ChatGPT

Microcosm XR – Deep Research & Design Report

Co-Located Multiplayer in Mixed Reality

Achieving a **shared spatial coordinate system** for co-located AR/VR players is crucial. Modern XR devices (like Quest 3) perform inside-out tracking (SLAM) independently, so a strategy is needed to **align each device's world coordinate frame** into one common frame [1]. Platform vendors offer solutions such as **shared spatial anchors** (Meta, ARCore Cloud Anchors, Azure Spatial Anchors) to establish a common reference point across devices [2]. For example, Meta's **Shared Spatial Anchors** on Quest allow one user (host) to scan the room and share an anchor with others, so all headsets see virtual content locked in the same real location [3]. Meta's Mixed Reality APIs provide a workflow: one device "advertises" a co-location session and shares its **room anchor** (a spatial map of the room) with the group; guests then load that anchor and **apply a transform to align** their space to the host's [4] [5]. By aligning to a common floor or reference, each user's coordinate systems are unified, which **prevents virtual object drift** relative to the real environment [6]. A best practice is to anchor any large virtual scene or object (approaching room-size) to a physical environment feature, ensuring **minimal drift over time** [7].

Networking and synchronization must also handle low-latency updates so players see each other's actions in real time. Typically, one device or a server is authoritative over physics or object states, and networking libraries (e.g. Photon Fusion, Unity Netcode for GameObjects) replicate transforms to others. Ownership rules are often used: e.g. the player who grabs an object becomes its authority to simulate physics, reducing conflicts [8] [5]. To keep physics in sync, **gentle state reconciliation** is needed (avoiding sudden teleports). In practice, small divergences can be smoothed rather than hard-corrected, and if a conflict arises (e.g. two users grab one object), a simple rule or "God mode" override decides the outcome.

For **cross-platform or offline scenarios**, relying on cloud anchors can be problematic due to interoperability and privacy concerns [9] [2]. Recent research shows it's possible to **align devices without cloud services** by using local visual markers or manual calibration. For instance, one approach uses **AprilTags (QR-code-like fiducials)** placed in the environment for each device to scan; by detecting the same marker, headsets can compute their poses relative to it and thus to each other [10] [11]. This method allowed multiple users to share an XR space **completely offline**, avoiding the need to upload maps to any cloud [12] [13]. It addresses privacy (no external server sees your room) and works even with mixed devices (as long as they can run the marker tracking) [14] [15]. In summary, **co-location** can be achieved via vendor APIs with **shared anchors** (convenient but tied to platform and internet) or via **custom alignment** techniques (like scanning a common marker or aligning to known points) for a more platform-agnostic approach. The chosen implementation should ensure **persistent anchors** (so worlds can be saved/reloaded in the same physical spot later) and have a way to correct for any accumulated **SLAM drift** over long sessions (e.g. occasionally re-aligning to a shared reference if needed) [7] [4].

Embodied Hand Tracking & World Authoring UX

A core pillar of Microcosm XR is **"embodied authoring"** – using natural hand gestures to build and manipulate the miniature worlds. Hand tracking in XR has matured to track full 6DoF poses of the user's

hands, enabling controller-free interaction. A standard interaction set has emerged around **pinch gestures**: pinching with thumb and index finger is widely used to *grab* or select virtual objects [16]. In our context, a **pinch-and-drag** will allow a user to grab an object in the micro-world, move it, or place it elsewhere. Similarly, using two hands in combination is intuitive for scaling and rotating objects – e.g. the user can grab a virtual block with both hands and then vary the distance/relative rotation of their hands to scale or rotate it [17]. This two-handed pinch gesture maps well to how people naturally manipulate real objects (like sculpting clay or resizing an image with multitouch).

However, designing hand interactions requires considering **ergonomics and reliability**. Hand tracking can be less precise than controllers and more fatiguing if not optimized [18] [19]. Research on 3D manipulation shows that directly mapping hand motion to object motion can strain users when both translation and rotation are controlled together with one gesture [20] [21]. One study suggests separating degrees of freedom: for example, using the hand's position to control translation while using its orientation to control rotation (instead of one-to-one coupling) improved accuracy in a docking task [22] [23]. We can apply this insight by ensuring our grab interactions aren't too "twitchy" – possibly **stabilizing rotations** or requiring a deliberate twist gesture to rotate, so small unintentional wrist rotations don't jolt the object.

Another concern is **accidental grabs** or interference. Users' hands might pass through the view or overlap an object unintentionally, so we should implement filters like pinch **gesture confirmation** (requiring a certain pinch strength or dwell time to register a grab) [24]. Additionally, visual or haptic feedback helps – e.g. highlighting an object when it's eligible to be grabbed (when the hand is nearby and pinching) so the user knows what will be picked up. Meta's interaction guidelines for Quest hand tracking emphasize using pinch poses with a tolerance and providing affordances such as a "hover state" outline on grabbable objects [25].

Bimanual gestures can enrich the authoring experience beyond just scale/rotate. For instance, a two-handed spreading motion could **"spawn" new terrain** (like pulling new land up from the table) if we designate a special gesture or tool. A one-handed **flick motion** after grabbing could serve as a delete action (pinch an object and *flick* it away to delete), which matches the spec idea of *pinch + flick to delete*. Such gestures should be tuned to avoid accidental triggers – perhaps requiring a certain flick speed or a secondary confirm if the object is important.

Overall, a **gesture vocabulary** will be defined with simplicity in mind: - **Grab/Move:** Pinch to pick up objects, drag to move, release to drop. - **Scale/Rotate:** Two-hand grab on an object, moving hands apart/together to scale; twisting hands to rotate (around an axis perpendicular to the line between hands). - **Spawn tool:** Picking up a special "wand" or using a specific pinch-and-draw motion to drop new items into the world. - **Delete:** Possibly pinching an object and drawing a quick circle or flicking it out of the world plane, followed by a confirmation (like the object turns red and deletes if not grabbed again in 2 seconds).

These choices are backed by prior UX research indicating that **mid-air hand gestures** work best when they mimic real-world analogues and when system feedback guides the user [26] [17]. It's important to note that pure hand tracking can suffer from **fatigue (the "gorilla arm" effect)** if users must keep arms raised too long [18]. Since Microcosm's focus is a tabletop scale world, users can keep their hands comfortably low (near a table), mitigating fatigue. We also will support hybrid input for precision or developer convenience: e.g. allowing use of Quest controllers or a keyboard/mouse in editor mode for fine adjustments (especially helpful during development or for power users who want absolute precision).

Eye Tracking Mechanics & Gaze Visualization

Although eye tracking is not in the MVP (especially since Quest 3 hardware does not include eye trackers), it offers exciting interaction "spice" that we plan to incorporate on supported devices (e.g. Quest Pro, future headsets). Two concrete gaze-based mechanics have been proposed:

- **"Looking makes stuff go towards you" (Gaze Magnetism):** This means using gaze direction as an implicit targeting mechanism. For example, if you look at a virtual agent or object and perform an attract gesture (like holding a pinch or pressing a button), that object could levitate or move toward your position. This is inspired by techniques like **gaze + pinch selection**, where the eyes select a target and the hand pinch confirms or pulls it [27] [28]. The design principle here is a **division of labor** between eyes and hands: *"the eyes perform selection tasks, the hands do the actual manipulation"* [27]. By gazing at an object, you effectively **highlight or focus** on it (like aiming with your eyes), then a pinch could reel it in as if telekinetically pulling it. This can feel magical and intuitive, as the user's attention is naturally on what they want to grab. We must ensure there's feedback (perhaps the object glows when your gaze locks onto it) and that unintentional looks don't trigger magnetism. Following best practices from multimodal research, we'd implement a **short dwell time or a subtle secondary action** to confirm gaze selection [29] [30]. That prevents the "Midas touch" problem where every glance activates something.

- **Gaze Heatmap ("Cognitive3D" style visualization):** This is more of a debug or analytical feature than a direct gameplay mechanic, but it can be very illuminating. The idea is to record where players look over time and project that as a colored overlay on the scene – redder or brighter in areas where gaze lingers frequently, cooler in areas seldom looked at. The company Cognitive3D, for instance, provides tools to aggregate user gaze data into **3D heatmaps** on virtual scenes [31]. They highlight "high-focus zones, overlooked areas, gaze duration, and fixation points" by overlaying heatmaps on objects and environments [31] [32]. In a multiplayer Microcosm session, a **real-time shared gaze visualization** could be enabled in a "God mode" or dev mode – allowing users (or developers) to see each other's attention distribution. Research prototype systems like *FocusMetrics XR* have even synchronized multiple users' gaze in real time to improve collaboration, by showing on a 3D model where everyone is looking [33] [34]. For our purposes, implementing a gaze heatmap could mean accumulating gaze hits on the world plane and objects: each time a user's gaze fixates on a point of interest, we add a little intensity to that location. Over time a glowing map emerges on the miniature world indicating, say, that both players spent a lot of time looking at the northwest corner where a village is, but hardly ever looked at the mountains in the south. This could guide us in UI design (ensure points of interest are indeed attracting attention) and could even become a gameplay element (e.g. certain plants grow where you look often – a whimsical idea to reward attention).

To implement gaze features, **fixation detection** is key – filtering raw eye-tracking data (which can be noisy and jittery) into stable points where the user's gaze stays for at least ~100–200ms. Many platforms provide this out-of-the-box (e.g. Tobii APIs or OpenXR eye tracking extensions will give you a current fixation point in 3D). We'll need to project gaze rays into the scene: since our world is small and likely sitting on a table, we might cast a ray from the user's eye origin through the gaze direction and find which object or terrain patch it intersects. Then we can highlight that object for magnetism or increment a heatmap counter for visualization.

Note: Since Quest 3 lacks eye trackers, these features will be developed and tested on devices that have them (like a PC VR setup with eye tracking, or a Quest Pro). We will keep the code path modular so that Microcosm XR can use gaze enhancements when available, but degrade gracefully (or use just head direction as a proxy, though that's far less precise) on devices without eye tracking. As eye-tracking becomes more common in future XR hardware, these gaze-based interactions can be gradually elevated from experimental to mainstream in our app. Early design principles from recent work (e.g. Apple Vision Pro's gaze-click paradigm) remind us to maintain **minimal latency** and to avoid conflicting signals when blending gaze with hand input [35] [36]. As one paper summarized: *"One moment matters for the eyes – the moment thumb and index finger contact, the hands take over, relieving the eyes"* from further control [37] [38]. We will follow such guidance to ensure that gaze augments the experience without causing confusion.

Real-Time Scene Capture with Gaussian Splatting

One innovative (and non-optional) feature in our vision is using **Gaussian Splatting** to capture the real environment and integrate it into the experience. Gaussian splatting is a cutting-edge technique from computer vision that represents a 3D scene as a set of thousands of tiny **Gaussian ellipsoids ("splats")** rather than a traditional mesh. The advantage is that it can reconstruct **photorealistic details** (including lighting and reflections) from just a set of images, and can be rendered in real time [39]. Essentially, it's an efficient form of a Neural Radiance Field optimized for speed: each splat stores color and opacity and is rendered as a disk that blends with its neighbors, creating a continuous-looking scene.

For Microcosm XR, the plan is to let the user scan their play space (e.g. the table and surrounding objects in the room) and generate a **3D splat-based representation**. This would allow the virtual micro-world to interact richly with the real environment – for example, creatures could hide under the real table's edge because the captured table exists as geometry, or virtual rain could splash on a real desk. Moreover, by semantically understanding the scene (see next section), agents could know "this is a sofa, don't spawn creatures there" or "avoid the laptop" etc. Gaussian splats are well-suited because they **preserve fine details and texture** of the scene (unlike a sparse point cloud) while still being *lightweight enough for real-time* on modern GPUs [40] [39]. A recent system demonstrated capturing a real object with a phone and, after ~10 minutes of cloud processing, rendering it in Unity at 150 FPS as ~500k splats [41] [40]. They achieved **photorealistic quality** with that pipeline, showing that even consumer hardware can handle splat rendering efficiently [39] [42].

For our use-case, the workflow might be: the user puts on the headset and does a one-time "scanning" of the play area using the passthrough cameras. This could involve moving around the table so the system can capture it from multiple angles. The images (or video frames) are then processed (likely on-device for Quest 3 if possible, or possibly offloaded to a server or companion phone if the computation is too heavy) to produce the Gaussian splat model of the scene. *Persistence* is key – we would save this model tied to a spatial anchor so that when the user returns later, the splatted scene re-aligns with the real world properly. The Gaussian model consists of many splats each with parameters (position, size, orientation, color, etc.) [43]. Rendering them in Unity or Unreal can be done with a specialized shader that draws each Gaussian as a billboard or volume sprite with appropriate blending. The original 3DGS research by Kerbl et al. (2023) introduced a **visibility-aware splatting algorithm with anisotropic Gaussians** and achieved real-time rendering by using acceleration structures and GPU rasterization of splats [44]. We would leverage open-source implementations of this – in fact, Kerbl's team released code (under Inria license) which we can integrate [45]. Additionally, projects like **Niantic's Scaniverse** have already brought Gaussian splats to mobile AR: Scaniverse (a phone app) can process scans into splats on-device and the **Into the Scaniverse**

Quest app lets users explore 3D splatted scenes at interactive framerates [46] [47]. Niantic optimized for Quest by reducing detail on distant background splats to maintain performance – *"visual fidelity is sharper with nearer objects, while backgrounds are less detailed"* to keep frame rates high [48]. We will likely need similar **LOD (Level of Detail) strategies**: for example, we can cluster and merge splats that are far away or very small, and only render full detail for the critical areas (e.g. the table surface where gameplay happens).

A potential challenge is that Gaussian splat models can be **memory intensive** if the scene is large (hundreds of thousands of splats). But given that initially we target a **tabletop scene**, the physical area is limited (a few square meters at most). This keeps the number of splats manageable. We might allow capturing **just the horizontal surface and a bit of its surroundings** rather than the entire room, to focus on what's necessary for the micro-world context (table, objects on it, maybe nearby walls for occlusion). Another challenge is **occlusion and collision**: how do virtual objects collide with or be occluded by the splat representation? Rendering-wise, our engine can treat splats similar to a point cloud or volumetric surface that writes to the depth buffer, thus correctly occluding virtual content. Unity's recent graphics experiments and community projects have shown how to draw splats with depth sorting. For collisions, we might generate a simplified mesh or depth map from the splat model to use with physics. For instance, after creating the splat point cloud of a table, we could fit a simple box collider to it (since tables are flat) or even sample the splats to create a heightfield collider.

In summary, Gaussian Splatting integration will give Microcosm XR a **"WOW factor"**: the real world becomes seamlessly merged with the tiny virtual world. Imagine a little civilization building a bridge from the virtual terrain onto your real coffee mug, or a virtual ball rolling off the constructed world and bouncing on your actual floor – all made possible because the system has a rich model of the real environment. This aligns with our pillar of **world persistence and context**: the microcosm is not an isolated instance but truly lives on *your table* and is informed by it. We will prioritize getting a basic splat capture working (even if at first it's a coarse capture) and rendering in Unity. We may use an initial offline process (take photos with phone -> cloud compute -> import splats) during development, then work on automating that in-headset as tech improves. Given that Scaniverse achieved on-device processing on smartphones [49], we are optimistic that the Quest 3's GPU can handle at least part of the training with optimized code, or we leverage an **edge-cloud** approach (local network PC does the heavy lifting, Quest streams the result).

Semantic Scene Understanding & Persistence

Building on the captured splatted scene, we want a **semantic layer** where the system recognizes key objects or surfaces in the environment. This is strategic because it enables more intelligent interactions: e.g., *"Agents should avoid the laptop and stay on the table"* or *"Spawn this microcosm on a flat surface (table) not on the floor rug"*. To achieve this, we can incorporate **3D semantic segmentation** techniques with our Gaussian splat representation.

Recent research is tackling exactly this intersection of NeRF/splat and semantics. One paper (Arafa & Stricker, 2025) points out that vanilla radiance fields or splats, when learned via 2D images, tend to mix up semantic labels at boundaries due to the alpha blending of volumes [50] [51]. They propose a method to instead cluster the Gaussians into objects and assign each an **object-level embedding** by aggregating multi-view features, enabling **open-vocabulary labeling** of objects [52] [53]. In simpler terms, rather than labeling each tiny splat with a category, they identify whole objects in the scene (like "chair", "monitor", "plant") by looking at all the images and using AI models (like CLIP) to describe those objects. This approach allows querying the 3D scene with text (e.g. find "laptop") and getting the relevant object, and also creating

a segmented 3D model where each object's splats are grouped with an ID [54]. Another approach (Identity-aware Language Gaussian Splatting, 2024) also addresses open-vocabulary 3D segmentation by distilling knowledge from 2D vision-language models into the splat representation [55].

For Microcosm XR, we might not need state-of-the-art research pipelines initially (which can be complex and require heavy ML processing), but we can start with some practical steps: - Use existing **2D segmentation** on the images we capture during the scan (for example, running a model like Segment-Anything or MobileNet segmentation on each camera frame). These would label pixels with categories (person, table, floor, screen, etc.). - Project those labels into the 3D splat cloud. Each splat originates from some pixels in training views, so one could give each splat a voted label from the source images. This basic approach might blur at edges (as noted), but it's a start. - Focus on key classes for privacy and gameplay: at least identify **people** (so we can remove them), **flat surfaces** (tables, floor), and maybe **walls** or large objects. Fine-grained labeling of every item isn't immediately needed for MVP.

With a semantic-labeled scene, we achieve a few things: - **Contextual spawning & physics:** The system can choose the appropriate place to initialize a micro-world (e.g., "spawn on a tabletop" – the user can point to a recognized table surface and drop the world there, avoiding other surfaces). Agents can query the scene, like an agent's behavior could include not walking beyond the "edge of table" if the table boundary is known. - **Safety and realism:** Virtual characters could, say, hide behind real objects (if we know object X is opaque and tall) or a virtual ball could roll under the real couch if the couch's bottom is detected. These touches greatly enhance immersion. - **Persistence and relocalization:** When saving a world state, we not only save the anchor but also could save some semantic descriptors – e.g., the world is anchored to "the dining table in room". If the anchor fails to fully resolve next time, having recognized it's the same table via vision could realign it. This might be advanced, but conceptually feasible with good 3D understanding.

It's worth noting that open-vocabulary 3D understanding is fast evolving, and solutions are becoming more efficient. A recent method introduces a **"MobileGaussian"** approach for efficient 3DGS segmentation, pruning redundant splats while adding semantic info, making it suitable for mobile/AR devices [56] [57]. Such developments indicate that within a year or two, we might have lightweight models that can run on-headset to do decent scene understanding on splat representations.

For development, we'll likely implement a **semantic cleanup tool in the Utility mode** (as per spec section 4C). After scanning, the user or developer can inspect the splat scene and see identified categories. There should be an interface to manually override or clean up: e.g., if the scanner mis-labeled the fish tank as a "screen" or failed to remove a person who walked by, the user can manually erase that region or tag it to ignore. This **review step** is important for both accuracy and privacy, which leads us to:

Privacy-Preserving Capture & Sharing

Capturing one's real environment raises **privacy concerns**: both for the user and for any bystanders. We must build in **non-negotiable privacy features** when using the passthrough cameras to scan. Key requirements include: - **Automatic removal or blurring of people:** We will integrate a person segmentation model (many exist that run in real-time on mobile GPUs [58]) to detect human figures in the video frames. During the Gaussian splat creation, any splats that belong to people should be omitted or masked out. The result: the captured 3D scene will not contain a detailed model of you or your friend's bodies, or any random person who walked into view. At most, there might be a "hole" where the person was, which we can fill with a generic placeholder or simply leave empty. The system could even go further

and **dynamically redact** people in live view – e.g., if someone walks into the play space while the micro-world is running, we could either not capture them at all or render them as a sanitized ghost outline. An open-source project called "Protector" demonstrated a **real-time privacy filter for AR glasses** that *"automatically blurs faces except for those who give consent, running entirely offline."* [59]. This shows it's feasible to do on-device face/person blurring for live video. Many contributors there noted that **blurring may be reversible** with AI, so a safer approach for recorded data is to replace the person with a solid mask or fake avatar [60] [61]. We will likely implement by default a black-out or neutral gray mesh in place of any detected person to be safe.

- **Removal of sensitive objects:** Similarly, certain object categories might be sensitive – e.g., laptop screens (could show personal info), phones, paper documents with text, or house windows (which could reveal outside address). Our semantic segmentation pass can flag these (for instance, a text-detection model could find areas with printed text and we could exclude those from capture). Or simpler, we might give the user a tool to manually "erase" these items from the scan. In the spec's privacy requirements, it was noted we should allow editing utilities like erase/crop in the scene capture. So after initial scan, the user might see a preview of the splat world in the Utility mode and have the option to click on any weird or private bits to remove them. We can then either inpaint or just drop those splats entirely. An academic approach called **OmniEraser** (2023) even discusses removing objects and their visual effects from images given a mask [62] – in 3D, removing an object might involve filling in the gap behind it. For now, a rough removal (leaving a hole or generic filler) is acceptable for privacy purposes.

- **On-device processing & consent:** To alleviate user worries, as much of the capture pipeline as possible should happen **locally on the device** or a user-controlled device. If we must use cloud processing (maybe for heavy Gaussian optimization), it should be opt-in and encrypted, and results should be returned to the user then deleted from the server. We will explicitly never upload raw video or scans to a shared cloud without permission. Ideally, with advances in efficiency, we keep everything on the Quest 3 / mobile hardware. As one privacy paper noted, forcing users to upload spatial data to vendor clouds for multi-user AR is a concern [12], and our design (with local anchors or local network sharing) aims to avoid that.

- **Access controls for shared worlds:** When the user decides to share a microcosm or invite others in, we'll provide **permissions** settings. The world owner can decide if others can just view or also edit (God Mode) [63]. If the world capture includes part of your home, maybe you only share it with trusted friends (we might integrate with platform-level multi-user security: e.g., Meta's accounts/Guardian has some features for sharing anchors only with certain people). Also, if a world is exported as a package file, it should respect privacy: by default, perhaps the package does not include the raw splat data of your environment unless you explicitly include it. (Think of it like sending someone a Mario Maker level vs. sending them a scan of your room – those should be separate choices.)

Additionally, a **redaction review interface** will make the system more transparent. After scanning, the app can present: "We detected and removed 2 people and 1 screen from the scene. **Review?**" The user can then see (maybe as black silhouettes) what was removed and confirm it. This aligns with the requirement *"redaction is visible & reviewable ('here's what's being removed')"* in the spec. It's important the user trusts that no sensitive data is hiding in the captured model.

By tackling privacy from the start, we not only prevent potential data leaks but also **build user trust** which is essential if this platform technology is ever used beyond our internal development. Given our target (initially just us developers), some of these features can be developed in parallel (we don't *need* them fully for an internal MVP demo, but we definitely design the architecture to accommodate them by the time of any public release).

Agents in the Micro-World: AI Behaviors and Simulation

One of the later but exciting components is populating the micro-world with **agents** – creatures or bots that have their own behaviors, giving the world a life of its own. The spec imagines things like little autonomous villagers, or robots with "tentacle AI" that can build things, etc. This requires designing an **agent AI architecture** suitable for a tiny sandbox, with constraints on resources (since it runs on a standalone headset) but enough complexity to exhibit interesting emergent behavior.

Scope and design philosophy: We will start with *episodic, limited world-model agents* (per spec). This means an agent doesn't need a giant brain or large language model; instead, it might have: - **Perception:** the agent can query the state of nearby world (e.g., see what objects are adjacent, or if another agent is in front of it). We can give agents access to the microcosm's object graph and perhaps the semantic context ("there is water on that side", "this area is high ground"), but *not* the entire world at infinite range. - **Memory:** limited short-term memory of recent events and maybe a simple long-term state (like "I am hungry" or "I remember that spot had food before"). Episodic memory means they remember specific events for a while, but it could decay or reset each session. - **Goals/Needs:** a set of motives like hunger, curiosity, reproduction, etc., depending on the agent type. For example, a creature might have a hunger meter that increases, prompting it to seek food periodically. - **Behavior Policy:** implemented via a **behavior tree or state machine** for simplicity. Game AI commonly uses behavior trees to sequence actions and utility scores to decide what goal to pursue [64] [65]. For instance, an agent could have states like *Wander*, *Seek Resource*, *Flee Enemy*, *Sleep*. A utility AI system could score options (if hunger > 50, "seek food" gets high score; if health low, "flee" gets high score) [66] [67]. This approach can yield **emergent behavior** by simple rules interplay, without needing explicit scripting for every scenario [68] [69].

Inspiration can be drawn from **sandbox games and simulations**: The Sims, Dwarf Fortress, and Minecraft mobs – all use relatively simple AI routines that together create surprising outcomes. Even classic "boids" (flocking birds) show how a few rules lead to lifelike movement. Our agents should interact with the environment: e.g. a villager agent might pick up virtual resources (we can introduce resource objects like wood or stone blocks in the world), build structures if they have the materials, or interact socially with other agents (a greeting, trading items, etc.). These can be stubbed initially (e.g., agents randomly chat or exchange an item) and grown in complexity with each iteration.

Social and collective behavior: Since it's multiplayer, we could also allow agents to perceive players (the "small gods"). Agents might, for example, react to the giant human hand descending – perhaps fleeing or showing curiosity. If we implement that, it adds to immersion (like you truly feel like a god observing little creatures). Socially, multiple agents could form simple communities – e.g., **flocking** together, forming leader-follower relations, or dividing tasks. We can script scenario-specific behaviors (for instance, one agent could be the "builder" who tries to assemble blocks into a house, while another is a "gatherer" who brings blocks to the builder).

Given performance constraints, we should limit the number of active agents initially (maybe a few dozens at most) and keep their logic simple. A tick rate of, say, 10 updates per second per agent might be enough for smooth behavior. We'll also run AI on a separate thread if possible to not hitch rendering.

It's worth noting recent buzz around **Generative Agents** (like the Stanford paper where 25 AI characters in a small town had memories and could initiate events autonomously) [70] [71]. Those agents used large language models to simulate human-like behavior (engaging in conversations, planning a party, etc.). While fascinating – e.g., characters in a sandbox could talk and even gossip about the player – that approach is far too computationally heavy for an on-headset game (it required running GPT-3.5 in the loop). We'll keep an eye on such research, but for now, stick to lightweight heuristic agents. If we ever want richer dialogue or complex planning, one idea is to offload that to a cloud service per agent, but that's future scope and would raise new privacy issues (transmitting world state to a server for AI).

Instead, we can achieve fun outcomes with simpler means. For instance, **utility AI** (scoring system) can already yield emergent tactics – e.g., in one Reddit discussion devs noted utility AI can create spontaneous behaviors but can be hard to debug [72] [73]. We'll keep the debug tools handy (like an overlay showing an agent's current goal and perception rays, to tune their decision-making).

Agent-environment interaction: We should define how agents "sense" and "act" in the microcosm. They could have a radius around them within which they detect objects (line-of-sight or just distance). If semantic info is available (like terrain type), an agent can prefer certain terrains (maybe a water creature stays in water splat areas). Actions could include moving (with simple physics or animation), picking up/placing objects, altering terrain (if our world allows modifications like digging or building blocks), and interacting with other agents (like "attack" or "hug" depending on type!). We can borrow from game AI: e.g., a finite state machine for a predator-prey dynamic (if a "wolf" agent spots a "sheep" agent within range and no player interference, it enters chase state; if close enough, it "attacks" and maybe the sheep is removed or respawns).

Finally, since the **Agents track** is after MVP, we plan to **stub complex AI initially**: possibly just have a few scripted events in the demo (like a cute creature that runs around randomly and reacts when you pick it up). As we progress, we add more autonomy incrementally. The design will keep the agents decoupled from frame rate (so simulation can be slowed or sped up via the God Mode time controls – e.g., you could *fast-forward* the micro-world and the agents should then simulate faster). We'll also include a notion of **snapshots or determinism** for agents if possible, to allow saving and replaying the world evolution (though true determinism can be tricky with floats, we might simplify by using fixed random seeds or so when reloading a world).

In summary, the agents will bring the micro-world to life within constraints. We'll ensure their complexity grows with our ability to manage it, always keeping an eye on performance (agents will be the first to slow things down if too complex, so we'll profile and perhaps limit concurrent behaviors). The goal is a delightful simulation where, as a player or "small god," you can watch these little beings *go about their day*, respond to your interventions, and even surprise you with emergent outcomes – fulfilling the "evolving micro-world" promise.

Platform & OpenXR Considerations

Our initial target platform is the **Meta Quest 3**, primarily because of its advanced mixed reality capabilities (high-resolution color passthrough, room-scale understanding, hand tracking) and standalone convenience. The Quest 3 will serve as our development and demo device. However, we intend to keep the design as **platform-agnostic** as possible, leveraging **OpenXR** standards and cross-platform SDKs.

- **Why Quest 3 first?** It's a widely available headset with the features we need (except eye tracking). Its hardware (Snapdragon XR2 Gen2) should handle the real-time graphics and physics for a tabletop scene, and Meta provides the Shared Spatial Anchors and co-location APIs that jumpstart our multiplayer feature [4]. Also, Quest 3's mixed reality is currently among the best in class for consumers, so it's ideal for showcasing a "micro world on your table."

- **OpenXR:** We will use Unity's OpenXR plugin (or Unreal's) rather than platform-specific SDKs when possible. Meta's co-location and anchors have Unity integrations, but if we use them through OpenXR extensions or an abstraction layer, it'll ease porting later. The Meta documentation confirms OpenXR is supported for their Space Sharing features as of v76+ of the Meta SDK [74]. This means we might be able to use a common anchor interface that could later map to ARCore or HoloLens anchors on other devices. We will isolate any platform-specific code (like anchor sharing, or hand tracking differences) behind a Unity subsystem or service interface.

- **Other platforms:** In the future, adding support for devices like Apple's Vision Pro (when accessible), Magic Leap, or even mobile AR (phones/tablets) could be interesting. Our architectural choices (network model, OpenXR, cloud processing option) keep that open. For example, a future version could allow a user with just an iPad to view a microcosm on their table through the screen. We note that cross-platform co-location is an unresolved challenge (cloud anchors don't easily mix between ARKit and ARCore and Oculus etc.), but research like Glasgow's suggests ways to align disparate devices [75] [76]. We might eventually implement a cross-platform anchor calibration (e.g. have all devices scan a common AprilTag as described earlier, or have one device host and others align to it via a known object).

- **Minecraft integration ("Minecraft frame"):** The spec posed the question whether the microcosm will literally be made of Minecraft blocks or just have a **toy-world Minecraft-like vibe**. The answer leans toward actually interfacing with Minecraft (the Java edition) to **"holographically stream Minecraft"** into our experience. This is an ambitious stretch goal, but there are a couple of avenues:

    - **Live link to a Minecraft server:** We could run a headless Minecraft server (Java) representing a world. A custom mod or plugin on the server could continuously send the chunk data (block types and positions) around the area of interest to our Unity app. Unity would then instantiate corresponding blocks in the AR scene (textured cubes) matching the Minecraft world. Essentially, the Quest would become a unique *3D client* to the Minecraft server, rendering it in MR. Projects like Vivecraft have turned Minecraft into a VR experience, but here we'd be filtering to a smaller scale and blending with reality. Latency and coordinate syncing would need careful handling, but it's doable on a LAN.
    - **Import static Minecraft maps:** As a simpler step, we could take a pre-built Minecraft world or structure (there are many open .schematic or world files) and import that as our starting microcosm. This wouldn't be interactive with a running MC game, but would give the

intended aesthetic (literal blocks). There are libraries to parse Minecraft world files which we can use in Unity.
    - The phrase "holographically stream… from the Java game" suggests ideally the **actual running game** is linked. We'll likely attempt a prototype where when Microcosm XR starts, it launches a local Minecraft server with a known seed or map. The players put on Quest, and they see that world on the table. If they edit it with our tools (or via an agent), we send those changes to the MC server (maybe via commands). Conversely, if something changes in the MC world (like an AI mob moves or day turns to night), we reflect that in AR. This essentially creates a **Minecraft AR mode**. Since our focus is on the micro-world dynamics, we might keep the Minecraft world simpler (not running full survival game logic) just to avoid complexity. But it's a tantalizing prospect because it means potentially **leveraging the rich simulation of Minecraft** (redstone, creatures, etc.) inside our MR experience.
    - We will have to ensure we're not tied to any closed source or expensive tech here. Minecraft's Java server is open for modding, and we can likely achieve this with open APIs (there's a robust community of mods and the Bukkit/Spigot server API we could hook into).
    - Initially, though, for MVP, we might actually **fake the Minecraft look** by using voxel blocks and pixelated textures, without connecting to an actual MC backend. That gives the "toy-world vibe" and delight (people love the familiarity of blocks) while we work later on actual streaming integration.

- **Tabletop vs Room-scale:** We confirm that while the first target experience is a table-bound microcosm (for practicality and user comfort), the system is not inherently limited to a table. We design it such that a microcosm can be anchored anywhere – on the floor for a larger scene, or even across multiple surfaces if we get creative. Room-scale scenarios (e.g., a microcosm city that sprawls across your entire living room floor) are possible when our networking and tracking are robust. We will start with tabletop because it's easier to manage and aligns with the **MVP deliverable** (two players around a table). As confidence grows, we can experiment with switching a world to *room mode*, where perhaps the world expands and the players can walk through it at life-size (that would be more like a traditional VR or AR game, but could be an awesome mode – imagine toggling from God's-eye small view to immersive first-person view among your creatures).

- **Performance and optimization:** Quest 3 being mobile means we keep an eye on draw calls, poly count, and use of advanced rendering like splats. We've already addressed some optimization for splats (LOD, limiting capture area). For physics and networking, using Quest's capabilities like **fixed foveated rendering** can free some performance. We'll test early the CPU cost of running physics with two players and ~20 objects – likely fine – and of running the network stack (likely using Wi-Fi Direct or local networking for co-location to minimize latency).

- **Open Source tools only:** We strive to use open-source or free components: for networking, **Mirror** or **Fish-Networking** (open source) could substitute for Photon (which is partly paid) if needed. For Gaussian splatting, rely on open research code (Kerbl's, etc.) or our own implementation – avoid any proprietary SDK that charges per use. For hand tracking and eye tracking, we stick to the platform SDK which is free with the device.

All these platform considerations ensure that Microcosm XR is not a one-off for Quest 3, but a **foundation that can grow** to other devices and contexts, aligning with the OpenXR and open-source ethos.

MVP Features: Product Requirements Outline

To deliver a minimum viable but delightful experience, we define the critical feature set and acceptance criteria for **Microcosm XR MVP – "Tabletop Scene (1)"**:

1. **Co-Located Multiplayer (2 users):** Two players can join the same physical space and see the same virtual micro-world aligned on a surface.
2. *Acceptance:* After a calibration step (e.g. one user hosts and the other joins), a virtual object placed by User A on the table appears at the correct aligned position for User B within ~2cm accuracy. Both users see real-time updates of object movements with <100ms latency. If one user walks around the table, the world remains anchored in place for both.

3. *Test:* User A places a block at a marked spot on the table. User B sees it exactly on that spot in their view [4] . They both flick the block back and forth and observe smooth, minimal-lag motion.

4. **Hand Tracking Interaction:** Users can select and manipulate virtual objects using their hands (no controllers).

5. *Acceptance:* A user can perform a pinch gesture on a virtual block to grab it [16] . While pinching, moving their hand moves the block correspondingly (1:1 within a reasonable mapping). Releasing pinch drops the block, which then falls with physics. Two-hand grab of an object enables scaling: moving hands apart enlarges it (with a visible scale factor change) [17] . Rotating hands rotates the object. These interactions should be reliable – e.g., >90% of intentional pinches result in a grab, and accidental grabs are minimized by requiring thumb-index touch for >0.1s (tunable).

6. *Test:* Pick up a test cube, stack it on another. Scale a sphere using two hands and confirm size change. Try to brush past an object without pinching – it should not be picked up, ensuring system isn't too eager.

7. **Physics & Colliders:** Basic physics simulation for objects in the micro-world.

8. *Acceptance:* Objects have gravity, collide with each other and the table. If a user builds a small tower of blocks and removes the bottom one, the ones above fall and maybe tumble [77] . Physics remains in sync across network (within minor differences – acceptable if two users see slightly different jiggle but major events consistent).

9. *Test:* Build a domino line of blocks. Knock one end; blocks collide in sequence like dominos. Both users see the chain reaction occur.

10. **World Persistence (Save/Load):** Ability to save the state of a microcosm and reload it anchored to the same real location later.

11. *Acceptance:* After building a scene, the user can invoke "Save" which stores the microcosm (object positions, any agent states, anchor info). Upon "Load" in a new session, if the user is in the same room, the system realigns (using stored anchor or prompting user to scan) and the micro-world appears where it was, with all objects as left. Evolution of the world is deterministic enough that the

loaded state matches the saved layout [41] (within any non-deterministic physics randomness, which we will document if not exact).

12. *Test:* User builds a small structure. Save and close app. Reopen app, perform any necessary alignment (e.g., scan anchor QR or just auto-align via stored anchor). Verify the structure reappears correctly on the table.

13. **Example Content & Agents (Toy Worlds):** Provide 2–3 pre-made microcosm templates that are immediately fun.

14. *Acceptance:* Include scenes like a **domino run** (a series of physics props user can trigger), a **tiny village** with a couple of simple AI villagers walking around, and a **marble maze** (ramps and tunnels for a marble). These should load and run without user setup to showcase possibilities.

15. *Test:* Launch each template: Domino world – able to knock dominos. Village world – villagers move around (even if just wandering). Marble run – drop a virtual marble and it follows the track.

16. **God Mode Tools (for host/admin):** A special mode where a user can spawn or delete objects, and control simulation state.

17. *Acceptance:* In God Mode, the user has an interface (either a radial menu or a physical "tool" they grab) that lets them do at least: **Spawn** new primitives (cube, sphere, maybe a prefab tree or house), **Delete** objects (perhaps by laser or by a special grab that highlights red), and **Time Control** (pause physics and un-pause). If one user is God Mode and the other is Play Mode, both should still see changes. Only a designated admin can use these unless permissions allow others.

18. *Test:* Spawn a new block via the menu; it appears for both users [3]. Delete an object; it disappears for both and is removed from simulation. Pause time; a thrown object freezes in air.

19. **Gaussian Splat Environment (prototype):** Capture the play-space table as a splatted 3D model for occlusion.

20. *Acceptance:* The system can scan the table and immediate surroundings. The virtual content properly occludes: e.g., if a virtual character walks behind a real vase that was captured, it disappears from view behind the vase [39]. The splat model need not be perfect for MVP, but should at least resemble the real surface and block visuals accordingly. Performance should stay interactive (aim 30-60 FPS with the splat in view).

21. *Test:* Place a real object (like a shoebox) on the table. Scan it. Then place a virtual block and slide it "behind" the box relative to user view – verify the block gets hidden by the box's splat. Check frame rate with splat rendering on.

22. **Privacy Filter (basic):** During capture, do not include people in the splat model.

23. *Acceptance:* If a person stands next to the table during scanning, the resulting model does **not** show a recognizable person shape. Ideally it shows nothing (hole) or a very blurred ghost. Also, if the user's own body/hands appear in some scan frames, those are not reconstructed as scenery. We also

ensure that if the user shares the world, any removed elements (people, etc.) truly are absent from the data.

24. *Test:* Have someone walk through the area while scanning. Inspect the model (perhaps via a debug view or by seeing if any points appear mid-air where the person walked). Expect no significant reconstruction of the person.

These represent the MVP criteria – achieving them would realize a 10-minute demo where two players **co-create and play in a tiny persistent world** as described in the spec. Each feature above will be thoroughly tested in isolation and in combined use.

Two-Week Research Sprint Plan

Finally, to ramp up our knowledge and validate tech choices, we propose a focused **2-week research sprint** before deep development. This sprint will involve literature review, prototyping with must-try SDKs, and small experiments. The goal is to answer open questions (especially around co-location, splatting, and interaction) with concrete data or demo code.

Week 1: Investigation & Rapid Prototyping

- **Co-Location & Networking (Days 1-2):**
- Read *"A Quest for Co-Located MR (VRST 2020)"* by McGill et al. [9] [75] to understand platform-agnostic alignment approaches.
- Try Meta's **Shared Spatial Anchors** in a Unity sample [78] [4] – run their Space Sharing demo on two Quests in our space to gauge accuracy and latency. Acceptance criteria: get a cube anchored and visible to two headsets correctly.
- Explore **AprilTag** alternative: use an OpenXR extension or ARToolkit to locate a printed AprilTag in Unity on Quest. See if we can compute an anchor from that (likely requires writing a small script). This will inform if we can have a fallback method without internet [10].

- *Output:* Brief document on best co-location approach, and a simple Unity scene where two Quests share an anchor (or if second Quest not available, simulate by moving one and re-localizing).

- **Hand Interaction UX (Day 3):**

- Read Meta's Interaction SDK docs on **Hand Grab** and **Poke** interactions [77], and any user study like "DOF-Separation for 3D Manipulation in XR" (Mikkelsen et al. 2025) [20].
- Prototype in Unity using XR Interaction Toolkit: set up a grabbable cube with XR Hands. Tweak grab sensitivity and test two-hand scaling on an object.

- *Output:* A test scene on Quest 3 where we can grab and throw a cube and scale a cube, to evaluate tracking stability. Note any issues (e.g., hand occlusion causing drop, etc.) and document mitigation ideas.

- **Eye Tracking Possibilities (Day 4):**

- Since Quest 3 has no eye tracking, allocate a half-day to study what we'd do when we have it. Read *"Design Principles for Gaze + Pinch"* [27] [79] to glean design guidelines.

- If possible, use a VR headset with eye tracking (maybe a Vive Pro Eye or Quest Pro if we can borrow) to run a quick demo: many SDKs have a "gaze pointer" sample. If unavailable, skip direct prototype but note how we'd integrate later.

- *Output:* Design notes on how gaze data is accessed via OpenXR, and a list of potential gaze interactions for future.

- **Gaussian Splat Familiarization (Day 5-6):**

- Read *3D Gaussian Splatting for Real-Time Rendering* (Kerp et al. 2023) introduction to grasp basics [39].
- Set up the **Toy Gaussian Splat Unity plugin** by Aras Pranckevičius (GitHub) – this plugin can load a precomputed splat model [80]. Acquire a sample splat file (maybe from the authors or a public dataset).
- Try rendering it on Quest or PC, observe performance and quality. Particularly, see how many splats can be drawn at 72fps on Quest. If plugin isn't readily usable, at least run on PC to gauge quality.
- Research Niantic's approach: read Scaniverse blog/FAQ [47] [48] for how they optimized for Quest (notes on resolution, etc.).
- *Output:* A short report on the feasibility of splat rendering on Quest 3 – including any bottlenecks found (fill rate, memory). Also compile any open-source repos for splatting (Kerbl's code, etc.) for later integration.

Week 2: Deep Dives & Integration Planning

- **Semantic Segmentation & Scene Understanding (Day 7):**
- Read *"Beyond Averages: Open-Vocabulary 3D Scene Understanding with GS"* (Arafa 2025) abstract to understand pitfalls [50] [52].
- Investigate available tools: e.g., **SAM (Segment Anything Model)** for 2D – test it on a photo of a table scene to see if it can pick out table, objects, person. Also look at any 3D segmentation libraries (maybe PointNet for point clouds, etc.).

- *Output:* Decide on an approach for MVP semantic labeling: likely using 2D segmentation on key frames and transferring to 3D. List of labels to focus on (person, table, screen, floor). Possibly a small script that runs SAM on an image and highlights those.

- **Privacy & Security Review (Day 8):**

- Research regulations/guidelines: what does Meta or others say about captured camera data? (e.g., read Meta's privacy policy snippet for Quest Pro camera [81] – find if they mention on-device processing).
- Check **Protector** on GitHub [82] – see what model it uses for face blurring and performance. Maybe try running its model on a PC with a sample image to estimate if Quest could handle it.

- *Output:* A checklist of privacy features to implement (face blur, sensitive object removal, user consent screens, etc.), and confirmation of technical feasibility (like a particular ML model that can run 15fps on Quest for person segmentation).

- **Multi-Agent Systems Brainstorm (Day 9):**

- Read summary of Stanford *"Generative Agents"* to get ideas for architecture (though we won't replicate it) [70].
- Look at classic Game AI: skim a chapter from *Game AI Pro* or a GDC talk on behavior trees vs utility AI [83] [69].

- *Output:* Outline the agent system design (senses, states, update loop). Also identify one **must-try prototype**: e.g., implement a simple flocking or wandering behavior for a capsule object in Unity and see it move around. This prototype can be very simple but will let us see how to integrate agent updates with Unity's frame loop.

- **Prototype Integration (Day 10-12):**

- Now combine some pieces: integrate hand interaction with networking: modify the Day 1-3 prototypes so that in a shared space, hand movements by user A moving an object replicate to user B (using basic Unity Netcode for GameObjects or even a quick Photon room if easier).
- Also, test physics sync: drop objects and see if both see them fall similarly. If divergence is too high, consider enabling periodic state corrections or just note it for future networking tuning.
- If time permits, integrate the splat prototype: maybe capture a few camera frames from Quest manually, run a quick offline process to get a splat (even a dummy one), and display it behind some virtual objects to test occlusion ordering.

- *Output:* A "MVP slice" demo running locally: perhaps one headset, but simulating two users (or if we have two Quests, actual two-user test). This demo would show: hand grabbing and moving a cube, cube is network-synced (or we fake the second user by just recording the transform), and a static real-world backdrop (could even be a rough point cloud from ARCore as placeholder) occluding the cube when appropriate. Essentially a vertical slice to prove core concepts work together.

- **Evaluation & Paper Checklist (Day 13-14):**

- Write up findings for each query area: Co-location, Interaction, Gaze, Splatting, Semantics, Privacy, Agents – noting what looks solvable and what needs more R&D. Make sure to highlight any "unknown unknowns" discovered.
- Compile a list of **key papers/libraries** with links (for future reference, e.g., Kerbl 2023 paper/code, Cognitive3D if we consider analytics, OpenXR extensions needed, etc.).
- Plan the next steps after research: assign tasks like "Implement anchor sharing using Meta API" or "Train small segmentation model for Quest".

By the end of this sprint, we expect to have confidence in the technical path and have prototype code to springboard into actual development. We will have also gathered the *must-read papers* (some cited above) and *must-try repos* (e.g., open-blocks for interaction, Aras's splat renderer, Mirror networking, etc.) as part of the outputs, ensuring we're not reinventing wheels that the community has already built [44] [56].

With the research sprint insights and the PRD feature list, we will be well-equipped to proceed with building Microcosm XR, turning the North Star vision into reality – a shared, persistent, smart little world on every table, realized with today's technology and tomorrow's imagination.

Sources:

- Meta Quest Shared Spatial Anchors & Co-Location – *Meta MR Dev Guide* [4] [6]
- McGill et al. 2020 – *Aligning & Assessing SLAM Tracking for Co-Located MR* [9] [75]
- Mishra et al. 2025 – *AprilTags Local Shared Space (no cloud)* [10] [14]
- Pfeuffer et al. 2024 – *Design Principles for Gaze + Pinch Interaction* [27] [79]
- Cognitive3D Analytics – *Gaze Heatmaps & Attention in 3D* [31] [32]
- Shukhratov & Gorinsky 2025 – *Rapid 3D Capture with Gaussian Splatting (Unity @150fps)* [41] [42]
- Niantic Scaniverse FAQ – *Gaussian Splatting on Quest (optimizations)* [48]
- Abdalla Arafa 2025 – *Open-Vocabulary 3D Scene Understanding with GS* [50] [52]
- Reddit AR thread – *Real-time Privacy Filter (blurring faces)* [59] [61]
- Game AI Discussions – *Utility AI for Emergent Behaviors* [83] [69]

---

[1] [2] [9] [75] [76] eprints.gla.ac.uk
https://eprints.gla.ac.uk/224062/1/224062.pdf

[3] [4] [5] [6] [7] [8] [74] [78] Meta for Developers
https://developers.meta.com/horizon/documentation/unity/unity-mr-utility-kit-space-sharing

[10] [11] [12] [13] [14] [15] AprilTags in Unity: A Local Alternative to Shared Spatial Anchors for Synergistic Shared Space Applications Involving Extended Reality and the Internet of Things - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC12301001/

[16] [20] [21] [22] [23] pure.au.dk
https://pure.au.dk/ws/files/433925553/DOFSeparation_ISMAR25.pdf

[17] [24] [PDF] An easy Hand Gesture Recognition System for XR-based ... - optimai
https://optimai.eu/wp-content/uploads/2023/05/An_easy_Hand_Gesture_Recognition_System_for_XR-based_collaborative_purposes.pdf

[18] [19] [27] [28] [29] [30] [35] [36] [37] [38] [79] eprints.lancs.ac.uk
https://eprints.lancs.ac.uk/id/eprint/223408/1/CGA_24_Design_Principles_2_.pdf

[25] [77] Hand Grab Interactions | Meta Horizon OS Developers
https://developers.meta.com/horizon/documentation/unity/unity-isdk-hand-grab-interaction/

[26] Comparison of hand tracking-based and controller-based interaction ...
https://link.springer.com/article/10.1007/s10055-025-01190-5

[31] [32] Analyze Aggregate Insights Across XR Sessions | Cognitive3D
https://cognitive3d.com/product/analyze/

[33] [34] Real-Time 3D Heatmap Visualization and Collaborative Gaze Analysis in XR Environments | Springer Nature Link
https://link.springer.com/chapter/10.1007/978-3-031-94150-4_2

[39] [40] [41] [42] [43] [45] Capture and Interact: Rapid 3D Object Acquisition and Rendering with Gaussian Splatting in Unity
https://arxiv.org/html/2510.06802v1

[44] 3D Gaussian Splatting for Real-Time Radiance Field Rendering - Inria
https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

46 47 48 49 Into The Scaniverse | 3D Gaussian Splats on Meta Quest
https://scaniverse.com/quest

50 51 52 53 54 55 [2509.12938] Beyond Averages: Open-Vocabulary 3D Scene Understanding with Gaussian Splatting and Bag of Embeddings
https://arxiv.org/abs/2509.12938

56 Open-Vocabulary 3D Semantic Segmentation - Emergent Mind
https://www.emergentmind.com/topics/open-vocabulary-3d-semantic-segmentation

57 MobileGaussian: Efficient 3D Gaussian-based Open Vocabulary ...
https://dl.acm.org/doi/10.1145/3744451.3744463

58 [PDF] Full-body Anonymization with Human Keypoint Extraction for Real ...
https://arxiv.org/pdf/2408.11829

59 60 61 63 82 Real-time camera privacy filter for smart glasses : r/augmentedreality
https://www.reddit.com/r/augmentedreality/comments/1mnwrih/realtime_camera_privacy_filter_for_smart_glasses/

62 OmniEraser: Remove Objects and Their Effects in Images with ...
https://arxiv.org/html/2501.07397v3

64 66 Utility AI / restructuring the AI system - Thrive Development Forum
https://forum.revolutionarygamesstudio.com/t/utility-ai-restructuring-the-ai-system/919

65 69 Game AI Planning: GOAP, Utility, and Behavior Trees
https://tonogameconsultants.com/game-ai-planning/

67 83 [PDF] Building Utility Decisions into Your Existing Behavior Tree
http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter10_Building_Utility_Decisions_into_Your_Existing_Behavior_Tree.pdf

68 72 Utility AI for extra challenge? : r/gamedesign - Reddit
https://www.reddit.com/r/gamedesign/comments/17r30c7/utility_ai_for_extra_challenge/

70 Generative Agents: Interactive Simulacra of Human Behavior - arXiv
https://arxiv.org/abs/2304.03442

71 Generative Agents: Interactive Simulacra of Human Behavior
https://dl.acm.org/doi/fullHtml/10.1145/3586183.3606763

73 Utility AI (Discussion) - Unity Discussions
https://discussions.unity.com/t/utility-ai-discussion/727280

80 Toy Gaussian Splatting visualization in Unity - GitHub
https://github.com/aras-p/UnityGaussianSplatting

81 Here's What Meta Says About Camera Privacy on Quest & Quest Pro
https://www.roadtovr.com/oculus-quest-camera-privacy-rift-s-facebook/