# Language-Driven Primitive-Based 3D Scene Generation with Infinigen

Eyrin Kim
Stanford University
eyrinkim@stanford.edu

Michelle Lau
Stanford University
mblau@stanford.edu

## Abstract

*This paper introduces a system that enables natural language control over procedural 3D content generation. Unlike existing approaches that rely on predefined templates, our system directly accesses geometric primitive functions within Infinigen to construct arbitrary 3D assets from scratch. By analyzing natural language descriptions with Large Language Models (LLMs) and translating them into Infinigen's primitive functions, we eliminate the need for domain-specific factories while maintaining procedural generation advantages. Our implementation demonstrates the viability of this approach through a complete pipeline converting language input to Blender-compatible assets. Experimental evaluation across representative prompts demonstrates that the prompt decomposition and post-generation validation steps significantly improve structural accuracy and semantic alignment. While the system is effective for standard geometries, it struggles with complex or unconventional forms, motivating future work in terms of spatial reasoning and adaptive validation. Overall, the capability achieved in this work steps towards greater flexibility in generating customized training environments for computer vision and robotics systems.*

## 1. Introduction

The generation of 3D scenes from natural language input represents a significant frontier in computer graphics, with broad applications across robotics, gaming, virtual reality, and the creation of synthetic datasets for computer vision. Recent advancements have enabled the synthesis of increasingly photorealistic environments; however, many of these systems operate within constrained frameworks that limit flexibility or control over individual scene elements.

Several tools today, such as SceneCraft, allow users to generate Blender-compatible scenes from text prompts, but rely heavily on pre-built assets and rigid template logic, restricting customization and asset diversity [2]. In contrast, more recent systems like Infinigen enable procedural generation of highly detailed and reproducible 3D environments entirely from scratch [3]. Infinigen leverages Blender's rendering engine and exposes a compositional asset construction pipeline that can, in principle, produce infinite variations. However, while procedural systems provide greater scalability and realism, they remain largely inaccessible to users without deep domain expertise, requiring a strong understanding of internal APIs, function parameters, and constraints, which in turn creates a barrier for those unfamiliar with scripting or geometry. Asset generation in Infinigen is often driven by "factories": modular code units (e.g., ChairFactory, TableFactory) that combine primitive geometry operations into structured templates. While these factories excel at creating objects within their defined categories by piecing together relevant primitive functions in correct structures, they inherently restrict the range of possible assets to these predefined classes. Creating new asset types a;sp requires extending the codebase with new factories, a process that demands intimate knowledge of both the system's architecture and 3D modeling principles. Furthermore, current tools offer limited support for natural language interfaces, a growing necessity in creative and technical workflows. While basic Blender primitives (e.g., boxes, spheres, curves) can be scripted directly, they lack the abstraction and interpolation necessary to produce high-fidelity assets from flexible prompts.

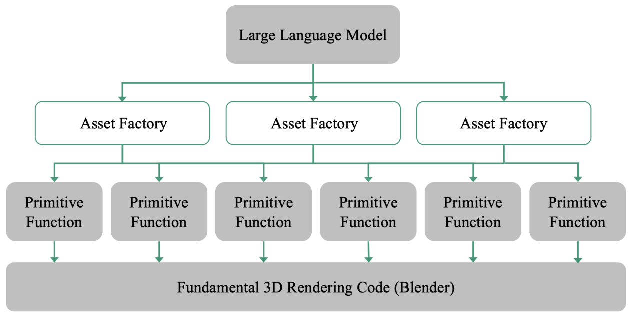


Figure 1. Infinigen Asset Generation Logic Hierarchy

This project therefore aims to bridge this gap by offering two solutions. First, our system enhances Infinigen's procedural generation with prompt-driven control. Second, unlike existing approaches that rely on predefined asset templates (”factories”), our system bypasses these constraints by directly accessing Infinigen’s primitive geometry functions that form the foundation of its procedurally generated 3D objects (see Figure 1). In doing so, we hope to unlock greater flexibility in procedural content generation, allowing users to specify and create virtually any geometric construct through natural language prompts. We also hope to push training computer vision systems’ ability to create precisely specified scenarios without the constraints of predefined asset libraries or the massive expense of physical data collection.

The broader impact of this work lies in its applicability to domains such as robotics and embodied AI, where fine-grained control over environmental variability is critical. For instance, a researcher developing a home navigation algorithm could use our system to generate thousands of procedurally varied room layouts, with configurable furniture arrangements, lighting conditions, and obstacles entirely via text. This capability could accelerate development cycles while improving system robustness through exposure to a much wider variety of scenarios than would be impractical to construct via current methods. The ability to specify rare edge cases or challenging scenarios through simple text prompts would also prove valuable in adjacent use cases. Notably, our system builds upon Infinigen’s native support for exporting indoor scenes to formats compatible with robotics simulators, such as NVIDIA IsaacSim via the Universal Scene Description format, enabling direct integration with IsaacSim for embodied learning and robotics experiments, expanding the utility of our framework.

## 2. Related Work

### 2.1. Text-to-3D Generation

Early efforts in 3D scene synthesis often rely on either extensive 3D model databases or fully procedural generators. Infinigen [3] epitomizes the latter approach: it procedurally creates photorealistic scenes entirely from scratch, with assets generated via mathematical rules rather than fixed external models. More recent text-to-3D methods generate complex scenes directly from prompts, each with different strategies and trade-offs [2–6]. GALA3D [4] uses an LLM to produce an initial coarse layout of a scene and then employs a layout-guided 3D Gaussian splatting representation, which achieves state-of-the-art fidelity and even allows some editing of generated scenes but operates in the space of implicit volumetric representations (Gaussian splats), differing from our use of explicit procedural geometry.

In other approaches, LLMs orchestrate scene creation via intermediate programs or code [5]. Open-Universe Indoor Scene Generation by Aguina-Kang et al. is a prime example: it avoids fixed vocabulary and training on 3D datasets by using an LLM to synthesize a domain-specific scene program to specify and their spatial relationships. However, unlike our approach, the pipeline depends on existing mesh databases and VLM-based search, potentially inheriting biases or limitations of those repositories. However, other systems, such as Scenecraft or 3D-GPT [2, 6], position LLMs as central to the creation of 3D content. In 3D-GPT, the LLM decomposes user instructions into subtasks handled by specialized agents (for conceptualization and actual modeling) and extracts parametric values to drive 3D generation. This has been shown to integrate well with Blender and enable dynamic user-in-the-loop adjustments. We similarly seek the integration of language directives with a powerful 3D engine, but rather than relying on Blender’s general toolkit or external assets, we harness Infinigen’s domain-specific procedural generation to ensure that scenes are synthesized as zero-shot generations.

### 2.2. Indoor Scene and Layout Manipulation.

A growing body of work focuses on the controllable generation of indoor layouts to ensure semantic realism [7–11]. LayoutGPT translates natural language prompts into spatially structured layouts using LLMs and in-context learning, enabling cross-domain use in 2D and 3D scenes with strong adherence to spatial constraints like object counts and relationships [7]. LayoutVLM builds on this by combining differentiable optimization with pre-trained vision language models, treating object poses as learnable variables [8]. This allows the system to iteratively refine scene layouts that are both semantically aligned with prompts and physically plausible. These approaches separate layout planning from final rendering, a philosophy we adopt by distinguishing layout reasoning (handled via LLMs and Infinigen constraints) from geometry creation.

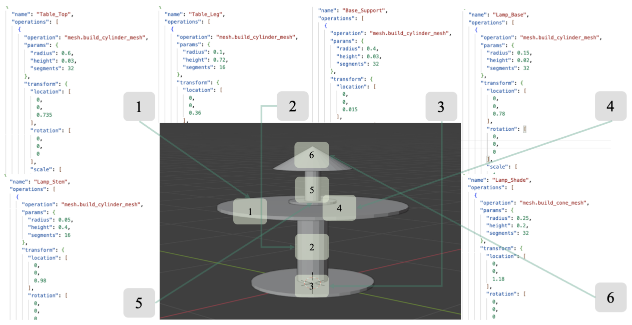
End-to-end indoor scene generators such as Holodeck and ControlRoom3D emphasize holistic synthesis of scene generation. Holodeck uses GPT-4 to plan object presence and relations and populates scenes with retrieved assets from large repositories like Objaverse, solving layout via constraint optimization [9]. ControlRoom3D enables user-sketched proxy rooms and stylistic prompts to generate richly textured, plausible 3D scenes [10]. While these works have produced notable results, we move away from asset-dependent pipelines and towards synthesizing both structure and content from scratch, allowing more flexibility and eliminating reliance on curated model banks. This procedural grounding, combined with natural language guidance, enables our system to generate highly diverse, photorealistic environments while maintaining structural coherence.

## 3. Methodology: Generation

Our approach fundamentally diverges from existing procedural generation systems by introducing natural language controllability and eliminating the intermediary factory layer, thereby establishing a direct mapping from language input to low-level geometric primitives that can define any 3D asset.

We built a system on top of the Infinigen codebase, introducing a natural language interface capable of translating textual descriptions of furniture and indoor scenes into structured 3D assets. Users can provide free-form prompts—ranging from individual furniture objects like “a sofa with a comfortable backrest” to full indoor scenes such as “a dining table with cups on top and four chairs around it.” These prompts are parsed through a multi-stage pipeline that culminates in the generation of a structured JSON specification. The output is then used to generate a Blender-compatible scene by invoking Infinigen’s low-level primitives directly, bypassing the need for template-based factories. Specifically, we utilize Infinigen’s mesh operations (e.g., `build_box_mesh`, `build_plane_mesh`, `build_prism_mesh`), draw functions (e.g., `spin`, `bezier_curve`, `surface_from_func`), and object-level transformations (e.g., `center`, `origin2lowest`, `join_objects`). These built-in utility functions collectively enable fine-grained control over geometry construction without reliance on predefined factory abstractions.

Compared to Blender’s basic primitives, which are limited to fixed shapes like cubes, spheres, and cylinders, Infinigen’s primitives offer significantly greater parametric flexibility, procedural expressiveness, and composability. For example, while Blender’s primitives are generally static and require manual scripting for custom forms, Infinigen’s primitives support high-level configuration of geometric structure—such as controlled curvature, tilts, and procedural surface variation—which is crucial for zero-shot synthesis of complex assets from natural language.



```
"name": "Table_Top",
"operations": [
  {
    "operation": "mesh.build_cylinder_mesh",
    "params": {
      "radius": 0.6,
      "height": 0.03,
      "segments": 32
    },
    "transform": {
      "location": [
        0,
        0,
        0.735
      ],
      "rotation": [
        0,
        0,
        0
      ],
      "scale": [
```

```
"name": "Table_Leg",
"operations": [
  {
    "operation": "mesh.build_cylinder_mesh",
    "params": {
      "radius": 0.1,
      "height": 0.72,
      "segments": 16
    },
    "transform": {
      "location": [
        0,
        0,
        0.36
      ],
```

```
"name": "Base_Support",
"operations": [
  {
    "operation": "mesh.build_cylinder_mesh",
    "params": {
      "radius": 0.4,
      "height": 0.03,
      "segments": 32
    },
    "transform": {
      "location": [
        0,
        0,
        0.015
      ],
```

```
"name": "Lamp_Base",
"operations": [
  {
    "operation": "mesh.build_cylinder_mesh",
    "params": {
      "radius": 0.15,
      "height": 0.02,
      "segments": 32
    },
    "transform": {
      "location": [
        0,
        0,
        0.78
      ],
      "rotation": [
        0,
        0,
        0
      ],
      "scale": [
```

```
"name": "Lamp_Stem",
"operations": [
  {
    "operation": "mesh.build_cylinder_mesh",
    "params": {
      "radius": 0.05,
      "height": 0.4,
      "segments": 16
    },
    "transform": {
      "location": [
        0,
        0,
        0.98
      ],
      "rotation": [
        0,
        0,
        0
```

```
"name": "Lamp_Shade",
"operations": [
  {
    "operation": "mesh.build_cone_mesh",
    "params": {
      "radius": 0.25,
      "height": 0.2,
      "segments": 32
    },
    "transform": {
      "location": [
        0,
        0,
        1.18
      ],
      "rotation": [
        0,
        0,
        0
```

Figure 2. Generated JSON File Mapped to Asset

### 3.1. Full Agent Breakdown

At its core, the ”Full Agent” employs a number of specialized agents: (1) The “Furniture Classifier” agent analyzes input prompts to validate furniture or indoor scene descriptions. (2) The “Semantic Decomposer” agent systematically breaks down objects into component structures with precise geometric and spatial specifications. (3) The “Primitive Calls Generator” transforms these abstract components into concrete Infinigen primitives. It is supported by a “Context Provider” agent that maintains a comprehensive database of factory examples within Infinigen. (4) A “Validator” agent checks and corrects any anomalies in component connections and spatial relationships. All data exchanged between agents is serialized in a consistent JSON-based interface. Each component’s representation includes the primitive type, geometric parameters, transformations, and semantic labels. Finally, this structured JSON is output in a Blender file that calls the corresponding Infinigen primitives. All LLM-based reasoning across agents is performed using OpenAI’s GPT-4o model for its improved spatial reasoning ability.

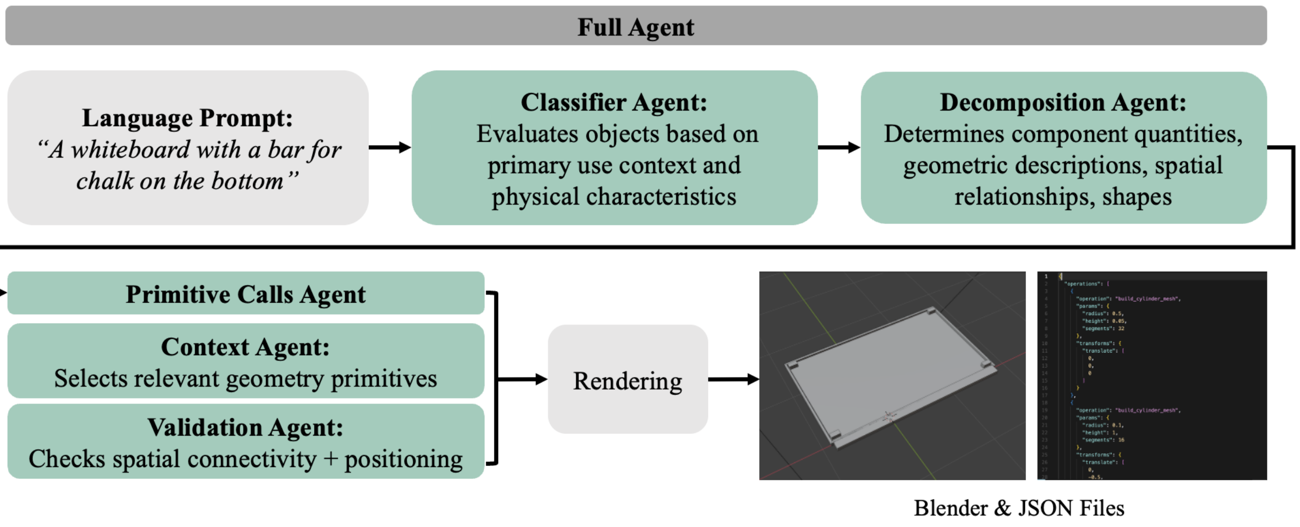
### 3.2. Classifier Agent

The “Furniture Classifier” agent employs a classification approach that enforces a three-dimensional analysis framework: context evaluation (indoor vs. outdoor usage), physical characteristic assessment, and spatial relationship analysis (integration with indoor environments). The classifier implements a binary classification schema returning either ”pass” or ”does not pass” with accompanying explanation, filtering out irrelevant prompts.

### 3.3. Decomposer Agent

The “Semantic Decomposer agent” translates natural language furniture descriptions into structured, component-based representations. Unlike existing systems that treat furniture as single, fixed assets, this agent modularizes each object into discrete parts—such as seats, legs, and backrests for chairs—each annotated with geometric, spatial, and relational metadata. This enables editable, prompt-aligned model generation and ensures all user-specified elements are represented with correct quantity, naming, and symmetry.

This agent contains a curvature specification system, classifying each component using six curvature types (e.g., straight, ergonomic curve, spiral) and three profile intensities (gentle, moderate, pronounced), along with directionality (e.g., vertical, outward). This allows the system to capture nuanced geometric intentions, such as S-curved backrests or subtly arched legs, which downstream agents use to select the appropriate primitives (e.g., Bezier curves for complex shapes).


Full Agent
Language Prompt: "A whiteboard with a bar for chalk on the bottom"
Classifier Agent: Evaluates objects based on primary use context and physical characteristics
Decomposition Agent: Determines component quantities, geometric descriptions, spatial relationships, shapes
Primitive Calls Agent
Context Agent: Selects relevant geometry primitives
Validation Agent: Checks spatial connectivity + positioning
Rendering
Blender & JSON Files

```
{
  "operations": [
    {
      "operation": "build_cylinder_mesh",
      "params": {
        "radius": 0.5,
        "height": 0.05,
        "segments": 32
      },
      "transforms": {
        "translate": [
          0,
          0,
          0
        ]
      }
    },
    {
      "operation": "build_cylinder_mesh",
      "params": {
        "radius": 0.1,
        "height": 1,
        "segments": 16
      },
      "transforms": {
        "translate": [
          0,
          -0.5,
```

Figure 3. Full Generation Agent Workflow

Each output is a clean JSON object containing component names, quantities, geometric dimensions, spatial relationships, and connection maps. Identical components are grouped and marked for replication, while nested and connected parts are clearly defined. This structured output forms the semantic foundation for the "Primitive Calls Generator" and "Validator" agents.

## 3.4. Primitive Calls Generator Agent

The "Primitive Calls Generator" agent serves as a translation layer between semantic component specifications and executable 3D geometry. It leverages a context-aware prompting system, drawing on a rich library of factory examples provided by the "Context Provider", to guide the selection of appropriate primitive operations. This contextual grounding allows the agent to generalize across familiar construction patterns while accommodating novel configurations. A central feature of its design is a structurally ordered generation protocol: components are assembled in a bottom-up fashion, beginning with ground-contact elements, followed by structurally dependent parts, and concluding with decorative additions.

To ensure physical plausibility, the agent operates under a tightly constrained specification framework. It restricts geometry construction to a curated set of operations across three domains: mesh generation for standard solids, drawing functions for curved or free-form elements, and object-level manipulations for compositional control. Each function is governed by predefined usage constraints, preventing invalid parameter combinations and preserving structural consistency. Curved components, such as legs or supports, are handled through specialized Bezier-based generation, while cushion-like forms use scaled spherical primitives for accurate soft-body representation.

Component replication is governed by a consistency protocol that guarantees geometric uniformity across instances. When multiple copies of a component are required, such as chair legs or drawer fronts, the system ensures that only spatial transformations are different, while all other parameters remain identical. Naming conventions and spatial arrangements follow strict rules as well.

Dimensional relationships are resolved through an embedded calculation engine that computes precise spatial dependencies between connected parts. Rules governing height alignment, span coverage, nesting clearances, and stacking ensure components meet seamlessly at shared boundaries. These calculations rely on vector-based distance measurements and enforce exact junction matching, which is particularly critical for components like crossbars or support beams. The final output gives a rigorous JSON schema, encapsulating all operations, parameters, and transformations, ready for downstream validation and rendering in Blender.

## 3.5. Validator Agent

A key innovation of this system is the "Component Validator" agent. While many spatial and geometric rules are embedded in the prompt in the previous step, we discovered that the LLM struggles to follow these instructions, often selecting inappropriate parameters. As a result, generated structures are often structurally invalid, with common issues such as seats or tabletops floating above their supports, or base components failing to make ground contact (Figure 3, Pre-validation). To address this problem, we implement a four-phase correction pipeline composed of explicit, rule-based procedures: (1) connection map generation, (2) ground contact enforcement, (3) horizontal alignment, and (4) connection integrity checking through rule-based proximity and priority logic.

The first stage constructs a connection map, where the LLM interprets a list of component names and infers which parts should be physically connected. If the LLM fails or produces an invalid format, the system falls back to default heuristics (e.g. connecting a tabletop to all leg components). Next, the validator enforces ground contact by detecting base components such as legs or frames, calculating their lowest Z-coordinate, and adjusting all components vertically to ensure the base sits at Z=0. This maintains the relative spatial relationships between components while aligning the structure with a virtual ground plane.

For chair-like structures, the validator performs horizontal alignment to ensure legs are positioned directly under the seat. Legs that deviate beyond a fixed XY threshold are translated to align with the bottom of the seat for structural realism. Then, it executes vertical snapping, raising or lowering the seat or tabletop so that its bottom surface is aligned precisely with the highest leg endpoint. Finally, the system performs connection validation, aligning remaining parts (e.g., backrests, arms) by analyzing their geometric anchor points and snapping them to their intended targets if they fall outside a proximity threshold. It handles a variety of primitive types, including boxes, cylinders, spheres, and Bezier curves, by using type-specific logic to determine connection points and priorities. These corrections are reflected in the component's parameters, which are passed forward as JSON objects.

## 4. Dataset

This project diverges from conventional machine learning workflows in that it does not utilize a traditional dataset in the supervised learning sense. Instead, our approach leverages several key resources that form the knowledge foundation of our system.

Central to our work is a comprehensive documentation of Infinigen's primitive geometry functions. We meticulously analyzed the codebase to extract and document functions from critical modules, including the fundamental building blocks that Infinigen uses to construct its complex procedural assets: primitives that are typically hidden behind the abstraction of factory classes. For each primitive, we created detailed documentation capturing its function, parameter types, default values, constraints, and expected behaviors. This documentation serves as a knowledge base for our language model to understand the capabilities and limitations of each primitive function. Our data processing approach focused on structuring the primitive function documentation in a way that facilitates accurate mapping between natural language concepts and appropriate geometric operations. The resulting knowledge base enables our system to "understand" both the capabilities of Infinigen's primitive functions and the natural language descriptions provided by users.

## 5. Experiment and Results

### 5.1. Experiments

To investigate the fidelity and flexibility of our generation pipeline, we wrote a suite of 150 representative indoor prompts encompassing both common and edge-case scenarios. Examples range from simple, canonical items (e.g., "a four-legged wooden table" or "a small office desk with two drawers") to challenging, open-ended prompts (e.g., "a chair with a comfortable round seat and three slanted legs" or "a modern shelving unit that twists around a vertical pole"). For each prompt, we generated outputs using four agent variants:

- Full Agent (with Validation): Utilizes the complete pipeline, including object classification, prompt decomposition, context, access to complex geometry primitives, and a post-generation validation layer that performs semantic and geometric checks.
- Full Agent (no Validation): Identical to the above but omits the post-hoc validation step. This comparison isolates the contribution of the validation step to generation fidelity.
- No-Decomposition Agent: Skips the decomposition step, meaning the model directly proceeds to scene synthesis without intermediate representations such as part counts, shape symmetry, or spatial relationships. This variant reveals how critical decomposition is to achieving structured results.
- Direct Primitives Agent: Bypasses higher-level primitive calls and instead generates assets using raw Bmesh operations (operations that allow for the manipulation of mesh data, such as vertices, edges, and faces). This removes the abstraction layer that translates decomposed object components into semantically meaningful Blender primitives, evaluating whether raw mesh control can match higher-level procedural generation.

Across all agents, we performed iterative testing and regeneration as necessary, simulating a realistic development workflow and surfacing where breakdowns most often occurred. Representative output scenes from these experiments are shown in Figure 3.

To further ensure prompt diversity and model robustness, we tested prompts across multiple asset categories (e.g., seating, storage) and small scene scales (single-object vs. room-scale scenes). This design enabled us to analyze not only the micro-level assembly of discrete components but also macro-level spatial reasoning, such as orientation, ground contact, and the preservation of object-object relationships.

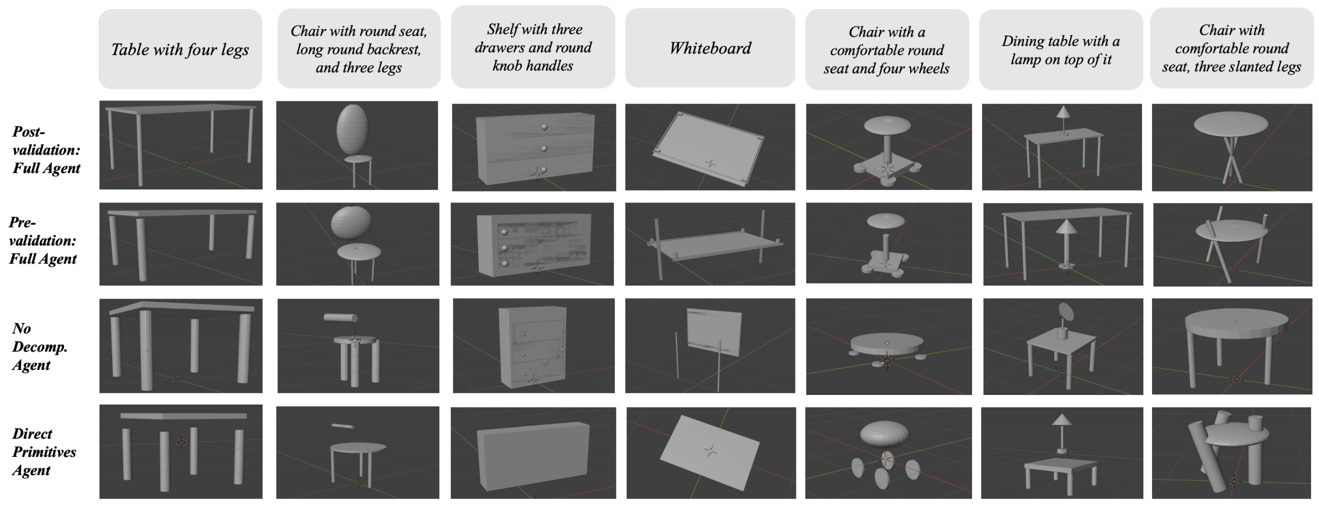


Table with four legs
Chair with round seat, long round backrest, and three legs
Shelf with three drawers and round knob handles
Whiteboard
Chair with a comfortable round seat and four wheels
Dining table with a lamp on top of it
Chair with comfortable round seat, three slanted legs
Post-validation: Full Agent
Pre-validation: Full Agent
No Decomp. Agent
Direct Primitives Agent

Figure 4. Assets Generated Post-Validation, Pre-Validation, Without Decomposition, Only Direct Primitives.

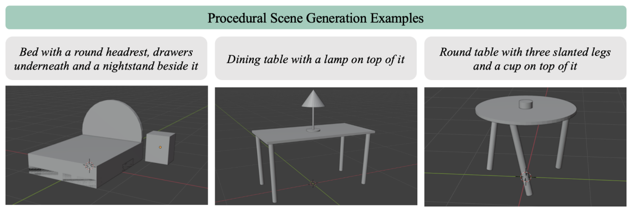

Procedural Scene Generation Examples
Bed with a round headrest, drawers underneath and a nightstand beside it
Dining table with a lamp on top of it
Round table with three slanted legs and a cup on top of it

Figure 5. Procedurally Generated Scene Examples

## 5.2. Evaluation

To assess the quality of the generated outputs, we employed a qualitative evaluation protocol designed to capture perceptual and structural aspects of realism not easily captured by existing quantitative metrics. Drawing from best practices in prior 3D generation literature [8, 12], we asked ten students with prior exposure to 3D modeling or graphics coursework to evaluate each generated asset or scene on three core rubrics: structural coherence (plausibility of physical composition and support), visual realism (object continuity and consistency), and prompt alignment (similarity to semantic and spatial directives in the input prompt).

The evaluation revealed that the full agent, particularly when combined with post-generation validation, consistently outperformed all other configurations. Participants noted higher consistency in geometric alignment, more accurate replication of object symmetry, and fewer violations of physical realism (e.g., floating objects or incorrectly intersecting parts). Multi-object scenes especially benefitted from the validation layer, as it corrected common errors like inconsistent orientation. However, feedback also indicated that certain complex scenes — particularly unconventional prompts — often lacked plausible object scaling and spatial composition, demonstrating that the system's understanding of spatial relationships still has room for improvement in edge cases.

## 5.3. Discussion

The system excelled at handling simple geometric furniture designs, particularly those with clear hierarchical structures like chairs and tables. Its modular architecture provides a clear separation of concerns, handling different mesh types through distinct pathways while maintaining a coherent validation pipeline. This approach allows for reliable automated corrections, particularly in basic spatial relationships such as ground contact and component connections, making it highly effective for standard furniture assembly tasks. Furthermore, across the evaluated prompts, iterative refinement through validation yielded consistent improvements, particularly in connectivity of the objects, alignment of duplicated parts, and adherence to spatial instructions. The full agent significantly outperformed both baselines in producing geometrically consistent and semantically meaningful outputs, especially in multi-object scenes (Figure 4).

When faced with complex geometries, particularly those involving unconventional base shapes or asymmetric components, the system's limitations became apparent. The handling of curves presented a particular challenge, as the current implementation struggles with rotations, orientations, and complex curve intersections. These issues stem from the current reliance on hard-coded validation rules for primitive mesh types (e.g., checking axis alignment or bounding box overlaps), which fail to generalize to novel mesh types. When we attempted to substitute these rules with LLM-generated logic, spatial reasoning shortcomings became evident. LLMs often failed to predict accurate bounds, orientations, or attachment points for irregular components. Thus, it is clear that substantial improvements can be made to our baseline system.

## 6. Conclusion and Future Work

This project successfully accomplished its goal of designing a novel system that uses sophisticated geometry primitives to procedurally generate assets and scenes, grounded in the previous work of Infinigen [3]. By integrating natural language prompting with granular geometry construction, our system demonstrates that complex scenes can be synthesized from scratch without relying on pre-built asset libraries. The full agent deployed a multi-step pipeline to classify, decompose, and contextualize a natural prompt before carefully rendering the request based on a selection of unlimited geometry options and validating its semantic plausibility. Compared to existing systems that either retrieve pre-generated assets, our pipeline offers modular control, infinite asset variability, and strong alignment with user intent. However, we also recognize the limitations of this tool, as there is much work to be done in enriching its ability to zero-shot more unconventional or complicated prompts.

Future work should focus on developing more sophisticated algorithms for bounds detection and curve analysis, while improving the LLM's spatial reasoning capabilities through specialized training. The system would benefit from enhanced validation robustness, particularly in handling non-standard bases and complex component hierarchies. These improvements would be crucial for expanding the system's capabilities beyond simple furniture assembly to more complex geometric relationships and curved components. In addition, fleshing out a more robust surface modeling stack (algorithms and primitives for generating realistic curvature, inorganic shapes, etc) is necessary to expand both the scope of the generated assets beyond indoor objects and enhance their realism. Finally, the system would benefit from an interactive feedback loop that allows users to guide refinement, correct misalignments, or iterate on generated outputs.

Although there are several key areas for improvement, we ultimately envision this framework as an enabling infrastructure for scalable, flexible, and user-controllable 3D generation, where creative expression and functional specification are enabled via natural language.

## 7. Contributions and Acknowledgments

Eyrin Kim and Michelle Lau collaboratively designed the methodology, implemented the agents to generate the scenes, conducted the evaluation, and wrote the paper. These authors contributed equally to the work. Fan-Yun Sun advised and mentored on the project.

The authors made use of public code, including the following repositories:

- Infinigen: https://github.com/princeton-vl/infinigen
- 3D GPT: https://github.com/Chuny1/3DGPT

## References

[1] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In International conference on machine learning, pages 8748–8763. PMLR, 2021.

[2] Peng, X., et al. (2023). "SceneCraft: Text-driven 3D Scene Generation with Retrieval-Augmented Code Generation." arXiv preprint arXiv:2302.01763. 1, 2

[3] Li, Z., et al. (2023). "Infinigen: Infinite Photorealistic Worlds using Procedural Generation." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 1, 2, 7

[4] Zhou, X., Ran, X., Xiong, Y., He, J., Lin, Z., Wang, Y., ... Yang, M. H. (2024). Gala3d: Towards text-to-3d complex scene generation via layout-guided generative gaussian splatting. arXiv preprint arXiv:2402.07207. 2

[5] Aguina-Kang, R., Gumin, M., Han, D. H., Morris, S., Yoo, S. J., Ganeshan, A., ... Ritchie, D. (2024). Open-universe indoor scene generation using llm program synthesis and uncurated object databases. arXiv preprint arXiv:2403.09675. 2

[6] Sun, C., Han, J., Deng, W., Wang, X., Qin, Z., Gould, S. (2023). 3d-gpt: Procedural 3d modeling with large language models. arXiv preprint arXiv:2310.12945. 2

[7] Feng, W., Zhu, W., Fu, T. J., Jampani, V., Akula, A., He, X., ... Wang, W. Y. (2023). Layoutgpt: Compositional visual planning and generation with large language models. Advances in Neural Information Processing Systems, 36, 18225-18250. 2

[8] Sun, F. Y., Liu, W., Gu, S., Lim, D., Bhat, G., Tombari, F., ... Wu, J. (2024). LayoutVLM: Differentiable Optimization of 3D Layout via Vision-Language Models. arXiv preprint arXiv:2412.02193. 2, 6

[9] Yang, Y., Sun, F. Y., Weihs, L., VanderBilt, E., Herrasti, A., Han, W., ... Clark, C. (2024). Holodeck: Language guided generation of 3d embodied ai environments. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 16227-16237). 2

[10] Schult, J., Tsai, S., Höllein, L., Wu, B., Wang, J., Ma, C. Y., ... Hou, J. (2024). Controlroom3d: Room generation using semantic proxy rooms. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 6201-6210). 2

[11] Epstein, D., Poole, B., Mildenhall, B., Efros, A. A., Holynski, A. (2024). Disentangled 3d scene generation with layout learning. arXiv preprint arXiv:2402.16936. 2

[12] Tong Wu, Guandao Yang, Zhibing Li, Kai Zhang, Ziwei Liu, Leonidas Guibas, Dahua Lin, and Gordon Wetzstein. Gpt-4v (ision) is a human-aligned evaluator for text-to-3d generation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 22227–22238, 2024. 6