# Multi-Input Telepresence and Shared Environment

Andrew Korzeniewski[*]
Michigan Technological University

Richard D. Pringle II[†]
Michigan Technological University

Jeremy DeVillers[‡]
Michigan Technological University

## ABSTRACT

The shared interaction experience that we created consists of a user wearing a NVIS nVisor ST-60 Head-Mounted Display (HMD) and a user using an XBox Kinect to interact with a shared virtual environment while being able to be physically present (telepresence) in separate locations. To create this virtually immersive effect, the Kinect user has a 3D avatar or virtual representation visible to the HMD user such that it allows the two users to interact with the virtual environment. Utilizing the OpenNI API and PrimeSenseNITE Middleware, the Flexible Action and Articulated Skeleton Toolkit (FAAST) streams the joints of the skeleton that are detected with the Kinect to Vizard, using VRPN, to display on the HMD. The users cooperatively navigate a custom maze to achieve a common goal to receive a reward. We learned that immersive virtual environments are significantly more challenging to work with when relying on non-standard computer input devices, i.e. HMDs and motion trackers rather than a keyboard and mouse. We found that our implementation of telepresence needs more refinement before it can be applied to more applications.

## 1 INTRODUCTION

### 1.1 Motivation

The potential for a wide range of interactive applications gave us the idea to do this project. We were most excited in exploring game-like tasks. During our brainstorming, we came up with several ideas for interactive tasks. Some of these include plowing snow and measuring the amount plowed, pushing boxes into a pit so a user could walk across, and have one user load a projectile into a gun and have the other user shoot it at a target. Despite these good ideas, we were not comfortable with their dependability, so we chose to implement a maze-navigating task to reliably demonstrate the overall mechanics of our final design.

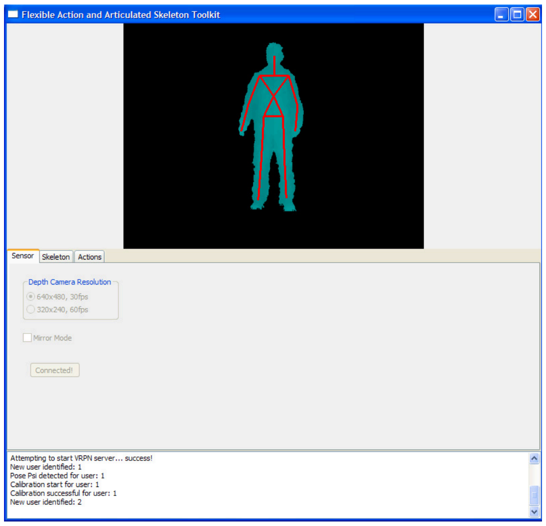
### 1.2 Resources

The two main interactive devices that we used are an Xbox Kinect and an NVIS nVisor ST-60 HMD. The HMD uses WorldViz Precision Point Tracker (PPT) [5] for position and orientation information and WorldViz Vizard [6] to render graphics. Functionality of the Kinect is provided by a combination of OpenNI [1], PrimeSenseNITE [2], and FAAST [4]. Because the Kinect user is in front of a stationary computer, this user is in a semi-fixed position to account for the limitations of the Kinect. To comply with Vizard and PPT, all computers run Windows.

## 2 IMPLEMENTATION

### 2.1 FAAST and Vizard

Body point position information was acquired using the third-party toolkit FAAST. We used VRPN to transfer data point position information between FAAST and Vizard. As a bonus for using VRPN, we obtained the ability to have remote data streams, since VRPN is a network protocol that uses TCP/IP to interface between client/server VRPN implementations. A screenshot showing the FAAST application while registering a particular user's skeletal joints is shown in Figure 1. As a proof of concept to learn how to create objects with Vizard, we created a simple cube in the world and removed the object as we became more adept at working with Vizard.



```
Attempting to start VRPN server... success!
New user identified: 1
Pose Psi detected for user: 1
Calibration start for user: 1
Calibration successful for user: 1
New user identified: 2
```

Figure 1: FAAST with the Skeleton detected.

A basic skeleton was then constructed in Vizard using the data points obtained from the Kinect. In our original design, we created a universal movement paradigm but later switched to a different movement method, both of which are discussed later in Section 2.3. The skeleton was represented by placing spheres at the data point locations in the virtual environment as shown in Figure 2. However, due to the inexactness of image processing that the Kinect does to determine joint locations, some of the joints of the skeleton of the Kinect user appear to not be completely stationary and may jump around to some degree even if the user is perfectly still. We discuss Filtering in Section 7.1 as future work to possibly resolve this problem. Using the joint information we constructed limbs to join the appropriate joints to make a semi-realistic avatar that is shown in Figure 2.

From there, we were confident that we had enough data to work on something more useful. One goal we had was to allow both the Kinect user and HMD user to interact with objects in the virtual environment. Therefore, we implemented custom methods for adding objects to the virtual environment for the users to interact with. Once custom methods for objects were implemented, we thought the next best step would be to allow the Kinect user to carry an object. We made this decision because until this point in the project we concentrated on the Kinect user being *in* the virtual environment. We believed that using the Kinect user's hands to carry a virtual object and having that object interacting with its environment was an intuitive next step.

[*]e-mail: apkorzen@mtu.edu
[†]e-mail: rdpringl@mtu.edu
[‡]e-mail: jadevill@mtu.edu

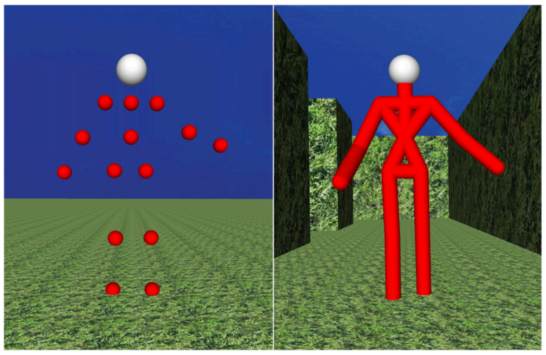


Figure 2: Vizard with the Skeleton (left) and Avatar (right).

### 2.2 Physics and its Moments

#### 2.2.1 Custom Collision Detection

In order to allow the Kinect user to pick up objects, we needed to be able to detect collisions between the Kinect user's hands and the virtual object the user wanted to pick up. Vizard has methods for using its physics engine to do collision detection but using the built in system did not provide us with a means to do what we needed. Therefore, we implemented a spherical-based collision system. We have two classes of objects: spheres and cubes. Both classes of objects use a similar implementation of radius-based spherical collision detection but the objects are categorized into two different classes simply because of the specific implementation details. Figure 3 depicts the collision detection schema.

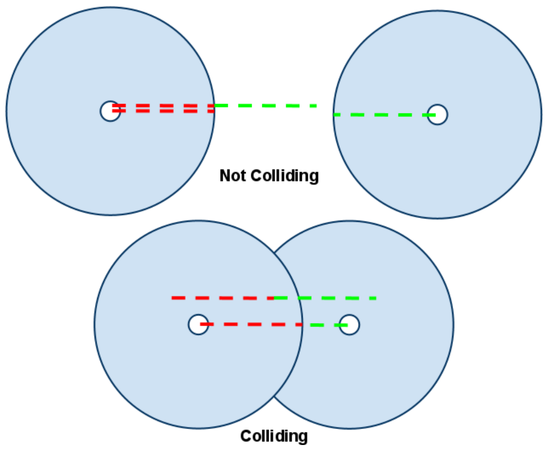


Figure 3: Collision Detection Schema.

Using this method for collision detection, if both of the Kinect user's hands are colliding with an object, then the Kinect user grabs the object (Figure 4) and the center of that object is set to the midpoint between the Kinect user's hands. This requires both of the Kinect user's hands to grab an object and if at any time both hands are not colliding with the object, the object reacts to its environment without the impact of the Kinect user's hands.

#### 2.2.2 Inertia and Gravity

After the Kinect user was able to interact with objects, it seemed natural to try and throw objects. This required velocity and acceleration information to correctly calculate projectile motion. We obtained the velocity by sampling a number of consecutive frames to determine the velocity and direction that the user is propelling an object. To add to the realism and to take advantage of the native

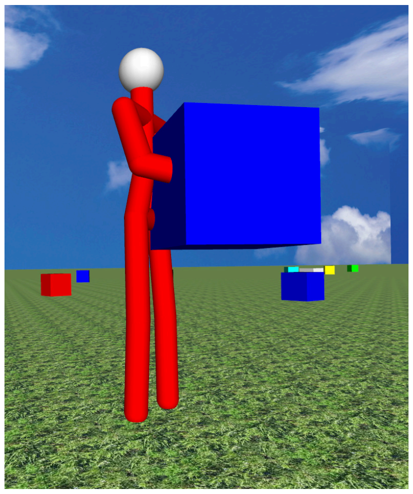


Figure 4: Kinect User Holding a Cube.

physics engine that Vizard implements, we added gravity effects to objects. Between calculating the velocity of an object when it is released and applying the forces of gravity, we achieved a somewhat realistic feel of throwing objects as shown in Figure 5. By extension, using Vizard's native physics engine allows an object to respond to other objects and its environment as shown in Figures 6 and 7.

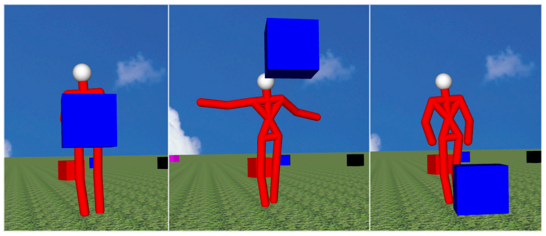


Figure 5: Throwing an object.

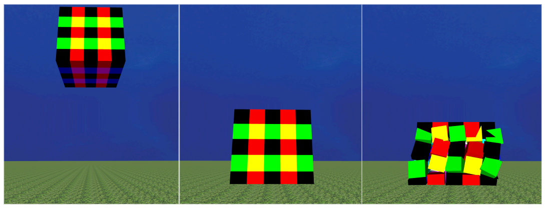


Figure 6: Cubes Falling and Starting to Bounce.

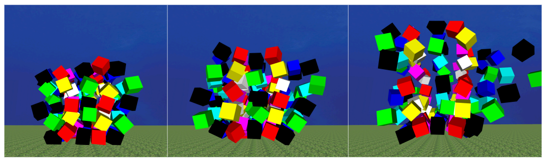


Figure 7: Cubes from Figure 6 Bouncing and Ricocheting.

#### 2.2.3 HMD and Kinect: Fatal Interaction

Since the Kinect user could interact with objects in the virtual environment, we decided to add the ability for the HMD user to keep things fair. However, the difference in details is that we only gave the HMD user one hand because the tracking system uses LED lights as data points and it seemed to jump between different sensor channels depending on the occlusion of the cameras. To that end, we allowed the HMD to grab objects in the virtual environment if the one hand was colliding with the object. In the event that both the Kinect user and the HMD user wanted to grab the same object we had to develop a method that would maintain the perception of reacting the same way that two people would interact in the event that they wanted to both grab the same object.

The method we developed was to use the three different hands as points on a triangle, then find the centroid of those points to update the origin of the object. As the users interact with the object, the origin automatically changes to reflect those interactions. Therefore, if both the Kinect user and the HMD user are not colliding with the same object, it reverts back to normal collision behavior. This allows the effect of users *giving* and *taking* objects.

### 2.3 Movement Paradigm

The first design of the movement paradigm was an absolute movement system that was designed so the Kinect user's movements and HMD user's movements in the real world were the exact same as the actions performed in the virtual world. This provided us with a good schema to test and develop our logic for the rest of our project. However, when exploring a virtual environment that is larger than the physical environment that is being used, this movement paradigm falls short.

We decided to revisit our movement paradigm and chose a relative position, vector-based movement system similar to the WASD or arrow key movement system commonly used with a computer keyboard. The farther the distance that a user moves from his origin, the faster that user will move in a vector direction using the origin as the tail of the vector and the user's position as the head of the vector. We liked this movement system better because it allowed us to navigate a larger virtual space than what was available in the physical space.

## 3 Demonstrating Telepresence

### 3.1 Cooperative User Task

We created all of this functionality so that two users could interact in a virtual environment. We chose a garden maze, as shown in Figure 8, as the task to demonstrate the features that should be available in any telepresence gaming application. It requires both users to navigate the maze from the starting location to user specific finish locations. When the goals are achieved we reward the users with a surprise. The surprise involves the maze disappearing and the cubes falling from the sky and exploding, as shown in Figures 6 and 7.

### 3.2 Virtual World Environment

The ground, sky, and maze were all textured. The boxes were colored using the basic Vizard color functionality. A simple repeating grass texture was mapped to the 100m x 100m ground. The repeating grass texture was also mapped to the walls of the maze. The height of the walls of the maze was 3m. The sky box was textured using a cube map of a blue sky with clouds. The sky was 100m x 100m placed 50m above the ground.

## 4 Goals

From small to large, we highlighted goals that we achieved to help guide us through development.

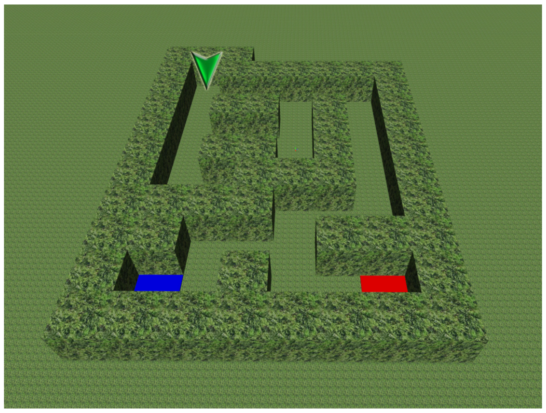


Figure 8: The Garden Maze with the Green Arrow showing the start location, the Red Square representing the Kinect finish location, and the Blue Square representing the HMD finish location.

### 4.1 Obtained Goals

We created an object in the virtual environment and used the position information of an LED tracker to define an object's position in the virtual space. We used the tracking information provided by FAAST to display objects at proper joints to construct an avatar. The tracking data was used from FAAST to update the Kinect user's avatar in real-time to allow visual confirmation of physical movement. Interaction with objects in the shared virtual environment was enabled for the HMD user and the Kinect user. To extend this interaction, we enabled the HMD user and the Kinect user to simultaneously interact with the environment at the same time. Once all of this was put together, we worked on our main goal to create a game using both the HMD user and the Kinect user to assist each other in order to explore it.

### 4.2 Unobtained Goals

We would have liked to get a display for the Kinect user that would display the avatar of the HMD user. However, extracting position information from Vizard was not achievable, nor could we write an OpenGL program within a reasonable amount of time. To achieve more immersion within the virtual environment, we would have liked to implement spatially-accurate sound effects but we chose to pursue other priorities in its place. We wanted to explore displaying the Kinect user in a third-person view so that the Kinect user could see his avatar and the HMD user's avatar in the virtual environment. However, by the time we reached this step in our project, constructing a custom VRPN server/client implementation seemed to be too difficult to finish within the time allotted, and instead we pursued other goals.

## 5 Responsibilities

The setting up, handling, and maintenance of the Kinect was handled by Richard throughout the project. Andy was in charge of the maintenance and troubleshooting of the HMD the majority of the time. Aside from the hardware handling, the responsibilities were largely shared and worked on as a team. The majority of the programming was handled by Richard as he had a lot of experience in Python. He also installed FAAST and the related Kinect drivers. The other portion of the programming was completed by Andy who knew the Vizard system and how to run PPT. Jeremy mainly worked with debugging and helped figure out what needed to be done to make the objects and scene work or appear correctly.

## 6 Conclusion

Overall, we believe that the project was a success and would recommend others to expand on our findings. We had a working demonstration that we felt was representative of all of our hard work. However, we would have liked to do more and Section 7 details those areas of interest.

### 6.1 Reflection

Generally, the project contained a healthy amount of challenging opportunities to keep our minds busy. We thought that it was a fun project. Overall, we found that it is fairly easy to create an *interesting* virtual environment. However, it is very difficult to refine the environment so that it is immersive, convincing, yet still easy to use. We all learned how to use Vizard and PPT. In addition, the use of Python helped refine our programming skills.

### 6.2 Challenges

There were several aspects of the project that were tough to work through. It was sometimes difficult to find meeting times with three people on the team, especially towards the end of the semester. We found that collision detection was fairly difficult because even though Vizard had native support for colliding-object primitives, they were not documented very well, requiring us to create custom procedures. Multiple user interaction of a shared object in a virtual environment was difficult, but necessary to implement because it is a prevalent aspect of normal interaction between individuals. Though we came up with an elegant solution for multiple users interacting with the same object, we were unsure as to whether it was the optimal solution, so it would be interesting to see what kind of research has been done on this particular topic. Movement in a confined space was also difficult, since our virtual environment was much larger than the physical space. It was unclear as to the best way to handle this type of movement, but once implemented it was even harder to perfect. Physics was also something that we found difficult to implement. Though Vizard has native physics, we still needed to design our own procedures and structures to store the related information for throwing objects and calculating velocities. It was also frustrating that the HMD tracking software, PPT, was not completely stable, for reasons that we were unable to figure out. It sometimes crashed multiple times per day as we were working. Lastly, our HMD unit was out for repairs for several weeks of the semester, so we were unable to demonstrate or test our project with the actual HMD unit. As explained above, all of these were remedied eventually, but these highlight our most significant obstacles we had to overcome.

### 6.3 What We Would Change

We discovered that designing an effective movement system for a virtual environment was much more difficult than we anticipated. We would have liked to work much more on this area and it would be a higher priority if we could do the project again. Also, utilizing more than one Kinect to track a user to avoid occlusion issues would be another priority if we were to do this project again.

## 7 Future Work

With the system that we have developed in this project, there are a number of other applications that we wanted to explore but due to various constraints were unable to attempt.

### 7.1 Filtering

With the image-based skeleton detection that is employed by the Kinect, the exact locations of particular joints for a person are not always accurate but may change each frame based on the image processing that is done. Therefore, the Kinect sometimes transmits noticeable movement for the Kinect user even if they are standing completely still. In the future, it would be very beneficial to implement some type of filtering to remedy this issue. Specifically, this would be a good application for Kalman filtering so the Kinect avatar has a smoother appearance.

### 7.2 Using Multiple Kinects or HMDs

Due to hardware constraints, task complexity was limited for the users in the virtual world because we were constrained by the quantity of the Kinects and HMD units that we had access to work with. Using just one Kinect to track a single user was problematic because it introduced occlusion-based issues that could be solved if more than one Kinect was used to detect a person. We feel that this would have increased the amount of immersion in the environment that the user felt. Another application we wanted to explore was having multiple HMDs or multiple Kinects to allow more than two users to interact in the same virtual environment. This would add to the functionality of telepresence for more than just two user tasks. In our current implementation, the Kinect user needed to rely on the HMD user for guidance in the virtual world because the Kinect user did not have any means of viewing the virtual environment. Therefore, in addition to helping immersion, using an HMD in place of or with the Kinect would allow both users the ability to explore the virtual environment independently and reliably.

### 7.3 More Tasks

We explored one of the tasks that telepresence has to offer, but in a virtual environment many tasks that would normally be unrealistic suddenly become practical. To that end, we wanted to explore other effective means for using the Kinect and HMD for either cooperative tasks or competitive tasks that utilize the users that share the same virtual environment. This project explored a maze that two users were required to navigate together as a team. In Section 1.1, we discussed other ideas for possible cooperative tasks to explore and in the future those would be something that we would like to accomplish as well. Competitive tasks that would be interesting to do would be an obstacle course or a downhill skiing application.

### 7.4 Different Methods of Movement

During this project, we investigated two different types of movement systems. One was a system that mimicked real-world locations in the virtual world and the other is a relative, vector-based method which are both discussed in Section 2.3. However, even though these systems can be helpful in completing tasks effectively, we wanted to explore other, possibly more intuitive, methods of moving around in a virtual space that is larger than a physical space. Implementing redirected walking as Razzaque et al. [3] described would be an excellent next step to do in the future. This way a user could walk naturally, more intuitively and not be required to worry about any vector-based movement schema. An alternative to that method of redirected walking could be to allow a user to walk naturally but account for the physical obstacles in the room with either virtual objects or by stopping movement *in* the virtual world and automatically restarting the movement once the user has moved to a safe, designated real-world position.

## References

[1] OpenNI Organization. OpenNI. `http://openni.org/`, Apr. 2011.

[2] PrimeSense Ltd. PrimeSense NITE Middleware. `http://www.primesense.com/?p=515/`, Apr. 2011.

[3] S. Razzaque, Z. Kohn, and M. C. Whitton. Redirected walking. In *Proc. Eurographics 2001*, pages 289–294, 2001.

[4] E. A. Suma, B. Lange, S. Rizzo, D. Krum, and M. Bolas. Flexible Action and Articulated Skeleton Toolkit (FAAST). `http://projects.ict.usc.edu/mxr/faast/`, Apr. 2011.

[5] WorldViz. Precision Point Tracker (PPT). `http://www.worldviz.com/products/ppt/index.html`, Apr. 2011.

[6] WorldViz. Vizard. `http://www.worldviz.com/products/vizard/index_b.html`, Apr. 2011.