LiteReality-Agent: An Agentic System for Interactable 3D Indoor Scene Reconstruction

PUBLISHED
August 4, 2026

AUTHORS
Zhening Huang, Yueyan Li, Johnathan Chiu, Xiaoyang Lyu, Matt Zhou, Yuxin Yao, Joan Lasenby, Shangzhe Wu

LINKS
LiteReality-Agent (https://litereality.github.io/Litereality-agent-site/) · Code (https://github.com/LiteReality/LiteReality-Agent) · App (https://apps.apple.com/ph/app/litereality/id6774158260)

Abstract

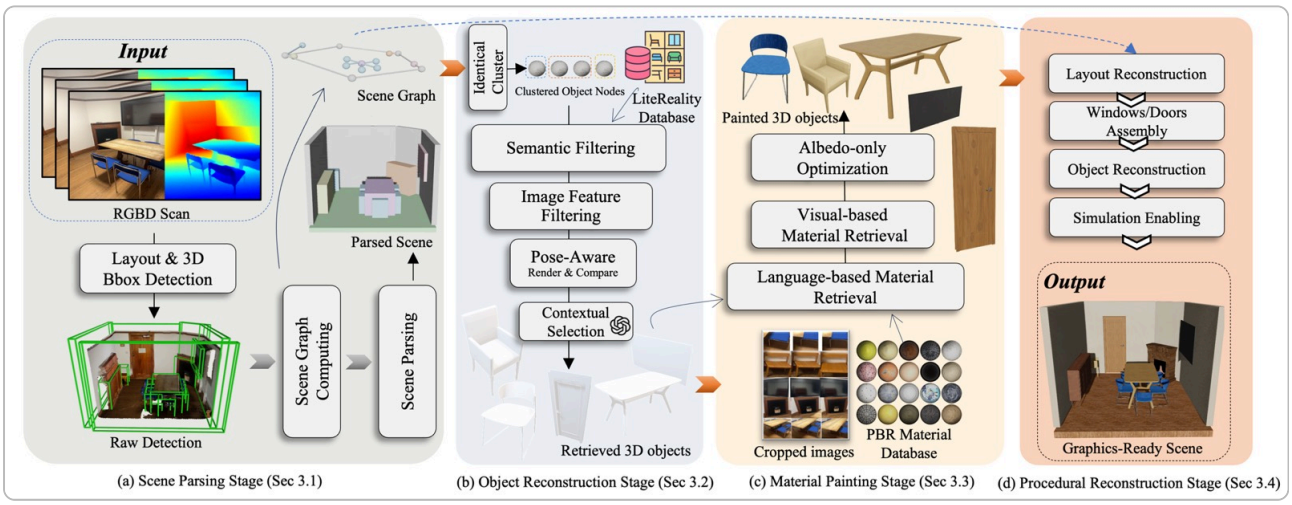
An open-source, end-to-end toolkit for reconstructing interactable indoor 3D scenes. Scan a room with the iOS app (https://apps.apple.com/ph/app/litereality/id6774158260); the agent turns it into a complete, graphics-ready scene with articulated assets.

LiteReality-Agent reconstructs interactable 3D scenes from RGB-D scans.

Intro and background

About a year ago, we introduced LiteReality (https://litereality.github.io/), a system that turns a real-world scan into a graphics-ready environment. The goal was to make reconstruction compatible with the traditional graphics pipeline, so a scanned room could drop straight into downstream applications like gaming, simulation, and virtual reality.

Much of that effort went into a handcrafted pipeline that included layout parsing, asset retrieval, and material painting, all stitched together to achieve a realistic result. While it worked to some extent, it was difficult to scale to more challenging environments and could only capture what it had been specifically designed to handle.



The LiteReality (https://litereality.github.io/) pipeline — a handcrafted workflow.

This year, we witnessed a surge in capabilities of coding agents for 3D modeling. Projects such as Articraft (https://articraft3d.github.io/), along with startups and tools such as Moonlake (https://moonlakeai.com/) and BlenderMCP (https://github.com/ahujasid/blender-mcp), demonstrated what agents can achieve in 3D modelling. While generating visually compelling 3D content from text prompts is relatively easy, faithfully reconstructing an environment grounded in reality remains much more difficult. This is especially true at the room or building scale, where the agent must coordinate a large amount of scan data across different viewpoints and spaces while preserving geometric and spatial consistency. This is why most existing work in this domain focuses on generation or reconstruction conditioned on a single image.

Motivations

We believe there is a lot of value in turning real-world scans of rooms into interactive 3D scenes. There are two use cases that we find value in.

1. Recreating a Place You Know is cooler than purely generative

People often have a stronger connection to places they are already familiar with — the rooms where they have spent time, made memories, or shared experiences with others. Because of this, a reconstructed real-world space can feel more meaningful and engaging than an entirely generated environment, even when the reconstruction is not perfectly accurate. A familiar room could, for example, be turned into an

interactive space where users can meet friends, rearrange furniture, test different decorations, or visualise possible renovations. Standard mesh or Gaussian reconstructions can capture the geometry and appearance of a space, but they cannot offer this kind of interactive experience. What could be interesting is a representation that balances these properties: it should remain visually realistic and recognisable while also being structured, editable, and interactive, in a form closer to how 3D environments are built in traditional graphics pipelines.

2. Reality-grounded simulation built fast

Another motivation is the possibility of significantly reducing the cost of building reality-grounded simulation environments. There is still a gap between an interactive scene and a fully simulation-ready one, but we believe this gap is manageable through continued engineering work. If this process can eventually become fast and reliable enough, it could offer an interesting way to adapt robots to the specific environments in which they operate.

Imagine a robot entering a new room and having a realistic simulation of that space built from a scan within a short period of time. That simulation could then be used to train, test, or fine-tune the robot's policy for that particular environment. This may be especially relevant in real-world deployment settings, where a robot is likely to remain in the same home, office, warehouse, or hospital for a long time and repeatedly perform a limited set of tasks. In such cases, adapting, or even overfitting, the policy to that specific room could be practical.

We do not yet know how useful this approach will be in practice, but making reality-grounded simulation faster and more accessible gives us a concrete way to test this idea.

The Gap is real: there is no such tool for this

There are many possible motivations for this work, and several of these hypotheses still need to be tested in real experiments/user feedback. **But the current gap is clear: there is no open-source, end-to-end tool that lets users scan an indoor space and turn it into an interactive 3D environment.**

Anyone who wants to do this today usually has to build their own pipeline from scratch. These systems are often fragile, hardly work, or require a large amount of human effort to fix afterwards. LiteReality aims to bridge this gap by providing a full-stack toolkit for the entire process.

In this release, we provide two main components:

- **LiteReality App** (https://apps.apple.com/ph/app/litereality/id6774158260), a scanner for capturing indoor spaces.
- **LiteReality-Agent** (https://github.com/LiteReality/LiteReality-Agent), an agentic system that takes the captured scan and turns it into an interactive 3D scene.

The whole reconstruction pipeline, from scanning the room to interacting with it. It really is this simple.

How does it work

3.1 LiteReality App

For the scanning device, we chose LiDAR-equipped iPhones and iPads because they are widely available consumer devices that can capture both visual and geometric information about an indoor environment.

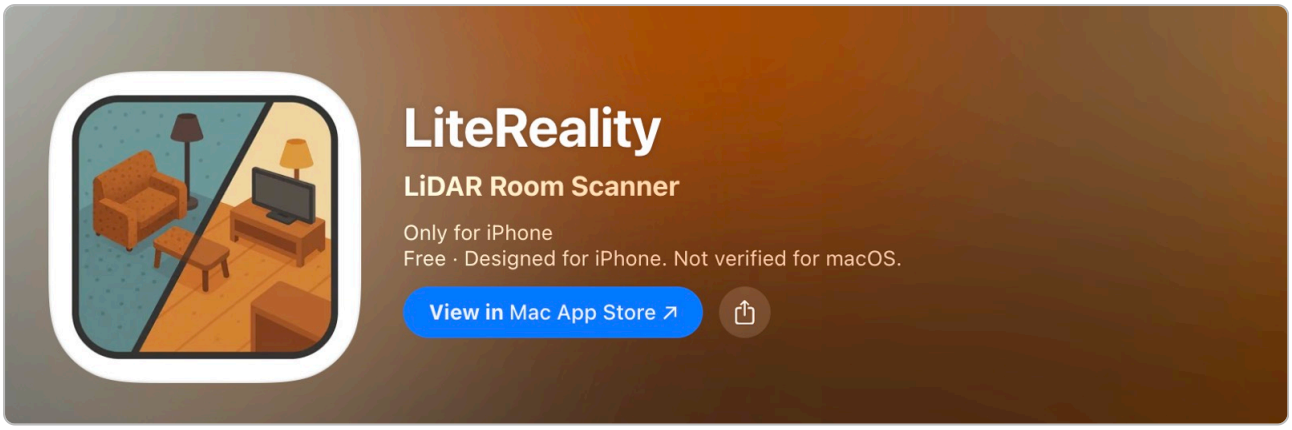
The app combines two Apple frameworks. ARKit provides the low-level, frame-by-frame sensor data, while RoomPlan provides a higher-level structural understanding of the room. During a scan, ARKit gives us camera frames together with per-frame depth measurements, camera calibration, and the estimated position and orientation of the device. RoomPlan uses the device's camera and LiDAR scanner to identify the main structure of the room, including walls, floors, doors, windows, and openings, as well as bounding boxes and semantic categories for recognised furniture and appliances.

A scan in the LiteReality App: the RoomPlan session builds up the layout as you walk the room, then the capture is named and exported.

The LiteReality App is designed primarily as a data-capture tool. It does not perform the full scene reconstruction on the device. Each scan contains:

Output	**From**	**What it is**
RGB frames	ARKit	the camera images captured through the sweep
Depth & confidence maps	LiDAR	per-frame depth, with a confidence map alongside it
Camera parameters	ARKit	per-frame intrinsics, poses, and timestamps
Point cloud	derived	a 3D point cloud reconstructed from the RGB-D observations
USDZ layout	RoomPlan	the room layout, plus the dimensions, categories, positions and orientations of detected objects and structural elements

Together, these outputs cover most of the information we would expect from a practical scanner. We make the LiteReality App freely available and allow users to access the full package of data, whether they want to use it with LiteReality-Agent or with their own reconstruction pipeline.



(https://apps.apple.com/ph/app/litereality/id6774158260)

LiteReality on the App Store (https://apps.apple.com/ph/app/litereality/id6774158260) — free, for LiDAR-equipped iPhones and iPads.

3.2 LiteReality-Agent (https://github.com/LiteReality/LiteReality-Agent)

LiteReality-Agent has two stages:

- a deterministic scene initialisation stage; and
- an agentic realism-authoring stage.

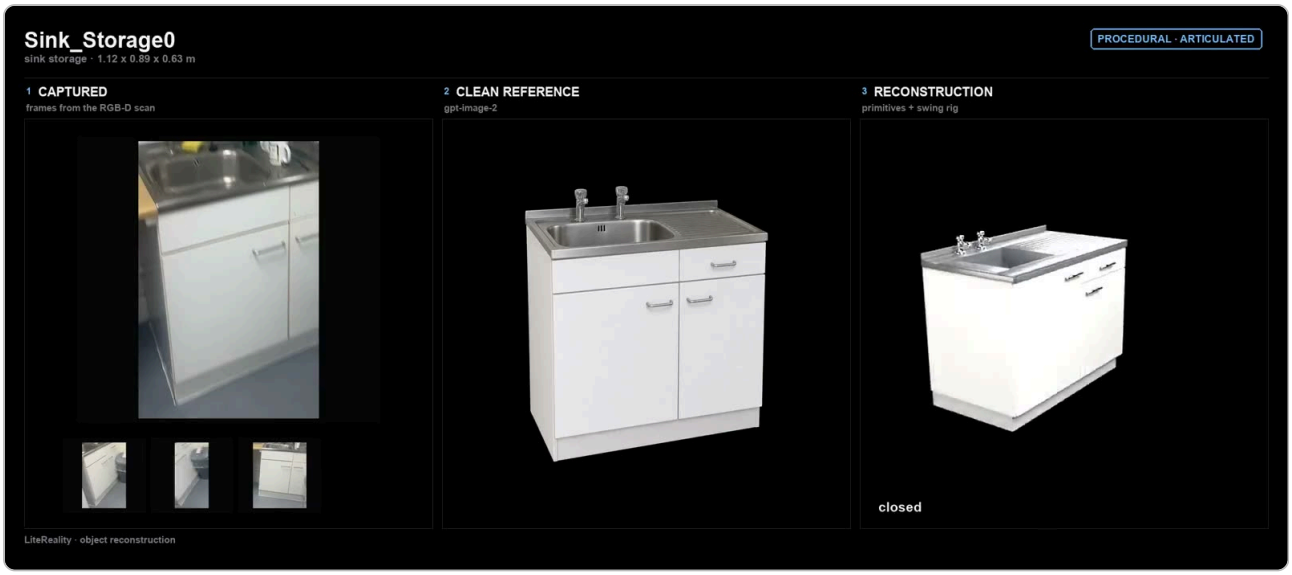
Scene Initialisation

The scanner provides a RoomPlan USDZ file containing the room layout and bounding boxes of the main objects. Following the original LiteReality (https://litereality.github.io/) pipeline, we reconstruct each object separately and then compose them into a complete room.

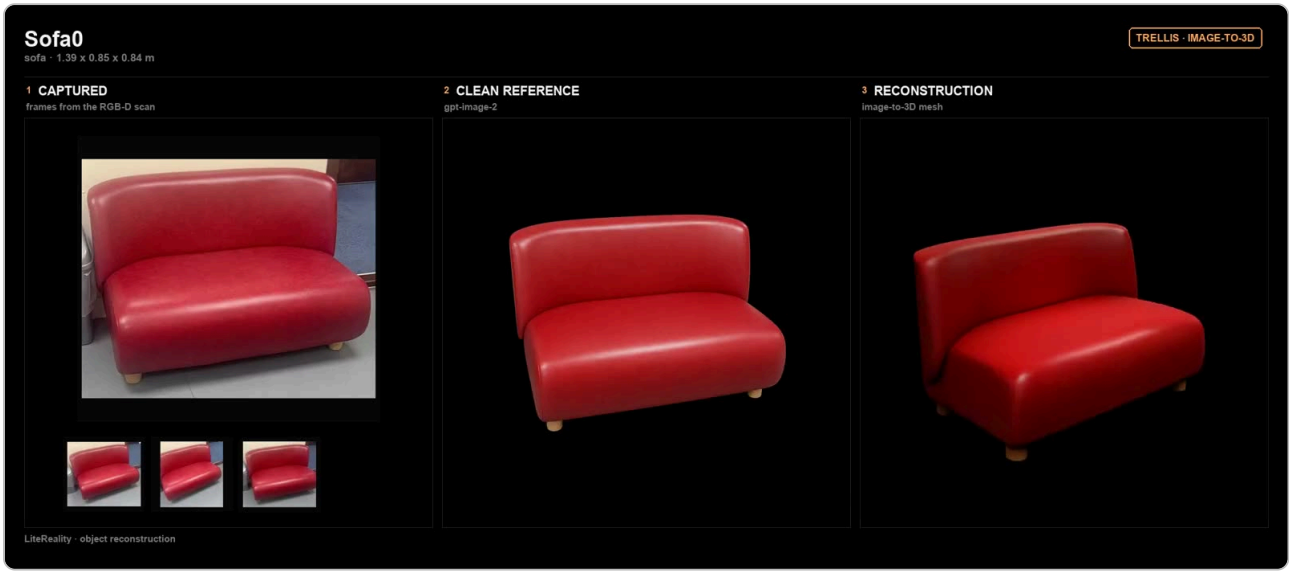
The object stage is where the two diverge. LiteReality *retrieved* objects: it searched a curated database of artist-made models and picked the closest visual match, which meant a room could only ever be furnished with what the database already held. Here nothing is retrieved. Every object is reconstructed from the scan's own images of it.

For each object, the system collects reference images from the scan and routes it to one of two branches:

- procedural generation for objects that require structured geometry, articulation, or interaction; or
- TRELLIS-based image-to-3D generation for visually complex or mostly static objects.



A procedural reconstruction example — the unit turns closed, then again with its doors open.



A 3D-generative reconstruction example — the same three panels, the other branch.

Procedural assets are generated using an adapted version of Articraft (https://articraft3d.github.io/) with Blender Python as the scene language. Adapting it also means we inherit a good deal of its quality control over generated assets, and some of its progress towards simulation-readiness. Once all assets are created, they are placed into the room using the positions, orientations, and dimensions estimated from the scan. The resulting scene then becomes the starting point for the agentic authoring stage.

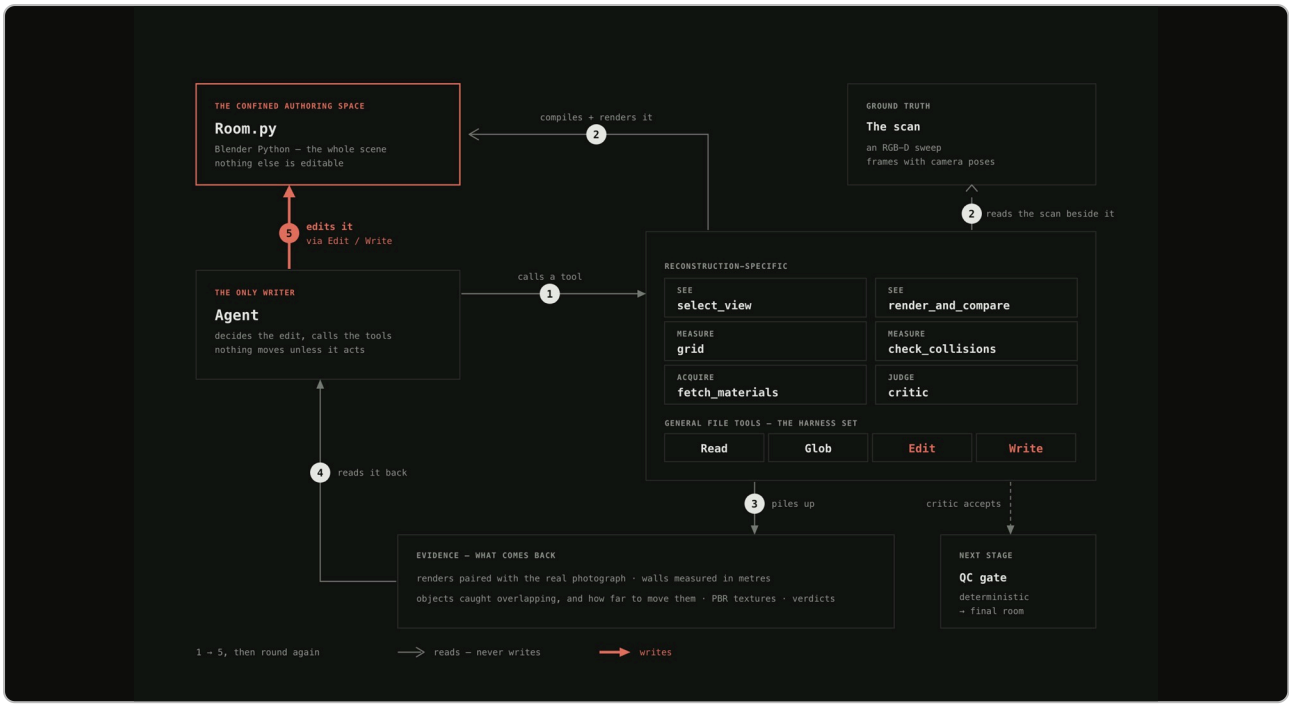
Initialisation on two rooms. Following the LiteReality pipeline, we build the room layout and position the reconstructed objects inside it.

4. The Agentic Authoring Loop

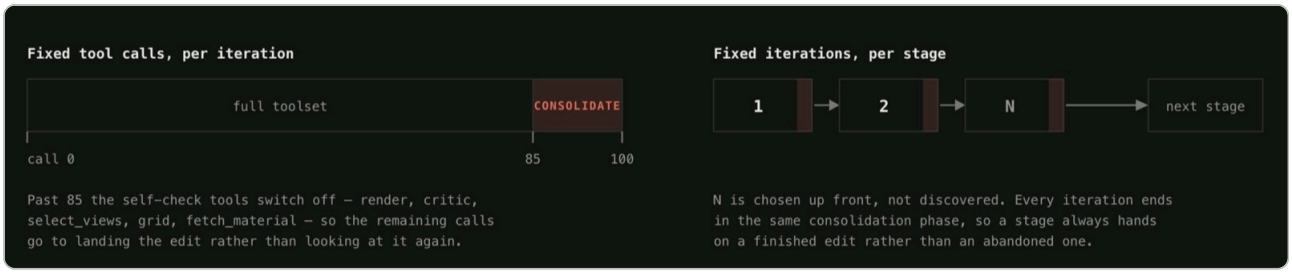
The second stage takes the initialised room and iteratively improves its realism through an agentic authoring loop.

Our authoring loop follows several key design principles:

1. **A confined authoring space.** The agent operates primarily on a single `Room.py` file — a Blender Python script that defines how the room is constructed.
2. **Task-specific tools.** We provide a set of tools designed specifically for this task. The agent can call them freely to gather and inspect evidence, make decisions, and edit the scene accordingly.
3. **A structured iteration process.** Because it is difficult to define a clear stopping condition for realism, we specify a fixed number of tool calls and refinement iterations. As the agent approaches the end of each iteration, it is explicitly instructed to consolidate its edits and complete the current refinement stage.
4. **A modular workflow.** The pipeline is divided into independent stages. It begins with refining wall PBR materials and fixtures, followed by improving the objects constructed during initialisation. Finally, the room passes through a series of quality-control checks to ensure the output meets a consistent standard.



One turn of the loop, numbered 1 to 5. Everything the agent learns arrives as evidence — renders paired with the real photograph, walls measured in metres, objects caught overlapping and how far to move them — and every change it makes goes back through one file.



What principle 3 looks like in practice. Realism has no natural stopping condition, so the budget supplies one: past call 85 the tools that let the agent look at its work switch off, and the calls that remain go to landing the edit rather than reopening it. N is chosen up front rather than discovered, so a stage always hands on a finished edit instead of an abandoned one.

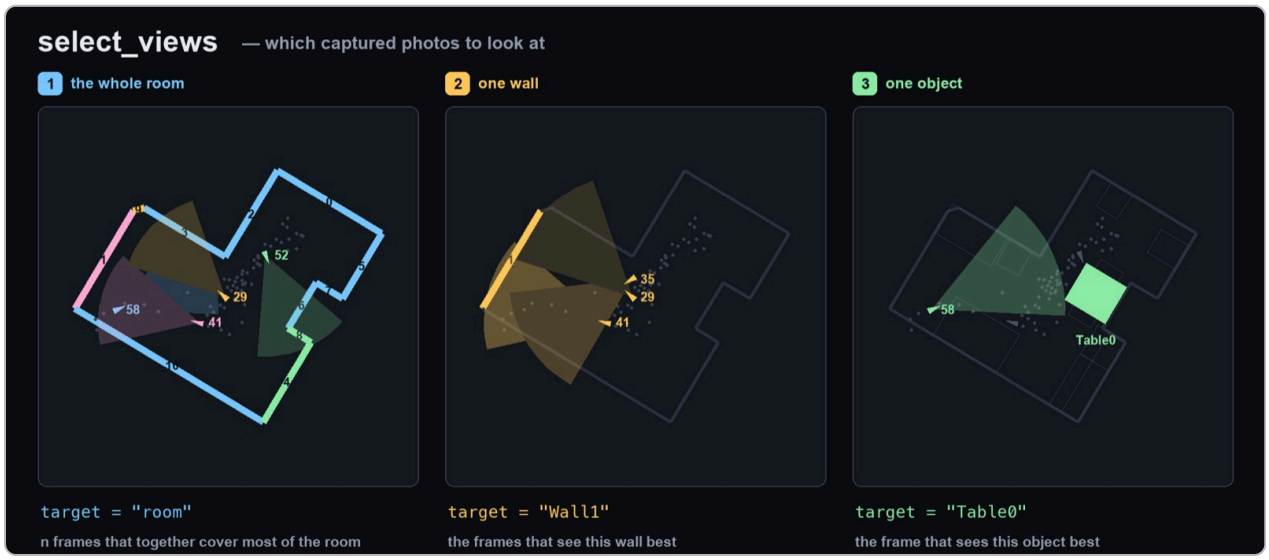
The tools

A closed set, designed for this task. Click one to see what it does.

select_view

SEE

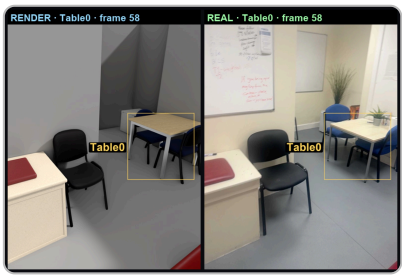
Which of the sixty-odd capture frames to actually look at. Ask for the room and it returns the fewest frames that between them see every wall; ask for one wall or one object and it ranks frames by how much of it they see. A model asked to pick a frame number just guesses.



render_and_compare

SEE

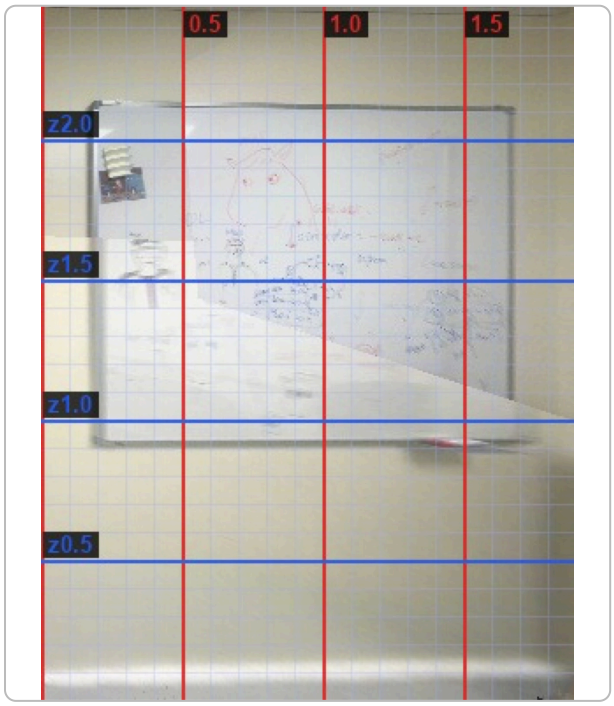
The render beside the real photograph, from the same camera pose, with the annotation narrowing as the question narrows — whole room, one wall, one object. The model is never asked to align two pictures in its head.



grid

MEASURE

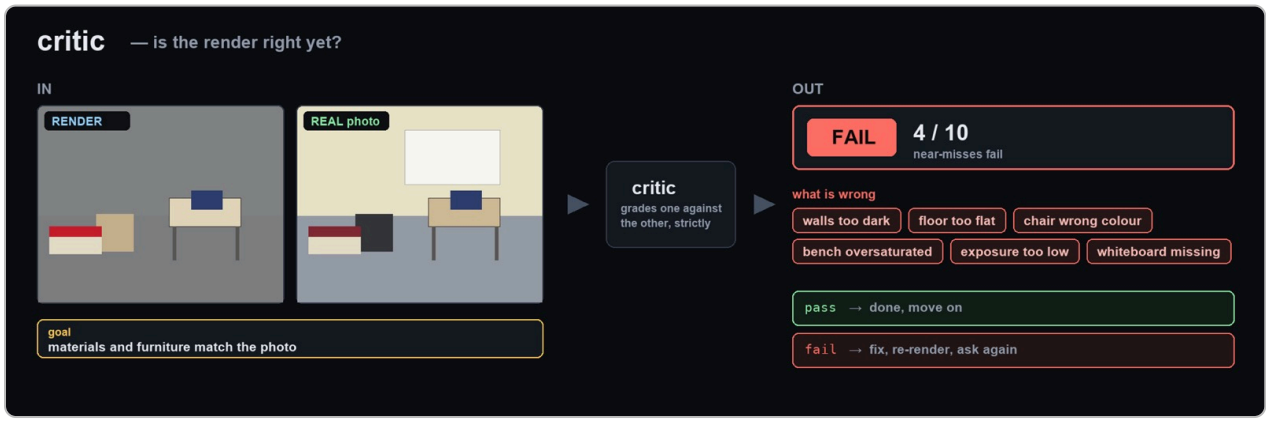
A metric ruler drawn on a rectified wall: metres along the wall in red, metres above the floor in blue. Reading beats estimating, and the numbers you read are the numbers that go into the scene. The agent invented this one — it hand-rolled the same thing in Bash thirteen times in a single run before it existed as a tool.



critic

JUDGE

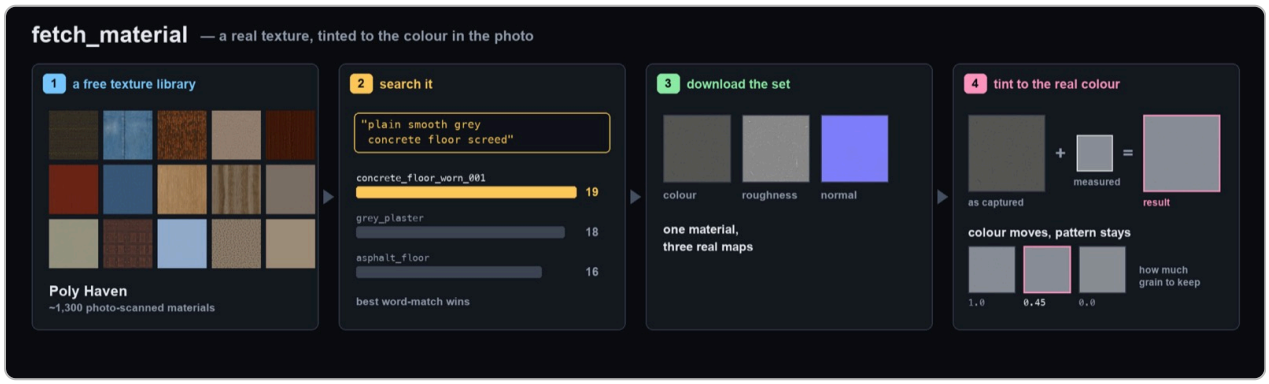
A verdict the loop can branch on, rather than prose: pass or fail, a score, and a list of what is wrong. The tool owns the grading prompt — not the agent — so strictness is a property of the system instead of whatever the model felt like asking for that turn. Near-misses fail.



fetch_materials

ACQUIRE

Procedural node materials look flat next to real captured ones. So: search a free library of photo-scanned textures, download the colour, roughness and normal maps, then tint the colour to what the photograph actually shows. The colour moves and the pattern stays.

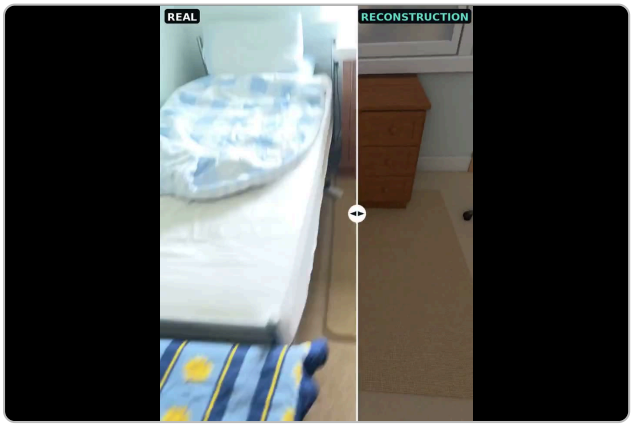


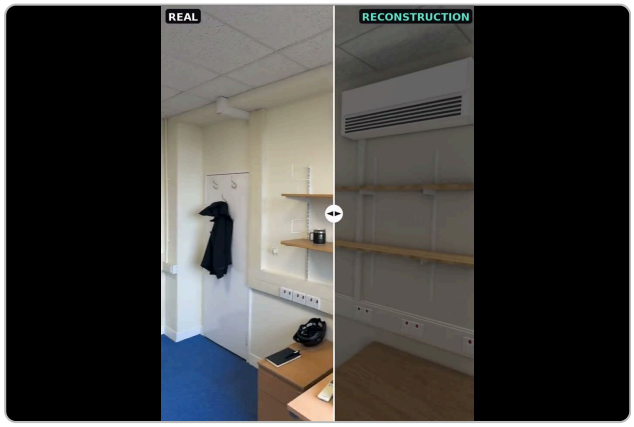
Quality Control as a Gate

After authoring, the scene must pass a set of deterministic quality-control checks before it can be exported. These checks include collision detection and other tests for geometry, placement, materials, articulation, and scene validity.

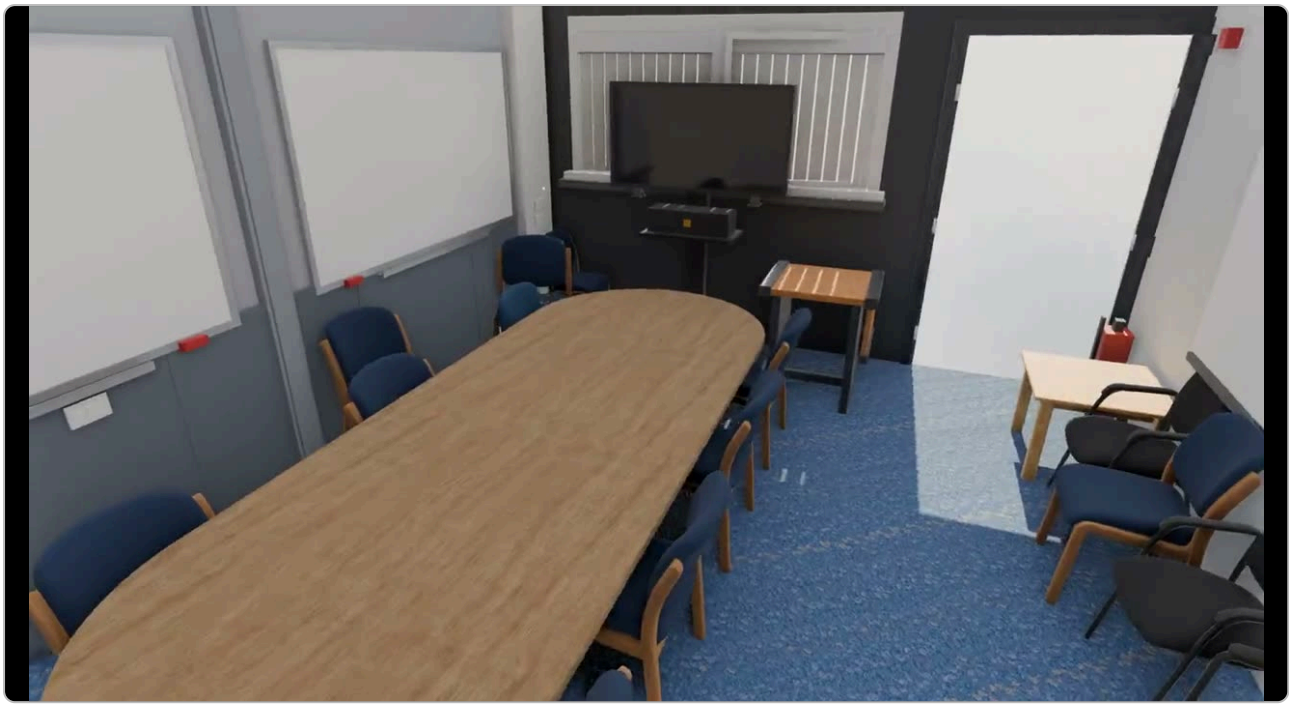
Most of that layer is arithmetic rather than a model judging its own work. The geometry and placement checks are deterministic, resolving what has a safe fix and reporting what does not; only the final pass over the room uses a model, working through a fixed checklist. A scene leaves the pipeline only once it clears the gate.

Results





Real capture against reconstruction, wiped side by side.



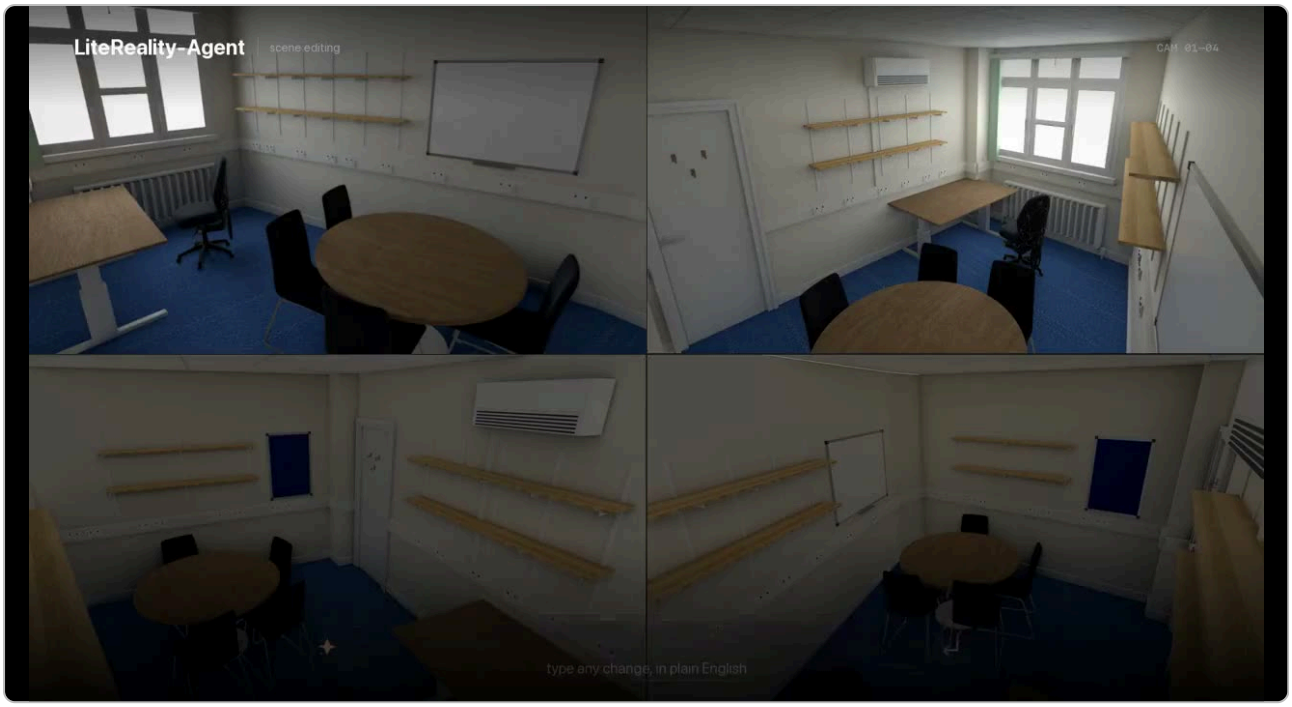
A flythrough of the finished meeting room.

These are two rooms out of more. See the rest in the gallery (https://litereality.github.io/Litereality-agent-site/vr/index.html).

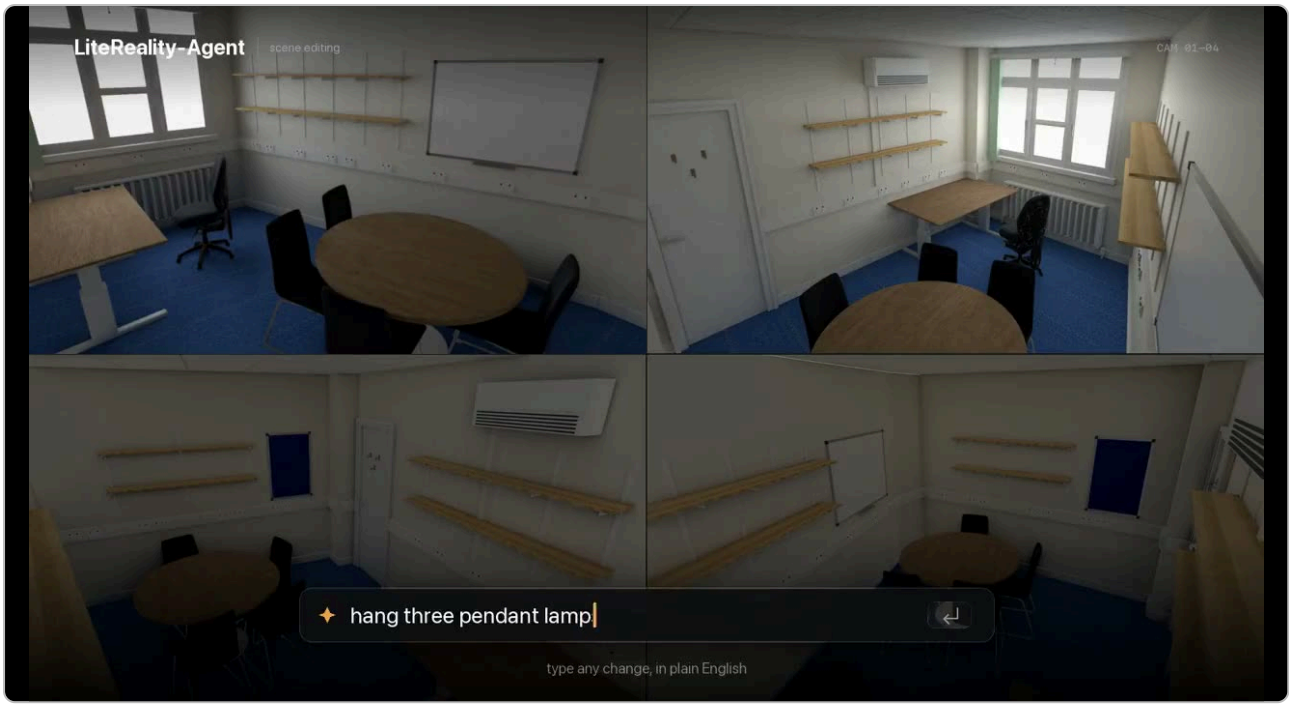
Editing the reconstructed scene

Because the room is a `Room.py` file, editing it is a prompt away.

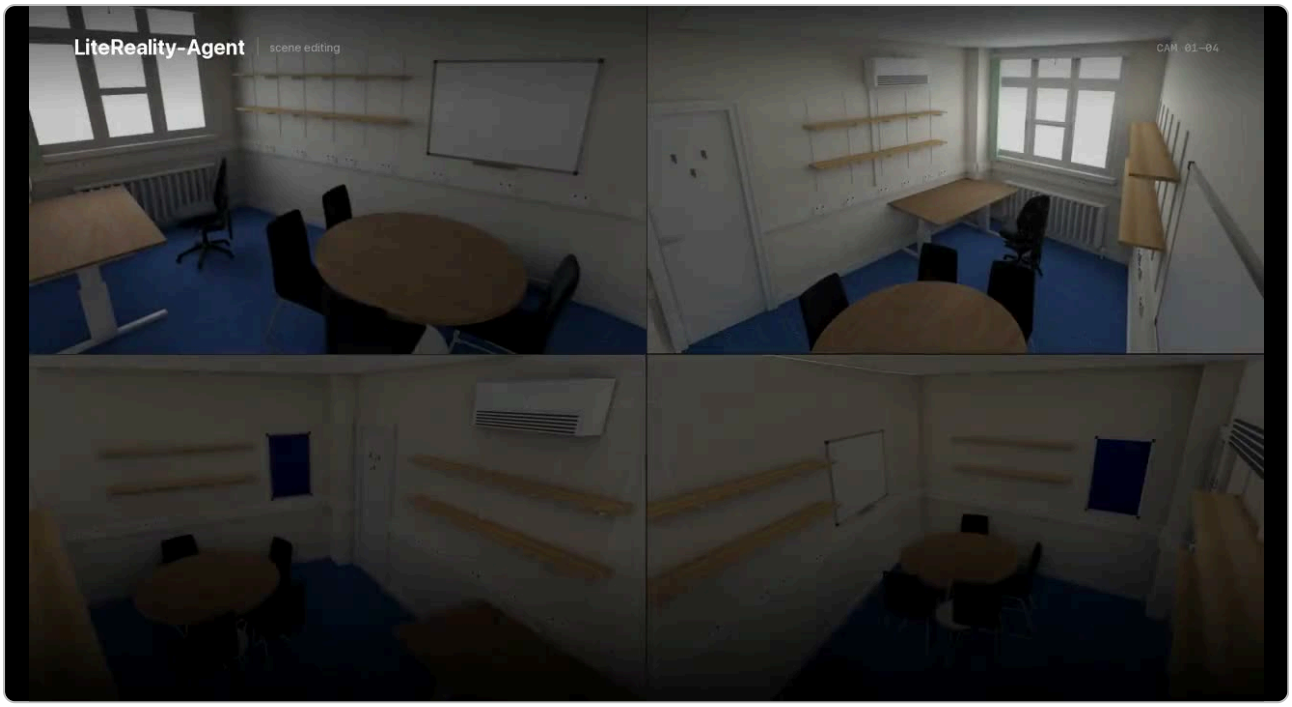
It is a programmable room, and a lot of what used to be hard is now one sentence. Rearrange the furniture. Redecorate, pulling assets straight from Sketchfab. Hand it a photograph and ask for that object, placed in the room. And if part of the reconstruction is not right, say so — the agent goes back, focuses on that piece, and does it again.



*A **material** edit. Nothing moves and nothing is added — the surface is repainted, and the four views update together because they are four renders of one file.*



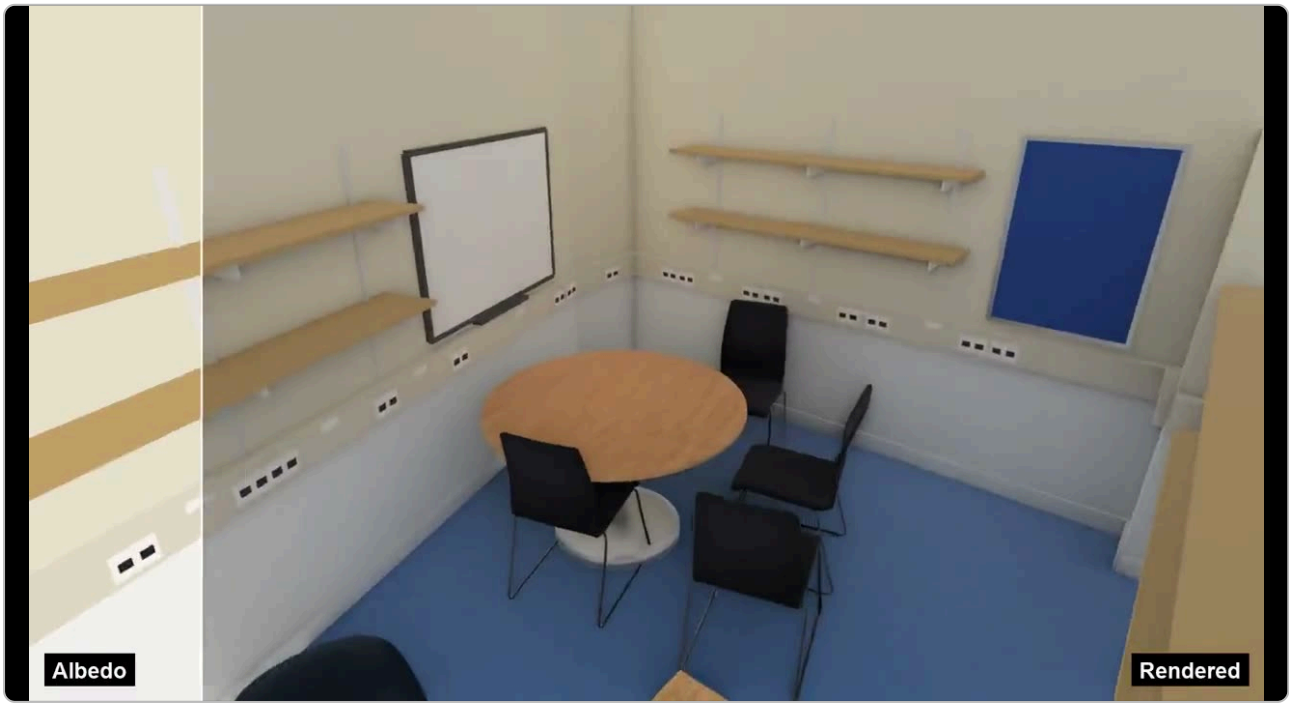
*A **lighting** edit, and the case that needs new objects. The lamps are fetched as real geometry, and the model decides where they hang — over the table, at a height that clears it.*



*A **layout** edit, from an instruction that never names an object. Working out that a standup means clearing floor space, and which pieces are in the way, is the part the scene being code does not solve — but collision checking is what confirms the furniture ended up somewhere legal.*

Intrinsics

A fully authored scene renders perfect intrinsics for free: pixel-exact segmentation, depth, normals and albedo, from the same cameras.



Wiping between passes on one view — albedo, depth, normals, instance and semantic segmentation, materials. They are exact rather than estimated, because the scene was authored rather than inferred.

Limitations

1. **The scenes are not yet simulation-ready.** Although the reconstructed environments are interactive, this is not the same as being ready for direct use in a simulator. Making an entire room simulation-ready still requires substantial additional effort. However, when combined with the progress made in Articraft (https://articraft3d.github.io/), we believe this goal is within reach. Once the individual objects are simulation-ready, the remaining challenge is primarily to ensure that the overall scene layout is physically valid and suitable for simulation. We expect this to be a more manageable problem.
2. **The system has not yet been tested at large scale.** Our current experiments focus on single-room environments of approximately 50 m² or less. Extending the system to larger spaces, more challenging layouts, and multi-room environments will require further engineering and evaluation. We do not consider this to be the most fundamental challenge, but the system has not yet been sufficiently stress-tested under these conditions. This is something we will keep working on.

Looking Ahead

The pace of progress in agentic systems is extraordinary, and many capabilities that seemed difficult to imagine only a short time ago are rapidly becoming possible. We are excited to see how this line of work develops, and to continue maintaining LiteReality-Agent as a useful tool for the community working on interactive 3D scene reconstruction.

Anyone interested in contributing, experimenting with the system, or exploring related ideas is very welcome to get in touch with us.

References

1. Huang, Z., Wu, X., Zhong, F., Zhao, H., Nießner, M., Lasenby, J. *LiteReality: Graphics-Ready 3D Scene Reconstruction from RGB-D Scans.* 2025. arXiv:2507.02861 · project page
2. *Articraft: An Agentic System for Scalable Articulated 3D Asset Generation.* project page · code
3. Xiang, J., Lv, Z., Xu, S., Deng, Y., Wang, R., Zhang, B., Chen, D., Tong, X., Yang, J. *Structured 3D Latents for Scalable and Versatile 3D Generation (TRELLIS).* 2024. arXiv:2412.01506 · code
4. Apple. *RoomPlan.* Developer documentation. developer.apple.com/documentation/roomplan
5. Apple. *ARKit.* Developer documentation. developer.apple.com/documentation/arkit
6. *BlenderMCP – Blender Model Context Protocol integration.* github.com/ahujasid/blender-mcp
7. *Moonlake.* moonlakeai.com
8. *LiteReality-Agent – code and documentation.* github.com/LiteReality/LiteReality-Agent

Citation

```
@article{huang2026litereality-agent,
  title   = {{LiteReality-Agent}: An Agentic System for
             Interactable 3D Indoor Scene Reconstruction},
  author  = {Huang, Zhening and Li, Yueyan and Chiu, Johnathan and
             Lyu, Xiaoyang and Zhou, Matt and Yao, Yuxin and
             Lasenby, Joan and Wu, Shangzhe},
  year    = {2026}
}
```