

SplatXR: A Toolkit for Multi-Layered Extended Reality via On-Headset 3D Gaussian Splatting

Cyrus Vachha
Princeton University
Princeton, NJ, USA
cvachha@princeton.edu

Mohamed Kari
Princeton University
Princeton, NJ, USA
mokari@princeton.edu

Parastoo Abtahi
Princeton University
Princeton, NJ, USA
parastoo@princeton.edu



Figure 1: *Left:* A developer composes an XR app using our SPLATXR toolkit in Unity. *Mid:* The resulting app can be deployed to a standalone Meta Quest 3 or Apple Vision Pro headset, featuring several toolkit capabilities for a rich, interactive user experience. *Right:* The toolkit leverages a set of custom SPLATXR representations.

Abstract

Recent advances in Gaussian splatting, 3D foundation models, and video-passthrough mixed reality offer unprecedented fidelity for capturing and blending physical and virtual worlds. However, established XR methods rely on polygon-based pipelines that treat physical and digital environments as separate domains, making seamless integration across recent novel representations difficult. Because splats lack explicit surface geometry, they impede standard graphics techniques necessary for cross-reality interactions, such as shadow or collision handling. Furthermore, while 3D foundation models can reconstruct physical objects on-the-fly into digital splats, current XR systems fall short of leveraging this capability for user experiences that transition seamlessly between physicality and virtuality. To address this, we organize XR as a multi-layered reality of interacting primitives, letting pixels and objects move across physical and digital boundaries. We contribute SPLATXR, a Unity toolkit that operationalizes this structure as building blocks for interactive, immersive apps leveraging 3D Gaussian splatting: a standalone renderer, splat-to-mesh reconstruction, splat physics, splat blending, and offloaded splat object reconstruction via on-headset capture. SPLATXR unites vision and graphics techniques to make splat-based XR development broadly available, and we validate it through demonstrative applications and a small developer study.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

UIST '26, Detroit, MI, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2856-3/26/11

<https://doi.org/10.1145/3830398.3830681>

CCS Concepts

• **Human-centered computing** → **Interactive systems and tools**; **Human computer interaction (HCI)**.

Keywords

Augmented Reality; Virtual Reality; Mixed Reality; Gaussian Splatting; XR Toolkit; Artifact

ACM Reference Format:

Cyrus Vachha, Mohamed Kari, and Parastoo Abtahi. 2026. SplatXR: A Toolkit for Multi-Layered Extended Reality via On-Headset 3D Gaussian Splatting. In *The 39th Annual ACM Symposium on User Interface Software and Technology (UIST '26)*, November 02–05, 2026, Detroit, MI, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3830398.3830681>

1 Introduction

In recent years, three fundamentally new ways of *reconstructing, representing, and rendering a scene* have emerged. First, vision foundation models such as SAM 3 [12] and SAM 3D [77] have made it possible to gain a detailed understanding of 3D scenes and their semantics. Second, differentiable rendering techniques such as neural radiance fields [50] and 3D Gaussian splatting (3DGS) [34] have enabled capturing complex 3D scenes with photorealism. Third, splat- or video-based world modeling (e.g., World Labs [37]) has the ability to simulate visual 3D worlds directly from the physical world, enabling the generation of interactive 3D environments from minimal visual input. All of these novel techniques enable the creation of 3D worlds at unprecedented fidelity grounded in physical reality by leveraging implicit representations.

At the same time, a new generation of video-passthrough head-worn displays, such as the Meta Quest 3, Apple Vision Pro, and Samsung Galaxy XR, has pushed the limits of *displaying a scene* and the achievable visual integration of physical-digital scene compositions. These extended reality (XR) devices leverage perspective reprojection and high-resolution displays to obtain scenes that seamlessly blend virtual elements with physical reality. However, despite the technical innovation underlying these new XR displays, their applications are still fundamentally based on and designed for displaying explicit polygon mesh representations, used for decades in computer graphics, as they provide visual information for rendering and geometric information for collisions, shadows, and occlusions.

This mismatch between the explicit representations in traditional polygon meshes and the implicit representations driving today’s computer vision and graphics innovations has hindered the adoption of the latter for interactive purposes in XR. With this work, we introduce a toolkit that is organized around a multi-layered composition that accounts for the complementary strengths and shortcomings of explicit and implicit scene representations, all while ensuring ease-of-use for XR application developers. In particular, we focus on 3DGS as it is at the core of many graphics innovations, also serving as a common interface to vision foundation models and world models (e.g., SAM 3D [77] can export its reconstruction as a splat, and Marble by World Labs [37] can also export its generation as a splat).

We organize these physical and digital layers into a manageable compositional structure, given that perceived reality is a composite of various structural, visual, and semantic components. By treating reality as a set of distinct but interacting layers across different representations (physical reality, Gaussian splats, environmental meshes, scene semantics), our framework allows developers to design cross-reality interactions where pixels and objects can move between these layers. To operationalize this, we lay out several example configurations that recur across different layers of reality. For instance, structural layers can influence visual layers through proxy phantom meshes, i.e., invisible geometries reconstructed from the physical environment which let virtual splats cast shadows and appear to collide with real-world surfaces. We showcase configurations for *semantic linking* across layers, demonstrating how inputs, such as hand tracking, can be used to anchor and manipulate digital splat objects. This provides developers with the mental model and technical foundation necessary to build experiences that bidirectionally interact with the real environment and captured or generated 3D scenes.

We contribute SPLATXR, an end-to-end Unity toolkit for building XR applications with Gaussian splats on standalone, untethered headsets, integrating real-time splat rendering, physics, and developer-facing abstractions. SPLATXR thus facilitates an integrated developer workflow that brings together and adapts computer vision and graphics techniques in one toolkit so that building XR applications with Gaussian splatting becomes available to the wider XR community, from research to development. Through building blocks, SPLATXR automatically handles proxy geometry computation, Gaussian splat materials, and render queue ordering, maintaining visual and physical consistency across splats, meshes, and proxy geometry without additional developer input. Unlike prior splatting work confined to VR, SPLATXR also enables AR by

simultaneously running wire- and server-free for unbounded outdoor use and providing AR-specific support, including passthrough compositing, shader-based alpha blending, color correction, and occlusion from reconstructed environment geometry. The toolkit also enables interactive reconstruction of objects directly from the headset, offloading compute to a desktop GPU running generative methods, so developers can integrate realistic digital object twins into applications to bridge physical and digital spaces. We showcase applications built with SPLATXR demonstrating how different elements are linked semantically between layers and interact with one another. A technical evaluation of the toolkit features and renderer shows the efficacy of the implementation, and a small preliminary developer evaluation (n=3) provides initial insights into the toolkit’s usability. SPLATXR and our technical evaluation setup are open sourced and available online¹.

2 Related Work

SPLATXR is a toolkit that enables developers to leverage Gaussian splats of captured physical objects and spaces for interactive XR applications. Its contributions therefore relate to cross-reality interaction, capture-based XR experiences, neural scene representations in XR, XR toolkits, and the emerging space of Gaussian splatting software tools and libraries.

2.1 Cross-Reality Interactions

Cross-reality interactions have been conceptualized in a variety of ways to support different research or development purposes. Cools et al. [15] examine the compositional structure of cross-reality interactions for design patterns. Auda et al. [6] survey cross-reality systems along the reality-virtuality continuum. Blending Spaces [14] transitions between physical and virtual layers on an object level, and RealityCheck [24] does so on a pixel level. GradualReality [69] further incorporates interaction state awareness. All of these works assume—even if implicitly—that the virtual world consists of polygon meshes and thus forms a single coherent layer. Therefore, they do not account for the challenges that arise in using Gaussian splatting for interactive and situated applications, e.g., ensuring lighting coherence or supporting physics simulations between explicitly and implicitly represented content. We instead implement a Unity toolkit that accounts for both.

2.2 Capture-Based XR Experiences

Several smartphone-based apps scan physical objects into 3D representations for downstream XR applications. While meshes have predominated, recent products from Luma [2], Scaniverse [67], and Polycam [59] export splats; SPLATXR expects developers to provide a splat created with such an app.

Captured 3D models underpin a multitude of XR experiences: remote robot operation in VRoxy [65], environmental design prototyping in PointShopAR [84], redirected walking in SwitchAR [87], collaboration in VirtualNexus [28], interface personalization in MineXR [13], educational experiences in Museum in the Classroom [47], memorable item interaction in InteRecon [43], and animating the physical environment in Scene Responsiveness [33]. SplatOverflow [36] anchors annotations to splats for asynchronous

¹<https://splatxr.github.io>

hardware troubleshooting, and Dreamcrafter [80] edits and creates VR scenes from captured and AI-generated splats. Others virtually represent physical objects with hand-designed 3D models [21, 58]. Beyond specific experiences, XR-Objects [17] contributes MLLM-driven interactions with everyday objects, and Programmable Reality [75] outlines a vision for programmable AR. Foundational depth reconstruction and tracking methods such as KinectFusion [29] reconstruct the environment geometry that can be used for occlusion and physics. These works demonstrate modular components or potential use cases that motivate SPLATXR’s goal to simplify splat integration into XR development and to provide toolkit features for recurring problems such as object import, rendering, and interaction.

2.3 Implicit Scene Representations in XR

The use of splats and other implicit scene representations in XR has ranged from rendering to interaction. VRsplat [79] and VR-Pipe [39] tackle the challenges of rendering splats under headset constraints, MetalSplatter [49] enables splat rendering on Apple Vision Pro, and Snap’s Lens Studio [72] does the same on Snap Spectacles. VR-GS [30] deforms otherwise static splats for interaction purposes, and GS-Verse [57] similarly enables single-object soft-body deformation by building on the mesh-based GaMeS representation [83]. However, both are tethered and VR-only, targeting neither passthrough AR nor an easily importable toolkit. Reality Promises [32] renders untethered scene and object splats, but does not offer a modular toolkit or mesh-splat interactions. Li et al. [42] render neural radiance fields (NeRFs) on a headset with simple transform manipulations, Instant-NGP [52] allows VR viewing of NeRFs and cleaning of floaters, and Splatshop [68], VR-Doh [46], and GaussianShopVR [70] facilitate editing splats in VR. GaussianNexus [27], CoCreateAR [54], Reality-Linked Gaussian-Splatted Proxies [89], Thing2Reality [26], and SharedNeRF [66] implicitly represent objects of interest across spatially distributed users for telepresence and remote collaboration. MiXR [20] similarly applies scanned-object interactions to a specific domain (generative-aided design) and, like GaussianNexus, targets a particular use case rather than a general-purpose XR toolkit. Additional applications of splatting and NeRFs in XR are discussed by Qiu et al. [62] and Li et al. [41], who survey various use cases for radiance fields.

Like prior work, SPLATXR builds on several existing splatting techniques for on-headset rendering, and, to this end, unifies and packages established algorithms into a single, coherent XR toolkit for developers, adapting techniques as needed for rendering splats in an HMD. Table 1 provides a summary of existing systems. SPLATXR differs by (1) lowering the barrier to splat-based XR development by uniting computer vision and graphics techniques, (2) enabling AR-specific support, unlike prior work, which largely remains VR-only, and (3) providing reusable components for recurring XR tasks.

2.4 XR Developer Toolkits

Toolkits constitute an integral part of HCI research [38, 53, 56], and a number of XR toolkits aim to simplify the XR development workflow. DART [22, 48] seminally presented foundational concepts and techniques for prototyping AR experiences. Unity-based toolkits target data visualization (DXR [71], Miria [11]), remote VR studies (RemoteLab [40]), UI adaptation (AUIT [7]), accessibility for

Table 1: Splatting and XR systems compared to SPLATXR. AR denotes physical-digital compositing on headsets. *Geom.* denotes geometry awareness for shadows, occlusions, and collisions. *Splat* denotes rendering of 3DGS. *Toolkit* denotes the availability of a package that can be simply imported into an existing project.

	Untethered	AR	Geom.	Splat	Toolkit
UnityGaussianSplatting [61]	–	–	–	✓	–
Ninjamode Unity GS [35]	✓	–	–	✓	–
GS-Verse [57]	–	–	–	✓	–
VR-GS [30]	–	–	–	✓	–
GaussianNexus [27]	–	✓	–	✓	–
MiXR [20]	✓	✓	✓	–	–
Reality Promises [32]	✓	✓	–	✓	–
SPLATXR (ours)	✓	✓	✓	✓	✓

blind and low-vision users (SeeingVR [92]), cross-device interaction (XDTK [23]), and distributed collaboration authoring (XSpace [25]). DepthLab [18] provides an ARCore-based toolkit for working with depth across application-specific representations, focusing on mobile AR rather than on-headset splatting, and Palimpsest [44] anchors spatial UI to physical objects in Vision Pro.

SPLATXR draws inspiration from these works in both their research processes and artifacts: we derive design objectives, outline our foundational concepts and techniques, and then present demonstrative applications.

2.5 Gaussian Splatting Tools

Since the inception of splatting [34], a variety of tools have emerged that seek to satisfy different purposes.

Training and rendering splats is at the core of Kerbl et al.’s original implementation [34], Brush [10], and Nerfstudio’s gsplat library [76, 90]; these low-level implementations require substantial domain expertise and are incompatible with XR engines such as Unity, Unreal, or WebXR. Postshot [60], Luma [2], Kiri Engine [19], and Polycam [59] simplify training for non-expert users through a higher-level user interface, but are self-contained applications instead of developer dependencies.

In-browser viewing and editing of splats motivates tools such as SuperSplat [45, 74], web renderers [4, 51], Nerfstudio’s viewer [76], Gracia [1], Niantic’s Scaniverse [67], Spline [73], Sparkjs [88], and Arrival.Space [5]. Built on the low-level WebGL API [85], on general-purpose 3D engines such as Three.js [78], or on special-purpose visualizers such as Viser [91], these tools support viewing and, in some cases, pre-defined editing; Gracia and SuperSplat go further with stereoscopic viewing, and other web-based experiences reach the headset via the WebXR API [86]. Their design nevertheless remains centered on viewing, and their browser-based architectures are incompatible with native XR development in engines such as Unity and Unreal.

Integrating splats into a 3D engine is facilitated by platform-specific engine plugins: Luma [2] and Volinga [82] for Unreal, NVIDIA Omniverse [55] for robotics simulations, the GSOPs plugin [64] for SideFX Houdini, and ReshotAI’s plugin [63] for Blender.

All of these focus on editing and compositing splats but lack proper tooling for building interactive experiences with them. Pranckevičius’ splat renderer for Unity [61] follows a PC-VR architecture requiring a tethered connection and it offers no XR-specific features, making it unsuitable for standalone deployment. Ninjamode [35] likewise provides a Unity splat renderer. Like the authoring tools GaussianShopVR [70] and Splatshop [68], it offers no splat physics, shadows, or AR blending, and several of these tools remain tethered.

In summary, while these tools are useful for different purposes, none addresses the need for building interactive XR applications. To this end, SPLATXR introduces a Unity-based toolkit, designed for standalone, untethered headsets including Meta Quest 3 and Apple Vision Pro, that provides real-time performance while leveraging game engine features to enable immersive experiences, simplifies recurring tasks such as preprocessing and importing splats, and provides building blocks for rendering and user-splat interactions.

3 Multi-Layered Reality Framework

Enabling inter-object and human-object interactions between splats, classical meshes, and the physical scene requires systematically managing these entities on both a technical and a conceptual level. To this end, we describe a multi-layered framework that conceptualizes reality as a set of logically isolated, yet operationally intertwined layers that align visible and invisible splat and mesh geometry with associated semantics, appearance, and physical context. In the following, we outline the proposed set of layers and the layer-bridging operations defined on them.

3.1 Reality Layers

Our framework uses five primitive layers:

3.1.1 Physical layer. The physical layer comprises the user’s environment, visually captured and displayed through the passthrough feed of the XR headset.

3.1.2 Photometric layer. The photometric layer refers to appearance-related components of a scene, such as splats or a rendered frame, which may lack explicit structure. Appearance can be obtained from captures of real spaces or generated by a 3D foundation model. Its content comes from the rendering engine, which provides the rendered view defined by the headset’s intrinsics and extrinsics.

3.1.3 Geometric layer. The geometric layer includes traditional 3D polygon objects which possess explicit geometry that can be rendered visually while interacting with the scene’s physics.

3.1.4 Phantom structure layer. The phantom structure layer contains invisible explicit geometries, meshes in our case, that are isometrically anchored or aligned to visual layers, letting a visual splat cast a shadow or a physical table support a virtual bouncing ball. Such invisible proxy geometry is a standard graphics technique; we name it as a layer here to make its role in splat-based composition explicit.

3.1.5 Semantic layer. The semantic layer encompasses a higher-level understanding of element function and identity in reality via passthrough (e.g., hand tracking or object recognition) to enable cross-layer interactions like linkage and scene understanding.

3.2 Operations

While layers define the static compositional structure of a mixed-reality environment, operations enable the dynamic transitions and interactions between them. Fig. 2 depicts possible configurations and operations, which we categorize into three primary types: transformation, blending, and linkage.

3.2.1 Cross-layer transformation. Elements can be transferred across layers (Fig. 2f). To move an element from the physical environment to a digital representation, a region of the passthrough feed (physical layer) is processed by a generative model such as SAM 3 or SAM 3D Objects, which lifts 2D visual data into an implicit 3D structure and decodes it into an explicit 3D Gaussian splat (photometric layer). To let particle-based splats interact meaningfully, we rely on a splat-to-mesh operation which derives an invisible phantom mesh (phantom structure layer) from the splat file and aligns the visual splat to it, yielding a mesh-augmented splat that can physically collide, cast shadows, and interact with classical meshes and other scene layers.

3.2.2 Layer blending. Operations also control how layers are composited within the user’s view. Additive operations selectively stack and reveal layers using spatial masks and custom splat blending shaders, as shown in Fig. 2e. This enables volumetric portals: the user sees their raw physical surroundings everywhere except within a defined boundary, through which an imported splat environment becomes visible, blending smoothly with the surrounding passthrough as the viewpoint shifts. Fig. 2a shows an example of diminished reality using a reality blending volume covering a real object, and Fig. 2b shows an instance of mediated reality, which inserts a modified version of that object into the physical space as a photometric layer.

3.2.3 Linkage. Whereas the operations above are transitions, linkage defines persistent connections that maintain spatial and contextual coherence in a mixed-reality scene by binding digital elements to the physical environment. We categorize these into two types.

Geometric Linkage. This establishes a spatial relationship between layers, anchoring virtual elements to specific coordinates or distinct physical features. A possible implementation is AR image target tracking, where a virtual splat or phantom mesh is persistently locked to the position and orientation of a physical marker, ensuring static spatial alignment across the physical, geometric, and photometric layers.

Semantic Linkage. Beyond rigid coordinates, semantic linkage dynamically binds a layer to contextual elements in physical reality, requiring a real-time understanding of the user or environment. In Fig. 2d, hand-tracking data from the physical reality layer links a virtual watch splat to the user’s wrist, ensuring continuous positioning and enabling behaviors like dynamic occlusion from the geometric layer.

4 SPLATXR Toolkit

In the following section, we first outline SPLATXR’s toolkit architecture and developer experience which together aim to provide developers with a set of *building blocks* that facilitate developing a splat-enhanced end-user experience. We then detail the developer workflow and implementation for each of these building blocks, and

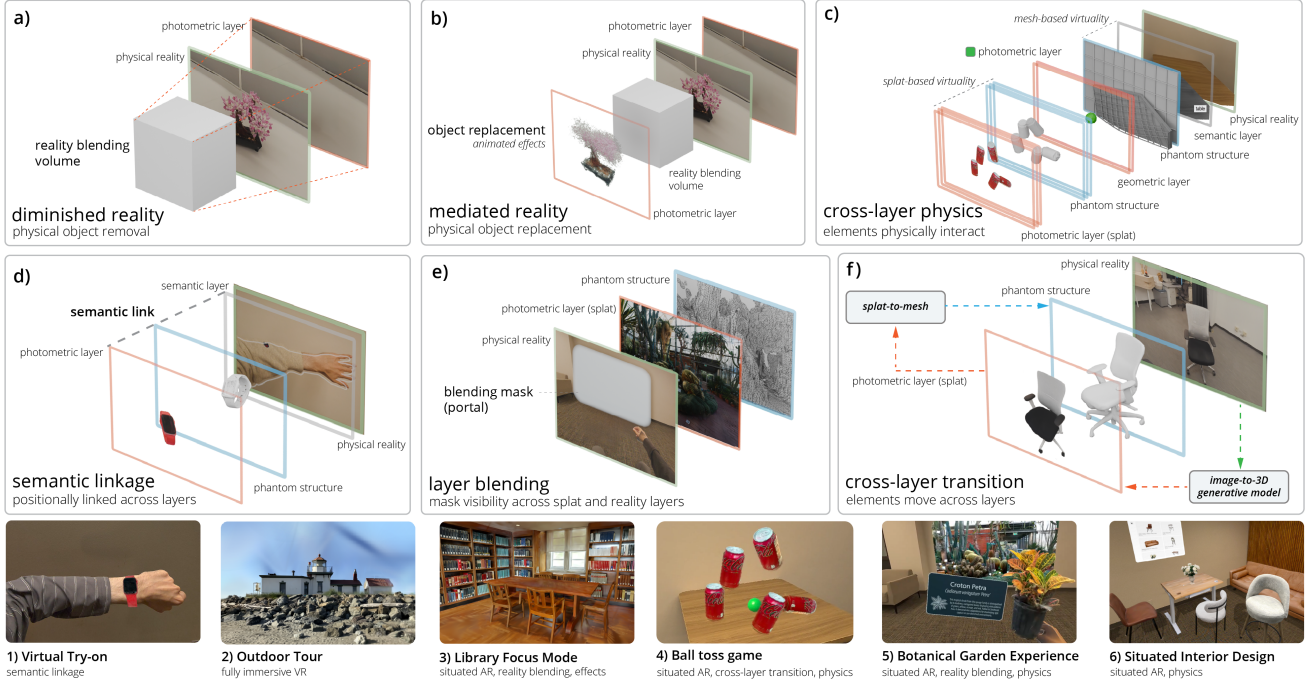


Figure 2: Configurations and operations. *Top*: Different multilayered reality operations across splats, meshes, and reality showcasing cross-layer transitioning, physics, linkage, and blending. *Bottom*: Demonstrative applications that use the above operations to enable XR experiences.

finally present SPLATXR’s capability for facilitating PC-offloaded 3D object reconstruction from the passthrough video feed.

4.1 Architecture and Developer Experience

SPLATXR is implemented as a Unity package, thus making it easily installable through Unity’s Package Manager. It features a developer-facing graphical interface, integrated into the Unity Editor, that provides access to a set of well-defined building blocks through panels, scripts, materials, and shaders. Building blocks can be implemented as needed, and previewed interactively in the Unity Editor scene view. Notably, this makes SPLATXR’s splat renderer compatible with non-XR applications as well.

4.2 Building Blocks

SPLATXR comprises four main building blocks:

- B1:** a *Splat Renderer* building block enabling splat rendering on XR headsets in passthrough AR or full VR with computational constraints
- B2:** a *Splat Physics* building block based on our mesh augmentation procedures, enabling both shadows, cast from splats onto the environment, and rigid-body collisions between splats, classical meshes, and the environment
- B3:** a *Splat Effects* building block based on customizable shader programs, enabling compelling visual experiences such as object dis- and re-integration
- B4:** a *Splat Blending* building block enabling blending between the physical scene and the virtual scene in AR

Building blocks can be accessed by developers via the *developer interface*, which is implemented as a dedicated panel in the Unity Editor. Actions performed in the developer interface add or modify assets in the project folder as well as game objects and attached components in the scene hierarchy. We show our editor interface in more detail in Appendix A.1.

4.2.1 B1: Splat Renderer. The *Splat Renderer* building block comprises a 3D Gaussian splat rendering pipeline designed for compatibility with standalone XR devices, including Meta Quest and Apple Vision Pro. On both Quest and Vision Pro, it can render fully immersive splat-based VR scenes, and on Quest, it can render splats and blend them with the passthrough layer for situated AR experiences.² It can be executed untethered on standalone headsets or tethered to a PC. In our multi-layered reality framework, this represents the photometric layer.

Developer Workflow. To leverage the renderer, developers simply import a splat into a Unity scene by making use of the SPLATXR Importer panel. Within the panel, they choose a standard splat ply file, which they either create themselves or download from a third party. Upon import, the plugin automatically adds a game object to the scene hierarchy, applies a predefined splat material associated with our splat shader, and adds a splat rendering script to the game object. The renderer not only works properly on-headset but is also aware of when it renders in-editor, switching to the correct editor camera and rendering parameters accordingly. This ensures faithful

²Apple Vision Pro is supported, but passthrough shader functionality is currently limited by Apple’s PolySpatial platform.

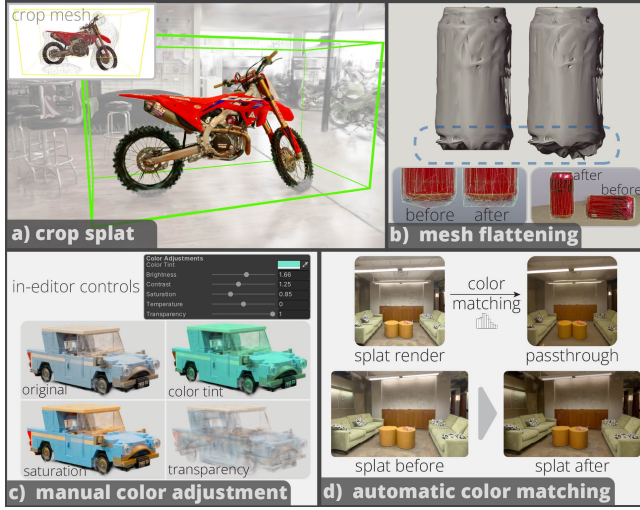


Figure 3: Adjustments: a) Cropping of splats and their aligned meshes; b) Planar flattening to fix reconstruction artifacts on flat surfaces; c) Manual control for visual attributes (tint, saturation, and transparency); d) Auto color adjustment to the passthrough environment via histogram matching.

in-editor previews of the splat and makes the splat feel like other GameObjects in Unity, allowing for streamlined development.

Implementation. We implemented the renderer as a shader program in Unity’s high-level shader language (HLSL). The shader program is based on the functionality presented in the original splatting paper [34]. At design time, once the developer specifies a splat ply file in the import panel, SPLATXR converts it into a custom CSV format by performing preprocessing steps that include computing the covariance matrix needed by our renderer, and de-normalizing color and transparency information. At run time, our splat rendering script reads the custom splat file, which contains the attributes of each 3D Gaussian (covariance, color, alpha), and then assembles an arrangement of triangles. The triangles are passed into the shader memory and sorted from the perspective of a given camera (either in-editor or on-headset) at an adjustable rate (1 Hz by default) in a background thread on the CPU in the rendering script. The shader takes care of rendering each splat-associated triangle and alpha-blending it for the rendering result.

Level-of-Detail Representation. High-resolution splats lead to noticeable frame drops, and very high-resolution splats, e.g., with 1 million Gaussians or more, become infeasible on current mobile GPUs, such as the Qualcomm Snapdragon XR2 Gen 2 found in the Quest 3, especially because the computational budget needs to be shared with many other tasks that are performed on the device. To ensure that end-users can still benefit from high-resolution splats, we propose and implement a level-of-detail (LoD) rendering approach. It displays the high-resolution splat only in close proximity to the user while rendering lower-resolution splats with increasing distance. In our implementation, we use three levels of detail. To make use of the level-of-detail representation, developers merely need to provide the three splats with varying splat

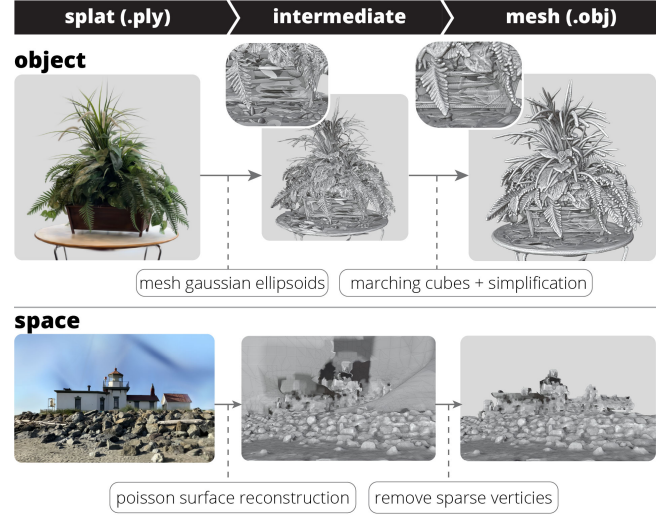


Figure 4: Splat-to-mesh algorithms enable splat physics. *Top: Marching-cubes-based implementation for objects. Bottom: Poisson-reconstruction-based implementation for spaces.*

counts (obtainable by tuning the corresponding parameter in the splat training tool) and select the respective renderer in SPLATXR’s Renderer panel.

Geometric and Photometric Splat Adjustments. The renderer also provides developers with additional options to pre-process the input splats’ properties by adjusting the Gaussians. Developers can interactively or programmatically specify a crop box to remove excess geometry surrounding a splat, as shown in Fig. 3a. Only Gaussians within the crop box are rendered, and the augmenting mesh is jointly cropped.

Overlaying splats on their physical counterpart can lead to a visual mismatch in AR due to the differing lighting conditions of the object at the time of capture and the user’s current environment visible through the passthrough. As a remedy, SPLATXR enables developers to interactively or programmatically adjust a splat’s photometric properties in different ways. First, they can manually tune color tint, brightness, saturation, contrast, temperature, and transparency via in-editor sliders and color pickers, as shown in Fig. 3c. Second, they can provide SPLATXR with a passthrough image from the target view and a screen capture of the virtual view. Using ASC-CDL color grading and histogram matching, our scripts align splat colors for seamless blending, as shown in Fig. 3d.

4.2.2 B2: Splat Physics. While splats offer astonishing visual renderings, their representation lacks surface information. This lack of surface information prohibits the direct use of standard graphics models for physics simulations, e.g., for shadow map computations or collision detection. We therefore augment the splat with an underlying surface mesh representation that can be accessed by the 3D engine for physics simulations, while we still rely solely on the splat to write to the visual framebuffer, thus obtaining the best of both worlds. In our multi-layered reality framework, this allows

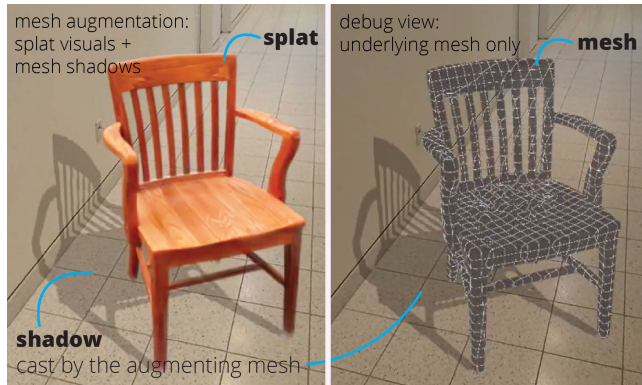


Figure 5: Shadows as part of the *Splat Physics* building block. Shadows can be cast by splat objects onto the physical environment, enabled through mesh augmentation. Collisions, also enabled through mesh augmentation, are supported.

us to create a phantom structure from the splat. We subsume this functionality in the *Splat Physics* block. Fig. 4 provides an overview.

Based on the selected algorithm and parameters, SPLATXR creates a mesh in the OBJ format from the splat, automatically co-aligns it with the input splat, parents it to the splat game object, and assigns mesh colliders as needed. This ensures only the splat is visible, while physics and interactivity take advantage of the surface mesh. SPLATXR implements two mesh generation algorithms. First, we implemented a custom marching-cubes-based algorithm that produces fine detail but hinges on the assumption of convexity of the splat to be reconstructed. While this assumption is sufficiently satisfied for objects, it does not hold for spaces, and therefore, we also implemented a Poisson-based reconstruction algorithm for spaces.

Developer Workflow. To enable physics simulations for an imported splat object, developers can check the *generate mesh* checkbox in SPLATXR’s import panel, select an appropriate mesh generation algorithm, and override required parameters as needed. To maintain simplicity in the developer interface, we allow developers to choose between an *object-optimized* reconstruction based on our marching cubes implementation (ideal for close interaction, physics simulations, casting shadows, and manipulation) and a *space-optimized* approach for indoor or outdoor spaces based on the Poisson-based implementation (apt for providing coarser collision meshes for larger environmental scenes).

Implementation: Object-Optimized Phantom Mesh Generation. To generate a mesh from an object splat, we first create a mesh for each Gaussian ellipsoid from the input PLY file, arranged in a mesh grid. The intermediate representation is a mesh reconstruction of each Gaussian, which together represent the entire object. Next, a splat grid and an isosurface are constructed based on the densities from the Gaussian ellipsoids. We then perform marching cubes over this result to create a mesh with a smoothed surface before decimating it for better performance. Object reconstructions can vary in accuracy depending on the structural characteristics of the input splat, and SPLATXR allows developers to tune parameters for the splat scale and meshing grid threshold. Problematically,

uneven surface reconstruction at the base of splat objects leads to undesirable behavior in interactions, e.g., toppling over when the object is put onto a flat surface. Therefore, we also implement a flattening procedure that allows developers to flatten vertices at the base of the mesh along a specified axis. Fig. 3b illustrates a can’s mesh-augmented splat after flattening, enabling upright placement.

Implementation: Space-Optimized Phantom Mesh Generation. For large-scale reconstructions, we employ the Poisson surface reconstruction algorithm, and then crop the results by removing vertex outliers to efficiently produce large-scale meshes. We use a coarser resolution for scenes than for individual objects to manage compute at scale. This approach recovers the largest surfaces, e.g., the floor, walls, and enclosing sides, needed for collisions.

Shadows and Occlusions. Splat objects that have been augmented with the generated and co-aligned mesh are registered with Unity’s native shadow mapping procedure, enabling them to cast shadows onto other objects in the scene. On Quest devices, SPLATXR leverages the Meta Scene API to obtain an environment mesh, automatically configures it as a phantom shadow catcher [9, 31], and thus enables the visual impression of casting splat shadows onto the physical world. Practically, SPLATXR automatically attaches a custom shadow-casting material to the augmenting mesh. The shadow-casting material uses an HLSL program that writes to neither the framebuffer nor the depth buffer—leaving the augmenting mesh invisible and non-interfering with the splat renderer—while still contributing a shadow-casting pass to Unity’s shadow mapping. Fig. 5 shows a shadow of a splat on a physical environment. Occlusions where splats cover parts of an otherwise visible mesh are implemented by culling occluded mesh geometry in the splat shader.

Collisions. SPLATXR further makes use of the augmenting mesh for collisions. To this end, it automatically attaches a mesh collider to the augmenting mesh, thus computing a tightly fitting convex hull over the vertices within that mesh. This in turn allows attaching Unity’s rigid-body behavior to the object for collision simulation. Fig. 2 shows an example of a splat object colliding with other splats (coke cans), other classical mesh objects (green ball), and the physical environment (table). Based on the rigid-body behavior, SPLATXR can automatically enable grab interactions with the splat. Furthermore, when used with Unity’s navigation mesh component on a space splat, this approach can turn the space splat into a walkable surface for end-users or virtual characters.

4.2.3 B3: Splat Effects. The smooth alpha falloff and the compositional characteristics of a splat lead to an “organic emergence” of its geometric structure as part of the optimization process. In contrast, the structure of a mesh is constructed rather mechanistically from the underlying point cloud. In the *Splat Effects* building block, we leverage this property of splats to our advantage for vivid animations, as shown in Fig. 6.

Developer Workflow. To build a custom splat effect, developers can create a new HLSL shader program by instantiating SPLATXR’s *SplatEffectShader* template. This provides the developer with an empty template within which all elementary splat rendering methods, including *ComputeSplatWorldCenters* and *ComputeSplatWorldCorners*, are callable as library functions. Developers can



Figure 6: Object reintegration effect as part of the *Splat Effects* building block.

insert custom shader code or override existing shader functions for maximum flexibility, and can access a global animation time index.

Implementation: Splat Dis- and Reintegration. We demonstrate the effectiveness of this library-templating approach by replicating Andrioli’s splat dis- and re-integration effect [3] shown in Fig. 6 in 12 lines of code (LoC), the same 8 LoC as the original implementation, plus 4 additional lines to improve XR integration by fading individual Gaussians into passthrough with increasing distance from their true pose. We created a collection of four additional splat animation effects: radial reveal, pulse, digital rain, and warp.

4.2.4 B4: Splat Blending. A recurring challenge in XR applications is blending a virtual scene with the physical scene, e.g., to provide real-world awareness [24], diminish or animate physical objects from the user’s perceived experience [33], or control the degree of immersion [14, 81]. SPLATXR facilitates building such applications by providing the Unity elements necessary to render the scene onto any desirable geometry.

Developer Workflow. Developers can simply drag the Virtual-EyeMaterial onto any “splat-revealing” mesh geometry in the scene, and tag the splats to be masked accordingly.

Implementation: 3D View-Dependent Portal. We demonstrate the ease of use of this approach by implementing the 3D portal shown in Fig. 1, where the virtual Gaussian space splat is rendered onto the physical wall. The splat is invisible *unless* viewed through the splat-revealing portal. Because the material can be applied to any custom mesh geometry, developers retain full control over where the mesh appears, its size and shape, and how it changes over time (e.g., for animations). SPLATXR provides the virtual eye material as part of its material library. The material is shaded by a VirtualEye-Renderer that samples the stereoscopic render texture, provided by two eye-aligned cameras, in image space. By allowing the developer to customize the rendering layers, SPLATXR can ensure that only the desired virtual splat geometry is rendered.

4.3 Headset-Guided Object Reconstruction

SPLATXR’s building blocks provide a fully self-contained library of reusable components for on-device splatting. To bring the advances of compute-intensive vision-foundation models for on-the-fly splat reconstruction to interactive XR applications on compute-constrained headsets, we also provide an edge-computing-based system for lifting physical objects into splat replicas using image-to-3D foundation models.

To reconstruct an object, users select a single point or multiple points in their physical space through their headset. The system

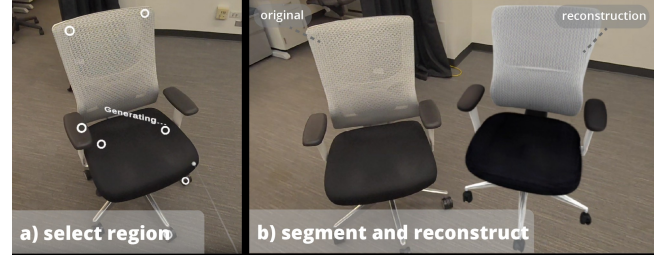


Figure 7: Offloaded object reconstruction via headset capture. Left: 3D points selected on the object of interest. Right: The object is reconstructed and placed next to the original object.

captures the current passthrough view and projects the 3D points into 2D coordinates to serve as prompts for SAM 3 [12]. The resulting image mask and passthrough frame are processed by SAM 3D Objects [77], which generates a 3D splat within minutes. Despite this latency, we expect it to be an acceptable tradeoff for developers as a means of quick 3D reconstruction.

To ensure correspondence with the physical world, the object is automatically scaled to life-size using the Meta Quest Depth API by measuring the real-world distance from the headset to the point of selection and appropriately scaling the generated splat from the estimated world coordinates. The headset sends image and point data wirelessly to a custom remote server and receives a generated splat file. The current implementation runs all generative models locally on an edge computer.

5 Applications

To provide an overview of SPLATXR’s use cases, we developed several applications demonstrating multi-layered reality configurations built from SPLATXR’s building blocks, as shown in Fig. 2. Three capabilities that no prior tool provides together recur across all of them: untethered on-headset splat rendering, which lets them run outside without a rendering server or PC-VR cable; automatic splat-to-mesh proxy generation, which supplies the collision and shadow geometry that splats lack; and AR passthrough compositing with occlusion from environment geometry.

Virtual Try-On. An application showcasing *semantic linkage* where a splat of a watch (*photometric layer*) is dynamically overlaid onto the user’s wrist (*physical layer*) as identified by the *semantic layer*. By exposing the splat as a Unity GameObject, SPLATXR’s *Splat Renderer* allows developers to parent it directly to the Meta SDK’s hand-tracking anchor. This demonstrates how splats can be driven by physical reality and leverage mesh augmentation in *Splat Physics* for occlusion, sparing developers from implementing complex rendering and mesh-occlusion logic.

Outdoor Tour. A full overlay application that places the user entirely within a captured splat environment, replacing the *physical layer* with a comprehensive splat *photometric layer* enabled by the *Splat Renderer*. The user’s physical movement is restricted by a *phantom structure* mapped to the splat using *Splat Physics*.

Its splat-to-mesh proxy generation yields a mesh collider that prevents users from walking or clipping through surfaces, keeping navigation bounded within physical limits.

Library Focus Mode. A *reality blending* application where the user replaces their busy peripheral surroundings with a quiet, static splat of an empty library (*photometric layer*) rendered by the *Splat Renderer*, keeping only their immediate desk visible through passthrough—an instance of diminished and mediated reality that retains grounding in the physical space. SPLATXR’s *Splat Blending* block provides a spatial masking volume around the desk along with shaders that render passthrough inside the volume and the library splat outside it.

Ball Toss Game. An AR game in which a user tosses a virtual ball at an arrangement of splat soda cans resting on a physical table. Mapping each component to its respective layer (Fig. 2c) defines how they interact across different representations of reality: the mixed-reality passthrough forms the underlying base in the *physical layer*, the splat cans are situated in the *photometric layer* using the *Splat Renderer*, and the virtual ball operates within the *geometric layer*, which encapsulates both its explicit geometry and its visual rendering. Cross-layer dynamics, such as the ball bouncing off the table and colliding with the cans, rely on *phantom structures* operationalized by *Splat Physics*. These act as invisible, anchored geometries related to both the physical environment mesh (via the Meta SDK scene mesh) and the splat cans, which underpin the visible elements and allow physics interactions across layers. This decomposition provides a mental model for how distinct visual, structural, and physical elements converge in a unified mixed-reality environment.

Botanical Garden Experience. A volumetric portal, created with additive layering and spatial masking from *Splat Blending*, reveals a botanical garden (*photometric layer* via the *Splat Renderer*). Users can reach through the portal to extract individual plant splats and place them into their physical room (*physical layer*). *Phantom structures* via *Splat Physics*, attached to the garden, plants, and physical environment, allow the splat plants to cast accurate shadows over real-world surfaces and enable collisions across layers.

Situated Interior Design. Users place high-fidelity furniture splats (*photometric layer* via the *Splat Renderer*) into their physical room (*physical layer*). To ensure plausible interaction, *Splat Physics* generates *phantom structures* for both the imported furniture and surrounding surfaces. This enables physical collisions and realistic shadow casting. SPLATXR handles proxy generation, environment occlusion, and passthrough color correction so captured furniture matches the room’s lighting—an AR-specific challenge unaddressed by VR-only systems.

6 Technical Evaluation

We evaluated our renderer and toolkit to investigate different components of the system and their support for a range of splat types for standalone experiences. We tested 27 configurations: 3 objects (plant, car, bonsai), 3 indoor scenes (office, botanical, library), and 3 outdoor scenes (sundial, lighthouse, plaza), each at 3 splat counts

(50k, 100k, and 200k). We include additional details and data in Appendix A.2 and supplementary materials.

6.1 Renderer

Because visible splat density affects performance, indoor and outdoor scenes were measured at both their densest and sparsest viewpoints. Frame rate was capped at 72 fps. CPU frame time was negligible across all conditions ($M=2.3$ ms, $SD=0.3$ ms), making the results GPU-bound. Metrics were averaged over 3 repetitions on a standalone Quest 3, measured with the OVR Metrics Tool, under Unity default quality, on a full battery, from a fixed position.

For objects, the renderer sustained a full 72 fps for both 50k and 100k splats (GPU time 7.6 and 9.7 ms, with negligible frame drops). At 200k, GPU time averaged 15.4 ms and frame rate averaged 59 fps, with frame drops averaging under 6%. For outdoor scenes at the sparsest viewpoint, the renderer held 72 fps through 100k and averaged 64 fps at 200k (GPU time 3–11 ms); at the densest viewpoint it averaged 72 fps at 50k, 53 fps at 100k, and 38 fps at 200k (GPU time 12–26 ms), with frame drops staying low. For indoor scenes at the sparsest viewpoint, the renderer averaged 63 fps at 50k, 61 fps at 100k, and 58 fps at 200k; at the densest viewpoint it averaged 58 fps at 50k, 46 fps at 100k, and 36 fps at 200k with GPU time ranging 13–32 ms, with GPU utilization at 92% and frame drops averaging 2%. On launch, memory was 1.3 GB at the first splat, increasing by 60–140 MB per additional 50k splats. Repeating the full suite reproduced nearly identical results. We observed no performance drop when splat objects were moved or augmented with meshes. Internal tests mixing objects, scenes, and passthrough showed that performance depends on the total number of visible splats. We also tested our system with combined splats at different scales and with XR scene interactions.

The average scene import and CSV conversion times for the 9 splats increase with the number of splats, measuring 0.73 ± 0.14 s for 50k, 1.38 ± 0.12 s for 100k, and 2.5 ± 0.23 s for 200k splats.

6.2 Splat-to-Mesh

We ran splat-to-mesh conversion on the same 27 splat files used in the rendering evaluation. The object algorithm was applied to the 3 objects and the space algorithm to the 6 scenes. Object meshing averaged 25.7 s for 50k splats, 29.9 s for 100k, and 41.4 s for 200k, scaling gradually with splat count. Space meshing was faster, averaging 18.8 s for 50k splats, 22.7 s for 100k, and 31.6 s for 200k.

6.3 Object Reconstruction

This pipeline, which offloads SAM 3D to a PC, reconstructed an object in an average of 2.38 min ($SD = 0.16$ min) across four trials.

6.4 Failure Cases

With our renderer, performance degraded with increased camera proximity and the associated rise in visible splat density. Large high-density splats close to the camera caused stuttering. For our splat-to-mesh operation, the coarse space meshing may struggle with scenes containing many detailed objects or intricate structures, as mesh simplification loses this detail.

7 Preliminary Developer Study

We evaluate SPLATXR following Ledo et al.'s [38] framework, pairing a *demonstration* of applications with a *usage* evaluation in the form of a first-use study structured around Olsen's UI systems criteria [56]: importance and STU context (Situations, Tasks, and Users), expressive leverage and match, power in combination, generality, and scale. We conducted four expert interviews during the development process of the building blocks and a small first-use study ($n=3$) after creating an initial version of the toolkit.

7.1 Formative Expert Interviews

We conducted four semi-structured interviews with experts from the XR domain and the computer vision/graphics domain to understand their potential needs. The first participant (E1) was a filmmaker of Unity-based VR films who uses Nerfstudio, Postshot, Blender, and Houdini to train, edit, and render splats as part of their artistic workflow. The second participant (E2) was a computer vision researcher working on splatting algorithms. The third participant (E3) was an HCI researcher who works with splats and XR. The fourth participant (E4) was a computer vision researcher with experience in neural rendering. We first asked a set of basic questions regarding their use of splatting, and then aimed to elicit thoughts on pain points for the problem of bringing splats into XR.

E1 stressed integration challenges, noting that “the biggest pain point is that splats are just not playing nicely with everything else in the Unity project” and wishing that features spread across many tools could be combined into “one mega tool.” E2 emphasized the need for a real-time preview in any splatting system to “get a sense of the [splat] density.” E2 further highlighted the computational efficiency of splats over neural radiance fields, but felt unsure about the feasibility of implementing splat rendering on a mobile GPU. E1 pointed out that splats really “shine” compared to other methods in terms of photorealism of lighting; however, this advantage would be invalidated if even basic lighting phenomena such as shadows were not properly rendered in XR. E1 and E3 mentioned the issue of blending and compositing multiple splats in a scene with current tools. E4 described the need for a “lightweight” real-time renderer. E3 described a limitation of splats compared to meshes: “[if] I want some physics... just throwing this object... mesh will be better.” E3 found current tools difficult to use and expressed interest in a toolkit that can “accelerate that process or make it simpler for people to use.” E4 mentioned that, despite splats' lack of accurate “3D structure,” meshing would enable “physics interaction... that is probably good enough.”

Using Olsen's strategy [56], these interviews establish the *importance* of the problem and its STU context: the users are XR developers, the task is assembling XR experiences that combine captured splats with virtual meshes, and the situation is on-headset development for standalone XR where compute is constrained and workflows must integrate with existing Unity/Meta SDK pipelines. The interviews also surface a *problem not previously solved*: splats are incompatible with Unity projects and difficult to build XR applications around. From this, we derive two criteria the toolkit must satisfy: a match with existing XR development practices, and splat-mesh augmentation to enable physics and shadows.

7.2 Small-Scale Developer Study

To evaluate the usability and utility of SPLATXR's building blocks, we conducted an IRB-approved remote user study with Unity developers ($n=3$) who had experience developing XR applications with the Meta Quest SDK. After a 30-minute onboarding, each participant built an XR application of their choice over up to a week, followed by a semi-structured interview, which we interpreted through Olsen's heuristics [56].

Generality and Demonstrations. Three developers explored diverse applications (Fig. 8): P1 an audio hide-and-seek game whose splat targets disintegrate and are swapped for an alarm clock; P2 a remote task-guidance system reconstructing objects from passthrough photos and rendering them at runtime with pulsing effects; and P3 a physics sandbox scene combining an imported splat with mesh physics, a looping plant effect, and a blending portal. The applications built by participants, in combination with our sample applications discussed earlier, support a *generality* claim beyond a single application or use case.

Expressive Leverage and Match. All three cleared the toolkit's *threshold* quickly. P2 installed it in less than a minute (“plug and play”), P3 integrated it in under ten minutes, and P1 viewed an in-headset splat render in about fifteen minutes. The SPLATXR building blocks also showed *expressive match*: P3 found them “really easy and intuitive” because they mirror the Meta building blocks they already use, and P1 noted a developer's “intuition... can transfer.” The toolkit expresses splat capabilities in the vocabulary of Meta's SDK, which XR developers already use. Participants could easily incorporate desired splat capabilities such as meshing or effects, reflecting the toolkit's *expressive leverage*.

Power in Combination. Developers valued composing splats with meshes, effects, and passthrough using the blocks: the automatic splat-to-mesh step let P2 add shadows and effects that their prior renderer could not, and let P3 make regular splats interactable. Participants rated the renderer and physics/effects blocks most useful; P2 said the blocks make the toolkit “easier to pick up completely from scratch.”

Problem Not Previously Solved and Importance. P2 contrasted SPLATXR with a prior Unity splat renderer [61] that was “incompatible with a lot of Unity versions,” whereas SPLATXR worked “straight out of the box.” All participants saw value in SPLATXR for their future research: P1 in simulating real-world contexts hard to stage in reality, P2 in photorealistic remote task guidance where “the photorealism of the splat... matters a lot more than just some approximate mesh,” and P3 in interacting with whole environments.

Limitations. Participants noted where the toolkit's functionality fell short: P3's scene “glitch[ed] and freeze[d]” after a portal added a second splat room, and P1 hit unstable physics in large scenes. P2 had to add runtime splat loading, which the package lacked. P1 wanted shader customization and relighting; both P1 and P3 asked for clearer effect configuration and easier base-flattening, which motivates future work on rendering and runtime asset handling.

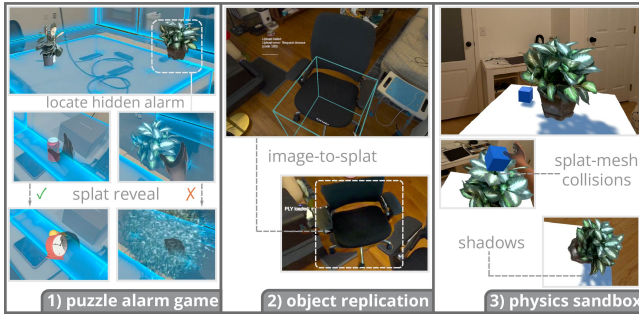


Figure 8: Experiences created by participants using SPLATXR.

8 Limitations and Future Work

While SPLATXR was designed to help developers solve recurring challenges in AR/VR, future research is needed to improve it.

Improving the Renderer. With our LoD representation, we can render scenes across an order of magnitude (e.g., 100K–1M Gaussians). However, to enable splats with even higher counts, research into potential computational improvements of the renderer is needed, e.g., by adapting techniques for foveated rendering. Furthermore, SPLATXR supports static splats, but a sequence of static splats is less expressive and computationally less efficient than a dedicated representation for dynamic splats. Therefore, it would be worth investigating options to bring dynamic splats to the headset, such as 4D video or splat-based avatars. Additionally, while our framework supports shadows, the splats themselves remain unresponsive to environmental lighting. Exploring real-time relighting techniques on splats would enable more realistic compositing.

Analyzing Splats for Interactive Purposes. SPLATXR abstracts away from the content of a splat. It would be interesting to add methods based on either standard geometric processing or machine learning to semantically analyze an imported splat and set it up for common uses in XR. For example, the walkable space of a splat could be derived to navigate a game character on it, to facilitate VR illusions, or to automatically align the splat with the physical walkable area.

Providing Training Capabilities. We implemented SPLATXR to support developers in integrating existing splats in their XR applications. We believe that it could be worth exploring interactions and system components needed to capture a splat from the XR device’s cameras, potentially using generative models.

Outlook. We believe that a fundamental mismatch has emerged: while established XR pipelines are explicitly geometric, emerging scene representations are increasingly implicit and lack structure. Because 3DGS lacks the explicit properties of traditional polygonal meshes, it remains isolated from standard graphics techniques like shadows and collisions. Furthermore, unlike 2D pixel-based generation or generative video, XR is defined by 6DOF movement through 3D physical space. This spatial nature demands explicit grounding; we cannot yet operate solely in pixel space when user perspectives and hand interactions are tied to physical 3D coordinates. While recent action-conditioned generative models allow world exploration via keyboard inputs (WASD) [16], such paradigms are fundamentally incompatible with the fluid spatial input of XR.

We designed our framework to bridge this divide by treating 3DGS strictly as a visual rendering layer, an interface that maps camera poses to pixel outputs. In this capacity, 3DGS serves as a performant proxy for the long-term vision of personalized generative *situated world models*, world models situated in reality which will eventually model immersive 3D scenes and dynamics based on an implicit understanding of the physical environment and the user. By decoupling the visual output from the underlying scene structure, our architecture remains representation-agnostic. Whether the underlying representation is 3DGS today or structured fully implicit world models in the future, our system provides a vision for an architecture to integrate these unstructured, implicit 3D representations with the explicit structure required for physical interaction.

9 Conclusion

We presented SPLATXR, a Unity toolkit that brings Gaussian splatting to standalone XR headsets through modular building blocks: a real-time on-device renderer, splat-to-mesh reconstruction, splat physics, splat blending, and offloaded object reconstruction from headset capture. Together, these components integrate computer vision and graphics techniques into a single developer workflow, making splat-based XR development accessible to the wider XR community. By organizing its components around a multi-layered composition of physical and digital primitives, SPLATXR allows pixels and objects to move across the boundaries between physical and virtual representations, so that virtual splats cast shadows on real surfaces, physical geometry constrains splat physics, and captured objects re-enter the scene as interactive digital twins. Unlike prior splatting work confined to VR, SPLATXR equally supports AR, running wire- and server-free with passthrough compositing, alpha blending, and occlusion from reconstructed geometry. By exposing these capabilities through a familiar Unity interface, SPLATXR provides the technical foundation and mental model to treat the physical and digital as a cohesive, multi-layered reality, where the transition between physicality and virtuality is architecturally supported rather than technically disjointed. We release SPLATXR as open source to support continued community development.

Acknowledgments

We thank Szymon Rusinkiewicz for advice and assistance with the meshing algorithm, Simon Bethke for providing the botanical garden dataset [8], and members of the ψ Lab for their advice. This research was supported by the Princeton NextG Innovation Grant.

References

- [1] Gracia AI. 2025. <https://www.gracia.ai/>.
- [2] Luma AI. 2023. luma.ai.
- [3] Pablo Andrioli. 2025. SparkJS - Splat Shader Effects. <https://github.com/sparkjsdev/spark/pull/141>
- [4] antimatter15. 2023. <https://github.com/antimatter15/splat>.
- [5] Arrival.Space. 2025. <https://arrival.space/>.
- [6] Jonas Auda, Uwe Gruenfeld, Sarah Faltaus, Sven Mayer, and Stefan Schneegass. 2023. A scoping survey on cross-reality systems. *Comput. Surveys* 56, 4 (2023), 1–38.
- [7] João Marcelo Evangelista Belo, Mathias N Lystbæk, Anna Maria Feit, Ken Pfeuffer, Peter Kán, Antti Oulasvirta, and Kaj Grønbaek. 2022. Auit—the adaptive user interfaces toolkit for designing xr applications. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–16.
- [8] Simon Bethke. 2025. Botanical Garden - America. <https://www.kaggle.com/datasets/simonbethke/botanical-garden-america> Dataset licensed under CC

- BY-SA 4.0.
- [9] David E Breen, Ross T Whitaker, Eric Rose, and Mihran Türcüyan. 1996. Interactive occlusion and automatic object placement for augmented reality. In *Computer Graphics Forum*, Vol. 15. Wiley Online Library, 11–22.
 - [10] Arthur Brussee. 2025. <https://github.com/ArthurBrussee/brush/>.
 - [11] Wolfgang Büschel, Anke Lehmann, and Raimund Dachselt. 2021. Miria: A mixed reality toolkit for the in-situ visualization and analysis of spatio-temporal interaction data. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–15.
 - [12] Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, Joseph Greer, Meng Wang, Peize Sun, Roman Rädle, Triantafyllos Afouras, Effrosyni Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, Sagar Vaze, Francois Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, Ho Kei Cheng, Piotr Dollár, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. 2025. SAM 3: Segment Anything with Concepts. [arXiv:2511.16719 \[cs.CV\]](https://arxiv.org/abs/2511.16719) <https://arxiv.org/abs/2511.16719>
 - [13] Hyunsung Cho, Yukang Yan, Kashyap Todi, Mark Parent, Missie Smith, Tanya R Jonker, Hrvoje Benko, and David Lindlbauer. 2024. Minexr: Mining personalized extended reality interfaces. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–17.
 - [14] Robbe Cools, Augusto Esteves, and Adalberto L Simeone. 2022. Blending spaces: Cross-reality interaction techniques for object transitions between distinct virtual and augmented realities. In *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 528–537.
 - [15] Robbe Cools, Jihae Han, Augusto Esteves, and Adalberto L Simeone. 2025. From display to interaction: Design patterns for cross-reality systems. *IEEE Transactions on Visualization and Computer Graphics* (2025).
 - [16] Google Deepmind. 2025. Genie 3: A new frontier for world models. <https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/>
 - [17] Mustafa Doga Dogan, Eric J Gonzalez, Karan Ahuja, Ruofei Du, Andrea Colaço, Johnny Lee, Mar Gonzalez-Franco, and David Kim. 2024. Augmented Object Intelligence with XR-Objects. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) (UIST '24). Association for Computing Machinery, New York, NY, USA, Article 19, 15 pages. <https://doi.org/10.1145/3654777.3676379>
 - [18] Ruofei Du, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso, Jose Pascoal, Josh Gladstone, Nuno Cruces, et al. 2020. DepthLab: Real-time 3D interaction with depth maps for mobile augmented reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 829–843.
 - [19] Kiri Engine. 2025. <https://www.kiriengine.app/>.
 - [20] Faraz Faruqi, Demircan Tas, Arthur Caetano, Niccolò G Meniconi, Oğuz Arslan, Misha Sra, Ruofei Du, Stefanie Mueller, and Mustafa Doga Dogan. 2026. Demonstration of MiXR: Harvesting and Recomposing Geometry from Real-World Objects for In-Situ 3D Design. In *Companion Publication of the 2026 ACM Designing Interactive Systems Conference (DIS '26 Companion)*. Association for Computing Machinery, New York, NY, USA, 613–617. <https://doi.org/10.1145/3802974.3808035>
 - [21] Andreas Rene Fender and Christian Holz. 2022. Causality-preserving asynchronous reality. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–15.
 - [22] Maribeth Gandy and Blair MacIntyre. 2014. Designer's augmented reality toolkit, ten years later: implications for new media authoring tools. In *Proceedings of the 27th annual ACM symposium on user interface software and technology*. 627–636.
 - [23] Eric J Gonzalez, Khushman Patel, Karan Ahuja, and Mar Gonzalez-Franco. 2024. XDTK: A cross-device toolkit for input & interaction in xr. In *2024 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. IEEE, 467–470.
 - [24] Jeremy Hartmann, Christian Holz, Eyal Ofek, and Andrew D Wilson. 2019. Realitycheck: Blending virtual environments with situated physical reality. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.
 - [25] Jaylin Herskovitz, Yi Fei Cheng, Anhong Guo, Alanson P Sample, and Michael Nebeling. 2022. Xspace: An augmented reality toolkit for enabling spatially-aware distributed collaboration. *Proceedings of the ACM on Human-Computer Interaction* 6, ISS (2022), 277–302.
 - [26] Erzhen Hu, Mingyi Li, Xun Qian, Alex Olwal, David Kim, Seongkook Heo, and Ruofei Du. 2024. Experiencing Thing2Reality: Transforming 2D Content into Conditioned Multiviews and 3D Gaussian Objects for XR Communication. In *Adjunct Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) (UIST Adjunct '24). Association for Computing Machinery, New York, NY, USA, Article 23, 3 pages. <https://doi.org/10.1145/3672539.3686740>
 - [27] Xincheng Huang, Dieter Frehlich, Ziyi Xia, Peyman Gholami, and Robert Xiao. 2025. GaussianNexus: Room-Scale Real-Time AR/VR Telepresence with Gaussian Splatting. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology* (UIST '25). Association for Computing Machinery, New York, NY, USA, Article 189, 18 pages. <https://doi.org/10.1145/3746059.3747693>
 - [28] Xincheng Huang, Michael Yin, Ziyi Xia, and Robert Xiao. 2024. Virtualnexus: Enhancing 360-degree video ar/vr collaboration with environment cutouts and virtual replicas. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–12.
 - [29] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. 2011. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) (UIST '11). Association for Computing Machinery, New York, NY, USA, 559–568. <https://doi.org/10.1145/2047196.2047270>
 - [30] Ying Jiang, Chang Yu, Tianyi Xie, Xuan Li, Yutao Feng, Huamin Wang, Minchen Li, Henry Lau, Feng Gao, Yin Yang, et al. 2024. VR-GS: A Physical Dynamics-Aware Interactive Gaussian Splatting System in Virtual Reality. *arXiv preprint arXiv:2401.16663* (2024).
 - [31] Denis Kalkofen, Christian Sandor, Sean White, and Dieter Schmalstieg. 2011. Visualization techniques for augmented reality. In *Handbook of augmented reality*. Springer, 65–98.
 - [32] Mohamed Kari and Parastoo Abtahi. 2025. Reality Promises: Virtual-Physical Decoupling Illusions in Mixed Reality via Invisible Mobile Robots. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology* (UIST '25). Association for Computing Machinery, New York, NY, USA, Article 172, 17 pages. <https://doi.org/10.1145/3746059.3747660>
 - [33] Mohamed Kari, Reinhard Schütte, and Raj Sodhi. 2023. Scene responsiveness for visuotactile illusions in mixed reality. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–15.
 - [34] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). <https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/>
 - [35] Constantin Kleinbeck, Hannah Schieber, Klaus Engel, Ralf Gutjahr, and Daniel Roth. [n.d.]. Multi-Layer Gaussian Splatting for Immersive Anatomy Visualization. ([n. d.]), 1–11. <https://doi.org/10.1109/TVCG.2025.3549882>
 - [36] Amritansh Kwatra, Tobias M Weinberg, Ilan Mandel, Ritik Batra, Peter He, Francois Guimbretiere, and Thijs Roumen. 2025. SplatOverflow: Asynchronous Hardware Troubleshooting. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 450, 16 pages. <https://doi.org/10.1145/3706598.3714129>
 - [37] World Labs. 2024. <https://www.worldlabs.ai/>.
 - [38] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation strategies for HCI toolkit research. In *Proceedings of the 2018 CHI conference on human factors in computing systems*. 1–17.
 - [39] Junseo Lee, Jaisung Kim, Junyong Park, and Jaewoong Sim. 2025. VR-Pipe: Streamlining Hardware Graphics Pipeline for Volume Rendering. *arXiv:2502.17078 [cs.GR]* <https://arxiv.org/abs/2502.17078>
 - [40] Jaewook Lee, Raahul Natarajan, Sebastian S Rodriguez, Payod Panda, and Eyal Ofek. 2022. Remotelab: A vr remote study toolkit. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–9.
 - [41] Ke Li, Mana Masuda, Susanne Schmidt, and Shohei Mori. 2025. Radiance Fields in XR: A Survey on How Radiance Fields are Envisioned and Addressed for XR Research. *arXiv:2508.04326 [cs.GR]* <https://arxiv.org/abs/2508.04326>
 - [42] Ke Li, Tim Rolf, Susanne Schmidt, Reinhard Bacher, Wim Leemans, and Frank Steinicke. 2023. Interacting with neural radiance fields in immersive virtual reality. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–4.
 - [43] Zisu Li, Jiawei Li, Zeyu Xiong, Shumeng Zhang, Faraz Faruqi, Stefanie Mueller, Chen Liang, Xiaojuan Ma, and Mingming Fan. 2025. InteRecon: Towards Reconstructing Interactivity of Personal Memorable Items in Mixed Reality. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–19.
 - [44] Gabriel Lipkowitz. 2024. Palimpsest: a spatial user interface toolkit for cohering tracked physical entities and interactive 3D content. In *Adjunct Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–4.
 - [45] Playcanvas Ltd. 2025. <https://playcanvas.com/>.
 - [46] Zhao Feng Luo, Zitong Cui, Shijian Luo, Mengyu Chu, and Minchen Li. 2025. VR-Doh: Hands-on 3D Modeling in Virtual Reality. *ACM Trans. Graph.* 44, 4 (July 2025), 12 pages. <https://doi.org/10.1145/3731154>
 - [47] Carina Ly, Eleanor Peng, Katie Liu, Anthony Qin, Grace Howe, Alan Y Cheng, and Andrea Quadra. 2025. Museum in the Classroom: Engaging Students with Augmented Reality Museum Artifacts and Generative AI. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–8.
 - [48] Blair MacIntyre, Maribeth Gandy, Steven Dow, and Jay David Bolter. 2004. DART: a toolkit for rapid design exploration of augmented reality experiences. In *Proceedings of the 17th annual ACM symposium on user interface software and technology*.

- 197–206.
- [49] MetalSplat. 2024. <https://github.com/scier/MetalSplat>.
 - [50] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.
 - [51] mkkelllogg. 2023. <https://github.com/mkkelllogg/GaussianSplats3D>.
 - [52] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. <https://doi.org/10.1145/3528223.3530127>
 - [53] Michael Nebeling. 2017. Playing the Tricky Game of Toolkits Research. In *workshop on HCI Tools at CHI*.
 - [54] Nels Numan, Gabriel Brostow, Suhyun Park, Simon Julier, Anthony Steed, and Jessica Van Brummelen. 2025. CoCreatAR: Enhancing Authoring of Outdoor Augmented Reality Experiences Through Asymmetric Collaboration. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. ACM, 1–22. <https://doi.org/10.1145/3706598.3714274>
 - [55] NVIDIA. 2025. <https://www.nvidia.com/en-us/omniverse/>.
 - [56] Dan R Olsen Jr. 2007. Evaluating user interface systems research. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. 251–258.
 - [57] Anastasiya Pechko, Piotr Borycki, Joanna Waczyńska, Daniel Barczyk, Agata Szymańska, Sławomir Tadeja, and Przemysław Spurek. 2025. GS-Verse: Mesh-based Gaussian Splatting for Physics-aware Interaction in Virtual Reality. arXiv:2510.11878 [cs.GR] <https://arxiv.org/abs/2510.11878>
 - [58] Fabian Pointecker, Judith Friedl-Knirsch, Hans-Christian Jetter, and Christoph Anthes. 2024. From real to virtual: Exploring replica-enhanced environment transitions along the reality-virtuality continuum. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–13.
 - [59] PolyCam. 2025. <https://poly.cam/>.
 - [60] Jawset Postshot. 2025. <https://www.jawset.com/>.
 - [61] Aras Pranckevicius. 2023. Gaussian Splatting playground in Unity. <https://github.com/aras-p/UnityGaussianSplatting>
 - [62] Shi Qiu, Binzhu Xie, Qixuan Liu, and Peng-Ann Heng. 2024. Advancing Extended Reality with 3D Gaussian Splatting: Innovations and Prospects. arXiv:2412.06257 [cs.CV] <https://arxiv.org/abs/2412.06257>
 - [63] ReshotAI. 2025. <https://github.com/ReshotAI/gaussian-splatting-blender-addon>.
 - [64] David Rhodes and Ruben Dhz. 2024. Gaussian Splatting Operators for SideFX Houdini.
 - [65] Mose Sakashita, Hyunju Kim, Brandon Woodard, Ruidong Zhang, and François Guimbretière. 2023. VRoxy: Wide-area collaboration from an office using a VR-driven robotic proxy. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–13.
 - [66] Mose Sakashita, Balasaravanan Thoravi Kumaravel, Nicolai Marquardt, and Andrew David Wilson. 2024. SharedNeRF: Leveraging photorealistic and view-dependent rendering for real-time and remote collaboration. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–14.
 - [67] Niantic Scaniverse. 2025. <https://scaniverse.com/>.
 - [68] Markus Schütz¹, Christoph Peters, Florian Hahlbohm, Elmar Eisemann, Marcus Magnor, and Michael Wimmer¹. 2025. Splatshop: Efficiently Editing Large Gaussian Splat Models. In *Computer Graphics Forum*, Vol. 44.
 - [69] HyunA Seo, Juheon Yi, Rajesh Balan, and Youngki Lee. 2024. Gradualreality: Enhancing physical object interaction in virtual reality via interaction state-aware blending. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–14.
 - [70] Yulin Shen, Boyu Li, Jiayang Huang, David Yip, and Zeyu Wang. 2025. GaussianShopVR: Facilitating Immersive 3D Authoring Using Gaussian Splatting in VR. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3746059.3747803>
 - [71] Ronell Sicut, Jiabao Li, Junyoung Choi, Maxime Cordeil, Won-Ki Jeong, Benjamin Bach, and Hanspeter Pfister. 2018. DXR: A toolkit for building immersive data visualizations. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 715–725.
 - [72] Snap. 2025. <https://developers.snap.com/lens-studio/features/graphics/gaussian-splatting>.
 - [73] Spline. 2025. <https://spline.design/>.
 - [74] Supersplat. 2025. <https://supersplat.com/editor/>.
 - [75] Ryo Suzuki, Parastoo Abtahi, Chen Zhu-Tian, Mustafa Doga Dogan, Andrea Colaco, Eric J. Gonzalez, Karan Ahuja, and Mar Gonzalez-Franco. 2025. Programmable reality. *Frontiers in Virtual Reality* Volume 6 - 2025 (2025). <https://doi.org/10.3389/frvr.2025.1649785>
 - [76] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. 2023. Nerfstudio: A Modular Framework for Neural Radiance Field Development. In *ACM SIGGRAPH 2023 Conference Proceedings (SIGGRAPH '23)*.
 - [77] SAM 3D Team, Xingyu Chen, Fu-Jen Chu, Pierre Gleize, Kevin J Liang, Alexander Sax, Hao Tang, Weiyao Wang, Michelle Guo, Thibaut Hardin, Xiang Li, Aohan Lin, Jiawei Liu, Ziqi Ma, Anushka Sagar, Bowen Song, Xiaodong Wang, Jianing Yang, Bowen Zhang, Piotr Dollár, Georgia Gkioxari, Matt Feiszli, and Jitendra Malik. 2025. SAM 3D: 3Dfy Anything in Images. (2025). arXiv:2511.16624 [cs.CV] <https://arxiv.org/abs/2511.16624>
 - [78] ThreeJS. 2025. <https://github.com/mrdoob/three.js/>.
 - [79] Xuechang Tu, Lukas Radl, Michael Steiner, Markus Steinberger, Bernhard Kerbl, and Fernando de la Torre. 2025. VRsplat: Fast and Robust Gaussian Splatting for Virtual Reality. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 8, 1 (2025), 1–22.
 - [80] Cyrus Vachha, Yixiao Kang, Zach Dive, Ashwat Chidambaram, Anik Gupta, Eunice Jun, and Björn Hartmann. 2025. Dreamcrafter: Immersive Editing of 3D Radiance Fields Through Flexible, Generative Inputs and Outputs. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery. <https://doi.org/10.1145/3706598.3714312>
 - [81] Dimitar Valkov and Steffen Flagge. 2017. Smooth immersion: The benefits of making the transition to virtual environments a continuous process. In *Proceedings of the 5th Symposium on Spatial User Interaction*. 12–19.
 - [82] Volinga. 2025. <https://web.volinga.ai/>.
 - [83] Joanna Waczyńska, Piotr Borycki, Sławomir Tadeja, Jacek Tabor, and Przemysław Spurek. 2024. GaMeS: Mesh-Based Adapting and Modification of Gaussian Splatting. arXiv:2402.01459 [cs.CV] <https://arxiv.org/abs/2402.01459>
 - [84] Zeyu Wang, Cuong Nguyen, Paul Asente, and Julie Dorsey. 2023. PointShopAR: Supporting environmental design prototyping using point cloud in augmented reality. In *Proceedings of the 2023 CHI conference on human factors in computing systems*. 1–15.
 - [85] WebGL. 2025. <https://www.khronos.org/webgl/>.
 - [86] WebXR. 2025. <https://immersive-web.github.io/webxr/>.
 - [87] Jonas Wombacher, Zhipeng Li, and Jan Gugenheimer. 2025. SwitchAR: Enabling Perceptual Manipulations in Augmented Reality Leveraging Change Blindness and Inattentional Blindness. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–4.
 - [88] WorldLabs. 2025. <https://sparkjs.dev/>.
 - [89] Benjamin Yang, Shanying Liu, Kirithana Natarajan, and Steven Feiner. 2026. Fast Digitization in XR of Reality-Linked Gaussian-Splatted Proxies. In *2026 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. IEEE.
 - [90] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. 2025. gsplat: An open-source library for Gaussian splatting. *Journal of Machine Learning Research* 26, 34 (2025), 1–17.
 - [91] Brent Yi, Chung Min Kim, Justin Kerr, Gina Wu, Rebecca Feng, Anthony Zhang, Jonas Kulhanek, Hongsuk Choi, Yi Ma, Matthew Tancik, and Angjoo Kanazawa. 2025. Viser: Imperative, Web-based 3D Visualization in Python. arXiv:2507.22885 [cs.CV] <https://arxiv.org/abs/2507.22885>
 - [92] Yuhang Zhao, Edward Cutrell, Christian Holz, Meredith Ringel Morris, Eyal Ofek, and Andrew D Wilson. 2019. SeeingVR: A set of tools to make virtual reality more accessible to people with low vision. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–14.

A Appendix

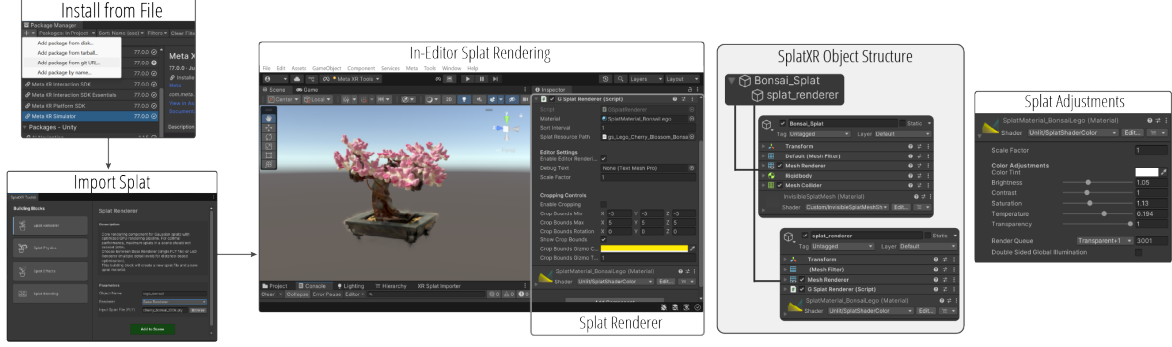


Figure 9: SPLATXR’s developer interface.

A.1 Developer Interface

Fig. 9 shows an overview of the developer workflow in SPLATXR.

Install from File. Developers can import the toolkit package downloaded from our repo into their existing or new Unity projects.

Import Splat. A splat PLY file is imported using the Splat Renderer block from our SplatXR Toolkit editor panel.

In-Editor Splat Rendering. Developers can visualize and interact with the splat as a GameObject in the Unity viewport. Fig. 10 shows a visualization of the LoD rendering in a debug mode where Gaussians are colored based on the LoD they fall in.

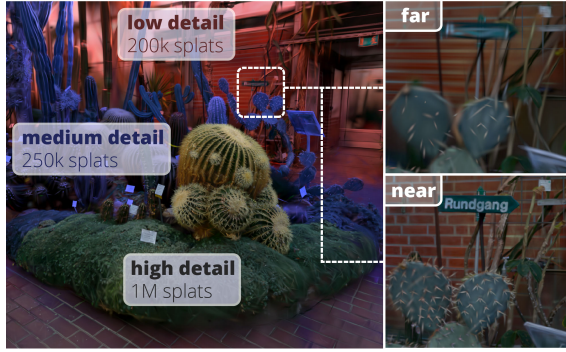


Figure 10: SPLATXR’s level-of-detail rendering. *Left:* Single LoD rendering. *Right:* Comparison of the same volume rendered from far and near.

SplatXR Object Structure. Splats are created as GameObjects with associated scripts, shaders, and components in a hierarchical structure that separates the renderer from the empty parent object.

Splat Adjustments. Splats’ photometric properties can be adjusted from the shader panel.

A.2 Technical Evaluation

To evaluate our renderer, we displayed 9 splat scenes (shown in Fig. 11) in a blank scene on Quest 3. We have open sourced our evaluation setup on our project page. To evaluate other toolkit

functionality (meshing and splat import), we timed the operations in the Unity editor on a laptop PC (4070 GPU, 32 GB RAM).



Figure 11: Splats used for technical evaluations: objects (top), indoor scenes (middle), and outdoor scenes (bottom).

Table 2: Renderer performance on a standalone Quest 3, averaged over three repetitions.

Scene type	Viewpoint	FPS			GPU (ms)
		50k	100k	200k	
Object	—	72	72	59	7.6–15.4
Outdoor	Sparsest	72	72	64	3–11
	Densest	72	53	38	12–26
Indoor	Sparsest	63	61	58	—
	Densest	58	46	36	13–32

Table 3: Splat-to-mesh conversion time in seconds (mean $\pm$ SD) over the 27 splats from the rendering evaluation.

Algorithm	50k	100k	200k
Object (3 objects)	25.7 $\pm$ 3.9	29.9 $\pm$ 5.5	41.4 $\pm$ 4.3
Space (6 scenes)	18.8 $\pm$ 1.5	22.7 $\pm$ 1.1	31.6 $\pm$ 2.7