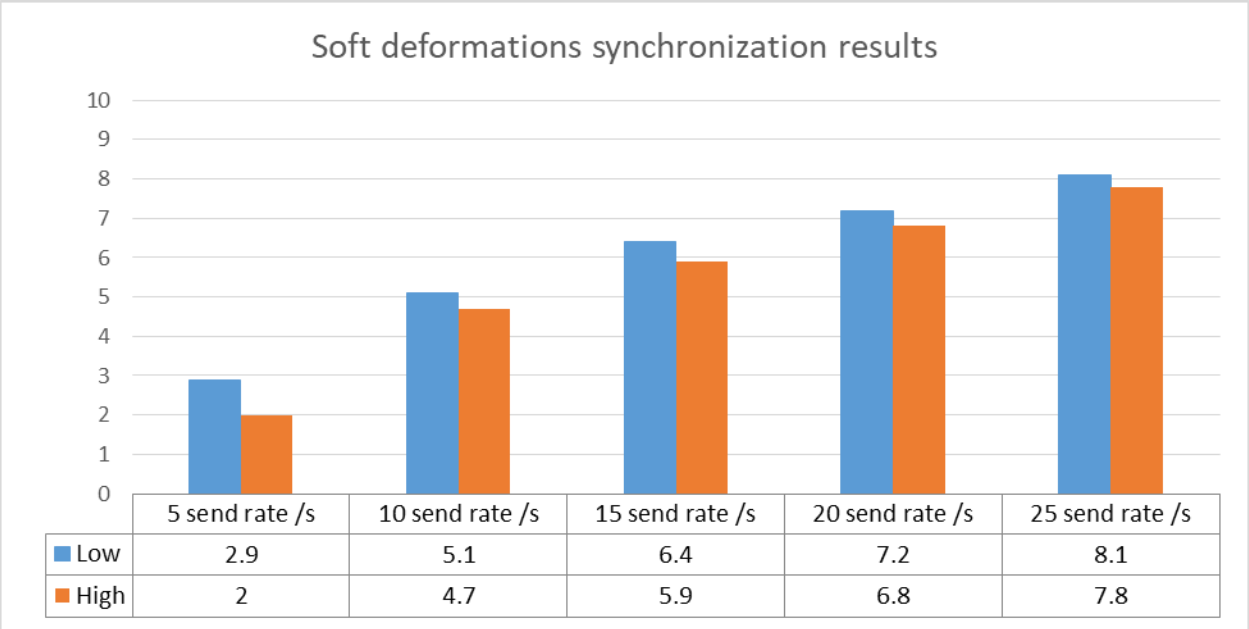


	5 send rate /s	10 send rate /s	15 send rate /s	20 send rate /s	25 send rate /s
Low	3.1	5.8	7.3	8.5	9.1
High	2	4.8	6.5	8.1	8.9

In Table 17, results are shows how natural the participants believe the soft body deformations synchronization are. The results are shown in scale 1-10 where 1 is not natural and

10 natural. Again, what we can observe is that as we increase the transformations send rate, users find the synchronization more natural and that after 20 send rate/s, there is a very small increase. What also is inserting here, is that even we are using the same synchronization rate for avatar and deformable objects, the users believes that soft objects was less natural synchronized.

Table 17 Soft body deformation synchronization, user evaluation



	5 send rate /s	10 send rate /s	15 send rate /s	20 send rate /s	25 send rate /s
Low	2.9	5.1	6.4	7.2	8.1
High	2	4.7	5.9	6.8	7.8

6.3 Limitations

However, this study is not without its limitations that we aim to solve in the near future.

- Change object authority
  Currently our objects cannot be passed by one user to an other
- Interaction with objects that the user doesn't have authority
  Users cannot interact with objects that they don't own

6.4 Future Work

In the future, we aim to improve our proposed system. Our development in the future will be focused more on system performance optimizations, deformation variety and network bandwidth-latency reduction.

- Cutting and tearing
  Since our soft meshes are based particles, we want to develop an algorithm for cutting and

tearing based on particles. Our particles as presented in this thesis are already connected with vertices. Finding the cutting or tearing segments from particles collisions, will allow us to recreate the geometry of our mesh only in this areas with a new one that will contain cutting or tearing results.

- Deformation for skinned mesh

  Next we want to continue with integrate to our system skinned meshes soft deformations. We want to keep the animations of the skinned meshes but at the same time the mesh vertices to be simulated as soft also. To achieve that, what we want to integrate to our system is to connect skinned mesh bones transformations.

- Synchronization transformations accuracy

  In our system we are using a transformation ignore rate. That means that if an object is moved in this rate will not be synchronized. We want to implement a method that will observe the objects to see if stop transforming for a while. If yes will check if the current transform is the one that was update and if it is not will update it to other users. This will increase our system accuracy in some cases without losing bandwidth,

- Network data compression

  Finally, we want to integrate a method for data compression. Data compression will enable our system to decrease the network bandwidth. By decreasing the bandwidth, we will be able to increase the send transformation rate of our system. Resulting we will be able to have better accuracy on our synchronization methods.

References

[1] Karolina Golec. *Hybrid 3D Mass Spring System for Soft Tissue Simulation. Modeling and Simulation.* Universite Lyon 1, 2018. English. fftel-01761851v1f

[2] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, Alan H. Barr. *Adaptive Simulation of Soft Bodies in Real-Time.* Computer Animation 2000, 2000, Philadelphie, United States. pp.133-144. ffinria-00510054f

[3] Kenwright Ben, Davison Richard, Morgan Graham. (2011). *Real-Time Deformable Soft-Body Simulation using Distributed Mass-Spring Approximations.*

[4] Junggon Kim and Nancy S. Pollard, *Fast Simulation of Skeleton-driven Deformable Body Characters*, ACM Transactions on Graphics, Volume 30, Issue 5, 2011.

[5] Danevicius, Edvinas & Maskeliunas, Rytis & Damasevicius, Robertas & Połap, Dawid & Woźniak, Marcin. (2018). *A Soft Body Physics Simulator with Computational Offloading to the Cloud.* Information (Switzerland). 9. 318. 10.3390/info9120318.

[6] Mesit, Jaruwan. (2011). *A general model for soft body simulation in motion*. Proceedings - Winter Simulation Conference. 2685-2697. 10.1109/WSC.2011.6147975.

[7] Terzopoulos, Demetri & Platt, John & Barr, Alan & Fleischer, Kurt. (1987). Elastically Deformable Models. ACM Siggraph Computer Graphics. 21. 10.1145/37402.37427.

[8] Sarah F F Gibson and Brian Mirtich. *A Survey of Deformable Modeling in Computer Graphics*. In: Mitsubishi Electric Information Technology Center America (Nov.1997).

[9] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. *Physically Based Deformable Models in Computer Graphics*. In: Computer Graphics Forum 25.4 (Dec. 2006), pp. 809–836. ISSN: 0167-7055

[10] Zhang, Jinao & Zhong, Yongmin & Gu, Chengfan. (2017). *Deformable Models for Surgical Simulation: A Survey*. IEEE Reviews in Biomedical Engineering. 11. 143-164. 10.1109/RBME.2017.2773521.

[11] P. Moore and D. Molloy, *A Survey of Computer-Based Deformable Models*, International Machine Vision and Image Processing Conference (IMVIP 2007), Kildare, 2007, pp. 55-66. doi: 10.1109/IMVIP.2007.31

[12] Vassilev, Tzvetomir & Spanlang, Bernhard. (2002). *A mass-spring model for real time deformable solids*.

[13] Camara, Mafalda & C., Susmita & Darzi, Ara & Pratt, Philip. (2016). *Soft tissue deformation for surgical simulation: a position-based dynamics approach.* International Journal of Computer Assisted Radiology and Surgery. 11. 919–928. 10.1007/s11548-016-1373-8.

[14] Steve Benford, Chris Greenhalgh, Tom Rodden, and James Pycock et al. *Collaborative Virtual Environments*.In: Commun. ACM 44.7 (July 2001), pp. 79–85. issn: 0001-0782. doi: 10.1145/379300.379322.

[15] Molet, Tom & Aubel, Amaury & Capin, Tolga & Carion, Stéphane & Lee, Elwin & Thalmann, Nadia & Noser, Hansrudi & Pandžić, Igor & Sannier, Gaël & Thalmann, Daniel. (1999). *Anyone for Tennis?.* Presence: Teleoperators and Virtual Environments. 8. 140-156. 10.1162/105474699566134.

[16] Jackson, Randolph & Fagan, Eileen. (2000). *Collaboration and learning within immersive virtual reality*. Proceedings of the Third International Conference on Collaborative Virtual Environments. 10.1145/351006.351018.

[17] Monahan, Teresa & McArdle, Gavin & Bertolotto, Michela. (2008). *Virtual reality for collaborative e-learning*. Computers & Education. 50. 1339-1353.10.1016 /j.compedu.2006.12.008.

[18] Ralf A. Kockro, Axel Stadie, Eike Schwandt, Robert Reisch, Cleopatra Charalampaki, Ivan Ng, Tseng Tsai Yeo, Peter Hwang, Luis Serra, Axel Perneczky, *A Collaborative Virtual Reality Environment for Neurosurgical Planning and Training*, Operative Neurosurgery, Volume 61, Issue suppl_5, 1 September 2007, Pages ONSE379–ONSE391

[19] C.Oliveira, Jauvane et al. "*Collaborative Virtual Environments for Industrial Training and e-Commerce*." (2002).

[20] George Papagiannakis, Nick Lydatakis, Steve Kateros, Stelios Georgiou, and Paul Zikas. *Transforming medical education and training with VR using M.A.G.E.S.* . In SIGGRAPH Asia 2018 Posters, SA '18, pages 83:1-83:2, New York, NY, USA, 2018. ACM.

[21] P Zikas, V. Bachlitzanakis, M. Papaefthymiou, S. Kateros, S. Georgiou, N. Lydatakis, and G. Papagiannakis. *Mixed Reality Serious Games and Gamification for smart education*. In European Conference on Games Based Learning 2016. ECGBL'16, 2016.

[22] Jorissen, Pieter & Wijnants, Maarten & Lamotte, Wim. (2005). *Dynamic Interactions in Physically Realistic Collaborative Virtual Environments*. IEEE transactions on visualization and computer graphics. 11. 649-60. 10.1109/TVCG.2005.100.

[23] Elvezio, Carmine & Ling, Frank & Liu, Jen-Shuo & Feiner, Steven. (2018). *Collaborative Virtual Reality for Low-Latency Interaction.* 179-181. 10.1145/3266037.3271643.

[24] Sumengen, Selcuk & Eren, Mustafa & Yesilyurt, Serhat & Balcisoy, Selim. (2007). *Real-time deformable objects for collaborative virtual environments.* GRAPP 2007 - 2nd International Conference on Computer Graphics Theory and Applications, Proceedings. 121-128.

[25] Z. Tang, Y. Yang, X. Guo and B. Prabhakaran, "On supporting collaborative haptic interactions with physically-based 3D deformations," 2010 IEEE International Symposium on

Haptic Audio Visual Environments and Games, Phoenix, AZ, 2010, pp. 1-6. doi: 10.1109/HAVE.2010.5623988

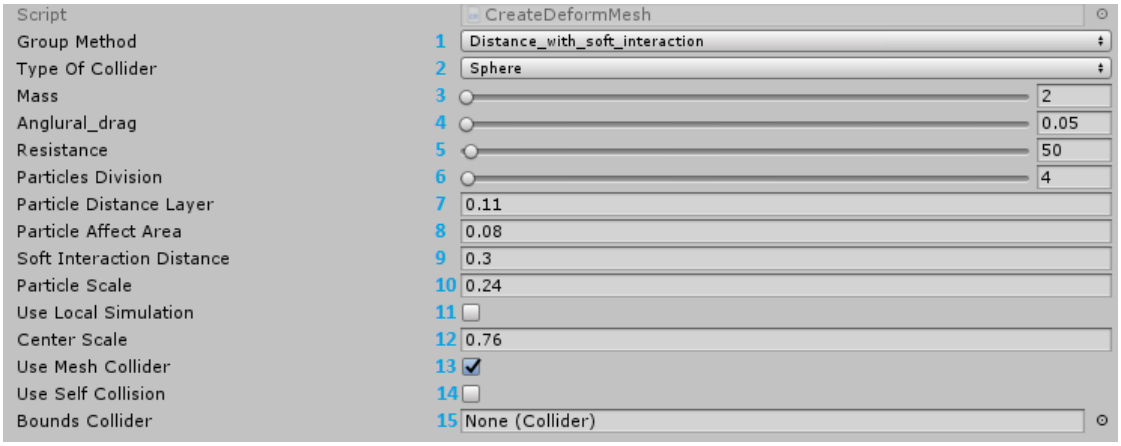
[26] Riffnaller-Schiefer, Andreas & Augsdörfer, Ursula & Fellner, Dieter. (2017). *Physics-based Deformation of Subdivision Surfaces for shared Virtual Worlds*. Computers & Graphics. 71. 10.1016/j.cag.2017.12.005.

[27] Chheang, Vuthea & Saalfeld, Patrick & Huber, Tobias & Huettl, Florentine & Kneist, Werner & Preim, Bernhard & Hansen, Christian. (2019). *Collaborative Virtual Reality for Laparoscopic Liver Surgery Training*.

[28] Qin, J., Choi, K. & Heng, P. *Collaborative Simulation of Soft-Tissue Deformation for Virtual Surgery Applications*. J Med Syst 34, 367–378 (2010) doi:10.1007/s10916-008-9249-2

Appendix A

Soft mesh deformations



Component to create a deformable soft mesh

1. Particle grouping method selection
   ```
   enum ParticleGroupMehod {distance,closest_point, ditance_with_soft_interaction};
   ```
2. Type of particles collider
3. Mass per particle
4. Particles Angular drag
5. Particles resistance
6. Particles Division for grouping based on closest point
7. Particles creation distance for clustering
8. Particle affection range
9. Particles soft interaction distance
10. Particle scale selection
11. Simulate in local space
12. Center object scale
13. Use mesh collider for center selection
14. Enable self-collisions
15. Restrict soft body in a selected area based on collider

Soft mesh deformations algorithm

1. Cluster based on closest vertex

```
   void Group_By_Closest_N_Points(Mesh mesh,ref Dictionary<int, bool> meshIDs)
   {
       //cache all mesh vertices
       var _vertices = mesh.vertices;

       int particleID = 0;
       for (int i = 0; i < _vertices.Length; i++)
       {
           //continue if current mesh is already clustered
           if (!meshIDs.ContainsKey(i))
               continue;
           //Create the particle
           GameObject g = GameObject.CreatePrimitive(TypeOfCollider);
           g.transform.localScale = ParticleScale * Vector3.one ;
           Destroy(g.GetComponent<MeshFilter>());
           Destroy(g.GetComponent<Renderer>());
           g.name = "ID" + particleID;
           g.transform.SetParent(this.transform,false);
           g.transform.localPosition = _vertices[i];

           Rigidbody rb = g.AddComponent<Rigidbody>();
           rb.isKinematic = false;
           rb.useGravity = false;
           rb.mass = mass;
           rb.angularDrag = anglural_drag;
           particleID++;
           ParticleHelper ph = g.AddComponent<ParticleHelper>();
           ph.Rb = rb;
           //store the current vertex to the particle and remove it from the
dictionary
           ph.AddAffectedVertex(i, 1.0f);
           meshIDs.Remove(i);
           particlesList.Add(ph);
           ph.BaseVertexID = i;
           if (ParticlesDivision == 1)
               continue;
           var sortedBydistance = meshIDs.ToList();

           //store all vertices ID sorted by ascending distance with current
vertex
           //Sort comparison: Distance(currentVertex,c1.vertex) < Dis-
       tance(currentVertex,c2.vertex)
           sortedBydistance.Sort(delegate (KeyValuePair<int, bool> c1, KeyValue
                                   Pair<int, bool> c2)
           {
               return Vector3.Distance(_vertices[i],
                                       _vertices[c1.Key]).CompareTo
                           ((Vector3.Distance(_vertices[i],
                            _vertices[c2.Key])));
                            });

           //Add the next N vertices to the particle and remove them from the
              dictionary
           for (int j = 0; (j < ParticlesDivision - 1) && (j < meshIDs.Count);
                     j++)
           {
```

```
            ph.AddAffectedVertex(sortedBydistance.ElementAt(j).Key, 1.0f);
            meshIDs.Remove(sortedBydistance.ElementAt(j).Key);
        }
    }
}
```

2. Cluster based on distance

```
void Group_By_Distance(Mesh mesh,ref Dictionary<int, bool> meshIDs)
    {
        var _vertices = mesh.vertices;

        int particleID = 0;
        for (int i = 0; i < _vertices.Length; i++)
        {
            if (!meshIDs.ContainsKey(i))
                continue;
            if (meshIDs[i] == true)
                continue;
            GameObject g = GameObject.CreatePrimitive(TypeOfCollider);
            g.transform.localScale = ParticleScale* Vector3.one /
                                                (ParticlesDivision);
            Destroy(g.GetComponent<MeshFilter>());
            Destroy(g.GetComponent<Renderer>());
            g.name = "ID" + particleID;
            g.transform.parent = this.transform;
            g.transform.localPosition = _vertices[i];
            Rigidbody rb = g.AddComponent<Rigidbody>();
            rb.isKinematic = false;
            rb.useGravity = false;
            rb.mass = mass;
            rb.angularDrag = anglural_drag;
            rb.collisionDetectionMode =
                        CollisionDetectionMode.ContinuousDynamic;
            particleID++;

            ParticleHelper ph = g.AddComponent<ParticleHelper>();
            ph.Rb = rb;
            ph.AddAffectedVertex(i, 1.0f);
            meshIDs[i] = true;
            meshIDs.Remove(i);
            particlesList.Add(ph);

            ph.BaseVertexID = i;

            var sortedBydistance = meshIDs.ToList();

            sortedBydistance.Sort(delegate (KeyValuePair<int, bool> c1,
                                KeyValuePair<int, bool> c2)
            {
                return Vector3.Distance(_vertices[i],
                        _vertices[c1.Key]).CompareTo
                           ((Vector3.Distance(_vertices[i],
                        _vertices[c2.Key])));
            });

            for (int j = 0; (j < meshIDs.Count); j++)
            {
```

```
                float currParticleDist = currParticleDist = Vec-
            tor3.Distance(_vertices[i],
                          _vertices[sortedBydistance.ElementAt(j).Key]);


            if(ph.affectedVertices.ContainsKey(sortedBydistance.ElementAt(j).K
            ey))
                {
                    continue;
                }

                //Layer1
                if (currParticleDist <= ParticleDistanceLayer)
                {
                    ph.AddAffectedVertex(sortedBydistance.ElementAt(j).Key,
                                                                    1.0f);
                    meshIDs[sortedBydistance.ElementAt(j).Key] = true;
                    meshIDs.Remove(sortedBydistance.ElementAt(j).Key);
                }
                else if (currParticleDist > ParticleDistanceLayer)
                {
                    break;
                }
            }
        }
    }
```

3. Cluster based on closest

```
   void Group_By_Distance_with_soft_interaction(Mesh mesh, ref Dictionary<int,
bool> meshIDs)
    {
        var _vertices = mesh.vertices;

        int particleID = 0;
        for (int i = 0; i < _vertices.Length; i++)
        {
            if (!meshIDs.ContainsKey(i))
                continue;
            if (meshIDs[i] == true)
                continue;
            GameObject g = GameObject.CreatePrimitive(TypeOfCollider);
            g.transform.localScale = ParticleScale * Vector3.one /
                                        (ParticlesDivision);
            Destroy(g.GetComponent<MeshFilter>());
            Destroy(g.GetComponent<Renderer>());
            g.name = "ID" + particleID;
            g.transform.parent = this.transform;
            g.transform.localPosition = _vertices[i];
            Rigidbody rb = g.AddComponent<Rigidbody>();
            rb.isKinematic = false;
            rb.useGravity = false;
            rb.mass = mass;
            rb.angularDrag = anglural_drag;
            rb.collisionDetectionMode =
                  CollisionDetectionMode.ContinuousDynamic;
            particleID++;

            ParticleHelper ph = g.AddComponent<ParticleHelper>();
            ph.Rb = rb;
```

```
                ph.AddAffectedVertex(i, 1.0f);
                meshIDs[i] = true;
                particlesList.Add(ph);

                ph.BaseVertexID = i;

                var sortedBydistance = meshIDs.ToList();

                sortedBydistance.Sort(delegate (KeyValuePair<int, bool> c1,
                      KeyValuePair<int, bool> c2)
                {
                    return Vector3.Distance(_vertices[i],
                       _vertices[c1.Key]).CompareTo
                                ((Vector3.Distance(_vertices[i],
                       _vertices[c2.Key])));
                });

                for (int j = 0; (j < meshIDs.Count); j++)
                {
                    float currParticleDist = currParticleDist =
                       Vector3.Distance(_vertices[i],
                      _vertices[sortedBydistance.ElementAt(j).Key]);

                    if
(ph.affectedVertices.ContainsKey(sortedBydistance.ElementAt(j).Key))
                    {
                        continue;
                    }

                    //Layer1
                    if (currParticleDist <= ParticleDistanceLayer)
                    {
                        ph.AddAffectedVertex(sortedBydistance.ElementAt(j).Key,
                                                                    1.0f);
                        meshIDs[sortedBydistance.ElementAt(j).Key] = true;
                    }
                    else if (currParticleDist <= SoftInteractionDistance)
                    {
                        ph.AddAffectedVertex(sortedBydistance.ElementAt(j).Key,
                                                                    1.0f);
                    }
                    else if (currParticleDist > ParticleDistanceLayer)
                    {
                        break;
                    }
                }
            }
    }
```

4. Center particle

```
    public void CenterParticle()
    {
        Vector3 pos = Vector3.zero;
        var meshVertices = meshContainer.vertices;
        foreach (KeyValuePair<int, float> entry in affectedVertices)
        {
            pos += meshVertices[entry.Key];
        }

        pos = pos / (float)affectedVertices.Count;
        this.transform.localPosition = pos;

    }
```

5. Create particle interaction weights

```
void CreateParticleInteraction()
{
    float baseDist = ParticleAffectArea;
    foreach (ParticleHelper p1 in particlesList)
    {
        foreach (ParticleHelper p2 in particlesList)
        {
            if (p1 == p2) continue;
            float curDist = Vector3.Distance(
                  p1.transform.position, p2.transform.position);
            if (curDist <= baseDist)
            {
                p1.AddAffectedParticle(p2, 1.0f - ( curDist / baseDist));
            }
        }
    }
}
```

6. Calculate particle velocity based on center movement

```
private void CalculateParticleVelocity(Vector3 pos,
Quaternion rot,bool applyToOtherParticles = true,
bool applyAfterFixedUpdate = false,float affect_value = 1.0f)
{
    Vector3 dest =  (initPosition + centerPoint.position);
    dest = DeformableHelper.RotatePointAroundPivot
           (dest, centerPoint.position, centerPoint.rotation);
    Vector3 direction = dest - pos;
    float expectedDelta = Time.fixedTime / 0.0111112f;

    if (applyAfterFixedUpdate)
        AddExternalVelocity(pos);
    else
        baseVelocityToApply = direction;
```

```
    if (applyToOtherParticles)
    {
        for(int i =0;i<affectedParticlesArray.Length;i++)
        {
                affectedParticlesArray[i].Key.ApplyExtraForceToOtherParticle
                  (this, direction * affectedParticlesArray[i].Value, rot,
           affectedParticlesArray[i].Value);
        }
    }
}
```

7. Update particle velocity

```
public void UpdateVelocities()
{
    if (velocityToApply != Vector3.zero)
    {
        baseVelocityToApply = baseVelocityToApply - velocityToApply;
    }
    float expectedDelta = Time.fixedDeltaTime / 0.0111112f;

    Rb.velocity = Vector3.MoveTowards(Rb.velocity, baseVelocityToApply *
           resistance * expectedDelta, 10.0f);
    velocityToApply = Vector3.zero;
    baseVelocityToApply = Vector3.zero;
}
```

8. Solve vertex movement

```
void SolveVertexPosition (Vector3 movement)
{
    for(int i = 0; i < affectedVerticesArray.Length; i++)
    {
        deformCreator.temp_mesh_verticies[affectedVerticesArray[i].Key] +=
                        (movement * affectedVerticesArray[i].Value);
    }
}
```

9. Remove base mesh properties

```
void FixBaseScaleAndRotation()
{
    Quaternion newRotation = this.transform.rotation;
    Vector3 local_scale = transform.localScale;
    if (GetComponent<MeshFilter>())
        mesh = GetComponent<MeshFilter>().mesh;

    Matrix4x4 trs = Matrix4x4.TRS(Vector3.zero, newRotation, local_scale);
    baseTRS = trs;
    Vector3[] vertices = mesh.vertices;
    for (int i = 0; i < vertices.Length; i++)
    {
        vertices[i] = trs.MultiplyPoint3x4(vertices[i]) ;
    }
```

```
    for (int i = 0; i < mesh.normals.Length; i++)
    {
        mesh.normals[i] = newRotation * (mesh.normals[i]);
    }
    for (int i = 0; i < mesh.tangents.Length; i++)
    {
        mesh.tangents[i] = newRotation * (mesh.tangents[i]);
    }
    for (int i = 0; i < mesh.tangents.Length; i++)
    {
        mesh.tangents[i] = newRotation * (mesh.tangents[i]);
    }

    mesh.vertices = vertices;
    mesh.RecalculateBounds();
    mesh.RecalculateNormals();
    mesh.RecalculateTangents();

    transform.rotation = Quaternion.identity;
    transform.localScale = Vector3.one;
}
```

10. Disable self-collisions

```
void DisableSelfCollision()
{
    foreach (ParticleHelper _ph1 in particlesList)
    {
        foreach (ParticleHelper _ph2 in particlesList)
        {
            if(_ph1!=_ph2)
            {
                if(_ph1.GetComponent<Collider>() &&
                  _ph2.GetComponent<Collider>())
                Physics.IgnoreCollision(_ph1.GetComponent<Collider>(),
                          _ph2.GetComponent<Collider>());
            }
        }
        if (_ph1.GetComponent<Collider>() && center.GetComponent<Collider>())
        {
            Physics.IgnoreCollision(_ph1.GetComponent<Collider>(),
                                center.GetComponent<Collider>());
        }

    }
}
```

11. Cache affected vertices for each particle to calculate center pivot for rotation

---

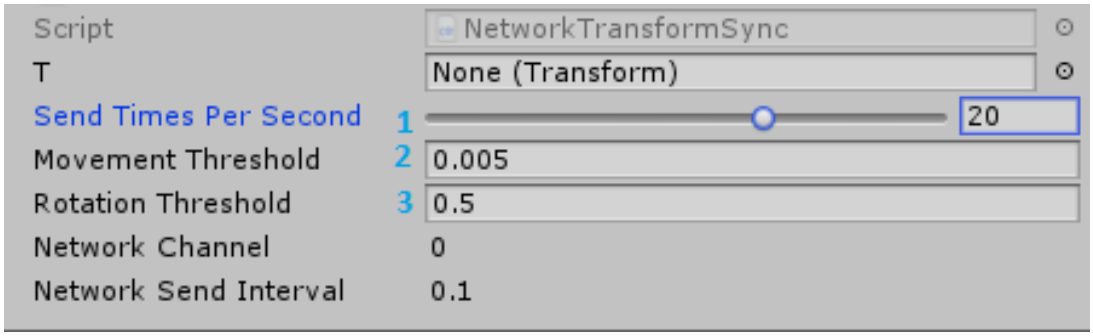
```
void ComputeParticleAffectedVertexMapping()

     ParticleHelper[][] particlesRotationMaping =
                 new ParticleHelper[mesh.vertices.Length][];
        for (int i = 0; i < mesh.vertices.Length; i++)
```

```
        {
            List<ParticleHelper> current_particle_id =
                  new List<ParticleHelper>();

            foreach (ParticleHelper _ph in particlesList)
            {

                if (_ph.affectedVertices.ContainsKey(i))
                {
                    current_particle_id.Add(_ph);
                }
            }
            particlesRotationMaping[i] = current_particle_id.ToArray();
        }
```

12. Compute rotation applied to each vertex

```
if (Quaternion.Angle(lastRotation, center.transform.rotation) > 0.5f)
{
    Quaternion rotationQuaternion = (center.transform.rotation *
                Quaternion.Inverse(lastRotation));
    for (int i = 0; i < temp_mesh_verticies.Length; i++)
    {
        Vector3 pivot_for_rotation = Vector3.zero;

        for (int j = 0; j < particlesRotationMaping[i].Length; j++)
        {
            pivot_for_rotation +=
           particlesRotationMaping[i][j].transform.localPosition *
           particlesRotationMaping[i][j].affectedVertices[i];
        }
        temp_mesh_verticies[i] =
        DeformableHelper.RotatePointAroundPivot
           (temp_mesh_verticies[i], pivot_for_rotation, rotationQuaternion);
    }
    lastRotation = center.transform.rotation;
}
```

Network transform synchronization component



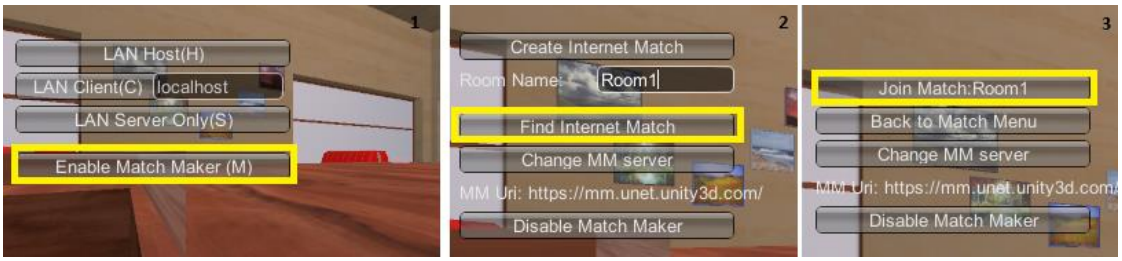
1. Transformation send rate per second
2. Position ignore threshold
3. Rotation ignore threshold

Appendix 2 – User guide

Here we will provide the steps needed to run the application and connect with other VR participants.

1. Room creation (The participant that creates the room is also the host)

2. VR participant connect



3. Ready to start (other participants can join during the simulation)

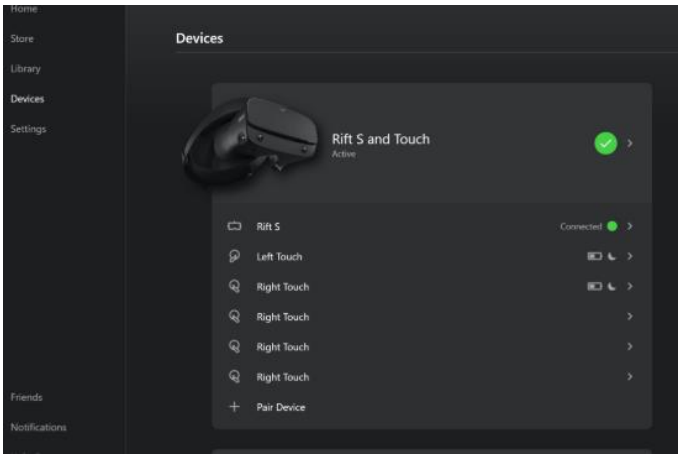
Appendix 3 – Installation guide

Our system require the installation of:

1. Oculus Home (only for oculus devices)

   1.1 Download and install oculus home https://www.oculus.com/setup/

   1.2 Set up the device according to the installation quid



2. SteamVR

   2.1 Download and install https://store.steampowered.com/app/250820/SteamVR/

   2.2 Setup in SteamVR headset and pair controllers