ScanEdit: Hierarchically-Guided Functional 3D Scan Editing

Mohamed El Amine Boudjoghra[1] Ivan Laptev[2] Angela Dai[1]

[1]Technical University of Munich [2]MBZUAI

Figure 1. ScanEdit enables instruction-driven editing of complex, real-world scenes by rearranging their 3D scans. Given an input 3D scan and its object-level decomposition, we formulate a hierarchically-guided, multi-stage LLM-based approach that transforms high-level user instructions into concrete, tractable local instructions for objects, which can then be globally optimized to achieve the functional instruction. In this example, ScanEdit rearranges chairs, tables and a coffee machine to create a coffee drinking area.

Abstract

With the fast pace of 3D capture technology and resulting abundance of 3D data, effective 3D scene editing becomes essential for a variety of graphics applications. In this work we present ScanEdit, an instruction-driven method for functional editing of complex, real-world 3D scans. To model large and interdependent sets of objects, we propose a hierarchically-guided approach. Given a 3D scan decomposed into its object instances, we first construct a hierarchical scene graph representation to enable effective, tractable editing. We then leverage reasoning capabilities of Large Language Models (LLMs) and translate high-level language instructions into actionable commands applied hierarchically to the scene graph. Finally, ScanEdit integrates LLM-based guidance with explicit physical constraints and generates realistic scenes where object arrangements obey both physics and common sense. In our extensive experimental evaluation ScanEdit outperforms state of the art and demonstrates excellent results for a variety of real-world scenes and input instructions. Our code is available at aminebdj.github.io/scanedit

1. Introduction

The abundance of multi-view RGB and RGB-D sensors, now available even on commodity phones, enables convenient capture and detailed reconstruction of complex real-world 3D scenes. This opens up exciting possibilities for various applications in content creation, virtual and augmented reality, architectural design, robotics, and more. To fully leverage the captured 3D data, however, it is often essential to perform editing and rearrangement of the captured scenes – for instance, re-organizing objects for content creation where iterative editing is a fundamental paradigm, or enabling a robot to visualize the goal state of the environment given the target task.

Recent works have established the potential of leveraging powerful, pre-trained vision-language models (VLMs) for 3D scene synthesis, employing object retrieval from a synthetic 3D object database [11, 44, 56] to populate constructed 3D layouts to produce a 3D scene. Such VLMs enable interpretation of a natural language prompt to construct a 3D scene layout supporting synthetic 3D object assets adhering to the prompt. While this showcases the potential for leveraging high-level VLM-based reasoning, such generated 3D layouts populated with synthetic assets are much simpler than complex, cluttered, real-world environments (e.g., often hundreds of objects in the real-world scans we operate on, while synthetic scene synthesis approaches often generate an average number of 5 objects per scene) [29, 45]). We thus propose the first approach to address text-based editing of complex, real-world 3D scans by leveraging a localized prompting approach which can

provide reliable per-object initializations for our proposed graph optimization. In order to handle the large number of objects common in real 3D environments (e.g., we operate on scenes with 69-306 objects, which can easily overwhelm the context size of a VLM/LLM), we decompose the editing task into multiple stages, tackled hierarchically. We first construct a hierarchical scene graph and identify a relevant subgraph for the input prompt. For each object in the subgraph, we then transform the input prompt into low-level per-object instructions grounded in the object's reference frame, which is used to generate concrete object placements, performed by iteratively traversing our hierarchical scene graph. Finally, we use these object placements as initialization, and combine LLM-generated scene graph constraints with convex differentiable losses based on 3D object relations to achieve a physically and semantically plausible edited 3D scene. An example of scene rearrangement generated by our method is illustrated in Figure 1.

Our contributions are summarized as follows:

- We construct a hierarchical scene graph representation, leveraging LLM reasoning capabilities to compose graph relations, in order to enable multi-level scene analysis for functional scene editing, capable of handling complex, real-world 3D scans.
- We develop a scene graph optimization suitable for the highly diverse nature of 3D scans, employing both LLM-based object constraints in tandem with 3D-based physical constraints to avoid collisions and boundary violations, producing meaningful and plausible edited 3D scenes.

2. Related Works

Classical retrieval-based 3D scene modeling. Creating realistic 3D scenes requires spatial understanding of object arrangements, in order to produce a 3D scene layout populated with objects from a 3D database.

One can capture such knowledge by analyzing object co-occurrence patterns, such as how often certain objects appear together in real-world environments [12]. Another approach involves learning from human interactions, using affordance maps that show how people move and use objects within a space [14, 15, 23, 26]. To generate well-structured layouts, previous works have explored various optimization techniques, including non-linear methods that refine object placements based on spatial constraints [4, 13, 31, 53, 57, 59]. Although these methods can generate plausible layouts, they do not support complex reasoning required for interpreting high-level natural language prompts to enable rearrangement of 3D scans.

Learning for retrieval-based 3D scene synthesis. Advances in deep learning encouraged adoption of various techniques for 3D scene synthesis, using feed-forward networks [51, 61], VAEs [30, 54], GANs [55], and autoregressive models [52]. A popular approach is to employ autoregressive models [24, 34, 47–49] to predict object arrangements in a sequential fashion, and explicitly model relations among objects at the cost of spatial complexity.

Diffusion-based methods [19, 37, 38, 40, 41] overcome some of these limitations by modeling object distributions holistically. Recent methods such as LEGO-Net [50], CommonScene [60], and DiffuScene [45] further improve scene plausibility and coherence. These methods excel at unconditional scene generation, but focus on relatively simple synthetic scenes with 10-20 objects maximum, while we develop a hierarchical approach to tractably address complex, real-world scenes with hundreds of objects.

3D shape and scene editing. 3D editing has been addressed by a number of works. Most methods focus on 3D shape editing [5–9, 18, 32, 36], while more recent approaches tackle the more complex task of editing larger-scale 3D scenes [2, 17, 28, 43, 46]. Text2Room[20] generates 3D scenes from text via mesh optimization, while RoomDreamer[39] relies on scene graph prediction and layout sampling. Such methods focus on flexibility in localized editing, which in complex scenes can often alter object instance identities. In contrast, we edit the original scene mesh representation by re-arranging objects to satisfy high-level, functional text prompts.

LLMs/VLMs for retrieval-based 3D scene synthesis. Another recent direction of work explores 3D scene generation by synthesizing intermediate representations, such as scene graphs or layouts, paired with an asset repository [11, 16, 25, 27, 33, 56, 62]. While older methods [27] perform scene editing using graph and text via rule-based scene graph matching and retrieval, the rise of Large Multimodal Models (LMMs) has opened up new possibilities for open-ended and flexible scene understanding [1, 3]. For instance, LayoutGPT [11] uses language models to directly produce 3D layouts for indoor scenes. Similarly, Holodeck [56] employs large language models (LLMs) to generate spatial scene graphs, which are then used to optimize object placements, while [21] uses a loop where an LLM generates Blender code to render a scene, then refines it based on the output. Another approach, LayoutVLM [44] and [10], adopt a differentiable optimization based method for synthesizing 3D scenes from textual instructions. A recent work, Fireplace [22], addresses object placement in 3D scenes by leveraging large multimodal models (LMMs) and solving geometric constraints to achieve a logical placement of new objects. Unlike retrieval-based methods, Gala3D [62] uses an LLM to generate object arrangements from text, then applies Gaussian splatting and optimization for plausible layouts. In contrast, we aim to perform functional editing of real-world 3D scans, which are often complex, with far more objects (our scenes contain 69-306 objects),

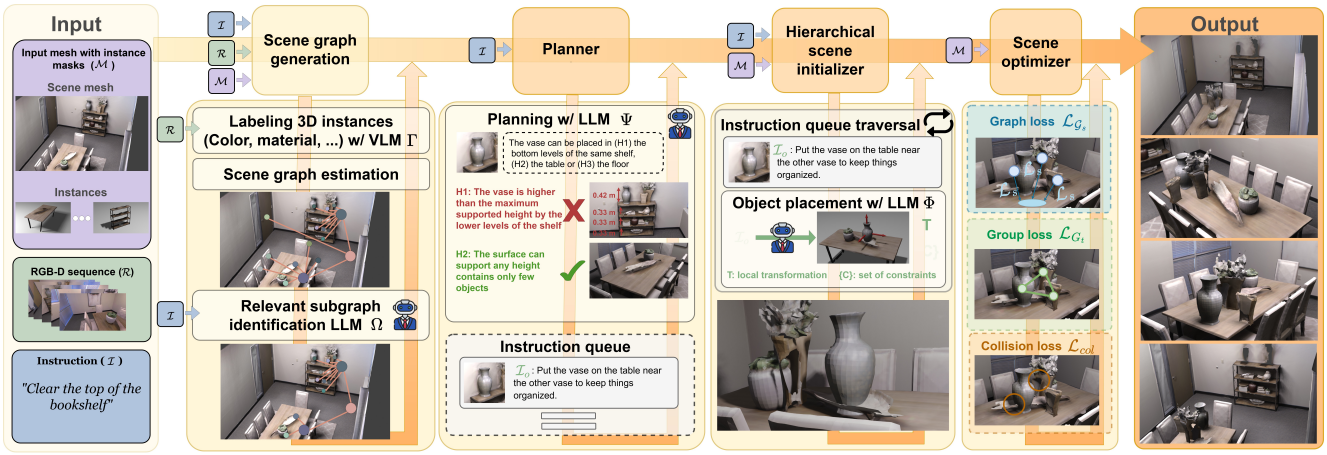

Figure 2. Overview of ScanEdit. Given an input instruction $\mathcal{I}$ for a scene mesh $\mathcal{M}$ that has an instance decomposition and is reconstructed from RGB-D sequence $\mathcal{R}$, we output an edited scene according to the instruction $\mathcal{I}$. We first construct a hierarchical scene graph $\mathcal{G}$ using 3D and VLM reasoning to annotate graph node and edge attributes. Since this graph may be very large in size, we then identify the relevant subgraph $\mathcal{G}_s$ for $\mathcal{I}$. Our planner then breaks down the high-level instruction $\mathcal{I}$ into low-level object instructions, validates them, and creates an instruction queue. We traverse the instruction queue hierarchically in order to place objects as initialization for the new output scene, followed by a scene optimization over both LLM-generated scene constraints as well as physical 3D collision constraints, to produce the output edited scene. In this example, the two vases on the top of the bookshelf are moved to the table in the 3D scan. Note that since real-world 3D scans are partial, some holes can be visible (e.g., in the output wall) after re-arranging objects.

requiring our hierarchical approach to characterize tractable editing while employing LLMs and VLMs.

3. Method

Given an input natural language text instruction $\mathcal{I}$ to edit a 3D scan $\mathcal{M}$ reconstructed from an RGB-D sequence $\mathcal{R}$ and decomposed into N semantic instances $\{o_1, ..., o_N\}$, we aim to predict an edited, rearranged scene $\mathcal{M}'$ as output based on $\mathcal{I}$. The edited output scene $\mathcal{M}'$ is composed by re-arranging the object instances $\{o_i\}$ with new locations and orientations defined as transformations $T_i \in \mathbb{R}^{4\times4}$. The T_i must lead to a physically plausible (without colliding or floating objects) scene configuration, while adhering to the instruction $\mathcal{I}$.

To achieve the above goal, we first construct a scene graph $\mathcal{G} = (N, E)$ representing $\mathcal{M}$, where we deploy a VLM Γ to annotate attribute features for the graph nodes N representing each object, as well as annotate the object-object edge hierarchical relations represented by E (Sec. 3.1). As the full scene graph $\mathcal{G}$ is typically large and contains hundreds of objects, we then identify the subgraph $\mathcal{G}_s$ relevant to $\mathcal{I}$ using an LLM agent (Sec. 3.2). For each object in $\mathcal{G}_s$, we then decompose $\mathcal{I}$ into localized, object-specific instructions that should be performed to achieve $\mathcal{I}$ (Sec. 3.3). These instructions are then interpreted through our hierarchical object placement for an initial output scene (Sec. 3.4). Finally, we optimize the edited scene $\mathcal{M}'$ by combining LLM-generated functional scene constraints with physical 3D scene constraints.

An overview of our method is presented in Figure. 2.

3.1. Hierarchical Scene Graph Construction

Our hierarchical scene graph $\mathcal{G} = (N, E)$ provides a compact and holistic representation for scene $\mathcal{M}$ to facilitate scene editing. Each node in $\mathcal{G}$ represents an object in $\mathcal{M}$, and the scene hierarchy is established based on several intra-object edge relationships: 'on top of', 'facing', and 'against wall'. Initially, nodes are simply given by the object instance masks of the geometry, and we further estimate node attributes and edges from the input scene information. We leverage LLMs for commonsense reasoning in object arrangement and enhance their limited spatial understanding by incorporating spatial information from 3D geometry into the scene graph. Our focus is on three specific edge type ('on top of', 'facing', and 'against wall') where LLMs typically struggle, unlike simpler relations such as distance or regular patterns, which they handle more reliably.

Node-specific attributes: Each object node $o_i \in N$ is characterized by the following: Class c_i; Color χ_i; Material m_i; Short Description d_i; Front Normal $\vec{F}_i$; Point Cloud P_i and Support Surfaces S_i.

Node-to-node relations: We encode these as directed edges $e^{o_i \to o_j} = o_i \to o_j$ in the scene graph, with possible edge types as 'on top of', 'facing', 'against wall'. $e^{o_i \to o_j}_{On-top-of}$ represents the relation where the object o_i is on top of one the surfaces of object o_j, $e^{o_i \to o_j}_{Against-wall}$ represents the relation where the object o_i is against the wall o_j, and $e^{o_i \to o_j}_{Facing}$

represents the relation where the object o_i is facing o_j.

Estimating node-specific attributes. Node attributes are annotated using a VLM agent Γ. To provide Γ with sufficient information, we use the RGB-D sequence $\mathcal{R}$ associated with the 3D scan $\mathcal{M}$ and rasterize 2D masks of 3D objects. Resulting image crops are used as input to VLM.
Color c_i, Material m_i, Short description d_i: These are all text attributes estimated by Γ. They describe the most dominant object color and material, as well as a short description of fine-grained details. *Sampled points P_i:* We describe the object geometry with sampled points on its surface at $\approx$ 1cm resolution.
Support surfaces S_i: are the regions of o_i that could potentially support another object.

A support surface S_i is defined by its set of contour points $S_{(i,c)}$ that define the boundaries of the support surface and the set of normals per contour point $S_{(i,\vec{n})}$ that point to the outside of the surface.
Estimating surface contour points $S_{(i,c)}$ and normals $S_{(i,\vec{n})}$: For a given surface level, we first identify contour points as those farthest from the barycenter. To estimate normals, we fit a spline interpolation to the contour, providing a smooth function from which we can sample points for improved normal estimation. These contour points and normals are then used to compute boundary loss in our method.
Front normal $\vec{F}_i$: We prompt the LLM to reason over the class name to identify objects likely to have a front face that can be extracted using a simple geometric cue: the region farthest from the centroid within the top-10% of points. For classes where the cue is less reliable, we fall back on geometric proxies such as the dominant axis of symmetry or the average vertex normal direction to estimate $\vec{F}_i$.

Estimating edges: We introduce an edge into the scene graph between nodes (o_i, o_j), based on 3D spatial reasoning in the scene to estimate the relations 'on top of', 'facing', and 'against wall'. We refer to the supplemental for further details.

3.2. Relevant Sub-graph Identification

Real 3D scans typically contain hundreds of objects, leading to thousands of relationships between them. This large amount of scene context can easily overwhelm the limited context window of a LLM agent. However, we observe that much of a scene graph may be irrelevant for a given instruction $\mathcal{I}$. To simplify planning and reduce hallucinations, we first employ LLM agent Ω to filter the input graph $\mathcal{G}$, identifying only the relevant objects and relationships, which results in a subgraph $\mathcal{G}_s \subseteq \mathcal{G}$ as the input for the planner. Relevant nodes include objects that need to be moved (including their children in the scene graph) and their potential target locations, while relevant relationships depend on spatial prompts in the instruction.

3.3. Localized Planning

Given the relevant subgraph $\mathcal{G}_s$ for the instruction $\mathcal{I}$, we employ a planner LLM Ψ to generate a new edited subgraph $\mathcal{G}'_s$ that should represent the edited scene according to $\mathcal{I}$. This is done by reconnecting edges of $\mathcal{G}_s$ to form the edited scene graph, as well as generating corresponding node-level instructions in the local reference frame of its parent.

Our LLM planner Ψ is invoked for $\mathcal{G}_s$, and prompted to generate a set of hypotheses as to how each object should be transformed to adhere to $\mathcal{I}$, and select the best out of the generated hypotheses. This multiple hypothesis generation and selection helps to maintain robustness against hallucinations and incorrect hypothesis generation. To promote the best outcome, we prompt Ψ to generate multiple hypotheses as object-level text instructions $\{I^{o_i}_k\}$ and select the most suitable one I^{o_i}. The LLM Ψ receives the sub-graph $\mathcal{G}_s$ to be edited, with each node described with its geometric information: *dimensions*, *maximum height for support surfaces* and *objects on top of each surface*. We prompt the LLM Ψ to take into account these details, generating each object's hypotheses and select the best one based on geometric plausibility. Since Ψ is also guided by the state of the target objects' support surfaces. It selects the most suitable surface based on available space and relevance.

The output of the planner will consist of an edited subgraph $\mathcal{G}'_s$, where each node has a corresponding instruction I^{o_i} in natural language. In case $\mathcal{G}_s$ contains identical objects but has to edit only one, Ψ is prompted to analyze their node attributes and locations to edit the most logical one.

3.4. Hierarchical object placement

Our hierarchical placement agent, LLM Φ, is called multiple times for each node in the edited sub-graph $\mathcal{G}'_s$. Given an object o_j in $\mathcal{G}'_s$, it determines the placement of its children relative to it. For each child, the agent generates an (x, y, z) position along with an orientation θ, specifying the desired pose for all children in **Children**(o_j). We denote the relative position and orientation with the matrix transform T (also shown in Figure 2). Additionally, it considers possible constraints (denoted $\{C\}$), selected from 'against wall', 'on top of', and 'facing' which are optimized during the optimization step.

3.5. Scene Subgraph Optimization

In this section, we introduce a set of convex differentiable loss functions designed to resolve graph edge constraints generated by the Placement LLM Φ. The surface loss ensures that an object stays within the boundaries of a designated support surface, while the against-wall loss ensures that the object is positioned correctly against a target wall

(if applicable). We also include a geometric collision loss, which helps resolve any collisions between the object and nearby objects. In general, we introduce a geometric group loss that preserves the structure of the groups during optimization. In our method, the 'facing' constraint is resolved in the hierarchical placement step where the child is placed to face the parent if 'facing' is required, while the other two edges types 'on top of' and 'against wall' are resolved in the optimization phase. Unlike LayoutVLM [44] and [1], our loss functions better capture the complexity of support surfaces and object geometry by operating directly on point clouds instead of simple bounding boxes.

"On-top-of" loss. Since LLMs are known to lack spatial awareness during reasoning, the Object Placement LLM Φ occasionally places objects in out-of-surface boundary locations to avoid collision with objects on the support surface. To ensure that objects are in their designated support surfaces, we introduce a loss to encourage an object to stay in the target support surface defined by constraint 'on top of' generated by the LLM.

Given an object o_i and a target surface S_j, we define the surface loss as the signed distance between the sampled points P_i of object o_i to S_j. The signed distance for a point p to S_j can be computed as $d(p) = (p - p_s) \cdot \vec{n_s}$ where p_s and $\vec{n}_s$ are the closest point in the contour points $S_{j,c}$ and the normal $S_{j,\vec{n}}$ of that contour point in S_j respectively. Finally, the loss is the average across all points:

$$\mathcal{L}_{\text{On-top-of}}(P_i, S_j) = \frac{1}{N} \sum_{p \in P_i} \max(0, d(p)).$$

"Against-wall" loss. Since some objects are naturally placed against walls, if the target location for an object is specified to be against a wall, we employ a loss to encourage the object to remain against its target wall during optimization.

Given a wall defined by its world-to-wall transformation T_{wall} and oriented bounding box (OBB) extents $Dx_{\text{wall}}, Dy_{\text{wall}}$, and an object o_i to be placed against the wall defined by its OBB extents Dx_o, Dy_o and center $c_{o_i} = [c_{o_i,x}, c_{o_i,y}]$ where $c_{o_i,x}, c_{o_i,y}$ are the *x*, *y* coordinates of the object o_i in the world coordinate system respectively. We initialize the object o_i with the same orientation as the wall, aligning their coordinate frames. The object's center is then adjusted to be positioned at a fixed distance from the wall, ensuring that the gap between their centers along the x-axis in the wall's coordinate frame is exactly $\frac{Dx_{\text{wall}} + Dx_o}{2}$. This guarantees that the object remains in contact with the wall without penetrating it.

After transforming both the wall and the object into the wall coordinate frame, they are aligned to face the positive x-direction. The object's position is then optimized to remain within $\pm \frac{Dy_{\text{wall}}}{2}$ along the y-axis in this frame. The object's center in this coordinate frame is computed as:

$$c_{o_i}^{\text{wall}} = c_{o_i} \cdot T_{\text{wall}}^{-T}$$

with OBB extents Dx_o, Dy_o. The **against-wall loss** is then defined as:

$$\mathcal{L}_{\text{AgainstWall}} = ||c_{o_i,x}^{\text{wall}} - \frac{Dx_{\text{wall}} + Dx_o}{2}|| + ||c_{o_i,y}^{\text{wall}}|| \cdot \text{OPT}$$

where

$$\text{OPT} = \mathbb{1}\left(|c_{o_i,y}^{\text{wall}}| > \frac{Dy_{\text{wall}}}{2}\right).$$

This formulation allows the object to slide freely along the y-axis within the boundaries of the wall.

Collision loss. LLM-generated placements often lead to collisions between objects. To resolve collisions during optimization, we propose the loss to discourage collisions.

Given the set of nodes $N_t \in N$ to be optimized for, where N are the nodes of the graph $\mathcal{G}$ and N_t are the nodes of $\mathcal{G}_s$, for all objects $o_i \in N_t$, we optimize a collision loss $\mathcal{L}_{\text{col},i}$ that pushes the object away from other object geometry in the scene. Specifically, the center of o_i should move away from the centers of other objects in $\mathcal{G}_s$, and any points belonging to other objects in $\mathcal{G}$ that lie in the bounding box of o_i should be pushed away from o_i's points P_i. We denote the set of points belonging to other objects' surfaces that lie in the bounding box of o_i as P_{o_i}.

$$\mathcal{L}_{\text{col},i}^{push} = \Big(\frac{1}{n} \cdot \sum_{d_i \in d(P_i, P_{o_i})} \max(0, \Delta - d_i) + \frac{1}{n_t - 1} \cdot \sum_{j \in N_t, j \neq i} \max(0, \Delta - ||c_i - c_j||)\Big),$$

where c_i is the center of the i^{th} object, $d : (\mathbb{R}^{n_1 \times 3}, \mathbb{R}^{n_2 \times 3}) \to \mathbb{R}^{n_1 \times n_2}$ is a distance function evaluating all pairwise distances between two sets of point clouds, $n = n_1 \times n_2$, and n_t is the total number of objects to be optimized for.

Finally, we multiply the push loss $\mathcal{L}_{\text{col},i}^{\text{push}}$ by a stop condition STOP_{col} which sets the loss to 0 if the object is not colliding with any other objects anymore. We measure collision by the minimum distance between the P_i and the P_{o_i}, where r is the resolution of the points P_i. which makes the final collision loss

$$\mathcal{L}_{\text{col},i} = \mathcal{L}_{\text{col},i}^{\text{push}} \cdot \text{STOP}_{\text{col}},$$

where

$$\text{STOP}_{\text{col}} = \mathbb{1}(\min(d(P_i, P_{o_i})) < 4 \cdot r).$$

The collision loss across all objects N_t is

$$\mathcal{L}_{\text{col}} = \sum_{o_i \in N_t} \mathcal{L}_{\text{col},i}.$$

Group loss. We define object groups by the sets of objects that share the common support surface and are assigned to the same parent in the scene graph. Generally, LLMs are capable of producing good arrangement of objects (e.g., chairs in a circle or vases in a line) but some of these objects can collide with other objects or be placed out of boundary. In order to preserve the structure of these groups during optimization, we introduce a loss that preserves the relative vectors between object centers similar to the initialization, and keeps group members in a distance similar to the initial distance to their group center.

Given a set of objects $\{o_0, o_1, o_2\}$ which belong to the same group where g_c is the parent node in the graph $\mathcal{G}'_s$ defined by LLM Φ, and their updated center at step t is $G_t = [c_{o_0,t}, c_{o_1,t}, c_{o_2,t}]$. The group loss is defined as:

$$\mathcal{L}_{G_\mathbf{t}} = \sum_{(o_{i,\mathbf{t}}, o_{i,\mathbf{0}}) \in (G_\mathbf{t}, G_\mathbf{0})} \sum_{(o_{j,\mathbf{t}}, o_{j,\mathbf{0}}) \in (G_\mathbf{t}, G_\mathbf{0})} ||v_{i \to j,\mathbf{0}} - v_{i \to j,\mathbf{t}}||,$$

where $v_{i \to j,\mathbf{t}}$ is the directional vector from object $o_{i,\mathbf{t}}$ to $o_{j,\mathbf{t}}$, and is defined as $v_{i \to j,\mathbf{t}} = c_{o_j,\mathbf{t}} - c_{o_i,\mathbf{t}}$.

Final optimization objective. For a set of constraints $\{C\}$ generated by the Placement LLM, the final objective is a weighted sum of the losses of all constraints $\{C\}$ that is described by the graph loss $\mathcal{L}_\mathcal{G}$, the group loss $\mathcal{L}_{G_t}$, and the collision loss $\mathcal{L}_{\text{col}}$.

$$\mathcal{L} = \mathcal{L}_{\mathcal{G}_s} + \alpha \mathcal{L}_{col} + \gamma \mathcal{L}_{G_t}.$$

Gradient Update. For a given object o_i placed on a support surface, we initialize its height to match that of the surface. This reduces the optimization to only the x, y translation and rotation around the z-axis. We optimize the objective function $\mathcal{L}$ using stochastic gradient descent (SGD) with a cosine annealing learning rate scheduler. For an object o_i with dimensions Dx_i and Dy_i along the x- and y-axes, respectively, we optimize a transformation T_i with three degrees of freedom: translation along x and y, and a rotation by angle θ.

Each object o_i is assigned a scheduler with a maximum step size in the x, y plane given by $(lr_{i,x,\max}, lr_{i,y,\max}) = 0.25 \cdot (Dx_i, Dy_i)$. This scheduler ensures that the step size is adapted to each object's dimensions, allowing for a more flexible and controlled optimization process.

4. Experimental setup

Our experimental setup includes 15 scenes, with 8 from ScanNet++ [58] (validation set) and 7 from Replica [42]. We use on average 8 instructions per scene, totaling 126 evaluation samples.

Evaluation Metrics. We assess performance using both perceptual and geometric metrics. The perceptual evaluation is based on the judgment of 36 human subjects, where each scores 20 random samples. It comprises a binary component, where users choose the best result of two given methods, and a unary component, where results of each method are rated on a five-point scale. For the geometric evaluation, we adopt PIoU from DiffuScene [45] to measure collisions. Since bounding box IoU only provides a coarse collision estimate, we also introduce the ColVol measure that approximates the cumulative volume of colliding object parts in a scene.

We also evaluate the percentage of objects that are not floating (NoFloat), as well as the percentage of objects within the bounds of the scene (InBound). For further details on the evaluation metrics we refer to the supplemental.

Baselines. We compare our method with state-of-the-art methods LayoutGPT [11] and LayoutVLM [44]. LayoutGPT relies on in-context samples retrieved to be similar to the input instruction, to generate a layout for retieved 3D assests. LayoutVLM uses a VLM for understanding the current state of the layout of 3D objects along with rendered images of the 3D scene to generate an arrangement of objects. This arrangement is obtained through VLM 3D intialization followed by differentiable optimization for a set of constraints also generated by the VLM.

5. Results

Comparison to state of the art. Table 1 shows a comparison to state-of-the-art methods LayoutGPT [11] and LayoutVLM [44], evaluating geometric plausibility of edited scenes from ScanNet++ [58] and Replica [42]. Our hierarchically-guided approach outperforms both baselines across all metrics. Both LayoutGPT and LayoutVLM struggle with complex, cluttered scenes, the former further struggling with spatial reasoning due to lack of optimization.

Figure 3 shows a qualitative comparison. Both LayoutGPT and LayoutVLM have difficulty handling the complex nature of real scan 3D scene environments, resulting in more collisions and misplaced objects. In contrast, our hierarchical decomposition enables more semantically and geometrically plausible edited scene outputs. This is further confirmed by our perceptual study, shown in Figure 5,

Table 1. Our method outperforms both LayoutGPT [11] and LayoutVLM [44] in all geometric and semantic metrics, due to our hierarchical approach and joint scene optimization.

Method	NoFloat (%) ↑	InBound (%) ↑	ColVol (m^3) ↓	PIoU ↓	CLIP score ↑
LayoutGPT [11]	51.34	62.12	1.3802	0.499	21.9
LayoutVLM [44]	66.47	85.01	1.4813	0.507	21.6
ScanEdit (Ours)	**88.87**	**99.22**	**1.3774**	**0.498**	**22.2**

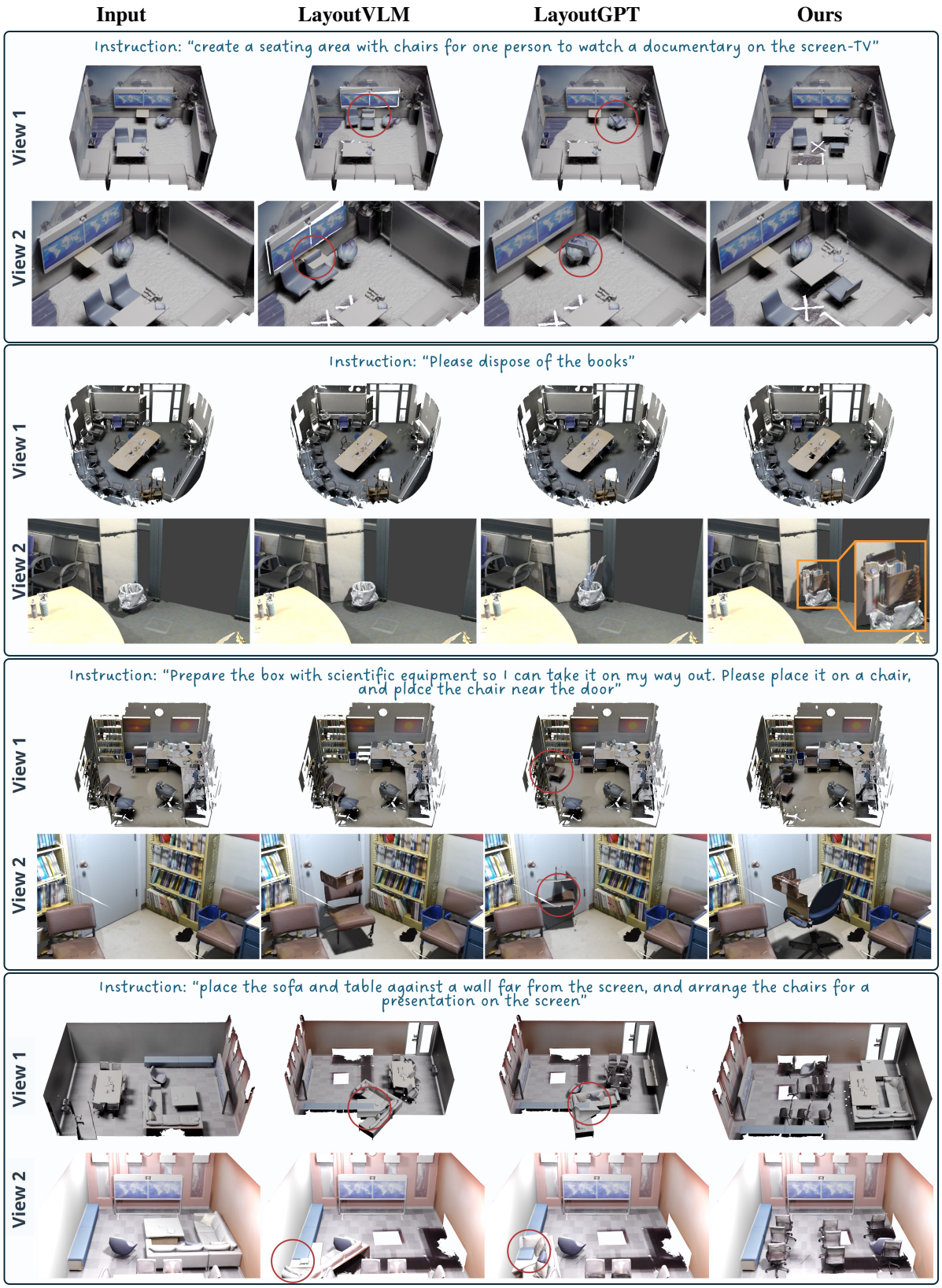

Figure 3. Qualitative comparison with baselines LayoutGPT [11] and LayoutVLM [44]. Red circles denote strong geometric errors (large collisions, out-of-boundary). Our method shows strong performance in adhering to the instruction while achieving physical plausibility.

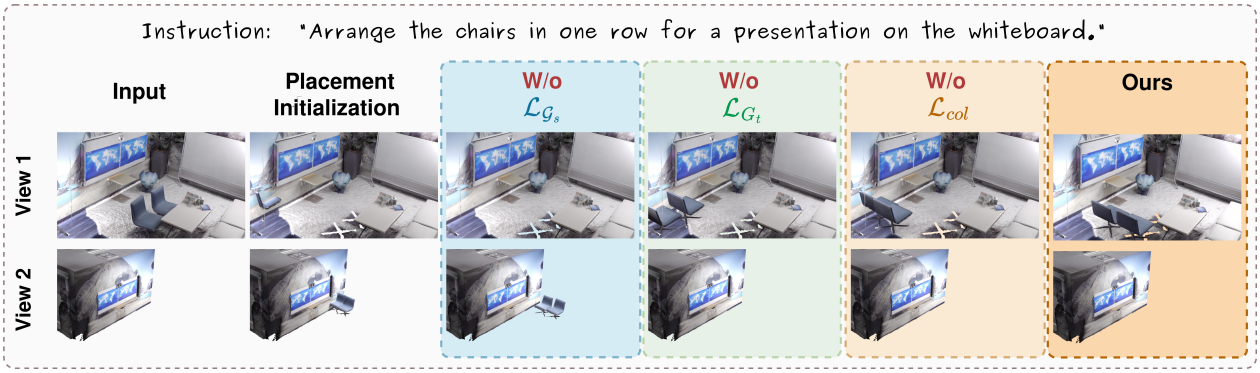

Figure 4. Ablation visualization over loss components. Our final loss with all components produces physically plausible results.

where our method is strongly preferred by participants, who also rate our edited scene results as having notably better layout quality and adherence to text inputs.

Editing with machine-generated masks. Figure 2 (supplementary) shows edits using class-agnostic masks from Mask3D [35], demonstrating strong adherence to instruction and physically plausible results.

Filling post-editing holes. Section 1 (supplementary) details our plane extrapolation method, with completed meshes shown in Figures 1 and 5.

5.1. Ablation studies

What is the impact of the joint scene optimization? In Table 4 we evaluate our approach without the final scene optimization (w/o Opt., using the object placement step as final object transforms). Our optimization shows a notable improvement, as it helps to resolve physical inconsistencies such as floating, colliding, or out-of-boundary objects.

What is the impact of our hierarchical planning, placement, and optimization? We also consider a simplified variant of our approach that relies on an LLM to directly generate object transforms for the relevant objects to be moved (w/o Plan. & Place w/o Opt. in Table 1). This yields low geometric performance, due to the LLM's limited 3D spatial understanding. This leads to more objects being placed outside the room boundaries, as reflected in the low InBound score, as well as increased collision cases, indicated by higher ColVol and PIoU values. Additionally, many objects are left floating, as seen in the low NoFloat score. In contrast, our proposed hierarchical planning and placement, followed by scene optimization significantly improves output scene plausibility, see also Figure 4.

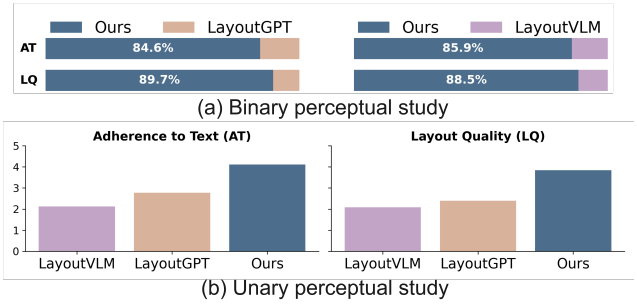

Figure 5. Our perceptual study shows that users strongly prefer our method compared to baselines LayoutGPT [11] and LayoutVLM [44], in both of adherence to text instruction (AT) and layout quality of the edited scene (QL).

Table 2. Ablation over eight scenes. Contribution of our Planner (Plan) together with Placer (Place), and optimization (opt).

Method	NoFloat (%)↑	InBound (%)↑	ColVol (m^3)↓	PIoU ↓	CLIP score ↑
w/o Plan. & Place w/o Opt.	51.47	61.78	1.3473	0.475	22.1
w/o Opt.	77.08	90.08	1.3693	0.482	22.2
ScanEdit (Ours)	**88.41**	**99.77**	**1.3381**	**0.472**	**22.4**

Limitations. While our approach shows effective editing results for various complex real-world 3D scans, multiple limitations remain. One challenge is accurately identifying small or long-tail objects when relying on machine-generated masks. Moreover, while LLMs are quite powerful in determining which objects should be transformed, and potential coarse locations, they nonetheless still lack knowledge in more perceptual reasoning, which our 3D optimization constraints also do not consider. For instance, we can find possible locations for a vase on a table to move onto a shelf based on what would spatially fit, but we cannot account for which possibilities would be the most common sense ones or the most aesthetically pleasing.

6. Conclusion

We have introduced ScanEdit, a hierarchically-guided approach to decompose high-level, natural language edits for 3D scenes into structured, holistic scene edits. This enables editing of complex, real-world 3D scans composed of hundreds of objects. Our use of LLM-generated high-level scene constraints together with 3D spatial constraints enables produced re-arrangements of 3D scans that achieve physically and semantically plausible edited 3D scans. We believe this represents an important step towards natural editing of complex 3D scans for various content creation scenarios.

Acknowledgements

This work was supported by the ERC Starting Grant SpatialSem (101076253).

References

[1] 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. *arXiv preprint arXiv:2403.09675*, 2024. 2, 5

[2] Alexey Bokhovkin, Quan Meng, Shubham Tulsiani, and Angela Dai. Scenefactor: Factored latent 3d diffusion for controllable 3d scene generation. *arXiv preprint arXiv:2412.01801*, 2024. 2

[3] Ata Çelen, Guo Han, Konrad Schindler, Luc Van Gool, Iro Armeni, Anton Obukhov, and Xi Wang. I-design: Personalized llm interior designer. *arXiv preprint arXiv:2404.02838*, 2024. 2

[4] Angel Chang, Manolis Savva, and Christopher D Manning. Learning spatial knowledge for text to 3d scene generation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 2028–2038, 2014. 2

[5] Hansheng Chen, Ruoxi Shi, Yulin Liu, Bokui Shen, Jiayuan Gu, Gordon Wetzstein, Hao Su, and Leonidas Guibas. Generic 3d diffusion adapter using controlled multi-view editing. *arXiv preprint arXiv:2403.12032*, 2024. 2

[6] Minghao Chen, Junyu Xie, Iro Laina, and Andrea Vedaldi. Shap-editor: Instruction-guided latent 3d editing in seconds. In *CVPR*, 2024.

[7] Dale Decatur, Itai Lang, Kfir Aberman, and Rana Hanocka. 3d paintbrush: Local stylization of 3d shapes with cascaded score distillation. In *CVPR*, 2024.

[8] Shaocong Dong, Lihe Ding, Zhanpeng Huang, Zibin Wang, Tianfan Xue, and Dan Xu. Interactive3d: Create what you want by interactive 3d generation. In *CVPR*, 2024.

[9] Ziya Erkoç, Can Gümeli, Chaoyang Wang, Matthias Nießner, Angela Dai, Peter Wonka, Hsin-Ying Lee, and Peiye Zhuang. Preditor3d: Fast and precise 3d shape editing. *arXiv preprint arXiv:2412.06592*, 2024. 2

[10] Aguina-Kang et al. Open-universe indoor scene generation using llm program synthesis and uncurated object databases. 2024. 2

[11] 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. *Advances in Neural Information Processing Systems*, 36, 2024. 1, 2, 6, 7, 8

[12] Matthew Fisher and Pat Hanrahan. Context-based search for 3d models. In *ACM SIGGRAPH Asia 2010 papers*, pages 1–10. 2010. 2

[13] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. Example-based synthesis of 3d object arrangements. *ACM Transactions on Graphics (TOG)*, 31(6):1–11, 2012. 2

[14] Matthew Fisher, Manolis Savva, Yangyan Li, Pat Hanrahan, and Matthias Nießner. Activity-centric scene synthesis for functional 3d scene modeling. *ACM Transactions on Graphics (TOG)*, 34(6):1–13, 2015. 2

[15] Qiang Fu, Xiaowu Chen, Xiaotian Wang, Sijia Wen, Bin Zhou, and Hongbo Fu. Adaptive synthesis of indoor scenes via activity-associated object relation graphs. *ACM Transactions on Graphics (TOG)*, 36(6):1–13, 2017. 2

[16] Rao Fu, Zehao Wen, Zichen Liu, and Srinath Sridhar. Anyhome: Open-vocabulary generation of structured and textured 3d homes. In *European Conference on Computer Vision*, pages 52–70. Springer, 2025. 2

[17] Ayaan Haque, Matthew Tancik, Alexei A. Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19740–19750, 2023. 2

[18] Ayaan Haque, Matthew Tancik, Alexei A Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *ICCV*, 2023. 2

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 2

[20] Lukas Höllein, Ang Cao, Andrew Owens, Justin Johnson, and Matthias Nießner. Text2room: Extracting textured 3d meshes from 2d text-to-image models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7909–7920, 2023. 2

[21] 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 *Forty-first International Conference on Machine Learning*, 2024. 2

[22] 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. *arXiv preprint arXiv:2503.04919*, 2025. 2

[23] Yun Jiang, Marcus Lim, and Ashutosh Saxena. Learning object arrangements in 3d scenes using human context. *arXiv preprint arXiv:1206.6462*, 2012. 2

[24] Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. Grains: Generative recursive autoencoders for indoor scenes. *ACM Transactions on Graphics (TOG)*, 38(2):1–16, 2019. 2

[25] Chenguo Lin and Yadong Mu. Instructscene: Instruction-driven 3d indoor scene synthesis with semantic graph prior. *arXiv preprint arXiv:2402.04717*, 2024. 2

[26] Rui Ma, Honghua Li, Changqing Zou, Zicheng Liao, Xin Tong, and Hao Zhang. Action-driven 3d indoor scene evolution. *ACM Trans. Graph.*, 35(6):173–1, 2016. 2

[27] Rui Ma, Akshay Gadi Patil, Matthew Fisher, Manyi Li, Sören Pirk, Binh-Son Hua, Sai-Kit Yeung, Xin Tong,

Leonidas Guibas, and Hao Zhang. Language-driven synthesis of 3d scenes from scene databases. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018. 2

[28] Başak Melis Öcal, Maxim Tatarchenko, Sezer Karaoğlu, and Theo Gevers. Sceneteller: Language-to-3d scene generation. In *European Conference on Computer Vision*, pages 362–378. Springer, 2024. 2

[29] Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. *Advances in Neural Information Processing Systems*, 34:12013–12026, 2021. 1

[30] Pulak Purkait, Christopher Zach, and Ian Reid. Sg-vae: Scene grammar variational autoencoder to generate new indoor scenes. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV 16*, pages 155–171. Springer, 2020. 2

[31] Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. Human-centric indoor scene synthesis using stochastic grammar. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5899–5908, 2018. 2

[32] Zhangyang Qi, Yunhan Yang, Mengchen Zhang, Long Xing, Xiaoyang Wu, Tong Wu, Dahua Lin, Xihui Liu, Jiaqi Wang, and Hengshuang Zhao. Tailor3d: Customized 3d assets editing and generation with dual-side images. *arXiv preprint arXiv:2407.06191*, 2024. 2

[33] Ohad Rahamim, Hilit Segev, Idan Achituve, Yuval Atzmon, Yoni Kasten, and Gal Chechik. Lay-a-scene: Personalized 3d object arrangement using text-to-image priors. *arXiv preprint arXiv:2406.00687*, 2024. 2

[34] Daniel Ritchie, Kai Wang, and Yu-an Lin. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6182–6190, 2019. 2

[35] Jonas Schult, Francis Engelmann, Alexander Hermans, Or Litany, Siyu Tang, and Bastian Leibe. Mask3d: Mask transformer for 3d semantic instance segmentation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8216–8223. IEEE, 2023. 8

[36] Etai Sella, Gal Fiebelman, Peter Hedman, and Hadar Averbuch-Elor. Vox-e: Text-guided voxel editing of 3d objects. In *ICCV*, 2023. 2

[37] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015. 2

[38] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 2

[39] Liangchen Song, Liangliang Cao, Hongyu Xu, Kai Kang, Feng Tang, Junsong Yuan, and Yang Zhao. Roomdreamer: Text-driven 3d indoor scene synthesis with coherent geometry and texture. *arXiv preprint arXiv:2305.11337*, 2023. 2

[40] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[41] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448, 2020. 2

[42] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 6

[43] Chunyi Sun, Yanbin Liu, Junlin Han, and Stephen Gould. Nerfeditor: Differentiable style decomposition for 3d scene editing. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 7306–7315, 2024. 2

[44] 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. *arXiv preprint arXiv:2412.02193*, 2024. 1, 2, 5, 6, 7, 8

[45] Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Denoising diffusion models for generative indoor scene synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20507–20518, 2024. 1, 2, 6

[46] Junjie Wang, Jiemin Fang, Xiaopeng Zhang, Lingxi Xie, and Qi Tian. Gaussianeditor: Editing 3d gaussians delicately with text instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20902–20911, 2024. 2

[47] Kai Wang, Manolis Savva, Angel X Chang, and Daniel Ritchie. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018. 2

[48] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X Chang, and Daniel Ritchie. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)*, 38(4):1–15, 2019.

[49] Xinpeng Wang, Chandan Yeshwanth, and Matthias Nießner. Sceneformer: Indoor scene generation with transformers. In *2021 International Conference on 3D Vision (3DV)*, pages 106–115. IEEE, 2021. 2

[50] Qiuhong Anna Wei, Sijie Ding, Jeong Joon Park, Rahul Sajnani, Adrien Poulenard, Srinath Sridhar, and Leonidas Guibas. Lego-net: Learning regular rearrangements of objects in rooms. *arXiv preprint arXiv:2301.09629*, 2023. 2

[51] Mingdong Wu, Fangwei Zhong, Yulong Xia, and Hao Dong. Targf: Learning target gradient field for object rearrangement. *arXiv preprint arXiv:2209.00853*, 2022. 2

[52] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion gans. *arXiv preprint arXiv:2112.07804*, 2021. 2

[53] Kun Xu, Kang Chen, Hongbo Fu, Wei-Lun Sun, and Shi-Min Hu. Sketch2scene: Sketch-based co-retrieval and co-placement of 3d models. *ACM Transactions on Graphics (TOG)*, 32(4):1–15, 2013. 2

[54] Haitao Yang, Zaiwei Zhang, Siming Yan, Haibin Huang, Chongyang Ma, Yi Zheng, Chandrajit Bajaj, and Qixing Huang. Scene synthesis via uncertainty-driven attribute synchronization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5630–5640, 2021. 2

[55] Ming-Jia Yang, Yu-Xiao Guo, Bin Zhou, and Xin Tong. Indoor scene generation from a collection of semantic-segmented depth images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15203–15212, 2021. 2

[56] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16227–16237, 2024. 1, 2

[57] Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D Goodman, and Pat Hanrahan. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Transactions on Graphics (TOG)*, 31(4):1–11, 2012. 2

[58] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. Scannet++: A high-fidelity dataset of 3d indoor scenes. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2023. 6

[59] Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley J Osher. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011, v. 30,(4), July 2011, article no. 86*, 30(4), 2011. 2

[60] Guangyao Zhai, Evin Pınar Örnek, Shun-Cheng Wu, Yan Di, Federico Tombari, Nassir Navab, and Benjamin Busam. Commonscenes: Generating commonsense 3d indoor scenes with scene graphs. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[61] Zaiwei Zhang, Zhenpei Yang, Chongyang Ma, Linjie Luo, Alexander Huth, Etienne Vouga, and Qixing Huang. Deep generative modeling for scene synthesis via hybrid representations. *ACM Transactions on Graphics (TOG)*, 39(2):1–21, 2020. 2

[62] Xiaoyu Zhou, Xingjian Ran, Yajiao Xiong, Jinlin He, Zhiwei Lin, Yongtao Wang, Deqing Sun, and Ming-Hsuan Yang. Gala3d: Towards text-to-3d complex scene generation via layout-guided generative gaussian splatting. *arXiv preprint arXiv:2402.07207*, 2024. 2