Physics-Based Free-Hand Virtual Grasping

Ethan Lin, Emma Li, Jiasheng Shi, Edward Conte, Cooper Proctor, Una Wu

Program of Computer Graphics at Cornell University
Ithaca, New York, USA

Abstract

In current VR systems, achieving realistic and intuitive interactions between the user and virtual objects is a significant challenge due to the disconnect between the virtual and real world. We seek to overcome this challenge by creating a physics-based, free-hand grasping model in Unreal Engine to make grasping virtual objects a more natural user experience in VR. Our model leverages Meta Quest Pro's hand-tracking technology and Unreal Engine's built-in physics to allow users to realistically interact with virtual objects using their bare hands without needing controllers, voice commands, or haptic gloves. We create our own force-calculation algorithm which calculates the normal force from each contact point in the hand to the virtual object. The normal forces are used to apply an impulse to the object and to find the static or kinetic friction between the hand and the virtual object. The static and kinetic friction relationship in our model mirrors real-world physics and gives the user a more free and natural experience of grabbing objects in VR.

1 Introduction

Virtual Reality (VR) technology has seen substantial advancements in recent years, providing increasingly immersive environments replicating real-world interactions. These advancements have opened new possibilities in various fields, including gaming, training simulations, education, and remote collaboration. A critical aspect of creating a seamless and engaging VR experience is enabling users to interact with virtual objects naturally and intuitively. Effective object manipulation is not only central to the realism of the virtual environment but also crucial for user satisfaction and task performance.

Traditionally, VR systems have relied on handheld controllers or specialized VR gloves to facilitate object interaction. While these tools have proven effective, they often detract from the sense of immersion and can impose limitations on the naturalness of user interactions. Controllers, for instance, can feel cumbersome and less intuitive than using one's hands directly, while VR gloves can be expensive and complex to set up.

Recent advancements in hand-tracking technology offer a promising alternative, enabling users to interact with virtual objects using their bare hands without the need for additional hardware. This direct hand-tracking approach has the potential to significantly enhance the realism and immersion of VR experiences by allowing more natural and intuitive interactions.

There are several approaches to free-hand virtual grasping. The first approach is to use pre-defined gestures to interact with virtual objects. A well-known example of this is utilized by the Apple Vision Pro, which employs a pre-defined “pinching” gesture combined with the direction the user is looking to interact with a virtual screen as if they were using a mouse. Pre-defined gestures like “pinching” or “tapping” dominate the UI of most current VR devices that support hand-tracking. These gestures are extremely useful and simplify the user experience in standard scenarios like a home screen for VR.

However, pre-defined gestures are not adept at handling complex and realistic interactions between a user’s hand and a virtual object. In scenarios such as training simulations, where it is crucial to model hand-object interactions as realistically as possible, a more sophisticated approach is needed.

This research focuses on developing a physics-based free-hand grasping model within Unreal Engine, leveraging the advanced hand-tracking capabilities of the Meta Quest Pro. By approximating the hand as a virtual object with the ability to apply both normal and friction forces on objects, this model aims to replicate the complexities of real-world object handling while minimizing computational costs. We hope the accuracy of our model will enhance user experience and immersion in virtual environments.

2 Relevant Literature

We relied most heavily on Holl et al. [1]’s paper for ideas for our model; the implementation in this paper is very similar to our implementation.

Both implementations determine a contact point by defining a small threshold distance for the virtual objects and creating “a contact point when a point on the hand model's surface gets closer to a virtual object than this threshold.” The main difference here is that the contact boundary in this paper is **outside** the virtual object's physical mesh, whereas the contact boundary is the physical mesh for our model. This gives our model a bit of a downside in terms of realisticness because our virtual objects are penetrable (which is not like real life) because of the way our force detection algorithm works. We both calculate the applied force to the object for every contact point on the hand in relatively the same way: the more the amount of finger bone/joint that is within the collision mesh of the object, the greater the magnitude of the applied force to the object. The major difference between both implementations is that the paper’s model heavily

utilizes friction, but our model does not incorporate friction yet. Friction allows the model in the paper to have states of grabbing: static and dynamic. Because we haven't implemented friction yet, our model only has the static state, which is that it is grabbing or not grabbing. This paper uses the dynamic state to update the location of the contact points. Our contact points are always fixed until the object is "ungrabbed."

The "Grip Strength" section of Pollard and Zordan [2]'s paper brought up an interesting point that helped us create our force-calculating algorithm. Because the implementation in this paper requires calculating joint torques to determine the grabbing state of the hand, the paper brought up an interesting situation when the user changes their applied force to a virtual object **after** they grab it. Because the hand and fingers are stationary, the velocity and acceleration values would be 0, so the original way of calculating the torque would not be viable, and a new way of determining whether the user was grabbing the object or not had to be found. To fix this issue, the section discusses how "clos[ing] the hand further will result in greater force applied to the object." Using this as an inspiration, we decided to base our entire force algorithm on the idea that the more the fingers closed inside an object, the greater the force applied to the object.

We followed a similar approach to calculate the grabbing force for an object, which is the method described in Chessa et al. [3]'s paper. The procedure detailed in the paper first finds the distance between the contact point on the object surface to the current location of the intersecting finger bone/joint. Then the distance is divided by the maximum distance between two points in the object and some other arithmetic. Our force calculator algorithm follows the same logic of finding the distance between the collision location and intersecting finger bone/joint with some differences in the specific arithmetic used.

Even though Schaffer et al. [4]'s paper is about creating customized hand gestures in VR, there is some important overlap with our project. We followed the implementation of switching between accepting and non-accepting states. In the paper, we switch between recognized customized gestures and unrecognized gestures. For our project, this would mean switching between a grabbing state provided that certain conditions were met and switching back to the ungrabbing state when any of the conditions failed.

While Nasim and Kim [5]'s paper uses a physics-based approach to achieve hand grasping in VR, it uses a 22-IMU-sensor glove, which is different from our approach. Nonetheless, we looked at their implementation for sources of inspiration for what we could do when starting to make our model. The IMU-loaded glove allows for very accurate tracking of linear and angular acceleration/velocity of each of the finger joints. These values are then used to calculate the torque applied to virtual objects, which is inputted into the physics simulation system. We were originally going to calculate the force applied to virtual objects using linear and angular acceleration/velocity; however, obtaining accurate measurements for those values in Unreal and given the headset cameras became too difficult and unpredictable.

3 Model Summary

Our model uses colliders to detect collisions at specific points on the hand. When a collision occurs, a force is applied to the object at the collision point where the magnitude is determined by the velocity of the collider and the depth of the collider into the object. A “grab” state occurs when perpendicular forces are applied to an object such that there is approximately zero net force. When in the grab state, the object's physics is disabled and the object attaches to the hand until the conditions for the grab state no longer hold.

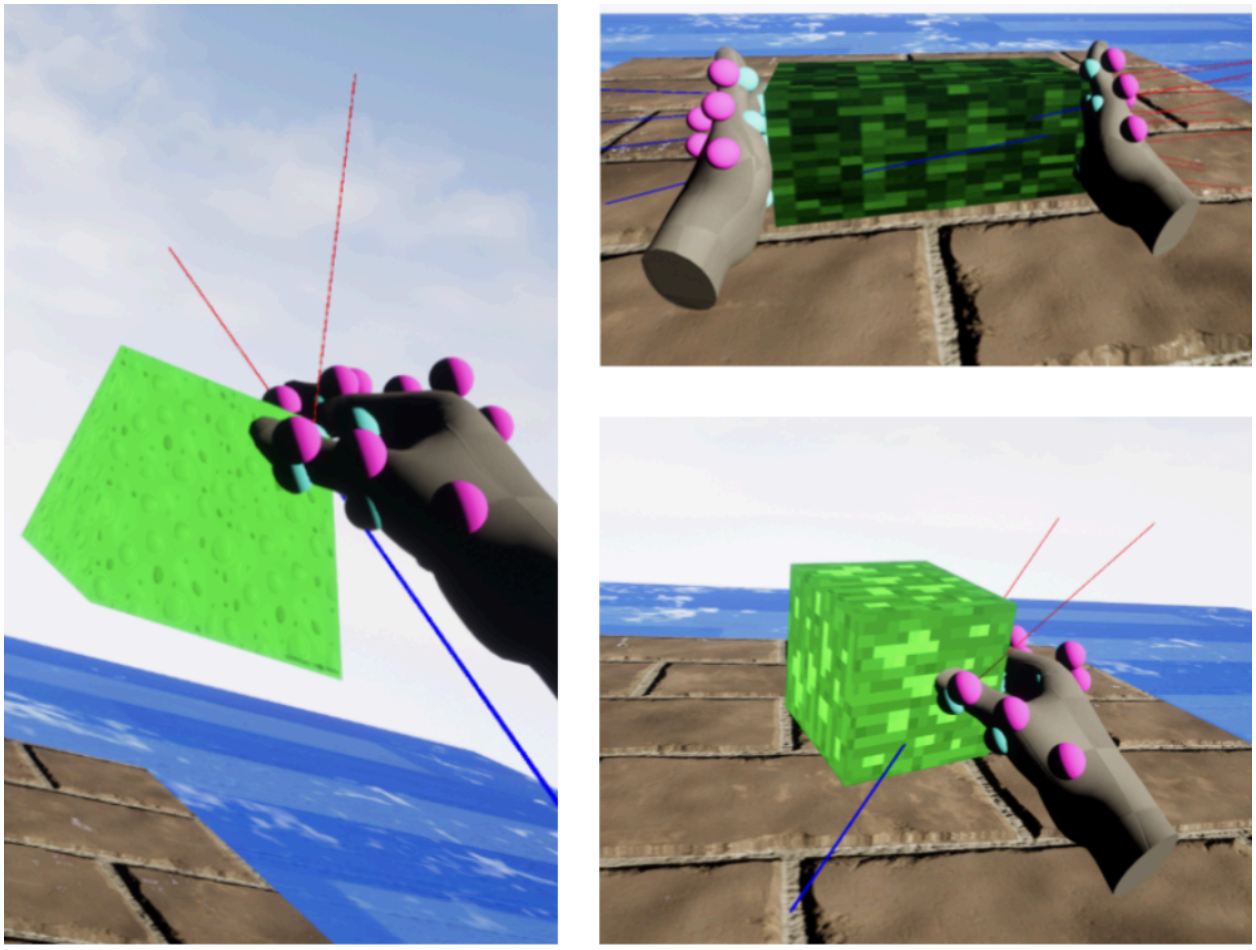

Figure 1: Depiction of various types of “grabs” our model supports. The left image is grabbing an object with two fingers. The top right shows grabbing an object with two hands. The bottom right displays grabbing an object with one hand. The blue lines indicate the direction of force for each collision point. The red lines are in the opposite direction to extend the vector and make it more visible.

4 Implementation Details

4.1 Colliders

A collider is a spherical object connected to a joint in the hand. There are 14 colliders on the inside of the hand and 14 colliders on the outside of the hand. Each pair of colliders represents a specific joint on the hand which can be seen in Figure 2a. The position and rotation of each collider are determined by the built-in hand tracking system, which tracks using multiple spatial cameras on the headset. The goal of each collider is to detect collisions with physical objects so that the model can apply interactive forces based on a specific location on the hand.

In an idealized world, we would have a collider for every molecule in the hand to pinpoint the exact collision location(s) and then apply forces with 100% accuracy. However, while the accuracy would increase, so would the computation time. Thus, there is some medium that we must find where we have high accuracy and the increased computation time is unnoticeable. In the future, we will try to detach the colliders from each joint and include many more of them.

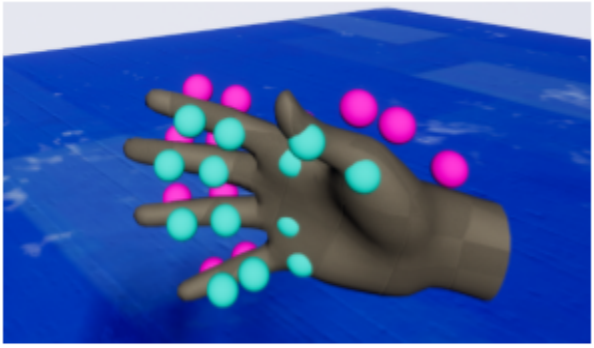

Figure 2a: Image of our **current** implementation in Unreal Engine. The 14 front colliders are colored blue. The 14 backs colliders are colored pink. Each pair of colliders is associated with a specific joint in the hand.

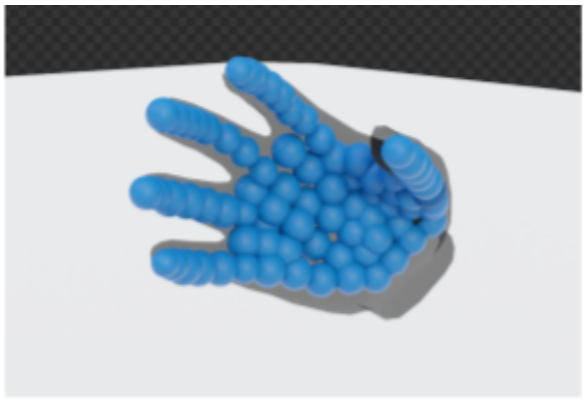

Figure 2b: Depiction of our **goal** implementation. There are many blue colliders that are independent of the joints but each have their own force directions and can detect collisions on their own.

4.2 Force Direction

The current implementation of force direction utilizes Unreal Engine's built-in *getVector* function that is applied to each joint. As seen in Figure 3, each collider has an associated vector perpendicular to the joint in the hand. The front colliders have force vectors that are normal to the palm and the back colliders have vectors that are normal to the back of the hand.

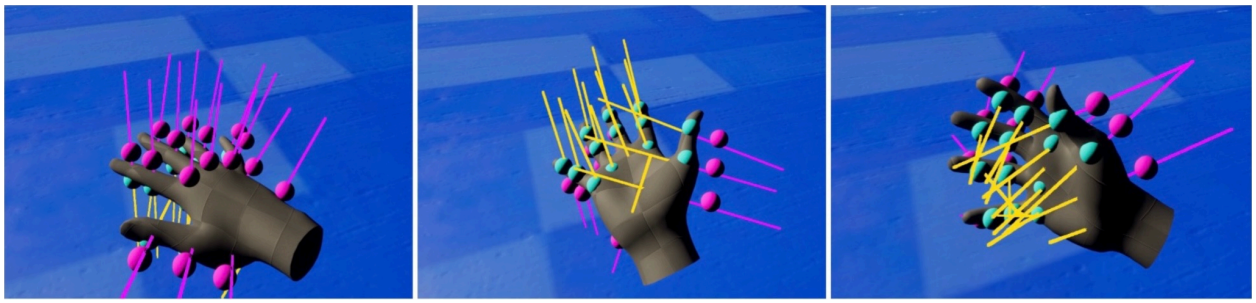

Figure 3: Depiction of the force vectors for each joint. Front collider vectors are yellow and back collider vectors are pink. These directions are the direction that force is applied when a collision between that collider and an object occurs.

There are several issues with this method of approximating force direction. One obvious example occurs when a person wants to push an object with the front of their finger. Instead of pushing the object forward, this implementation would apply a sideways force.

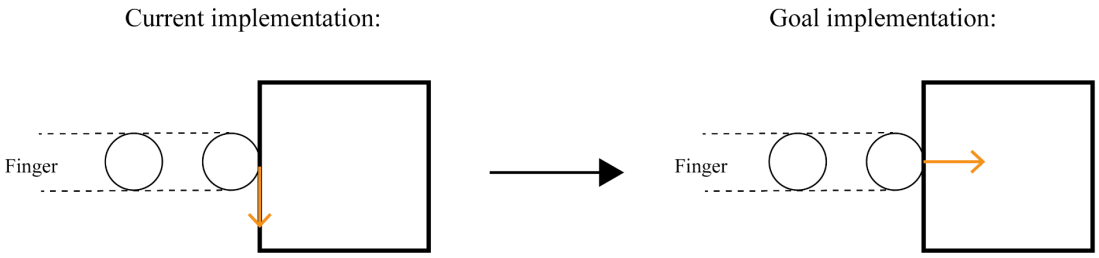

Figure 4: Depiction of force direction limitations for pushing. In our current implementation, if the user tries to push with the front of their finger it will just apply the force in the normal direction to the finger joint, in this case downwards. Ideally, we would like the direction of force to be independent of the joint itself and instead based on the joint's movement.

While the current force direction derivation is inaccurate, it was useful as a proof of concept for our model. In the future, we will explore other ways to accurately calculate the direction of the force at the time of collision independent of the joint.

4.3 Force Magnitude

From physics, we know that force equals mass times acceleration. Unfortunately, due to the constraints of Unreal Engine, we do not have access to the acceleration of each joint. While we could try to store and calculate the acceleration ourselves, we determined that using the **velocity** of each joint would be sufficient for our future model. Velocity not in our current model.

Additionally, as we are in a virtual environment, the hand can pass through any virtual object. To counteract this problem, we add **depth** as another factor to calculate the magnitude of force. The magnitude should increase the farther the joint is inside of the object.

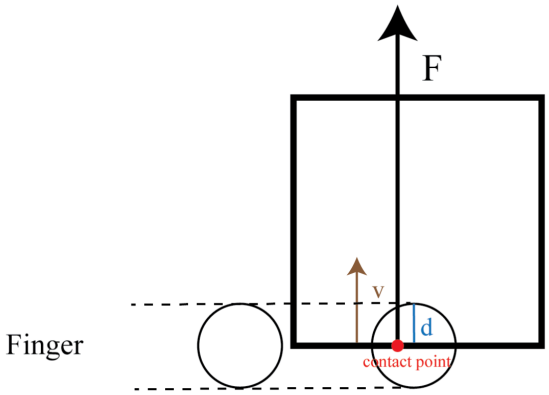

Figure 4: Diagram of how a finger applies an impulse to an object. The force is applied at the contact point and the magnitude of the force is derived from the following equation. $F = \alpha v + \beta d$ where v is the finger velocity and d is the depth of the finger inside the object, and α and β are constants set to 0 and 1.6.

4.4 Depth Calculation

The ideal depth would be the distance from the point of contact to the location of the collider's farthest point inside of the object. While other papers use Unity's PhysX package which includes collision point detection functionality, this functionality is not available to consumers on Unreal Engine. What Unreal Engine does have available is the function *Get Closest Point on Collision*, which takes in a primitive object and a point and outputs the point on the body that is closest to the input point.

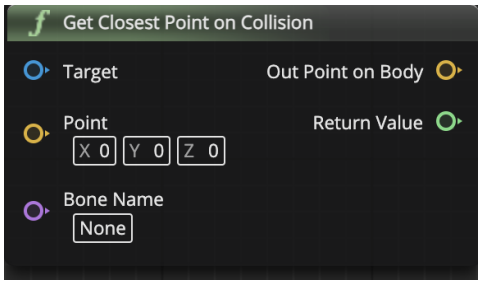

Figure 5: The block in Unreal Engine for the *Get Closest Point on Collision*. "Bone Name" and "Return Value" are not useful for the model.

We make use of this function to loosely approximate the depth. The implementation is shown below. The object we are grabbing, with center P_{obj}, is the *Target* input, the center of our collider denoted P_c, is the *Point* input, and the output *Out Point on Body* is denoted as P_{close} which is the closest point on the object. The distance *d* is calculated based on various conditions described in the figures below.

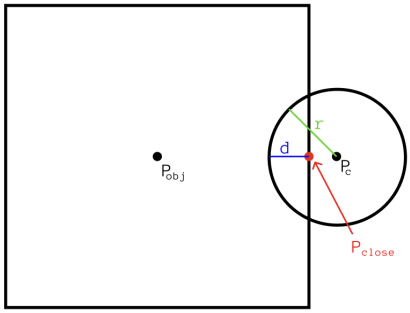

Figure 6a: Collision point is outside of the object. This condition occurs when there is a $||P_c - P_{obj}|| - r < ||P_{close} - P_{obj}||$ collision, , and when $||P_c - P_{obj}|| > ||P_{close} - P_{obj}||$. When either condition is fulfilled, a collision occurs, and $d = r - ||P_c - P_{close}||$.

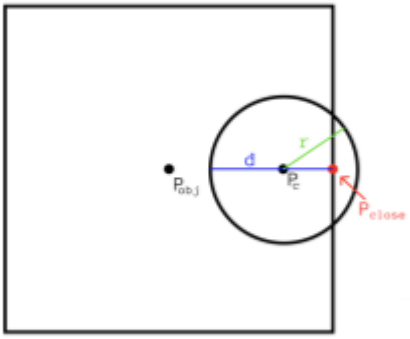

Figure 6b: Collision point is inside of the object, but the collider is not completely inside. This condition occurs when there is a collision, $||P_c - P_{obj}|| - r < ||P_{close} - P_{obj}||$ and $||P_c - P_{obj}|| - r < ||P_{close} - P_{obj}||$. In this case, $d = r + ||P_c - P_{close}||$.

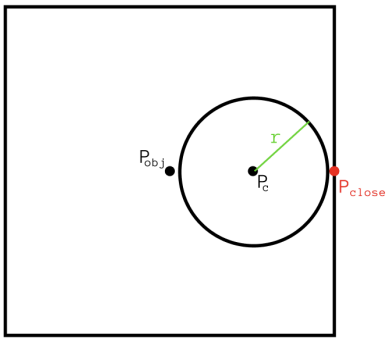

Figure 6c: The collider is $||P_{close} - P_c|| > r$ entirely inside the object when there is a collision and This scenario is what we want to avoid.

.

It is important to note that this method does not work on more complex objects like a donut, where the center of the object is not physically inside of the object. However, for our initial model with simpler objects, it is sufficient.

4.5 Grabbing Condition

In an ideal scenario, a grab occurs when the net force on the object is zero and there are intersections with the hand and the object. However, due to the various approximations in our model, this ideal scenario is virtually impossible. As a result, our grab condition has looser constraints. While it varies based on different grab circumstances (one-hand, two-hand, pinch, etc.), the general criteria to determine if a grab is present on an object at any time are:

1. The number of colliders in contact with the object on hand is at least two and at least one collider's normal vector opposes at least one other's.

2. The depth of each collider into the object is satisfactory to apply a force large enough to overcome and cancel any external forces.

Our current implementation relies on several hardcoded constraints that take away from the physics-based nature of our method. However, as we continue to improve the accuracy of other parts of our implementation the goal is to have our grab condition based on the following diagram and equation.

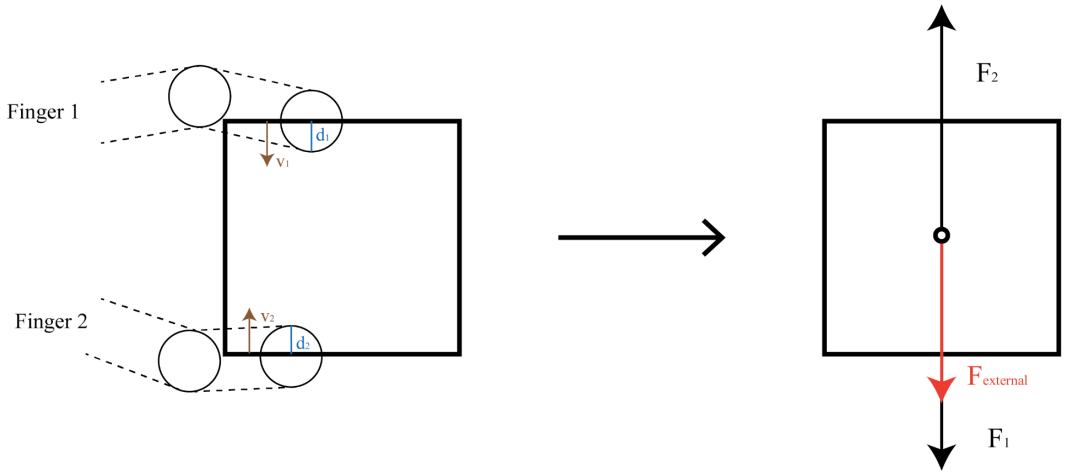

Figure 7: Depiction of when a user grabs an object where $F_1 = \alpha v_1 + \beta d_1, F_2 = \alpha v_2 + \beta d_2$. If $F_1 + F_2 + F_{external} < T_{threshold}$, then the object is put into a grab state. If not, then no grab occurs.

4.6 Non-Grabbing Interactions

When the grab conditions are not satisfied, our model accurately applies forces from the hand onto objects. Forces applied by contact points on the hand will push the object. The direction of the push is based on the force direction described above.

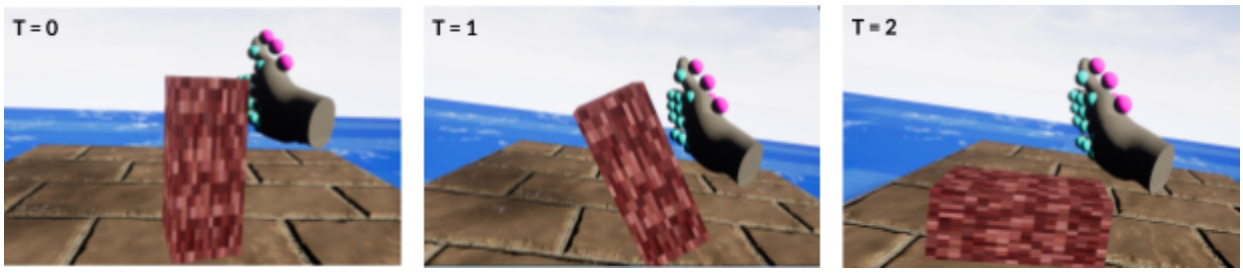

Figure 8: Depiction of knocking an object over with the palm of your hand.

4.7 Friction

While our model does not currently implement friction, the **depth** variable we computed above can be used to approximate both static and dynamic friction. In the real world, when you squeeze something in your hand, the soft padding of your hand compresses which is an indication of the force applied to an object. This force is proportional to the friction force. Thus, in virtual reality, the depth a collider is inside of an object can represent this compression which can then be used to estimate the friction force.

Including the friction force improves the accuracy of the model significantly. It introduces "slipping" states that our current model cannot account for in which the friction force is not strong enough to oppose other forces (Figure 9).

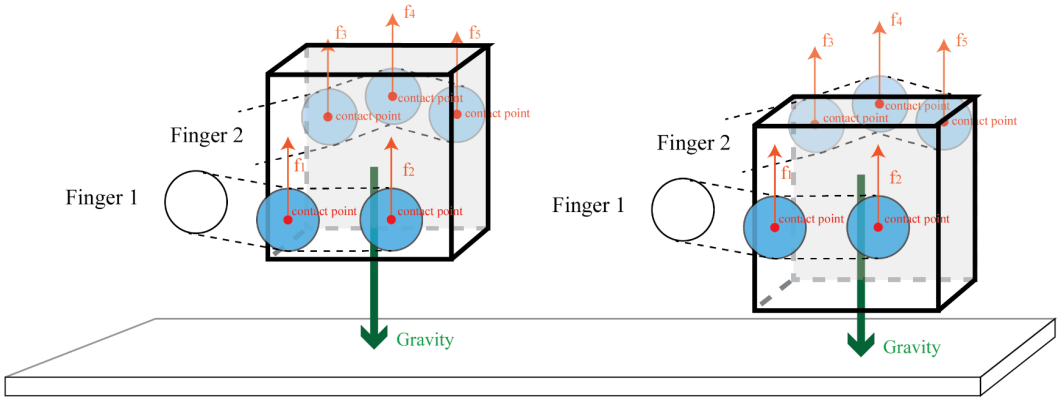

Figure 9: Depiction of the "slipping" phenomenon. In the left model, the static friction force generated from the two fingers is sufficient to withstand the force of

gravity. In the right model, the pressure is released and the object starts to move. The sum of the dynamic friction force is not enough to offset the force of gravity so the object slips through the fingers.

There are many other more complex friction-based interactions that we hope to take into account when we add friction forces to our model. Figure 10 highlights a scenario in which friction force is necessary to determine whether an object can be grabbed. Figure 11 displays another example of "sliding" an object across a surface using friction.

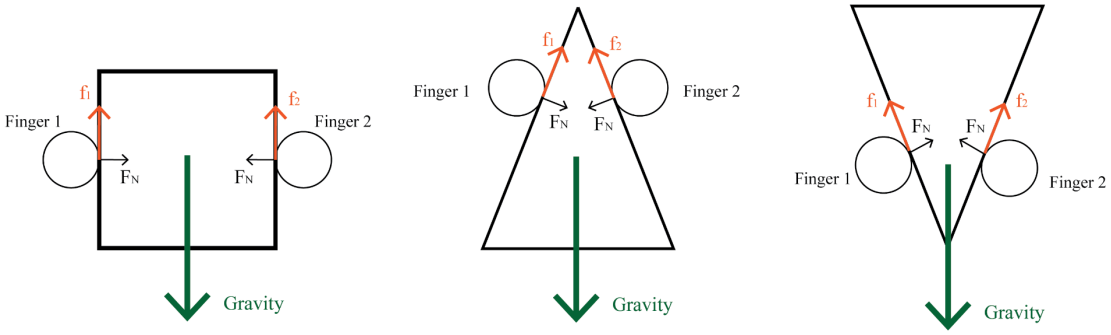

Figure 10: In each of the three scenarios, the friction force is the force that opposes gravity and enables grabbing. Assume each of the objects has the same mass. For the square (left), the normal force is horizontal and negligible with respect to gravity. For the cone (middle), the normal force is downwards, meaning the grab must be stronger to increase the upward friction force. For the upside-down cone, the friction force does not need to be as great as the normal force to offset gravity to some degree. Our current frictionless model does not handle these intricacies.

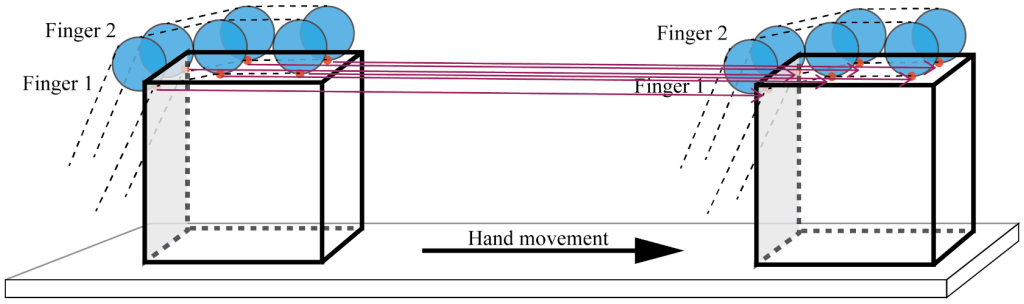

Figure 11: Depiction of "sliding" example. The downwards normal force from a user's hand creates a static friction force between the hand and object. When the hand moves, if this friction force is strong enough the object moves with the hand.

5 Conclusion

Our physics-based, free-hand grasping model in Unreal Engine accurately and intuitively simulates users' interactions with virtual objects. The implementation is centered around our force-calculation algorithm, which finds the magnitude of the normal force applied to virtual objects from each finger joint and phalanx bone in the hand. We attach a collider sphere to each of these joints and bones. By utilizing Meta Quest Pro's hand-tracking system, we can locate the coordinate points of each collider sphere in Unreal's world space. The collider spheres' world location helps determine where a hand collision is detected on a virtual object's surface.

The collision points and depth values of each collider sphere inside the virtual object are then fed into our force-calculation algorithm, which calculates the normal force applied to the virtual object(s) from each collider sphere. If multiple normal forces oppose each other and balance out on different sides of a virtual object, then the gesture qualifies as a grip. The strength of this grip is determined by the depth detection on each collider in contact.

If the depth of the grip passes a set threshold, the normal forces are used to calculate the static friction for each contact point. Static friction would then be applied, allowing the virtual object to stay firmly in the user's grasp as long as the user applies enough force.

On the other hand, if the depth of the grab was detected to be too shallow, kinetic frictional force would be applied instead, causing the virtual object to slip out of the hand at a rate slower than gravity. Since our force-calculation algorithm is run once for every frame in Unreal, there is an easy transition between the kinetic state and static state, adding nuance and making the user experience more intuitive and natural, aligning with how humans interact with real objects.

References

[1] M. Holl, M. Oberweger, C. Arth, and V. Lepetit, “Efficient Physics-Based Implementation for Realistic Hand-Object Interaction in Virtual Reality,” in *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, Reutlingen: IEEE, Mar. 2018, pp. 175–182. doi: 10.1109/VR.2018.8448284.

[2] N. S. Pollard and V. B. Zordan, “Physically based grasping control from example,” in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Los Angeles California: ACM, Jul. 2005, pp. 311–318. doi: 10.1145/1073368.1073413.

[3] M. Chessa, G. Maiello, L. K. Klein, V. C. Paulun, and F. Solari, “Grasping objects in immersive Virtual Reality,” in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, Osaka, Japan: IEEE, Mar. 2019, pp. 1749–1754. doi: 10.1109/VR.2019.8798155.

[4] A. Schäfer, G. Reis, and D. Stricker, “The Gesture Authoring Space: Authoring Customised Hand Gestures for Grasping Virtual Objects in Immersive Virtual Environments,” in *Mensch und Computer 2022*, Darmstadt Germany: ACM, Sep. 2022, pp. 85–95. doi: 10.1145/3543758.3543766.

[5] K. Nasim and Y. J. Kim, “Physics-based Interactive Virtual Grasping,” in *HCI Korea 2016*, The HCI Society of Korea, Jan. 2016, pp. 114–120. doi: 10.17210/hcik.2016.01.114.