Sculpting in Virtual Reality with Real Human Hands

by

Salar Rezayani

A thesis submitted to the
Department of Computer Science
in conformity with the requirements for
the degree of Master of Science

Bishop's University
Canada
August 2023

Abstract

VR sculpting stands at the intersection of technology and art, with the potential to revolutionize creative expression and digital design. The challenge lies in establishing realistic and intuitive interaction with the virtual world objects. This thesis presents several innovative approaches that utilize hand joint positions from a robust tracking package to replicate the nuanced interactions of physical sculpting within the virtual realm. We test three basic approaches: 1. A spring-like motion simulation for vertex manipulation around objects, 2. A Minecraft-style approach involving manipulation of small cubes and 3. Fluid dynamics using a lightweight neural network. We also investigated Entity Component System (ECS) for efficient memory chunk management. Preliminary experiments displayed promising results in terms of precision, usability, and computational efficiency, suggesting that our method could significantly advance the current state of VR sculpting.

Acknowledgments

I am deeply grateful to the Computer Science department at Bishop's University for providing me with the wonderful opportunity to pursue my master's degree.

I would like to express my appreciation to Dr. Russell Butler for introducing me to the world of augmented reality. His support and patience were heartwarming. Without his mentorship and encouragement, none of this would have been possible. We have been searching through and testing multiple subjects in a short period to find out what best suits me to follow. I feel grateful for this achievement and believe it has created the way for my future success.

I also extend my gratitude to all the other professors in the Department of Computer Science at Bishop's University, particularly Dr. Stefan D. Bruda, Dr. Majid Alili and Dr. Mohammed Ayoub Aloui Mhamad, from whom I have gained valuable knowledge and insights.

Finally, I want to acknowledge and thank my wife and parents for their enduring love and unwavering support throughout this journey. They believed in me, and their encouragement played a crucial role in my success.

Contents

Table of Figures

Chapter 1

Introduction

1.1 Background

As technology keeps advancing rapidly, virtual reality (VR) is changing how we live in many ways. One exciting area where VR is making an impact is in art and design, especially with virtual sculpting.

In VR sculpting, artists get to combine their creativity with the power of VR technology, breaking the boundaries of traditional art and digital design. They can create and manipulate 3D forms in a virtual space, opening new possibilities in the world of art and beyond. However, making VR sculpting feel as realistic as physical sculpting is a big challenge. Traditional sculpting relies heavily on the artist's touch and feel of the materials, something that current VR systems struggle to replicate.

Although VR has improved a lot in recent years, most of the progress has been in visuals and sound. The sense of touch, known as haptic feedback, is still behind. Capturing and reproducing the subtle textures, resistance, and movement sensations of real-life sculpting in VR is complicated. The current solutions for VR sculpting involve using handheld controllers to track the artist's movements and provide some feedback. While they can follow the general motions, they lack the precision needed for detailed work. The feedback they provide also falls short in matching the experience of real sculpting, creating a disconnection between the artist and their virtual creation.

To address these challenges, this research aims to propose a new method to improve the realism and tactile responsiveness of VR sculpting by combining hand tracking with different mesh manipulation techniques. By doing so, it seeks to offer a more intuitive and immersive experience for artists, bridging the gap between physical and virtual sculpting.

1.2 Research Problem

The problem this research aims to solve is improving VR sculpting techniques to make them more precise and intuitive. The proposed method combines hand joint positions, spring-like motion simulation for object vertices, liquid-style objects, and Minecraft-like curving. These techniques have not been extensively studied in VR sculpting, so the research is necessary.

Integrating VR into artistic practices, especially sculpting, is exciting, but it comes with challenges. Current digital sculpting tools are mostly non-immersive, limiting artists' engagement with their work. The use of conventional peripherals like mice or tablets creates a disconnect and hampers the fluid expression seen in traditional sculpting.

VR sculpting tools typically use handheld controllers, but they can't capture detailed hand movements accurately. This limits the depth and realism of artistic expression. Exploring the

use of detailed hand joint data in VR sculpting could offer a more natural and precise interaction.

Accurately tracking and interpreting real-time hand movements in VR sculpting is a significant challenge, as it involves multiple joints with different ranges of motion. Handling these intricate manipulations without lag requires powerful computational resources and optimized algorithms.

The application of VR to live sculpting, especially with detailed hand joint data, is a new and unexplored area. There are no established methodologies or benchmarks, making it a pioneering research field that requires innovation and resilience.

To address these challenges, the research aims to develop an innovative method for live VR sculpting. It focuses on using detailed hand joint tracking for precise and realistic object manipulation. The goal is to improve hand detection accuracy, computational efficiency, and real-time responsiveness in the VR environment. By doing so, the research aims to enhance the intuitiveness, realism, and immersion of VR sculpting experiences, offering new possibilities for artists and designers.

1.3 Objectives

The primary objective of this thesis is to develop and evaluate methods for VR sculpting that have better precision, usability, and computational efficiency. To achieve this goal, the following objectives has been set:

i. Develop and test and test Solid elasticate style with the application of a spring-like motion simulation on vertices around the objects to mimic the natural movements in sculpting.
ii. Develop and test the curving style in Unity like the game Minecraft.
iii. Develop and test Liquid elasticate style and assume that our object is liquid and have sculpting with a liquid.
iv. Create and explore a particle system with managing memory system in Unity in terms of have a better memory management.

While we do these approaches we will test and examine the potential of hand joint positions obtained from the tracking package for object manipulation in VR. Also, we need to explore the application of these methods. So, we conducted a series of experiments to evaluate the effectiveness and performance of the proposed method.

1.4 Thesis Structure

This thesis is organized into several chapters, each focusing on a different aspect of the research. Below is a summary of the structure and content of each chapter.

i. **Literature Review**: In this chapter we present a comprehensive review of the existing literature related to VR sculpting. Some apps also have been reviewed in this chapter.

ii. **Materials and Methods**: In this chapter the materials, like the device and the methods that is used is discussed.
iii. **Approaches and results**: In this chapter, we discuss how we can implement the mentioned method and see the results of implementation.
iv. **Future Work**: At this chapter we conclude with a reflection on the research's significance and a discussion of potential future directions, including enhancements to the current system and new research opportunities that have emerged from the study.

Chapter 2

Literature Review

This section reviews some famous existing VR sculpting tools, discussing their features, capabilities, and limitations. These tools rely on traditional VR controllers, which fail to capture the intricate movements of human hands. Also, we review some research on these criteria.

2.1 Interactive 3D Sketching in Virtual Reality

The work by Yu (2020) focuses on addressing the challenges associated with 3D sketching in VR environments. Traditional design sketches are inherently 2D, requiring artists to project their 3D ideas onto a 2D surface for viewers to decode. To enhance the creation process, the study proposes an approach based on automatic stroke beautification and surface inference. By allowing users to sketch freely in 3D and providing occlusion cues, this method improves the accuracy and understanding of free-hand 3D sketching [1].

2.2 Immersive Sketching

Since the early days of immersive environments, researchers have explored the potential of using immersive displays and input for artistic creation. Works like Holo Sketch, Surface Drawing, and Free Drawer have demonstrated novel interfaces for 3D object creation using direct 3D position input and sketching in mid-air [2]. Although some of these interfaces are no longer in use, their ideas continue to inspire modern immersive sketching applications. VR sketching interfaces, however, present new challenges, such as imprecision in free-hand sketching and the lack of supporting surfaces. Various approaches, including reducing degrees of freedom and using haptic feedback, have been proposed to overcome these challenges [3].

The most advanced app released is Gravity Sketch (Figure 1). In their app they use controllers for manipulation. Using human hands for sculpting is not applicable in this app.

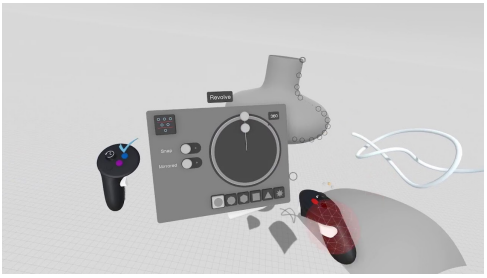


Figure 1: A view of the app Gravity sketch with using controller and lack of human hand in editing

2.3 Modeling Techniques

The field of VR modeling in arts and design has witnessed significant advancements over the past decade. Various modeling techniques have been developed to improve complex surface modeling, shape prototyping, and structural enhancement [4]. Free-form surface modeling (Figure 2), subdivision surfaces, mesh blending, relief modeling, volumetric sculpting, and multi-level voxel modeling are among the methods that enhance creativity and visual interaction [4] but still no human hands and not real-time in-game manipulation.

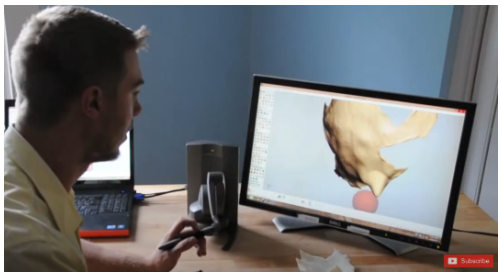


Figure 2: A simple view of Freeform a PC app and using a pen, not using human hand [1]

2.4 VR Modeling Systems

The available VR modeling systems can be divided into research-based and game-based categories. Research-based systems, such as those by Lin et al., POTEL, Wow Tao, and Pottery Go, aim to provide novice users with immersive pottery modeling experiences, using technologies like Leap Motion and VR kits [5]. None of them use real human hands.

In conclusion, this chapter presented a review of some related work on interactive 3D sketching in VR and VR modeling in arts and design. The examined research covers various techniques and systems that contribute to the understanding and advancement of 3D sketching, VR modeling, and creativity in virtual reality environments. We found that there is no usage of human hands in the sculpting apps and not enough ground research for that (Figure 3).

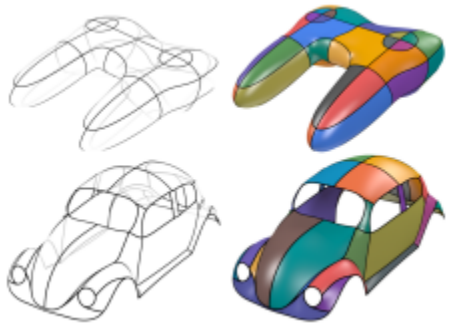


Figure 3: How most of the 3d sketching app works [2]

In the continuing of this section, we review some studies and works that will be needed for manipulation and deformations.

2.5 Mesh Deformation and Graphical Physics deformation background works

Most of the previous works in research have had a gap in being able to have a real-time manipulation. Their methods are mostly based on working grids and solving equations in a system of equations that need iteration to be solved and achieve the final value. Although there are some cases that can have a predefined answer but most of them are not real-time.

The deformation studies gained attention in 1999[16] where Terzopoulos mentioned elasticity for graphics mesh.

Since Terzopoulos and his team's seminal work [16], there has been a lot of excitement in the research community about simulating materials that can change shape, especially in computer graphics. This is super important when it is about mesh deformation, where the goal is often to accurately portray materials as they break, stretch, or bend.

Various techniques have come up to make mesh deformation more realistic, particularly when materials are breaking or being cut. Methods like local re-meshing [17], the Virtual Node Algorithm [18, 19], and XFEM [20] have been key players. While these methods offer solutions, they are not without challenges, especially in maintaining the quality and efficiency of the mesh and not being real-time.

Meshless techniques, championed by researchers like Pauly and his team [21], offer another promising direction. They might eliminate some of the complications tied to traditional mesh-based approaches. In this vein, the Material Point Method (MPM) [22] has gained traction. It's even been used in the special effects for movies like Disney's Frozen, which is an animation and pre-rendered. MPM combines particles and grids, and has proven versatile for a variety of simulations, including mesh deformation [23].

What is interesting about MPM is that it can handle extreme changes in material, like when something splits apart or fuses back together. This is crucial for complex mesh deformations

where traditional methods, like Smoothed Particle Hydrodynamics (SPH), sometimes fall short. MPM is not perfect; it struggles with simulating sharp cuts and rapidly changing speeds. Luckily, an improved version called MLS-MPM, discussed by Fang and his colleagues [24], looks promising in tackling these issues. The MLS-MPM method is remarkably close to reality in most of cases. The only problem is that it needs some time and calculation for rendering which is not particularly useful in real-time.

Chapter 3

Materials and Methods

This chapter presents the methodologies, tools, and software used to implement the four approaches explored in this thesis: spring-like motion simulation, Minecraft-style sculpting, liquid-style sculpting, and Entity Component System (ECS) memory management.

3.1. Unity and Oculus Virtual Reality System

The core technology utilized in this study is the Unity game engine, known for its flexibility and robustness in developing interactive experiences. Unity served as our development platform where we built and tested our VR sculpting application. The Oculus VR system [figure 4], with its high resolution and immersive experience, was employed as the primary interface for our VR sculpting application.

Figure 4: Oculus Device view

Renowned for its versatility in creating interactive 3D content, Unity offered us the ideal platform to develop our VR sculpting application due to its extensive VR support and compatibility with Oculus VR system. Unity's powerful rendering engine, physics simulator, and scripting API allowed us to explore and implement a range of sculpting interactions with varying levels of complexity. Additionally, the engine's robust profiling and debugging tools were instrumental in optimizing our implementations and tracking their performance metrics.

The Oculus VR system was chosen for its hand tracking capabilities, high-resolution display, and ergonomic design. Its integration with Unity provided us a streamlined process to create and test immersive VR experiences. The Oculus hand tracking system served as our primary input method, capturing the users' hand joint positions and movements with high accuracy. This data was then used to manipulate objects within our VR sculpting application, ensuring an intuitive and realistic sculpting experience.

3.1.1 Hand Tracking Package

In conjunction with Unity and Oculus, we utilized a robust hand tracking package that accurately tracks and records the positions of the hand joints. This package uses high-speed cameras and advanced computer vision algorithms to translate real-world hand movements into the virtual environment, enabling dynamic manipulation of virtual objects.

The Oculus Hand Tracking library provides developers with the capability to track hand and finger movements in real time, presenting a 3D skeletal representation of users' hands in VR. It recognizes various hand shapes and poses and is capable of accurately estimating the position and orientation of individual joints.

The Oculus Hand Tracking library operates through a sequence of stages:

i. **Image Acquisition:** The Oculus Quest's four wide-angle monochrome cameras capture images of the user's environment at a high frame rate. These images provide the raw data used to track hand and finger positions.
ii. **Pre-processing:** The captured images undergo a series of pre-processing steps, including noise reduction and normalization, to ensure they are optimal for the hand tracking algorithm.
iii. **Hand Detection and Localization**: The pre-processed images are fed into a deep learning model trained to detect and localize hands in the images. This model generates a 3D bounding box around each detected hand.
iv. **Hand Pose Estimation:** Once the hands are localized, another deep learning model estimates the pose of each hand, outputting a 3D skeletal representation of the hand. This skeletal model includes positions of various hand joints and the orientation of the fingers.
v. **Post-processing:** The raw skeletal data is further refined using various heuristic rules to smooth movements and ensure physical plausibility of the hand poses.

By leveraging the Oculus Hand Tracking library, this research was able to focus on novel contributions like the development of the real-time VR sculpting process and optimization of vertex manipulation.

3.2 An overview of methods

We have tested four styles of sculpting methods:

i. **Spring-like Motion Simulation:** Our first approach, implemented within Unity, involves a spring-like motion simulation for vertex manipulation around objects. Vertices of the object in the virtual space are manipulated based on the movement of the user's hands, recorded by the hand tracking package. This algorithm tries to replicate the tactile feel of sculpting in a physical environment within the virtual space.
ii. **Minecraft-style Sculpting:** The Minecraft-style sculpting approach, also developed within Unity, entails manipulating a large number of objects (10,000 in our tests). This

approach provided essential insights into the limits and capabilities of our system, despite being computationally intensive. This approach is by destroying objects by collusion.

iii. **Liquid-style Sculpting:** For the liquid-style sculpting approach, we employed a neural network model to emulate fluid dynamics within Unity. Each particle in this model carries information about its neighboring particles. By creating a state of zero speed, gravity, and defined boundaries, we were able to mimic sculpting with a liquid substance in a virtual environment.

iv. **Entity Component System (ECS) on physics:** The final approach experimented with Unity's Entity Component System (ECS) to apply physics and movements on entities. It's done with managing memory chunks efficiently. ECS offers high performance by grouping data into different chunks based on their components, facilitating efficient data access and manipulation.

Performance of our VR sculpting system was evaluated using Unity's built-in analytics tool, which allowed us to measure factors such as physics calculations, rendering performance, memory allocation, and time complexity. We also assessed user experience qualitatively, focusing on factors like ease of use, intuitiveness of interaction, and overall user satisfaction within the Oculus VR environment.

3.2.1 Deeper dive into liquid simulation MLS-MPM methods

There are multiple kinds of simulation approaches for liquid simulation or in better words, continuous physics simulations. MPM (Material Point Method) is a hybrid Lagrange discretization scheme for solid mechanics [5]. MLS (Moving Least Squares) is a method often used for curve fitting or surface reconstruction. It uses a weighted least squares approach to better approximate the values of unknowns in the domain. These methods assume an object containing a collection of particles and apply Newtonian laws on a fixed computational grid. the MLS-MPM method aims to achieve greater accuracy and efficiency in simulating complex material behavior. This approach is extremely useful in continuous physics simulations and consists of many particles. This method, because of its lightness, has been used in computer graphics, and animation of liquids, gas, and particle solids like sands. It can also be applied to clothes. However, most of its applications that we reviewed were not in real-time which is crucial for a VR application.

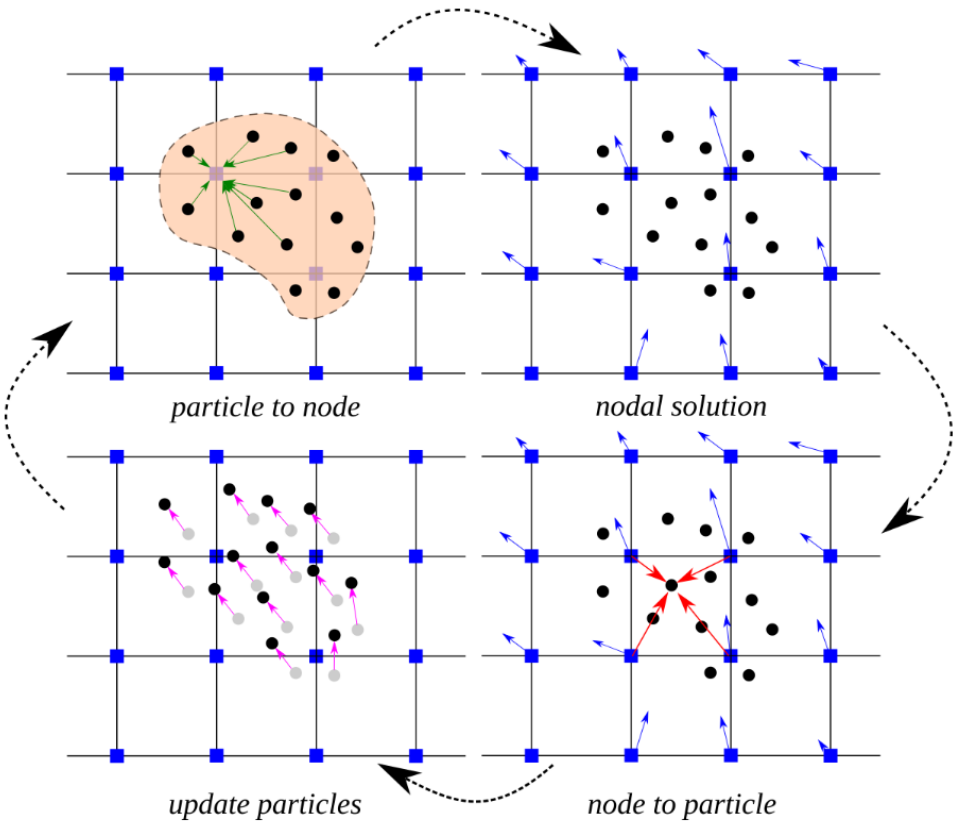


Figure 5: Illustration of the MPM algorithm: (1) a representation of material points overlaid on a computational grid. Arrows represent material point state vectors (mass, volume, velocity, etc.) being projected to the nodes of the computational mesh; (2) the equations of motion are solved on the nodes, resulting in updated nodal kinematics; (3) the updated nodal kinematics is interpolated back to the material points; (4) the state of the material points is updated, and the computational mesh is reset[15].

We found a package named Zibra Liquids which is a fluid simulation plugin developed by the company ZibraAI. The plugin is based on machine learning and is the first of its kind to be available for commercial use.

At the core they use Moving Least Squares Material Point Method (MLS-MPM) (figure 6). The governing equations are Eulerian equations [5] and applying Finite Element Methods and assumptions for simple boundaries. The governing equation is:

$$(D\rho/\,Dt) \;+\; \rho\nabla\cdot v \;=\; 0\;(conservation\;of\;mass) \tag{1}$$

$$\rho\,.\,(Dv/\,Dt) \;=\; \nabla\cdot\sigma \;+\; \rho g\;(conservation\;of\;momentum) \tag{2}$$

Where v is Velocity, ρ is mass density, g is gravity vector, and σ is shear stress. For any

$$\phi(\,(x,t):\;\; D\phi/Dt \;=\; \partial\phi/\partial t \;+\; v\cdot\nabla\phi \tag{3}$$

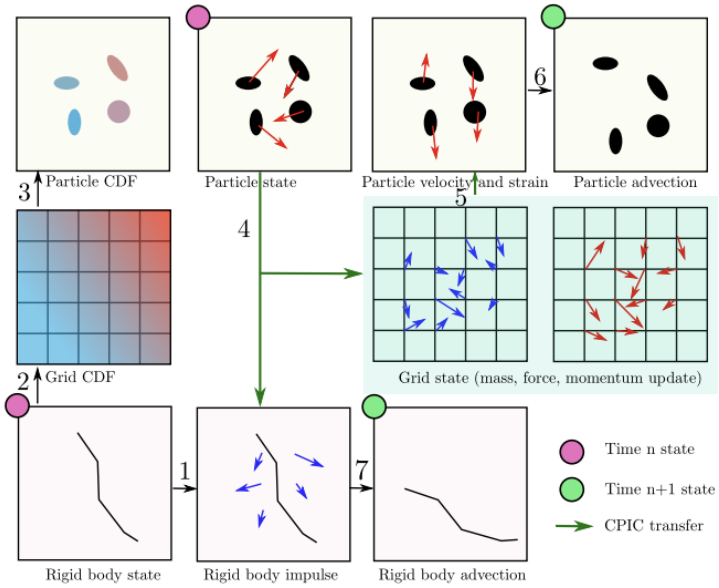


Figure 6. Algorithm overview from time to(n) to t(n+1), two-time steps, for MLS-MPM with CPIC (Convected Particle-In-Cell). Steps: (1) Rigid-rigid collision and rigid body articulation update rigid body velocities; (2) Splat rigid body to grid CDF(Cumulative Distribution Function); (3) Reconstruct particle CDF from grid CDF(; (4) CPIC particle-to-grid transfer and rigid body impulses; (5) CPIC grid-to-particle transfer; (6) MPM particle advection; (7) Rigid body advection. [5]

ZibraAI's core technology uses machine learning to encode the Signed Distance Field (SDF) of a 3D object into compact vectors that can then be unpacked by a neural network to restore the original SDF. The object is divided into a sparse voxel grid, and the local geometry inside each voxel is compressed into a vector using another neural network. This process does not require training new networks to represent an unseen object and takes milliseconds to compute.

Zibra Liquids applies this technology to fluid simulation in games. It allows developers to simulate the interaction of fluids with animated objects of any shape in real-time. This plugin is integrated directly into the game engine, which eliminates the need for expensive and time-consuming third-party software. It can be installed in a few minutes and enables new gameplay mechanics, such as water interaction.

At its core, the plugin uses a combination of proprietary physical solvers and machine learning-based neural object representations. The fluid simulation technology is based on the Moving Least Squares Material Point Method. The initial prototype was able to simulate 300,000 particles in 7 milliseconds on a mid-range laptop [7]. A custom rendering technique was developed for particle visualization, using a local voxel particle ray-trace method.

The plugin interfaces with the Unity engine, receiving a mesh from Unity and generating an SDF. It then processes the SDF using ZibraAI's machine learning technology, packing the data into a series of per-voxel neural embeds stored on the client's site. This allows users to set up

visual effects and improve visual quality.

The plugin is natively compiled on C++, which helps control resources, minimize CPU load, and optimize GPU shaders. Optimization techniques include resource packing, register usage optimization, memory access patterns optimization, and reducing CPU involvement using indirect draw/dispatch calls. Other optimizations include a custom radix sort implementation for the solver that is six times faster than the previous bitonic sort, using an atomic-based rasterizer instead of quads, and optimizing the particle to grid pass of the MPM algorithm. These optimizations boosted performance by about 3.5 times over the previous implementation.

3.2.2 Deeper Dive into ECS and DOTS (Data Oriented Technology Stack)

In Unity, DOTS (Data Oriented Technology Stack) and ECS (Entity Component System) are two powerful technologies that revolutionize game development by improving performance and scalability. Let us explore each of them in more detail:

DOTS is a high-performance programming model introduced by Unity to leverage the full potential of modern hardware, such as multicore CPUs and GPUs (Graphics processing units). It focuses on data-oriented design principles to maximize efficiency and minimize memory usage.

Traditionally, game development revolves around object-oriented programming (OOP), where game entities are represented as classes with data and behavior encapsulated within them. While object-oriented programming provides flexibility, it can suffer from performance issues when handling massive amounts of data.

DOTS introduces a paradigm shift by emphasizing data layout and processing. It encourages developers to organize data in contiguous memory and perform operations in bulk, rather than dealing with individual objects. This approach enhances cache coherence and allows for efficient parallel processing, resulting in significant performance gains.

DOTS comprises several key technologies, including:

i. Entity Component System (ECS): A core part of DOTS, ECS is an architectural pattern that separates entities (game objects) into three distinct components: entities, components, and systems.
ii. Burst Compiler: A high-performance just-in-time (JIT) compiler that optimizes C# code for maximum execution speed. It produces highly optimized machine code, taking full advantage of the target hardware.
iii. Unity. Mathematics: A lightweight math library tailored for performance-critical operations, such as vector and matrix computations. It offers efficient implementations of common mathematical functions and data types.
iv. Unity. Physics: A physics engine built with DOTS in mind, allowing for efficient and scalable physics simulations.

By adopting DOTS, we can achieve faster frame rates, better scalability, and improved

utilization of modern hardware.

ECS is a programming paradigm associated with DOTS. It offers a data-driven approach to game development by breaking down game entities into their fundamental components and systems. In ECS, entities represent the objects in a game, while components are the individual data elements associated with each entity. Systems define the behavior and logic that operate on these components. This separation allows for better code organization, reusability, and performance optimization.

i. **Entity Entities**: An entity is a unique identifier representing a game object. It does not contain any data but acts as a container for components.
ii. **Components**: Components define the data associated with an entity. For example, a Transform component might contain position, rotation, and scale data. Each component is typically a simple data structure (type) without any behavior.
iii. **Systems**: Systems are responsible for processing components and implementing game logic. They iterate over the relevant components and perform operations on them. Systems operate in parallel, taking advantage of multiple CPU cores.Component System (ECS): A core part of DOTS, ECS is an architectural pattern that separates entities (game objects) into three distinct components: entities, components, and systems.
iv. **Burst Compiler**: A high-performance just-in-time (JIT) compiler that optimizes C# code for maximum execution speed. It produces highly optimized machine code, taking full advantage of the target hardware.

ECS provides several benefits, such as:

i. **Performance:** By using a data-oriented approach, ECS allows for efficient memory access and parallel processing, resulting in improved performance and scalability.
ii. **Modularity:** Components and systems can be easily reused across different entities and game objects, promoting code modularity and reducing redundancy.
iii. **Cache coherence:** ECS optimized memory access patterns, improving cache utilization and reducing the number of cache misses, leading to faster execution.
iv. **Scalability:** The data-driven nature of ECS makes it well-suited for handling large numbers of entities and components, making it easier to scale our game.

3.2.3 Memory management ECS and DOTS

In Unity, memory management and speed differ significantly between ECS (Entity Component System) with DOTS (Data-Oriented Technology Stack) and the traditional Mono Behaviour approach. Let's explore the differences in memory management and speed between these two paradigms. ECS with DOTS memory management consists of:

i. **Data-Oriented:** ECS focuses on organizing data in contiguous memory blocks, improving cache utilization and reducing memory fragmentation. It ensures data locality, resulting in faster memory access and reduced overhead.
ii. **Chunk-based Storage:** ECS stores components in chunks, where each chunk contains a

fixed number of entities which have the same type of component. If an entity gets another type of component, it will be moved to another chunk. This storage layout optimizes memory access patterns and enables efficient parallel processing.

iii. **Automatic Memory Allocation:** ECS automatically manages memory allocation and deallocation for entities and components, reducing the risk of memory leaks and manual memory management errors.

Mono Behaviour specifications memory management consists of:

i. **Object-Oriented:** Mono Behaviour objects are managed individually and have a more scattered memory layout. This can lead to slower memory access due to poor cache utilization. Especially when dealing with many objects or complex computations, CPU should go across memory, and it takes more time than the other.

ii. **Manual Memory Management:** As Mono Behaviour objects are managed manually, developers need to handle memory allocation and deallocation themselves, increasing the risk of memory leaks and performance issues if not handled properly.

When it comes to speed, ECS with DOTS offers significant performance advantages over the traditional Mono Behaviour approach.

ECS with DOTS total process speed specification:

i. **Parallel Processing:** ECS inherently supports parallel processing by processing entities and components in bulk. It leverages modern hardware, such as multi-core CPUs, to achieve high-performance execution across multiple threads.

ii. **Burst Compiler:** DOTS includes the Burst Compiler, which optimizes C# code for maximum execution speed. It generates highly optimized machine code and takes advantage of the target hardware, resulting in faster execution.

iii. **Reduced Overhead:** The data-oriented approach of ECS reduces overhead by eliminating unnecessary abstractions and focusing on efficient data processing. It minimizes function call overhead and improves cache coherence, leading to faster execution.

On the other hand for Mono Behaviour processing speed we should assume that it is:

i. **Single-Threaded:** Mono Behaviour objects operate on the main Unity thread, limiting parallel processing capabilities. This can lead to lower performance, especially when dealing with many objects or complex computations.

ii. **Higher Overhead:** Mono Behaviour objects carry additional overhead due to the object-oriented approach, method calls, and Unity's internal management of Mono Behaviour components. This can result in slower execution speeds compared to ECS with DOTS.

In summary, each approach offers unique strengths and potential areas for further exploration. In the next chapter, we will present and discuss the results of our experiments with these approaches, including assessments of precision, usability, and computational efficiency. This analysis will help to establish a clearer understanding of how these methods could advance the state of the art in VR sculpting.

Chapter 4

Approaches and Results

In this chapter, we discuss the four different approaches to dynamic object manipulation in the VR sculpting context, developed and tested in this study. Each approach comes with its unique characteristics and strengths, and understanding these can guide future research and development in the field.

4.1 Spring-like Motion Simulation

The spring force defines how a spring behaves when subjected to tension or compression. An ideal spring follows Hooke's Law, named after the 17th-century British physicist Robert Hooke. It states that the force exerted by a spring is directly proportional to the distance it is stretched or compressed from its neutral or "rest" position. Mathematically, this relationship is expressed as:

$$F = -k * x \tag{4}$$

In this equation, F is the force exerted by the spring, k is the spring constant, which quantifies the stiffness of the spring, and x is the displacement of the spring from its equilibrium position. The negative sign denotes that the spring force always acts in a direction to restore the spring to its equilibrium position. This property of springs—resisting displacement from equilibrium and exerting a force to restore the equilibrium—is the critical aspect we leverage in our VR sculpting application.

Consider a single vertex on the 3D model. When not being interacted with, this vertex is in its equilibrium state. As the sculptor's tool approaches and applies a force, we conceptualize this interaction as the displacement of a spring. This displacement is proportional to the force applied by the user and is resisted by a restoring force governed by the "stiffness" of the spring (the spring constant k).

In practical terms, this makes the 3D object feel pliable, as though it is composed of many individual springs responding to the sculptor's touch. Different materials can be simulated by changing the virtual "stiffness" of the springs, thus altering how the vertices respond to the user's input.

4.1.1 Expanding on the Benefits and Challenges of Using Spring Forces

Implementing spring forces in VR sculpting carries several benefits. The most significant is the increased realism and interactivity it provides. By closely mirroring the physical laws that govern real-world object manipulation, spring forces allow users to sculpt with intuition and

experience the tactile feedback they would expect from manipulating physical materials.

Determining appropriate spring constants to simulate different materials is a process of trial and error, requiring testing and user feedback. These values must strike a balance between realism, usability, and system stability. Additionally, the damping forces, which simulate resistance to changes in motion, add another layer of complexity and must be carefully managed to prevent oscillations or sluggish behavior.

Another challenge lies in performance optimization. With potentially millions of vertices (springs) interacting with the user's input, the computational demands are high, necessitating optimization strategies like spatial partitioning with octrees to maintain real-time responsiveness without compromising the quality of the user's experience.

To further enhance the capabilities of our VR sculpting application and offer more precise control to the users, we incorporated a slicing feature. Slicing is the process of intersecting a 3D object with a plane, dividing the object into two or more parts and revealing a cross-section. In the context of our VR sculpting application, slicing allows users to isolate and view portions of their sculpture, enabling them to make detailed edits that would be difficult or impossible from the exterior.

Implementing a slicing feature in a VR environment involved several steps:

i. Defining the Cutting Plane: The slicing process begins by defining a cutting plane. In our VR application, the user can position and orient this plane within the 3D space using their VR controllers.
ii. Computing the Intersection: Once the cutting plane has been defined, the system calculates the intersection between the plane and the 3D model. This process involves identifying which vertices of the model lie on either side of the plane and then determining where the model's edges cross the plane.
iii. Creating the Slice: After identifying these crossing points, the system constructs new edges and vertices to form the cross-sectional slice. This is performed by creating new polygons that follow the path of the cutting plane through the 3D model.
iv. Filling the Gap: Once the slice is complete, the exposed cross-section of the model is typically left open. To provide a solid appearance, the system generates a fill surface that covers the open section, giving the impression of a solid object that has been cleanly cut.

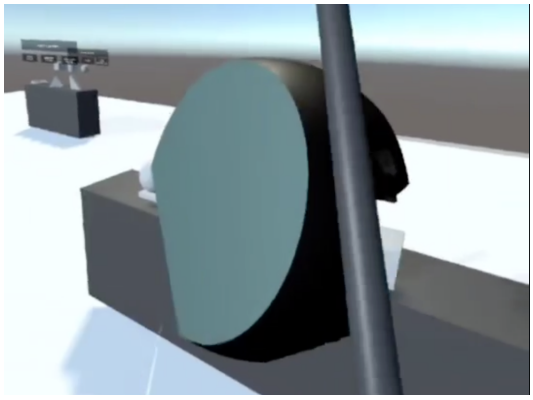


Figure 7: Slicing an object in the build app.

4.1.2 Vertex Manipulation Spring Force Methodology and Approach

In the context of Unity, rendering objects requires the use of meshes. A mesh represents a complex 3D object by defining a collection of vertices, which are points in 3D space, and a set of triangles that connect these vertices, forming the surface of the object. Here we can see how a surface is created by vertices and triangles (figure 8 and 9).

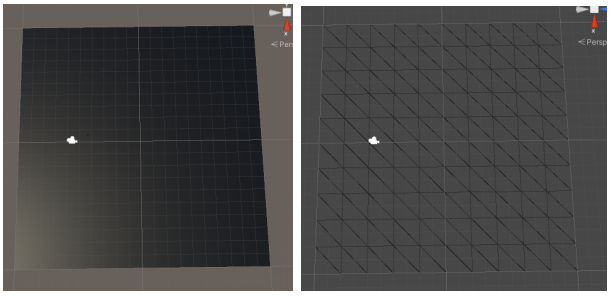


Figure 8: Mesh of a plane.

To illustrate flat and straight shapes, such as the faces of a cube, triangles work perfectly due to their flat and straight edges. But for curved or round surfaces, many small triangles must be used to approximate the shape accurately. If the triangles are small enough, the approximation is not noticeable, but this level of detail can be computationally intensive for real-time applications, resulting in some degree of jaggedness in the rendered surface.

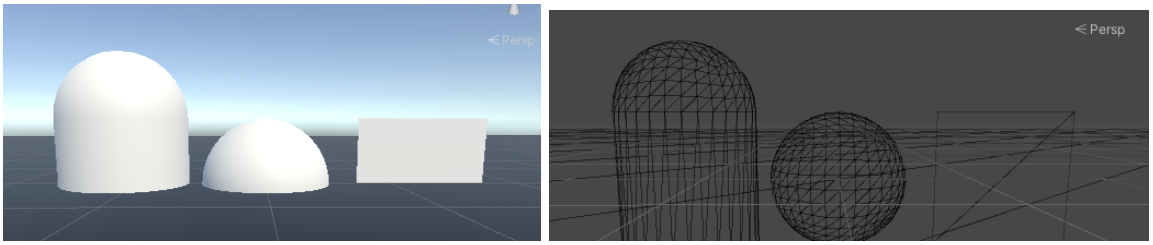


Figure 9: Curved mesh triangles.

Unity employs a mesh filter and a mesh renderer to display 3D models on game objects. The mesh filter holds a reference to the mesh to be displayed, while the mesh renderer configures how the mesh is rendered, such as which material to use and whether it should cast shadows or receive them. We created an object to test manipulations on a controlled surface, figure 10.

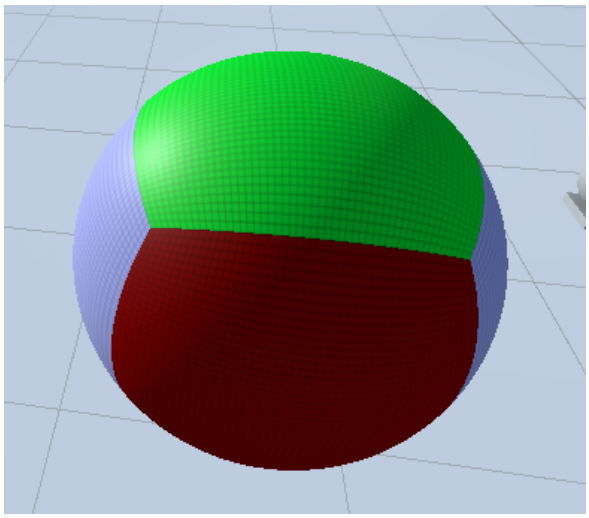


Figure 10: Geometry created for vertex manipulation use

The goal is to apply forces to the mesh and make it bounce back to its original shape, even when the object is transformed (moved, rotated, or scaled).

Our mesh deformation with spring force methodology is as follows:

i. **Create a grid of vertices**
ii. **Define Normals**
iii. **Create mesh deformer script:** that stores the original and displaced vertices and vertex velocities. It updates vertices based on forces applied on a point. It applies the force to the nearest point. This is done by calculating vertex velocities based on applied force and the distance. Meanwhile a damping is applied to stop at the proper place and show a natural movement. All of the vertices and their speed is saved as an array, and we iterate through it.

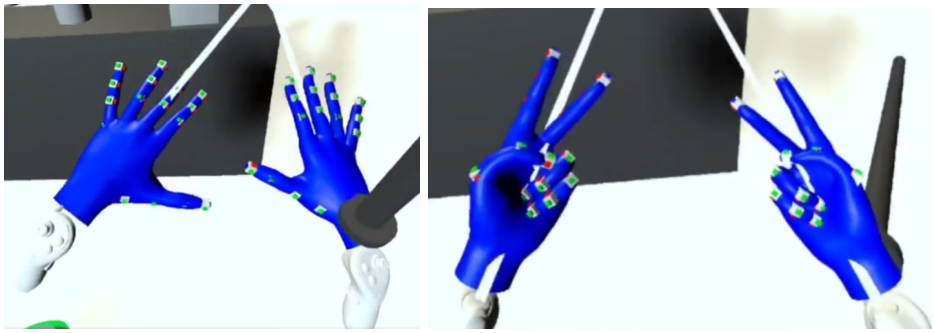
iv. **Controlle the force and damping**: We are controlling the spring force and damping with variables in the "Mesh Deformer" script. These parameters affect the responsiveness and stability of the deformation.

v. **Store and update**: Store and update scale of the object in the "Mesh Deformer" script. Scaling affects the deformation, so it needs to be compensated for.

vi. **Find the point to apply force**: A ray from hand joints is cast and after hitting the surface of the cube, we transform the force point from world space to local space in a method of the "Mesh Deformer" script. For better visualization we also add gizmos to see where it is touched.

On the deformer mesh algorithm, we first initialize arrays of original vertices, displaced vertices, and vertex velocities. Then while deforming is applied on each frame, we iterate for all of the vertices in displaced vertices. After that we get velocity of vertex from vertex velocities array, and we calculate displacement from original position. Then scale displacement based on the object's uniform scale and apply the spring force to velocity. Then we apply damping to velocity and finally we update vertex position by adding velocity and displacement calculated. It is important that force is applied only one time through each vertex. After that everything is automatic based on the spring force. and in each frame the speed of vertices is calculated. For all of the vertices in the displaced vertices array, we calculate the distance from the hit point and if it was in the criteria that we set before the attenuated force is applied. Then the velocity is calculated, and it is interpolated to the new velocity in each frame. Then with this in mind we have the new vertex position.

4.1.3 Integration to VR device



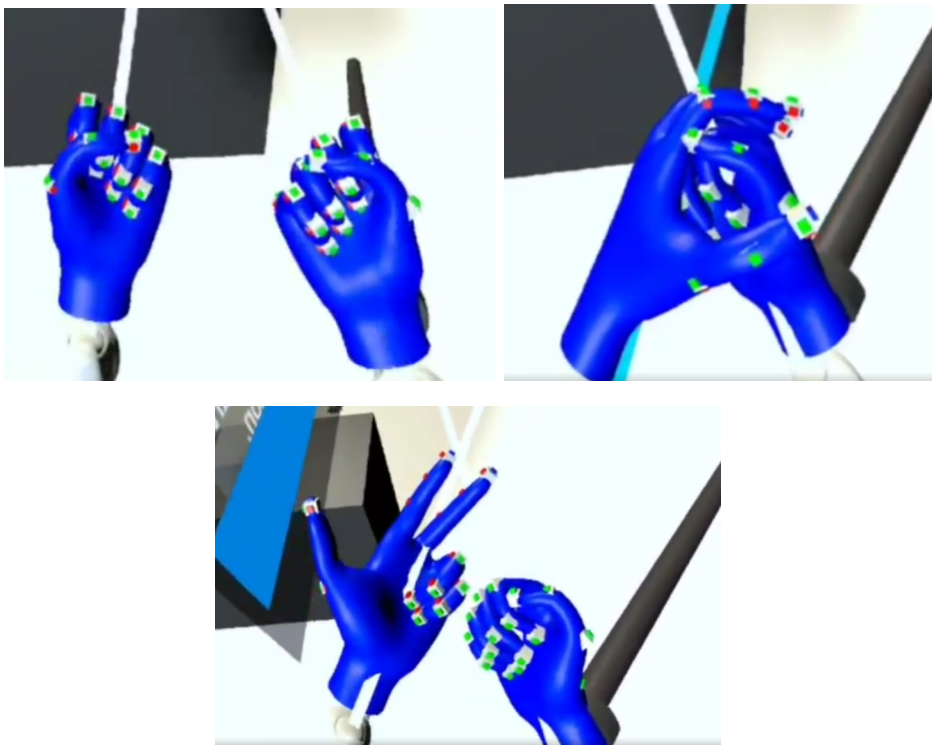


Figure 11: Hand with deformation cube on the joints.

For integrating to a VR device, we create a "Mesh Deformer" script that sends a sphere ray cast out from its object toward a direction which is the inside direction of the hand. If the ray cast hits an object with a "Mesh Deformer" script, it will apply the force to the hit point with the "Add Force" function in the hit object such that at that object a vertex movement starts.
Applying a point of force we perform Sphere Cast from ray Origin in the direction of the hand sole using sphere Radius as radius. The store resulted in a hit. check Sphere Casting if it hits mesh with Mesh Deformer component and if yes, calculate force application point by adding hit normal" multiplied by force Offset to hit point. Then we call 'Add Deforming Force Function' of Mesh Deformer component with the calculated point and force and Set hit Point.

We added a Deformer script in the Hand Visualizer code of the Oculus Device where joints are instantiated. By following these steps and integrating the hand tracking system with the Mesh Deformer Input and Mesh Deformer components, we can now interactively sculpt the 3D mesh using our hands. The cubes attached to hand joints act as sculpting tools, allowing us to deform the mesh in real time. By adding the Mesh Deformer to any game object, we can manipulate its vertices. We should note this approach depends on the number of vertices of the mesh, making it slow. In future work, an octree could help to improve performance.

4.1.4 Result, Approach, and improvements of the method

Here we show the results of our first approach, mesh deformation by spring force. The results depend heavily on parameter selection. The first problem arises when trying to apply multiple forces in a small amount of time. While there is a calculation running another would interrupt. We fixed this using Unity's coroutines to achieve concurrent execution of multiple spring force computations.

Our starting parameters were spring Force = 2, damping = 2, radius = 0.1, radius deform = 0.1, exponential factor for vertex manipulation = 2.0, the uniform scale = 1, and Force = 10 (figure 12).

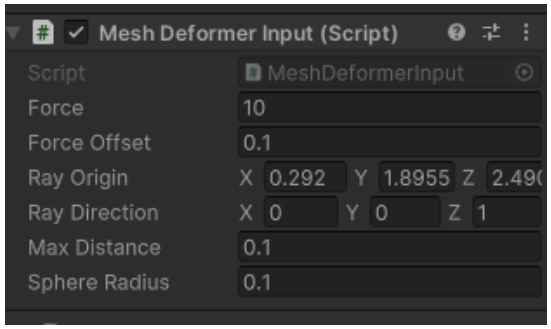


Figure 12: Unity editor mesh deformer starting point inspector view

In this modified version, the *falloff* variable is calculated based on the normalized distance from the deformation point. The normalized distance is obtained by dividing the actual distance by the deformation radius. The falloff value represents the force strength at each vertex and is determined by the inverse square of the normalized distance. This results in a smooth falloff of force from the center towards the edges of the deformation area (figure13).

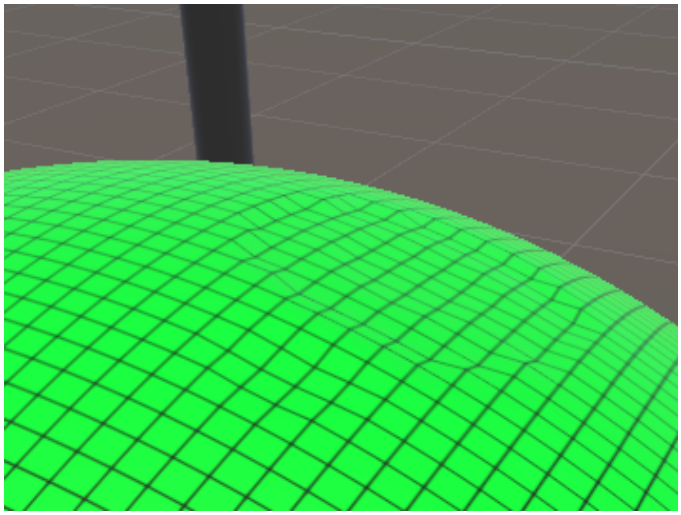


Figure 13: Smooth factor 5 by using a lerp function.

Changing the smoothness factor to .5 changed nothing. With smoothness factor 6:

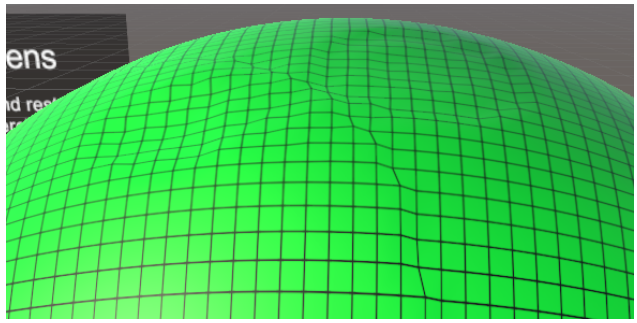


Figure 14: Changing parameters

We applied a falloff value. In this version, the falloff variable is calculated as:

falloff = 1 - (distance / radius Deform) (5)

Subtracting this value from 1 provides a smooth falloff where the force is maximum at the center (distance = 0) and gradually decreases towards the edges of the deformation area (distance = radius Deform). We can adjust the radius Deform value to control the size of the deformation area and the rate of force falloff. Smaller values will result in a smaller deformation area with a steeper force falloff, while larger values will increase the size of the deformation area with a more gradual force falloff.

With radius of 0.1 we have in figure 15:

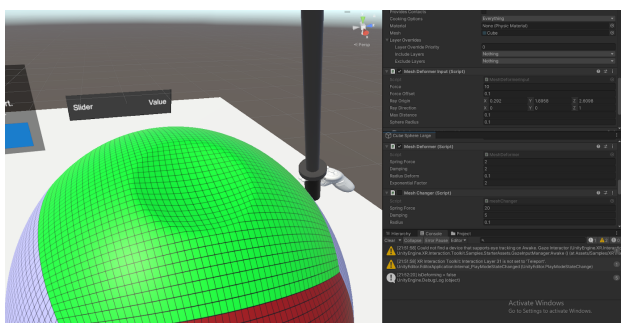


Figure 15: Changing parameters r = .1 and picture of the editor section

By changing to radius of 0.3 in figure 16:

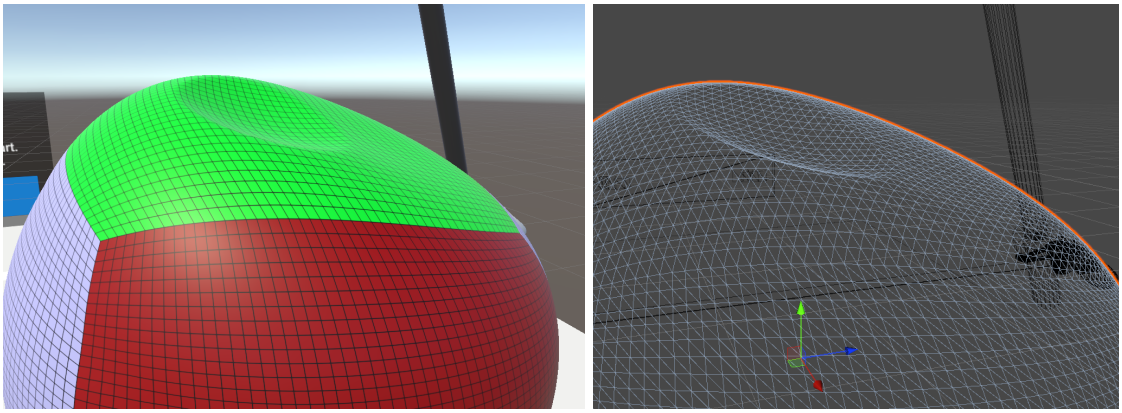


Figure 16: Changing parameters r =.3

After a one-time push we can see in figure 17:

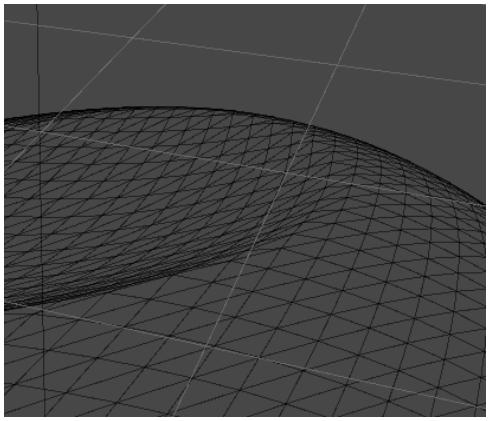


Figure 17: One time push

After 3-time push we can see how vertex moved in figure 18:

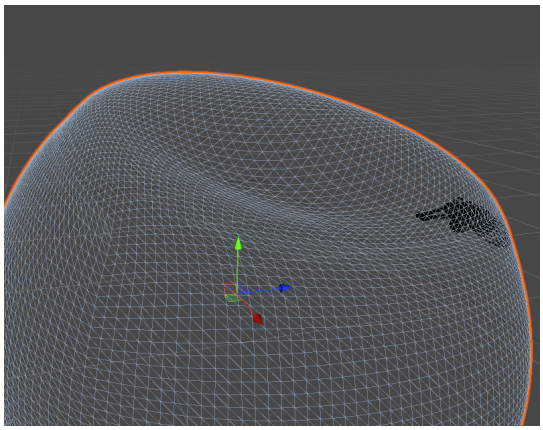


Figure 18: Three-time push

This shows how the deformation depends on the current number of vertices.

One of the major problems with this method is in applying all the fingers. When the number of fingers goes above two fingers some performance issues arise. In each frame, the array of data changes based on a force. Meanwhile, if we have multiple fingers, we all will try to change the velocity array. We tried to handle it by having changes in a queue, but it got an issue in term of calculation speed.

We experimented with some different constants:

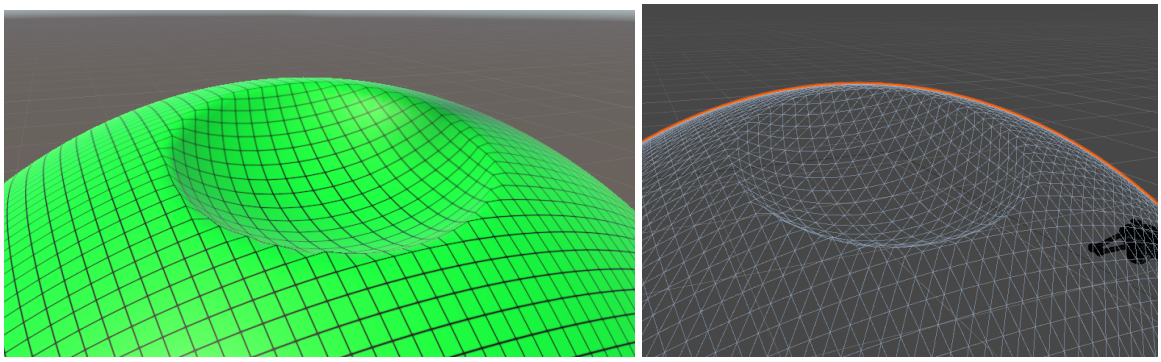


Figure 19: Radius .1, changing force 10 - 30

The tips are not well deformed (figure 19-20), it has a cut rather than a sculpted appearance.

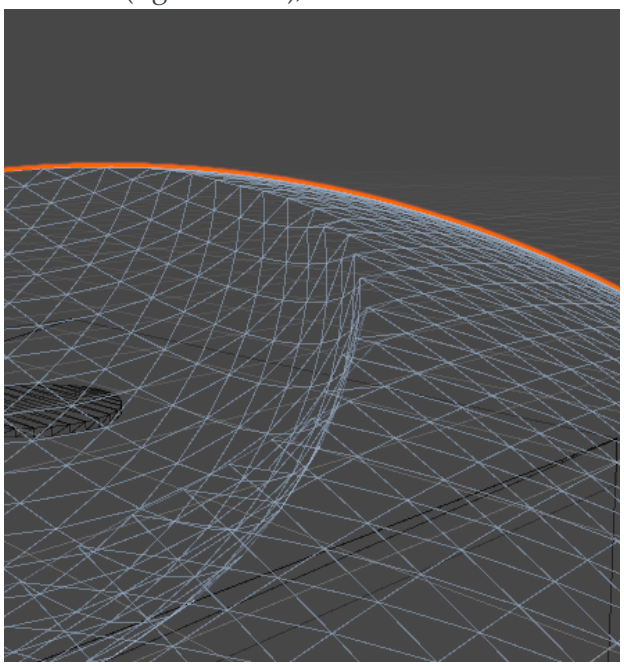


Figure 20: Zoomed sphere radius .1

Changing radius deform (figure 20) to .05, force 30, and damping force to 20:

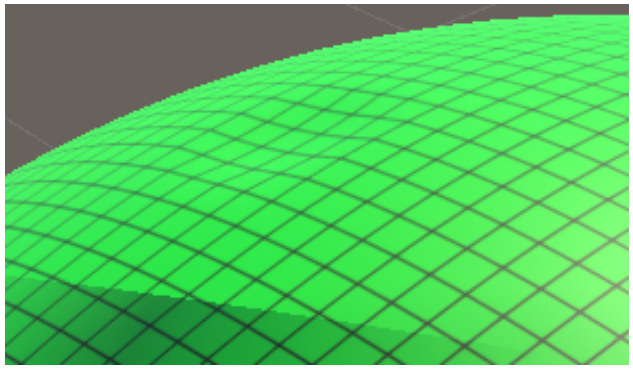


Figure 21: Change parameter

Then we Adjust smoothing function. The Gaussian smoothing function falls off quickly. After applying smoothing step on the edges, we can see the results in figure 22:

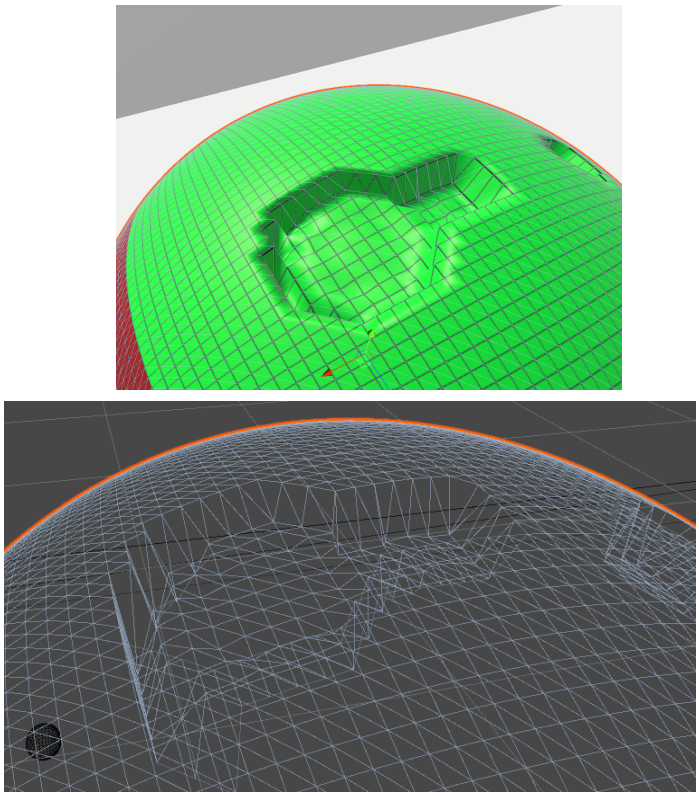


Figure 22: Smoothed with function

The appearance is still a bit rough. We added a smoothing step with cosine:

$$Smoothstep = 0.5f \times (1f - cos(radiusDeform/distance \times \pi)) \tag{6}$$

After changing some variables in figure 23 and 24 we see the results:

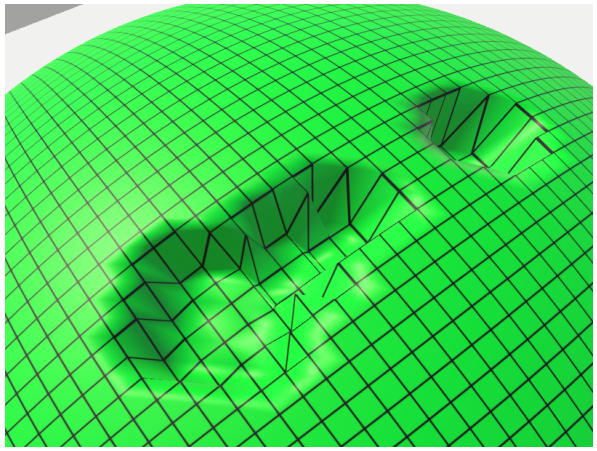


Figure 23: Results fully smoothed

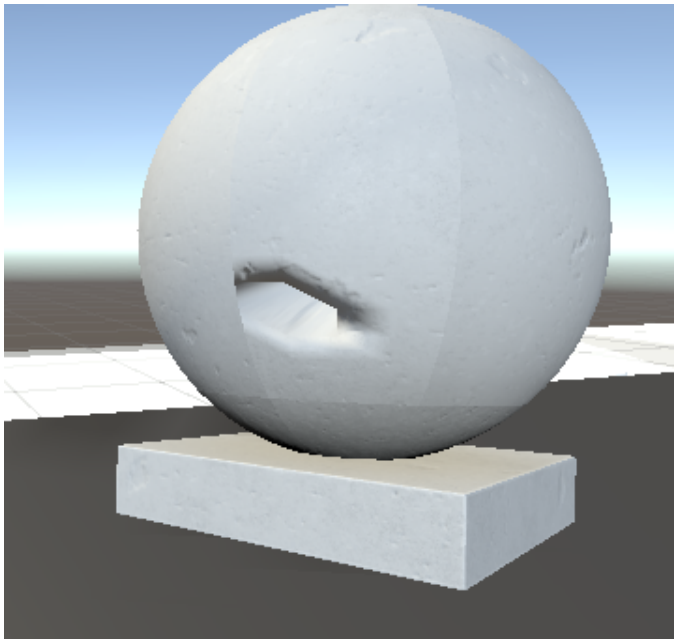


Figure 24: After adjusting parallel manipulation.

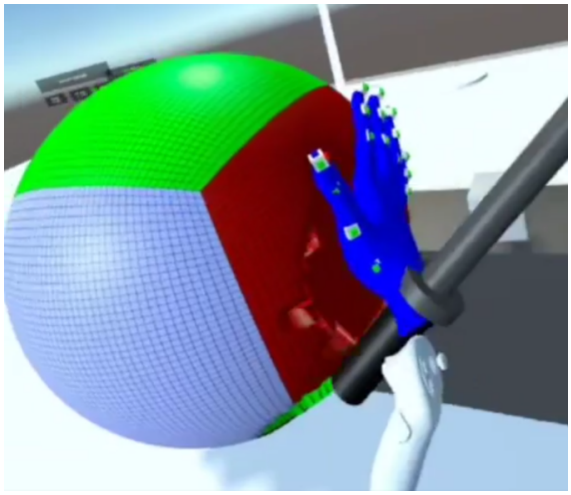


Figure 25: Applying hand on the surface.

The major issue is vertex density. It plays a key role. If the mesh does not have enough vertices, it may not deform smoothly no matter what deformation function we use. On the other hand, if it has a dense mesh, it will take too long to deform. In Figure 25 we see a hand applied force on a sphere.

4.2 Minecraft-style Sculpting

Our second approach was inspired by the popular Minecraft game (figure 27), on a cube created with 8,000 micro cubes (figure 26) and involving the manipulation of numerous virtual clay objects — up to 10,000 in our experiments. While this approach allows for a high degree of control over the sculpting process, it proved to be computationally intensive and inefficient. The heavy computational requirement may limit the scalability and real-time responsiveness of this method, making it less ideal for practical applications. However, certain techniques derived from this approach may be useful in more specialized contexts.

Result of Minecraft style:

The processing of 10,000 cubes was computationally intense. Rendering and physics was handled very slowly by Unity, limiting the practicality of this approach, especially when 10,000+ cubes are needed to form a smooth surface.

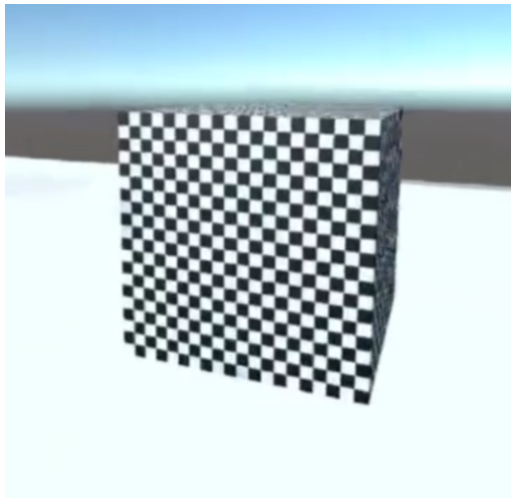


Figure 26: Box for method3.

We placed a sphere on the tip of each finger (figure 28), and deleted cubes when an intersection was detected with the sphere.

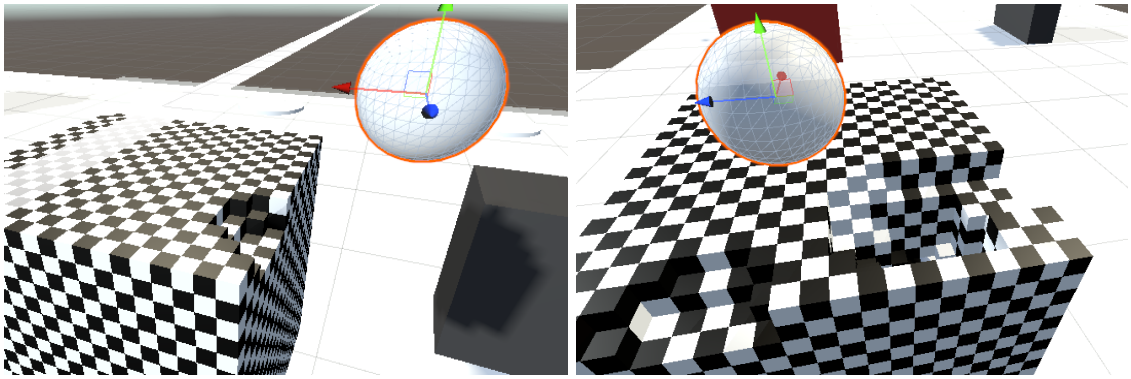


Figure 27: Curving Minecraft style.

This method had a heavy computational cost for 10,000 separate cube game objects but might be feasible if we were to use an octree in conjunction with one game object containing a large mesh representing all the cubes (instead of 10,000 separate game objects).

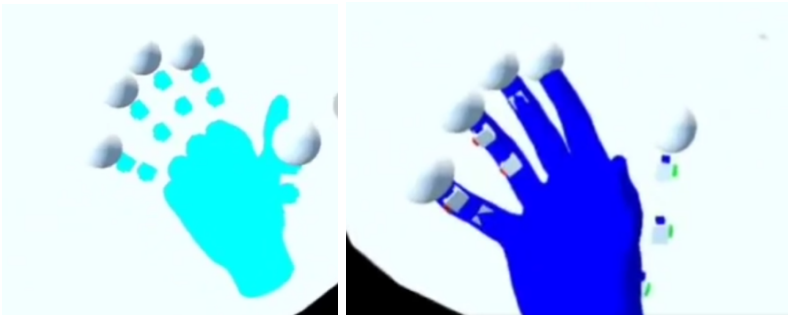


Figure 28: Adding Sphere for carving at fingertips.

4.3 Liquid-style Sculpting

The third approach we explored was liquid-style sculpting, based on the principles of fluid dynamics. This method utilizes a lightweight neural network where each particle carries information about its nearest neighboring particles. With zero speed, gravity, and specific boundaries applied, this method showed encouraging results, achieving a realistic and responsive simulation of liquid behavior. The liquid-style sculpting offers unique possibilities for artistic expression in the VR environment, further expanding the horizons of digital art and design.

Because we needed to have a sculpting style, we assumed just the displacement of points of the fluid (the sculpting material) with gravity equal to zero. Utilizing a ‘mud’ style was suitable for sculpting.

4.3.1 Signed Distance Field

At its core, a Signed Distance Field is a way to represent a shape or an object in a 3D space, but it can also be used for 2D shapes. Essentially, it's a grid of values where each value (or 'cell') holds the shortest distance from that point to the surface of the shape it represents. If the point is inside the shape, the value is negative, while if it's outside, the value is positive. Hence, the term "signed" - the sign of the value determines whether the point is inside or outside the shape.

One of the main advantages of using SDFs is that they allow for efficient calculations. For instance, when dealing with physics simulations, collision detection becomes simpler because we can quickly calculate whether a point is inside or outside an object, and how far it is from the object's surface.

Additionally, SDFs are particularly useful in graphics and game development. They allow for smooth rendering of shapes, even when close-up, as they offer a continuous gradient to the shape's edge. This is why they are often used in ray marching, a technique frequently employed in 3D computer graphics.

Moreover, operations like blending and morphing between different shapes become straightforward with SDFs. We can easily combine the distance fields of two or more objects to create complex shapes and animations.

In the context of VR sculpting, SDFs offer a more efficient way to handle object manipulations, since they allow for real-time calculations of object deformation, collision detection, and other physical interactions. However, like any other technique, SDFs also come with their own set of challenges, such as complexity in managing large distance fields and difficulties in texturing.

We used SDF for our liquid manipulation. Its boundary is assumed as a solid boundary in the Zibra Package. In each SDF we keep the liquid still. Liquids velocity has been set to zero and

also its gravity to be at the position in the liquid. Results were promising (figure 29 to 33).

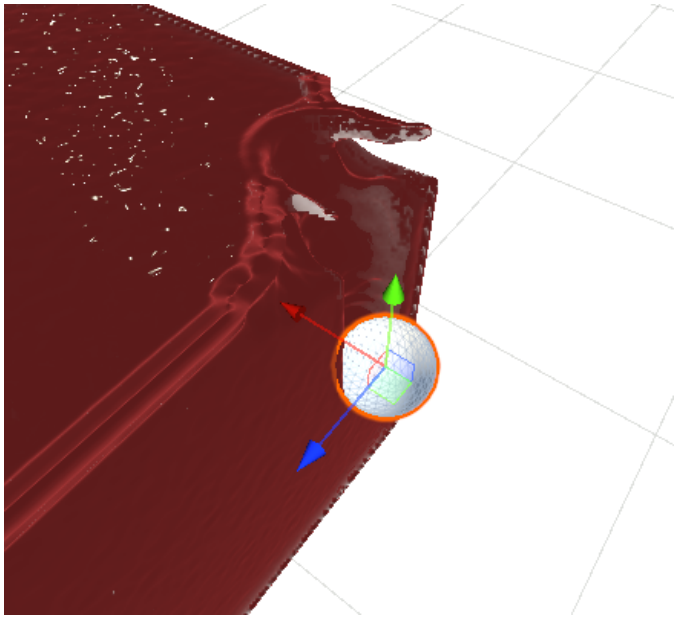


Figure 29: Negative Signed SDF liquid with gravity and speed zero - applying and moving an SDF collider triggered rigid body.

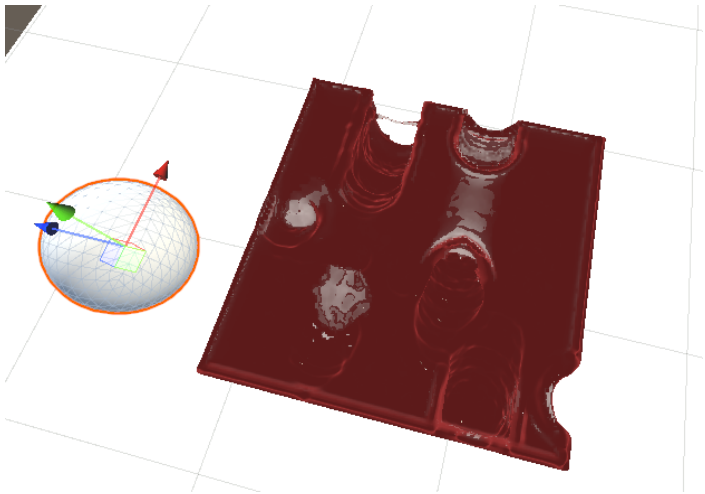


Figure 30: Change rendering calculations to be like metal-clay style - roughness, refraction color of brown, and scattering of the liquid has been changed.

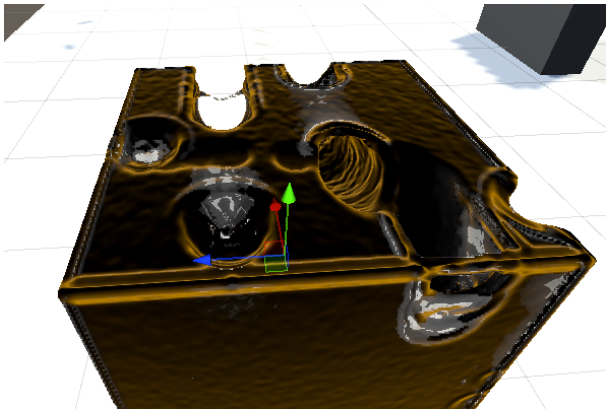


Figure 31: Change rendering calculations to be like metal-clay style - roughness, refraction color of brown, and scattering of the liquid has been changed.

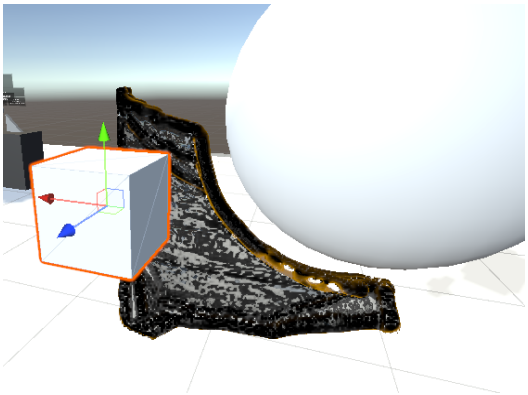


Figure 32: Using multiple styles of colliders.

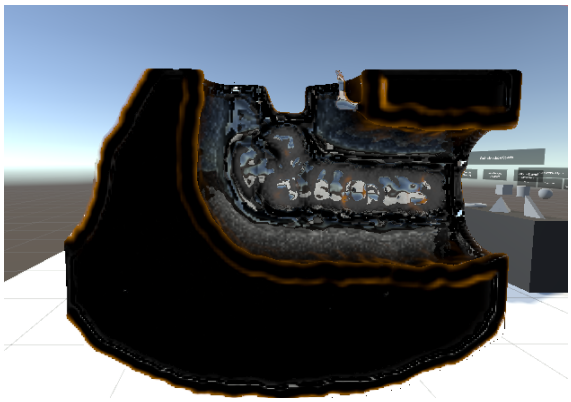


Figure 33: For usage in medical formations, a blood vessel.

4.4 Using ECS for Memory Management, Carving style

Finally, the fourth approach we investigated was the Entity Component System (ECS) for efficient memory chunk management. ECS shows promise as a means of optimizing memory usage and improving the overall performance of the VR sculpting system, particularly in scenarios involving complex models or high levels of user interaction. While still in its preliminary stages, our experiments indicate potential for the application of ECS in the first and second approaches, which could help overcome some of the limitations identified in these methods.

In this part, we tested 5,000 and 10,000 objects under a force field. On each of the particles a force is applied - figures 34 to 40. As we can see there is no lag or memory over 1ms allocation. This is because of how ECS manages memory chunks.

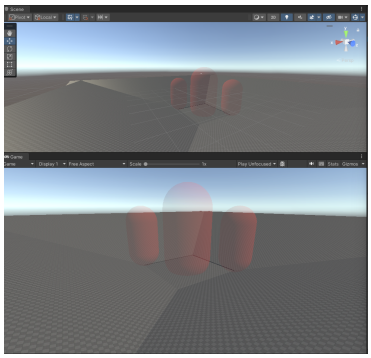


Figure 34: ECS scene before starting application.

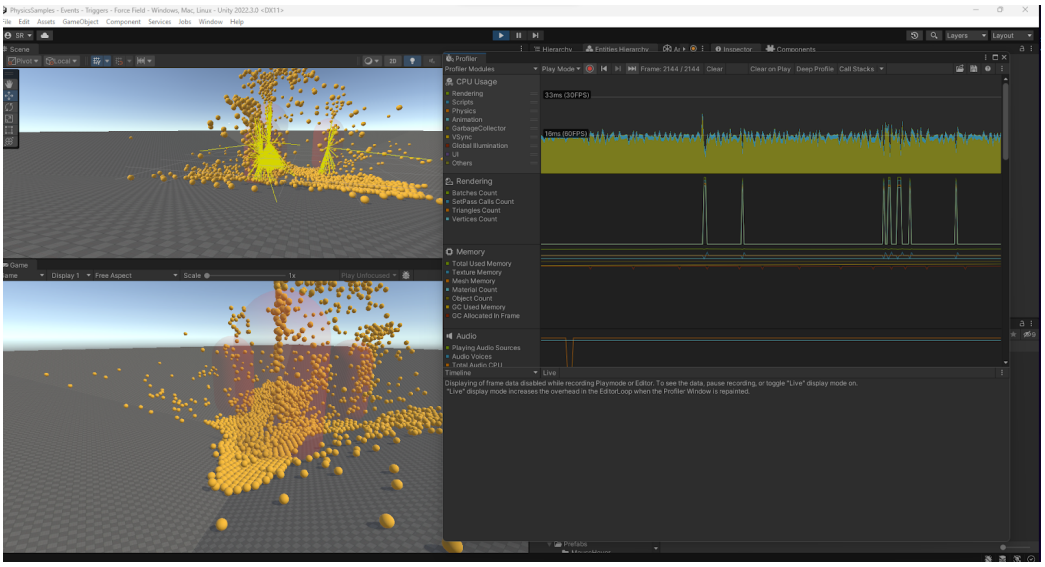


Figure 35: Memory usage of ECS application on 5000 objects with a force field. 30fps under 16ms.

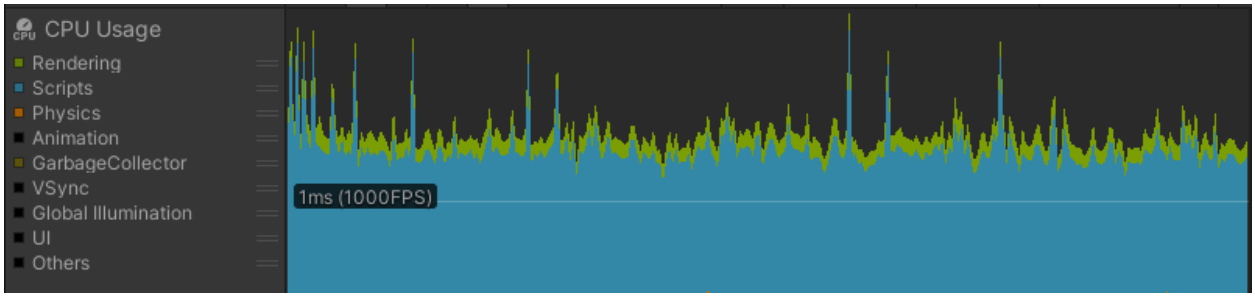


Figure 36: Rendering and scripts for 5000 objects under 1 mms 1000FPS

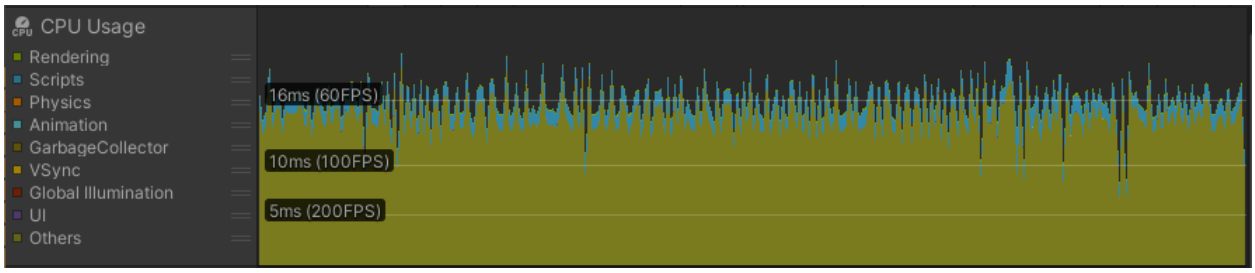


Figure 37: Memory usage with 60 FPS running under 16ms.

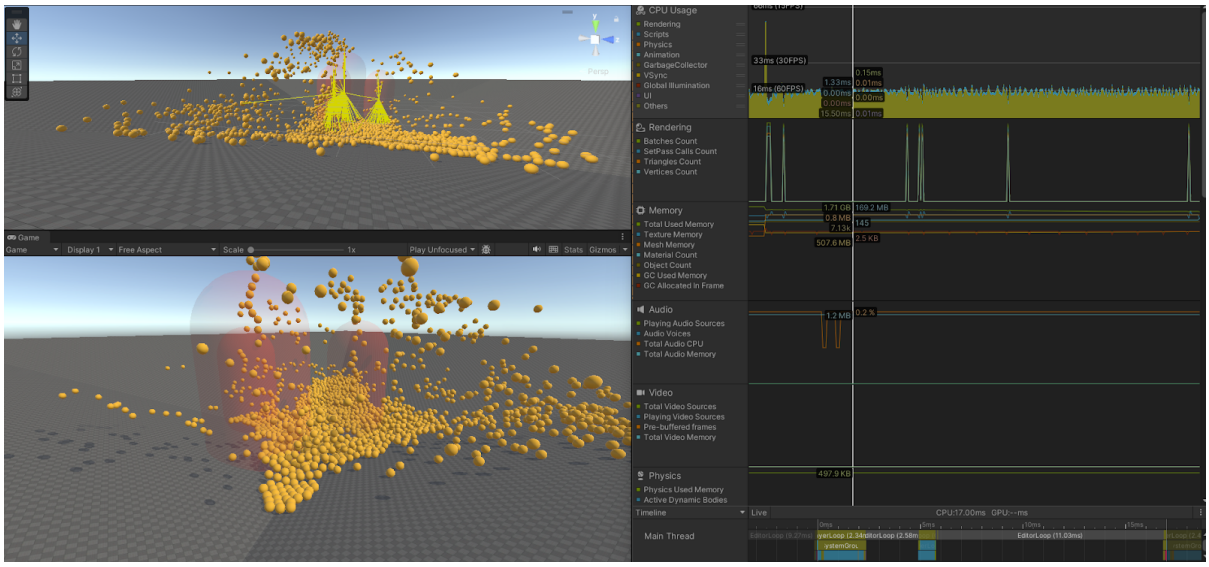


Figure 38: View of the app

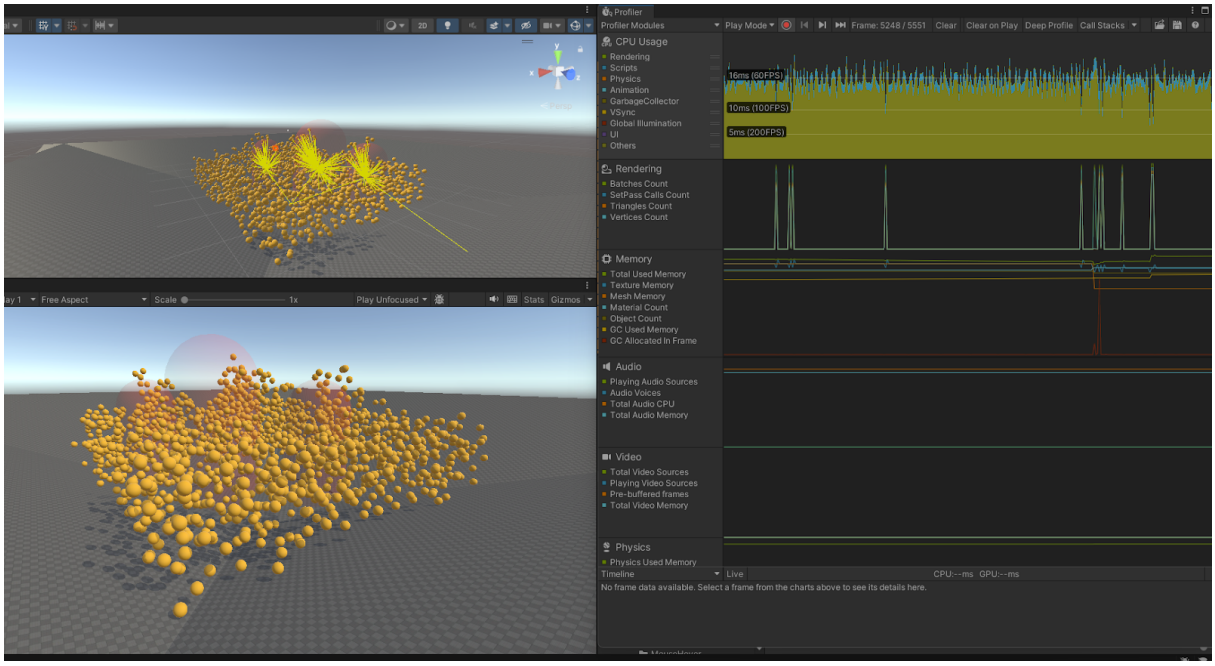


Figure 39: Start of the scene - 60 FPS 10000 particle ECS force field to apply force on particles - all under 16ms.

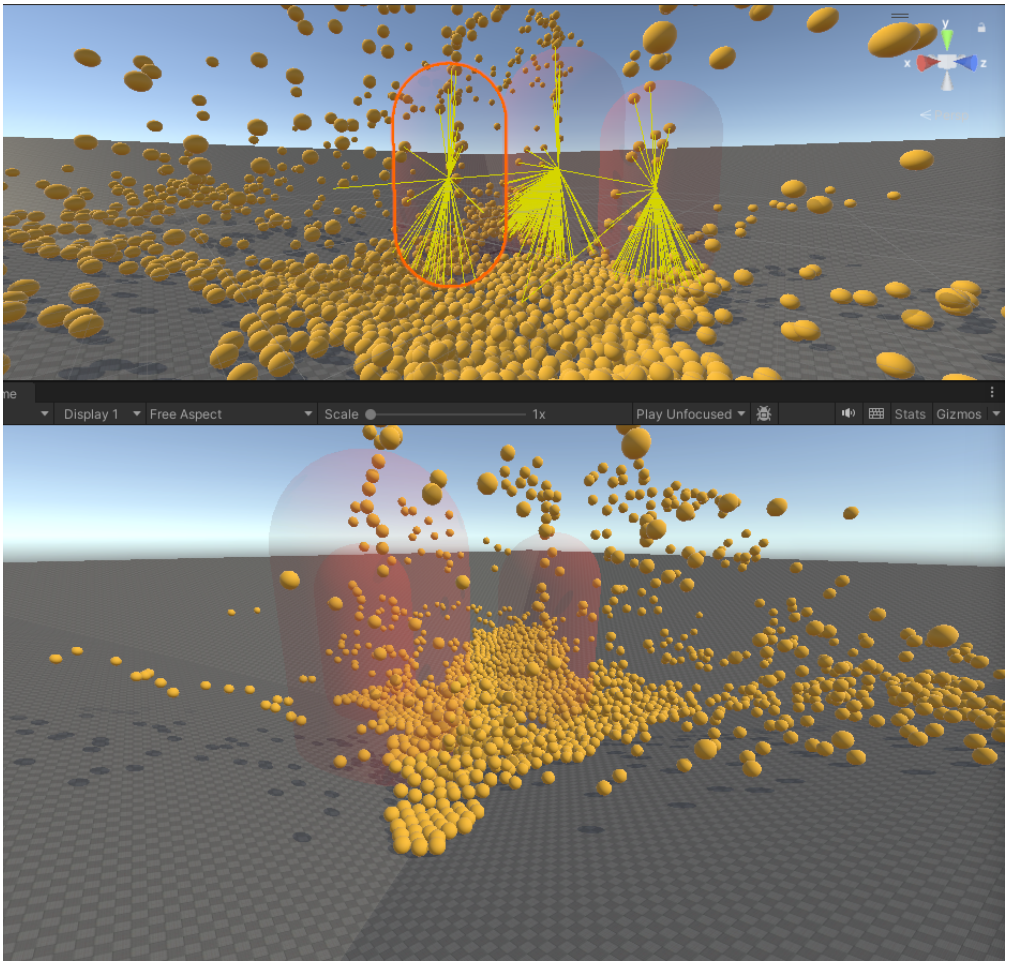


Figure 40: Game playing without lag and any additional memory on 120 FPS.

In these pictures (Figures 34 to 41) there are over 5000 particles, and each particle is under 3 fields of force that influence velocity. This method can be especially useful in mine craft style manipulation where we can improve the detail by converting an object to micro cubes and applying our sculpting tool which here is our hand and the method, we mentioned in Minecraft style.

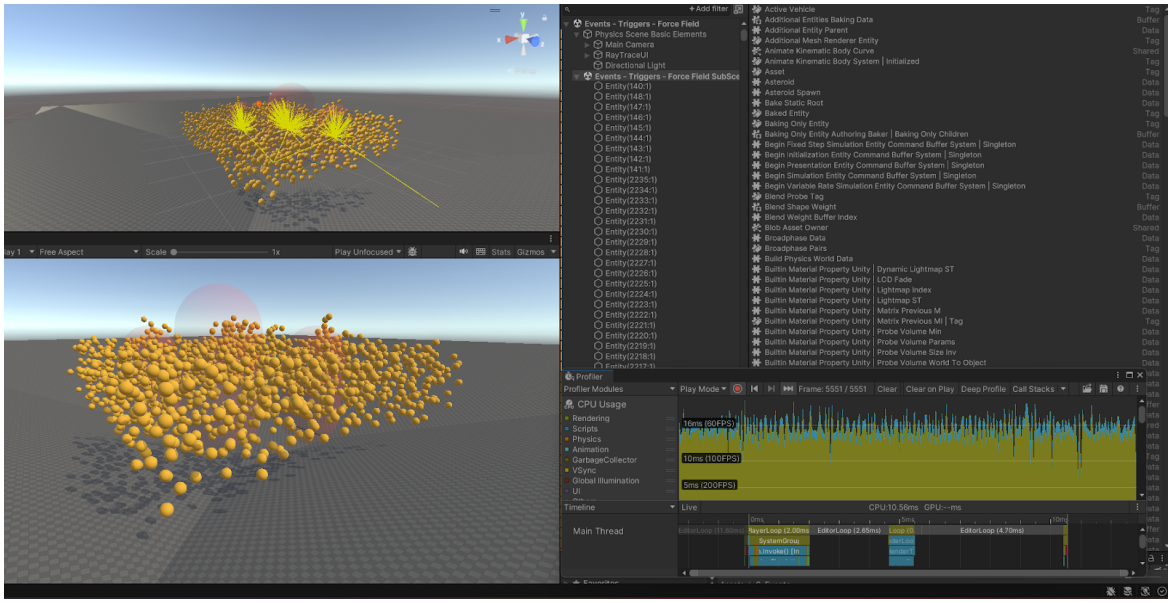


Figure 41: All of the calculations under 16ms on 60 FPS - 10000 objects.

If we can apply this to our system, the result would be fascinating. It can be a kind of drilling approach to do the sculpting job.

Chapter 5
Future Work

5.1 Application of Entity Component System (ECS)

Despite the limitations encountered with the Minecraft-style approach, the experience provided valuable insights into optimization methods. To mitigate the computational challenges, we began exploring the application of the Entity Component System (ECS) within Unity.

Applying ECS to our Minecraft-style approach can be a useful approach. By the end of this study, initial tests showed promising improvements in computational efficiency. Our experience highlights the potential of ECS as a strategy to optimize voxel-based VR sculpting, suggesting a path for further exploration and refinement in future research.

Also applying ECS can be usefull in vertex manipulation where we discussed in the spring simulation chapter.

However, we should keep in mind that using ECS, has some challenges. Because it is in its primary stage and is complicated, there are few developers working on ECS and so the community is small. Also, there are not enough materials and documentation of ECS yet. The last thing that we should have in mind is it is no applicable for all usages, like animations.

5.2 Applying Octree

An octree is a three-dimensional tree data structure used primarily in computer graphics, 3D game development, and computational geometry for the efficient organization and management of spatial data. The term "octree" is derived from the fact that each node in this tree can have up to eight children, symbolizing eight smaller partitions of the parent node's spatial domain.

The concept of the octree can be visualized as a cube (the root of the tree) being split into eight smaller, equal-sized cubes (the children's nodes). This process of subdivision continues recursively, creating deeper levels within the tree until a particular condition is met. Each node in the tree represents a specific spatial subdivision and can store various data types, including points, lines, and polygons.

This can help Minecraft carving style, and mesh deforming style. To handle the nearest vertex or cube to delete, without too much memory problem.

While the benefits of using octrees are significant, their implementation does come with challenges. One of the primary considerations is managing the computational and memory cost of maintaining the octree structure, particularly for dynamic models that change over time. As vertices move or are added or removed due to user interaction, the octree needs to be updated

to reflect the model's new state. Balancing the efficiency gains from using an octree against the overhead of maintaining it is a key challenge that can be examined.

Furthermore, determining the appropriate level of subdivision (tree depth) is a non-trivial task. Too little subdivision, and the efficiency benefits of the octree may not be fully realized. Too much, and the octree becomes overly complex and expensive to maintain. Finding the right balance requires careful analysis and tuning.

5.3 Mine Crafting style improvements

Instantiating objects and rendering them for Unity is computationally heavy. We can use the render instead. With this, we do not have the overhead of the calculation for the object in Unity. We just use its rendering. It is like there is no real object, but we render them in the player view. Although they are not physical objects, we still can manipulate them by having our hands in a world position. However, render codes are a bit more complicated.

5.4 Liquid Simulation improvements

FIn the liquid simulation chapter the Moving Least Squares Material Point Method (MLS-MPM) was used. One approach that may be helpful to consider is the finite volume method (FVM) which is another computational fluid dynamic method [6] and it is used for solving differential equations in finite volume over a grid. Although it may seem computationally heavy, but with having ECS in mind it should be considered for the future work.

Bibliography:

[1] Liu, Y.-S., et al. (2005). Mesh blending. The Visual Computer, 21, 915-927.
[2] Yu, E. (2021). Interactive 3D Sketching in Virtual Reality. Technical University of Denmark, Department of Applied Mathematics and Computer Science.
[3] Rodrigues, L. S., Riehm, F., Zachow, S., & Israel, J. H. (2023). Vox Sculpt: An Open-Source Voxel Library for Tomographic Volume Sculpting in Virtual Reality. In Proceedings of the 9th International Conference on Virtual Reality (ICVR), pp. 515-523. Xianyang, China. Doi: 10.1109/ICVR57957.2023.10169420.
[4] Flick, J. (2018). CatLikeCoding.com.
[5] Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A., & Jiang, C. (2018). A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. ACM Transactions on Graphics, 37, 1-14. Doi: 10.1145/3197517.3201293.
[6] UNITY. (2005). UNITY Company Website. Retrieved from https://docs.unity3d.com/.
[7] Eymard, R., Gallaudet, T., & Herbin, R. (2000). Finite Volume Methods. Handbook of Numerical Analysis, 7. Doi: 10.1016/S1570-8659(00)07005-8.
[8] Dashti, S., Prakash, E., Navarro-Newball, A.A., et al. (2022). Pottery: virtual reality pottery. The Visual Computer, 38, 4035–4055. doi:10.1007/s00371-022-02521-2.
[9] Hanson, A. J., & Ma, H. (1995). Parallel transport approach to curve framing. Indiana University, Tec Reports-TR425, 11:3–7.
[10] Arora, R., Kazi, R. H., Anderson, F., Grossman, T., Singh, K., & Fitzmaurice, G. W. (2017). Experimental evaluation of sketching on surfaces in VR. In Proceedings of CHI Conference on Human Factors in Computing Systems, 17, pp. 5643–5654.
[11] Bae, S.-H., Balakrishnan, R., & Singh, K. (2008). I love ketch: as natural-as-possible sketching system for creating 3d curve models. In Proceedings of the 21st annual ACM symposium on User interface software and technology, pp. 151–160.
[12] Oculus Medium by Facebook. (2022). Retrieved from: https://www.oculus.com/medium/.
[13] Tilt Brush by Google. (2019). Retrieved from:
https://www.tiltbrush.com.
[14] Verhoeven, F., & Sorkin-Hornung, O. (2019). Rod mesh: Two-handed 3d surface modeling in virtual reality.
[15] Soga research group. (2023) Berkeley Engineering. Retrieved from:
https://geomechanics.berkeley.edu/research/comp-geo/mpm/
[16] Terzopoulos, D., & Fleischer, K. (1988). Modeling inelastic deformation: viscoelasticity, plasticity, fracture. SIGGRAPH Comp Graph, 22(4), 269–278.
[17] O'Brien, J., & Hodgins, J. (1999). Graphical modeling and animation of brittle fracture. In Proceedings of the 26th annual conference on Computer graphics and interactive techniques (pp. 137–146). ACM Press/Addison-Wesley Publishing Co.

[18] Hegemann, J., Jiang, C., Schroeder, C., & Teran, J. (2013). A level set method for ductile fracture. In Proc ACM SIGGRAPH/Eurogroup Symp Comp Anim (pp. 193–201).
[19] Wang, Y., Jiang, C., Schroeder, C., & Teran, J. (2014). An adaptive virtual node algorithm with robust mesh cutting. In Proc ACM SIGGRAPH/Eurograph Symp Comp Anim (pp. 77–85). Eurographics Association.
[20] Koschier, D., Bender, J., & Thuerey, N. (2017). Robust extended finite elements for complex cutting of deformable. ACM Trans Graph, 36(4), 55.
[21] Pauly, M., Keiser, R., Adams, B., Durte, P., Gross, M., & Guibos, L. J. (2005). Meshless animation of fracturing solids. ACM Trans Graph, 24(3), 957–964.
[22] Sulsky, D., Zhou, S., & Schreyer, H. (1995). Application of a particle-in-cell method to solid mechanics. Comp Phys Comm, 87(1), 236–252.
[23] Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J., & Selle, A. (2014). Augmented MPM for phase-change and varied materials. ACM Trans Graph, 33(4), 138:1–138:11.
[24] Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A., & Jiang, C. (2018). A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. ACM Trans Graph, 37(4), Article 150, 14 pages. https://doi.org/10.1145/3197517.3201293