Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

MESH for Incremental Digital-physical system (MESH4ID): A Headset-Centred Virtual Reality Pipeline

Simone Cantarelli
simone.cantarelli3@unibo.it
University of Bologna
Daniela Francia
University of Bologna
Gian Maria Santi
University of Bologna
Alfredo Liverani
University of Bologna

Research Article

Keywords:

Posted Date: July 22nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9280500/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

Additional Declarations: No competing interests reported.

Abstract

This paper presents MESH for Incremental Digital-physical system (MESH4ID), a headset-centred virtual reality pipeline for indoor acquisition, curation, and reconstruction implemented entirely in Unity/OpenXR. The system combines a head-mounted Intel RealSense L515 solid-state LiDAR with a Meta Quest 3 headset to enable live point-cloud preview, persistent snapshot logging, interactive dataset management, and on-demand surface reconstruction within a single immersive environment. Rather than continuously integrating all incoming data into a global model, the proposed workflow is based on user-triggered snapshot acquisition. Each capture is frozen from the live stream, enriched with baked colour information, and stored as a traceable CSV/PLY pair with progressive naming, allowing reproducible and auditable processing. Selected point clouds can then be inspected, merged, and geometrically cleaned in VR before being integrated into a truncated signed distance function (TSDF) volume. An explicit surface mesh is subsequently extracted via Marching Cubes for in-headset inspection. Experimental results show that the workflow is practically feasible in a PC-tethered setting: snapshot persistence and reconstruction remain within a few seconds, while multi-snapshot merging emerges as the principal computational bottleneck. Overall, the system demonstrates the viability of an interaction-driven and traceable XR reconstruction workflow for indoor spatial capture.

1. Introduction

Recent advances in affordable depth sensing have enabled the acquisition of dense three-dimensional data outside specialised laboratories, thereby fostering new applications in robotics, indoor mapping, and extended reality (XR). Concurrently, game engines such as Unity have evolved into extensively adopted platforms for interactive visualisation and rapid prototyping, encompassing immersive inspection through consumer VR headsets. Despite this convergence, end-to-end workflows that combine live depth sensing, persistent logging, user-driven dataset curation, and volumetric surface reconstruction within a single Unity-based XR application remain comparatively underexplored. Existing approaches frequently treat these components in isolation. Robotics-oriented stacks provide mapping and reconstruction with limited immersive interaction, while Unity-based pipelines typically focus on rendering precomputed assets or transient live previews without traceable intermediate artefacts.

A practical limitation of many reconstruction pipelines is that surface generation and data inspection are deferred to a post-processing stage outside the immersive session. This delay prevents rapid iteration between evidence and output: users cannot immediately validate whether a captured dataset is sufficient, nor can they selectively correct inputs before committing to meshing. The workflow proposed here is motivated by the need for fast, on-demand mesh availability in VR and for near-real-time curation of input data (selection, merging, and cleaning) while still on site. By materialising each intermediate state as a persistent artefact (CSV/PLY) and as an interactive entity in the scene, the system supports traceability and auditable reconstruction decisions, enabling the user to reconcile what is reconstructed with what was actually observed.

The present paper sets out a headset-centred Unity/OpenXR system for interactive indoor reconstruction using a head-mounted solid-state LiDAR (Intel RealSense L515) and a Meta Quest 3 VR headset. In opposition to the conventional approach of integrating all incoming frames into a single, constantly updated global model, the proposed workflow places emphasis on deliberate acquisition and traceability. The user is able to capture coloured point-cloud snapshots on demand, with each snapshot being stored as a CSV/PLY pair with systematic naming. The resulting datasets can then be managed directly in VR through selection, merging, and geometric cleaning. When the dataset is deemed adequate, selected clouds are integrated into a Truncated Signed Distance Function (TSDF) volume, and an explicit surface mesh is extracted via Marching Cubes for in-headset inspection.

The present work contributes to the field in two ways. Firstly, it demonstrates a reproducible acquisition-to-reconstruction pipeline embedded entirely in Unity, where every intermediate state is materialised both as an interactive scene entity and as a persistent on-disk artefact, thereby enabling auditability and repeatable experimentation. Secondly, it presents baseline performance figures for the entire pipeline, from the creation and persistence of snapshots to the integration of TSDFs and the extraction of meshes. These figures highlight the practical trade-offs between user-controlled curation, computational cost and reconstruction complexity under a tethered VR setup.

The remainder of the paper is organised as follows. The State of the Art section provides a comprehensive review of the extant literature pertaining to solid-state LiDAR sensing, TSDF-based volumetric fusion, Marching Cubes isosurface extraction, and point-

cloud rendering in XR. The Methodology section details the proposed methodology, including acquisition, snapshot persistence, in-VR curation, and volumetric reconstruction. The System Architecture and Unity Implementation section provides a detailed exposition of the Unity system architecture and implementation choices that ensure coordinate consistency, traceability, and interactive usability. In the Results section, quantitative results for representative workloads are reported, and the observed bottlenecks are discussed. Finally, the Conclusion summarises the main findings, discusses limitations, and outlines future work.

2. State of the Art

2.1. Depth sensing and real-time 3D acquisition

The advent of affordable depth-sensing hardware has rendered the acquisition of dense 3D data accessible beyond the confines of specialised scanning laboratories, thereby opening up novel avenues of application in domains such as robotics, autonomous navigation and extended reality. Recent advancements in the field have seen the emergence of solid-state LiDAR cameras, such as the Intel® RealSense™ LiDAR Camera L515, which offer high-resolution depth measurements (up to 1024 × 768 at 30 fps) over indoor ranges of approximately 0.25–9 m. These cameras boast a compact "hockey-puck" form factor and low power consumption of around 3.5 W [1]. These characteristics render the L515 an attractive choice for mobile and wearable setups where both weight and power budget are constrained, for example when the sensor is rigidly mounted on a VR headset.

In addition to the hardware sensors, a significant amount of research has been dedicated to addressing the fundamental challenge of long-term indoor reconstruction, namely the accumulation of pose drift and the occurrence of tracking failures during extended sessions. It has been demonstrated that dense SLAM systems, such as ElasticFusion, are capable of maintaining globally consistent reconstructions online through continuous frame-to-model tracking and map refinement, without reliance on a classical pose graph. However, this is achieved at the cost of a more complex estimation pipeline [2]. More recent approaches, such as BundleFusion, further target global consistency by optimising camera poses over time and performing on-the-fly surface re-integration. These approaches explicitly address relocalisation events and large drift corrections during live scanning [3]. The present work is informed by the findings of the aforementioned studies, which provide a useful reference point. While the aforementioned studies prioritise continuous automated mapping, the pipeline proposed in this paper instead emphasises deliberate snapshot acquisition and traceable, user-driven dataset curation within VR.

A substantial body of work has been developed on the basis of such sensors, addressing simultaneous localisation and mapping (SLAM) and dense 3D reconstruction. Although many SLAM systems were originally developed for mechanically spinning LiDAR sensors, more recent frameworks explicitly target solid-state LiDAR and demonstrate accurate localisation and high-quality mapping with devices such as the L515, thereby highlighting their suitability for indoor environments and small-scale robots [4]. However, it should be noted that these frameworks are typically embedded in robotics stacks and are not directly integrated into interactive VR visualisation pipelines.

2.2. Volumetric fusion with Truncated Signed Distance Functions (TSDF)

From a reconstruction perspective, volumetric fusion based on signed distance fields has become a foundational technique. Curless and Levoy's seminal work introduced a volumetric method for integrating multiple range images into a Truncated Signed Distance Function (TSDF), which represents surfaces implicitly as the zero level-set of a scalar field defined over a voxel grid [5]. In their formulation, each voxel stores a signed distance to the nearest surface, truncated to a finite band around the surface and accumulated with confidence weights. This approach offers a number of desirable properties, including incremental updating, robustness to sensor noise and outliers, the ability to fill small gaps, and a convenient implicit representation for subsequent mesh extraction.

The advent of real-time depth cameras, such as the Microsoft Kinect, catalysed the development of KinectFusion, which demonstrated the capacity for TSDF fusion and camera tracking to be performed in real time on commodity GPUs [6]. KinectFusion employs a moving depth sensor to construct a dense TSDF volume of an indoor scene, while concurrently estimating the camera pose via dense Iterative Closest Point (ICP) alignment between the live depth frame and a synthetic depth image raycast from the TSDF. This work established TSDF as a practical representation for interactive 3D reconstruction and provided a blueprint for many subsequent systems (*ibid.*).

In the field of robotics, the TSDF has been adopted as a mapping primitive that facilitates both mesh extraction and the derivation of Euclidean Signed Distance Fields (ESDFs). The Voxblox framework proposed by Oleynikova et al. presents an incremental TSDF-ESDF pipeline for micro aerial vehicles, demonstrating that TSDF volumes can be constructed with enhanced efficiency and precision in comparison to traditional occupancy grids. Subsequent conversion of TSDFs into ESDFs enables rapid motion planning [7]. Voxblox validates real-time performance on on-board hardware and emphasises the advantage of TSDF in smoothing sensor noise over multiple observations while still producing high-quality surface meshes for human operators.

Subsequent to these pioneering contributions, TSDF-based methods have been expanded in numerous directions, including the development of adaptive or hierarchical grids, voxel hashing for large-scale environments [8], and the integration of semantic labels or multi-modal data. However, the fundamental concept remains constant: a truncated signed distance volume integrates numerous noisy depth measurements into a coherent implicit surface that can be queried, visualised and differentiated for subsequent tasks. Concurrently, modular volumetric frameworks have been proposed with a view to unifying tracking, integration, and scene representation choices within a single pipeline. InfiniTAM v3 serves as a representative exemplar, offering a flexible TSDF-based architecture and extending conventional KinectFusion-style systems towards larger-scale reconstruction with loop closure via submap-based strategies [9]. This body of research is of particular relevance when positioning dense-grid prototypes (such as the one adopted here for a bounded indoor volume) against methods that explicitly address scalability and long-session consistency.

The architecture outlined in this paper adopts a TSDF-centric perspective on reconstruction, adapting it to a Unity-based virtual reality environment and to point clouds acquired from a head-mounted L515 sensor. In lieu of the construction of a TSDF which is real-time in nature, the system under consideration logs a series of coloured point-cloud snapshots. Thereafter, selected clouds are integrated into a TSDF volume on demand. The design of this system is predicated on the principles of user control, the reproducibility of intermediate point clouds (via CSV/PLY files), and interactive tuning of TSDF parameters such as voxel size, truncation distance and grid placement.

2.3. Isosurface extraction with Marching Cubes

Following the construction of a TSDF volume, a common strategy is the extraction of an explicit polygonal representation of the zero level-set. The classical Marching Cubes algorithm, as introduced by Lorensen and Cline in 1987, is a widely used methodology for isosurface extraction from scalar fields [10]. The Marching Cubes algorithm processes the scalar field cell by cell, treating each voxel cube as a local configuration of eight signed samples. For each configuration, a pre-computed case table specifies how to connect edges to form one or more triangles that approximate the isosurface within that cube. The vertex positions are obtained by linear interpolation along edges where the scalar field changes sign.

The original Marching Cubes case table is characterised by its compactness and efficiency; however, it is also recognised to manifest ambiguous configurations that have the potential to induce cracks or inconsistent topology when implemented in a rudimentary manner. Chernyaev's "Marching Cubes 33" addresses this issue by refining the case taxonomy to ensure that the extracted triangulation matches the topology of the trilinear interpolant within each cell, thereby improving robustness in challenging configurations [11]. Building on this foundation, Lewiner et al. provide an efficient, implementation-oriented treatment of MC33, offering practical rules and tables that make topologically consistent extraction feasible in real-time pipelines, including GPU-friendly settings. These contributions are pertinent to Unity-based TSDF workflows, as they elucidate the methods by which manifoldness can be preserved and the generation of artefacts can be circumvented during the extraction of meshes from discretised signed distance fields [12].

The development of Marching Cubes was originally for the purpose of medical imaging, yet it rapidly became the standard for extracting surfaces from volumetric data in the fields of computer graphics and scientific visualisation. A plethora of extensions have been developed to enhance the topological correctness of the system, reduce ambiguities in the case table, and accelerate execution on GPUs. However, the majority of practical implementations adhere to a similar fundamental pattern, which involves a table-driven traversal of the voxel grid with local interpolation. When combined with TSDF volumes, Marching Cubes is particularly effective: the zero-crossing of the TSDF encodes the reconstructed surface, and the sign of the TSDF values provides a clear inside/outside partition.

Within the scope of this study, Marching Cubes serves as the conduit between the implicit TSDF representation and the explicit meshes that are rendered in Unity and inspected in VR. The system utilises a custom Marching Cubes implementation that traverses the TSDF grid, identifies cubes whose corners straddle the designated iso-value (typically zero), and generates triangle meshes expressed in the reconstruction frame via voxel-to-world mapping (voxel size and grid origin). This choice ensures that the reconstruction pipeline is consistent with established practices in graphics and vision, while still enabling interactive control over grid resolution and bounding volume within the Unity environment.

2.4. Point-cloud visualisation in game engines and XR

Concurrent with these developments in reconstruction, game engines such as Unity and Unreal have become extensively adopted platforms for interactive visualisation, simulation and XR prototyping. For point clouds specifically, Unity's standard mesh pipeline is not optimised for very large numbers of unstructured points. Consequently, a variety of community tools and plugins have emerged to handle point-cloud data more efficiently.

One such tool is Pcx, an open-source point-cloud importer and renderer for Unity that supports binary little-endian PLY files containing position and colour attributes [13]. Pcx employs bespoke vertex formats and shaders to render point clouds as GPU-friendly point primitives, and has been utilised in a range of applications, including cultural heritage visualisation, spatial effects and prototype XR experiences. The Pcx4D extension generalises this approach to time-varying point clouds, thereby further emphasising the flexibility of point-based rendering within the Unity framework.

In a broader context, point-based rendering has a substantial academic literature that anticipates many of the challenges encountered when visualising LiDAR point clouds in XR. In this domain, head motion and close-range inspection can amplify aliasing, density variation, and perceptual clutter. Rusinkiewicz and Levoy's QSplat was a pioneering innovation in the field of point rendering, introducing a multiresolution hierarchy that enabled progressive refinement and interactive display of very large scanned datasets. This was achieved by adapting sampling density to the viewer's distance and screen-space footprint, a technique that has since become a standard in the field [14]. Zwicker et al.'s surface splatting, in turn, formalised high-quality point set rendering through screen-space reconstruction filters (EWA splats). This improvement resulted in enhanced visual continuity and reduced artefacts when point spacing was irregular or when points project to sub-pixel footprints. Collectively, these approaches inform design choices in Unity-based XR pipelines, including level-of-detail management, screen-space point sizing, and filtering strategies. These strategies aim to preserve readability and stability while maintaining rendering costs within acceptable limits [15].

Existing Unity-based workflows, however, typically assume that point clouds are pre-computed offline and imported as static assets. While certain projects integrate Pcx-like renderers with real-time depth sensors, these projects frequently prioritise visualisation over the comprehensive reconstruction pipeline, which encompasses logging, selection, merging, cleaning, and TSDF-based meshing within a unified environment. Furthermore, a significant proportion of research prototypes in the field of robotics depend on custom visualisation tools or ROS-based front-ends as opposed to VR headsets for the purpose of immersive exploration of reconstructed scenes.

When considered as a whole, these contributions provide mature building blocks for each element of the pipeline under consideration in this work. Solid-state LiDAR devices, such as the Intel RealSense L515, offer accurate, high-density depth data in a form factor suitable for head-mounted operation. TSDF-based volumetric fusion, originating from Curless and Levoy's volumetric integration and popularised by real-time systems such as KinectFusion, yields a robust implicit representation of surfaces that naturally accommodates noisy and partial observations. In addition, Marching Cubes is recognised as a conventional and comprehensible technique for extracting polygonal isosurfaces from scalar fields, and it is especially well suited to TSDF volumes, where the zero level-set directly encodes the reconstructed geometry.

Concurrently, Unity and associated game-engine ecosystems, in conjunction with point-cloud renderers such as Pcx, facilitate the interactive visualisation of substantial point sets and reconstructed meshes, including within immersive VR environments. However, the majority of extant workflows treat these components in isolation. Specifically, depth sensors feed robotics-oriented SLAM or mapping frameworks; TSDF and Marching Cubes are implemented in standalone reconstruction pipelines; and Unity is primarily used as a visual front-end for pre-computed assets rather than as the locus of acquisition, processing and interaction.

The system delineated in this paper is situated precisely at the intersection of these strands. The system integrates a head-mounted L515 with an Unity/OpenXR environment designed for Meta Quest 3, captures coloured point-cloud snapshots as CSV/PLY pairs, and enables user-driven operations such as selection, merging, and geometric cleaning directly within VR. Furthermore, it integrates TSDF-based volumetric fusion and Marching Cubes mesh extraction as first-class components of the Unity application, rather than external post-processing steps. The subsequent section delineates the methodology underlying this architecture, elucidating the manner in which these established techniques are amalgamated and adapted to facilitate interactive, headset-centred spatial reconstruction.

3. Methodology

As shown in Fig. 1, the system proposed in this work is conceived as an interactive, headset-centred pipeline for the spatial reconstruction of indoor environments. Instead of adopting a fusion strategy that is purely real-time, in which all incoming frames are instantly assimilated into a global model, this approach is structured around the intentional acquisition of coloured point-cloud snapshots. These snapshots are captured from a head-mounted LiDAR camera, stored as persistent CSV and PLY files, curated and edited by the user within a VR interface, and finally integrated into a volumetric TSDF representation from which a triangle mesh approximating the TSDF iso-surface is extracted using Marching Cubes.

The following subsections describe hardware alignment, live acquisition, snapshot persistence, in-VR curation, and volumetric reconstruction. Together, these stages define a closed-loop workflow in which live preview supports user judgement, curated snapshots enable traceable dataset editing, and on-demand TSDF fusion produces an inspectable mesh in-headset.

3.1. Hardware configuration and spatial alignment

The system is characterised by a tight coupling between the physical sensing hardware and its virtual representation in Unity. The acquisition device is an Intel RealSense L515 LiDAR camera, which is rigidly mounted on the front of a Meta Quest 3 headset. This physical arrangement is not merely a practical convenience; it is a methodological choice that ensures that the pose of the LiDAR can be treated as a stable function of the headset pose. Provided that the mechanical mount remains undistorted and the camera is not repositioned, the relative transformation between the two frames can be considered constant during a scanning session.

As shown in Fig. 2, within the virtual environment this configuration is realised through an XR rig in Unity, where the head pose is continuously updated via OpenXR. A virtual LiDAR node is rigidly attached to the head-tracked transform, such that sensor motion and user motion share the same transformation chain. Initially, depth samples are back-projected into three dimensions in the sensor frame, and subsequently mapped to the scene frame through the current tracked transform, facilitating live rendering and interaction. It is important to note that this establishes a consistent in-application reference frame for inspecting acquisitions in VR, even though the persistent snapshot files themselves store point coordinates in the local frame of the point-cloud object.

This alignment has direct repercussions for the remainder of the pipeline. Upon capture of a snapshot by the user, the system responds by freezing the current point-cloud state and assigning it a scene transform corresponding to the sensor pose at the time of capture. However, the exported CSV/PLY pair encodes the point samples in the local coordinate frame of the snapshot object (i.e. in the same frame used by the point-cloud mesh). Within the Unity session, this is achieved by the retained object transforms and the shared tracking origin, which facilitates interactive inspection, selection, and downstream processing. Conversely, this design choice suggests that the exported files alone do not fully specify an absolute world pose outside the application, and the pipeline is primarily intended to remain traceable and reproducible within the same tracked session.

From a methodological perspective, the hardware configuration also includes an implicit calibration step. The intrinsic parameters of the L515 are taken from the manufacturer's calibration and used in the back-projection of depth pixels, while the extrinsic relationship between the LiDAR and the headset is enforced by the mechanical mount and the virtual parenting hierarchy. Further practical considerations, such as the avoidance of significant obstruction of the field of view of the camera by the headset shell and the minimisation of any compromise to user comfort that may result from additional weight, influence the duration of a session and the extent to which the user is able to move freely. However, these factors do not affect the fundamental geometric relationships on which the reconstruction relies.

3.2. Live acquisition and VR rendering

Once the device is configured and the Unity scene has been initialised, the L515 begins to stream depth and colour images. These streams are consumed in real time by a Unity component that acts as the acquisition and visualisation front-end. At each frame, a depth image is read, and its valid pixels are back-projected into 3D using the intrinsic parameters of the sensor. The back-projection step of the process is responsible for the generation of points in the LiDAR (sensor) frame. These points are stored as vertex positions in the local space of a dedicated point-cloud object, while the object's transform (driven by the tracked headset pose) maps them into Unity world space during rendering.

Figure 3 illustrates the live in-headset point-cloud preview generated from the LiDAR stream; in parallel, the colour image associated with the current depth frame is uploaded to a texture bound to the point-cloud material. The acquisition component assigns UV coordinates to each vertex in the point cloud so that they index the appropriate pixels in the colour texture. During rendering, the shader uses these UV coordinates to retrieve the correct colour for each point. The result is a dynamic Unity mesh whose vertices correspond to the 3D points derived from the LiDAR, and whose per-vertex colours are sampled from the RGB stream.

Crucially, this live point cloud is drawn directly in the VR display of the Meta Quest 3. As the user moves, tilts their head or steps around the scene, the apparent position of the cloud evolves accordingly. Surfaces that are in front of the user appear as dense clusters of coloured points; surfaces that are occluded or outside the sensor's range appear as gaps. Because the LiDAR is mounted on the headset, the simple act of moving through the environment becomes an act of "painting" the scene with 3D samples. When the user leans around a corner or approaches a piece of furniture, new parts of the scene come into view and are immediately represented in the live cloud.

The live visualisation is deliberately ephemeral in nature. The mesh utilised in this process undergoes a process of overwriting at every frame, with no integration over time into a persistent model. Its function is to serve as a constantly updated preview of what the sensor is currently seeing. This design choice entails the transfer of a substantial proportion of the responsibility for data selection from the algorithm to the user. The user is not a passive observer of an automatic mapping process, but an active participant who explores the space, monitors the quality and coverage of the live point cloud, and decides when the current view is informative enough to warrant a snapshot.

3.3. Snapshot logging and file formats

Persistent data enters the pipeline solely when the user initiates a snapshot. At that particular moment, the system takes the current state of the live point cloud and transforms it into a static dataset, which is then stored on disk for subsequent processing. Theoretically, this is the point at which transient sensor data become part of a reproducible reconstruction experiment.

Upon the request for a snapshot, the acquisition component decouples the current mesh from the live stream. Instead of reusing the same vertex buffer across successive frames, a frozen copy of the current vertices (and their associated UVs) is created as a static mesh, and the snapshot is instantiated as a dedicated scene object whose transform corresponds to the sensor pose at the time of capture. The decoupled object is no longer subject to the effects of subsequent frames. This results in the representation of a fixed observation from a specific viewpoint, while ensuring that pose information remains available in the Unity scene graph.

Prior to the snapshot being written to disk, its colours are rendered explicit. The colour texture corresponding to the current frame is sampled at each vertex's UV coordinate, and the resulting RGB value is written into the mesh's per-vertex colour array. Upon completion of this process, the snapshot becomes independent of the original texture, with vertex positions and colours collectively providing a comprehensive description of the coloured point cloud. This "baking" step is integral to the preservation of self-contained data, even in scenarios where textures are subsequently discarded or recomputed.

The snapshot is then exported in two complementary formats. In the CSV representation, each line corresponds to a single point and contains its local coordinates and colour channels, typically arranged as x, y, z, r, g, b. This file is plain text, easily inspected and processed by external tools, and suitable for numerical analysis or conversion into other formats. Concurrently, the same data are written into a binary little-endian PLY file. The header of this file describes the structure of the point records, while the body

contains interleaved position and colour attributes in a compact format. This PLY encoding has been demonstrated to be directly compatible with point-cloud renderers such as Pcx.

As shown in Fig. 4, each snapshot is exported in a PLY representation derived from the CSV point list, enabling immediate reloading in Unity point-cloud renderers. In order to maintain the organisation of the dataset, snapshots are stored in accordance with a simple yet systematic convention. The original CSV files are placed in a dedicated directory, and their corresponding PLY files are placed in a parallel directory with the same base file names. Each snapshot is assigned an identifier that increases incrementally, ensuring that the order of acquisition is reflected in the names and the association between CSV and PLY versions is unambiguous. Subsequent stages of the pipeline, such as merging or cleaning, result in the creation of additional CSV and PLY files in directories dedicated to derived products. However, clear links to the original snapshots are retained through their naming. This organisation supports reproducibility, as every reconstruction step can be traced back to specific input files, and it facilitates external inspection or reuse of the point clouds outside the Unity environment.

3.4. Interactive point-cloud management in VR

Following the acquisition and storage of a series of snapshots, the system then renders them as manipulable entities within the VR environment. The instantiation of each snapshot is initiated as a Unity GameObject, with the corresponding PLY file being loaded and rendered at the appropriate location in the scene. A user interface panel, presented as a floating canvas in front of the user, lists these clouds by name or index. Each entry in the list corresponds to one snapshot and provides controls for toggling its visibility, selecting it as part of a group, or removing it from the scene.

This interface has the capacity to transform the reconstruction process into a form of dataset curation that is carried out directly in VR. By enabling and disabling individual clouds, the user can assess the contribution of each snapshot to coverage, identify regions that are oversampled by overlapping views, and detect acquisitions that suffer from artefacts such as motion blur or sensor saturation. The elimination of such problematic clouds at an early stage in the process serves to minimise the amount of noise and redundancy that will subsequently be propagated into the TSDF volume.

The same interface also serves as the entry point for more complex operations, such as the merging of multiple point clouds into a single dataset or the geometric cleaning of clouds using an interactive clipping plane. It is important to note that these operations are not executed in the background; rather, they are initiated by the user and their outcomes are perceptible as new point-cloud objects within the scene. This approach enables the user to maintain a clear mental model of the evolution of the dataset over time, and to understand how each manipulation affects subsequent reconstruction. The resulting user-facing workflow is summarised in Fig. 5.

3.4.1. Merging multiple point clouds

Merging is employed when the user wishes to combine several partial snapshots into a more complete representation of a region of interest. Within the VR interface, the user designates a set of clouds as being associated with one another, for example, by selecting multiple views of the same room captured from different vantage points, thereby initiating the merge operation. Subsequently, the system processes the CSV files associated with the designated clouds, and the point sets contained within them are then concatenated into a single list.

Within the Unity session, merging is simplified because each snapshot is associated with a scene transform defined in a shared tracking origin. Since the CSV/PLY files encode points in each snapshot's local frame, points are first mapped into a common reconstruction frame by applying the corresponding object transform (captured at snapshot time). Under stable tracking, this provides practical co-registration without requiring an explicit registration stage (e.g., ICP), although slow drift and occasional relocalisation discontinuities can still introduce residual misalignments. When such misalignments become visually apparent during in-headset inspection, the workflow supports manual realignment by allowing the user to adjust the pose of individual snapshots (or derived merged artefacts) before proceeding to volumetric fusion. The merged set can optionally be filtered to reduce redundancy, for instance via a coarse voxel-grid downsampling or by discarding near-duplicate samples. This reduces the computational cost of subsequent volumetric integration and helps maintain a more balanced sample density across the scene.

Once the merged point set has been constructed, it is written to a new CSV file in a directory reserved for such merged data and is converted into a corresponding PLY file. In the Unity environment, a new GameObject is instantiated to load and display the merged cloud, and it is added as a new entry in the UI list. The original snapshots remain in existence unless the user elects to remove them; therefore, merging does not result in the destruction of information. Instead, it creates a new, more comprehensive cloud that the user can treat as a higher-level building block in the reconstruction. In methodological terms, merging thus provides a means of transitioning from a collection of local, viewpoint-specific snapshots to a more global representation while retaining visibility into every intermediate step. In the current prototype, this operation is performed entirely on the CPU. Consequently, runtime is primarily driven by the total number of points loaded from disk and processed in memory, which directly impacts perceived latency when many snapshots are selected.

3.4.2. Geometric cleaning via interactive clipping plane

In a multitude of indoor scanning scenarios, the raw point clouds contain structures that the user does not wish to include in the reconstruction. Typical examples include parts of the user's own body, areas beyond a doorway that fall outside the region of interest, or large expanses of background that contribute little to the intended model. In order to empower the user with control over undesired geometry, the system provides a geometric cleaning tool that is based on an interactive clipping plane.

Within the virtual reality environment, the user has the capacity to generate a planar object that manifests as a semi-transparent rectangle within the scene. The utilisation of these controllers enables the user to grasp and manipulate the plane, thereby translating it through space and rotating it around its centre. This action is analogous to the movement and tilting of a physical board. The plane thus functions as a delineator between two distinct half-spaces: one that the user wishes to retain and one which they wish to discard. By visually aligning the plane with architectural elements such as walls or floors, or by placing it in close proximity to unwanted clutter, the user effectively defines a spatial filter.

When the user applies this filter to one or more selected point clouds, the system reads the corresponding CSV files and evaluates, for each point, on which side of the plane it lies. Points falling within the half-space designated as retained are retained, while those falling within the opposite half-space are discarded. The retained points are written to new CSV files with a suffix indicating that they have been cleaned, and new PLY files are generated accordingly. The process of cloud cleaning results in the instantiation of new GameObjects, which are then listed in the UI alongside the originals. The user has the option to hide or delete uncleaned versions; however, these remain available until explicitly removed, thereby ensuring that the cleaning operation is reversible at the level of data management.

This plane-based cleaning mechanism is designed to be simple in its conception, yet it has been shown to be surprisingly effective. The system does not necessitate a comprehensive semantic understanding of the scene or the employment of sophisticated machine learning models. Instead, it relies on the user's capacity to interpret the geometry visually and to position the plane accordingly. From a methodological perspective, it provides a coarse but controllable method of reducing clutter and removing systematic artefacts prior to volumetric fusion. This process enhances the quality and interpretability of the final reconstruction.

3.5. TSDF construction and Marching Cubes mesh extraction

Following the acquisition of a curated set of point clouds, the system initiates volumetric reconstruction. At this stage, the data consists of one or more CSV/PLY snapshots containing coloured points in their object-local frames, accompanied (within the Unity session) by the scene transforms that place each snapshot in a shared tracking origin. Prior to TSDF integration, samples are mapped into the reconstruction frame using these transforms, and the objective becomes to convert the resulting spatial point set into a continuous implicit surface via a truncated signed distance field, from which a polygonal mesh is extracted using Marching Cubes.

The initial step in this process is to determine the spatial domain over which the TSDF will be defined. The process entails mapping each snapshot's local samples into the reconstruction frame using its scene transform, then computing an axis-aligned bounding box over the union of the mapped points. The bounding box is extended slightly in order to ensure that the entire set of points lies strictly inside the volume. This process is then used to define the origin and extent of a regular voxel grid. The selection of voxel size dictates the number of voxels along each axis, thereby determining the resolution of the TSDF. While finer voxels

ensure greater detail capture, they also necessitate greater memory consumption and computational demands. The truncation distance, expressed in physical units and typically set to a small multiple of the voxel size, specifies the thickness of the band around each observed surface within which depth measurements influence the TSDF values.

The TSDF is stored as a three-dimensional array of floating-point values and initialized to a large positive constant to denote unobserved space. In the present study, the TSDF is represented as a dense 128³ grid. With Δ = 0.04 m, this corresponds to a cubic workspace of approximately 5.12 m per side, which is sufficient for the preliminary indoor reconstructions targeted here. A dense 128³ grid contains 2,097,152 voxels; storing a single floating-point TSDF value per voxel requires approximately 8 MB, with additional per-voxel attributes (e.g., weights or gradients) increasing this budget accordingly. This fixed-volume design was selected to keep integration and meshing predictable within a PC-tethered Unity runtime. In our setting, the input consists of point samples rather than a dense projective depth image; therefore, the sign of the distance field must be defined through an observation model. The common volumetric fusion assumption for range data is adopted, namely that each sample corresponds to the first surface intersection along a ray originating from the sensor and that the sensor remains in free space [5], [7]. For each snapshot, we use the sensor origin o at capture time and each sample point p mapped to the reconstruction frame. Let $\mathbf{r} = (\mathbf{p} - \mathbf{o})/\ \|\ \mathbf{p} - \mathbf{o}\|$ be the viewing direction and $s = \|\ \mathbf{p} - \mathbf{o}\|$ the measured range. For a voxel centre $\mathbf{v}$, the signed distance along the ray is defined as:

$$\varphi\ (v) = \mathrm{clamp}\left(\frac{r \cdot\ (v - o) - s}{\mu}, -1{,}1\right)$$

where μ is the truncation distance (i.e., the half-width of the truncation band). Intuitively, voxels in front of the observed surface along the ray yield positive values (free space), whereas voxels behind the surface within the truncation band yield negative values [5], [7]. As samples are integrated, each voxel stores the minimum between its current TSDF value and the newly computed value. This conservative fusion rule biases the field toward the closest observed surface and preserves sharp localisation, at the expense of the noise-smoothing behaviour typically obtained through confidence-weighted averaging in classical TSDF pipelines [5], [6], [7].

For each intersected edge, the algorithm interpolates between the two endpoint voxel centres, using their TSDF values to estimate the precise location of the zero crossing along the edge. The resulting vertex positions are initially expressed in grid coordinates; they are converted into world coordinates by multiplying by the voxel size and adding the grid origin. By establishing connections between these vertices in accordance with the patterns delineated in the reference table, the algorithm generates a series of triangles that approximate the zero level-set of the TSDF over the entire volume.

The aggregation of all triangles constitutes a polygonal mesh that collectively represents the reconstructed environment. The mesh is then assigned to a Unity Mesh object, given an appropriate material and inserted into the scene as a new GameObject. The TSDF grid is defined within the same reconstruction frame as the curated point clouds and the XR rig; therefore, the extracted mesh is co-registered with both the input samples and the user's tracked viewpoint. Consequently, the mesh manifests at the intended spatial location relative to the original data, thereby facilitating direct in-headset inspection.

4. System Architecture and Unity Implementation

The methodology described above is realised as a Unity-based XR application, in which acquisition, persistence, curation, and reconstruction are implemented as a single, tightly coupled workflow. The overarching architectural principle is that every intermediate result is materialised—first as a visible object in the virtual environment and, in parallel, as a persistent file on disk—so that reconstruction remains reproducible and user-auditable at each stage.

In contrast to the conventional approach of utilising Unity as a passive visual front-end for offline datasets, the present system employs Unity as the execution environment. Within this environment, point clouds are generated from live sensor streams, captured as snapshots, curated through in-VR interaction, and ultimately converted into an explicit mesh through volumetric integration and isosurface extraction.

4.1. Unity scene organisation and coordinate consistency

The Unity scene is organised around a standard XR rig, in which head pose is updated by OpenXR tracking and drives the user's viewpoint and interaction frame. The LiDAR device is modelled as a virtual sensor node rigidly attached to the head-tracked transform. This choice is not merely a matter of convenience for visualisation; it defines the reference frame in which measurements are interpreted and ultimately stored. In practice, each depth sample is back-projected into a 3D point in the sensor frame and then mapped into Unity world coordinates through the current tracked pose. Consequently, each snapshot can be *placed* in a shared world frame through its retained object transform, while the persisted CSV/PLY files store point samples in the snapshot's local frame.

The design is associated with two significant architectural implications. Firstly, it enforces pose-consistent accumulation across time. As the user moves through the physical environment, the headset's tracking continuously updates the sensor-to-world transform. This means that snapshots captured at different times share a coherent spatial relationship without requiring a separate point-cloud registration stage. While this does not entirely eliminate all sources of misalignment (e.g. tracking drift or depth noise), it provides a stable foundation for interactive multi-snapshot workflows such as cloud merging, clipping-based cleaning, and volumetric fusion.

Secondly, the same coordinate convention is preserved downstream for reconstruction outputs. When a TSDF volume is defined for a selected set of clouds, its bounding region is derived from the world-space extent of the curated point set, and the grid origin is placed in world coordinates accordingly. Following the extraction process, the reconstructed mesh vertices are converted back into world coordinates using the same voxel-to-world mapping. The utilisation of a shared geometric language is imperative in the context of VR, wherein the user assesses the reconstruction quality by directly comparing the mesh to the original point samples through occlusion relationships, parallax, and spatial inspection at multiple distances. Therefore, co-registration is not only a correctness criterion but also a perceptual requirement for reliable in-headset validation.

From a Unity engineering perspective, the scene hierarchy is kept explicitly defined: persistent point clouds and reconstructed meshes are instantiated as independent scene objects with stable transforms, rather than being continuously overwritten. This facilitates predictable interaction and enables the user to toggle visibility, select subsets, and position auxiliary tools (e.g., a cleaning plane) in the same spatial frame. Furthermore, it facilitates subsequent extensions, including anchoring results, exporting spatial metadata, and adding session-level provenance, thereby ensuring the traceability of each derived artefact back to an acquisition pose and a corresponding set of input clouds.

4.2. Runtime data flow: from live stream to persistent snapshots

During the execution of the system, a two-tier data flow is observed, whereby high-frequency visual feedback is distinguished from low-frequency persistent logging. The initial tier constitutes a continuous, frame-driven preview, whereby depth and colour frames are streamed from the LiDAR, converted into a point set, and rendered instantaneously as a dynamic point cloud. This preview is explicitly treated as ephemeral, with points being updated or replaced as new frames arrive, because its primary function is interactive guidance. The headset enables the user to transition to novel viewpoints, assess the adequacy of observation of a region, and identify problematic conditions (e.g., missing surfaces due to range limitations, specular materials, or occlusions) prior to the storage of data. This choice has the effect of reducing unnecessary disk writes and preventing the persistence layer from being flooded with redundant frames.

The second tier comprises a user-initiated snapshot pathway. The process of recording initiates upon the selection of the relevant option by the user. This action results in the current point set being rendered as a static dataset and subsequently converted into a self-contained representation. A pivotal step in this process is the consolidation of colour information: rather than leaving colour implicit in a transient texture stream, the system samples the RGB image at each point's texture coordinates and stores the resulting colour as a per-point attribute. From an architectural perspective, this "baking" step (i.e., materialising transient sensor streams into persistent per-point attributes) is significant, as it renders the snapshot independent of the acquisition device state. This enables the point cloud to be reloaded, merged, filtered, and reconstructed without the necessity for a live stream or external texture references.

Following the freezing process, the snapshot is exported through a dual persistence strategy. A CSV file provides a transparent, inspection-friendly representation suitable for debugging, external analysis, and reproducible experimentation. Concurrently, a

binary PLY file furnishes a standardised and efficient asset that can be expeditiously reloaded for visualisation and downstream processing. The naming policy maintains a one-to-one correspondence between these two formats and encodes acquisition order through progressive identifiers. This directly supports provenance, insofar as each visual object shown in VR corresponds to a specific on-disk dataset, which can be referenced unambiguously in later processing steps.

This process is repeated until the newly created snapshot is reintroduced into the Unity scene as an independent object, thereby becoming part of the user's working set. The system then simultaneously registers the image in a VR UI panel, thereby enabling subsequent selection and curation operations (e.g., inclusion in a merge or reconstruction, exclusion of defective captures). Overall, the runtime flow forms a closed loop: live previews inform user judgement, which triggers snapshot creation; snapshots then become interactive entities that feed subsequent curation and reconstruction. The separation between the two tiers is not merely an optimisation; rather, it is a deliberate interaction design decision that aligns computational cost with the user's intent and preserves an auditable sequence of intermediate results.

4.3. File-System persistence and dataset traceability

A fundamental stipulation of the proposed workflow is that each acquisition is preserved in a form that is both inspectable and reusable across different processing stages. Consequently, persistence is implemented as a dual-representation strategy, whereby every snapshot is written to disk in two different formats. Firstly, it is written as a human-readable, line-based tabular format. Secondly, it is written as a compact, widely supported point-cloud format containing the same geometric and photometric attributes. The initial representation facilitates transparency and debugging (e.g., verifying coordinate ranges, detecting outliers, or auditing colour values), while the subsequent representation supports efficient reloading and integration with point-cloud visualisation pipelines.

From an architectural perspective, persistence is not merely regarded as a "export option"; rather, it is considered a premier mechanism for provenance. The system enforces a stable folder structure and a monotonic naming convention so that each stored dataset has an unambiguous identity and a predictable location. This decision assumes particular significance once point clouds are curated and transformed, since the processes of merging, cleaning, and reconstruction generate new artefacts rather than overwriting existing ones. Consequently, the progression of data through the pipeline can be reconstructed post hoc from the file system alone, thereby enabling reproducible experimentation (e.g., re-running TSDF integration with different voxel sizes) and supporting paper-grade reporting of which inputs produced which outputs.

Furthermore, the separation of original snapshots from derived datasets (e.g., merged or cleaned clouds) has been shown to reduce the risk of accidental data loss and to encourage systematic comparisons. For instance, users can verify the removal of structurally relevant points during a cleaning operation by toggling between the original and the filtered version, or assess the introduction of density artefacts in a merged cloud in comparison to its constituent parts. This "non-destructive editing" model mirrors established practices in 3D content creation and is particularly valuable in immersive workflows, where immediate visual feedback and reversible operations improve user trust in the reconstruction outcome.

4.4. In-VR point-cloud management and interaction design

Once snapshots are persisted, they become interactive entities in the XR environment. The system provides an in-VR management panel that exposes the set of available point clouds and supports basic operations such as visibility control and selection for downstream processing. This UI layer serves a dual purpose: it is an ergonomic solution for interacting with potentially large numbers of acquisitions, and it formalises dataset curation as an explicit step between capture and reconstruction.

The curation of these point sets is of critical importance, given the fact that LiDAR-based point sets acquired in mobile conditions are seldom uniform in quality. It should be noted that some captured images may contain occlusion gaps, depth discontinuities at object boundaries, or noise induced by motion and reflective materials. By enabling the user to review and selectively include snapshots, the pipeline can prevent these issues from propagating into volumetric fusion, where they may manifest as inflated surfaces, holes, or spurious geometry. In this sense, the VR interface does not merely "control" the pipeline; it embodies an interaction strategy in which the user is kept in the loop as a quality gate.

Beyond the initial selection stage, the system facilitates higher-level operations that convert collections of snapshots into new datasets. A primary example of this process is merging: selected point clouds can be aggregated into a single combined cloud

expressed in a shared world coordinate frame. In instances where it is deemed necessary, an additional redundancy reduction step may be implemented with a view to mitigating oversampling and reducing computational load in subsequent stages. The amalgamated dataset is subsequently stored and initiated as a novel interactive entity, thereby enabling the user to conceptualise it as a consolidated depiction of a scene region while retaining the capacity to revert to the original constituent clouds.

A complementary operation is geometric cleaning via an interactively positioned clipping plane. In this instance, the user defines a spatial constraint in VR (a half-space) and removes points outside the region of interest. This mechanism is particularly effective for the following processes: the elimination of background clutter; the truncation of incomplete scans near range limits; and the isolation of specific objects or room portions prior to TSDF integration. As with merging, cleaning is implemented as a non-destructive operator. This generates a new artefact that can be compared against its source, thereby maintaining both usability and auditability.

4.5. Reconstruction integration and VR inspection of results

Subsequent to the curation of the dataset, the reconstruction stage transforms discrete point samples into an explicit surface that can be inspected and utilised in downstream applications. The system performs this transformation through volumetric integration, defining a TSDF volume over the spatial extent of the selected point clouds. The volume bounds are derived from the world-space distribution of input points, and the grid is parameterised by a voxel size and a truncation distance that determine the resolution and the robustness of the implicit field. A representative in-VR visualisation of the resulting TSDF grid is shown in Fig. 6.

The TSDF representation offers a practical advantage in this architecture by providing a method to regularise noisy, non-uniform point density and to derive a continuous surface, even when the input sampling is sparse. The integration of observations into a voxel grid, in conjunction with the encoding of surface proximity via signed distance values, results in the generation of a volumetric model that is optimally suited to isosurface extraction. The subsequent Marching Cubes step produces a triangle mesh that approximates the zero level-set of the TSDF, yielding a surface suitable for Unity rendering and interaction.

A fundamental implementation prerequisite is the assurance of spatial coherence between inputs and outputs. Both the TSDF grid definition and the mesh vertex generation are expressed in the same reconstruction (scene) frame via voxel-to-world mapping (voxel size and grid origin), anchored to the shared tracking origin. Consequently, the reconstructed surface is co-registered with the curated point clouds and with the XR rig. This facilitates direct qualitative validation in VR: users can toggle clouds on and off while observing whether the mesh follows the same structural boundaries, whether it bridges gaps plausibly, and whether artefacts appear in regions with poor coverage. Such inspection is facilitated by embodied perception (i.e. parallax and viewpoint changes), which is challenging to replicate in a 2D desktop workflow.

Furthermore, the mesh is instantiated as a persistent scene object rather than as a transient visualisation. This facilitates iterative workflows in which the user can adjust reconstruction parameters, regenerate the surface, and compare multiple results within the same session. The architecture therefore naturally supports empirical tuning – an essential aspect of TSDF-based reconstruction – without requiring the user to leave the immersive environment.

4.6. Practical constraints and extensibility considerations

The present Unity implementation has been shaped by pragmatic decisions that prioritise traceability and rapid iteration, while also delineating clear targets for future refinement.

Firstly, the file-based persistence strategy introduces an asset synchronisation requirement: newly generated point clouds must become available for immediate reloading and visualisation. This approach is effective for development and experimentation because it externalises intermediate states and makes them inspectable. In the present setup, where the VR experience is tethered to a PC to support the live LiDAR connection, reliance on an editor-friendly import pathway is an acceptable engineering trade-off. Nonetheless, a natural refinement for future iterations would be to adopt a runtime-safe import mechanism (e.g., streaming parsing or direct GPU buffer uploads) to reduce dependency on editor-only refresh behaviours while preserving the same traceability guarantees.

Secondly, the separation between live preview and snapshot persistence is a deliberate interaction and performance trade-off. The fusion of all frames into a global model would increase computational pressure and complicate provenance. In contrast, the

present approach aligns heavy processing with explicit user intent (capture, curate, reconstruct). The principal disadvantage of this methodology is that reconstruction is not an ongoing process; it is executed by the user when the dataset is deemed to be ready. In practice, this trade-off is consistent with the application goal of enabling reliable, interpretable reconstruction rather than maximising automation at the expense of control.

Thirdly, the quality of spatial consistency is constrained by upstream sensing and tracking. While the utilisation of head-pose attachment facilitates a coherent transformation chain, factors such as depth noise, sensor-specific artefacts, and XR tracking drift may still have an impact on alignment and surface quality, and short-lived tracking failures may lead to discrete pose discontinuities upon relocalisation. The architecture has been developed to address these issues through the implementation of curation tools (selection, merging policies, clipping-based cleaning) and through volumetric regularisation. However, the study also identifies the need for further research in the form of optional refinement modules such as outlier rejection, point-to-plane filtering, or lightweight registration between snapshots when tracking uncertainty is detected.

In conclusion, the division of tasks into distinct modules, with acquisition, curation and reconstruction occurring sequentially, offers a potential solution to the challenge of achieving scalability. For instance, the same pipeline can be extended to incremental reconstruction (updating the TSDF as new snapshots arrive), to spatial partitioning strategies for large environments, or to multi-resolution volumes that adapt voxel size to distance or region complexity. It is important to note that these extensions can be introduced without altering the core data-management principle of explicit, auditable intermediate artefacts, which remains central to the system's scientific and engineering objectives.

5. Results

The system was evaluated on a PC-tethered VR setup using an Intel Core i5-9600K CPU and an NVIDIA GeForce RTX 2080 GPU, with the headset and sensor operating within the same tracked Unity session. As illustrated in Table 1, an overview of the end-to-end performance for the acquisition, persistence, and reconstruction pipeline is provided. The mean value ± standard deviation is reported over N = 10 runs for a representative workload (three snapshots). Each incoming depth frame has a resolution of 786,432 pixels (1024×768), from which valid depth samples are back-projected into points. During the process of snapshot export, an average of approximately 7,864 points (i.e. 1%) is removed by the cleaning step in this configuration, resulting in 778,568 points per snapshot and 2,335,704 points prior to merging. The merge stage reduces the aggregated set to 748,305 points, representing the dominant cost in the tested configuration (12.0 ± 4.0 s). In comparison, TSDF integration and Marching Cubes extraction are comparatively lightweight (2.0 ± 1.0 s and 1.0 ± 0.5 s, respectively) under a 128³ grid with Δ = 0.04 m and μ = 0.10 m. Mesh extraction frequently approaches the imposed triangle budget, yielding 370k–400k triangles (cap = 400k), indicating that the chosen volumetric resolution produces high surface complexity for the considered scenes.

Table 1
End-to-end pipeline performance and reconstruction settings (mean ± std over N = 10 runs).

Stage	Input (typical)	Output	Time (s) mean ± std
Snapshot freeze (mesh copy)	786,432 points/snapshot	snapshot GO	0.2 ± 0.1
CSV export + cleaning	786,432 points/snapshot	778,568 points (≈ 7,864 removed) + CSV file	3.0 ± 1.0
PLY export (binary little-endian)	778,568 points/snapshot	PLY file	2.0 ± 1.0
Merge selected snapshots	k = 3; total points = 2,335,704	748,305 merged points	12.0 ± 4.0
TSDF integration	748,305 points	TSDF grid (128³)	2.0 ± 1.0
Marching Cubes	TSDF grid (128³)	370k–400k triangles (cap=400k)	1.0 ± 0.5

The measurements indicate that the proposed workflow is practically usable for on-demand reconstruction within a tethered VR setting. The snapshot persistence and TSDF/Marching Cubes reconstruction remain within a few seconds, while the merge stage

constitutes the primary computational bottleneck for the tested workload. While this latency is deemed acceptable for smaller snapshot sets when utilising a tethered setup, it has the potential to disrupt larger collections. This prompts the scalability measures outlined above. This finding indicates that subsequent optimisation endeavours should accord a high priority to merge scalability (e.g., multi-threaded CPU aggregation with spatial hashing/voxel bucketing to avoid global de-duplication passes, and streaming writes to reduce peak memory usage). Conversely, volumetric integration and mesh extraction have already demonstrated favourable runtime under the selected 128^3 configuration. The findings of this study provide a foundation for the conclusions that are presented herein. These conclusions summarise the main architectural trade-offs and outline directions for improving robustness under tracking drift and increasing scalability to larger environments. A representative surface reconstruction outcome produced by the TSDF and Marching Cubes stages is shown in Fig. 7.

6. Conclusion

The paper presented a Unity/OpenXR workflow for headset-centred acquisition, curation, and reconstruction of indoor environments using a head-mounted solid-state LiDAR. The system has been designed around the concept of traceable intermediate artefacts. Specifically, snapshots are captured intentionally, persisted as CSV/PLY pairs, curated through in-VR interaction (selection, merging, and optional clipping-plane cleaning), and reconstructed on demand through TSDF volumetric integration followed by Marching Cubes mesh extraction. It is evident that by ensuring all stages are contained within a single XR application, the workflow facilitates direct in-headset validation of reconstruction outputs against the underlying point-cloud evidence.

The intention behind the work's scope is to contribute to the fields of architecture and workflow. Consequently, the evaluation is oriented towards end-to-end feasibility (i.e. latency, persistence, interaction, and traceable provenance) as opposed to metric accuracy against external ground truth. While voxel size (Δ) defines the sampling resolution of the volumetric representation, it does not constitute an accuracy guarantee; a dedicated geometric fidelity study (e.g. point-to-surface distances against laser scans or CAD references) is therefore left to subsequent work.

Quantitative results under a representative workload indicate that snapshot persistence and volumetric reconstruction remain feasible within a few seconds. Multi-snapshot merging constitutes the primary computational bottleneck in the tested configuration. Frequently, mesh extraction approaches the imposed triangle budget under a 128^3 volume, suggesting that surface complexity is strongly influenced by volumetric resolution and the chosen mesh cap. The findings support the practicality of an interaction-driven reconstruction strategy, whereby computationally intensive operations are triggered by user intent rather than executed continuously.

However, it should be noted that there are several limitations that persist. The spatial consistency of the resultant dataset is contingent upon the accuracy of the upstream sensing and tracking processes. Various factors, including depth noise, the presence of reflective materials, slow drift, and intermittent relocalisation discontinuities, have the potential to introduce residual misalignments. These misalignments propagate into the merged dataset and the reconstructed surface, compromising the integrity and accuracy of the final product. Furthermore, while the present pipeline prioritises auditability and reproducibility, further optimisation of merge scalability and memory management will be required to support substantially larger environments and longer sessions. Consequently, future endeavours will concentrate on enhancing scalability through the implementation of more efficient aggregation and spatial partitioning strategies. Furthermore, enhancing robustness will be achieved through the incorporation of lightweight outlier rejection mechanisms and optional inter-snapshot refinement procedures, as well as user-driven pose correction (manual realignment) when tracking uncertainty becomes visible during in-headset inspection. Finally, the reconstruction stage will be extended towards incremental updates that preserve the same principles of intermediate artefact traceability within immersive workflows.

Declarations

Informed Consent

Not applicable. This study did not involve human participants or patient data.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Author Contribution

S.C. conceived and designed the study, developed the research methodology, carried out the investigation, and drafted the manuscript. G.M.S. contributed to the evaluation and validation of the research and revised the manuscript critically for important intellectual content. D.F. validated the research data and revised the manuscript critically for important intellectual content. A.L. proposed the project, supervised the research, and validated its overall development. All authors read and approved the final manuscript and agree to be accountable for all aspects of the work.

References

1. Intel, 'Intel® RealSenseTM LiDAR Camera L515 Datasheet'. [Online]. Available: https://dev.intelrealsense.com/docs/lidar-camera-l515-datasheet
2. T. Whelan, S. Leutenegger, R. Salas Moreno, B. Glocker, and A. Davison, 'ElasticFusion: Dense SLAM Without A Pose Graph', in *Robotics: Science and Systems XI*, Robotics: Science and Systems Foundation, Jul. 2015. doi: 10.15607/RSS.2015.XI.001.
3. A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, 'BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration', *ACM Trans. Graph.*, vol. 36, no. 3, pp. 1–18, Jun. 2017, doi: 10.1145/3054739.
4. H. Wang, C. Wang, and L. Xie, 'Lightweight 3-D Localization and Mapping for Solid-State LiDAR', *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 1801–1807, Apr. 2021, doi: 10.1109/LRA.2021.3060392.
5. B. Curless and M. Levoy, 'A Volumetric Method for Building Complex Models from Range Images', *Proc. 23rd Annu. Conf. Comput. Graph. Interact. Tech.*, pp. 303–312, Aug. 1996, doi: 10.1145/237170.237269.
6. R. A. Newcombe *et al.*, 'KinectFusion: Real-Time Dense Surface Mapping and Tracking', *Proc. 2011 10th IEEE Int. Symp. Mix. Augment. Real.*, pp. 127–36, Oct. 2011.
7. H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, 'Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning', Sep. 2017, pp. 1366–1373. doi: 10.1109/IROS.2017.8202315.
8. M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, 'Real-time 3D reconstruction at scale using voxel hashing', *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1–11, Nov. 2013, doi: 10.1145/2508363.2508374.
9. V. A. Prisacariu *et al.*, 'InfiniTAM v3: A Framework for Large-Scale 3D Reconstruction with Loop Closure', Aug. 02, 2017, *arXiv*: arXiv:1708.00783. doi: 10.48550/arXiv.1708.00783.
10. W. E. Lorensen and H. E. Cline, 'Marching cubes: A high resolution 3D surface construction algorithm', *SIGGRAPH Comput. Graph.*, vol. 21, pp. 163–169, Aug. 1987, doi: 10.1145/37402.37422.
11. E. V. Chernyaev, 'Construction of Topologically Correct Isosurfaces Marching Cubes 33':, *Graphicon'95*, Jul. 1995, [Online]. Available: https://www.researchgate.net/publication/2651412_Marching_Cubes_33_Construction_of_Topologically_Correct_Isosurfaces
12. T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares, 'Efficient Implementation of Marching Cubes' Cases with Topological Guarantees', *J. Graph. Tools*, vol. 8, no. 2, pp. 1–15, Jan. 2003, doi: 10.1080/10867651.2003.10487582.
13. T. Keijiro, *Pcx: Point Cloud Importer & Renderer for Unity*. (2019). [Online]. Available: https://github.com/keijiro/Pcx
14. S. Rusinkiewicz and M. Levoy, 'QSplat: a multiresolution point rendering system for large meshes', in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, Not Known: ACM Press, 2000, pp. 343–352. doi: 10.1145/344779.344940.
15. M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, 'Surface splatting', in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, Aug. 2001, pp. 371–378. doi: 10.1145/383259.383300.

Figures

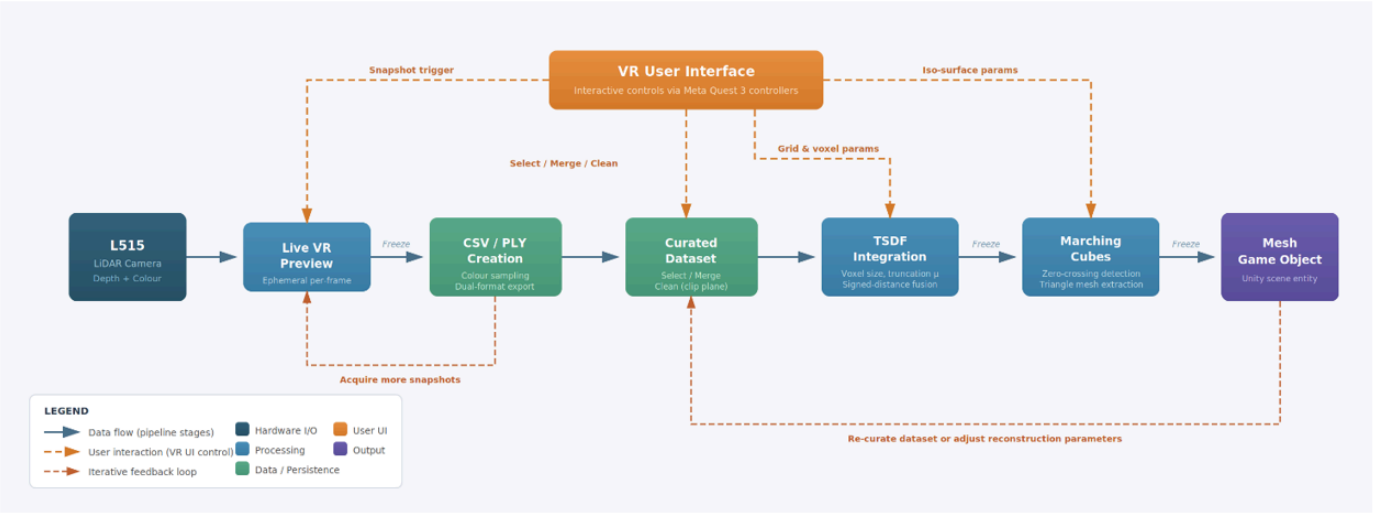


Figure 1

System Overview.



Figure 2

Hardware setup. The Lidar sensor is mounted on the Meta Quest 3.

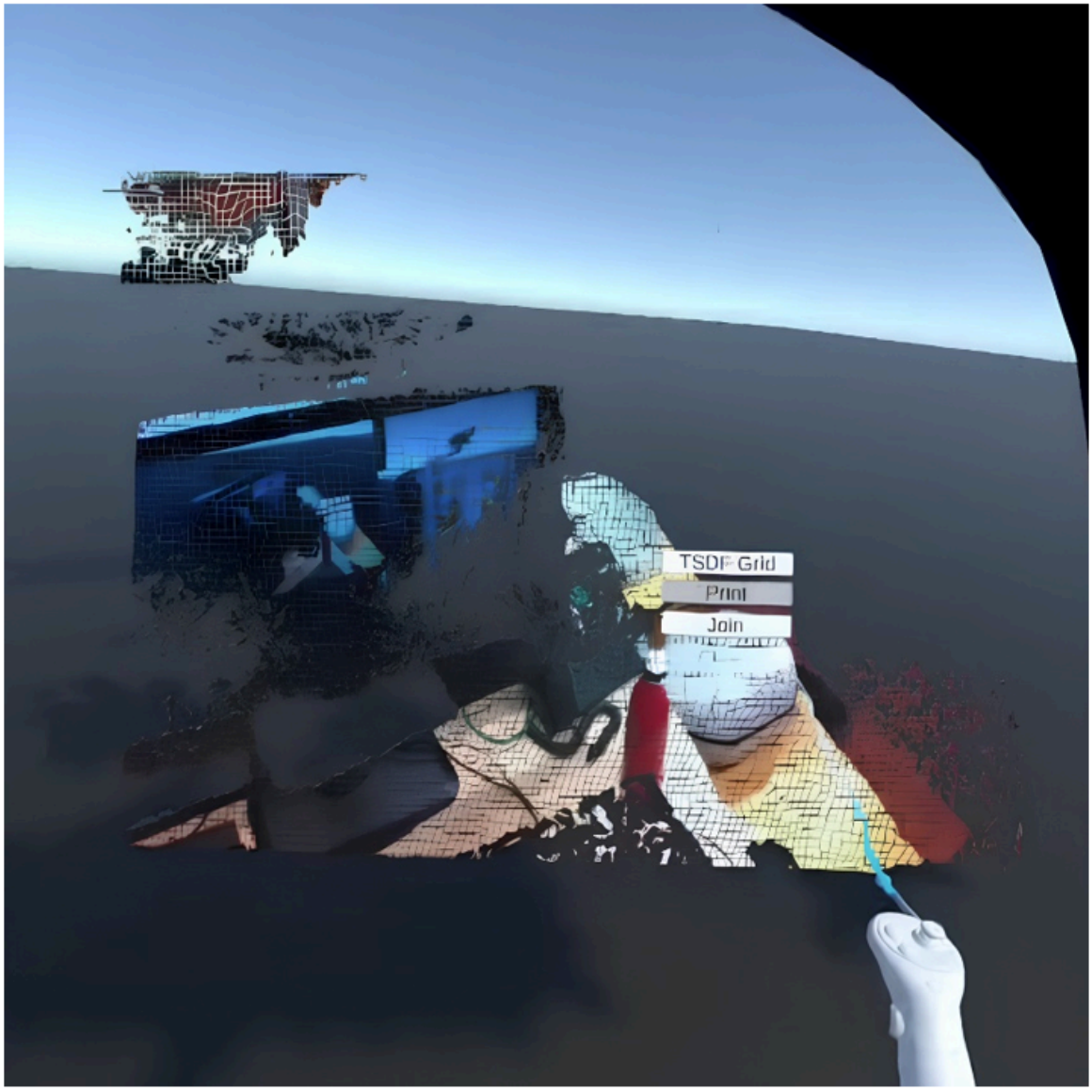


Figure 3

Live streaming of the Lidar Camera.

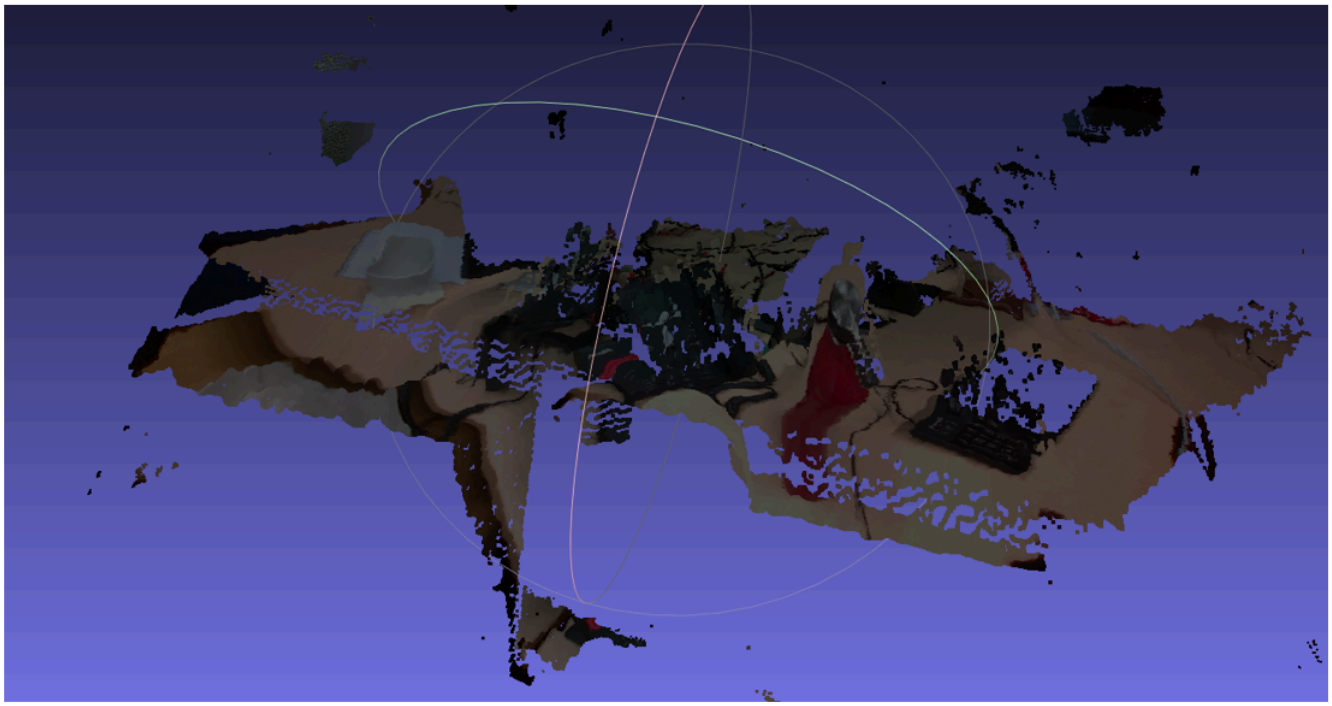


Figure 4

PLY-File created from the CSV.

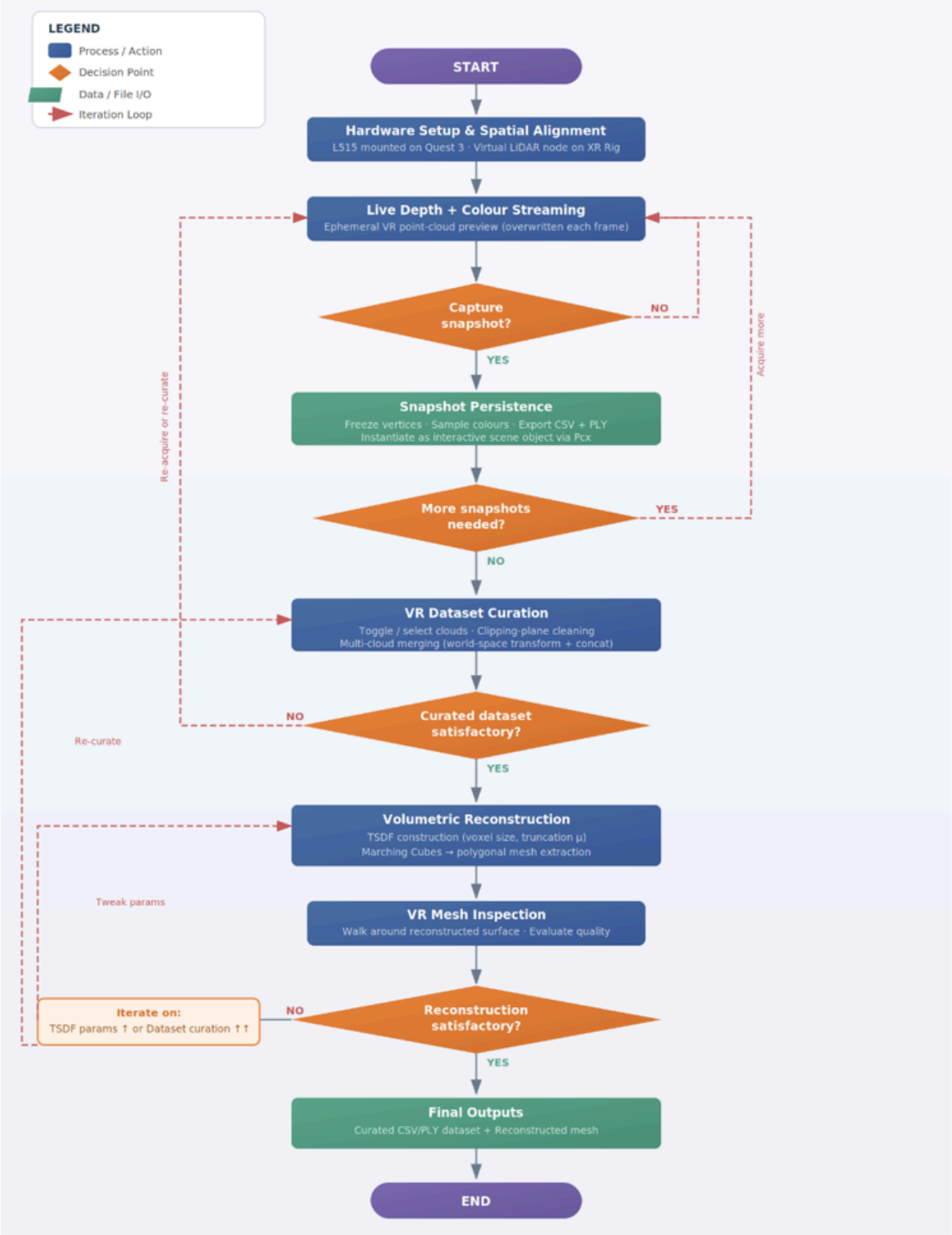


Figure 5

VR UI Workflow.

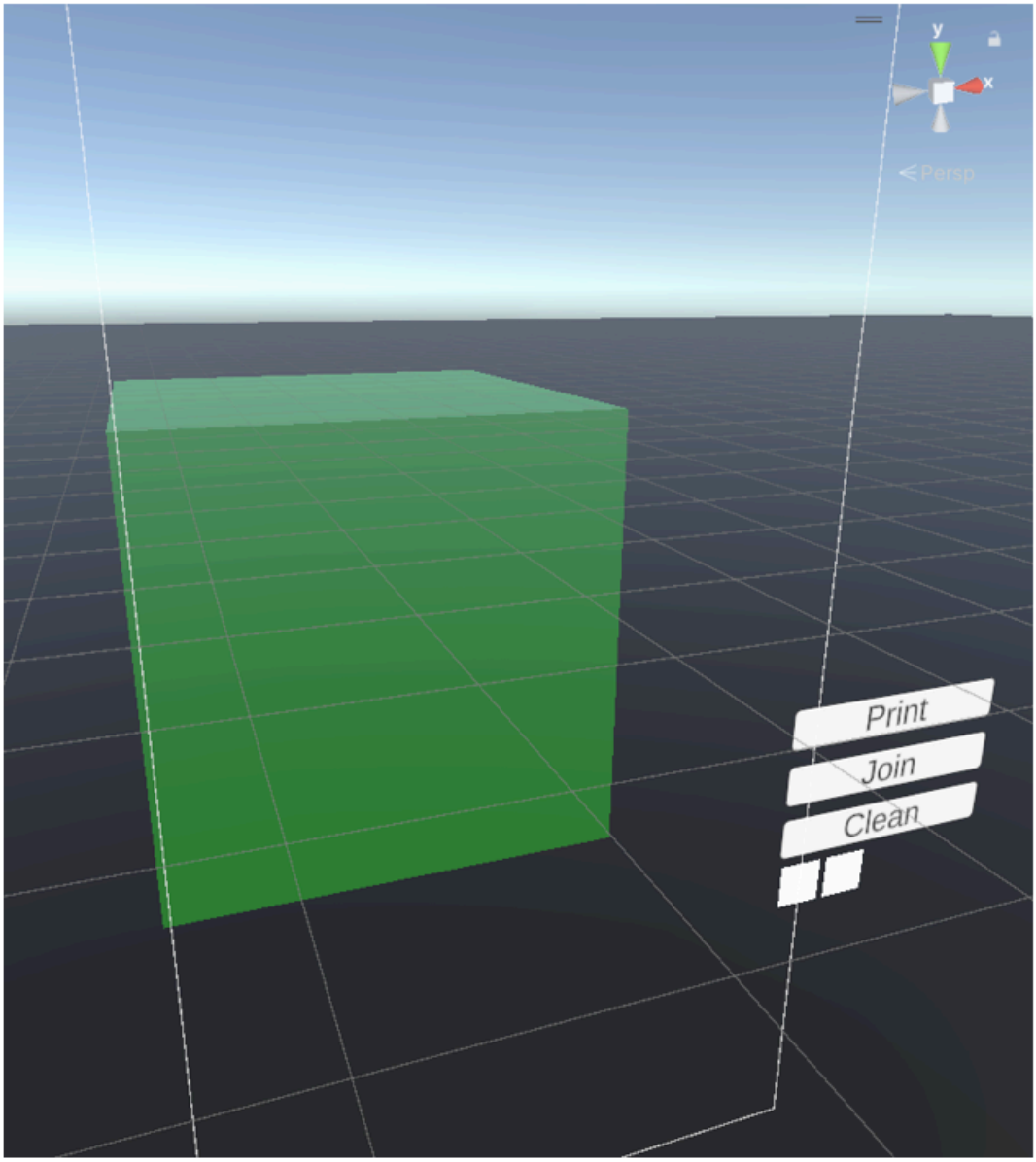


Figure 6

Visualisation of the TSDF grid (128^3, $\Delta = 0.04$ m) in the reconstruction frame.