LITEREALITY-AGENT: AN AGENTIC SYSTEM FOR INTERACTABLE 3D INDOOR SCENE RECONSTRUCTION

Anonymous authors
Paper under double-blind review

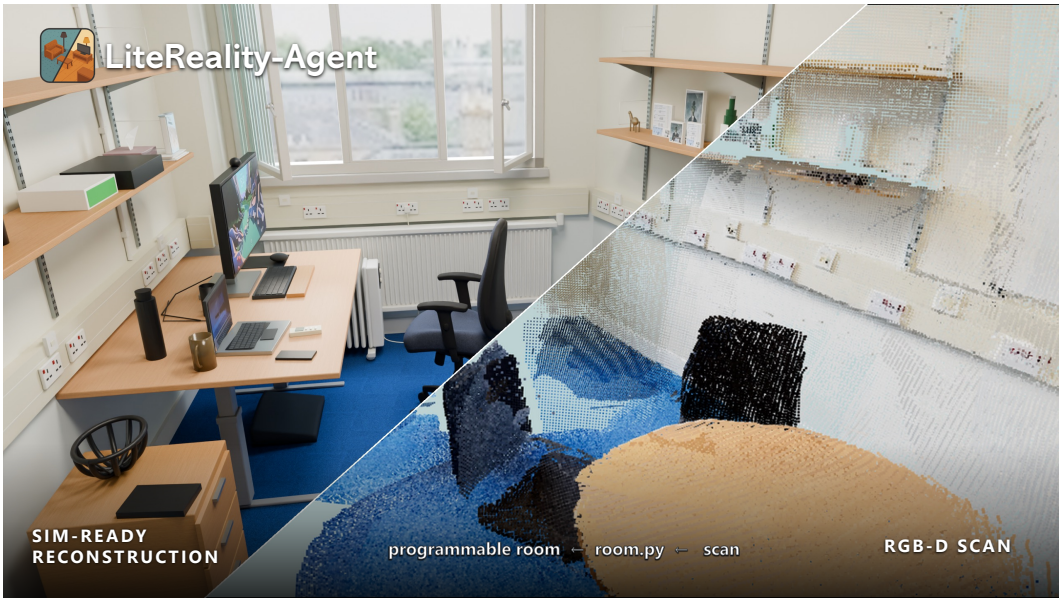


Figure 1: Reconstructing a real room with LiteReality-Agent. LiteReality-Agent repurposes the reconstruction as a coding task, where a coding agent writes a `Room.py` file that can be compiled into a realistic, interactive, and simulation-ready 3D room.

ABSTRACT

We present LiteReality-Agent, an agentic system for reconstructing real indoor environments as realistic, articulated, and simulation-ready 3D scenes from RGB-D scans. At its core, LiteReality-Agent formulates 3D reconstruction as a coding problem, where a coding agent gathers evidence through designed tools and iteratively edits a Python script, `Room.py`, which can be executed to produce a 3D digital twin of the room. With this formulation, we build a robust observe-edit-verify harness that supports evidence gathering, measurement, verification, layout optimisation, simulation-readiness, and quality control throughout the reconstruction process. LiteReality-Agent produces high-quality reconstructions suitable for simulation and downstream embodied AI tasks. Furthermore, as agent capabilities continue to improve rapidly, the system introduced by LiteReality-Agent remains a strong orchestration framework for agents: it equips them with specialised tools, structured workflows, and robust verification mechanisms that substantially improve reconstruction quality and reliability. We have demonstrated that LiteReality-Agent produces reconstructions that are more geometrically accurate, visually realistic, and simulation-compatible than those generated by the latest frontier models, such as Astra and Fable. We therefore view LiteReality-Agent as a practical and important building block for robust real-to-sim systems. Both the source code and the data-capture application are publicly available.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Building a functional, simulation-ready digital twin has long been a holy grail of inverse graphics. Achieving this goal could unlock a range of applications, including robotic simulation, manufacturing digital twins, and virtual reality. Unlike point-based 3D reconstruction methods (Campos et al., 2021; Wang et al., 2025) , or conventional mesh reconstruction methods (Murez et al., 2020; Sun et al., 2021), our goal is to recover a structured environment that can be directly used for interaction and physical simulation (Huang et al., 2025b). This requires jointly understanding the structure of the scene, reconstructing objects and their articulations (Zhou et al., 2026; Li et al., 2026), estimating material and physical properties, and resolving spatial relationships. Integrating these capabilities into a robust end-to-end system remains a longstanding challenge.

Previous work has taken this direction through several approaches. CAD-based scene reconstruction retrieves CAD models and positions them within the scene to form a compact representation (Avetisyan et al., 2019; Kuo et al., 2020; Gao et al., 2024). Compositional 3D reconstruction from single or multiple images instead uses 3D generative models to reconstruct individual objects and place them in the scene (Yao et al., 2025), while inverse-rendering pipelines recover object geometry, orientation, and materials through differentiable rendering (Yan et al., 2023; Yeh et al., 2022). Despite this progress, a gap remains in building a robust and scalable system that can turn large-scale real-world scans into high-quality environments suitable for interaction and physical simulation. LiteReality (Huang et al., 2025b) takes a step toward this goal with a multi-stage pipeline that converts indoor scans into graphics-ready scenes by integrating scene understanding and parsing, asset retrieval, appearance recovery, and assembly into a conventional graphics representation. However, its largely deterministic pipeline limits its ability to handle cases beyond predefined scenarios.

Meanwhile, a growing body of work has explored the use of large language models for 3D scene generation. Systems such as Holodeck (Yang et al., 2024b), SceneCraft (Hu et al., 2024), and SceneSmith (Pfaff et al., 2026) employ LLMs to generate and validate scene layouts and object placements, producing simulation-ready indoor environments in graphics engines. At the object level, Articraft (Zhou et al., 2026) formulates articulated asset generation as a programming task supported by a domain-specific SDK and automated verification. More recently, proprietary LLMs such as GPT-6 Astra and Claude Fable 5 further suggest that coding agents have developed strong capabilities for understanding and modelling 3D environments. Together, these advances make the previously daunting problem of functional digital-twin reconstruction increasingly tractable. What remains missing is a system that can orchestrate such agents while grounding their outputs in real-world observations and ensuring that the resulting scenes are consistently high quality and suitable for their intended applications.

To this end, we introduce LiteReality-Agent, a full-stack agentic system for transforming consumer RGB-D room scans into realistic, articulated, and simulation-ready environments. At its core, we formulate reconstruction as a coding task: the entire room is represented as an executable `Room.py` program that serves as a persistent workspace. A coding agent can access the observations, modify the program, and compile it into a 3D scene. Because all modifications operate through the same executable representation, the scene remains readable, reproducible, and directly editable throughout the reconstruction process.

LiteReality-Agent consists of two stages: *Scene Initialisation* and *Agentic Authoring*. *Scene Initialisation* first reconstructs each object from its corresponding image evidence as either a static or articulated asset. The noisy scan-derived layout and object placements are then processed by the *Layout Agent*, which resolves collisions, validates object placements and support relationships, and produces a clean, physically plausible scene layout. The resulting scene is instantiated as an initial `Room.py` file. *Agentic Authoring* subsequently refines the scene through an **observe-edit-verify** harness. The agent is equipped with task-specific tools for image selection, rendering and comparison, geometric and metric measurements, and scene editing. Explicit quality-control checks, including collision and support verification, help ensure that the scene remains complete, structured, and physically plausible throughout the authoring process. Physical properties and collision geometry are also estimated by the object reconstruction agents, following a formulation similar to Articraft (Zhou et al., 2026). The resulting environment is simulation-ready and fully programmable through `Room.py`. Representing the entire scene as an executable Python program also enables down-

stream editing via natural-language instructions carried out by a coding agent. We demonstrate this capability through scene decoration, object replacement, and layout adjustment.

We rigorously benchmark LiteReality-Agent on diverse real-world indoor scenes against the deterministic LiteReality pipeline and general-purpose coding agents: GPT-6 Astra running in Codex and Fable 5.1 running in Claude Code. Our evaluation combines image-based measures of visual fidelity, geometric measures of reconstruction accuracy and physical plausibility, and human and agent-based perceptual assessments. LiteReality-Agent achieves the highest mean scores across all reported visual metrics and perceptual criteria. These results demonstrate the effectiveness of LiteReality-Agent as an orchestration framework for agentic indoor real-to-simulation reconstruction. Both the source code and the data-capture application are publicly available.

2 Related Work

Indoor Scene Reconstruction. Work in the domain of object-centric 3D scene reconstruction recovers scene structure by retrieving and aligning CAD models to observations. Scan2CAD (Avetisyan et al., 2019) aligns CAD assets in RGB-D scans, while Mask2CAD (Kuo et al., 2020) and DiffCAD (Gao et al., 2024) extend retrieval and alignment to image-based settings. In parallel, NeRF (Mildenhall et al., 2020) and 3D Gaussian Splatting (Kerbl et al., 2023) achieve high-fidelity novel-view synthesis from posed images, and decompositional reconstruction (Ni et al., 2025) introduces object-level priors into neural scene representations. These approaches emphasise different aspects of geometry, appearance, and object structure. Recent systems integrate these elements into environments that support editing, simulation, and reuse. Phone2Proc (Deitke et al., 2023) turns mobile captures into procedural environments for embodied-agent training, and automated digital-cousin construction (Dai et al., 2025) converts observed spaces into asset-based environments for robot learning. HoloScene (Xia et al., 2025) reconstructs interactive, simulation-ready worlds from a single video, while PhyScene (Yang et al., 2024a) targets physically interactable indoor scene synthesis. Most directly, LiteReality (Huang et al., 2025b) reconstructs graphics-ready scenes from RGB-D scans with individually represented objects and conventional graphics assets. LiteReality-Agent extends this reconstruction setting with a persistent scene program that an agent revises against calibrated observations, allowing reconstruction to continue through targeted edits and verification after initialisation.

Generative 3D Object Reconstruction. Single-image 3D generation has progressed from diffusion-based novel-view prediction and multi-view reconstruction (Liu et al., 2023b; 2024; 2023a; Long et al., 2024) to feed-forward large reconstruction models such as LRM (Hong et al., 2024) and InstantMesh (Xu et al., 2024). More recent systems improve structured latent representations, geometry, texture, and production-ready PBR materials (Xiang et al., 2025; Hunyuan3D et al., 2025). SAM 3D (SAM 3D Team et al., 2025) reconstructs objects in cluttered natural images. These models complement procedural modelling for visually complex objects. Our system integrates both routes within a captured room, where asset dimensions, placement, support, and appearance must remain consistent with the observations.

Agentic 3D Authoring and Inverse Graphics. Language models support layout planning and structured scene synthesis. LayoutGPT (Feng et al., 2023) predicts compositional layouts, OpenUniverse (Aguina-Kang et al., 2024) generates scene programs over uncurated assets, and AnyHome (Fu et al., 2024) produces open-vocabulary structured homes. I-Design (Çelen et al., 2025) and Holodeck (Yang et al., 2024b) use language-guided planning, while FirePlace (Huang et al., 2025a) and LayoutVLM (Sun et al., 2025) refine object placement through geometric or vision-language feedback. For programmatic authoring, SceneCraft (Hu et al., 2024) synthesises Blender scenes as code and LL3M (Lu et al., 2025) uses language models as interactive 3D modelers. Scenethesis (Ling et al., 2026), SceneSmith (Pfaff et al., 2026), SAGE (Xia et al., 2026), and SceneWeaver (Yang et al., 2025) develop agentic pipelines for simulation-ready and self-reflective scene generation; Articraft (Zhou et al., 2026) extends agentic authoring to articulated 3D assets. Iterative authoring also draws on tool use and self-improvement in language agents (Schick et al., 2023; Madaan et al., 2023; Shinn et al., 2023; Weng, 2026). Our system grounds these authoring capabilities in a captured room, checking edits against visual evidence and geometric constraints.

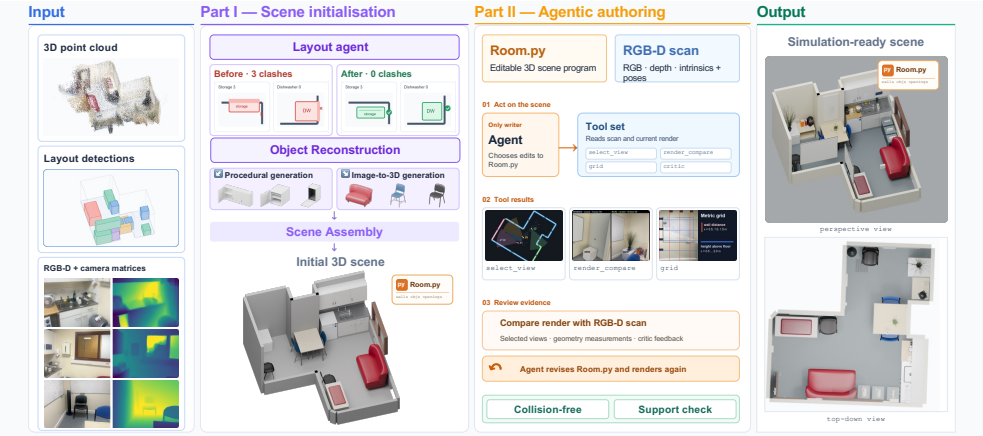


Figure 2: **End-to-end LiteReality-Agent pipeline.** RGB–D scans and detections guide layout repair and object reconstruction to produce an executable initial room. The authoring agent refines `Room.py` through an observe–edit–verify loop, using capture views, measurements, materials, and rendered comparisons. Control gates bound authoring and check geometry and support. The editable scene and physical metadata support MuJoCo export.

3 LiteReality-Agent

LiteReality-Agent reconstructs a room in two stages: *scene initialisation* establishes the layout and major objects, and *agentic authoring* refines an executable scene program through an observe–edit–verify loop (Fig. 2). Implementation details and supporting examples are provided in Appendix B.

3.1 Input: RGB–D Scans and Detection Results

The input consists of RGB–D scans and detection results from the *LiteReality Scan* application, including the room layout and object bounding boxes.

3.2 Scene Initialisation

We follow LiteReality (Huang et al., 2025b) in parsing the room shell and reconstructing objects separately before assembly, replacing asset retrieval with generation from the scan's own images.

Layout Agent. The layout is represented as a scene graph of structural elements and objects, with relations for support, wall adjacency, functional grouping, and potential clashes. These relations distinguish physical support from association: a chair belongs to a table but rests on the floor. They also distinguish incompatible overlaps from expected box overlap, such as a chair tucked beneath a table.

The Layout Agent corrects noisy detections before object reconstruction. Geometric checks identify objects outside the room, embedded in walls, unsupported, or overlapping incompatibly. Deterministic repairs make small placement and dimension adjustments; image-based reasoning handles ambiguous cases, such as an incorrectly sized or merged detection. Proposals are accepted only when they improve geometric consistency and remain within cumulative limits relative to the scan. The corrected boxes guide both asset generation and final placement, preventing later assembly from restoring the original errors.

Object reconstruction. For each object, selected capture views are processed by an image-generation model to produce a clean reference image. Structured or articulated objects are routed to procedural generation, while visually complex, predominantly static objects use TRELLIS.2 (Xiang et al., 2025). Generated meshes are checked for malformed geometry, detached fragments, and fused background surfaces. A bounded retry loop for generated chairs regenerates the reference and mesh when these checks fail.

Algorithm 1: Stage 2 agentic authoring loop: the agent iteratively edits `Room.py` based on observations using the tools in Table 1.

$env, tools \leftarrow$ `Room.py`, $defined_tools$
$observations \leftarrow$ SCAN( )
$history, turn, qc_passed \leftarrow [], 0,$ FALSE
while $turn < T$ **and not** qc_passed **do**
  $actions \leftarrow$ LLM($observations, env, tools, history$)
  $result \leftarrow$ EXECUTE($actions, env, observations$)
  append $(actions, result)$ to $history$
  if $result$ edited the environment **then**
    $qc_passed, feedback \leftarrow$ QC($env, observations$)
    append $feedback$ to $history$
  end if
  $turn \leftarrow turn + 1$
end while

Table 1: Authoring tools and shared evidence utilities. View and render tools support room, wall, and object scopes.

Tool / utility	Purpose
`select_views`	Select informative capture frames for a target
`render_and_compare`	Compare calibrated scene renders with photographs
`measurement`	Measure dimensions and positions in metres
`stitch_wall`	Assemble a rectified wall image and coverage mask
`fetch_material`	Fetch PBR texture maps and adjust their colour
`bash, glob read, edit`	Find files, read code and image evidence, and create or edit programs

The procedural branch follows Articraft (Zhou et al., 2026): a coding agent writes an `object.py` program defining geometry, separate moving parts, and joints. We compile physical properties alongside the visual asset, retaining authored values and otherwise estimating mass from category and size, inertia from geometry, and friction from material priors. Convex collision geometry and per-object simulation checks help identify unusable assets before room assembly.

The scene program. The shell and reconstructed objects are assembled at their corrected scales and poses. `Room.py` defines the room, object instances, materials, and added geometry, while procedural assets retain their `object.py` programs. This is the agent's persistent workspace: edits are written to code and reproduced by rebuilding the scene.

3.3 Agentic Authoring

The second stage observes the capture, edits the scene program, and verifies the result through rendered comparisons and geometric checks. Algorithm 1 summarises the loop, and Table 1 lists its tools.

3.3.1 Evidence Tools

Image selection at three levels. The agent queries the whole room, one wall, or one object. Room-level selection greedily chooses complementary frames that cover the walls. Wall-level selection ranks views by the wall's projected on-screen area. Object-level selection uses per-object visibility from the render manifest to find clear views of the target. A minimum frame separation reduces near-duplicate observations. These scopes let the agent move from inspecting the overall layout to checking a local detail.

Rendering and comparison. The same three levels determine how comparisons are labelled (Fig. A7a–c). Room views label visible objects by name; wall views outline only the selected wall; object views draw its bounding box. Each pair uses the same calibrated camera for the render and photograph. Thus, a room-level mismatch can be traced to a named object or surface and inspected in isolation. Changed scene and object programs are rebuilt before rendering so that comparisons show the current edit.

Wall stitching and measurement. Capture frames are projected onto a metric wall plane and combined into one head-on reference, selecting visible observations while marking unobserved regions. A grid expresses distance along the wall and height above the floor. For example, the whiteboard in Fig. A8d can be located and sized from its edges relative to the red and blue ticks, then placed using those coordinates in `Room.py`. Surface queries also pair this rectified reference with an orthographic render for checking materials and fixtures. Raw views remain necessary for objects protruding from the wall. PBR retrieval supplies texture maps whose colour can be adjusted to match the observations.

Authoring harness. The harness guides four ordered modules: *first*, matching wall materials; *second*, adding fixtures and lighting; *third*, placing small objects; and *finally*, refining existing procedural assets through multi-view renders of their geometry, appearance, and articulation. Each module builds on the previous result. To keep the scene organised for later editing, added geometry is grouped into named objects with explicit support (`rests_on`) or attachment (`attached_to`) relationships. Subsequent agents can then identify and edit objects while accounting for these relationships.

3.3.2 Control and Quality Gates

Budgeted control. Visual quality has no reliable automatic stopping criterion, so we control authoring through a bounded loop with explicit gates. Sessions have turn and tool-call limits. During the final 10% of calls, exploration and self-checks are disabled so the agent completes and saves its edits. The program is then rebuilt and validated. The budget controls when authoring stops; geometric and support gates determine whether the result passes validation.

Geometric and support gates. Layout repair uses coarse boxes, so the authored room requires another check on its actual geometry. This is particularly important for chair–table pairs with valid box overlap and for newly added objects. Mesh intersections determine whether a collision exists; convex approximations estimate corrective translations. Bounded corrections move unanchored objects, are saved to `Room.py`, and are checked again after rebuilding.

Support is checked separately: an object may avoid collisions yet float above its intended surface. We compare object undersides with declared supports, report missing supports or invalid contact heights, and check that articulated assets retain separate moving parts. Unresolved findings return to authoring; checklist-based visual review can provide additional appearance feedback. Table 3 reports support and intersection measurements on the resulting scenes.

3.4 Simulation-Ready Export

We export the room to MuJoCo (Todorov et al., 2012) as individual objects with mass, inertia, collision geometry, and joints. Walls and attached fixtures stay fixed; movable objects have free joints, and articulated parts use hinge or slide joints. Separate collision shapes preserve space beneath tables and inside furniture. The exporter accounts for asset placement and scale, estimating physical properties when needed.

4 Results

We evaluate visual fidelity, geometric consistency, and perceived similarity on ten indoor scenes, comparing LiteReality-Agent with LiteReality and two general-purpose agent baselines.

4.1 Experimental Setup

Scenes, baselines, and inputs. We evaluate LiteReality-Agent on ten indoor scenes with 620 capture views, comparing against LiteReality (Huang et al., 2025b) and two general-purpose agents: GPT-6 running in Codex and Fable 5.1 running in Claude Code. Each general-purpose agent is evaluated with RGB photographs alone (RGB-only) and with additional geometric observations from LiteReality Scan (+Scan), including depth, camera calibration, a point cloud, and a RoomPlan layout. All methods use the same photographs for each scene; LiteReality and LiteReality-Agent receive the same scan information as the +Scan baselines.

Evaluation protocol. All reconstructions are aligned to the recorded camera coordinate system without changing scene scale or individual object placements, and rendered at 960×720 using the recorded camera poses and intrinsics. We evaluate fidelity at the captured viewpoints rather than held-out novel views. All methods share the lighting and colour-management settings of the corresponding LiteReality-Agent scene while retaining their own materials and textures, including material emission. This lighting is not independently calibrated to the reference photographs. Additional input and evaluation details are provided in Appendix C.

Table 2: Mean visual fidelity over 620 capture views from ten scenes, under shared cameras and scene-specific lighting. Bold is best.

Method	SSIM ↑	PSNR (dB) ↑	RMSE ↓	LPIPS ↓
LiteReality	0.5609	11.881	0.2635	0.6709
Fable 5.1 (RGB-only)	0.5307	10.681	0.3008	0.7197
Fable 5.1 (+Scan)	0.5649	12.287	0.2506	0.6422
GPT-6 (RGB-only)	0.5742	11.549	0.2694	0.6910
GPT-6 (+Scan)	0.6004	12.967	0.2323	0.6250
LiteReality-Agent (ours)	**0.6105**	**14.129**	**0.2033**	**0.5392**

Table 3: Geometric consistency and plausibility over ten scenes. Depth MAE over 611 frames; near-contact supported/total. Bold is best.

Method	Depth MAE (cm) ↓	Bottom near-contact ↑	Inters. pairs ↓
LiteReality	15.03	85/85 (**100%**)	43
Fable 5.1 (RGB-only)	58.64	71/102 (69.6%)	36
Fable 5.1 (+Scan)	16.32	78/97 (80.4%)	54
GPT-6 (RGB-only)	40.51	65/94 (69.1%)	14
GPT-6 (+Scan)	13.33	58/92 (63.0%)	24
LiteReality-Agent (ours)	**10.50**	86/86 (**100%**)	**0**



Figure 3: **Qualitative comparison with reconstruction baselines.** Three representative scenes compare the RGB input and RGB–D scan with LiteReality-Agent, GPT-6 Astra and Fable 5.1 with and without scan input, and LiteReality. All reconstruction results are rendered from the same calibrated camera viewpoint.

4.2 Visual Fidelity

We evaluate SSIM, PSNR, RMSE, and LPIPS on reference photographs and renders paired at the same resolution, without additional image alignment or appearance correction. We retain all 620 capture views and average per-frame scores with equal weight, so scenes with more views contribute more to the aggregate; metric configurations are provided in Appendix C. As shown in Table 2, LiteReality-Agent achieves the best mean on all four metrics. Compared with GPT-6 (+Scan), the strongest alternative, it improves PSNR by 1.162 dB and reduces LPIPS by 13.7%; compared with LiteReality, the corresponding gains are 2.248 dB and 19.6%. Figure 3 presents qualitative comparisons on three representative scenes.

4.3 Geometric Consistency and Physical Plausibility

We evaluate three aspects of geometric consistency: depth MAE (cm) over 611 frames with shared valid depth, bottom near-contact (the proportion of furniture objects with at least one sampled bottom point within 1 cm of an external supporting surface), and the number of furniture pairs with intersecting triangle surfaces. Furniture counts vary across methods, so object-based results must be interpreted alongside these counts. These measures assess specific aspects of physical plausibility, rather than establishing full physical validity. Implementation details are provided in Appendix D.

Results. Table 3 shows that LiteReality-Agent outperforms both scan-informed agent baselines on all three criteria, achieving a depth MAE of 10.50 cm (21.2% lower than GPT-6 (+Scan)), bottom near-contact for all 86 identified furniture objects, and no detected furniture–furniture surface intersections. Compared with LiteReality, it maintains 100% bottom near-contact while reducing depth MAE from 15.03 to 10.50 cm and detected intersection pairs from 43 to zero.



Figure 4: **Representative reconstruction comparisons.** A kitchen (top) and meeting room (bottom), with captured RGB photographs, reconstructed views, and point-cloud/scene plan views.

4.4 Effect of Scan Data on General-Purpose Agents

Adding LiteReality Scan data to the same RGB photographs improves all four visual metrics for both general-purpose agents. PSNR increases from 11.549 to 12.967 dB for GPT-6 and from 10.681 to 12.287 dB for Fable 5.1. Depth MAE decreases from 40.51 to 13.33 cm and from 58.64 to 16.32 cm, respectively, corresponding to reductions of 67.1% and 72.2%.

These fidelity gains do not consistently extend to the object-based layout measures: detected furniture intersection pairs increase from 14 to 24 for GPT-6 and from 36 to 54 for Fable 5.1, while GPT-6's bottom near-contact rate decreases from 69.1% to 63.0%. These comparisons support the complete system but do not isolate the harness from its other components; intersection counts also depend on the differing furniture populations.

4.5 Qualitative Results

End-to-end reconstruction gallery. Figure 4 shows a kitchen and meeting room with complementary capture and reconstruction views. Additional rooms and views appear in Appendix A.

From reconstruction to interaction. The editable reconstruction supports changes to materials, furniture, and fixtures without repeating capture or reconstruction. Figure 5 shows controlled programmatic edits; Appendix B.7 gives their implementation and complementary natural-language editing examples.

From reconstruction to simulation. Figure 6 shows the exported kitchen in MuJoCo, its response to shaking, and a Franka manipulation example. Separate bodies, support relations, and joints enable physical interaction; these demonstrations do not measure robot-task success or identify real-world dynamics.

4.6 Human and Agent-Based Perceptual Evaluation

Human and agent evaluators rate LiteReality (Huang et al., 2025b), Fable 5.1 (+Scan), GPT-6 (+Scan), and LiteReality-Agent on layout accuracy, object similarity, and overall scene similarity, each on a five-point scale (higher is better).

Evaluation protocol. Human participants compare reference images with matched-view reconstruction renders and orthographic plan views, with method identities anonymised and positions randomised per scene. Each participant rates all four methods on five scenes from the ten-scene



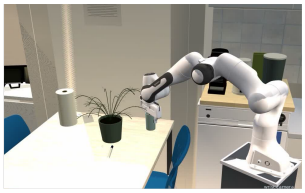
(a) Original (b) Grey carpet

(c) Cleared area (d) Pendant lamps

Figure 5: **Scene editing.** Programmatic edits to the reconstructed office: carpet recolouring, furniture removal, and pendant-light addition.



(a) MuJoCo scene (b) After shaking



(c) Franka pick-and-place

Figure 6: **Simulation and interaction.** The reconstructed kitchen in MuJoCo, after shaking, and in a Franka pick-and-place example.

	Layout ↑			Object ↑			Overall ↑		
Method	Hum.	A1	A2	Hum.	A1	A2	Hum.	A1	A2
LiteReality	2.03	3.00	3.00	1.95	1.70	1.70	2.11	1.80	1.80
Fable 5.1 (+Scan)	2.80	3.10	3.00	2.66	2.50	2.30	2.82	3.00	2.30
GPT-6 (+Scan)	3.11	4.10	3.60	2.98	3.20	2.90	3.00	3.60	2.90
LiteReality-Agent	**4.60**	**5.00**	**4.90**	**4.58**	**4.80**	**4.20**	**4.54**	**4.90**	**4.20**

Table 4: Perceptual scores (1–5; higher is better). Human means: 13 submissions, 65 ratings per method. A1 and A2: agent evaluators.

benchmark, giving 65 participant–scene evaluations across 13 verified submissions; ratings are averaged with equal weight and scene coverage is uneven. Agent-1 and Agent-2 rate the same methods and criteria. Appendix E gives the full protocol.

Results. LiteReality-Agent has the highest human means (Table 4): layout 4.60, object similarity 4.58, and overall similarity 4.54, exceeding GPT-6 (+Scan) by 1.49, 1.60, and 1.54, respectively. For overall similarity, our method wins 53 of 65 paired participant–scene comparisons against GPT-6 (+Scan), with eleven ties and one loss. Both agent evaluators also rank LiteReality-Agent first and GPT-6 (+Scan) second; their scores differ from human ratings. These ratings measure perceived fidelity. The small sample, uneven coverage, and dependent ratings do not establish statistical significance or physical validity. Agent ratings do not replace human judgments.

Limitations. While our approach produces strong visual results, physical properties in the simulation remain approximate, and layout and collision adjustments may cause the reconstructed scene to deviate from real-world observations. Addressing these limitations is an important direction for future work, as discussed in Appendix F.

5 Conclusion

We presented LiteReality-Agent, an open-source system that treats 3D reconstruction as a coding problem, turning RGB-D scans into realistic, articulated, and simulation-ready scenes. Its observe-edit-verify harness combines executable scene programs, specialised tools, and evidence-driven verification to improve reconstruction quality and reliability. Across ten scenes, our results show gains in visual fidelity, geometric consistency, and perceptual quality over frontier-model agent baselines. As coding agents advance, this orchestration framework offers a practical foundation for robust real-to-sim systems and downstream embodied AI.

AI USE STATEMENT

We used generative AI tools for two purposes in this work. First, we used them to edit the text of this paper: improving readability, correcting grammar. Second, we used AI coding assistants to create and edit the software code of our system, which includes code implementing the methods described in this paper. We have not used generative AI tools to generate synthetic data sets, to develop theoretical models or conceptual frameworks, to formulate mathematical claims, to propose or refine hypotheses, to design or provide feedback on our research methodology or experiments, to assist with translation, to clean or reformat datasets, to support qualitative or thematic data analysis, or to interpret results. The remaining required-disclosure tasks, which concern the proving of mathematical claims, are not applicable to this work, as it contains no proofs.

We note separately that large language models are the object of study in this paper rather than only an authoring aid: LiteReality-Agent is built around a coding agent, and the baselines we compare against are general-purpose coding agents. Their use is described in Sections 3 and 4.

We have reviewed all AI-assisted work. The authors read and checked every AI-edited passage in the manuscript and verified all references against their original sources. All AI-generated code was reviewed and tested by the authors, and every quantitative result and figure reported here was produced by executing that code. We take responsibility for the final content of this work, including its text, claims, and results.

REFERENCES

Rio Aguina-Kang, Maxim Gumin, Do Heon Han, Stewart Morris, Seung Jean Yoo, Aditya Ganeshan, R. Kenny Jones, Qiuhong Anna Wei, Kailiang Fu, and Daniel Ritchie. Open-Universe Indoor Scene Generation using LLM Program Synthesis and Uncurated Object Databases, February 2024.

Armen Avetisyan, Manuel Dahnert, Angela Dai, Manolis Savva, Angel X. Chang, and Matthias Nießner. Scan2CAD: Learning CAD Model Alignment in RGB-D Scans. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2609–2618, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-7281-3293-8. doi: 10.1109/CVPR.2019.00272.

Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. *IEEE Transactions on Robotics*, 2021. URL https://arxiv.org/abs/2007.11898.

Ata Çelen, Guo Han, Konrad Schindler, Luc Van Gool, Iro Armeni, Anton Obukhov, and Xi Wang. I-Design: Personalized LLM Interior Designer. In *Computer Vision – ECCV 2024 Workshops*, volume 15624, pp. 217–234. Springer Nature Switzerland, Cham, 2025. ISBN 978-3-031-92386-9 978-3-031-92387-6. doi: 10.1007/978-3-031-92387-6_17.

Tianyuan Dai, Josiah Wong, Yunfan Jiang, Chen Wang, Cem Gokmen, Ruohan Zhang, Jiajun Wu, and Li Fei-Fei. Automated creation of digital cousins for robust policy learning. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4912–4943. PMLR, 2025.

Matt Deitke, Rose Hendrix, Ali Farhadi, Kiana Ehsani, and Aniruddha Kembhavi. Phone2Proc: Bringing Robust Robots into Our Chaotic World. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9665–9675, Vancouver, BC, Canada, June 2023. IEEE. ISBN 979-8-3503-0129-8. doi: 10.1109/CVPR52729.2023.00932.

Weixi Feng, Wanrong Zhu, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato. Basu, Xin Eric Wang, and William Yang Wang. LayoutGPT: Compositional visual planning and generation with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 18225–18250. Curran Associates, Inc., 2023.

Rao Fu, Zehao Wen, Zichen Liu, and Srinath Sridhar. AnyHome: Open-Vocabulary Generation of Structured and Textured 3D Homes. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, volume 15097, pp. 52–70. Springer Nature Switzerland, Cham, 2024. ISBN 978-3-031-72932-4 978-3-031-72933-1. doi: 10.1007/978-3-031-72933-1_4.

Daoyi Gao, David Rozenberszki, Stefan Leutenegger, and Angela Dai. DiffCAD: Weakly-Supervised Probabilistic CAD Model Retrieval and Alignment from an RGB Image. *ACM Transactions on Graphics*, 43(4):1–15, July 2024. ISSN 0730-0301, 1557-7368. doi: 10.1145/3658236.

Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. LRM: Large reconstruction model for single image to 3D. In *International Conference on Learning Representations*, volume 2024, pp. 50678–50702, 2024.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. SceneCraft: An LLM agent for synthesizing 3D scenes as blender code. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 19252–19282. PMLR, 2024.

Ian Huang, Yanan Bao, Karen Truong, Howard Zhou, Cordelia Schmid, Leonidas Guibas, and Alireza Fathi. FirePlace: Geometric Refinements of LLM Common Sense Reasoning for 3D Object Placement. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13466–13476, Nashville, TN, USA, June 2025a. IEEE. ISBN 979-8-3315-4364-8. doi: 10.1109/CVPR52734.2025.01257.

Zhening Huang, Xiaoyang Wu, Fangcheng Zhong, Hengshuang Zhao, Matthias Nießner, and Joan Lasenby. LiteReality: Graphics-ready 3D scene reconstruction from RGB-D scans. In *Advances in Neural Information Processing Systems*, volume 38, pp. 162794–162827. Curran Associates, Inc., 2025b.

Team Hunyuan3D, Shuhui Yang, Mingxin Yang, Yifei Feng, Xin Huang, Sheng Zhang, Zebin He, Di Luo, Haolin Liu, Yunfei Zhao, Qingxiang Lin, Zeqiang Lai, Xianghui Yang, Huiwen Shi, Zibo Zhao, Bowen Zhang, Hongyu Yan, Lifu Wang, Sicong Liu, Jihong Zhang, Meng Chen, Liang Dong, Yiwen Jia, Yulin Cai, Jiaao Yu, Yixuan Tang, Dongyuan Guo, Junlin Yu, Hao Zhang, Zheng Ye, Peng He, Runzhou Wu, Shida Wei, Chao Zhang, Yonghao Tan, Yifu Sun, Lin Niu, Shirui Huang, Bojian Zheng, Shu Liu, Shilin Chen, Xiang Yuan, Xiaofeng Yang, Kai Liu, Jianchen Zhu, Peng Chen, Tian Liu, Di Wang, Yuhong Liu, Linus, Jie Jiang, Jingwei Huang, and Chunchao Guo. Hunyuan3D 2.1: From Images to High-Fidelity 3D Assets with Production-Ready PBR Material, June 2025.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics*, 42(4):1–14, August 2023. ISSN 0730-0301, 1557-7368. doi: 10.1145/3592433.

Weicheng Kuo, Anelia Angelova, Tsung-Yi Lin, and Angela Dai. Mask2CAD: 3D Shape Prediction by Learning to Segment and Retrieve. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision – ECCV 2020*, volume 12348, pp. 260–277. Springer International Publishing, Cham, 2020. ISBN 978-3-030-58579-2 978-3-030-58580-8. doi: 10.1007/978-3-030-58580-8_16.

Ruining Li, Yuxin Yao, Matt Zhou, Chuanxia Zheng, Christian Rupprecht, Joan Lasenby, Shangzhe Wu, and Andrea Vedaldi. Instruct-particulate: Scaling Feed-Forward 3D Object Articulation with Kinematic Control, 2026. URL https://arxiv.org/abs/2606.14699.

Lu Ling, Chen-Hsuan Lin, Tsung-Yi Lin, Yifan Ding, Yu Zeng, Yichen Sheng, Yunhao Ge, Ming-Yu Liu, Aniket Bera, and Zhaoshuo Li. Scenethesis: A language and vision agentic framework for 3D scene generation. In *International Conference on Learning Representations*, 2026.

Minghua Liu, Chao Xu, Haian Jin, Linghao Chen, Mukund Varma T., Zexiang Xu, and Hao Su. One-2-3-45: Any single image to 3D mesh in 45 seconds without per-shape optimization. In *Advances in Neural Information Processing Systems*, volume 36, pp. 22226–22246. Curran Associates, Inc., 2023a.

Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot One Image to 3D Object. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9264–9275, Paris, France, October 2023b. IEEE. ISBN 979-8-3503-0718-4. doi: 10.1109/ICCV51070.2023.00853.

Yuan Liu, Cheng Lin, Zijiao Zeng, Xiaoxiao Long, Lingjie Liu, Taku Komura, and Wenping Wang. SyncDreamer: Generating multiview-consistent images from a single-view image. In *International Conference on Learning Representations*, volume 2024, pp. 27676–27697, 2024.

Xiaoxiao Long, Yuan-Chen Guo, Cheng Lin, Yuan Liu, Zhiyang Dou, Lingjie Liu, Yuexin Ma, Song-Hai Zhang, Marc Habermann, Christian Theobalt, and Wenping Wang. Wonder3D: Single Image to 3D Using Cross-Domain Diffusion. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9970–9980, Seattle, WA, USA, June 2024. IEEE. ISBN 979-8-3503-5300-6. doi: 10.1109/CVPR52733.2024.00951.

Sining Lu, Guan Chen, Nam Anh Dinh, Itai Lang, Ari Holtzman, and Rana Hanocka. LL3M: Large Language 3D Modelers, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision – ECCV 2020*, volume 12346, pp. 405–421. Springer International Publishing, Cham, 2020. ISBN 978-3-030-58451-1 978-3-030-58452-8. doi: 10.1007/978-3-030-58452-8_24.

Zak Murez, Tarrence van As, James Bartolozzi, Ayan Sinha, Vijay Badrinarayanan, and Andrew Rabinovich. Atlas: End-to-End 3D Scene Reconstruction from Posed Images. In *European Conference on Computer Vision (ECCV)*, 2020. URL https://arxiv.org/abs/2003.10432.

Junfeng Ni, Yu Liu, Ruijie Lu, Zirui Zhou, Song-Chun Zhu, Yixin Chen, and Siyuan Huang. Decompositional Neural Scene Reconstruction with Generative Diffusion Prior. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6022–6033, Nashville, TN, USA, June 2025. IEEE. ISBN 979-8-3315-4364-8. doi: 10.1109/CVPR52734.2025.00565.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. SceneSmith: Agentic generation of simulation-ready indoor scenes. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026.

SAM 3D Team, Xingyu Chen, Fu-Jen Chu, Pierre Gleize, Kevin J Liang, Alexander Sax, Hao Tang, Weiyao Wang, Michelle Guo, Thibaut Hardin, Xiang Li, Aohan Lin, Jiawei Liu, Ziqi Ma, Anushka Sagar, Bowen Song, Xiaodong Wang, Jianing Yang, Bowen Zhang, Piotr Dollár, Georgia Gkioxari, Matt Feiszli, and Jitendra Malik. SAM 3D: 3Dfy Anything in Images, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023.

Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. LayoutVLM: Differentiable Optimization of 3D Layout via Vision-Language Models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 29469–29478, Nashville, TN, USA, June 2025. IEEE. ISBN 979-8-3315-4364-8. doi: 10.1109/CVPR52734.2025.02744.

Jiaming Sun, Yiming Xie, Linghao Chen, Xiaowei Zhou, and Hujun Bao. NeuralRecon: Real-Time Coherent 3D Reconstruction from Monocular Video. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. URL https://arxiv.org/abs/2104.00681.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, Vilamoura-Algarve, Portugal, October 2012. IEEE. ISBN 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. doi: 10.1109/IROS.2012.6386109.

Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual Geometry Grounded Transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. URL https://arxiv.org/abs/2503.11651.

Lilian Weng. Harness engineering for self-improvement. Lil'Log, July 2026. URL https://lilianweng.github.io/posts/2026-07-04-harness/. Blog post, accessed 26 September 2026.

Hongchi Xia, Chih-Hao Lin, Hao-Yu Hsu, Quentin Leboutet, Katelyn Gao, Michael Paulitsch, Benjamin Ummenhofer, and Shenlong Wang. HoloScene: Simulation-ready interactive 3D worlds from a single video. In *Advances in Neural Information Processing Systems*, volume 38, pp. 32501–32524. Curran Associates, Inc., 2025.

Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, Shuran Song, and Fangyin Wei. SAGE: Scalable Agentic 3D Scene Generation for Embodied AI, 2026.

Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, and Jiaolong Yang. Native and Compact Structured Latents for 3D Generation, 2025.

Jiale Xu, Weihao Cheng, Yiming Gao, Xintao Wang, Shenghua Gao, and Ying Shan. InstantMesh: Efficient 3D Mesh Generation from a Single Image with Sparse-view Large Reconstruction Models, 2024.

Kai Yan, Fujun Luan, Miloš Hašan, Thibault Groueix, Valentin Deschaintre, and Shuang Zhao. PSDR-Room: Single Photo to Scene using Differentiable Rendering. In *SIGGRAPH Asia*, 2023.

Yandan Yang, Baoxiong Jia, Peiyuan Zhi, and Siyuan Huang. PhyScene: Physically Interactable 3D Scene Synthesis for Embodied AI. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16262–16272, Seattle, WA, USA, June 2024a. IEEE. ISBN 979-8-3503-5300-6. doi: 10.1109/CVPR52733.2024.01539.

Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. SceneWeaver: All-in-one 3D scene synthesis with an extensible and self-reflective agent. In *Advances in Neural Information Processing Systems*, volume 38, pp. 140319–140351. Curran Associates, Inc., 2025.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli Vanderbilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language Guided Generation of 3D Embodied AI Environments. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16277–16287, Seattle, WA, USA, June 2024b. IEEE. ISBN 979-8-3503-5300-6. doi: 10.1109/CVPR52733.2024.01536.

Kaixin Yao, Longwen Zhang, Xinhao Yan, Yan Zeng, Qixuan Zhang, Lan Xu, Wei Yang, Jiayuan Gu, and Jingyi Yu. CAST: Component-Aligned 3D Scene Reconstruction from an RGB Image, 2025. URL https://arxiv.org/abs/2502.12894.

Yu-Ying Yeh, Zhengqin Li, Yannick Hold-Geoffroy, Rui Zhu, Zexiang Xu, Miloš Hašan, Kalyan Sunkavalli, and Manmohan Chandraker. PhotoScene: Photorealistic Material and Lighting Transfer for Indoor Scenes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Matt Zhou, Ruining Li, Xiaoyang Lyu, Zhaomou Song, Zhening Huang, Chuanxia Zheng, Christian Rupprecht, Andrea Vedaldi, and Shangzhe Wu. Articraft: An Agentic System for Scalable Articulated 3D Asset Generation, 2026.

A Additional Reconstruction Comparisons

Figures A1 and A2 show all nine reconstructed rooms using the same camera calibration and image crop as the main gallery, pairing human-height views with orthographic plan views.



Figure A1: Additional complementary view comparisons. Each case shows the RGB input, the corresponding RGB–D scan.

Figure A2 complements Figure A1 with a second matched viewpoint for each room, revealing additional scene regions and enabling a broader assessment of reconstruction consistency across viewpoints.



Figure A2: **Complementary reconstruction comparisons across all evaluation rooms.**

B LiteReality-Agent Implementation Details

This appendix supplements Sec. 3 with the capture format, implementation choices, and intermediate visual examples.

B.1 Capture Data and Layout Repair

Table 5 details the scan data used for reconstruction.

Capture component	Role in reconstruction
RGB frames	Appearance references for objects, materials, and fixtures.
Depth and confidence	Geometric evidence and visibility checks.
Camera calibration	Intrinsics and poses for projection and matched-view rendering; timestamps associate observations.
Derived point cloud	A spatial view of the captured surfaces and missing regions.
RoomPlan detections	Room structure, openings, and semantic object boxes with metric dimensions and poses.

Table 5: Capture data used by LiteReality-Agent. The point cloud is derived from RGB–D observations; the detections provide initial layout estimates rather than final object geometry.

Graph construction and support. Graph nodes represent the room, structural surfaces, openings, and objects. Relations record opening hosts, support, wall adjacency, and functional groups. Height and footprint overlap identify support candidates, with wall-hung objects distinguished from floor-standing furniture. Objects without plausible support are flagged. Expected containment, including chairs beneath tables, is distinct from conflicting footprints.

Bounded repair. Before cropping and generation, candidate layouts are compared by error count, then summed error magnitude. Fidelity limits apply cumulatively against the original scan. The implementation caps horizontal displacement at $0.6\,\mathrm{m}$ and vertical displacement at $0.08\,\mathrm{m}$. Per-axis size changes are limited to 15% for tables, desks, chairs, sofas, and beds, and 35% for other categories. Erroneous merges may be split into their original members; measured objects cannot be deleted to remove violations. Repair stops on validation, repeated lack of progress, or six agent-assisted rounds. Corrected boxes are handed to room assembly (Fig. A3).

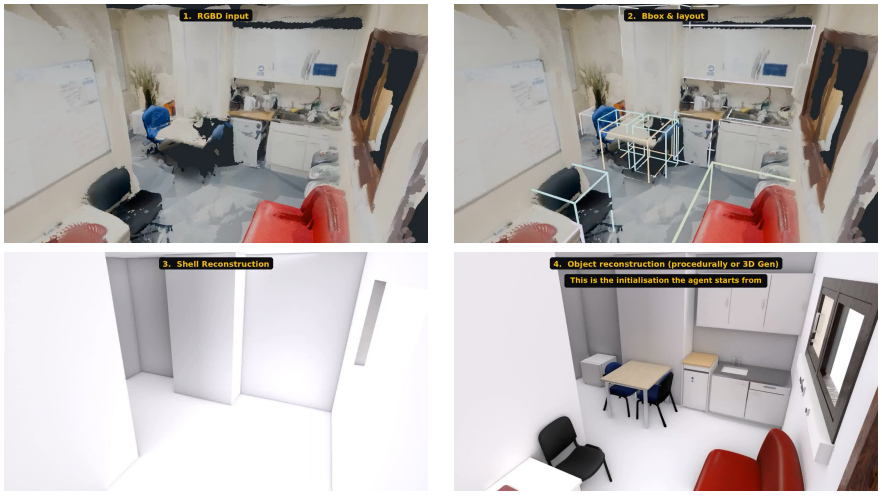


Figure A3: **From capture to an initial room.** The running kitchen example shows the RGB–D point cloud, detected boxes and layout, the reconstructed shell, and the assembled initial scene. Fine surface appearance, fixtures, and small objects are refined during authoring.

B.2 Layout Agent: Relations, Repair, and Before–After Results

The layout stage corrects the detected boxes before they determine object crops, reconstruction dimensions, and placement. It combines a geometric scene graph, a deterministic repair search, and an optional image-guided agent for unresolved cases. This ordering avoids propagating a misplaced or oversized detection into asset generation.

Relations constrain the edits. The graph distinguishes physical support from functional grouping. An object has one support parent—a floor, wall, ceiling, or another object—or is flagged as unsupported. A chair may belong to a desk's functional group while being supported by the floor. This distinction allows a desk and its chair to move together without claiming that the desk supports the chair. Separate relations encode wall adjacency, opening hosts, and expected containment, such as a chair tucked under a table. These category-aware exceptions prevent plausible box overlaps from being treated as errors.

Propose, validate, and retain. The geometric search tries wall alignment, corner fitting, separation, and small changes to box dimensions. Each candidate is checked against the room boundary, walls, other objects, and the scan-fidelity limits above. When geometry alone leaves an unresolved case, the optional agent receives the current box, nearby walls and objects, the violation, reference photographs, and feedback from rejected proposals. It can propose translation, resizing, attachment to one or two walls, or splitting a merged box into its original members. Proposals must preserve existing anchors, satisfy cumulative fidelity bounds, and improve the geometric score before they are retained. Rejected edits leave the accepted layout intact; unresolved findings remain in the report.

Recorded outcomes and scope. The saved runs reduce the layout checker's error count from 30 to 0. Eight scenes required changes; the two initially valid scenes were unchanged. Across 107 input boxes, 29 were moved and four were resized. These runs were resolved by the deterministic search and therefore do not measure the additional benefit of image-guided proposals. A zero count means that the boxes pass this checker's tolerances and category exemptions. Generated meshes and later authored objects still require the separate collision and support checks in Appendix B.6.

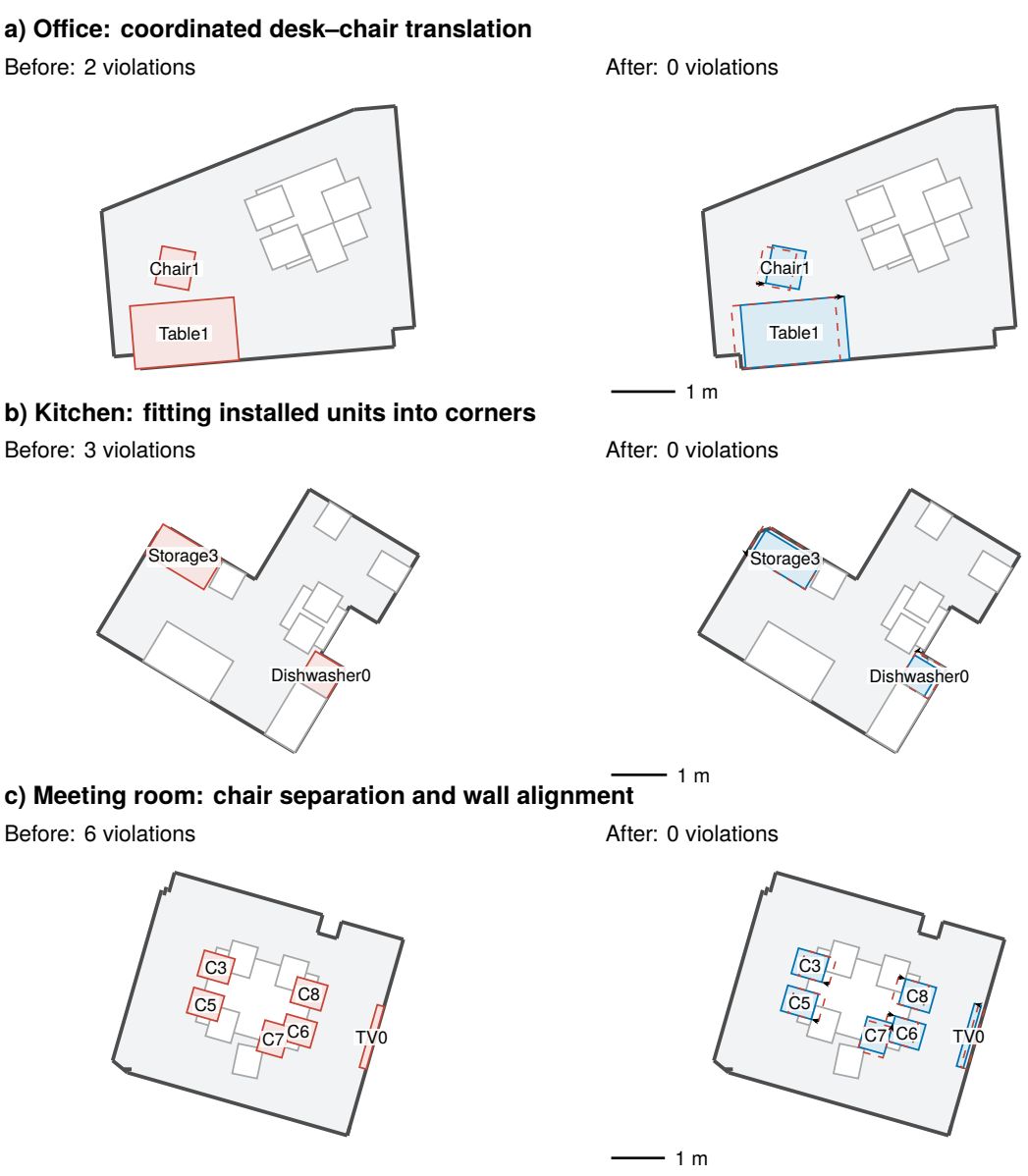


Figure A4: **Before and after layout repair.** Plans are drawn directly from the saved input and output boxes for three scenes. Red outlines in the left column identify boxes that are changed; blue outlines show their corrected states on the right. Dashed red outlines retain their original footprints, and arrows show centre translations. Each pair uses the same scale and viewpoint; grey objects and walls provide unchanged context. Meeting-room labels abbreviate chairs as C and the television as TV. These are layout-stage comparisons, before detailed mesh reconstruction and authoring.

Coordinated movement. In the office (Fig. A4a), `Table1` and `Chair1` both move by $(0.1445, 0.0124, 0)$ m. The shared 14.5 cm translation preserves their relative arrangement and removes the two recorded violations without resizing either box.

Corner fitting and local separation. In the kitchen (b), `Storage3` and `Dishwasher0` each move about 5.3 cm and reduce their two horizontal dimensions by about 8.5 cm, resolving three boundary/wall violations. In the meeting room (c), five chairs are separated and the television is aligned to its wall, reducing six violations to zero. All six boxes retain their dimensions; the largest centre shift is 14.1 cm.

B.3 Object Reconstruction and Physical Metadata

Figure A5 shows both reconstruction routes using clean references generated from scan views. Raw captures remain the evidence for checking fidelity: completed occluded parts are estimates.

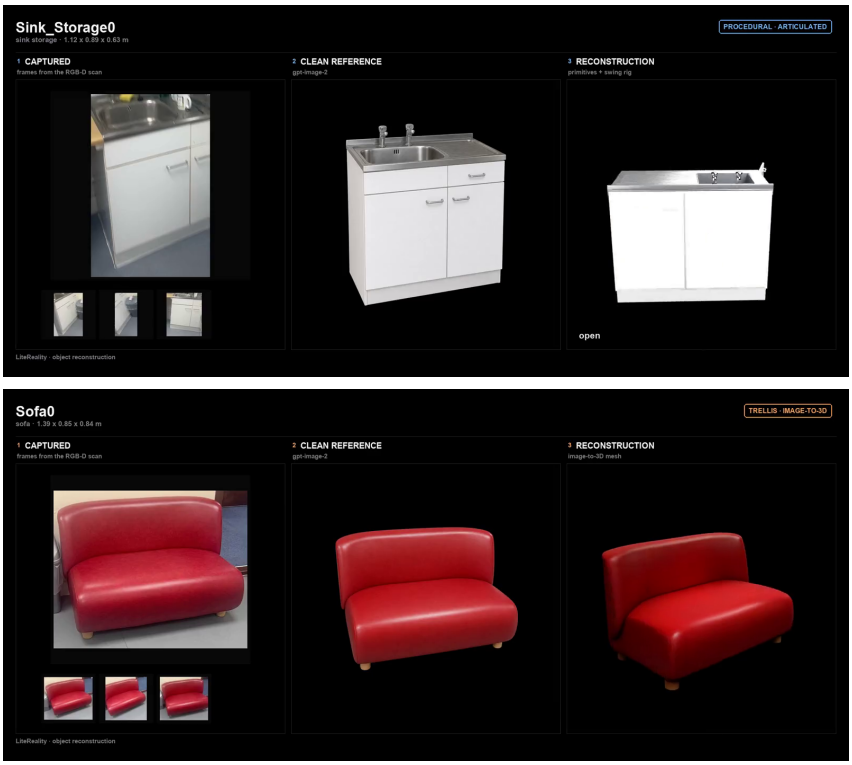


Figure A5: **Object reconstruction from capture references.** Each row shows captured views, the cleaned reference, and a reconstructed asset. The sink cabinet uses procedural geometry with articulated parts; the sofa uses image-to-3D generation.

Generated-mesh rejection and retries. Mesh checks look for implausible aspect ratios, disconnected fragments, extensive flat boundary surfaces, and dense bottom geometry suggesting that a floor or background was fused into the asset. The chair repair loop regenerates both the image reference and the mesh, varying the reconstruction seed between attempts. A retry is checked again before acceptance. If the retry budget is exhausted, the implementation retains the original asset and records the unresolved failure.

Estimating physical properties. The procedural program defines links, joint axes, pivots, and limits. Authored mass or density overrides automatic estimates. Otherwise, category-dependent occupancy density times bounding-box volume estimates total mass, which is distributed across links using geometric volume estimates. Inertia and centre of mass are computed from link geometry; convex-hull and box estimates provide fallbacks for degenerate meshes. Material-name priors supply friction and restitution when unspecified. Restitution is retained in the asset metadata but is not directly used by the MJCF exporter. Each physical-property record retains its source.

Collision geometry and solver checks. Near-convex parts can use a single hull; larger concave parts use convex decomposition. The decomposition budget depends on link mass to limit excessive contact complexity for lightweight objects. Per-object MuJoCo checks comprise a short drop, individual joint release, and tilted-gravity sliding. They report penetration, separation, unstable motion, and friction behaviour. Joint tests distinguish frame collisions from sibling interactions that may require a particular opening sequence.

B.4 Evidence Tools and Authoring State

Selecting and comparing views. The view selector supports room, wall, and object queries (Fig. A6). Room queries seek complementary coverage; local queries rank views of the requested target. The render tool rebuilds the current program and pairs the output with its capture photograph using recorded calibration. This exposes discrepancies in shape, placement, and appearance without asking the agent to align unrelated viewpoints.



Figure A6: **View selection at three scopes.** Room queries cover several surfaces, wall queries focus on one structural element, and object queries return observations of an individual asset. Numbers identify capture frames.



(a) Room: object labels (b) Wall: selected outline (c) Object: bounding box

Figure A7: **Evidence for authoring.** (a–c) Render–photograph pairs use room, wall, and object annotations to locate discrepancies at progressively finer scopes.

Persistent editing and materials. `Room.py` contains room-level construction and placement, while `object.py` contains a procedural asset's build recipe. Material retrieval saves colour, roughness, and normal maps together with a reusable recipe; recolouring changes the diffuse appearance while retaining the spatial pattern. Rebuilding these programs is the handoff between modules, ensuring that the exported geometry reflects the saved edits.

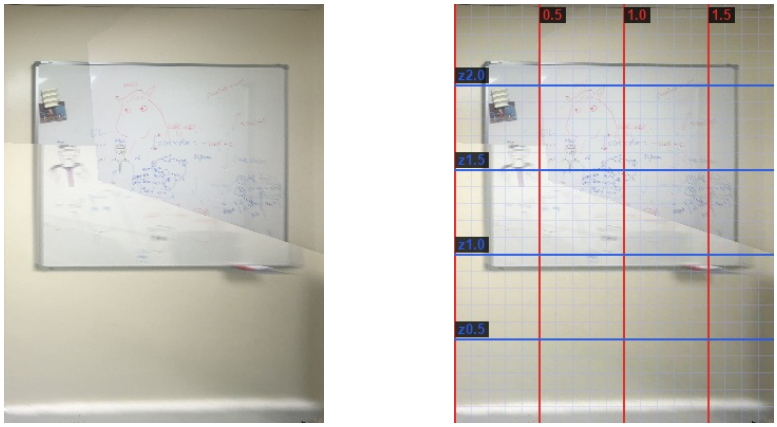
B.5 Stitched Wall Visualizations

Wall stitching and measurement. Figure A8 shows stitched wall references, their coverage, and the metric overlay. A metric grid is laid on the detected wall plane. Each output pixel corresponds to a 3D location that is projected into candidate frames using their calibration. Selection favours frontal, central, nearby observations, with depth visibility used when available. Selected frames fill the rectified image without blending; a coverage mask records unobserved pixels. The metric-grid tool overlays distance along the wall and height above the floor. These coordinates can be used directly to place fixtures. The planar projection can distort protruding objects, so raw capture views remain necessary for lamps, clutter, and furniture.



(a) Stitched wall reference

(b) Observed-pixel coverage mask



(c) Stitched whiteboard wall

(d) Metric-grid overlay

Figure A8: **Wall stitching and metric measurement.** Two walls from the running kitchen example are rectified from calibrated capture frames. (a) The stitched reference retains visible seams and projection artefacts around protruding furniture. (b) Its coverage mask distinguishes observed pixels (white) from unobserved pixels (black). (c–d) A second stitched wall is shown before and after adding a metric grid: red ticks measure distance along the wall and blue ticks measure height above the floor, in metres. The whiteboard edges can be read against this grid to guide placement and sizing in the scene program.

B.6 Quality Control and MuJoCo Integration

Collision and support are separate checks. A chair can overlap a table's box without intersecting its actual legs or top. We therefore use mesh intersections to establish clashes and convex approximations to estimate a correction. The resolver moves unanchored objects horizontally within a bounded correction budget, writes the accumulated changes to the scene program, and rebuilds before rechecking. It reports cases that cannot be solved by these local motions.

For a declared support, the validator compares the object's underside with the support's top surface and measures footprint overlap. Gaps above 1 cm and penetration exceeding 2 cm are blocking findings; less than half of the footprint overlapping the support is reported for review. Missing or unknown support declarations also block validation. For recognised floor-standing objects without an explicit declaration, the floor-contact fallback uses a 5 cm tolerance. Structural attachments are recorded separately and are exempt from gravity-support checks. These implementation checks differ from the evaluation metric in Appendix D, which measures sampled bottom near-contact and is not the authoring gate.

Visual review and bounded sessions. Deterministic checks address geometric failure modes; a separate, checklist-based model review can assess visual discrepancies. The maximum turn count and tool-call budget constrain the authoring session, while a reserved final portion of calls directs completion of edits. The reserve is configurable. Enforcing it by disabling exploratory tools requires a pre-tool hook; adapters without this hook implement a hard call-budget stop. In either case, the saved program must still be built and checked, and termination is not a QC verdict.

Exporting the physical scene. The exporter combines four sources: shell geometry from `Room.py`, object poses and support declarations from `room_layout.json`, appearance from `Room.glb`, and per-object physics records. Walls are decomposed into structural boxes around door and window openings. Attached fixtures are fixed, movable objects have free joints, and articulated parts use their specified hinge or slide joints.

The transform between an asset's local geometry and its placed instance also places its collision geometry and joint frames. When size changes, derived mass scales with volume, authored mass is retained, and inertia is recomputed from the placed geometry. Collider bounds are checked against the recorded object bounds. Joints with compiled effort metadata receive motor actuators; velocity limits are retained as controller metadata. Newly authored objects without physics records use export-time mass and collider estimates, and these fallbacks appear in the export report.

Figure 6 in the main paper illustrates the exported scene under perturbation and robot interaction.

B.7 Additional Scene-Editing Demonstrations

Figure A9 shows natural-language editing examples that complement the controlled programmatic edits in Fig. 5. The requests change the wall material, clear space for a meeting, and add decorations to the reconstructed office.

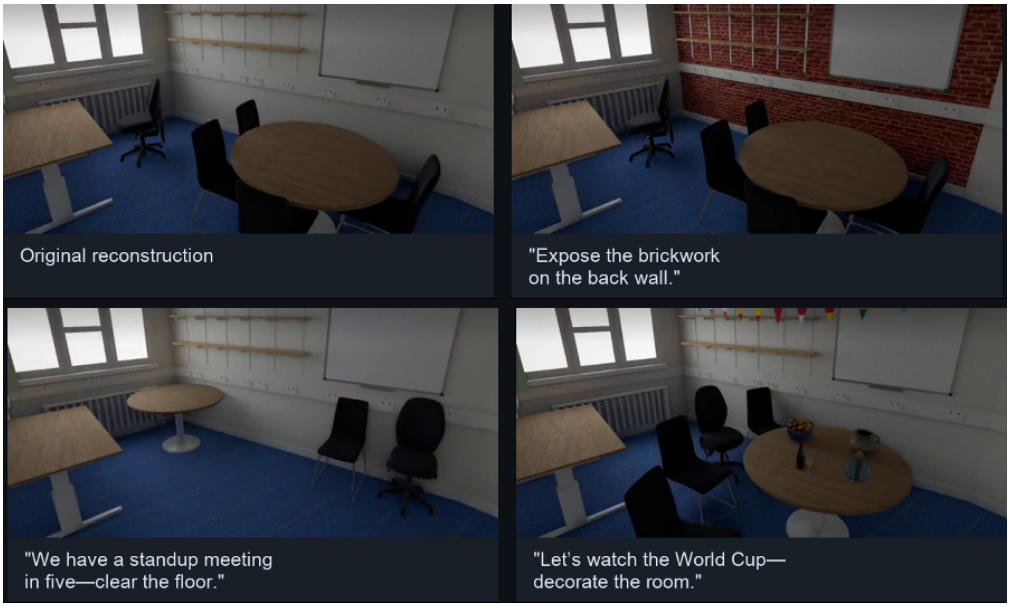


Figure A9: **Natural-language scene editing.** The original reconstruction is shown with three requested edits: expose the brickwork, clear the floor for a standup meeting, and decorate the room for watching the World Cup. Each instruction is applied as an edit to `Room.py`.

Controlled programmatic demonstrations. The variants in Fig. 5 each start independently from one saved 3D office scene and use an identical camera, resolution, and render configuration. The following details describe these main-paper examples.

Material editing. The carpet edit changes its colour while retaining the spatial texture, room geometry, and furniture placement. This isolates an appearance change from structural reconstruction.

Object removal. The layout edit removes the meeting-table group and four guest-chair groups. Walls, shelves, and other fixtures remain in place, exposing the floor previously occupied by the furniture.

Fixture addition. The lighting edit adds three pendant assemblies above the existing table. Each assembly contains a shade, suspension cord, and light source, with a ceiling-attachment declaration. The change affects both geometry and illumination.

Provenance and scope. The images in Fig. 5 are Blender renders of the saved office reconstruction, produced by a reproducible scene-editing script. They are qualitative examples of what the editable representation permits, not additional measurements of autonomous agent success or physical validity. The render manifest records the source scene, camera, resolution, and sampling settings for each variant.

C Additional Experimental Details

Scan inputs. For each scene, the +Scan condition supplements the RGB photographs with aligned LiDAR depth and confidence maps, camera intrinsics, poses, timestamps, a derived point cloud, and a RoomPlan USDZ layout collected or generated by LiteReality Scan. The RoomPlan layout encodes room structure and the semantic categories, dimensions, positions, and orientations of detected objects. LiteReality and LiteReality-Agent receive the same scan information as the +Scan baselines.

Evaluation views and camera alignment. The ten evaluation scenes contain 27–91 capture views each. All six method–input configurations are evaluated on the same scene–frame pairs, yielding 3,720 reference–render comparisons. Reconstructed Blender scenes are aligned to the coordinate system of the recorded camera poses while preserving scene scale and individual object placements. Rendering uses the recorded camera poses, with intrinsics adjusted to 960×720.

Rendering configuration. All scenes are rendered in Blender 5.2 using Cycles, with up to 64 samples per pixel and OptiX denoising. For each scene, all methods use the same light objects, world illumination, and colour-management settings from the corresponding LiteReality-Agent scene, while retaining their own materials, textures, and material emission. Architecture hidden for cutaway views is restored where identified. The shared lighting is not independently calibrated to the reference photographs.

Image preprocessing. Reference photographs are downsampled from 1920×1440 to 960×720 using Lanczos interpolation and paired with renders by scene and frame identifier. No additional warping, cropping, masking, rotation, or exposure/colour correction is applied. All capture frames, including dark renders, are retained.

Metric implementation. SSIM, PSNR, and RMSE are computed on stored RGB values normalised to $[0, 1]$, without linearisation. SSIM uses an 11×11 Gaussian window with $\sigma = 1.5$, population covariance, and averaging across RGB channels. LPIPS uses pretrained AlexNet with version-0.1 calibration weights, evaluated at 960×720 with inputs normalised to $[-1, 1]$.

Aggregation. For each metric, we report the arithmetic mean of per-frame scores over all 620 views. In particular, PSNR is averaged over individually computed per-frame PSNR values. Views receive equal weight, so scenes with more capture views contribute more to the aggregate.

D Geometric Evaluation Details

Depth agreement. We compute the mean absolute error between rendered and captured depth using pixels with valid depth in the capture and all evaluated outputs, including RoomPlan. We average per-frame errors over the 611 frames with shared valid pixels and report depth MAE in centimeters.

Bottom near-contact. We count furniture objects with at least one sampled bottom point within 1 cm of an external supporting surface. We report both the count and its percentage relative to all identified furniture objects in each method's output. This criterion measures bottom support proximity; it does not require every leg or base component to contact a supporting surface.

Furniture intersections. We detect strictly intersecting triangle surfaces in the evaluated source meshes using a 0.1 mm plane-side tolerance. This tolerance is not a penetration-depth threshold. Each furniture pair is counted once per scene, and counts are summed across the ten scenes. Intersections within a single furniture object and between furniture and architecture are excluded. The resulting counts describe detected surface intersections and require semantic interpretation.

Object populations. Object-based metrics use the furniture objects identified in each method's output. These populations differ across methods, so near-contact rates and intersection counts must be interpreted alongside the reported furniture counts.

E Additional Perceptual Evaluation Details

Rating criteria. We evaluate LiteReality, Fable 5.1 (+Scan), GPT-6 (+Scan), and LiteReality-Agent on three criteria using a five-point scale, with higher scores indicating closer agreement with the reference scene. Layout accuracy concerns room structure and object positions and orientations. Object similarity concerns object shape, scale, and appearance. Overall scene similarity assesses the reconstruction as a whole. We report the criteria separately without constructing a composite score.

Questionnaire design. The questionnaire presents a reference RGB image alongside reconstruction renders from the corresponding camera viewpoint, together with an orthographic plan-view comparison. Methods are displayed anonymously, with their positions randomised independently for each scene; displayed letters therefore do not identify a fixed method across scenes. Each participant evaluates five scenes sampled without replacement from the ten-scene benchmark and rates all four methods on all three criteria.

Sample inclusion and aggregation. The reported human results use 13 completed submissions with resolved method identities, comprising 65 participant–scene evaluations and 780 individual ratings. Each method receives 65 ratings per criterion. We report arithmetic means, giving each participant–scene evaluation equal weight. Scene coverage ranges from three to eleven participants, so scenes are not equally weighted. Three additional submissions containing unresolved anonymous method identifiers are excluded from the current aggregation pending verification of the identifier mapping.

Agent-based evaluation. We report two sets of agent-generated ratings, designated Agent-1 and Agent-2, for the same four methods and three criteria. Agent ratings are aggregated and presented separately from human ratings.

Additional comparisons. LiteReality-Agent achieves the highest human mean overall scene similarity in each of the ten scenes. Its mean advantage over GPT-6 (+Scan) on this criterion is 1.54 points for humans and 1.30 points for each agent evaluator. Agent-1 scores LiteReality-Agent above the human mean on all three criteria, whereas Agent-2 assigns a higher layout score but lower object and overall similarity scores. Thus, the evaluations agree on the two leading methods while differing in absolute score levels.

F LIMITATIONS AND FUTURE WORK

Physical properties lack ground-truth validation. The physical properties used for simulation rely largely on language-model inference and category or material priors, with geometry-based estimates for quantities such as inertia. We do not have ground-truth measurements of these properties for the reconstructed objects. Simulation export and consistency checks therefore do not establish that the scenes reproduce real-world dynamics. Collecting measured physical properties and validating simulated behaviour against real interactions are important future work.

Layout consistency does not guarantee accuracy. RoomPlan detections can contain errors in object dimensions and poses that persist in the reconstructed layout. Our collision and support corrections improve geometric consistency, but neither these checks nor visual agreement with captured images guarantee recovery of the true layout. Future work should more tightly integrate image evidence with collision and support constraints during layout refinement, and evaluate the resulting layouts against measured ground truth.

Broader evaluation should also cover larger and more structurally diverse environments.