Course Title: Introduction To Virtual Reality
Course Code: ENSF 545
Lab Number: B01
Instructor Name: Dr Y. Hu
Student Names: Anand Patel, Priyanka Gautam
Submission Date: December 5th, 2023

Introduction

In the realm of medical training and practice, the precision of needle placement is extremely important. This skill is crucial for various procedures, from routine intravenous injections to more complex surgical applications. Traditional training methods rely heavily on physical practice, often with limitations in terms of availability of resources and realistic experiences. Virtual Reality (VR) technology, coupled with haptic feedback, opens new horizons in medical training, offering a simulated environment that closely mimics real-life scenarios. This project explores the potential of VR in enhancing the accuracy of needle placement, focusing on the simulation of human skin with distinct layers and the integration of haptic feedback.

Project Objectives and Scope

The primary objective of this project is to investigate the impact of haptic feedback on the accuracy of needle placement in a simulated human skin model within a VR environment. This study aims to answer whether consistent haptic feedback across all skin layers or variable feedback corresponding to different layers improves accuracy in needle placement. The scope encompasses the development and testing of two VR applications: HaptoSkin Pro and LayerSense Precision, each mimicking the three layers of human skin: epidermis, dermis, and hypodermis. HaptoSkin Pro provides a consistent feedback sensation, while LayerSense Precision varies its feedback to represent different tissue densities.

Project Details

Tasks & Target Users

The target users initially envisioned for our applications are students interested in needle placement, providing a foundation for broader applications in medical training. While our end goal is to benefit medical professionals and trainees, we're currently using student participants for our research. In our study, students will be instructed to complete the needle placement three times for each application by using a VR needle to aim for the third layer of skin (hypodermis).

Success or failure is indicated by a color change on a cloth located next to the hand. With success, the cloth will turn green and with failure, the cloth will turn red.

Changes from Original Proposal

We've stuck closely to our original plan, with only minor tweaks during the user testing and development phase. In our user testing phase, we decided to collect more quantitative data by asking more questions regarding the experiment which will be discussed further in later sections. For the development phase, we found that it was more intuitive to place the cloth beside the layers of skin as opposed to on top of it. This is because our applications provide a top level view, where the user can't see anything past the first layer of skin, so placing the cloth on top was not reasonable. Additionally, we added a fourth layer (muscle layer) after the third layer which would increase difficulty, otherwise the user would be able to go all the way to the bottom to succeed.

Developed Techniques

Original Interactive Objects

The creation of interactive objects for our virtual reality application was both a complex and exciting process. We utilized a 3D modeling and game development tool, Blender. Blender is a versatile

open-source 3D object creation application. Using Blender, we were able to craft accurate layers of the human skin which was one of the foundations of both our VR applications.

Starting with a simple cube (Figure B1)**,** we initiated the sculpting process to form a detailed hand. This hand model was crucial for providing users with a visually realistic representation of the layers within human skin, depicted with different colors. Different techniques in Blender were employed, picture overlay and a subdivision modifier. By superimposing a real image of a hand onto the cube (Figure B2) we achieved precise shaping, resulting in a realistic view of a hand. We also used a subdivision modifier, which split the faces into smaller faces, giving it a smooth appearance. This technique transformed the hand from square fingers to rounded fingers. Additionally, a smoothing tool was used to refine the geometry and enhance qualities of the hand (Figure B3). Finally, to simulate the complexity of multiple skin layers, we duplicated the colored hand model, each instance representing a distinct depth within the skin structure (Figure B4).

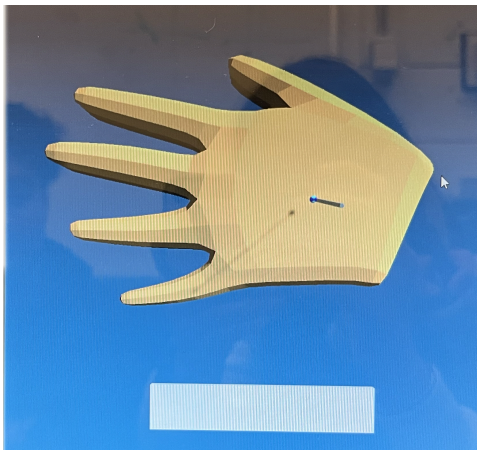
Blender's set of features ensured that the virtual skin layers closely resembled their real-world counterparts. The user of Blender not only allowed for a realistic design of the interactive objects but also provided a platform for further experimentation.

After completing the model in Blender, the focus shifted to integrating them into Unity. The mesh data of the sculpted hands needed to be exported with specific parts in order to be imported into Unity. This integration ensured that the virtual skin layers would respond authentically to user interactions within the VR applications.

For a visual understanding of the developed objects, we have documented this process in our appendices. Appendix A houses our original objects, elucidating the techniques employed in their creation.

Original Scripts

We implemented original interactive scripts in order to achieve our goal in this experiment. We started by integrating pre-defined haptic scripts into each virtual object, representing the distinct layers of simulated skin, this enabled different haptic effects. We then assigned unique IDs to each object within the scene which enabled us to track user interactions.



Our design centered around two central scripts; the haptic effect script and haptic grabber script. Within the haptic effect script, executed in each frame, we added a conditional. This conditional verified whether the haptic grabber was positioned within the predefined ID of the object representing the third layer of the skin. If true, the script dynamically changed the ID of the cloth to green. Conversely, if the grabber was positioned elsewhere within the layers, the ID of the cloth was changed to red. This dynamic change served hand-in-hand with the haptic grabber in order to change color when the button is clicked.

The haptic grabber script, also executed per frame, played a role in the user interaction as well. Activated upon the user's button press, this script has a conditional that verifies whether the cloth object ID was green or red. If green was identified, signifying successful placement of the grabber within the third layer, the script triggered a color transition of the cloth to green. Conversely, if the grabber failed to intersect with the third layer, then, the cloth object ID would be identified as red and the color of the cloth would transition to red.

These two scripts work together in order to provide users with immediate feedback in the accuracy of their needle placement. We have dedicated Appendix B to showcase these scripts which were used for both user and object interactions within Unity.

Detailed Hypotheses

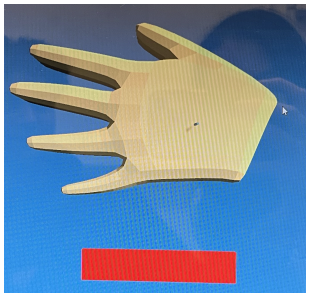
We hypothesize that varying the type of haptic feedback per layer in application 2, will result in better user performance compared to application 1, which has the same type of haptic feedback across all 4 layers. We expect that by providing users with distinct haptic feedback per layer, it will enhance their needle placement accuracy in the third layer.

We hypothesize that application 2 will have a higher user success rate than application 1. This is because we believe that application 2 allows for additional control and precision.

Data Collection

Procedure

1. Preparation
   a. Develop 2 VR Applications: Application 1 - Same type of haptic feedback per layer. Application 2 - Different types of haptic feedback per layer.
2. Conduct user studies
   a. Ask a team to participate in our user studies & get their consent.
   b. Open Application 1
   c. Explain the needle placement task to the users
   d. Ask one person to use the phantom omni stylus to place the needle in the third layer. When the person believes the needle is in the third layer, they press the bottom button on the stylus. Stop the timer. The cloth next to the hand turns red if the needle is not in the third layer, otherwise, it turns green if the needle placement was successful. Record the result.



   e. Ask the person to repeat the task 3x - recording their result per attempt. Calculate how many times the user was successful/failed in completing the task.
   f. Open Application 2
   g. Ask the person to perform the same needle placement task in this new application 3 times. Record their result per attempt and calculate how many times the user was successful/ failed.
   h. Upon completion, ask them to fill out google form (Figure A1) and fill out a likert scale.
   i. Repeat steps B-H for other team-mate.
   j. Repeat step A-I at least three times. (We asked 10 users participate in our study)

3. Analyze data, review google form submissions & likert scale surveys.
4. Put away equipment

Performance Analysis

Methods & Measurement Metrics

We had 2 primary methods for conducting a qualitative evaluation of our applications and evaluating user interaction with our system. We interviewed the users who participated in our user studies using user Feedback and surveys (used google forms and Likert scale).

As for our quantitative evaluation, the primary measurement metric we used for our data analysis was comparing the precision between the two applications.

Our definition of precision involves evaluating how consistently users placed needles in the third layer. To quantify precision, we used two primary metrics; success and failure rate, both relatively to the needle placement in the third layer. Success is defined as placing the needle in the appropriate location, layer 3, (cloth turning green) and failure is described as placing the needle NOT in layer 3. The application with the **higher success rate** indicates **better performance.** The application with a **higher failure rate** indicates **poorer performance**. Using success and failure rate, we were able to deduce which application had a higher precision in user interactions.

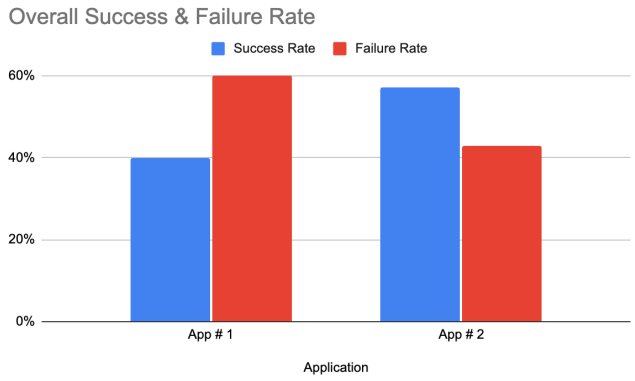
Results

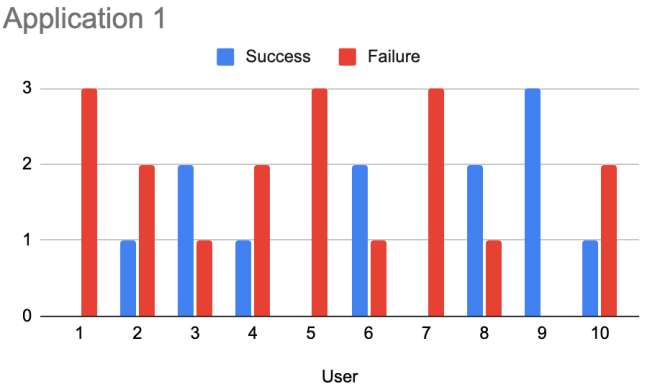
Comparison between two applications

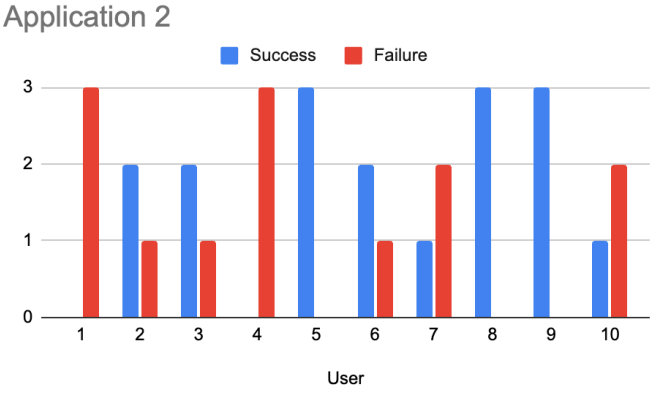
Application	Success Rate	Failure Rate
App # 1	40%	60%
App # 2	57%	43%

Application 1		
User	Success	Failure
1	0	3
2	1	2
3	2	1
4	1	2
5	0	3
6	2	1
7	0	3
8	2	1
9	3	0
10	1	2

Application 2		
User	Success	Failure
1	0	3
2	2	1







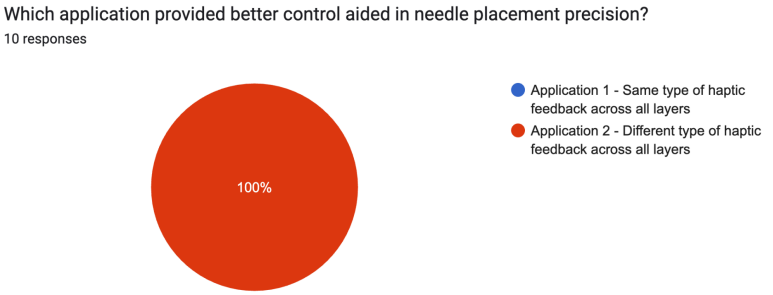
3	2	1
4	0	3
5	3	0
6	2	1
7	1	2
8	3	0
9	3	0
10	1	2

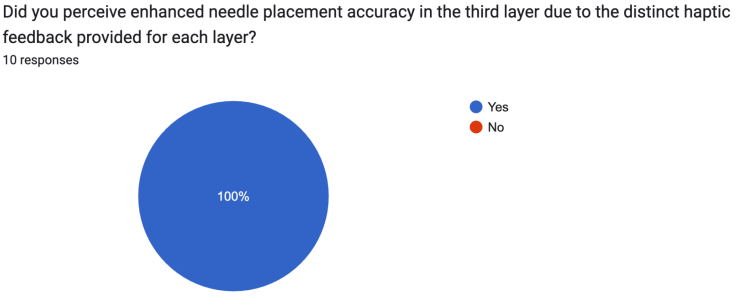
Observations of User Interaction

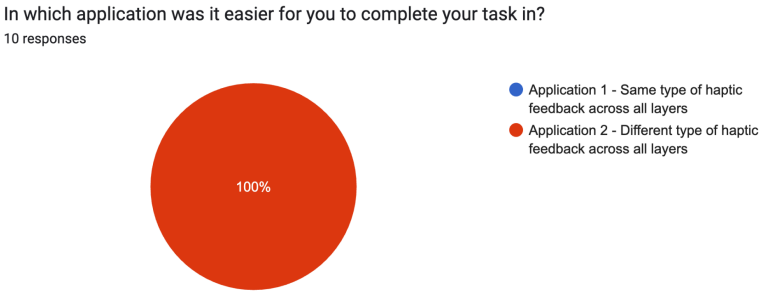
Responses from the google form shows that 100% of the users that participated in the user study felt that varying the haptic feedback felt at each layer of the human skin enhanced their accuracy of needle placement in the third layer. Responses also show that 100% users favored application 2. They believed that not only did application 2 provide better control and aided their needle placement precision but also was easier to accomplish their task as is evident by the pie charts.

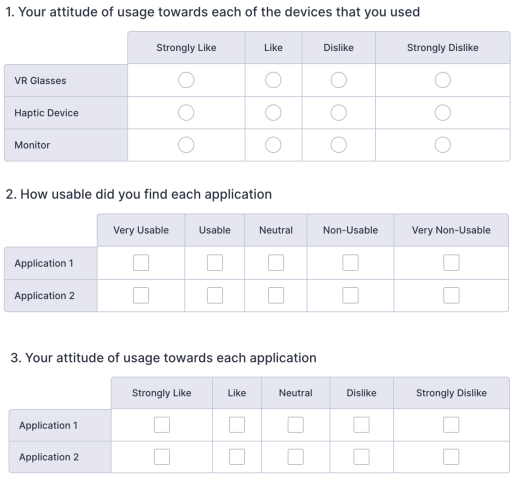
For application 1 some of the users expressed: *"Had to guess at which level you are in and it felt random" & "felt a bit lost" & "It was too smooth, which made needle placement harder"*

For application 2 some of the users expressed: *"A lot better because you know where you are" & "Better sense of where the needle is" & "Different Feedback made placement easier and more intuitive"*









Likert Scale that each user filled out

After evaluating user feedback from the questionnaire, the majority, specifically 8 out of 10 users, marked "strongly like", for VR Glasses, Haptic Device, and Monitor. When it comes to the usability of the applications, application 1 had 6 out of 10 users select "Very Usable". On the other hand, application 2, achieved a slightly higher rating, with 8 out of 10 users choosing "Very Usable." Both applications succeeded in that all users selected "Strongly Like" for both application 1 and 2 in regards to their attitude towards each application.

Critical Analysis & Discussion of Obtained Results

After evaluating the overall performance of user interactions with the both applications and analyzing the success and failure of the applications, it can be seen that our initial hypotheses have been validated.

Our quantitative analysis shows that application 2 had a 17% higher success rate than application 1. Application 2 also had a lower failure rate than application 1. We defined being successful as placing the needle in layer 3, and failure as the needle not being placed in layer 3. This means that more users were successful in application 2, which aligns with our hypothesis that varying haptic feedback per layer results in better user performance. Furthermore, from a quantitative perspective our hypothesis that application 2 would have a higher user success rate was also proven correct.

Our qualitative analysis shows that users favored application 2 because 100% of the users felt that it was easier to complete the task of placing the needle in layer 3 in application 2. All the users that participated in the study found that application 2 allows for additional control and precision which proves our hypothesis correct. Furthermore, it can be deduced from the likert scale surveys that application 2 had a 20% higher Usability rating. This shows that users were leaning towards application 2 because it offers a more favorable experience, especially in terms of control and precision. Some users remarked that Application 2 "...[was] a lot better because you know where you are". We believe that because users could feel the haptic feedback changing as they traversed through the different layers, they could physically feel when they had exited one layer and entered another layer. Application 1 provides a consistent feedback sensation, where the user is unable to differentiate the boundaries of each skin layer leading them to believe they are "...a bit lost" in terms of the needle's position in the layer. When the haptic feedback is uniform across all the layers, it is harder for the user to guess blindly what layer the needle is in. By employing different haptic sensations per layer, the user can make an informed guess because they are able to count how many times the feedback has changed.

While application 2 was more positively received by the users compared to application 1, in terms of control, precision and a higher success rate, there were some challenges that were observed during user interactions. There are certain areas within the challenges that were identified where we could improve the application. For example, the haptic feedback did not vary drastically from layer to layer. Although the haptic feedback was different, the change felt was not not as dramatic. This makes it harder for the user to distinguish the feedback they are experiencing as they are traversing the layers. A future improvement that could be made is to make the haptic feedback change more substantial. By having a more distinct, drastic variation in feedback for each layer could further enhance user experience. It will allow the user to more easily differentiate the feelings of each layer instead of experiencing a blending effect of different layer feedback amalgamating where the feelings of each layer tend to merge together. Moreover, for both of the applications, we noticed a trend of users struggling to position the needle on top of the hand. This resulted from a top-view camera angle that made interacting with the grabber slightly visually difficult. This camera angle was chosen to avoid exposing the boundaries of each layer to the user. However, a future improvement would be to choose a better camera angle for the scene in both of the applications. Lastly, a future improvement that can be made to both of the applications to enhance user experience and aid in better data collection would be that upon failure of a task, tell the user which layer they actually placed the needle in. By displaying the layer the needle was accurately placed in, it will help us as developers identify which layers typically users are more naturally inclined to place the needle in. It will also help the users gain some idea as to where they actually have placed the needle for their next such that they can make a more educated guess for the second or third attempt at successfully placing the needle.

Overall, application 2 had a higher preference rating by the users, particularly in terms of control, precision and overall success rate validating our initial hypotheses.

Final Comments:

Software Engineering Insights

Both of our group members are in the software engineering program where we have learnt about the software development process. Utilizing what we have learnt, we employed agile methods, which means to work on the project through a step by step process focusing on short-term goals but never losing the final goal of the product. We used these agile methods because it meant that we could improve on our applications as we went along. Additionally, we followed the software engineering lifecycle because it helps ensure success by ensuring that we are working on the right activities at the right time. For instance, starting with the implementation phase prior to the completion of the design phase would have led to significant challenges, as it would have left us without a plan on how to begin implementing.

Contributions to Course Project:

Anand

During the execution of the course project, Anand's contributions primarily centered around modeling 3D hands within Blender. This involved experimenting within Blender and learning its capabilities in order to create a detailed and accurate representation of a hand which served as the skin layers in our application. Additionally, he completed the setup of the overall scene within Unity. Lastly, he provided ideas on how to complete the functionality of changing the cloth color which was vital to the overall application.

Priyanka

During the execution of the course project, Priyank's contributions primarily centered around the scripting, which brought the functionality to the applications. This involved implementing additional functionalities in the haptic scripts which allowed for user interactions with the skin layers. Additionally, she provided input on the aesthetic aspects of our project, specifically, on how the scene should look within Unity. Lastly, she brought up the project idea that we elected to undertake and implement.

Collaboration

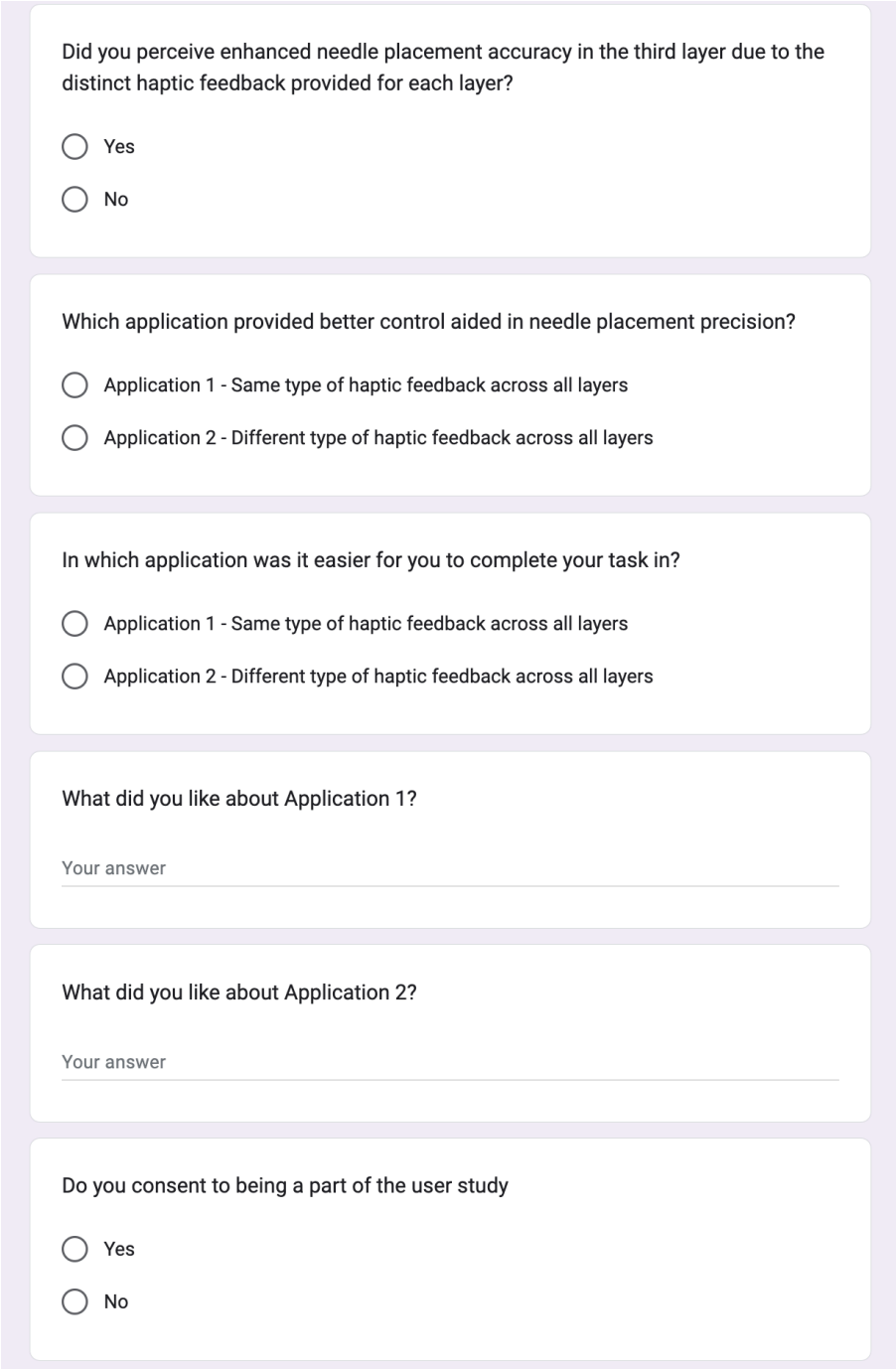
Through collaborative efforts, we worked on the questionnaires and split up the tasks evenly for all reports involved within this project.

References

1. Rodriguez, J. OpenGL API Documentation. docs.GL https://docs.gl/
2. Unity Technologies. Unity Documentation. Unity. https://docs.unity.com/#
3. Blender Foundation. Blender 4.0 Reference Manual. docs.blender.org https://docs.blender.org/manual/en/latest/

Appendix A: Survey

Figure A1: Google Form Survey



Appendix B: Original Objects

Figure B1: Simple Cube

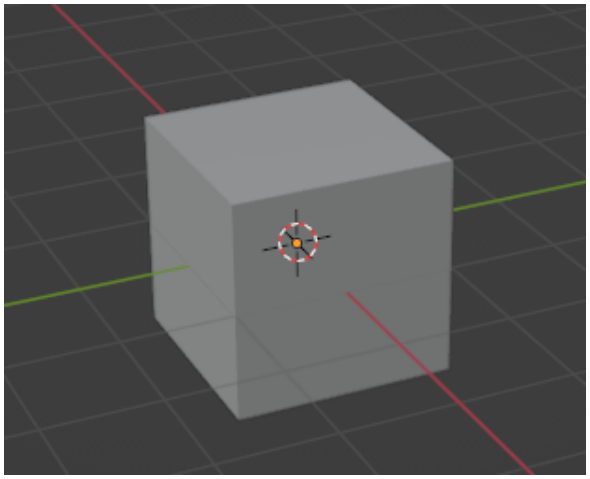


Figure B2: Superimpose image of model on image

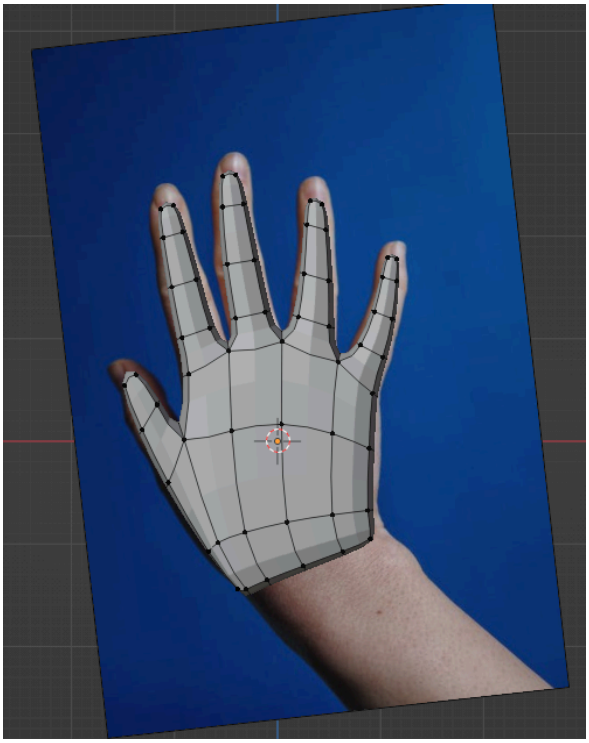


Figure B3: Final Model of Hand

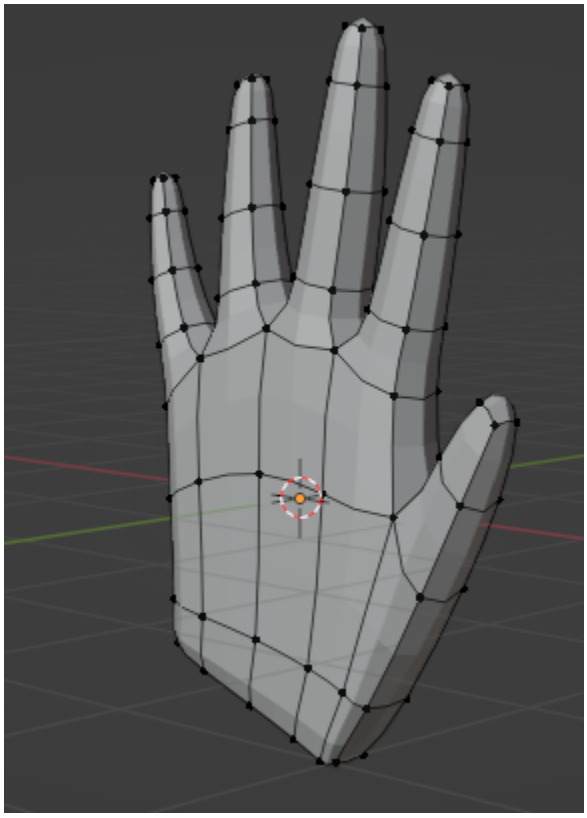


Figure B4: Three Layers of Skin

Appendix C: Unity Scene

Figure C1: Side View of Layers of Skin

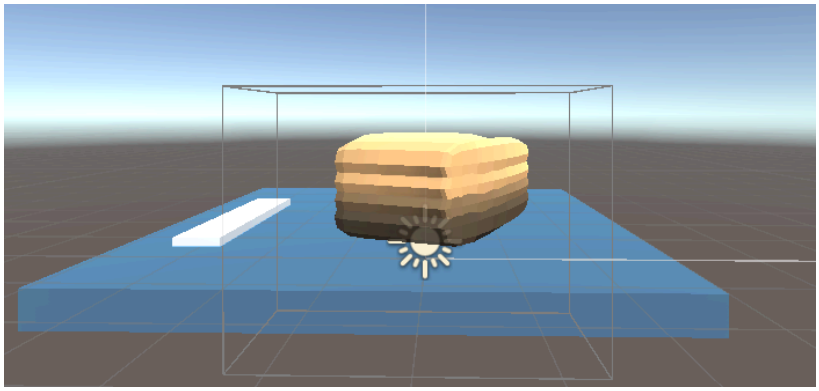


Figure C2: Game View

Appendix D: Original Scripts

Script D1: HapticEffect

```
using System.Timers;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

#if UNITY_EDITOR
using UnityEditor;
#endif

//! This MonoBehavior can be attached to any object with a collider. It will apply a haptic "effect"
//! to any haptic stylus that is within the boundries of the collider.
//! The parameters can be adjusted on the fly.
public class HapticEffect : MonoBehaviour {


        public enum EFFECT_TYPE { CONSTANT, VISCOUS, SPRING, FRICTION, VIBRATE };


        // Public, User-Adjustable Settings
        public EFFECT_TYPE effectType = EFFECT_TYPE.VISCOUS; //!< Which type of effect
occurs within this zone?
        [Range(0.0f,1.0f)] public double Gain = 0.333f;
        [Range(0.0f,1.0f)] public double Magnitude = 0.333f;
        [Range(1.0f,1000.0f)] public double Frequency = 200.0f;
        private double Duration = 1.0f;
        public Vector3 Position = Vector3.zero;
        public Vector3 Direction = Vector3.up;
        public GameObject hand;
        public GameObject Cube1;
        public GameObject hand3;
        public GameObject hand2;


        // Keep track of the Haptic Devices
        HapticPlugin[] devices;
        bool[] inTheZone;                   //Is the stylus in the effect zone?
        Vector3[] devicePoint;   // Current location of stylus
        float[] delta;                      // Distance from stylus to zone collider.
        int[] FXID;                                 // ID of the effect.  (Per device.)
```

```
// These are the user adjustable vectors, converted to world-space.
private Vector3 focusPointWorld = Vector3.zero;
private Vector3 directionWorld = Vector3.up;



//! Start() is called at the beginning of the simulation.
//!
//! It will identify the Haptic devices, initizlize variables internal to this script,
//! and request an Effect ID from Open Haptics. (One for each device.)
//!
void Start ()
{
        //Initialize the list of haptic devices.
        devices = (HapticPlugin[]) Object.FindObjectsOfType(typeof(HapticPlugin));
        inTheZone = new bool[devices.Length];
        devicePoint = new Vector3[devices.Length];
        delta = new float[devices.Length];
        FXID = new int[devices.Length];

        // Generate an OpenHaptics effect ID for each of the devices.
        for (int ii = 0; ii < devices.Length; ii++)
        {
                inTheZone [ii] = false;
                devicePoint [ii] = Vector3.zero;
                delta [ii] = 0.0f;
                FXID [ii] = HapticPlugin.effects_assignEffect(devices [ii].configName);
        }
        hand = GameObject.Find("hand");
        Cube1 = GameObject.Find("Cube1");
        hand3 = GameObject.Find("hand3");
        hand2 = GameObject.Find("hand2");
}


//!  Update() is called once per frame.
//!
//! This function
//! - Determines if a haptic stylus is inside the collider
//! - Updates the effect settings.
//! - Starts and stops the effect when appropriate.
void Update ()
{
```

```
// Find the pointer to the collider that defines the "zone".
Collider collider = gameObject.GetComponent<Collider>();
if (collider == null)
{
        Debug.LogError("This Haptic Effect Zone requires a collider");
        return;
}

// Update the World-Space vectors
focusPointWorld = transform.TransformPoint(Position);
directionWorld =  transform.TransformDirection(Direction);

// Update the effect seperately for each haptic device.
for( int ii = 0; ii < devices.Length; ii++ )
{
        HapticPlugin device = devices [ii];
        bool oldInTheZone = inTheZone[ii];
        int ID = FXID [ii];

        // If a haptic effect has not been assigned through Open Haptics, assign one now.
        if (ID == -1)
        {
                FXID [ii] = HapticPlugin.effects_assignEffect(devices [ii].configName);
                ID = FXID [ii];

                if (ID == -1) // Still broken?
                {
                        Debug.LogError("Unable to assign Haptic effect.");
                        continue;
                }
        }

        // Determine if the stylus is in the "zone".
        Vector3 StylusPos = device.stylusPositionWorld;//World Coordinates
        Vector3 CP = collider.ClosestPoint(StylusPos);   //World Coordinates
        devicePoint[ii] = CP;
        delta[ii] = (CP - StylusPos).magnitude;

        //If the stylus is within the Zone, The ClosestPoint and the Stylus point will be
identical.
        if (delta[ii]  <= Mathf.Epsilon)
        {
                inTheZone [ii] = true;
```

```
                        // Convert from the World coordinates to coordinates relative to the
haptic device.
                        Vector3 focalPointDevLocal =
device.transform.InverseTransformPoint(focusPointWorld);
                        Vector3 rotationDevLocal =
device.transform.InverseTransformDirection(directionWorld);
                        double[] pos = { focalPointDevLocal.x, focalPointDevLocal.y,
focalPointDevLocal.z };
                        double[] dir = { rotationDevLocal.x, rotationDevLocal.y,
rotationDevLocal.z };

                        double Mag = Magnitude;

                        if (device.isInSafetyMode())
                                Mag = 0;

                        // Send the current effect settings to OpenHaptics.
                        HapticPlugin.effects_settings(
                                device.configName,
                                ID,
                                Gain,
                                Mag,
                                Frequency,
                                pos,
                                dir);
                        HapticPlugin.effects_type(
                                device.configName,
                                ID,
                                (int)effectType);

                } else
                {
                        inTheZone [ii] = false;

                        // Note : If the device is not in the "Zone", there is no need to update the
effect settings.
                }

                // If the on/off state has changed since last frame, send a Start or Stop event
to OpenHaptics
                if (oldInTheZone != inTheZone [ii])
                {
                        if (inTheZone [ii])
                        {
```

```
                                        HapticPlugin.effects_startEffect(device.configName, ID );
                                        if (collider.gameObject.name == "hand3")
                                        {
                                                //isTouchingHand3 = true;
                                                Debug.LogError("Touching hand 3 is true");
                                                //Cube1.GetComponent<Renderer>().material.color
= new Color(0,255, 0); // red
                                                Cube1.name = "green";
                                        }
                        else
                        {
                                                Cube1.name = "red";
                                        }

                                        //Cube1.GetComponent<Renderer>().material.color = new
Color(255, 0, 0); // red
                                        //hand.GetComponent<Renderer>().material.color = new
Color(255, 0, 0); // red

                                } else
                                {
                                        HapticPlugin.effects_stopEffect(device.configName, ID);
                                }
                        }

                }
        }

        void OnDestroy()
        {
                //For every haptic device, send a Stop event to OpenHaptics
                for (int ii = 0; ii < devices.Length; ii++)
                {
                        HapticPlugin device = devices [ii];
                        if (device == null)
                                continue;
                        int ID = FXID [ii];
                        HapticPlugin.effects_stopEffect(device.configName, ID);
                }
        }
        void OnDisable()
        {
                //For every haptic device, send a Stop event to OpenHaptics
                for (int ii = 0; ii < devices.Length; ii++)
```

```
		{
			HapticPlugin device = devices [ii];
			if (device == null)
				continue;
			int ID = FXID [ii];
			HapticPlugin.effects_stopEffect(device.configName, ID);
			inTheZone [ii] = false;
		}
}


//! OnDrawGizmos() is called only when the Unity Editor is active.
//! It draws some hopefully useful wireframes to the editor screen.

void OnDrawGizmos()
{
		Gizmos.matrix = Matrix4x4.identity;
		Gizmos.matrix = this.transform.localToWorldMatrix;

		Gizmos.color = Color.white;

		Ray R = new Ray();
		R.direction = Direction;

		if (effectType == EFFECT_TYPE.CONSTANT)
		{
			Gizmos.DrawRay(R);
		}

		Vector3 focusPointWorld = transform.TransformPoint(Position);


		Gizmos.matrix = Matrix4x4.identity;
		Gizmos.color = Color.white;
		if (effectType == EFFECT_TYPE.SPRING)
		{
			Gizmos.DrawIcon(focusPointWorld, "anchor_icon.tiff");
		}

		if (devices == null)
			return;

		// If the device is in the zone, draw a red marker.
		// And draw a line indicating the spring force, if we're in that mode.
```

```
                for (int ii = 0; ii < devices.Length; ii++)
                {
                        if (delta [ii] <= Mathf.Epsilon)
                        {
                                Gizmos.color = Color.red;
                                Gizmos.DrawWireSphere(devicePoint[ii], 1.0f);
                                if(effectType == EFFECT_TYPE.SPRING)
                                        Gizmos.DrawLine(focusPointWorld, devicePoint [ii]);
                        }
                }

        }


}

#if UNITY_EDITOR
[CustomEditor(typeof(HapticEffect))]
public class HapticEffectEditor : Editor
{
        override public void OnInspectorGUI()
        {
                HapticEffect HE = (HapticEffect)target;

                if (HE.gameObject.gameObject.GetComponent<Collider>() == null)
                {

EditorGUILayout.LabelField("************************************************************");
                        EditorGUILayout.LabelField("   Haptic Effect must be assigned to an object with
a collider.");

EditorGUILayout.LabelField("************************************************************");

                } else
                {
                        HE.effectType =
(HapticEffect.EFFECT_TYPE)EditorGUILayout.EnumPopup("Effect Type", HE.effectType);


                        switch (HE.effectType)
                        {
                        case HapticEffect.EFFECT_TYPE.CONSTANT:
                                HE.Direction = EditorGUILayout.Vector3Field("Direction",
HE.Direction);
```

```
                                HE.Magnitude = EditorGUILayout.Slider("Magnitude",
(float)HE.Magnitude, 0.0f, 1.0f);
                                break;
                        case HapticEffect.EFFECT_TYPE.FRICTION:
                                HE.Gain = EditorGUILayout.Slider("Gain", (float)HE.Gain, 0.0f, 1.0f);
                                HE.Magnitude = EditorGUILayout.Slider("Magnitude",
(float)HE.Magnitude, 0.0f, 1.0f);
                                break;
                        case HapticEffect.EFFECT_TYPE.SPRING:
                                HE.Gain = EditorGUILayout.Slider("Gain", (float)HE.Gain, 0.0f, 1.0f);
                                HE.Magnitude = EditorGUILayout.Slider("Magnitude",
(float)HE.Magnitude, 0.0f, 1.0f);
                                HE.Position = EditorGUILayout.Vector3Field("Position", HE.Position);
                                break;
                        case HapticEffect.EFFECT_TYPE.VIBRATE:
                                HE.Gain = EditorGUILayout.Slider("Gain", (float)HE.Gain, 0.0f, 1.0f);
                                HE.Magnitude = EditorGUILayout.Slider("Magnitude",
(float)HE.Magnitude, 0.0f, 1.0f);
                                HE.Frequency = EditorGUILayout.Slider("Frequency",
(float)HE.Frequency, 1.0f, 1000.0f);
                                HE.Direction = EditorGUILayout.Vector3Field("Direction",
HE.Direction);
                                break;
                        case HapticEffect.EFFECT_TYPE.VISCOUS:
                                HE.Gain = EditorGUILayout.Slider("Gain", (float)HE.Gain, 0.0f, 1.0f);
                                HE.Magnitude = EditorGUILayout.Slider("Magnitude",
(float)HE.Magnitude, 0.0f, 1.0f);
                                break;

                        }
                }

        }

}

#endif
```

Script D1: HapticGrabber

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
//using HapticPlugin;


//! This object can be applied to the stylus of a haptic device.
//! It allows you to pick up simulated objects and feel the involved physics.
//! Optionally, it can also turn off physics interaction when nothing is being held.
public class HapticGrabber : MonoBehaviour
{
	public int buttonID = 0;		//!< index of the button assigned to grabbing.  Defaults to the first button
	public bool ButtonActsAsToggle = false;		//!< Toggle button? as opposed to a press-and-hold setup?  Defaults to off.
	public enum PhysicsToggleStyle{ none, onTouch, onGrab };
	public PhysicsToggleStyle physicsToggleStyle = PhysicsToggleStyle.none;	//!< Should the grabber script toggle the physics forces on the stylus?

	public bool DisableUnityCollisionsWithTouchableObjects = true;

	private  GameObject hapticDevice = null;	//!< Reference to the GameObject representing the Haptic Device
	private bool buttonStatus = false;				//!< Is the button currently pressed?
	private GameObject touching = null;				//!< Reference to the object currently touched
	private GameObject grabbing = null;				//!< Reference to the object currently grabbed
	private FixedJoint joint = null;		//!< The Unity physics joint created between the stylus and the object being grabbed.
	public GameObject hand;
	public GameObject Cube1;
	public GameObject hand3;
	bool isTouchingHand3;
	public GameObject hand2;

	//! Automatically called for initialization
	void Start ()
	{
		if (hapticDevice == null)
		{

			HapticPlugin[] HPs = (HapticPlugin[])Object.FindObjectsOfType(typeof(HapticPlugin));
			foreach (HapticPlugin HP in HPs)
			{
```

```
                        if (HP.hapticManipulator == this.gameObject)
                        {
                                hapticDevice = HP.gameObject;
                        }
                }

        }

        if ( physicsToggleStyle != PhysicsToggleStyle.none)
                hapticDevice.GetComponent<HapticPlugin>().PhysicsManipulationEnabled =
false;

        if (DisableUnityCollisionsWithTouchableObjects)
                disableUnityCollisions();
        hand = GameObject.Find("hand");

        hand3 = GameObject.Find("hand3");
        hand2 = GameObject.Find("hand2");
    }


    void disableUnityCollisions()
    {
        GameObject[] touchableObjects;
        touchableObjects =  GameObject.FindGameObjectsWithTag("Touchable") as
GameObject[];  //FIXME  Does this fail gracefully?

        // Ignore my collider
        Collider myC = gameObject.GetComponent<Collider>();
        if (myC != null)
                foreach (GameObject T in touchableObjects)
                {
                        Collider CT = T.GetComponent<Collider>();
                        if (CT != null)
                                Physics.IgnoreCollision(myC, CT);
                }

        // Ignore colliders in children.
        Collider[] colliders = gameObject.GetComponentsInChildren<Collider>();
        foreach (Collider C in colliders)
                foreach (GameObject T in touchableObjects)
                {
                        Collider CT = T.GetComponent<Collider>();
                        if (CT != null)
```

```
                                        Physics.IgnoreCollision(C, CT);
                        }

        }


        //! Update is called once per frame
        void FixedUpdate()
        {
                bool newButtonStatus =
hapticDevice.GetComponent<HapticPlugin>().Buttons[buttonID] == 1;
                bool oldButtonStatus = buttonStatus;
                buttonStatus = newButtonStatus;


                if (oldButtonStatus == false && newButtonStatus == true)
                {

                        if(GameObject.Find("red"))
        {

(GameObject.Find("red")).GetComponent<Renderer>().material.color = new Color(255, 0, 0);

                        }
                        else if(GameObject.Find("green"))
        {

(GameObject.Find("green")).GetComponent<Renderer>().material.color = new Color(0, 255, 0);
                        }
                }
                if (oldButtonStatus == true && newButtonStatus == false)
                {
                        if (ButtonActsAsToggle)
                        {
                                //Do Nothing
                        }
                        else
                        {
                                release();
                        }
                }

                // Make sure haptics is ON if we're grabbing
                if (grabbing && physicsToggleStyle != PhysicsToggleStyle.none)
```

```
                hapticDevice.GetComponent<HapticPlugin>().PhysicsManipulationEnabled =
true;
        if (!grabbing && physicsToggleStyle == PhysicsToggleStyle.onGrab)
                hapticDevice.GetComponent<HapticPlugin>().PhysicsManipulationEnabled =
false;

        /*
        if (grabbing)
                hapticDevice.GetComponent<HapticPlugin>().shapesEnabled = false;
        else
                hapticDevice.GetComponent<HapticPlugin>().shapesEnabled = true;
                */

    }

    private void hapticTouchEvent( bool isTouch )
    {
        if (physicsToggleStyle == PhysicsToggleStyle.onGrab)
        {
                if (isTouch)

hapticDevice.GetComponent<HapticPlugin>().PhysicsManipulationEnabled = true;
                else
                        return; // Don't release haptics while we're holding something.
        }

        if( physicsToggleStyle == PhysicsToggleStyle.onTouch )
        {
                hapticDevice.GetComponent<HapticPlugin>().PhysicsManipulationEnabled =
isTouch;
                GetComponentInParent<Rigidbody>().velocity = Vector3.zero;
                GetComponentInParent<Rigidbody>().angularVelocity = Vector3.zero;

        }
    }

    void OnCollisionEnter(Collision collisionInfo)
    {
        Collider other = collisionInfo.collider;
    Debug.Log("OnCollisionEnter : " + other.name);
        GameObject that = other.gameObject;
         Debug.Log("Touching : " + that.name + " Has no body. Finding Parent. ");
         Debug.Log("Touching : " + collisionInfo.gameObject.name + " Has no body. Finding
Parent. ");
```

```
		/*if (collisionInfo.gameObject.name == "hand3")
		{
			isTouchingHand3 = true;
			Debug.LogError("Touching hand 3 is true");
		}*/
		Rigidbody thatBody = that.GetComponent<Rigidbody>();

		// If this doesn't have a rigidbody, walk up the tree.
		// It may be PART of a larger physics object.
		while (thatBody == null)
		{
			//Debug.logger.Log("Touching : " + that.name + " Has no body. Finding Parent.
");
			if (that.transform == null || that.transform.parent == null)
				break;
			GameObject parent = that.transform.parent.gameObject;
			if (parent == null)
				break;
			that = parent;
			thatBody = that.GetComponent<Rigidbody>();
		}

		if( collisionInfo.rigidbody != null )
			hapticTouchEvent(true);

		if (thatBody == null)
			return;

		if (thatBody.isKinematic)
			return;

		touching = that;



	}
	void OnCollisionExit(Collision collisionInfo)
	{
		Collider other = collisionInfo.collider;
		//Debug.unityLogger.Log("onCollisionrExit : " + other.name);

		if( collisionInfo.rigidbody != null )
			hapticTouchEvent( false );
```

```
                if (touching == null)
                        return; // Do nothing

                if (other == null ||
                   other.gameObject == null || other.gameObject.transform == null)
                        return; // Other has no transform? Then we couldn't have grabbed it.

                if( touching == other.gameObject ||
other.gameObject.transform.IsChildOf(touching.transform))
                {
                        touching = null;
                }
        }

        //! Begin grabbing an object. (Like closing a claw.) Normally called when the button is pressed.
        void grab()
        {
                GameObject touchedObject = touching;
                if (touchedObject == null) // No Unity Collision?
                {
                        // Maybe there's a Haptic Collision
                        touchedObject = hapticDevice.GetComponent<HapticPlugin>().touching;
                }
                if (grabbing != null) // Already grabbing
                        return;
                Cube1.GetComponent<Renderer>().material.color = new Color(255, 0, 0); // red
                /*if (touchedObject.name == "hand3")
      {
                        isTouchingHand3 = true;
      }*/
                Debug.Log("touching " + touching);
                //Debug.Log("Touching : " + collisionInfo.gameObject.name + " Has no body. Finding
Parent. ");
                /*

                                if (touchedObject == null) // Nothing to grab
                                        return;

                                // Grabbing a grabber is bad news.
                                if (touchedObject.CompareTag("Gripper"))
                                        return;

                                Debug.Log( " Object : " + touchedObject.name + "  Tag : " +
touchedObject.tag );
```

```
                        grabbing = touchedObject;

                        //Debug.logger.Log("Grabbing Object : " + grabbing.name);
                        Rigidbody body = grabbing.GetComponent<Rigidbody>();

                        // If this doesn't have a rigidbody, walk up the tree.
                        // It may be PART of a larger physics object.
                        while (body == null)
                        {
                                //Debug.logger.Log("Grabbing : " + grabbing.name + " Has no
body. Finding Parent. ");
                                if (grabbing.transform.parent == null)
                                {
                                        grabbing = null;
                                        return;
                                }
                                GameObject parent = grabbing.transform.parent.gameObject;
                                if (parent == null)
                                {
                                        grabbing = null;
                                        return;
                                }
                                grabbing = parent;
                                body = grabbing.GetComponent<Rigidbody>();
                        }

                        joint = (FixedJoint)gameObject.AddComponent(typeof(FixedJoint));
                        joint.connectedBody = body;
                */
        }
        //! changes the layer of an object, and every child of that object.
        static void SetLayerRecursively(GameObject go, int layerNumber )
        {
                if( go == null ) return;
                foreach(Transform trans in go.GetComponentsInChildren<Transform>(true))
                        trans.gameObject.layer = layerNumber;
        }

        //! Stop grabbing an obhject. (Like opening a claw.) Normally called when the button is released.
        void release()
        {
                if( grabbing == null ) //Nothing to release
                        return;
```

```
                Debug.Assert(joint != null);

                joint.connectedBody = null;
                Destroy(joint);



                grabbing = null;

                if (physicsToggleStyle != PhysicsToggleStyle.none)
                        hapticDevice.GetComponent<HapticPlugin>().PhysicsManipulationEnabled =
false;

        }

        //! Returns true if there is a current object.
        public bool isGrabbing()
        {
                return (grabbing != null);
        }
}
```

Appendix E: Consent Form