Towards Naturally Grabbing and Moving Objects in VR

Jonathan Lin, Jürgen P. Schulze; Qualcomm Institute, University of California San Diego, La Jolla, CA

Abstract

Many VR applications require picking and moving of virtual objects. Many gesture based solutions do not work reliably, whereas controller based methods are not as natural as hand pose recognition. Tethered controllers add the issue of a cable being in the user's way. We developed a gesture interface using a Leap Motion finger tracker attached to an Oculus Rift DK2 and implemented three ways of interacting with objects: innate pinching, magnetic force, and a physical button attached to the index finger. We built a virtual reality test scenario, in which the user needs to move virtual objects between shelves to sort them. Initial testing shows that grabbing with the button works better than the other two more natural methods. Besides the user interaction techniques, we also report on our practical experiences using Oculus Rift, Leap Motion and the button with the Unity 3D development platform.

Introduction

The purpose of this project is to create a software application for researchers in the psychiatry department to observe how people organize objects. Subjects wear the Oculus Rift virtual reality head mounted display, and sit in a swivel chair. Attached to the front of the Oculus Rift is the Leap Motion, a stereo infrared camera that detects hand and finger movements. While wearing these devices, subjects see a virtual room full of objects, where they can pick up an item on a table or bookshelf and place the object elsewhere in the room. Behavioral researchers can record the interactions for later study. The goal of this project is not only to benefit researchers in studying human organizational habits, but also to optimize gesture based 3D user interaction. This paper focuses on the latter.

In conjunction with the hardware we use the game development system Unity 3D and its asset store, which provides ready-made 3D models and animations. We modeled a 3D room with a table, shelves, and dishes of different types and in different colors, which vaguely resemble a kitchen scenario. The Leap Motion is used to detect the user's hands and surroundings and operates in pass through mode, so that the user will not unintentionally collide with real-world objects. With the Oculus Rift's position tracker, the users are able to move around the virtual world much like in the real one so they can interact with the 3D objects.

One of the challenges for this project is to reliably detect the motions of grabbing and releasing objects in the virtual world. It is hard for interactive 3D systems which allow users to grab objects to do this realistically, because it is hard to accurately track hand movement, and to measure pressure and friction applied towards an object. We are not addressing the latter two aspects either, but we do make an effort to track hand motion more realistically by employing the Leap Motion camera.

Another challenge is to provide the user a large enough working volume, because it has to be within the range the Oculus Rift's positional tracker supports. Our kitchen scenario does not require walking but is still large enough to barely fit within the tracking volume.

Related Work

Interaction with objects in virtual environments has been a subject of research since the advent of virtual reality. The use of direct interaction is generally considered more intuitive than indirect approaches. However, it limits the reach of the user to nearby objects. Indirect interaction techniques such as the virtual laser pointer technique allow interaction with objects further than an arm's length away by casting a 3D ray into the world and test for intersections with objects [6]. Other examples of indirect interaction are arm-extension techniques, such as HOMER (Hand-centered Object Manipulation Extending Ray-casting) or Go-Go. The HOMER technique also casts a ray to virtual objects for selection, but the virtual hand moves to the object position instead of the object being attached to the casted ray [7]. The Go-Go technique interacts with virtual objects by nonlinearly scaling the hand positions once the physical hand goes past the threshold to reach virtual objects [8]. Although indirect interaction may reach faraway objects, it has reduced performance compared to manipulating objects co-located with one's hand [9]. We chose not to use indirect interaction techniques because they do not resemble real life interaction with objects enough to draw conclusions from the VR experience for the real world in our behavioral research setting.

Direct interaction creates a more natural experience for the user in an immersive VR environment. However, according to Paul et al. [5], direct interaction has perceptual limitations, where lower regions that are further from the viewing axis are worse for precise tasks. They propose that when designing 3D user interfaces for direct selection with immersive HMD environments that require high precision, they should be restricted to a level close to the eyes despite more comfortable interaction in the lower regions. Direct interaction either uses a constrained virtual hand or a penetrating hand to grasp virtual objects [5]. A common approach for a constrained virtual hand is the physically based approach. Zachmann et al. [10] used the distribution of contacts from fingers and palm to differentiate between three states: push, precision grasp, and power grasp to grasp virtual objects. Borst and Indugula [11] use linear and torsional spring dampers between the virtual palm and fingers of the interconnected rigid bodies hand model. However, a high friction coefficient is needed for grasping.

Even though the constrained virtual hand is the most natural approach, the penetrating hand approach gives better performance [3]. Prachyabrued and Borst [3] use visual feedback for virtual grasping to improve behavior after real fingers enter a virtual object. They evaluated eight visual feedback methods for grasping: Outer Hand, Inner Hand, See-through, 2-hand, finger color, object color, arrow and vibration. Object color and finger color gave the best user experience and inner hand had the best performance.

As indirect and direct interaction have their own benefits, techniques such as Bendcast and Expand that use both were proposed. Cashion et al. [4] created an auto-select framework that contains techniques: Bendcast and Expand choose the most optimal technique in a given situation. The Bendcast technique selects the closest object to the cursor [4]. Expand takes clones of all objects that are inside the cursor when the selection is triggered and place them into a virtual grid. The virtual grid is spaced evenly to avoid object occlusion and collision, and is used for object selection [4, 12], but the implementation uses a Play Station Move controller to control a cursor, instead of gestures. Of the three techniques, Bendcast performed the fastest, but had the highest error rate.

In our implementation, direct interaction was chosen for its naturalness. The Bendcast selection technique was chosen for its performance and object color visual feedback technique was used to balance the error rate. As our application requires high precision for selecting virtual objects, we followed Paul et al.'s [5] guidelines and restricted interactions to a level close to the eyes despite more comfortable interactions in the lower regions. Our setup of the Oculus Rift with the attached Leap Motion tracker automatically ensures that interactions only happen around the center of the user's field of view.

Description of the Application

We implemented our software application with Unity 5.1.2p2, including its VR support module. It runs on an Intel Core i7-5820K CPU, with a Nvida GeForce GTX 980 GPU, which renders to an Oculus Rift Development Kit 2, to which a Leap Motion tracker is attached. Oculus PC SDK 0.7.0.0-beta was used.

Our simulation scene contains two sets of shelves, a table and dinnerware: four plates, cups, bowls, and pots. Since our objective is a realistic simulation of organizing objects in a room, the distances and directions the user moves in the virtual world needs to be proportional to the amount moved in the physical world. Thus, the simulated room was chosen to be a small size due to the tracking limit of Oculus Rift positional tracker as shown in Figure 1 [13]. Due to this reason, the shelves are positioned with just enough space in-between them for the user to comfortably interact with all the objects.

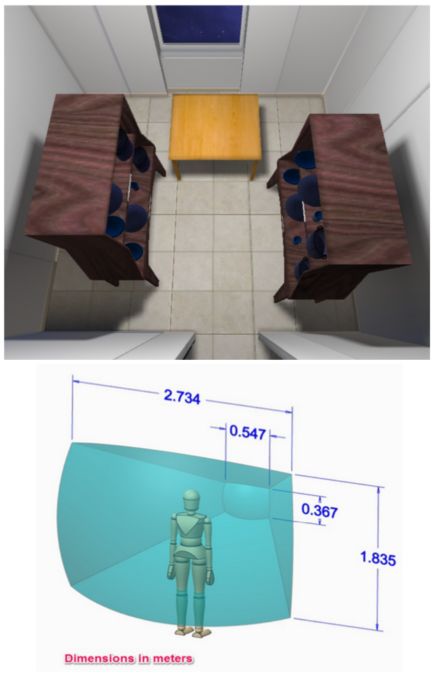

Figure 1: *Above: Simulated room setup; below: Oculus DK2 positional tracking volume visualization*

Limitations

Hardware

The Leap Motion is used for real-time hand tracking, but it cannot detect a hand when it is occluded because the controller uses optical sensors and infrared light for tracking. Another hardware limitation is the tracking range of 25 to 600 millimeters (1 inch to 2 feet) above the device [14], which forces users to pick up only objects within their view. As a result of this limitation, some users are not be able to use the full length of their arms.

Software

When the leap motion initially detects a hand, it often identifies it as the wrong hand (right vs. left). This poses a challenge for the application. Another problem presented by the Leap Motion is its false interpretation of inanimate objects in its range as another hand, such as a floor lamp or curtains. This interferes with the implementation, as it is able to pick objects up if it is in a pinching state. When more than one hand is used, the interaction accuracy drops dramatically, so we did the best we could to keep the tracking space free from objects in the tracking range.

Implementation

Scaling

All 3D objects in the scene are from the Unity asset store. We scaled them to match their real-world counterparts, using Unity's unit size of one meter. The scaling of the x, y, z axes is done using the object's axis-aligned bounding box. We constructed all objects in reference to measurements found in IKEA's catalogue, see Table 1. An interesting thing to note is that the Oculus SDK for Unity uses the same unit size.

The user is placed in the middle of two opposing shelves, facing a table, so that the user's hands are in range of both shelves and the table.

Table 1: Sizes of various Ikea objects.

	Width	**Height**	**Diameter**
Cup	0.092m	0.120m	0.092m
Bowl	0.16m	0.07m	0.16m
Pot	0.23m	0.125m	0.23m
Plate	0.254m	0.03m	0.254m
Book Shelf	0.77m	1.68m	0.39m
Table	1.0m	0.74m	0.6m
Floor tile	0.4064m	0.0127m	0.4064m

Furniture

We made the room the users find themselves in 10ft x 10ft to optimize it to the range of the Oculus' positional tracker and a natural speed of motion. This way the user will feel that they have access to the whole room. The shelf width was scaled 1.5 times more in width so that it is easier for the user to pick up and place items correctly. The plaque or labels on the shelves are 0.42m wide and 0.1m high. The width of the labels was chosen so that the user can read them without issues.

Sorting Detection

At the start of the application, the positions of dinnerware and labels are randomized. When a dinnerware object collides with a shelf row, it checks the label of the row to determine if it is placed correctly. If the dinnerware object collides with another dinnerware object, it checks if the collided object is colliding with a shelf row, then looks up the label of the shelf row.

Hand

The player controller, hand, and hand controller are implemented by using the LeapOVRPlayerController, RigidHand, and the HandController prefab from the LeapMotionCoreAsset 2.3.1. Physical interaction with objects other than grabbable objects is turned off.

Grabbing Objects

The implementation of grabbing an object can be found in LeapMotionCoreAsset 2.3.1 for Unity. It contains two utility scripts named GrabbingHand.cs and GrabbableObject.cs, where the RigidHand prefab needs to attach GrabbingHand.cs. All interactive objects need to have GrabbableObject.cs attached. The grabbing object distance used was 0.2m as it seems just enough distance to correctly select an object. The grab trigger distance used was 70cm. Lower values make it hard to determine if an object is grasped. The release trigger distance used was 1.5m because it is harder to release when set higher, and when set lower the grasped object would release unintentionally.

The first implementation was done with innate pinching. This approach requires grasping an object with a pinch gesture. The pinch gesture is determined by finding the closest distance from the tip of the thumb to the joints of all fingers. If the distance is below a threshold, we detect a pinch. We release the object when the closest distance is greater than the grab trigger threshold less than the release trigger distance. When releasing, we reduce the holding force on the object so the object will gradually get dropped instead of abruptly. The released state is determined when the closest thumb to finger distance is greater than the release trigger distance. This technique depends on accurate finger tracking, which cannot always be provided by an optical tracking system like the Leap Motion which views the hands from only one direction.

Our second implementation treats each finger as a magnet. To grab an object, a finger will magnetically attract an object within the grabbing object distance. We trigger magnetic grasping when the distance between the closest object and the halfway point between the tips of the thumb and index fingers is below a threshold. We release the object when it collides with a shelf or the table. This implementation solves the accuracy problem the pinching implementation presented. However, it also requires more concentration to select and deselect objects. It is often hard to correctly select objects which are close together. And it can happen that a grasped object gets dropped accidentally by colliding with other objects or the furniture.

Our third implementation uses a Genius Ring Mouse to trigger grasping. The Ring Mouse is put on a finger like a ring and provides two buttons, of which we only use one. To grab an object, the user clicks and holds down the right button on the ring mouse when the user's hand is within range of an object. When the button is released, the object is dropped. This implementation solves the issues from the previous implementations with the exception of selecting objects, which can still be inaccurate due to tracking jitter.

Object Selection

To indicate which object is about to be grasped, we highlight the object this would apply to. The object's highlight is created with a halo effect centered at the object's rendered axis-aligned box center.

The ring mouse implementation limits the user to using only one hand to grab objects because only one ring mouse may be used. Thus a problem arises when there are two hands because the other hand will still be able to grab objects near it although it is not the one with the ring mouse. We solve this problem by defining that only one hand can grab objects.

Holding Objects

As a result of Leap Motion's hardware limitations, hand tracking is not always accurate, which can result in the loss of tracking of the hand. When an object is being held and this happens, the object will register that it is not being held and falls down, as Unity's physics engine defines. The problem is solved by having the held object float in air when the hand is lost before heading towards the recreated hand with a magnetic force. A magnetic force rather than a sudden reappearance of the object is used because it is more visually pleasing. The object can be deselected at any time during this process. However, when a hand is recreated, the Leap Motion may create the wrong hand and will fix itself by recreating the correct hand once the user physically moves the hand. Another problem occurs when a hand is accidentally lost and the object is in a held state. The object will register any new hand that appears as the one being held. This creates a problem when the hand was created from a nearby inanimate object in reality. This issue has been resolved by creating a ray in the user view direction, and use the hand with the shortest distance intersecting the ray.

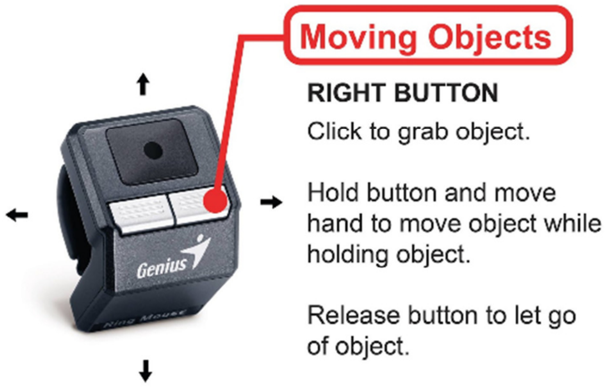
Moving Objects
RIGHT BUTTON
Click to grab object.
Hold button and move hand to move object while holding object.
Release button to let go of object.
Genius
Ring Mouse

Figure 2: *Using the Genius Ring Mouse to pick up and move objects.*

Ring Mouse

When using the ring mouse, the application is dependent on the mouse cursor inside the application window. This is an issue when the cursor is outside of the window. To solve this issue, we capture the device with Direct Input in DirectX Version 9.29.1974.

Since Unity does not specifically target an OS, Microsoft.DirectX.DirectInput is not supported. Even though Unity is C# 3.5, it is .NET 2.0 CLR compatible; therefore, a dll is made compiled with NET 2.0 CLR to access Direct Input.

Object Meshing

To achieve higher frame rates, all object meshes were imported as optimized meshes. All model collisions are handled with Unity's Convex Mesh collider with the exception of the bookshelves. The bookshelf colliders were manually created with Unity Box Colliders. The labels centered on the rows and the sorted results at the back wall were displayed as TextMesh. All models not including the interactable dinnerware objects are Kinematic objects.

Grabbing Hand

The hand controller and grabbing hand are implemented using the HandController.cs and GrabbingHand.cs from the LeapMotionCoreAsset. Modifications to both scripts have been made for the Ring Mouse implementation and the Halo effect.

The scale of the HandController GameObject is 1 in the x direction, 1.5 in the y direction, and 1 in the z direction. The initial rotation of it is 270 degrees in x direction, 180 degrees in y direction and 0 degrees in z direction. Because the Leap Motion sensor defaults to the y axis for the sensing of hands, it needs to be rotated 270 degrees in the x direction for head mounted displays and 180 degrees in the y direction to correct its view with the main camera. The hand controller is also responsible for the deselection of objects when the hand is destroyed, as well as reassigning the object when a hand is accidentally destroyed.

To find the closest object, the method Physics.OverlapSphere is used with the position between thumb and index finger reported by the Leap library and the object's grabbing distance derived in the previous section. This method returns a Collider[] with the closest objects. When the hand is destroyed and recreated, the active object is explicitly set in GrabbingHand.cs and the object will move where it is currently positioned. To force the object to move to the hand, an offset is needed to magnetically attract the object to the hand with a velocity. A velocity and an angular velocity is given to the object to move it to the correct place when the object is pinched. When the object is released, a weaker force is first applied to it to simulate a state of sliding down the hand.

Object Highlighting

When implementing the highlight of an object with Unity's Halo, the halo should be created in the editor, which can be activated with a boolean. To programmatically disable or enable the halo, the method gameObject.GetComponent("Halo") is used.

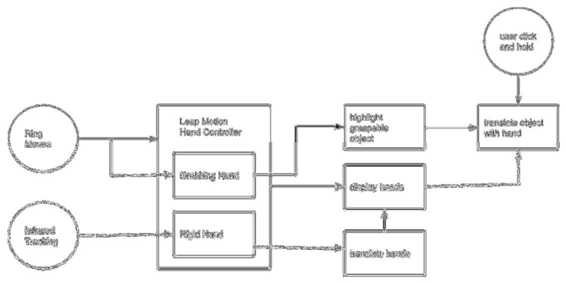

Figure 3: *User interaction data flow.*

Figure 3 shows the data flow of user interaction. The Leap Motion controller script creates and destroys hands. Each hand contains the scripts Grabbing Hand and Rigid Hand. Both Leap Motion Hand Controller and Grabbing Hand use the Ring Mouse data. The Rigid Hand script updates the hand positions with the data from Leap Motion's infrared tracking. The Grabbing Hand script manages the interaction with the hand and the grabbable object. When the user clicks and holds the button on the ring mouse, the hand grabs the nearest object within a distance of 0.2m.

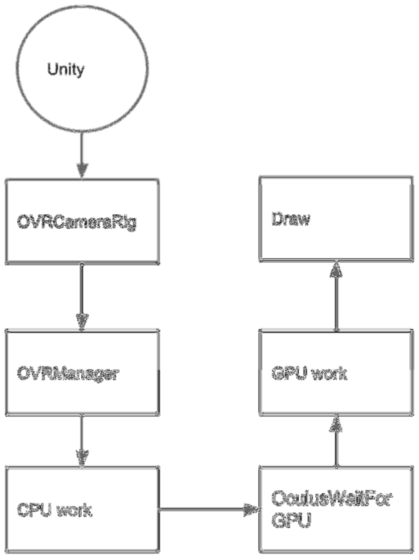

Figure 4: *Oculus Rift's rendering pipeline*

Figure 4 shows the Oculus Rift data flow. The OVRCameraRig is a head-tracked stereoscopic virtual reality camera rig. The OVRManager script is the configuration data for Oculus virtual reality. OculusWaitForGPU is called once we finish work on the CPU and we are now waiting for the GPU to do its work. After the GPU finishes it waits for the next draw window. In order to maintain a frame rate of 75fps, the CPU and GPU need to finish their work within ~13.3ms per frame. I.e., if a frame takes 15 ms to complete (CPU spends 10ms and GPU spends 7ms) it won't draw until the next draw window at ~26.6ms. With OculusWaitForGPU taking 16.6 ms.

Use cases:

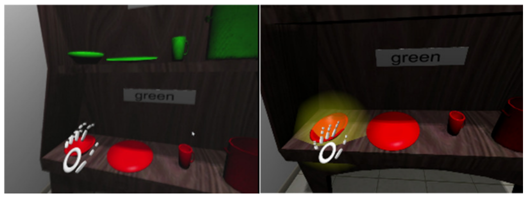

Figure 5: *Left: before grasp, Right: selected item is highlighted.*

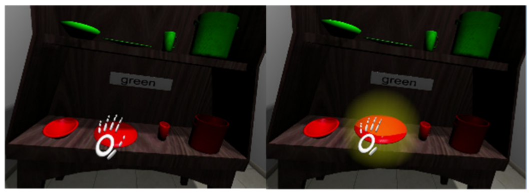

Figure 6: *Left: before grasp; Right: object grasped, magnetic force moved object to hand.*

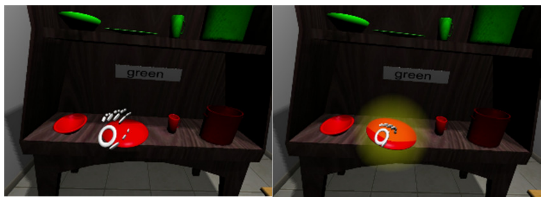

Figure 7: *Left: before grasp; Right: object grasped using innate pinch.*

We anecdotally compare three grasping techniques: innate, magnetic, and ring mouse pinching. We want to identify which technique is most consistent. Grasping a specific virtual object among others shows precision and accuracy. The ring mouse technique shows the best result for selection. As shown in Figure 5, selecting an object with a visual cue gives high precision and accuracy. The ring mouse technique gives a better performance than the magnetic pinch technique because the magnetic pinch technique requires the user to be more careful to accurately select a virtual object. When moving grasped objects, the hand will sometime lose tracking. Unlike the other two techniques, the ring mouse technique can still decide if an object should be in a grab or release state. When the hand is recreated and the grasped object is still in the grasp state, it will magnetically attract the grasped object. Releasing an object at a specific location with the ring mouse gives the best result. Similarly to grasping, the magnetic pinch technique shows a performance decrease because the grasped object will release when colliding with another object. Occasionally, the innate pinching technique will not release because the fingers are not well tracked.

Performance

To get the frame rate to the desired 75fps, optimizations were made using static and dynamic batching. We also use deferred rendering and a point light to generate soft shadows. We are currently using Oculus latest SDK 0.7.0.0-beta with NVidia driver version 355.84 that Oculus recommended [1], and the current stable version of Unity with VR is Unity 5.1.2p2 [15].

Whenever we had low framerates, the profiler shows CPU spikes from the function OculusWaitForGPU. This function call is capable of using up all available CPU time. An application targeted for Oculus VR should always render at 75fps; therefore, an approximate 13.3ms per frame can be spent between CPU and

GPU. OculusWaitForGPU is called once all CPU processes are finished and waits for the GPU to finish its work. However, if in a frame the CPU finishes processing more than ~13.3ms, OculusWaitForGPU will have to wait for the next draw at ~26.6ms. This is discussed by a Unity Representative in [2].

Discussion

One of the challenges for this project was reliably detecting the motions of grabbing and releasing objects in the virtual world.

Another challenge was programming distances and directions into the Oculus Rift to make use of its positional tracker. The distances and directions the user moves in the virtual world need to be proportional to the amount moved in the physical world. Using the positional tracker limits the user to the space physically in front of the tracking camera.

In our first pilot tests, we noticed that many users did not embrace the Oculus Rift's ability to do positional tracking. Instead of moving their head close to the object they try to grab before they reach out to it with their hand, they often do not move their head but try to grab the object with their arm stretched out far. First anecdotal feedback tells us that most users felt that the ring mouse implementation allowed for the most reliable grabbing actions, but some preferred innate pinching because it is more intuitive.

Conclusions

We presented a new suitable way to simulate hand interactions with virtual objects by utilizing a ring mouse. Although the ring mouse technique is not an exact simulation of a natural grasp, it allows the user to precisely and accurately select and grasp objects. It has also shown to be both efficient and user friendly. However, it lacks operation of both hands and grasping multiple objects, which should be considered for future work. A user study is needed for future work. In addition, the current movement space allowed with this implementation is overly constrained and would benefit from an improved form of navigation.

References

[1] Oculus Web Site: "Oculus Runtime for Windows", URL: https://developer.oculus.com/downloads/pc/0.7.0.0-beta/Oculus_Runtime_for_Windows/

[2] Unity Forum: "Major VR Performance Issue: OculusWaitForGPU running on CPU?", URL: http://forum.unity3d.com/threads/major-vr-performance-issue-oculuswaitforgpu-running-on-cpu.328442/

[3] M. Prachyabrued, C.W. Borst, "Visual Feedback for Virtual Grasping", In Proceedings of IEEE 3D User Interfaces (3DUI) 2014, pp. 19-26

[4] J. Cashion, C. Wingrave and J. J. LaViola, "Optimal 3D Selection Technique Assignment Using Real-Time Contextual Analysis," 3D User Interfaces (3DUI), 2013 IEEE Symposium on, pp. 107-110, 2013.

[5] P. Lubos, G. Bruder and F. Steinicke "Analysis of direct selection in head-mounted display environments", *3DUI*, pp.1-8 2014

[6] M. Mine, "Virtual Environments Interaction Technqiues",Technical Report TR95-018, UNC Chapel Hill Computer Science, 1995.

[7] D. A. Bowman, L. F. Hodges, "An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments", In Symposium on Interactive 3D Graphics (I3D '97), pp. 35–38. ACM Press, 1997

[8] I. Poupyrev, M. Billinghurst, S. Weghorst, T. Ichikawa, "The Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR", In ACM Symposium on User Interface Software and Technology (UIST), pp. 79–80, 1996.

[9] M. R. Mine, F. P. Brooks Jr, C. H. Sequin, "Moving objects in space: exploiting proprioception in virtual-environment interaction", In Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pp. 19–26. ACM Press/Addison-Wesley Publishing Co., 1997.

[10] G. Zachmann, A. Rettig, "Natural and Robust Interaction in Virtual Assembly Simulation", In Proceedings of the 8th ISPE International Conference on Concurrent Engineering: Research and Applications, 2001.

[11] C. W. Borst, A. P. Indugula, "Realistic virtual grasping", In Proceedings of IEEE Virtual Reality Conference, pp. 91–98, 2005.

[12] J. Cashion, C. Wingrave and J. LaViola, "Dense and Dynamic 3D Selection for Game-Based Virtual Environments," IEEE Transaction on Visualization and Computer Graphics, vol. 18, no. 4, pp. 634-642, 2012.

[13] vrwiki: "Oculus Rift Development Kit 2", URL: https://vrwiki.wikispaces.com/Oculus+Rift+Development+Kit+2#cite_note-3

[14] Leap Motion Web Site: "API Overview", URL: https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html

[15] Unity Forum: "Oculus Runtime 0.7: better performance | extended mode dropped", URL: http://forum.unity3d.com/threads/oculus-runtime-0-7-better-performance-extended-mode-dropped.347654/

Author Biographies

Jonathan Lin is an undergraduate student at UC San Diego, his major is computer science, with a focus on computer graphics. He is currently in his senior year, expecting to graduate in 2016. He did the work that led to this paper as part of his undergraduate honors thesis.

Dr. Jürgen Schulze is an Associate Research Scientist at UC San Diego's Qualcomm Institute, and an Adjunct Professor in the computer science department, where he teaches computer graphics and 3D user interfaces. He holds an M.S. degree from the University of Massachusetts (1998) and a Ph.D. from the University of Stuttgart, Germany (2003). After his graduation he spent two years as a post-doctoral researcher in the Computer Science Department at Brown University.