

Ada: A Distributed, Power-Aware, Real-Time Scene Provider for XR

Yihan Pang , Sushant Kondguli , Shenlong Wang , Sarita Adve 

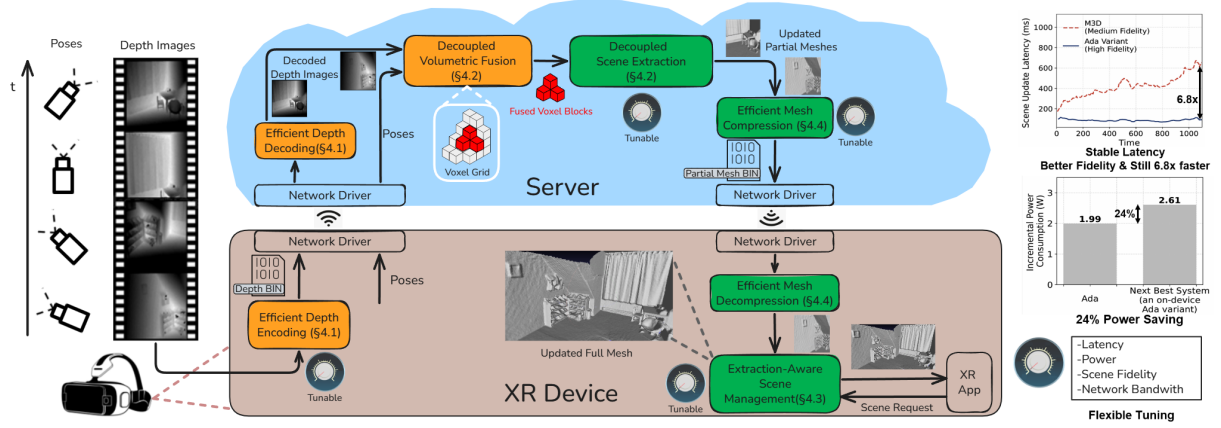


Fig. 1: Ada—a distributed, power-aware, real-time scene provisioning system for XR. Ada delivers higher-fidelity meshes with lower latency compared to Mobile3DRecon (M3D) [82], the closest prior work. For example, an on-device CPU-only variant of Ada is 6.8x faster than M3D (also on-device and CPU-based) while delivering higher mesh fidelity. The final distributed, GPU-accelerated Ada achieves latency comparable to the next-best on-device GPU-accelerated Ada variant while consuming 24% less device-side incremental power. Ada also exposes tuning knobs that let developers trade off latency, power, scene fidelity, and network bandwidth. The orange and green blocks respectively denote scene reconstruction and scene delivery components in the scene provisioning workflow.

Abstract—Real-time scene provisioning—reconstructing and delivering scene data to requesting XR applications during runtime—is central to enabling spatial computing in modern XR systems. However, existing solutions struggle to balance latency, power and scene fidelity under XR device constraints, and often rely on designs that are either closed, application-specific designs, or both. We present Ada, the first open distributed, power-aware, application-agnostic real-time scene provisioning system. Through computation offloading along with algorithmic and system innovations, Ada provides high-fidelity scenes with stable performance across all evaluated scene sizes and with low power consumption. To isolate the benefits of Ada’s algorithmic and design innovations over the closest prior work [82], which is on-device and CPU-based, we configure a comparable on-device, CPU-based variant of Ada (AdaLocal-CPU). We show this variant achieves up to $6.8\times$ lower scene request latency and higher scene fidelity compared to the prior work. Furthermore, Ada’s final distributed GPU-accelerated implementation reduces latency by an additional $2\times$, highlighting the benefits of GPU acceleration and distributed computing. Additionally, Ada also lowers the incremental power cost of scene provisioning by 24% compared to the best on-device variant (AdaLocal-GPU). Finally, Ada flexibly adapts to diverse latency, power, scene fidelity, and network bandwidth requirements.

Index Terms—Extended Reality (XR), Spatial Computing, Real-Time Scene Reconstruction, Computation Offloading, Power-Aware Systems, Distributed Systems, Scene Management

1 INTRODUCTION

XR systems—encompassing virtual, augmented, and mixed reality (AR, VR, and MR)—have the potential to profoundly transform a wide range of domains, including entertainment [24, 67], education [2], healthcare [4], and industrial settings [16, 45]. At the core of many XR applications is spatial computing [9, 19], which enables AR and MR applications to perceive, understand, and interact with real-world environments using real-time scene data.

When an XR application performs a spatial computing task, the task may either request real-time scene data (e.g., mesh) directly or invoke

another task that does so through the XR runtime. To fulfill such requests, the runtime retrieves and provides the latest reconstructed scene data. While this scene request–response process is fundamental to XR systems, prior work lacks a consistent term to describe it. We refer to it as **real-time scene provisioning**: the process of reconstructing and delivering real-time scene data to applications or other XR runtime components during runtime. Figure 2 illustrates how an XR application uses real-time scene provisioning to fulfill a user command. The application first processes the command and requests the latest classified scene from the XR runtime. To fulfill this request, the runtime’s scene understanding component retrieves the latest scene data from real-time scene provisioning, performs object classification, and then returns the scene with classified objects to the application. Finally, the application uses this classified scene to accurately place the virtual teapot onto the recognized table.

Despite its critical role in XR, real-time scene provisioning in resource-constrained XR systems remains challenging. Existing designs prioritize responsiveness over reconstruction fidelity [5] and consume significant power during XR tasks [7], making them unsuitable for high-accuracy tasks and prolonged use. Furthermore, commercial

- Yihan Pang, Shenlong Wang, and Sarita Adve are with University of Illinois Urbana-Champaign. E-mail: yihanp2 | shenlong | sadve@illinois.edu.
- Sushant Kondguli is with Meta Reality Labs Research. E-mail: sushantkondguli@meta.com.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxx/TVCG.201x.xxxxxxx

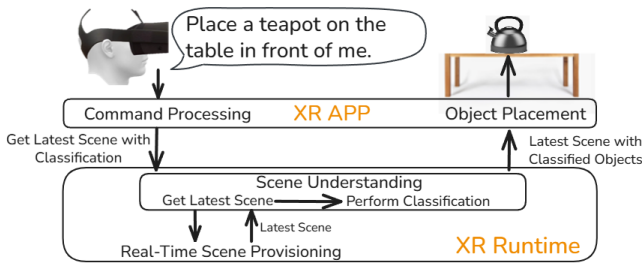


Fig. 2: Example of an XR application utilizing real-time scene provisioning to enable spatial computing.

manufacturers rely on proprietary on-device solutions [6, 44, 49, 54], restricting broader access and innovation.

Recent academic works have explored real-time scene provisioning [21, 36, 40, 64, 82], but face key limitations: they often adopt application-specific strategies, overlook power constraints, and lack open-source implementations. Many target narrow use cases—such as telepresence [21, 36, 40, 64]—making them ill-suited for the growing diversity of XR use cases. Mobile3DRecon [82] stands out as one of the few application-agnostic designs. However, it achieves real-time performance by using low-resolution, low-frequency depth input and coarse reconstruction, limiting its suitability for demanding applications. Power evaluation is also absent across these works, leaving a critical gap for battery-constrained XR devices. Open-source support is also limited—only CHISEL [36] is publicly available—hindering reproducibility and follow-up research. These limitations motivate our design, which supports high-fidelity reconstruction from depth input captured using modern depth sensors, operates under realistic power constraints, promotes reproducibility through open-source support, and is broadly applicable across XR use cases.

Supporting the breadth of XR use cases complicates real-time scene provisioning, requiring careful trade-offs between latency, power, and scene fidelity. For instance, collision detection may tolerate lower-fidelity scenes but requires fast updates, while medical applications demand high-fidelity scenes that are computationally intensive and power-hungry. Varying device capabilities further shape these trade-offs. Unlike application-specific solutions, an application-agnostic, power-aware design should provide flexible controls that enable tuning of scene fidelity, compute usage, and other system parameters according to application needs, hardware constraints, and network conditions.

These challenges highlight the need for a real-time scene provisioning system for XR that is application-agnostic, supports high-fidelity reconstruction using modern sensors, operates under realistic power constraints, and offers tunable controls to adapt to diverse usage scenarios and device capabilities. To support a wide range of XR applications, the system must maintain stable performance across varying scene sizes and content complexity. We propose Ada, a system that achieves these goals through computation offloading, enabled by algorithmic and system-level design innovations (Figure 1).

Computation offloading offers several benefits: (1) it leverages remote resources to enable compute-intensive algorithms that would be infeasible on resource-limited XR headsets; (2) it reduces on-device power consumption and battery drain, even when the algorithm could possibly run locally; (3) unlike custom hardware acceleration, which also lowers power usage, offloading is not constrained by long hardware development cycles or the rapid pace of algorithmic innovation, which often renders custom accelerators obsolete; and (4) it frees device hardware resources, such as the GPU, for other latency-sensitive XR tasks. While offloading has been explored for various real-time XR tasks like head tracking [30], remote rendering [37, 39, 42, 47, 59, 71], SLAM [15], and teleconferencing [21, 40, 64], its potential for application-agnostic real-time scene provisioning remains largely unexplored.

Contributions: Ada makes the following contributions:

(1) *First distributed, power-aware, real-time scene provisioning design for XR.* Ada leverages computation offloading and algorithmic and system design innovations to provide low scene update latency, reduced on-device power consumption, and high-fidelity scenes with

stable performance across scene sizes. Ada introduces several innovations, including a novel depth encoding scheme, decoupled scene reconstruction and extraction, an efficient scene management method, and real-time-optimized mesh compression. These components also expose tunable controls to balance latency, power, scene fidelity and network bandwidth to support diverse XR applications and devices.

(2) *High-performance on-device baseline implementations.* While Ada is designed for distributed execution, its core components generalize to on-device deployments. We demonstrate this through AdaLocal-GPU, a high-performance on-device implementation that achieves similar fidelity to Ada but with higher power consumption. We also develop AdaLocal-CPU, a CPU-only variant designed for a head-to-head comparison with the state-of-the-art (CPU-only, on-device) Mobile3DRecon.

(3) *Open-source release.* To foster future research and ensure reproducibility, our implementation is publicly available as open-source.¹

Key results include:

- We compared Ada with Mobile3DRecon at three scene fidelity levels. Ada maintains low and stable scene update latency at the highest level. In contrast, Mobile3DRecon achieves stable latency only at the lowest level; at medium fidelity, its latency increases substantially over time (as scene size increases) and reaches up to $6.8\times$ that of AdaLocal-CPU at the *highest* level, highlighting its inability to support higher-fidelity reconstruction. AdaLocal-GPU and Ada both reduce latency further by $2\times$ at the highest fidelity level over AdaLocal-CPU (medium fidelity), establishing AdaLocal-GPU as a strong on-device baseline for Ada.
- Ada reduces incremental device-side power consumption by 24% compared to AdaLocal-GPU, while maintaining similar latency and scene fidelity.
- Ada supports flexible system-level tuning to meet diverse application needs and hardware constraints. Depending on the chosen configuration, Ada can achieve targeted goal such as $\sim 20\text{ms}$ average scene update latency, up to 3W device-side power savings, high-fidelity 1cm scene reconstruction, or effective operation under network bandwidth constraints as low as 10 Mbps in both directions.

2 BACKGROUND & MOTIVATION

This section outlines the core components of real-time scene provisioning systems: scene reconstruction and scene delivery. We first motivate our use of voxel-based reconstruction and mesh-based delivery (Sec. 2.1). We then describe these components and their interaction (Sec. 2.2 and Sec. 2.3). Finally, we examine Mobile3DRecon [82], a related system whose limitations motivate our design (Sec. 2.4).

2.1 Key Design Choices in Ada

Real-time scene provisioning involves many design decisions, but two are central: the reconstruction method (classical vs. neural) and the delivery format (mesh vs. point cloud). Ada adopts a *classical* voxel-based reconstruction approach and delivers scenes as *meshes*. These choices align with prior systems like Mobile3DRecon [82] and we further justify them below.

Classical vs. Neural. Ada adopts a classical, voxel-based scene reconstruction approach, which is favored in real-time systems for efficiently reconstructing all visible surfaces into a detailed 3D scene that implicitly encodes surface geometry (e.g., distance to the nearest surface) [86]. Neural methods (e.g., NeRF [55], 3D Gaussian Splatting [34]) also represent scenes implicitly, modeling them as continuous functions learned from data. While neural-based methods offer impressive fidelity, they are currently computationally infeasible for real-time use—even on powerful machines—and largely designed for offline rendering, where scene fidelity is prioritized over speed [34, 55, 83, 84].

In addition to the high reconstruction cost, neural representations are also less compatible with spatial computing tasks. Most such tasks still rely on explicit scene representations—such as meshes and point clouds—which offer immediate access to geometry and are widely

¹<https://github.com/ILLIXR/ILLIXR>

supported. In contrast, spatial computing tasks that operate on implicit scene representations remain limited [10, 11, 29, 31, 80]. While both voxel- and neural-based methods require an additional step to extract explicit scene representations from their implicit formats, voxel-based systems benefit from mature, real-time-capable extraction pipelines [41]. Extracting high-fidelity meshes from neural representations remains computationally expensive and unsuitable for real-time use [22, 83, 84], further limiting their applicability.

Mesher vs. Point Clouds. Among explicit scene representations, Ada adopts a *mesh-based* format for real-time scene delivery, as meshes are the dominant choice in XR platforms [7, 44, 49, 54]. Meshes provide surfaces with defined shape and connectivity, represented by vertices and faces—critical for core XR tasks like collision detection, occlusion handling, and spatial audio. Meshes are also extensively supported by commercial XR frameworks [6], game engines [78, 79], and other domains [8, 20, 85], facilitating seamless integration with XR applications. In contrast, point clouds lack surface connectivity and represent scenes as unstructured sets of points, often requiring surface reconstruction before use. Nevertheless, a mesh-based scene provisioning system such as Ada inherently supports point clouds, since point clouds can naturally be viewed as meshes without face information.

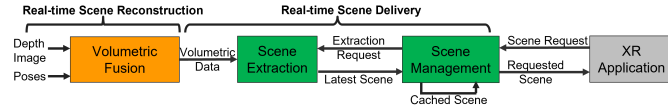


Fig. 3: Voxel-based scene provisioning components.

2.2 Voxel-based Scene Reconstruction

Most voxel-based scene reconstruction algorithms rely on depth and pose (i.e., the camera’s 3D position and orientation) information, either captured directly from sensors or inferred through computational methods such as depth estimation. While Ada is compatible with a range of input sources, in this work we use depth information from depth sensors [53] and poses provided by XR devices [50], both of which are commonly available on XR devices.

The left orange portion of Fig. 3 illustrates the real-time scene reconstruction workflow. Voxel-based scene reconstruction uses pose information to integrate depth measurements into a consistent 3D space composed of voxels—small units that represent volumes in the 3D world. This integration, known as *volumetric fusion*, updates voxels based on incoming depth measurements to represent the latest observed geometry; for simplicity, throughout this paper, we use volumetric fusion to refer to the overall scene reconstruction process. Using smaller voxels (e.g., 2cm) allows the system to capture finer scene details, resulting in higher-fidelity reconstruction. To optimize memory usage and enable GPU parallelism, voxel-based scene reconstruction often uses spatial data structures for efficient storage and retrieval. A common approach is to use voxel blocks (VBs)—small 3D grids that improve memory access by storing neighboring voxels contiguously.

A widely used method for managing voxel blocks is voxel hashing [56], which uses a spatial hash table to efficiently look up and store only the observed parts of the scene, reducing memory usage by skipping empty space. When new depth frames arrive, the system queries the hash table to determine if a voxel block needs to be allocated or updated based on the observed geometry. If a corresponding voxel block exists, its pointer is retrieved, allowing access to the block stored in memory. Individual voxels within the block are then updated accordingly. This structured representation enables fast voxel access while optimizing memory, suitable for resource-constrained XR devices.

2.3 Voxel-based Scene Delivery

Scene delivery refers to the process of providing reconstructed 3D scene data—produced through volumetric fusion—in a format usable by XR applications or other runtime components, such as a mesh. This process typically involves two components: *scene extraction*, which extracts an explicit format (e.g., mesh) from the internal implicit representation, and *scene management*, which handles application requests.

Commercial XR devices implement similar scene management components [51, 77], offering two main retrieval options commonly via API calls: fetching the latest scene or accessing a cached version.

The right green portion of Fig. 3 illustrates the scene delivery workflow in response to XR application requests. For a latest-scene request, the system extracts the most recent data from the internal volumetric representation, which is continuously updated via volumetric fusion. Because this internal format is implicit, scene extraction must convert it into an explicit one before providing it to the application. In contrast, when an application requests a cached version, scene delivery bypasses extraction and directly serves the cached version.

Marching Cubes [41] is a widely used method for extracting meshes from voxel-based representations. It traverses the entire voxel grid, examining each cube formed by eight neighboring voxels to identify isosurface intersections and generate triangles that approximate the surface. This full-grid traversal ensures complete mesh coverage, but is computationally expensive—especially in real-time scenarios where only parts of the scene change. To improve efficiency, more recent methods such as Mobile3DRecon [82] and CHISEL [36] adopt incremental scene extraction: they process only regions affected by newly integrated depth data, avoiding recomputation of unchanged areas. However, this introduces new challenges: (1) scene management must merge the new geometry with previously unmodified regions to maintain a consistent scene, and (2) depending on the system structure, additional conversion may be required before providing the scene to the application [82].

2.4 Scene Provisioning in Mobile3DRecon

Mobile3DRecon [82] (hereafter referred to as M3D) is the closest published system that implements a real-time scene provisioning pipeline, making it a strong baseline for comparison. However, its code is not publicly available. Like Ada, it adopts a classical voxel-based reconstruction approach and delivers scenes in mesh format. Unlike Ada, which is agnostic to how depth is obtained, M3D was designed for mobile devices without dedicated depth sensors and devotes significant design effort to depth estimation—a component orthogonal to our focus on scene provisioning.

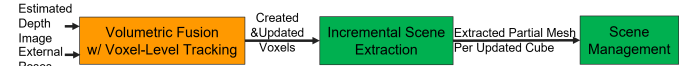


Fig. 4: Scene provisioning in Mobile3DRecon (M3D) [82].

We focus on the scene provisioning design present in M3D, which the authors refer to as incremental mesh generation. Figure 4 illustrates M3D’s scene provisioning design. While the terminology differs, its structure closely aligns with our definition of scene provisioning shown in Fig. 3: it includes volumetric fusion, incremental scene extraction, and a scene management-like component. M3D achieves real-time scene provisioning through three key mechanisms: volumetric fusion with voxel-level status tracking, incremental scene extraction of updated voxels, and a list-based scene management system. These components operate in a tightly coupled manner, executing sequentially at the same frequency—once per input depth frame. M3D extracts and manages the scene at the cube level, where each cube consists of 8 spatially neighboring voxels (a 2x2x2 VB), and each voxel spans a relatively large 6cm volume—trading off scene fidelity for performance.²

During volumetric fusion, M3D tracks voxel updates using status flags, marking voxels as newly created, updated, or unchanged. Incremental scene extraction then applies Marching Cubes selectively, processing only cubes that contain at least one created or updated voxel. The extracted partial mesh from each updated cube is stored in a list-based structure that serves as the scene manager, with each node storing the partial mesh extracted from a single cube. To support the Unity-based applications used in their evaluation, a serialization step converts this internal representation into a standard mesh format.

²The voxel size (6cm) and the serialization step (described later) are not explicitly stated in the published paper but were confirmed through email communication with one of the main authors.

While M3D demonstrates the feasibility of real-time scene provisioning, it faces three key limitations that hinder scalability and responsiveness, especially in emerging XR use cases that increasingly demand high-fidelity, low-latency scene updates. First, M3D relies on low-resolution, low-frequency depth input and low-fidelity reconstruction to maintain real-time performance. This under-utilizes modern depth sensors, which can provide high-resolution, high-frequency data essential for reconstructing high-fidelity scenes. While M3D can technically operate on such inputs, our evaluation shows that doing so results in substantial latency increases that scale with scene size, making it unsuitable for high-fidelity real-time delivery. Second, its tightly coupled pipeline forces reconstruction, extraction, and management to run at the same frequency—even when no request is made. This causes the system to update the scene every frame, wasting computation and degrading performance. Third, M3D uses a list-based scene storage in scene management that requires a costly serialization step to convert its internal format into a standard mesh. This overhead grows with scene size and fidelity, further hurting performance.

These limitations highlight the need for a scalable and efficient scene provisioning system—one that delivers real-time, high-fidelity scenes with stable performance across varying scene sizes. In the next section, we identify key design challenges in achieving these goals, including those that arise in distributed settings.

3 CHALLENGES

3.1 Architectural Constraints of On-Device Processing

Prior application-agnostic designs like M3D [82] perform computations on a single CPU core to minimize resource usage and leave the GPU free for tasks like rendering. While resource-efficient, limiting computation to one CPU core constrains processing throughput, forcing trade-offs in real-time performance and reconstruction fidelity. As a result, these systems only support low-fidelity scene reconstructions, making them unsuitable for modern XR applications that require detailed geometry.

In contrast, several application-specific real-time scene provisioning systems have instead adopted distributed designs [21, 40, 64], offloading heavy computation to a remote server. These approaches highlight the potential of distributed processing to improve performance and scene fidelity. However, they are narrowly tailored to specific use cases and lack the generality needed to support diverse XR applications (Sec. 8).

These limitations motivate a distributed, application-agnostic scene provisioning design that supports high-fidelity reconstruction while maintaining real-time performance with low on-device overhead.

3.2 Hardware-Efficient Precision Depth Encoding

Real-time scene provisioning in XR relies on real-time depth sensing, which captures scene geometry by measuring the distance to surfaces in the environment. These measurements are typically stored per pixel in depth images. Depth sensing can be achieved through various technologies, including structured light and time-of-flight sensors, which emit light and compute depth based on its return. To preserve fine geometric detail, these depth images are typically encoded as 16-bit values [1, 13, 43, 68], allowing each pixel to represent a wide range of distances with high precision—critical for accurate reconstruction. However, transmitting raw 16-bit depth frames requires substantial bandwidth. For example, streaming 640×480 depth at 30 Hz (as in ScanNet [13]) consumes about 147 Mbps.

To reduce bandwidth, depth images are often compressed using image encoding techniques [21, 40, 63, 69]. While some specialized depth encoding methods exist [63, 69], they are not widely supported by standard multimedia frameworks [73, 74] or hardware video encoders [3, 62], which were originally designed for compressing color images. These encoders are attractive for real-time XR systems because they offer efficient, low-latency compression without burdening general-purpose compute resources like the CPU or GPU. Prior approaches [21, 40, 70] have attempted to repurpose these color-focused hardware encoders for depth transmission by converting 16-bit depth images into encoder-supported color-based formats. However, standard formats like RGB and YUV are typically limited to 8-bit precision, and

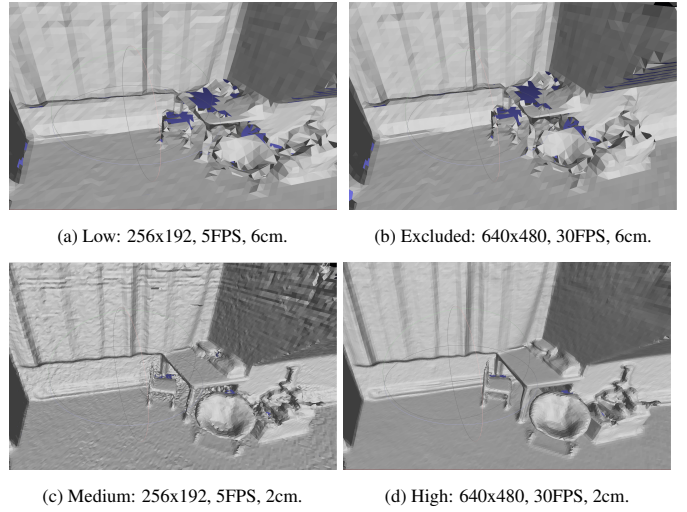


Fig. 5: Scenes generated at different fidelity configurations. The numbers in each part indicate depth image resolution, depth frames per second, and voxel size.

most encoders do not support 16-bit color. As a result, 16-bit depth data must be quantized to 8-bit, introducing precision loss that degrades reconstruction fidelity.

These limitations motivate the need for a real-time encoding approach that preserves depth accuracy, minimizes bandwidth, and runs efficiently on existing hardware encoders.

3.3 Limitations of Coupled Reconstruction & Extraction

Prior works tightly couple volumetric fusion with scene extraction, running both at the same frequency. In rendering-focused applications (e.g., telepresence) [21, 36, 40, 64], this design ensures that scene updates generated from each input are promptly delivered for visualization. However, such tight coupling is excessive for many spatial computing applications, where scene changes are infrequent. Even when changes occur, the application may not request an updated scene; generating one yields no benefit. Moreover, because scene extraction and subsequent scene management are typically slower than reconstruction, coupling these stages forces reconstruction to wait—delaying the fusion of new input and undermining scene timeliness. In such cases, tight coupling leads to unnecessary computation in the scene delivery pipeline.

M3D [82] adopts this tightly coupled design but reduces the processing frequency below the input rate to mitigate delays and lower computational cost. While this gives the system more time to complete reconstruction and scene delivery between inputs, it limits how often new data is fused—reducing opportunities to refine geometry and affecting scene fidelity. This limitation is further compounded by M3D’s use of low-resolution depth input and larger voxels, which reduce processing but also miss finer details available with higher-resolution input and smaller voxels.

Figure 5 shows four configurations that illustrate how input and reconstruction settings affect scene fidelity. (1) Low fidelity (Fig. 5a) uses 256×192 depth image resolution at 5 frames per second (FPS) with 6cm voxels, replicating the original M3D setting. It produces coarse geometry and often misses thin structures, resulting in noticeable artifacts and holes. (2) Medium fidelity (Fig. 5c) uses the same input but with finer 2cm voxels, improving surface detail and capturing small features. (3) High fidelity (Fig. 5d), the configuration used by Ada and all its variants, increases both input resolution and frequency (640×480 @ 30 FPS) while keeping the 2cm voxel size, yielding the most detailed and complete reconstruction. (4) An excluded configuration (Fig. 5b) uses high input resolution but with 6cm voxels. Despite the denser input, there is little visual improvement over the low fidelity configuration due to the coarse voxels.

These findings motivate a decoupled architecture, where volumetric fusion proceeds independently of scene extraction and its downstream

processing. By eliminating the need to wait for extraction and delivery to complete, the system can sustain high-frequency fusion and process high-resolution input—enabling high-fidelity reconstruction while providing more flexible control over when scene updates are handled.

3.4 Efficient Scene Update Management

Scene management is responsible for maintaining a cached local copy of the scene and serving it to applications upon request. When scene updates are delivered as complete meshes—typically generated using full-grid Marching Cubes—the process is straightforward: scene management can serve the updated full mesh directly to the application while replacing the old version in the cache. However, repeatedly extracting and delivering full meshes is inefficient, since typically only a portion of the scene changes between updates. To reduce redundancy and delivery cost, systems like M3D deliver partial, incremental scene updates that include only modified regions. While more efficient, this introduces a key challenge for scene management: integrating partial updates into the cached scene. Scene management must identify and remove outdated regions from the cached scene, then merge in the new geometry without introducing spatial inconsistencies or duplication.

In addition to integrating partial updates, scene management must also deliver the updated scene in a format that matches application expectations. Systems like M3D store the local copy in intermediate formats optimized for internal processing, but these require explicit serialization into the desired mesh format before they can be served to applications—adding latency and computational overhead.

Together, these challenges call for a scene management design that integrates partial updates and serves scenes with low latency.

3.5 Real-Time Mesh Compression for Scene Updates

In distributed setups, the system must transmit mesh updates from the server to the device. These updates can be large—especially in high-fidelity scenes—making raw mesh transmission a significant bottleneck under limited bandwidth. Compression libraries like Google Draco [18] help reduce mesh size, but Draco’s default implementation is designed for offline use and runs single-threaded on the CPU, making it unsuitable for real-time scene updates. In our measurements, compressing partial scene updates with Draco took over 60 ms on average, creating a significant bottleneck for real-time applications and motivating the need for a real-time mesh compression method.

4 ADA

Ada is a distributed, power-aware scene provisioning system that offloads scene reconstruction and extraction to a remote server, supplying real-time, high-fidelity scenes to resource-constrained XR devices. Figure 1 shows an overview of Ada, which introduces four key innovations: efficient depth encoding (Sec. 4.1), decoupled volumetric fusion and scene extraction (Sec. 4.2), extraction-aware scene management (Sec. 4.3), and efficient mesh compression (Sec. 4.4).

4.1 Efficient Depth Encoding

Figure 6 shows our depth encoding method, which avoids quantization loss and enables hardware-accelerated encoding. The design is inspired by bit-plane slicing in image compression [35, 66]. We decompose each 16-bit depth pixel into two 8-bit pixels: the most significant byte (MSB) and the least significant byte (LSB). For the MSB, we use lossless encoding, while for the LSB, we use lossy encoding with a tunable bitrate to reduce network bandwidth. This design is motivated by the observation that lossy encoding of the MSB reduces pixel accuracy, leading to degraded scene fidelity. In contrast, errors in the LSB contribute much less to depth accuracy and are further amortized as more depth observations are fused during reconstruction. The impact of lossy LSB compression can be tuned by adjusting the bitrate to balance accuracy and bandwidth requirements.

In our system, both the MSB and LSB images are first converted without loss [14] to a color-based format (YUV444) commonly supported by hardware encoders. Both images are then encoded using hardware video encoders: the MSB is encoded in a lossless mode to ensure full-precision recovery upon decoding, while the LSB image

is encoded using lossy settings to achieve more aggressive bandwidth reduction at the cost of slight scene fidelity degradation, as studied in Sec. 7. On the server, the encoded MSB and LSB images are decoded using hardware decoders and then combined to reconstruct the original 16-bit depth image. While our experiments specifically use NVIDIA’s NVENC hardware encoders and decoders [62], our encoding strategy—using lossless encoding for the MSB and lossy encoding for the LSB—is broadly applicable to any hardware-accelerated or software-based encoder that supports these encoding modes, with hardware acceleration typically offering reduced latency.

Ada also transmits pose data and optional RGB images. Pose data is lightweight and adds minimal bandwidth overhead, so it is transmitted as-is. RGB images can be encoded using standard hardware video encoders without requiring customization.

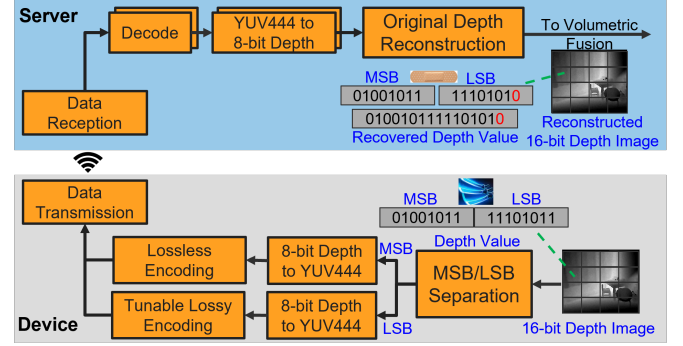


Fig. 6: Efficient depth encoding in Ada. Each 16-bit depth pixel is split into an 8-bit MSB (lossless) and LSB (lossy, tunable). The red bit in the recovered LSB marks a bit flipped by lossy compression.

4.2 Decoupled Volumetric Fusion & Scene Extraction

Ada decouples volumetric fusion from scene extraction, allowing scene reconstruction to run independently and at a higher frequency than extraction. This reduces unnecessary extraction overhead while preserving high-fidelity reconstruction. To keep scene delivery fast and responsive under this design, Ada uses a technique we call *Proactive Scene Extraction*. It uses a lightweight voxel-block tracking method in volumetric fusion to identify updated regions. Together, these components reduce scene request latency by limiting how much data must be processed upon request.

Proactive Scene Extraction enables efficient scene extraction while providing flexibility to balance latency and power consumption. Instead of waiting for an actual request to trigger large, potentially slow extractions, the system incrementally processes updates in the background based on configurable thresholds—reducing the amount of data that must be handled when an actual application request arrive. While the system can operate effectively with default settings, configurable thresholds allow it to adapt runtime behavior to different performance goals. Ada periodically checks whether the extraction threshold has been met in the absence of application requests. When exceeded, the system proactively performs scene extraction on updated regions and resets the tracking state within volumetric fusion to begin tracking new changes. If a request arrives, the system performs an immediate extraction to provide the latest scene, followed by a tracking reset.

Lastly, each extracted face is tagged with its voxel block coordinates, and a compact list of updated voxel block coordinates—called the *Unique Voxel Block List* (UVBL)—is generated during extraction. These coordinates identify spatial regions of the scene, and because voxel blocks are globally aligned, a given coordinate always refers to the same region across updates. They are also naturally available during extraction, as they are needed to locate and fetch the corresponding volumetric data. Ada retains this metadata—normally discarded after extraction—for lightweight region tracking during scene management (Sec. 4.3). As an additional optimization to reduce latency, the partial mesh and UVBL are transmitted independently, allowing scene management to begin processing while the mesh is still in transit.

Volumetric Fusion with Voxel Block-Level Tracking adds a lightweight tracking mechanism that enables Proactive Scene Extraction to process only modified scene regions. During volumetric fusion, tracking is performed by marking the hash table entry for each accessed voxel block. Tracking updates at the voxel-block level provides improved performance by reducing tracking overhead. Although this approach occasionally re-extracts some unchanged voxels, the extra computation is low and does not degrade scene fidelity. This tracking mechanism is broadly applicable: it operates on generic voxel block-based representations and can be integrated with various volumetric fusion pipelines. The voxel block size can also be tuned to balance tracking granularity and performance. Similar update-tracking mechanisms have also been explored in neural scene reconstruction [46, 65], suggesting that both the tracking mechanism and Proactive Scene Extraction could naturally extend to these emerging methods once real-time extraction from neural representations becomes viable.

4.3 Extraction-Aware Scene Management

Proactive Scene Extraction maintains fast and stable performance by proactively extracting updated regions, limiting the work required at each step and at request time. While this design minimizes extraction overhead, it introduces a key challenge: no single extraction contains the complete scene. To deliver an up-to-date, complete scene in mesh format to applications, the system requires a mechanism that incrementally reconstructs the full scene over time. This mechanism must: (1) resolve overlaps by removing outdated regions from the cached scene that have been superseded by new updates, (2) integrate the new partial scene into the cached scene, and (3) efficiently serve the current scene mesh when requested.

To support this functionality, Ada introduces a new device-side scene management design called Extraction-Aware Scene Management, built around three key insights. First, it reuses voxel block coordinates retained during extraction to detect outdated regions and efficiently group updated geometry by spatial region—avoiding the need to infer spatial grouping from mesh geometry [32]. Second, Ada reuses memory from outdated mesh regions to integrate new geometry efficiently. Lastly, it maintains the local scene directly in mesh format, requiring no additional step before serving requests.

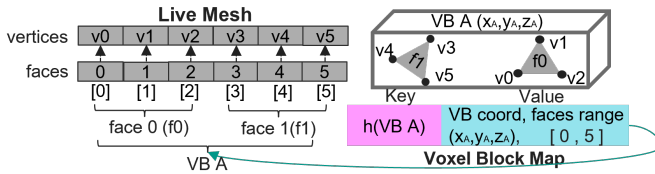


Fig. 7: Data structures for extraction-aware scene management. The *Live Mesh* and *Voxel Block Map* maintain contiguous indices for each voxel block, tracked using hashed 3D coordinates.

Internal Data Structures and Organization Figure 7 illustrates the internal data structures used by Extraction-Aware Scene Management. Ada stores scene data in a Structure of Arrays (SoA) layout, where each mesh attribute (e.g., vertices, colors, faces) is stored in its own array. This collection of SoA-formatted arrays is referred to as the *Live Mesh*. Each voxel block is assigned a contiguous index range across all attribute arrays, encapsulating the mesh content extracted from that block. Within each voxel block’s range, per-face attributes are stored consecutively, and Ada maintains *aligned indexing* across attributes: if a face uses vertices 1, 2, and 3, their colors are stored at the same indices in the color array.

To manage this layout, Ada maintains a hash map called the *Voxel Block Map*, which maps the hashed 3D coordinate of each voxel block to its start and end indices in the face array. While this range could technically be tracked using any attribute array, we use the face array because it is universally required by downstream mesh-based applications. Since indexing is aligned across arrays, this range implicitly applies to most attributes.

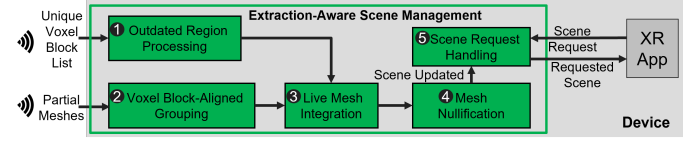


Fig. 8: Extraction-aware scene management workflow. The workflow illustrates how incremental partial mesh updates are integrated into the Live Mesh and how scene management handles application requests.

Scene Management Workflow Figure 8 illustrates the scene management workflow. We next describe each stage in detail.

① **Outdated Region Processing:** Upon receiving the Unique Voxel Block List from the server, scene management compares each block’s coordinate against the Voxel Block Map. A match in the Voxel Block Map indicates that previously extracted scene data from that voxel block now overlaps with newly extracted content and is therefore outdated. The corresponding index range in the face array is recorded in a list for future reuse. After all matching ranges are collected, adjacent ranges are merged to simplify region reuse. A key benefit of transmitting the Unique Voxel Block List alongside the partial mesh is that it enables pipelined computation: while the larger mesh is still in transit, the device can begin processing outdated regions.

② **Voxel Block-Aligned Grouping:** Upon receiving the partial mesh, scene management uses the voxel block coordinates tagged to each face during extraction (Sec. 4.2) to reorganize geometry, aligning it with the layout of the Live Mesh. It creates a temporary structure for each updated voxel block, grouping non-face attributes (e.g., vertices) in the same SoA format as the Live Mesh. After grouping, the system records the vertex array size of each voxel block and sorts the blocks in descending size to improve efficiency during the Live Mesh Integration.

③ **Live Mesh Integration:** After marking outdated index ranges in the Live Mesh and grouping geometry updates by their associated voxel block, scene management integrates each block’s non-face attributes into the Live Mesh. The goal of this stage is to reuse memory from outdated mesh regions by assigning newly grouped geometry to available index ranges. Scene management uses a simple greedy strategy: the voxel block with the largest geometry (i.e., most faces) is assigned to the largest available outdated region. If no region is large enough, the data is appended to the end of the Live Mesh. Once placement is determined, the new data replaces the old content or is appended as needed. Any leftover space is recorded for future reuse. The voxel block’s coordinate and face array range are then updated in the Voxel Block Map.

④ **Mesh Nullification:** After Live Mesh Integration, any outdated ranges that could not be reused must be invalidated to prevent stale geometry from being rendered. Ada applies a lightweight technique called mesh nullification, which leverages the fact that many mesh-consuming applications [12, 85] ignore faces where all vertex indices are identical. For each unused range, scene management overwrites the associated faces by assigning identical vertex indices, collapsing them into a single point. This ensures that no visual artifacts are introduced, even in applications that do not explicitly skip invalid faces. The nullified ranges are retained as outdated regions for future use, enabling efficient memory reuse.

⑤ **Scene Request Handling:** This component monitors incoming application requests and waits for an update signal from Mesh Nullification, indicating that the Live Mesh is ready to be served. Applications interact with Ada through a simple request-response API, issuing scene requests to the scene management module. This design mirrors the scene access patterns of commercial XR runtimes [6, 49], enabling integration with minimal developer effort.

While Extraction-Aware Scene Management is optimized for meshes from voxel-based reconstruction, it can also be applied to any mesh by processing the geometry to assign faces to virtual voxel blocks using a predefined size and layout. However, unlike voxel-based reconstruction, where voxel block coordinates are naturally available during extraction, this approach incurs additional computation to determine the block corresponding to each face.

4.4 Efficient Mesh Compression

Figure 9 illustrates our compression design. We analyzed Draco for mesh compression and identified a key trade-off to improve its suitability for real-time use. Profiling showed that the deduplication stage—used to find duplicate attributes like vertices—is the main bottleneck, accounting for nearly 90% of compression time and worsening with mesh size. While deduplication improves compression ratio by reducing the total amount of data that needs to be encoded, it does not affect mesh correctness. To enable parallel compression and significantly reduce latency, Ada relaxes the deduplication stage by partitioning the updated mesh into chunks and allowing limited attribute duplication across these chunks. Specifically, Ada evenly partitions the face and attribute arrays of the extracted mesh, which are stored in Structure of Arrays format, into chunks before compression with Draco. However, partitioning slightly reduces compression efficiency because it breaks global mesh connectivity: traversal-based encoders like Draco rely on long, continuous face adjacency to exploit repeating patterns for compact encoding, and partition boundaries interrupt these connections.

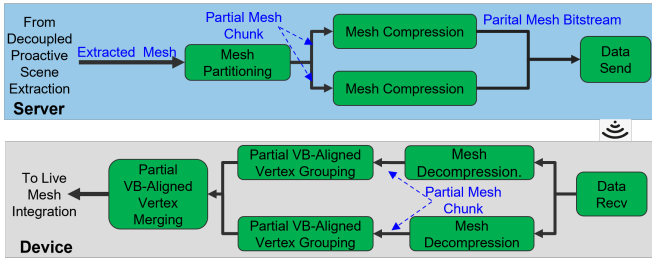


Fig. 9: Efficient mesh compression in Ada. Ada partitions the mesh into large chunks and relaxes deduplication to enable parallel encoding and decoding across CPU cores, significantly reducing latency with minimal impact on compression efficiency.

While our mesh compression builds on Draco, the idea of partitioning the mesh for parallel encoding and decoding can be applied to other compression methods that exploit mesh connectivity.

To support Extraction-Aware Scene Management, Ada extends Draco to encode voxel block coordinates as metadata attached to each mesh face. To further accelerate scene management, Ada leverages parallel mesh decoding to perform Voxel Block-Aligned Vertex Grouping concurrently: each decoder thread performs partial grouping within its chunk, followed by a merge step that consolidates these groupings into a unified structure. Live Mesh Integration proceeds unchanged.

5 EVALUATION METHODOLOGY

Software Overview. Ada is implemented in C++ and built on top of ILLIXR, an open-source full-system XR research testbed [25, 26, 75]. We use InfiniTAM [33], a popular open-source voxel-based reconstruction algorithm that employs voxel hashing for volumetric fusion and Marching Cubes [41] for scene extraction. Ada is not tied to InfiniTAM; it supports alternative fusion and extraction algorithms. We modified InfiniTAM to accept externally provided camera poses, bypassing its built-in pose estimation. This is a common approach in other scene reconstruction research [27, 76].

Evaluated Systems. We evaluated four systems. *M3D* is our initial baseline. As the code is not open-source, we reimplemented its scene provisioning logic and consulted one of the authors to ensure fidelity. Using similar hardware settings, our reimplementation closely matches the reported latency results. *Ada* is our complete distributed real-time scene provisioning design, offloading volumetric fusion and scene extraction to a remote server while the device performs scene management. *AdaLocal-GPU* is an on-device variant of Ada that runs the same volumetric fusion and scene extraction logic locally. *AdaLocal-GPU* omits transmission but shares the GPU with co-running XR tasks (described below). *AdaLocal-CPU* is a CPU-only variant designed to isolate design-level differences from M3D. Like M3D, it runs volumetric fusion and scene extraction on a single CPU thread.

Hardware Settings. We use a Jetson Orin [58] as the device—an embedded platform configurable to emulate commercial XR devices and support detailed power measurements. Ideally, we would use an XR headset; however, most commercial headsets are closed platforms, preventing customized XR runtimes, detailed system-level instrumentation, and the precise power measurements necessary for our evaluation. For the server, we use a system with an Intel i7-12700KF CPU [28] with 64GB RAM and an NVIDIA RTX 3080TI GPU [57]. While not the latest hardware, this setup shows that Ada runs efficiently without cutting-edge components, supporting real-world deployment. The Jetson connects wirelessly (Wi-Fi) to the local network router, while the server connects via Ethernet. For all systems, we kept the Wi-Fi radio on to reflect realistic XR usage as most commercial XR devices require Wi-Fi for core functionalities like cloud services. Device-server communication uses TCP sockets, with messages serialized via Protocol Buffers [17]. Bandwidth is capped at 150 Mbps, following prior real-time telepresence systems [21, 32, 40].

We configure the Orin to emulate the hardware characteristics of commercial XR devices [48, 52], as summarized in Table 1. For *AdaLocal-GPU*, volumetric fusion and scene extraction share the Orin GPU with the local rendering task. We adjust the Orin’s GPU frequency to the minimum level required to sustain a 90 FPS target.

Max GPU Freq (MHz)	714 (Ada, AdaLocal-CPU, M3D)	Max CPU Freq (MHz)	1728
	1020 (AdaLocal-GPU)	CPU Core Count	8
NVENC Freq (MHz) [62]	110 (Ada)	CUDA Core Count	256
	Off (AdaLocal-CPU, AdaLocal-GPU, M3D)	Mem Controller Freq (MHz)	2133

Table 1: Orin’s system parameters for evaluation.

Ada-specific settings. We implemented our depth encoding using GStreamer [74], utilizing NVIDIA’s NVENC hardware encoder [62]. The bitrate for the lossy Least Significant Byte Depth Image was set to 0.5 Mbps. We empirically found that setting a lower bitrate than 0.5 Mbps did not further reduce the effective bandwidth usage, making 0.5 Mbps the optimal choice for maximum bandwidth savings. For efficient mesh compression, the extracted partial mesh was divided into 8 chunks to match the number of performance CPU cores. Proactive scene extractions were triggered every 15 input frames. All systems use 8×8×8 voxel blocks.

Evaluation Scenarios. We evaluate two scenarios. The first compares M3D and AdaLocal-CPU under different fidelity configurations to establish a baseline for evaluating Ada. As introduced in Section 2, we use three scene fidelity configurations—low, medium, and high (Fig. 5a, Fig. 5c, and Fig. 5d respectively)—which vary in input resolution, frame rate, and voxel size. The three configurations allow us to evaluate the latency–scene fidelity trade-offs across different fidelity settings and analyze latency and power consumption within each configuration. For a fair comparison, M3D and AdaLocal-CPU are evaluated without background rendering, so that neither uses the GPU.

The second scenario evaluates Ada, AdaLocal-GPU, and AdaLocal-CPU under a more realistic workload, where scene provisioning co-runs with a GPU-intensive local task. Ada and AdaLocal-GPU are optimized for low latency, while AdaLocal-CPU serves as a single-threaded CPU baseline. All three systems are evaluated using the high fidelity configuration.

We selected a local rendering task as the co-running XR workload, inspired by recent work on offloading rendering [37], for two reasons. First, rendering is a common, GPU-intensive task in XR applications, making it a representative stressor to evaluate Ada’s distributed design, which offloads volumetric fusion and scene extraction to the server, freeing on-device GPU resources. Second, both academic and industry trends are increasingly adopting offloading-based solutions [37, 39, 59]. Evaluating Ada in this context demonstrates its practical relevance. Our rendering task produces a 2K image with 10 dynamic objects—matching the maximum object count in [37], and targets a 90 FPS frame rate, consistent with the refresh rate of commercial XR devices [48].

Datasets. We use sequences from the ScanNet dataset [13], which contains real-world indoor scenes. For the high fidelity configuration, we use the original 640×480 resolution at 30 FPS. For the low fidelity

configuration, we downscale depth to 256×192 and subsample frames to 5 FPS. The medium fidelity configuration uses the same resolution and frame rate as the low fidelity setup but sets the voxel size used in volumetric fusion to 2cm instead of 6cm.

Evaluation Metrics. We use *scene update latency* as our latency metric, defined as the time required to perform volumetric fusion, scene extraction, and scene management. This conservative metric estimates the longest possible scene request latency—assuming a request arrives just as volumetric fusion begins and must wait for the entire pipeline to complete. We report scene update latency only at input frames where the full pipeline runs: every frame for M3D and AdaLocal-CPU, and at proactive extraction intervals for Ada, AdaLocal-GPU, and AdaLocal-CPU.

For power measurements, we use Jetson’s *tegrastats* utility [61], sampling at 10ms intervals. *Tegrastats* monitors power consumption across various subsystems, including the CPU+CV, GPU+SoC (including hardware codecs), DDR, and SYS (display ports, storage, and I/O) [26, 60]. We compute total power by summing these components. Due to high idle power on the Jetson (especially GPU+SoC & SYS), even substantial execution-related differences between systems appear small when expressed as a fraction of total power. To isolate the effect of scene provisioning, we report *incremental power*: the additional power consumed by scene provisioning compared to the minimum power required to sustain the local rendering workload at 90 FPS when running alone.

We evaluate the fidelity of the served scene using the F-score metric, which measures mesh accuracy and completeness based on specified distance thresholds (2cm and 5cm). These thresholds indicate how closely a reconstructed mesh vertex must match a corresponding vertex in the reference mesh to be considered correct. They differ from voxel sizes (e.g., 2cm or 6cm voxels), which determine reconstruction granularity. Following standard practice [23, 69, 72, 81], we compare Ada and AdaLocal-GPU against full meshes extracted from an unmodified version of InfiniTAM, with full scene extraction triggered to match Ada’s proactive extraction schedule. These full (vs. incremental) meshes serve as an upper bound on scene fidelity under the constraints of the underlying reconstruction algorithm. Fidelity is not evaluated for AdaLocal-CPU and M3D since these systems were not designed to support high-fidelity scene reconstruction.

Each data sequence is evaluated at least three times, and we report the average. Latency is reported in both scenarios.

6 RESULTS

6.1 M3D vs. AdaLocal-CPU

We first compare M3D with AdaLocal-CPU to establish an appropriate baseline for evaluating Ada.

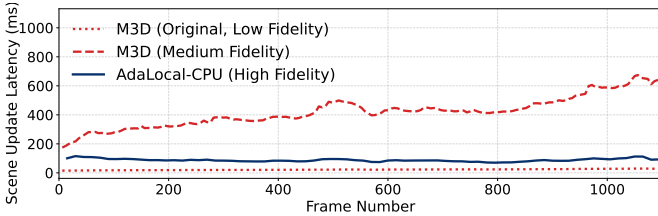


Fig. 10: Scene update latency for M3D and AdaLocal-CPU for different fidelity configurations.

Figure 10 shows the scene update latencies over time incurred by M3D and AdaLocal-CPU at different fidelity configurations. It shows that at low fidelity, M3D incurs an average latency of 22.4ms (consistent with the original paper [82]). However, this low latency comes at the cost of scene fidelity (Fig. 5a). At medium fidelity, however, M3D latency is much higher – it starts at about an order of magnitude higher than the low fidelity case (174.2 ms vs. 15.6 ms) and increases by over 3x across the length of our experiment. We omit high fidelity M3D latency (which is even higher than the medium fidelity) to limit the Y-axis range for greater clarity. In contrast, AdaLocal-CPU at *high*

fidelity maintains a stable average latency of 89.6ms. For clarity, we do not plot AdaLocal-CPU latencies at low or medium fidelity since those are only lower and do not add further insight (for reference, at low fidelity, the average latency is 21.2ms). Compared to M3D at medium fidelity, AdaLocal-CPU at high fidelity sees a factor of 6.8x lower latency by the end of our experiment and 4.8x lower on average. Our detailed experiments (omitted for space) show that all latency related optimizations in AdaLocal-CPU contribute to the reduction of latency relative to M3D.

Takeaway: M3D faces a steep trade off between fidelity and latency—even the medium fidelity configuration sees latencies above 600 ms. In contrast, AdaLocal-CPU delivers higher-fidelity scenes at consistently lower latency. The following sections therefore use AdaLocal variants at high fidelity as our baselines.

6.2 Evaluating Ada

Latency. Based on Sec. 6.1, we use AdaLocal-CPU and AdaLocal-GPU, both running at high fidelity configurations alongside a local rendering task, as our baselines to evaluate Ada.

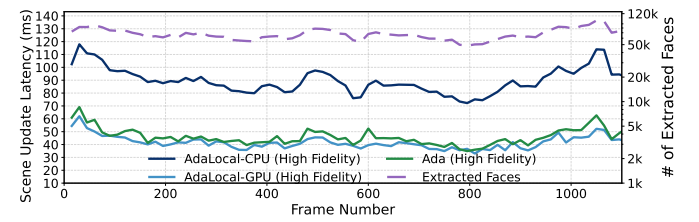


Fig. 11: Scene update latency for Ada, AdaLocal-CPU, and AdaLocal-GPU under high-fidelity configurations.

Figure 11 shows scene update latencies over time for AdaLocal-CPU, AdaLocal-GPU, and Ada. As previously shown in Sec. 6.1, AdaLocal-CPU sees an average latency of 89.6ms. AdaLocal-GPU and Ada both roughly halve this latency, achieving averages of 41.7ms and 45.9ms, respectively, largely due to using a GPU for fusion and extraction. Latency remains stable across frames for all Ada-related systems, indicating they maintain consistent performance even for high fidelity configurations. Figure 11 also plots the extracted mesh size, highlighting its correlation with scene update latency.

Ada achieves latency competitive with AdaLocal-GPU despite off-loading computations to a remote server. The powerful server GPU accelerates scene reconstruction and extraction, effectively compensating for the additional latencies from mesh compression and decompression, and the roughly 4ms network round-trip time (RTT). Such low RTT is common in local networks, which are realistic for currently typical XR deployments. Higher network RTT (e.g., as with wide-area networks) can be offset by upgrading server hardware—for instance, more powerful GPUs can further reduce scene reconstruction and extraction latency, and CPUs with higher core counts or clock frequencies can reduce mesh compression latency—helping maintain comparable overall latency.

Crucially, Ada’s distributed design significantly reduces on-device power consumption (described next) and frees GPU resources on the device, benefiting other latency-sensitive XR tasks.

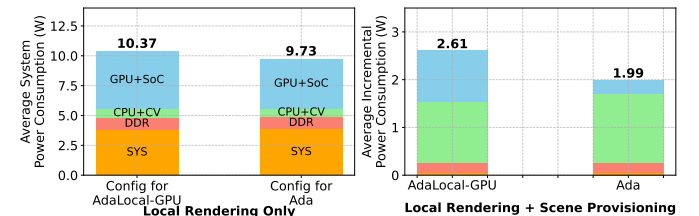


Fig. 12: Device system power (left) and incremental power from scene provisioning (right) for high fidelity configurations.

Power. Figure 12 shows device-side power measurements. The left panel reports power during local rendering alone, while the right panel

shows the incremental power from adding scene provisioning. Compared to AdaLocal-GPU, Ada reduces incremental power by 0.62W (24% lower). This is primarily due to AdaLocal-GPU using the GPU for on-device volumetric fusion and scene extraction, which raises GPU+SoC power by 0.80W. In contrast, Ada offloads these tasks to the server and performs only lightweight operations—such as decompression and scene management—on the device. Ada does incur a small increase (0.19W) in CPU+CV power due to mesh decompression, but overall power is still significantly lower. There is also a slight increase in power due to encoding and transmission of depth images and poses. The encoder power is included in the GPU+SoC rail, slightly reducing the net GPU power savings. Jetson does not provide isolated Wi-Fi power measurements, and official documentation does not clearly specify which rail includes Wi-Fi power consumption. Nevertheless, our reported total device power shows that power savings from computation offloading outweigh the additional costs of encoding and wireless transmission. Furthermore, transmitted depth data and pose data could potentially be reused by other offloaded components, potentially amortizing the transmission and encoding power overhead across XR subsystems.

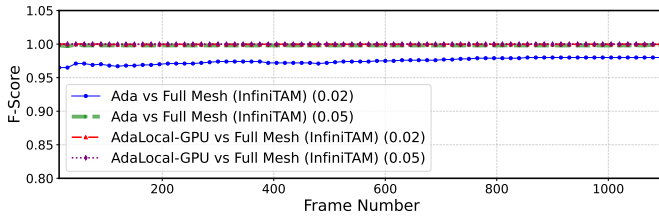


Fig. 13: F-scores (0.02m/0.05m thresholds) for Ada and AdaLocal-GPU against fully (vs. incrementally) extracted InfiniTAM meshes.

Scene Fidelity. Figure 13 compares the scene (mesh) fidelity of Ada and AdaLocal-GPU against fully (vs. incrementally) extracted meshes by InfiniTAM, using F-score at 2cm and 5cm thresholds (Section 5).

AdaLocal-GPU achieves an average F-score of 1.0 under both thresholds, indicating that its incrementally built meshes are identical to the reference, which are extracted in a single pass without incremental updates. This validates the correctness of our scene provisioning design.³ Ada achieves slightly lower but still high average F-scores of 0.971 (2cm) and 0.998 (5cm), despite aggressive compression. This minor F-score drop stems from depth encoding and mesh compression. Our encoding strategy applies lossy compression to the least significant byte of each depth pixel. Still, even at a total upload bandwidth of 2.2 Mbps (1.7 Mbps for lossless MSB and 0.5 Mbps for lossy LSB encoding), Ada maintains high-fidelity reconstructions.

We also compared our depth encoding strategy against alternative depth encoding methods from prior work, as summarized in Tab. 2. These alternative methods typically reduce depth-image precision by converting 16-bit depth images first to an 8-bit format (RGB, YUV, or HSV), followed by standard encoding methods. In contrast, Ada splits depth into lossless and lossy 8-bit segments, significantly reducing image distortion. As shown, Ada achieves the lowest image distortion (highest PSNR) among all evaluated methods, measured directly against reference depth images.

Encoding Path	PSNR (dB)	Encoding Path	PSNR (dB)
16-bit → 8-bit D → 8-bit RGB (e.g., RGB/BGR) [21]	56.3	16-bit → 8-bit D → 8-bit YUV (e.g., NV12/Y444) [40]	59.0
16-bit D → HSV → RGB [70]	61.0	Ada (Efficient Encoding)	70.0

Table 2: PSNR comparison of depth encoding methods. Arrows indicate format conversion paths applied to depth images (D) before encoding. PSNR is computed by comparing decoded depth images directly against reference depth images.

Takeaway. Ada achieves low and consistent latency under high fidelity configurations, outperforming AdaLocal-CPU and matching

³AdaLocal-CPU also achieves perfect F-scores under both thresholds, as it shares the same reconstruction, extraction, and management.

AdaLocal-GPU without using the on-device GPU. By offloading compute-intensive tasks, Ada reduces device power, making it well-suited for energy-constrained XR devices. Further, Ada delivers high-fidelity meshes that closely match the reference.

7 EXPLORING THE ADA CONFIGURATION SPACE

Beyond traditional tuning options such as CPU frequencies and core usage, Ada introduces a unique set of tuning parameters, categorized into two areas: (1) those affecting reconstruction accuracy and (2) those influencing scene update latency and power. These extra controls allow system designers to tune between scene fidelity, latency, and power consumption, enabling Ada to be optimized for varying XR workloads.

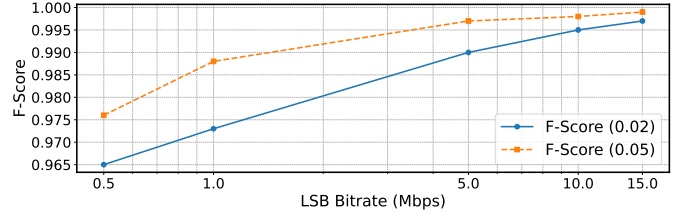


Fig. 14: Impact of LSB bitrate on F-score.

Ada exposes the LSB encoding bitrate as a tuning knob to adjust reconstruction accuracy. Figure 14 shows average F-Score under different LSB bitrates in the high fidelity configuration, compared to full meshes from InfiniTAM. Increasing the LSB bitrate improves F-score, reaching near-identical fidelity (≈0.995) at 10 Mbps. The MSB bandwidth remains stable around 1.7 Mbps.

Ada also provides tuning parameters for optimizing scene update, including the frequency of proactive scene extraction, compression parallelism, scene fidelity, and allocated network bandwidth for scene provisioning. Figure 15 highlights key trade-offs by showing latency and power impacts when varying each parameter individually while keeping all others constant. Specifically, increasing extraction frequency or compression parallelism reduces latency but increases power; using larger voxels reduces both latency and power at the cost of fidelity; and lowering bandwidth caps increases latency, though the power impact remains modest due to efficient encoding.

Figure 15 also includes a standalone "Jointly Tuned 1cm Voxel Configuration" data point, illustrating a configuration that simultaneously tunes multiple parameters—including a smaller voxel size (1cm) and increased compression parallelism (12T/12T)—to reduce latency compared to adjusting voxel size alone, albeit at increased power cost. Broadly, allocating more compute or network capacity reduces latency but raises power, while supplying higher-fidelity scenes tends to increase both. With appropriate tuning, Ada can achieve specific targets such as around 20ms scene update latency, up to 3W device-side power savings (going from Jointly Tuned 1cm to Baseline), acceptable real-time performance under 10 Mbps bandwidth in both directions (below the 250ms threshold recommended for interactive spatial computing applications such as telepresence [38]), or supporting ultra-high-fidelity 1cm meshes, the latter typically feasible only in offline systems [32].

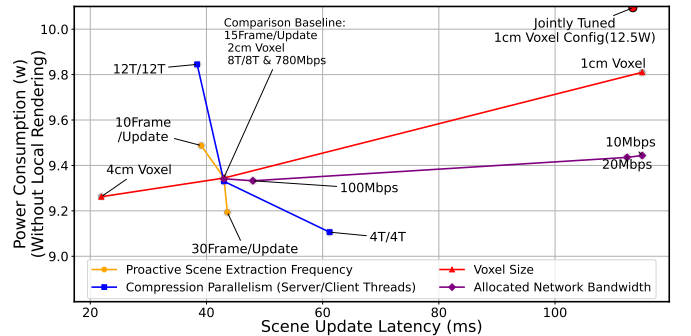


Fig. 15: Effect of tuning Ada's scene-update-related parameters on latency and power consumption.

8 RELATED WORK

Several works [21, 36, 40, 64] optimize real-time scene provisioning for specific applications but rely on task-specific techniques that limit generalizability. Among them, [21, 40, 64] adopt distributed designs: [40] separates static and dynamic facial components for teleconferencing, while [21, 64] use foreground segmentation to isolate regions of interest. Both [21, 40] compress depth by quantizing 16-bit depth to 8-bit color formats—a strategy we show significantly reduce the precision of decoded depth images. In contrast, our method retains substantially higher depth precision. CHISEL [36] performs extraction lazily, extracting geometry only for voxel blocks that are both visible and recently updated. This frustum-driven strategy suits local rendering but yields incomplete coverage and inconsistent, potentially redundant mesh topology—limiting its suitability for applications that need a more complete mesh. Mobile3DRecon (M3D) [82] is the closest prior system, targeting general-purpose scene provisioning on mobile devices. We have implemented it and evaluated it thoroughly, showing that at medium fidelity configuration, M3D has far higher latency than AdaLocal-CPU in high fidelity configuration, establishing a new baseline for efficient on-device scene provisioning. Ada further reduces power without compromising latency or fidelity (the M3D paper did not evaluate power).

9 LESSONS LEARNED & FUTURE DIRECTIONS

Lessons Learned. (1) **End-to-end thinking enables holistic optimization.** Focusing on individual components in a system can overlook broader opportunities. By analyzing how scenes are generated to used—from sensing to application use—we identified many optimization opportunities such as reusable voxel block coordinates to enable efficient scene management. (2) **Cross-layer co-design exposes new efficiencies.** While our depth encoding could be implemented purely in software, doing so would be inefficient and would miss opportunities to leverage hardware capabilities. By co-designing the encoding method with available hardware and aligning with their strengths, we achieve a more efficient solution.

Future Directions. Ada opens several promising avenues for further research. First, one could study how many bits within the MSB require lossless encoding to preserve reconstruction fidelity. Insights from this study could guide the design of dedicated depth-aware hardware encoders, enabling higher compression efficiency. Second, since extracted mesh size directly influences scene update latency, smarter proactive extraction strategies could further reduce latency variations. Third, although Ada considers classical voxel-based reconstruction techniques and meshes, a rich direction for future work is to adapt the techniques in Ada for recent neural methods and alternate scene representations. Finally, our Ada implementation currently assumes a single device per server, simplifying deployment. Scaling the server to support multiple devices requires addressing GPU contention and multi-tenant GPU scheduling. These are currently fertile areas of research and orthogonal to the innovations in Ada.

10 CONCLUSION

We presented Ada, the first distributed, power-aware, real-time scene provisioning system for XR that supports high-fidelity scene. AdaLocal-CPU, our CPU-only on-device variant, achieves up to 6.8× lower scene update latency than the prior state-of-the-art while delivering higher-fidelity scenes. Ada matches the 2× speedup of AdaLocal-GPU over AdaLocal-CPU at 24% lower incremental power and comparable scene fidelity, demonstrating the benefits of offloading. Lastly, Ada exposes tunable parameters to balance latency, power, scene fidelity, and bandwidth, making it suitable for diverse XR workloads.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable feedback and the ILLIXR team for their technical support. This work is supported in part by the National Science Foundation (NSF) under grants 2120464, 2217144, 2331878, 2340254, 2312102, 2414227, and 2404385; the IBM-Illinois Discovery Accelerator Institute (IDAI); and Meta Platforms Technologies, LLC.

REFERENCES

- [1] R. AI. Realsense stereo depth cameras. realsenseai.com, 2025. 4
- [2] A. Alnagrat, R. C. Ismail, S. Z. S. Idrus, and R. M. A. Alfaqi. A review of extended reality (xr) technologies in the future of human education: Current trend and future opportunity. *Journal of Human Centered Technology*, 1(2), 2022. 1
- [3] AMD. Handbrake. AMD Documentation Hub, 2024. 4
- [4] C. Andrews, M. K. Southworth, J. N. Silva, and J. R. Silva. Extended reality in medical practice. *Current treatment options in cardiovascular medicine*, 21, 2019. 1
- [5] Apple. Advanced scene understanding in ar. Apple Tech Talks, 2024. 1
- [6] Apple. Implementing scene understanding and reconstruction in your realitykit app. RealityKit Documentation, 2025. 2, 3, 6
- [7] Apple. Incorporating real-world surroundings in an immersive experience. Apple Developers: VisionOS Documentation, 2025. 1, 3
- [8] Assimp. The open asset importer library. Web page, 2025. 3
- [9] T. Caelli and H. Bunke. Spatial computing: issues in vision, multimedia and visualization technologies. *International journal of pattern recognition and artificial intelligence*, 11, 1997. 1
- [10] T. Chen, P. Culbertson, and M. Schwager. Catnips: Collision avoidance through neural implicit probabilistic scenes. *IEEE Transactions on Robotics*, 2024. 3
- [11] T. Chen, O. Shorinwa, J. Bruno, A. Swann, J. Yu, W. Zeng, K. Nagami, P. Dames, and M. Schwager. Splat-nav: Safe real-time robot navigation in gaussian splatting maps. *IEEE Transactions on Robotics*, 41, 2025. 3
- [12] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*, EG-Italy '08, 2008. 6
- [13] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. CVPR*, CVPR '17, 2017. 4, 7
- [14] DaneLLL. Lossless and near lossless compression with gstreamer. NVIDIA Developer Forums, 2021. 5
- [15] A. Dhakal, X. Ran, Y. Wang, J. Chen, and K. Ramakrishnan. Slamshare: visual simultaneous localization and mapping for real-time multi-user augmented reality. In *Proc. CoNEXT*, CoNEXT, 2022. 2
- [16] P. Fite-Georgel. Is there a reality in industrial augmented reality? In *Proc. VSMAR*, ISMAR '11, 2011. 1
- [17] Google. Protocol buffers. protobuf.dev, 2008. 7
- [18] Google. Introducing draco: compression for 3d graphics. Google Open Source Blog, 2017. 5
- [19] S. Greenwold. Spatial computing. Master's thesis, MIT, USA, 2003. 1
- [20] Q. Group. Qt. Web page, 2024. 3
- [21] Y. Guan, X. Hou, N. Wu, B. Han, and T. Han. Metastream: Live volumetric content capture, creation, delivery, and rendering in real time. In *Proc. MobiCom*, MobiCom '23, 2023. 2, 4, 7, 9, 10
- [22] A. Guédon and V. Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, CVPR '24, 2024. 3
- [23] H. Guo, S. Peng, H. Lin, Q. Wang, G. Zhang, H. Bao, and X. Zhou. Neural 3d scene reconstruction with the manhattan-world assumption. In *Proc. CVPR*, CVPR '22, 2022. 8
- [24] S.-W. Hung, C.-W. Chang, and Y.-C. Ma. A new reality: Exploring continuance intention to use mobile augmented reality for entertainment purposes. *Technology in Society*, 67, 2021. 1
- [25] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, B. Tian, H. Yuan, J. Zhang, and S. V. Adve. Illixr: Enabling end-to-end extended reality research. In *Proc. VISWC*, IISWC '21, 2021. 7
- [26] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, B. Tian, H. Yuan, J. Zhang, and S. V. Adve. Illixr: An open testbed to enable extended reality systems research. *IEEE Micro Special Issue on Top Picks From the 2021 Computer Architecture Conferences*, 42(4), 2022. 7, 8
- [27] M. Huzaifa, B. Tian, Y. Pang, H. Che, S. Wang, and S. Adve. Adaptive-fusion: Low power scene reconstruction. In *Proc. VR*, VRW '23, 2023. 7
- [28] Intel. Intel core i7-12700kf processor. Intel Products, 2021. 7
- [29] H. Jiang, H.-Y. Hsu, K. Zhang, H.-N. Yu, S. Wang, and Y. Li. Phystwin: Physics-informed reconstruction and simulation of deformable objects from videos. In *International Conference on Computer Vision*, ICCV '25,

- [30] Q. Jiang, Y. Pang, W. Sentosa, S. Gao, M. Huzaifa, J. Zhang, J. Perez-Ramirez, D. Das, D. Gonzalez-Aguirre, B. Godfrey, and S. Adve. Remote-vio: Offloading head tracking in an end-to-end xr system. In *Proc.MMSys*, MMSys '25, 2025. 2
- [31] Y. Jiang, C. Yu, T. Xie, X. Li, Y. Feng, H. Wang, M. Li, H. Lau, F. Gao, Y. Yang, and C. Jiang. Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In *ACM SIGGRAPH 2024 Conference Papers*, SIGGRAPH '24, 2024. 3
- [32] T. Jin, M. Dasa, C. Smith, K. Apicharttrisorn, S. Seshan, and A. Rowe. Meshreduce: Scalable and bandwidth efficient 3d scene capture. In *Proc.AVR*, VR '24, 2024. 6, 7, 9
- [33] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. H. S. Torr, and D. W. Murray. Infinitam v3. arXiv, 2018. 7
- [34] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42, 2023. 2
- [35] H. Kikuchi, K. Funahashi, and S. Muramatsu. Simple bit-plane coding for lossless image compression and extended functionalities. In *2009 Picture Coding Symposium*, PCS '09, pp. 1–4, 2009. 5
- [36] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Proc.VRSS*, RSS '15, 2015. 2, 3, 4, 10
- [37] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proc.MobiCom*, MobiCom '19, 2019. 2, 7
- [38] J. Lawrence, D. Goldman, S. Achar, G. M. Blascovich, J. G. Desloge, T. Fortes, E. M. Gomez, S. Häberling, H. Hoppe, A. Huibers, C. Knaus, B. Kuschak, R. Martin-Brualla, H. Nover, A. I. Russell, S. M. Seitz, and K. Tong. Project starline: a high-fidelity telepresence system. *ACM Trans. Graph.*, 40(6), 2021. 9
- [39] M. Leap. Remote rendering overview. Magic Leap Developer Doc, Aug 2024. 2, 7
- [40] K. Lee, J. Yi, and Y. Lee. Farfetchfusion: Towards fully mobile live 3d telepresence platform. In *Proc.MobiCom*, MobiCom '23, 2023. 2, 4, 7, 9, 10
- [41] W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Computer Graphics*, 21(4):163–169, Aug. 1987. doi: 10.1145/37402.37422 3, 7
- [42] E. Lu, S. Bharadwaj, M. Dasari, C. Smith, S. Seshan, and A. Rowe. Renderfusion: Balancing local and remote rendering for interactive 3d scenes. In *Proc.VSMAR*, ISMAR '23, 2023. 2
- [43] MagicLeap. MagicLeap developer docs: Physical occlusion. MagicLeap2 Developer Docs, 2025. 4
- [44] MagicLeap. Meshing subsystem component overview. Magic Leap Developer Docs, 2025. 2, 3
- [45] T. Masood and J. Egger. Augmented reality in support of industry 4.0—implementation challenges and success factors. *Robotics and computer-integrated manufacturing*, 58, 2019. 1
- [46] H. Matsuki, R. Murai, P. H. Kelly, and A. J. Davison. Gaussian splatting slam. In *Proc.CVPR*, CVPR '24, 2024. 6
- [47] J. Meng, S. Paul, and Y. C. Hu. Coterie: Exploiting frame similarity to enable high-quality multiplayer vr on commodity mobile devices. In *Proc.ASPLOS*, ASPLOS '20, 2020. 2
- [48] Meta. Meta Quest Pro. Meta, 2022. 7
- [49] Meta. Access scene data with ovranchor. Meta Horizon Developer Dashboard, 2024. 2, 3, 6
- [50] Meta. Get pose. Meta Horizon OS Developers, March 2025. 3
- [51] Meta. Mixed reality utility kit - working with scene data. Meta Horizon Developer Documentation, 2025. 3
- [52] Microsoft. Microsoft hololens 2. Microsoft Mixed Reality, 2019. 7
- [53] Microsoft. Hololens research mode: Using sensor data in your apps. Microsoft Mixed Reality, 2022. 3
- [54] Microsoft. Spatial mapping. Mixed Reality documentation, 2023. 2, 3
- [55] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1), 2021. 2
- [56] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6), 2013. 3
- [57] NVIDIA. GeForce RTX 3080 Family. NVIDIA Products, 2021. 7
- [58] NVIDIA. NVIDIA AGX Orin. NVIDIA Robotics and Edge AI, 2024. 7
- [59] NVIDIA. Nvidia announces omniverse cloud apis to power wave of industrial digital twin software tools. NVIDIA Press Release, March 2024. 2, 7
- [60] NVIDIA. Software-based power consumption modeling. NVIDIA Jetson Linux Developer Guide, 2024. 8
- [61] NVIDIA. tegrastats utility. NVIDIA Docs, 2024. 8
- [62] NVIDIA. Video Codec SDK. NVIDIA Developer, 2024. 4, 5, 7
- [63] J.-V. J. C. T. on 3D Video Coding Extension Development". 3d high efficiency video coding (3d-hevc). Fraunhofer HHI: Fraunhofer Heinrich-Hertz-Institut, 2015. 4
- [64] S. Orts-Escolano, C. Rhemann, S. Fanello, W. Chang, A. Kowdle, Y. Degtyarev, D. Kim, P. L. Davidson, S. Khamis, M. Dou, V. Tankovich, C. Loop, Q. Cai, P. A. Chou, S. Mennicken, J. Valentin, V. Pradeep, S. Wang, S. B. Kang, P. Kohli, Y. Lutchyn, C. Keskin, and S. Izadi. Holoportation: Virtual 3d teleportation in real-time. In *Proc.UIST*, UIST '16, 2016. 2, 4, 10
- [65] Z. Peng, T. Shao, Y. Liu, J. Zhou, Y. Yang, J. Wang, and K. Zhou. Rtgs-lam: Real-time 3d reconstruction at scale using gaussian splatting. In *Proc.SIGGRAPH*, SIGGRAPH '24, 2024. 6
- [66] A. Podlasov and P. Fränti. Lossless image compression via bit-plane separation and multilayer context tree modeling. *Journal of Electronic Imaging*, 15(4), 2006. 5
- [67] K. Č. Pucihar and P. Coulton. Exploring the evolution of mobile augmented reality for future entertainment systems. *Computers in Entertainment*, 11(2), 2015. 1
- [68] S. Sensor. Structure sensor. structure.io, 2025. 4
- [69] S. Song, Z. Cui, and R. Qin. Vis2mesh: Efficient mesh reconstruction from unstructured point clouds of large scenes with learned virtual view visibility. In *Proc. the IEEE/CVF International Conference on Computer Vision*, CVPR '21, 2021. 4, 8
- [70] T. Sonoda and A. Grunnet-Jepsen. Depth image compression by colorization for intel realsense depth cameras. RealSense Documentation, 2021. 4, 9
- [71] G. Steven, L. Jeffrey, J. Qunjun, S. William, G. Brighten, and A. Sarita. Xrgo: Design and evaluation of rendering offload for low-power extended reality devices. In *Proc.MMSys*, MMSys '25, 2025. 2
- [72] J. Sun, Y. Xie, L. Chen, X. Zhou, and H. Bao. Neuralrecon: Real-time coherent 3d reconstruction from monocular video. In *Proc.CVPR*, CVPR '21, 2021. 8
- [73] F. Team. Ffmpeg: A complete, cross-platform solution to record, convert and stream audio and video. FFmpeg Documentation, 2025. 4
- [74] G. Team. Gstreamer: a flexible, fast and multiplatform multimedia framework. GStreamer API References, 2024. 4, 7
- [75] T. I. Team. ILLIXR Consortium. <https://illixr.org>, 2021. 7
- [76] B. Tian, Y. Pang, M. Huzaifa, S. Wang, and S. Adve. Towards energy-efficiency by navigating the trilemma of energy, latency, and accuracy. In *Proc.VSMAR*, ISMAR '24, 2024. 7
- [77] Unity. Spatial mapping low level api. Unity User Manual, 2019. 3
- [78] Unity. Mesh. Unity Documentation, 2022. 3
- [79] Unreal. Add a collision hull to a static mesh using the auto convex collision tool. Unreal Developer Documentation, 2024. 3
- [80] H. Wang and G. Zhao. Risk-aware safe feedback motion planning in gaussian splatting world. In *2024 IEEE International Conference on Unmanned Systems*, ICUS '24, 2024. 3
- [81] Y. Xin, X. Zuo, D. Lu, and S. Leutenegger. Simplemapping: Real-time visual-inertial dense mapping with deep multi-view stereo. In *Proc.VSMAR*, ISMAR '23, 2023. 8
- [82] X. Yang, L. Zhou, H. Jiang, Z. Tang, Y. Wang, H. Bao, and G. Zhang. Mobile3drecon: Real-time monocular 3d reconstruction on a mobile phone. *IEEE Transactions on Visualization and Computer Graphics*, 26(12), 2020. 1, 2, 3, 4, 8, 10
- [83] L. Yariv, P. Hedman, C. Reiser, D. Verbin, P. P. Srinivasan, R. Szeliski, J. T. Barron, and B. Mildenhall. Bakedsd: Meshing neural sdfs for real-time view synthesis. In *Proc.SIGGRAPH*, SIGGRAPH '23, 2023. 2, 3
- [84] Z. Yu, T. Sattler, and A. Geiger. Gaussian opacity fields: Efficient and compact surface reconstruction in unbounded scenes. *ACM Transactions on Graphics*, 2024. 2, 3
- [85] Q.-Y. Zhou, J. Park, and V. Koltun. Open3d: A modern library for 3d data processing. arXiv, 2018. 3, 6
- [86] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb. State of the art on 3d reconstruction with rgb-d cameras. *Computer graphics forum*, 37(2), 2018. 2