



AutogrASPing pose of virtual hand model using the Signed Distance Field real-time sampling with fine-tuning

Marcin Puchalski 
Jan Dlugosz University
Faculty of Science & Technology
Armii Krajowej 13/15
42-200 Częstochowa, Poland
m.puchalski@ujd.edu.pl

Bożena Woźna-Szcześniak 
Jan Dlugosz University
Faculty of Science & Technology
Armii Krajowej 13/15
42-200 Częstochowa, Poland
b.wozna@ujd.edu.pl

ABSTRACT

Virtual hands have a wide range of applications, including education, medical simulation, training, animation, and gaming. In education and training, they can be used to teach complex procedures or simulate realistic scenarios. This extends to medical training and therapy to simulate real-life surgical procedures and physical rehabilitation exercises. In animation, they can be used to generate believable pre-computed or real-time hand poses and grasping animations. In games, they can be used to control virtual objects and perform actions such as shooting a gun or throwing a ball. In consumer-grade VR setups, virtual hand manipulation is usually approximated by employing controller button states, which can result in unnatural final hand positions. One solution to this problem is the use of pre-recorded hand poses or auto-grasping using physics-based collision detection. However, this approach has limitations, such as not taking into account non-convex parts of objects, and can have a significant impact on performance. In this paper, we propose a new approach that utilizes a snapshot of the Signed Distance Field (SDF) of the area below the user's hand during the grab action. By sampling this 3D matrix during the finger-bending phase, we obtain information about the distance of each finger part to the object surface. Comparing our solution to those based on discrete collision detection shows better visual results and significantly less computational impact.

Keywords

Grasping; Virtual Reality; Visualization; Interaction; Animation

1 INTRODUCTION

The Virtual Reality (VR) technology has shown significant potential in studying disabilities, injuries, and rehabilitation. It can be used to help patients with motor impairments or neurological disorders regain functions and improve their quality of life [COC22]. By providing a virtual representation of patients' hands, therapists can design tailored rehabilitation programs that allow patients to practice movements and exercises in a safe and controlled environment [IWL⁺22, FSY⁺19].

VR technology has demonstrated great promise for use in education. One study [ŻCJ18] shows how a prototype application can be used to teach mathematics, while the feedback received reveals new potential research problems and, for example, the importance of virtual hands.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Visualizing virtual hands in VR has a significant impact on how users perceive the overall experience [JYM⁺20]. Studies [AHTL16, DMF⁺19, LKRI20] have shown that, while the resemblance of human hands is not important for achieving a sense of agency (the quality of tracking is more important), it is crucial for achieving a sense of ownership.

Research [EWB20] has shown that even with passive haptic feedback, low-cost visualization of virtual hands and snapping them to predefined positions, rotations, and poses on manipulated objects significantly increase presence and user experience. Another study [KSF⁺18] has revealed that introducing virtual hands while using a keyboard paired with a VR headset raises typing speed.

The recent study [CLL22] has demonstrated that providing feedback using a virtual hand and a virtual targeted object in a virtual environment produces kinematic patterns similar to those observed in physical environments.

To prevent the interpenetration of two dynamic objects moving in space (such as a virtual hand and the object one interacts with), the term God-object was introduced [ORC07]. It can be generalized to individual

phalanges [JSF12]. The solution assumes that the visual representation of the hand pursues the target which is the position of the user's hand, as read by any sensors while maintaining collision detection with other objects and taking into account constraints between objects.

Another way to increase the realism of a VR experience is to depict the grasping of virtual objects in a way that appears more lifelike. One effective technique for achieving this is to use a realistic hand pose which shows the hand holding the object as it would be in the real world [LC21].

Earlier research [BI05] described a spring model in which the finger parts and the base of a virtual hand follow the movement of tracked hand parts. This model includes collision detection, which prevents the spring hand from penetrating grasped objects. It can also produce forces and torques for a physically realistic response to the grasped objects. Authors assumed that visual feedback, due to the psychological phenomenon of visual capture [WW80], which is not an exact representation of the real hand configuration is less distracting to users than an exact representation that exhibits the visual interpenetration artifacts. This was confirmed by the later study [CNS⁺19], where the more general term of outer hand (OH) was used instead of spring hand.

Other research [LGAMB06] proposes, in addition to a rigid model for dynamic simulation, using a deformable model to resolve local contact with friction and surface deformation of fingertips.

These types of auto-grasping solutions are often referred to as hybrid solutions with physics hands in VR and are still widely researched with different approaches, for example, using machine-learning [TWMZ19]. In the latter by using discrete collision detection, the preprocessing algorithm is sampling multi-dimensional grasp space using random configurations of hand poses. This approach requires offline preprocessing for each object but produces plausible online grasp poses after initial user-guided movement and pre-grasp pose from tracked hand.

The auto-grasping problem also extends to other fields including robotics where it is important to properly position and move the robotic arm so that it can achieve a stable grip and avoid collisions with the environment.

In [BCO⁺21] authors propose a Volumetric Grasp Network (VGN) - a 3D convolutional neural network that can detect the pose of a grasp in real-time with 6 degrees of freedom (DOF). The network takes in a Truncated Signed Distance Function representation of a scene and produces a volume of the same spatial resolution as the output. Each cell in the volume contains information about the predicted quality, orientation, and width of a grasp that would be executed at the center of the voxel. The network has been trained

on a synthetic grasping dataset created using physics simulation.

A more recent paper [TWH⁺22] proposes a method of synthesis of grasping poses with a machine learning-enabled gradient method using dense 256^3 precomputed Signed Distance Field (SDF) of the grasped object as an input. Although the proposed method synthesizes stable poses with a high contact surface, it is too slow for online usage.

Another recent paper [SHX⁺22] proposes using a machine learning approach with novel sampling of the Voronoi diagram between two close 3D geometric objects. Although the resulting method yields grips with high success rate, it is still prone to unnatural final poses.

In many cases VR environments are implemented using game engines such as Unity [Uni05] with commercial autograsping libraries ([Clo20], [Ear20]). For performance reasons, these libraries rely on a method using discrete physics collision and overlap detection based on approximating the physical collision of each grasped object using one or many simple shapes or proxy convex meshes. For more complex concave objects, for example, introducing holes, it is necessary to decompose their surface into f.e. a set of convex shapes [Uni16]. This preprocessing step is required for each interactive object in the virtual environment. Due to its offline nature, this method is not suitable for animated skinned meshes.

Taking all of this into consideration, we ask whether it is possible to skip the preprocessing stages and use an online method that will work with animated skinned meshes while still working in a real-time environment.

In this paper, we propose an approach to autograsping that, unlike existing methods that use discrete physical collision and overlap detection, uses a snapshot of the SDF of the grasped object. In addition, unlike other methods using the SDF, we propose to use only a snapshot of a closed region under the virtual hand. We compare our method with existing methods based on discrete physical collision and overlap detection. We also indicate possible future work that can be developed based on our approach.

The paper is organized as follows. In Section 2, we define the research problem and existing physics-based method; in Section 3, we describe the proposed method and its implementation; and in Section 4, we report on the simulation results. Section 5 discusses possible limitations of proposed method and current implementation. Finally, in Section 6 we conclude the paper.

2 BACKGROUND

2.1 Visual representation of hands

Several software platforms are commonly used to create VR experiences, including Unity [Uni05] and Un-

real Engine. Virtual hand presence is widely implemented in most of them using popular VR frameworks like SteamVR [Val20] and OculusSDK [Met22b].

Virtual hands are displayed instead of or along with controller models. The movement of the controllers is translated into the approximate movement of the hand relative to the position and orientation of the controller in question. The pose of the virtual hand when the controller itself is also displayed is set to resemble the pose of the hand wrapping around the given controller. However, in order to increase the immersion, the display of the controller is usually omitted while using the display of the virtual hand in a neutral pose, such as an open palm. This is somewhat counter-intuitive as the user's hand is clamped around the controller. After a while, the impression of detachment from the experience blurs, but VR hardware manufacturers are making attempts to create controllers that would solve this problem. An example of this is the Valve Index Controller [Val19], which, with the help of a suitable hand strap, allows the user to relax and open his hand.

2.2 Physics Based Hands

Physics-based hands are used in VR applications to prevent virtual hands from intersecting with virtual objects [PZ05]. These hand models consist of a visual model that is visible to the user, and a physics model that is made up of simple 3D shapes (such as capsules and boxes) implemented into the physics engine. The physics model follows the movements of the tracked hand (i.e. controller). Simulated as a non-kinematic object it is not penetrating static objects while exerting forces on other dynamic objects upon collision.

2.3 Hand poses

Visualization of the selected hand pose is done with the help of a rigged skinned mesh hand model. The rig usually consists of 3-4 bones for each finger (one for each phalanx and metacarpal bone) and at least one bone for the base of the hand. As for the movement proximal phalanges joints' have 2DOF, while intermediate and distal phalanges have 1DOF.

Commonly used VR libraries [Val20, Met22b] also allow us to customize the appearance and animating poses of the virtual hand. These poses must be predefined for a given 3D hand model and its skeleton. The defined poses most often include closed and open hand poses, between which one blends the hand configuration. For example, depending on the state readings of the controller's buttons and of touchpads, we simulate the user's hand poses such as fist clenching, finger pointing, and palm waving. These libraries also allow for the creation of predefined poses used when detecting the action of grasping various objects. However, these poses must be predefined for each

object with similar shapes and sizes (e.g., a sphere and an apple). It involves setting the hand model in the target position and orientation relative to the object and then adjusting the rotation of the individual finger bones to reflect the grasping pose. In some cases, we can achieve mirror poses for the left and right hands through symmetry.

The OculusSDK library [Met22a] provides a tool for recording predefined poses for grasping objects using hand tracking, which has been implemented in the Meta Quest software. Thanks to it developers can quickly define natural hand configuration for the grasping pose of selected objects without the need for their manual rig setup.

Several possible grasping positions and poses can be defined for a single object but in each case, the entire process of manual posing must be repeated. Further, when grasping a virtual object in runtime, the hand pose is adjusted, with the position and orientation of the object snapped to the hand to match the defined pose (e.g., the handle of a cup should wrap around the index finger) or the hand to the object when the object is static or attached to environmental elements (e.g., a door handle). Snapping a dynamic object to the hand may result in the abrupt movement of other objects in clutter due to the immediate change in position and orientation of the grasped object.

2.4 Virtual hand auto-grasping

Auto-grasping is a method that allows the automation of grasping tasks, eliminating the need for manual preparation of specific poses for different types of objects in a variety of grasping configurations. It can heavily speed up the creation of predefined grasping poses and, more importantly, enable the online generation of the poses. It also means that virtual objects can be grasped from any position, eliminating the need to align them with predetermined positions and orientations, which can be cumbersome in cluttered environments.

The auto-grasping technique can be found in commercial toolkits for Unity [Uni05] such as Auto-Hand [Ear20] and HurricaneVR [Clo20]. These implementations require specifying only two hand configurations: an open, slightly exaggerated pose (like with muscles of the back of the hand clenched) and a fully closed one - opposite from the former one (like a clenched fist).

To enable the auto-grasping of a selected virtual object, it is necessary to define one or more colliders for the grasped object. These colliders can be simple shapes such as spheres, cubes, or proxy convex mesh colliders. It is not possible to use a convex mesh collider for skinned meshes, like those of animated characters.

Algorithm 1: Physics-based auto-grasping pose estimation

```

Input : fingers list of finger objects. Each object
          contains Squish property taking values 0..1
          and is responsible for bending a given
          finger from open to closed pose, and Tip
          property referencing to finger's tip object
           $\Delta\alpha$  step from open to closed pose in
          normalized time
           $r_{tip}$  - radius of the area around fingertip in game units,
          typically in meters
    foreach finger in fingers do
      for  $\alpha \leftarrow 0$  to 1 step  $\Delta\alpha$  do
        // bend current finger to
        //  $\alpha$  value
        finger.Squish  $\leftarrow \alpha$ ;
        // Check if tip overlaps the
        // grasped object
        if OverlapSphere(finger.Tip.Position,  $r_{tip}$ )
          then
            break
          end
        end
      end
    end
  
```

Additionally, single convex mesh colliders do not accurately reflect the details of virtual objects, such as cavities and holes.

Algorithm 1 illustrates the simplified process of the grasping phase of the virtual object within the hand range using physics-based method. During this phase, an interactive blending between the open and closed poses occurs for each finger. This blending is mapped to $\alpha \in [0, 1]$, where 0 means open pose, and 1 means closed pose. We change this value by a small step $\Delta\alpha \in (0, 1)$, typically 0.1, and using discrete overlapping detection, we check if the target object enters the sphere around the fingertip, then we stop bending the finger. It is repeated for each finger individually. Intuitively, we can identify two factors affecting performance - finger pose blending and subsequent collision detection.

The method proposed in this paper does not rely on physics, discrete overlapping detection, iterative pose blending, or multiple proxy convex meshes to model mesh holes. It can also work with animated skin meshes as shown in one of the attached video files.

3 THE SDF SAMPLING-BASED METHOD

In contrast to the physics-based approach (Algorithm 1) we do not rely on discrete overlap detection. Instead, we sample the SDF of the grasped object at voxel positions mapped to fingers' tips positions during the bending of each finger. Subsection 3.1 describes the initial

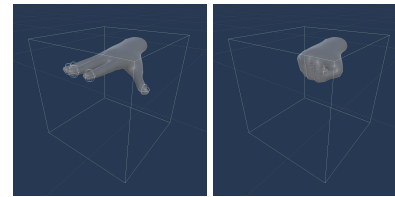


Figure 1: Hand model with an area of SDF generation and with fully closed pose

setup and prerequisites for the hand model we use for our method. At the start of the grasp phase, we compute the SDF (Subsection 3.2) of the grasped object but, for performance reasons, only in the clipped area around the virtual hand and in relatively low resolution. For optimizations, we also cache all possible fingertips positions mapped to the SDF texture space (Subsection 3.3). Values sampled from SDF at these positions (Subsection 3.4) are the Euclidean distances to the object's surface, and by comparing them with a minimum threshold we can determine when each fingertip penetrates the object and stop bending the finger. This is further described in Subsection 3.5. However, as opposed to the discrete overlap detection the sampled distance values can be used in a broader way to fine-tune the bending of the finger (Subsection 3.6).

3.1 Virtual hand configuration

For visualization, the 3D model of the hand provided with the OculusSDK library was used and further configured (Figure 1). Although we can use any rigged hand model even with a different number of fingers than the human hand. There is an added component on each finger responsible for bending the finger from the open to close position and for storing the finger's tip position.

Around the hand, we have defined an area with two purposes in mind. First, it acts as a trigger to detect a virtual object under the palm and to start interactive SDF computation of that object relative to hand position and orientation. Secondly, it is the clipping area of the computed SDF.

3.2 SDF computation

SDFs and implicit surfaces have been used for many years in computer graphics for creating and displaying shapes [Bli82, BW97, Har95, WGG99]. SDF can be described by the following mathematical function:

$$\phi(x) : \mathbb{R}^3 \rightarrow \mathbb{R} \quad (1)$$

that maps a point x in 3D space (represented in some predefined coordinate frame) to a real number. The value of $\phi(x)$ at a given point is the signed Euclidean distance from that point to the nearest point on a surface, where $\phi(x) = 0$. Points that are outside of the object being represented by the SDF have a value of

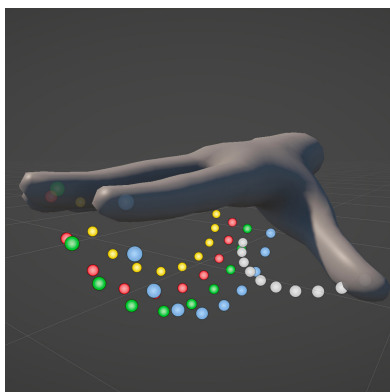


Figure 2: Hand model with all possible fingers' tip positions during auto-grasping

$\phi(x) > 0$, while points inside the object have a value of $\phi(x) < 0$.

In this paper, we've used Unity's light and fast real-time SDF generator Mesh-to-SDF [Uni22] that implements an algorithm from AMD TressFX library [Adv20]. Its main optimizations rely on taking into account the area near each triangle and then filling the rest of the SDF using linear or jump flood algorithm [RT06]. It is fast enough for real-time, especially for meshes with less than 10k triangles. In our implementation, we have assumed the SDF volumes size to be 16^3 , which in tests turned out to be sufficient and resulted in a fast computation.

3.3 Fingertip positions cache

During the auto-grasp, each finger's configuration blends from a fully open to a closed pose. The blend happens on each part of the finger by changing its local positions and orientation relative to the parent bone in the hierarchy. As a result of this blending, the finger's tip travels on a constant curve. For our implementation, we sample the tip positions on that curve by constant step $\Delta\alpha = 0.1$, where $\alpha = 0$ is the start of the curve, and $\alpha = 1$ is its end. All tip positions are stored in a 1D vector of positions. Each position in this vector is then transformed using Equation 2 from the hand local space to the SDF 3D texture space, where $x, y, z \in \mathbb{R}$ are the coordinates of transformed position, $bbox$ is the local bounding box size reflecting the area from which the SDF is created. This results in the constant cache of all possible fingertips' positions in SDF texture space.

$$T : \mathbb{R}^3 \rightarrow \mathbb{R}^3, T \left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} \right) = \begin{pmatrix} \frac{2x}{bbox_x} \\ \frac{2y}{bbox_y} \\ \frac{2z}{bbox_z} \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} \quad (2)$$

3.4 SDF sampling

To retrieve values stored in the SDF created by the Mesh-To-SDF algorithm we use a simple

multi-threaded GPU compute shader described by Algorithm 2. It takes a vector cache of possible tips' positions in 3D texture space and returns a 1D vector of sampled SDF values. By running this code on GPU we do not need to transfer the whole SDF texture from GPU memory to computer memory, and then do the linear sampling for non-integer texture coordinates on CPU. Additionally, we can sample many SDF values parallel using multiple GPU threads. After dispatching SDF sampling on GPU we use Unity's *AsyncGPUReadback* class allowing the asynchronous readback of sampled values without any stall on the CPU or GPU side.

Algorithm 2: Multithreaded SDF Sampling Compute Shader

Input : *tex* - SDF 3D texture;
 input_buffer - holding texture coordinates to sample;
 threadID - vector of three unsigned integer components *x, y, z* with thread indexes.
Output : *output_buffer* - holding sampling results
id \leftarrow *threadID*_{*x*}
coord \leftarrow coordinate at(*id*) in *input_buffer*
texel \leftarrow Sample3DTexture(*tex*, *coord*)
output_buffer[*id*] \leftarrow *texel*

3.5 SDF based auto-grasping

Algorithm 3: SDF based auto-grasping

Input : *resultArr* - result of the SDF sampling compute shader;
 $\Delta\alpha$ - the predefined step of finger on bending curve;
 minTipDistance - threshold value of SDF sample;
 fingers - list of finger objects.
alpha \leftarrow 0
stopped \leftarrow *false*
eFinger \leftarrow *fingers*.GetEnumerator()
eFinger.MoveNext()
foreach *result* in *resultArr* **do**
 if !*stopped* & *result* < *minTipDistance* **then**
 stopped \leftarrow *true*
 eFinger.Current.Squish \leftarrow *alpha*
 end
 alpha \leftarrow *alpha* + $\Delta\alpha$
 if *alpha* > 1.0 **then**
 if !*stopped* **then**
 eFinger.Current.Squish \leftarrow 1.0
 end
 alpha \leftarrow 0
 stopped \leftarrow *false*
 eFinger.MoveNext()
end
end

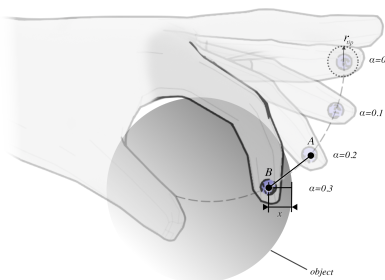


Figure 3: Index finger bending phase before and after penetrating object

After receiving a callback with sampled SDF values they can be processed on the CPU side by Algorithm 3. Its main loop iterates through the *resultArr* array holding the sampled SDF values in discrete positions on each finger’s bending curves traversed by the $\Delta\alpha$ step. During this iteration, our algorithm also iterates through each finger component responsible for its bending. If the current SDF value from the *resultArr* is less than the specified minimum threshold value, then the current finger is bent to the corresponding position on the bending curve, any remaining samples for that finger are skipped, and the algorithm iterates to the next finger.

3.6 Fine tuning using SDF sample

The fingertip bend value is stored in the α variable taking a normalized value in the range 0..1 (from a fully open pose to a close one). Assuming the value of $\Delta\alpha = 0.1$ as in Figure 3, it may happen that for successive discrete positions, the fingertip is above the surface of the object (point A), and in the next step it is inside the object (point B). In methods using discrete overlap detection, one retracts the finger to point A by changing the value of the bend $\alpha \leftarrow \alpha - \Delta\alpha$. The finger bending is then repeated but this time for a smaller step, e.g. $\Delta\alpha = 0.01$.

This approach is not suitable when using SDF, as it would require another cache of fingertip positions and another round of SDF sampling. Fortunately, SDF samples give us more information than just discrete collision - specifically the distance to the nearest surface. In the proposed method, we use the SDF value x read at the time of penetration into the object. \overline{AB} is the segment between the last position of the fingertip before penetrating the object and the current position inside the object. \widehat{AB} denotes the arc between two points on the bending curve of the current finger. When $\Delta\alpha \rightarrow 0$ then $|\overline{AB}| = |\widehat{AB}|$, so we can assume for small values of $\Delta\alpha$ that $|\overline{AB}| \approx |\widehat{AB}|$. If $|\overline{AB}|$ is the approximated distance the finger traveled from point A to B on the curve, then the correction of α value to push the finger from the surface can be approximated with the following equation:

$$\alpha_{corr} = \frac{x - r_{tip}}{|\overline{AB}|} \Delta\alpha \quad (3)$$

4 RESULTS

The proposed method was implemented and tested in the Unity Editor environment [Uni05]. Our implementation is available online as a GitHub repository [SDF23]. Screenshots and timing measurements were done on the Apple MacBook Air M1 laptop.

METHOD	Bunny	Armadillo	Dragon	Pixel
Physics	0.08ms	0.04ms	0.1ms	0.08ms
Ours	0.01ms	0.02ms	0.03ms	0.02ms

Table 1: Avg timings of grasping methods for test objects

Figure 4 shows the comparison of the physics-based and the SDF sampling-based grasps on test objects. The first image in each set shows a physics-based grasp relying on overlap detection. The second image shows a simple convex mesh collider used for overlap detection. The third image shows the result of the proposed SDF sampling-based method, and the last one adds a visualization of the SDF sample. As we can infer from examples, the physics-based grasp produces worse grasps than the SDF sampling based. In most distinctive examples, the fingers stop before reaching the object’s surface because they overlap already with a simple convex mesh collider.

Table 1 shows the average timings of physics-based and SDF sampling-based grasping methods for each of the tested models. On a note, the latter does not include the timings of SDF computation as it is not a part of the proposed method, depends on the selected implementation, and heavily relies on a specific model’s triangle count. For a Pixel test model (with 499,548 triangle count) online SDF computation can result in >0.6ms overhead while for models with $\leq 10k$ triangle count the overhead is insignificant. Additionally, each *GPUAsyncCallback* can stall SDF sampling results for an additional frame.

5 LIMITATIONS

In addition to the last note, it is worth mentioning other limitations of the proposed method. First, the results are highly dependent on the initial position of the virtual hand. In the context of this paper, however, we do not elaborate on the stage of hand approach to the object surface. The proposed method and implementation focus only on detection around the fingertips and is subject to, for example, interpenetration of other parts of the fingers with grasped objects. We can also imagine a situation in which, during the bending phase of a finger with $\Delta\alpha = 0.1$, the fingertip misses a small obstacle on



Figure 4: Example grasps of test objects including Stanford Bunny, Stanford Dragon, Stanford Armadillo, and Pixel mascot presented in four configurations: physics-based grasp, convex collider used for physics-based grasp, SDF sampling based grasp and SDF visualization

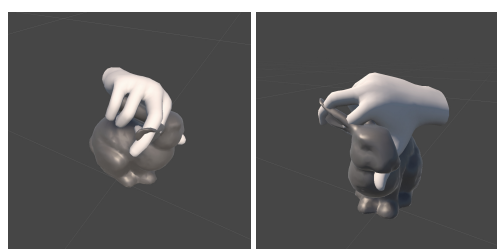


Figure 5: Examples of interpenetration

it's bending curve. This, however can be resolved by using smaller $\Delta\alpha$ values.

Figure 5 shows examples of results failed due to the above limitations.

6 CONCLUSIONS

In this paper, we have proposed the SDF sampling-based auto-grasping method as a replacement for the physics-based one used for example in popular commercial auto-grasping toolkits for the Unity game en-

gine. Using SDF sampling produces better visual results and the sampling method implementation is faster than the one based on discrete sphere overlap detection. The trade-off is that we also need to implement a selected SDF computation algorithm that for 3D models with bigger triangle counts can have an impact on performance. But taking into account the dynamic development of new SDF computation methods and that SDF computation can be tailored for our specific usage (low resolution and small clipped area of SDF computation) we can assume future improvements in this field.

Regarding possible interpenetration, in future work, we can focus on sampling SDF values for multiple contact points on the hand. We could also benefit from using different bending strategies like bending parts of each finger separately, using SDF sample values for wider optimizations, and leveraging more GPU-side computations.

As for the hand approach phase, it can also be solved using sampling the SDF around, for example, the

hand's palm center. Additional calculation of the SDF derivative at the same point will result in a near-surface normal that can be used to determine the orientation of the whole hand. For this reason, we do not present Q-distance comparisons as these depend strongly on the aforementioned stage.

Our auto-grasping method is not the first one using SDFs, but in opposition to other methods [BCO⁺21, TWH⁺22], it does not need any precomputations. It targets online usage, can be used with dynamic skinned meshes, and focuses only on the small clipped area for the SDF computations, not the entire object or environment. In future work, we would like to study the contextless clipped SDF samples as an input for a generalized machine learning approach to auto-grasping. Additionally, it can leverage from using imitation learning with grasping poses obtained using, i.e., Meta Quest hand tracking.

7 REFERENCES

- [Adv20] AMD, Inc. TressFX (2020). URL: <https://github.com/GPUOpen-Effects/TressFX>.
- [AHTL16] Argelaguet, F., Hoyet, L., Trico, M., and Lecuyer, A. The role of interaction in virtual embodiment: Effects of the virtual hand representation. In *2016 IEEE Virtual Reality (VR)*, pages 3–10. IEEE, New York (2016). DOI: 10.1109/VR.2016.7504682.
- [BCO⁺21] Breyer, M., Chung, J.J., Ott, L., Siegwart, R., and Nieto, J. Volumetric Grasping Network: Real-time 6 DOF Grasp Detection in Clutter. In J. Kober, F. Ramos, and C. Tomlin, editors, *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 1602–1611. PMLR, Cambridge MA (2021). DOI: 10.48550/arXiv.2101.01132.
- [BI05] Borst, C. and Indugula, A. Realistic virtual grasping. In *IEEE Proceedings. VR 2005. Virtual Reality, 2005.*, pages 91–98. IEEE, New York (2005). DOI: 10.1109/vr.2005.1492758.
- [Bli82] Blinn, J.F. A Generalization of Algebraic Surface Drawing. *ACM Trans. Graph.*, 1(3):235–256 (1982). DOI: 10.1145/357306.357310.
- [BW97] Bloomenthal, J. and Wyvill, B. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco (1997). ISBN 155860233X.
- [CLL22] Cai, Q., Li, J., and Long, J. Effect of Physical and Virtual Feedback on Reach-to-Grasp Movements in Virtual Environments. *IEEE Transactions on Cognitive and Developmental Systems*, 14(2):708–714 (2022). DOI: 10.1109/TCDS.2021.3066618.
- [Clo20] Cloudwalkin Games. Hurricane VR - Physics Interaction Toolkit (2020). URL: <https://assetstore.unity.com/packages/tools/physics/hurricane-vr-physics-interaction-toolkit-177300>.
- [CNS⁺19] Canales, R., Normoyle, A., Sun, Y., Ye, Y., Luca, M.D., and Jörg, S. Virtual Grasping Feedback and Virtual Hand Ownership. In *ACM Symposium on Applied Perception 2019, SAP '19*. ACM, New York (2019). ISBN 9781450368902. DOI: 10.1145/3343036.3343132.
- [COC22] Chen, J., Or, C.K., and Chen, T. Effectiveness of Using Virtual Reality-Supported Exercise Therapy for Upper Extremity Motor Rehabilitation in Patients With Stroke: Systematic Review and Meta-analysis of Randomized Controlled Trials. *J Med Internet Res*, 24(6):e24111 (2022). DOI: 10.2196/24111.
- [DMF⁺19] D'Alonzo, M., Mioli, A., Formica, D., Vollero, L., and Di Pino, G. Different level of virtualization of sight and touch produces the uncanny valley of avatar's hand embodiment. *Scientific Reports*, 9(1):19030 (2019). DOI: 10.1038/s41598-019-55478-z.
- [Ear20] Earnest Robot. Auto Hand - VR Interaction (2020). URL: <https://assetstore.unity.com/packages/tools/game-toolkits/auto-hand-vr-interaction-165323>.
- [EWB20] Elbehery, M., Weidner, F., and Broll, W. Haptic Space: The Effect of a Rigid Hand Representation on Presence When Interacting with Passive Haptics Controls in VR. In *19th International Conference on Mobile and Ubiquitous Multimedia, MUM '20*, page 245–253. ACM, New York (2020). ISBN 9781450388702. DOI: 10.1145/3428361.3428388.
- [FSY⁺19] Furmanek, M.P., Schettino, L.F., Yarossi, M., Kirkman, S., Adamovich, S.V., and Tunik, E. Coordination of reach-to-grasp in physical and haptic-free virtual environments. *Journal of NeuroEngineering and Rehabilitation*, 16(1):78 (2019). DOI: 10.1186/s12984-019-0525-9.
- [Har95] Hart, J. Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer*, 12 (1995). DOI: 10.1007/s003710050084.
- [IWL⁺22] Isenstein, E.L., Waz, T., LoPrete, A., Hernandez, Y., Knight, E.J., Busza, A., and Tadin, D. Rapid assessment of hand reaching using virtual reality and application in cerebellar stroke. *PLOS ONE*, 17(9):1–21 (2022). DOI: 10.1371/journal.pone.0275220.
- [JSF12] Jacobs, J., Stengel, M., and Froehlich, B. A generalized God-object method for plausible finger-based interactions in virtual environments. In *2012 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 43–51. IEEE, New York (2012). DOI:

- 10.1109/3dui.2012.6184183.
- [JYM⁺20] Jörg, S., Ye, Y., Mueller, F., Neff, M., and Zordan, V. Virtual Hands in VR: Motion Capture, Synthesis, and Perception. In *SIGGRAPH Asia 2020 Courses*, SA '20. ACM, New York (2020). ISBN 9781450381123. DOI: 10.1145/3415263.3419155.
- [KSF⁺18] Knierim, P., Schwind, V., Feit, A.M., Nieuwenhuizen, F., and Henze, N. Physical Keyboards in Virtual Reality. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–9. ACM, New York (2018). DOI: 10.1145/3173574.3173919.
- [LC21] Lavoie, E. and Chapman, C.S. What's limbs got to do with it? Real-world movement correlates with feelings of ownership over virtual arms during object interactions in virtual reality. *Neuroscience of Consciousness* (2021). DOI: 10.1093/nc/niaa027.
- [LGAMB06] Le Garrec, J., Andriot, C., Merlhiot, X., and Bidaud, P. Virtual Grasping of Deformable Objects with Exact Contact Friction in Real Time. *WSCG '2006: short communications proceedings*, pages 87–92 (2006).
- [LKRI20] Lougiakis, C., Katifori, A., Roussou, M., and Ioannidis, I.P. Effects of Virtual Hand Representation on Interaction and Embodiment in HMD-based Virtual Environments Using Controllers. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 510–518. IEEE, New York (2020). DOI: 10.1109/VR46266.2020.00072.
- [Met22a] Meta. Creating Hand Grab Poses (2022). URL: <https://developer.oculus.com/documentation/unity/unity-isdk-creating-handgrab-poses/>.
- [Met22b] Meta. Interaction SDK Overview (2022). URL: <https://developer.oculus.com/documentation/unity/unity-isdk-interaction-sdk-overview/>.
- [ORC07] Ortega, M., Redon, S., and Coquillart, S. A Six Degree-of-Freedom God-Object Method for Haptic Display of Rigid Bodies with Surface Properties. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):458–469 (2007). DOI: 10.1109/TVCG.2007.1028.
- [PZ05] Pollard, N.S. and Zordan, V.B. Physically Based Grasping Control from Example. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, page 311–318. ACM, New York (2005). ISBN 1595931988. DOI: 10.1145/1073368.1073413.
- [RT06] Rong, G. and Tan, T.S. Jump Flooding in GPU with Applications to Voronoi Diagram and Distance Transform. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, I3D '06, page 109–116. ACM, New York (2006). ISBN 159593295X. DOI: 10.1145/1111411.1111431.
- [SDF23] SDF2HandPose (2023). URL: <https://github.com/nosferathoo/SDF2HandPose>.
- [SHX⁺22] She, Q., Hu, R., Xu, J., Liu, M., Xu, K., and Huang, H. Learning High-DOF Reaching-and-Grasping via Dynamic Representation of Gripper-Object Interaction. *ACM Trans. Graph.*, 41(4) (2022). DOI: 10.1145/3528223.3530091.
- [TWH⁺22] Turpin, D., Wang, L., Heiden, E., Chen, Y.C., Macklin, M., Tsogkas, S., Dickinson, S., and Garg, A. Grasp'D: Differentiable Contact-Rich Grasp Synthesis for Multi-Fingered Hands. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part VI*, page 201–221. Springer-Verlag, Berlin, Heidelberg (2022). ISBN 978-3-031-20067-0. DOI: 10.1007/978-3-031-20068-7_12.
- [TWMZ19] Tian, H., Wang, C., Manocha, D., and Zhang, X. Realtime Hand-Object Interaction Using Learned Grasp Space for Virtual Environments. *IEEE Transactions on Visualization and Computer Graphics*, 25(8):2623–2635 (2019). DOI: 10.1109/TVCG.2018.2849381.
- [Uni05] Unity Technologies. Unity Real-Time Development Platform (2005). URL: <https://unity.com>.
- [Uni16] Unity Technologies. The V-HACD library decomposes a 3D surface into a set of "near" convex parts. (2016). URL: <https://github.com/Unity-Technologies/VHACD>.
- [Uni22] Unity Technologies. Mesh-To-SDF (2022). URL: <https://github.com/Unity-Technologies/com.unity.demoteam.mesh-to-sdf>.
- [Val19] Valve Corporation. Valve Index - Upgrade your experience (2019). URL: <https://www.valvesoftware.com/en/index/controllers>.
- [Val20] Valve Corporation. Skeleton Input | SteamVR Unity Plugin (2020). URL: https://valvesoftware.github.io/steamvr_unity_plugin/articles/Skeleton-Input.html.
- [ŽCJ18] Žilak, M., Car, v., and Ježić, G. Educational Virtual Environment Based on Oculus Rift and Leap Motion Devices. *WSCG '2018: short communications proceedings*, pages 143–151 (2018). DOI: 10.24132/CSRN.2018.2802.18.
- [WGG99] Wyvill, B., Guy, A., and Galin, E. Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Comput. Graph. Forum*, 18:149–158 (1999). DOI: 10.1111/1467-8659.00365.
- [WW80] Welch, R. and Warren, D. Immediate perceptual response to intersensory discrepancy. *Psychological bulletin*, 88(3):638–667 (1980). URL: <http://europemc.org/abstract/MED/7003641>.