ChatGPT

3D Object Selection and Manipulation Techniques in WebXR (VR/AR) Interfaces

Introduction

Enabling users to select and manipulate virtual objects is a core challenge for 3D user interfaces in VR/AR. In web-based XR applications (e.g. using WebXR with Three.js or A-Frame), developers can draw on a variety of techniques to make interacting with objects intuitive. Unlike a 2D screen (where a mouse or touch defines a point), immersive 3D interaction is open-ended – users might point with controllers, reach with their hands, or even use gaze and voice. No single technique works best for all scenarios [1]. In fact, research shows the optimal selection method can depend on context (object size, distance, density of objects, device input available); advanced systems even **adapt or combine techniques** on the fly to cover different needs [2]. Below, we survey the primary interaction patterns – focusing on those relevant to WebXR and modern VR/AR UX – and discuss how they're implemented (especially with frameworks like Three.js, A-Frame, and Babylon.js) along with findings from HCI research on their effectiveness.

Raycasting (Laser Pointer) Selection

Raycasting is one of the most common techniques for selecting distant objects in VR. It works like a laser pointer: a virtual ray (usually visible as a thin beam) projects out from a controller or the user's hand, and whichever object the ray intersects can be "hit" for selection. This is essentially a 3D extension of a pointing cursor. WebXR libraries make raycasting straightforward to use. For example, Three.js provides a `Raycaster` utility to compute intersections; it can even take a **WebXR controller** pose as the origin/direction of the ray [3]. A-Frame (an entity-component framework on top of Three.js) includes a built-in `laser-controls` **component** that attaches a visible ray to 6DoF controllers, generating events when the ray intersects objects [4]. Likewise, Babylon.js' WebXR features enable *pointer selection* by default – the "default XR experience" spawns a controller-linked laser and lets users interact with scene objects via trigger presses [5]. In practice, using these frameworks, a developer can register event listeners on objects for pointer events (e.g. "select" or "trigger pressed while pointing at this object"), and the framework handles the ray-object intersection math under the hood.

From a UX perspective, raycast selection is intuitive and efficient for **far targets**. The user simply points the controller at an object and clicks (e.g. presses the trigger button) to select – similar to clicking with a mouse, but in 3D space. In user studies, controller-based pointing has proven highly effective. For example, one study found that using a handheld controller to aim a ray at targets yielded the **fastest selection times** compared to head-gaze or eye-gaze aiming [6]. This is likely because a controller can be aimed quickly and steadied with two hands if needed, offering good control. To enhance usability, VR systems usually provide **visual feedback** for raycasting. A-Frame's `laser-controls` will shorten or "truncate" the beam when it hits something [7], so you get a clear visual indication of which object you're pointing at. Many applications also highlight the targeted object (e.g. changing its color or outlining it) to confirm what will be selected on click [8]. Such feedback is critical – as a 2009 study on dense 3D environments showed, giving users adequate visual cues significantly improved their success in selecting objects in cluttered scenes [9].

Limitations: The basic raycast is essentially a straight line, so it can be challenging for very small or distant targets. The farther away an object is, the smaller the angular "target" for the user to hit with the ray. Liang and Green (1994) observed that selecting a small distant object with a simple ray can be difficult due to the required precision of aiming [10]. They introduced the **"spotlight" technique** as a remedy: instead of a thin ray, use a **cone** that spreads out, like a flashlight beam, so that small objects at distance fall inside the cone even if you're not perfectly exact [11]. This improves hit success, at the cost of potentially having multiple objects inside the cone. (In such cases, the system may need a disambiguation step – e.g. pick the closest, or cycle through hits – which can be designed via UI prompts.) Similarly, if one object lies directly behind another from the user's view, a normal ray will hit the front object only. Researchers have proposed **depth-based ray selection** that allows the user to cycle through or push the ray past the first hit, and even a 3D version of the bubble cursor (which dynamically adjusts a selection volume to pick the nearest object) to tackle **occluded or dense target scenarios** [12] [13]. These advanced techniques (evaluated by Vanacken et al. 2007/2009) outperformed a standard single-ray cursor in complex scenes, with a **"depth ray"** (allowing users to adjust which intersected object is selected) being especially effective [13].

In typical WebXR applications today, such elaborate methods are not built-in, but developers can script similar logic (e.g. allow a second button or thumbstick to switch the ray to the "next" intersection down the line of sight). In many cases, a simpler solution is to design the scene or interaction to avoid extreme occlusions or tiny targets, or to complement raycasting with the ability to **zoom in or teleport closer** to the target if precision becomes an issue. Indeed, even with a laser pointer, selecting very small far-away items might require the user to move nearer or use magnification. A recent study in an immersive analytics context noted that while ray pointers *extend the user's reach*, picking very small distant points was still hard without zooming, due to hand stability limits (a slight tremor of the hand translates to large jitter downrange) [14]. The takeaway is to use raycasting for what it's best at – quick targeting of reasonably sized or distant objects – and to provide support (visual aids, alternate modes) for edge cases where precision targeting is needed.

Direct Hand Interaction (Virtual Hands and Grabbing)

Another fundamental interaction metaphor is the **virtual hand**: the user reaches out (with a tracked controller or hand-tracking) and directly touches or grabs the virtual object, as if it were real. This is very natural for objects within arm's reach. When the user's hand (or controller proxy) overlaps an object's position, the system can **select** that object (often by requiring the user to press a grab button or make a grasp gesture, to avoid accidental pickups). Implementation-wise, this relies on **collision detection** (intersecting volumes) rather than a ray. In WebXR frameworks, developers often attach an invisible collider (sphere or box) to the user's hand or controller model. When this collider intersects an object's bounding box, that object is "hovered"; if the user then triggers the grab, the object can be attached to the hand for manipulation. A-Frame provides the `super-hands` component (an open-source add-on by the community) that encapsulates this pattern [15]. It interprets the VR controller's pose and buttons into high-level gestures – e.g., *Hover* (controller touching an object), *Grab* (trigger pull while hovering), *Release* (letting go of trigger), etc. – and it works with companion components on objects like `hoverable`, `grabbable`, and `drag-droppable` to handle the object's response [16] [17]. With super-hands, you can make any entity grabbable with one line of HTML, and get events when it's grabbed or released. This makes it easy to implement dragging, throwing, stacking, etc., all through natural movement. Other frameworks have similar approaches: for instance, Babylon.js recently added a **near interaction** feature to its WebXR pointer system, so that when a controller comes close to an object, it can switch to a direct grabbing mode (this was introduced to support grabbing and was discussed in their GitHub/PR and forum posts [18] [19]). In Three.js, one can also code this manually: e.g., use `XRFrame.getJointPose` (for hand tracking joints) or controller grip pose, test distances to objects each frame, and if within a threshold and trigger pressed, parent the object to the controller. Several **open-source projects** demonstrate direct hand manipulation in WebXR – a notable example is Mozilla's **A-Painter**, a VR painting app built with A-Frame. Its custom `paint-controls` (source available) show how to use tracked hand positions for actions like painting strokes in 3D space [20] (and it uses grabbing to pick up or scale the canvas, etc.). These real-world examples can be a great reference for implementing similar interactions.

From a human-factors standpoint, **direct interaction excels at precision and intuitiveness** for near objects. People have strong hand-eye coordination within arms' reach, and tasks like positioning or rotating an object can be done by naturally moving one's hands. Early VR researchers noted that the human hand is a "ideal direct manipulation device" in 3D [21] – it gives us six degrees of freedom control in a very natural way. This makes grabbing feel immediately satisfying (e.g. picking up a virtual block and placing it somewhere). Two-handed operations are also possible: for example, grabbing an object with both hands and moving them apart can **stretch/scale** the object, a common metaphor in VR (supported by super-hands via a *stretchable* component) [16]. Another example is using two hands to rotate a large object by spinning it, or doing a "grab in one hand and tap with the other" to trigger something. These mimic real-world bimanual actions and can increase control for larger objects.

Limitations: The obvious drawback of a purely literal hand metaphor is **limited reach**. In a virtual world, objects might be beyond the user's physical arm length or on the other side of a room. Walking in VR to grab something is possible (if the user has room or via teleportation), but constantly having to move just to select far objects would be inefficient. As VR pioneers Bowman and Hodges noted, the user's real hand cannot interact beyond their arm span [22] – so virtual-hand-only interaction breaks down in a typical scenario where interesting objects are spread throughout an environment. To address this, a few clever techniques were invented:

- **Go-Go Technique:** Introduced by Poupyrev et al. (1996), this technique *non-linearly scales* the range of the user's arm. Up to a certain distance, the virtual hand follows the real hand normally; but once the user pushes out beyond a threshold, the virtual hand *"grows" longer* (through an exponential mapping) allowing the user to reach faraway objects [23]. Essentially, your arm becomes Go-Go Gadget arm in VR. This allows seamless direct grabbing both near and far – you can snatch an object from across the room by simply reaching out further. It's a very intuitive metaphor (users often don't realize the math happening; they just feel like they could reach farther in VR than in reality). The trade-off is that extreme non-linear mapping can distort proprioception (you might not precisely judge distances when your arm is scaled), but for coarse selection it works well. Research found the Go-Go technique particularly advantageous as target distance increases, compared to no extension [24].

- **Ray-Casting + "Reel" (HOMER):** Another approach is to combine raycasting for selection with hand manipulation for fine control. Bowman's **HOMER technique** (1997) is a prime example: the user points with a ray to select a distant object (essentially "hooking" it), then the object is programmatically **pulled to the user's hand** – allowing the user to grab it and move/rotate it as if it were picked up, then place it at a new location [25] [26]. This "select at a distance, manipulate up close" approach retains the precision of direct hand motions but extends reach via the ray. Many VR games use a variant of this (sometimes called "telekinesis" or "force pull" – think grabbing an object from afar like a Jedi). For instance, if you've seen VR apps where you point at an object, hit grab, and the object flies to your hand, that's HOMER in spirit. It's useful for mid-sized objects where you want to carefully orient them – by bringing it near, you avoid trying to do tiny rotations at a distance. A-Frame's default interactions don't automatically implement this, but a developer can simulate it (e.g. on trigger press, tween the object's position to the hand and then parent it for manipulation).

These techniques, along with others like the **"virtual hand + ray" hybrids**, underline a key point: **combining metaphors can yield the best usability**. Modern VR interaction design often enables both direct grabbing and ray pointing simultaneously. For example, Oculus Quest apps allow you to use the controller as a laser pointer for far UI menus, but also use it to touch and grab nearby items. A recent study on immersive analytics interactions argued that each metaphor has inherent strengths and that giving users a way to fluidly switch **without explicit mode toggling** is ideal [27]. In their "Mixed" condition, users could always use their index finger as a virtual hand *and* had a ray emanating for distant pointing [28] – essentially supporting whatever the user tries to do. This kind of parallel availability might be advanced to implement, but even a simpler approach (e.g. automatically switch to raycast if the user points at something out of reach) can improve experience. Older comparative studies found, unsurprisingly, that **virtual ray pointers outperform hands for distant targets in speed**, whereas **hands outperform rays for tasks requiring precise manipulation up close** [14]. Thus, an application that involves both (say, an educational demo where you sometimes grab objects to inspect them, but also have far-off items to activate) may benefit from using both techniques in tandem.

In WebXR development, you can combine these by using rays for hover selection at a distance and colliders for near objects. For example, Super-hands allows "hover" events via collider, but one could also attach a short ray to the hand for a similar effect. The Babylon.js framework, as of 2021, was discussing adding *auto-switching* between far and near interactions [18] (the idea being the system decides if a laser or direct grab should be active based on what's targeted). These nuances are still evolving in web libraries, but the building blocks are there.

Gaze-Based and Other Selection Methods

Not all VR/AR interactions rely on held controllers or physical reach. Especially in mobile or web XR scenarios (e.g. simple Cardboard-style VR, or AR on a phone), **gaze-based selection** is a prevalent pattern. Here, the user uses their head orientation or eye gaze as the pointer – essentially, *"look at the object you want"*. For example, A-Frame's `<a-cursor>` component implements a gaze cursor: it renders a reticle in the center of view and uses a ray straight out from the camera (the user's view) to intersect objects [29]. By default, a cursor in A-Frame will emit events like it's "hovering" whatever object is at the center of view, and if the user dwells for a certain time (or if they click a button on a cheap Bluetooth remote, etc.), it counts as a selection. This is crucial for 3DoF headsets or hands-free experiences – *e.g.* Oculus Go, or when using only head movement and a single button. Many WebXR frameworks treat gaze as just another **"pointer origin"**; in Babylon.js, an XR input source can have a `targetRayMode` of `"gaze"` (meaning the pointer ray comes from the head direction) or `"screen"` (for AR on phones, where tapping on screen is like a pointer) [30]. So the same pointer selection logic can apply, just anchored to the head instead of a controller.

The appeal of gaze-based pointing is simplicity – every VR device has head tracking, so it's universally available. However, it comes with usability challenges. One is the **Midas touch** problem: if simply looking at something activates it, users may trigger things unintentionally just by letting their attention wander [31]. To mitigate this, systems often use a **dwell time** (e.g. stare for ~1 second to "click") or require an explicit action (like a blink, voice command, or button press) to confirm selection. Another issue is that our head (or eyes) are always moving in small saccades; holding one's gaze steady on a small target for a countdown can be fatiguing and slower than a quick button press. In fact, studies have found that a **physical button press tends to be faster and less straining** than a dwell for confirming gaze selections [32] – but of course a button may not always be available (e.g. if the user has no controller and can't easily do a hand gesture). Newer AR systems like HoloLens and Apple Vision Pro combine eye gaze with a hand gesture "pinch" to select, which is both quick and intentional, avoiding a long dwell. Research is validating these combinations: one experiment showed that using an **eye-gaze pointer + a pinch gesture** can be as fast as a traditional controller click, and significantly faster than a pure dwell, without increasing errors [33]. In that study, the pinch was slightly slower than a controller trigger and had a bit higher error rate, but neither difference was statistically significant [33] – making it a "reasonable alternative" to clicking in gaze-based systems.

Another variant is **head-gaze vs eye-gaze**: if a device has actual eye-tracking, it can be far more precise than using head direction. One recent paper (Sharmin Shishir et al. 2025) explored a "Jedi Force Pull" scenario with eye tracking: the user would look at an object and then perform a grab (via a button or gesture) to pull it. They found that **eye-directed selection and pulling outperformed traditional hand-based distance grabbing** in both speed and accuracy [34]. This suggests future WebXR (as eye-tracking becomes available in browsers/headsets) might leverage gaze for very rapid targeting, with hand gestures as confirm – a potent combo for fast interactions. Already, simpler forms of this exist: some VR UIs let you aim with your eyes and click with the controller, which users report can feel very efficient for selection (the hand doesn't have to point, only confirm).

Outside of gaze, we should mention **voice commands and symbolic gestures** as ancillary selection methods. Voice is a natural modality in AR (e.g. saying *"delete object"* or *"select red cube"*). In combination with pointing, voice can disambiguate selection ("put that there" where "that" is resolved by a point [35]). However, voice alone (naming objects) requires robust speech recognition and is not always practical for arbitrary 3D object selection, so it's usually a supplement to the physical methods. Hand gestures (without trackers) are more relevant in AR – e.g. on HoloLens you could do a specific gesture (like air-tap) while targeting something with gaze. These are more platform-specific and less common in WebXR (since WebXR hand tracking is still an evolving area, though APIs do exist for detecting common gestures).

In summary, gaze-based selection is a valuable technique for cases where controllers or hands aren't in use, or as an accessibility option. It benefits greatly from carefully designed feedback and timing. Best practices include showing a **progress indicator** for dwell (so the user knows the selection will fire, and can abort by looking away), and using larger activation targets if possible to account for the less precise nature of head movement. If implementing gaze in a WebXR app, the A-Frame cursor gives a head start with events like `mouseenter` (object focus) and `click` after dwell. Just be sure to adjust dwell time to balance speed vs. false activations (some apps use ~500 ms with a shrinking ring cursor to show progress).

Interaction Feedback and UX Guidelines

Designing effective 3D interactions isn't just about choosing a technique; it's also about the **UX polish** that makes the technique usable. Here are some key guidelines (supported by research and industry best practices) for 3D selection/manipulation interfaces:

- **Provide clear visual feedback for targeting and selection:** Users should never be in doubt about what object their action will affect. Techniques like highlighting the object under the cursor or ray, adding an outline or glow, or changing its scale slightly can all indicate "this is the current target."

Studies recommend rendering visual hints like this to **indicate the target and pointing direction** clearly in VR [8]. For raycasting, the beam itself acts as a strong cue; many apps also use a small dot or reticle at the ray's hit point. For direct hand interactions, highlighting an object as the hand nears it (or a subtle vibration if using haptics) can confirm you're touching it.

- **Support error prevention and easy corrections:** Because 3D targeting can be imprecise, consider techniques like **sticky selection** (once something is highlighted, slight hand jitter won't switch off the target immediately) or **confirmation steps** for critical actions. For example, if deletion is bound to grabbing and throwing an object away, maybe require holding for 2 seconds or a secondary confirm (so it's not done accidentally). Empirical results often show a trade-off between speed and errors – e.g. a quick click might lead to more misses than a dwell select [32]. Tuning the interaction (like adding a tiny delay or larger hit zones) can mitigate error rates without hugely sacrificing speed [32].

- **Handle occlusion and dense environments:** If your scene can have many objects close together, provide ways for the user to accurately pick one. This could be a **"highlight all"** on ray hover (so the user sees if multiple objects are along the line), or a cycling mechanism (e.g., thumbstick up/down to cycle through overlapping hits). Research into dense selection techniques found that even simply allowing multiple objects to be selectable by a single ray (like a depth ray) plus giving visual feedback for the selection depth helped users acquire occluded targets better [12]. In 2D GUIs we have drop-down lists for overlapping items; in 3D, you might implement a similar concept if needed (like a small menu that pops up if your selection is ambiguous: "Did you mean X or Y?"). It's extra work, but important for complex scenes.

- **Use two-handed and gesture interactions for manipulation:** As mentioned, allowing the user to use both hands can make manipulations like resizing or rotating objects more natural [16]. For instance, a common pattern in VR educational apps (e.g. anatomy explorers or molecule viewers) is: one hand grabs an object, the second hand can then scale or spin it. If using A-Frame, enabling the `stretchable` and `rotatable` (from super-hands or related) on objects gives this behavior – the user grabs with two hands and moves them to scale, or twists to rotate. These multi-hand gestures leverage proprioception and can be easier to learn than fiddling with a UI slider for scale.

- **Match the technique to the task and user's context:** There is no one-size-fits-all, so consider providing multiple means of selection if your app has varied tasks. For example, a VR modeling tool on the web might use raycast clicking for creating or selecting an object at a distance, but once the user is editing that object, switch to direct hand controls for fine adjustments. Ensuring the transition is smooth (perhaps automatically detected, or via an obvious mode switch like a "Edit Mode" toggle) will help. Also, keep in mind user preferences and physical differences. Some users may find holding a controller arm outstretched (for raycasting) tiring after a long period and might prefer using gaze for a while, or vice versa. If feasible, offer options (e.g. in settings: "Select by gaze + tap vs. controller ray"). In professional applications or longer sessions, fatigue is a real concern – minimizing required physical effort (by using comfortable poses, selection time-outs, etc.) improves overall experience.

- **Leverage the web's ability to integrate 2D UI as needed:** WebXR allows overlaying HTML/CSS or using 2D HUDs. Sometimes a 2D panel or traditional menu can be easier for certain selections (especially abstract commands). A hybrid UI (part 3D, part flat UI) can work well: for example, an educational VR scene might let the user point at an object to get its name to appear in a tooltip (2D overlay), then the user can gaze at that tooltip to confirm they want info about it. Don't feel pressured to do everything with direct physics if a simple button can suffice – the key is usability. The **web stack** makes it relatively easy to incorporate such overlays or to update the content dynamically based on selection events.

Finally, it's worth noting that decades of research have culminated in some standard guidelines and even checklists for 3D interaction design. A recent review by Bergström et al. (2021) distilled **20 years of VR selection/manipulation studies into recommendations** [36]. The high-level message is to carefully evaluate new techniques on multiple metrics (speed, accuracy, learning, user comfort, etc.) and to use consistent tasks for comparison so we can build a body of knowledge. For a developer, a practical takeaway is: **test your interaction with users**, even informally. Watch if they struggle to select certain objects or if they get tired or confused, then iterate. The beauty of web-based XR is you can often push updates quickly and gather feedback widely. Also, keep an eye on evolving WebXR APIs – features like hand tracking, eye tracking, and even upcoming standards for haptic feedback can open new interaction possibilities (for example, using hand skeleton tracking to allow grabbing without controllers, which A-Frame and Three.js have experimental support for). These can enhance or replace older techniques as the tech matures.

Conclusion

In web-based VR/AR development, you have a rich toolkit of interaction techniques at your disposal. The **raycasting pointer** is a proven solution for distant object selection – easy to implement (Three.js's `Raycaster` and frameworks' built-ins handle it) and efficient for users [4] [3]. The **virtual hand** metaphor enables natural grabbing and manipulation of close objects, leveraging 3D colliders and physics to make interactions feel real [37] [15]. Each approach has its place: raycast and other "at-a-distance" methods shine for reaching and activating far targets, while direct hand interactions shine for nuanced control and tangible "pick and place" tasks. HCI research and industry practice both suggest that combining these – and letting the user smoothly transition – yields the best results for complex applications [27]. We also have alternative modalities like gaze and voice that can fill in when hands are busy or hardware is limited, ensuring even users without full motion controllers can interact (critical for inclusivity on the web platform).

When designing your WebXR UI, start by considering the primary use cases: Is it more like **"point and click"** (e.g., selecting items in a visualization) or **"grab and manipulate"** (e.g., building or crafting in VR)? Choose the techniques that fit, but also consider offering multiple ways if users might benefit (for instance, experts might prefer a faster eye-pinch shortcut, whereas newbies use the laser pointer until they're comfortable). Implement using the tools from frameworks (they save a ton of time and have cross-device compatibility baked in), and don't forget to refine the UX with thoughtful feedback and testing. By grounding your design in both the **proven patterns** (from prior art in VR and existing libraries) and the **lessons from research** (regarding selection accuracy, ergonomics, etc.), you can create a 3D interaction experience that is robust, user-friendly, and effective for your web-based XR application. The end goal is an interface that feels as "natural" as possible – where users can focus on the task or content (be it a virtual model or an educational simulation) rather than struggle with the controls – and with the wealth of techniques available today, that goal is more achievable than ever.

Sources: The insights above draw from a mix of academic research and practical documentation. For implementation specifics in WebXR, refer to frameworks' docs (e.g., Three.js and A-Frame for raycasting and controller setup [3] [4], A-Frame's interaction primitives like super-hands [15], and Babylon.js's WebXR pointers [5]). For deeper background on 3D interaction techniques and their evaluations, see the work of Bowman et al. on virtual hand vs. ray-casting techniques [23] [14], the Go-Go technique by Poupyrev [23], and studies on selection in dense environments by Vanacken/Grossman [11] [12]. Recent user studies (e.g., on gaze vs. controller input [6] or pinch vs. click selection methods [33]) provide empirical support for the usability points discussed. All these sources reinforce the recommendations given, and they are cited inline to guide you to more detailed information on each topic. Enjoy building your interactive 3D experience!

---

[1] [2] "Intelligent Selection Techniques For Virtual Environments" by Jeffrey Cashion
https://stars.library.ucf.edu/etd/4600/?utm_source=stars.library.ucf.edu%2Fetd%2F4600&utm_medium=PDF&utm_campaign=PDFCoverPages

[3] Raycaster - Three.js Docs
https://threejs.org/docs/pages/Raycaster.html

[4] [7] [15] [16] [17] [20] [29] [37] Interactions & Controllers – A-Frame
https://aframe.io/docs/1.7.0/introduction/interactions-and-controllers.html

[5] [18] [19] WebXR Interact With Scene And Meshes - Questions - Babylon.js
https://forum.babylonjs.com/t/webxr-interact-with-scene-and-meshes/21002

[6] [31] [32] [34] Comparison of Gaze Cursor Input Methods for Virtual Reality Devices | Request PDF
https://www.researchgate.net/publication/325890070_Comparison_of_Gaze_Cursor_Input_Methods_for_Virtual_Reality_Devices

[8] [36] Toward Making Virtual Reality (VR) More Inclusive for Older Adults: Investigating Aging Effects on Object Selection and Manipulation in VR
https://www.mingmingfan.com/papers/CHI24_OlderAdults_VR_Accessibility_Modeling.pdf

[9] [10] [11] [12] [13] doi:10.1016/j.ijhcs.2008.09.001
https://www.tovigrossman.com/papers/Multimodalselection.pdf

[14] [25] [26] [27] [28] Comparing and Combining Virtual Hand and Virtual Ray Pointer Interactions for Data Manipulation in Immersive Analytics
https://vvise.iat.sfu.ca/user/data/papers/stcinteraction.pdf

[21] [22] [23] [35] (PDF) Evaluation of pointing techniques for ray casting selection in virtual environments
https://www.researchgate.net/publication/228806596_Evaluation_of_pointing_techniques_for_ray_casting_selection_in_virtual_environments

[24] Reaching further in VR: a comparative study with a novel velocity ...
https://pmc.ncbi.nlm.nih.gov/articles/PMC12660419/

[30] WebXR Controllers Support - Babylon.js Documentation
https://doc.babylonjs.com/features/featuresDeepDive/webXR/webXRInputControllerSupport

[33] Pinch, Click, or Dwell: Comparing Different Selection Techniques for Eye-Gaze-Based Pointing in Virtual Reality
https://vvise.iat.sfu.ca/user/data/papers/pinchclickdwell.pdf