

# 3D Scene Generation and Editing using Foundational Models and Geometric Algebra

*Dimitrios Angelis*

Thesis submitted in partial fulfillment of the requirements for the  
*Masters' of Science degree in Computer Science and Engineering*

University of Crete  
School of Sciences and Engineering  
Computer Science Department  
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *George Papagiannakis*

---

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).



UNIVERSITY OF CRETE  
COMPUTER SCIENCE DEPARTMENT

**3D Scene Generation and Editing using Foundational Models and  
Geometric Algebra**

Thesis submitted by  
**Dimitrios Angelis**  
in partial fulfillment of the requirements for the  
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: \_\_\_\_\_  
Dimitrios Angelis

Committee approvals: \_\_\_\_\_  
George Papagiannakis  
Professor, Thesis Supervisor

\_\_\_\_\_  
Yannis Tzitzikas  
Professor, Committee Member

\_\_\_\_\_  
Anastasios Roussos  
Principal Researcher, Committee Member

Departmental approval: \_\_\_\_\_  
Polyvios Pratikakis  
Associate Professor, Director of Graduate Studies

Heraklion, June 2024



# 3D Scene Generation and Editing using Foundational Models and Geometric Algebra

## Abstract

In the realm of Embodied AI, the creation of 3D simulated environments holds paramount significance, yet it often demands specialized expertise and substantial manual labor, consequently limiting their diversity and expansiveness.

In this thesis, we introduce a novel framework designed to address this limitation by facilitating the fully automated generation of 3D environments tailored to user-supplied prompts. Our framework automates scene generation and exhibits versatility in crafting diverse scenes, adjusting designs to various styles, and comprehending the semantics of intricate queries. Central to our approach is the utilization of a large language model (LLM), which imbues the framework with common-sense knowledge to envision plausible scene configurations. Additionally, we harness a vast collection of 3D assets sourced from Objaverse to populate scenes with a rich array of objects. We further enhance the framework by integrating a sophisticated agent capable of providing feedback to the generation process. This agent, powered by Multimodal Models like GPT-4 Vision, operates as a feedback agent, guiding the generation towards desired outcomes. Furthermore, we harness the capabilities of Retrieval Augmented Generation (RAG) to enrich the generation process, and incorporate the use of reference images, leveraging the advanced visual understanding of GPT-4 Vision.

User evaluations indicate a strong preference for our approach, with 75% of users favoring scenes generated using the feedback agent, 55.6% preferring scenes generated using RAG, and 83.3% agreeing that there is a resemblance of the generated scene to the referenced image. In comparison with the state of the art, our implementation is faster and more modular, enhancing user experience and system efficiency.

Additionally, this thesis introduces a novel algorithm that integrates Large Language Models (LLMs) with Conformal Geometric Algebra (CGA) to revolutionize controllable 3D scene editing, particularly for object repositioning tasks. Conventional methods typically suffer from reliance on large training datasets or lack a formalized language for precise edits. Utilizing CGA as a robust formal language, our framework precisely models spatial transformations necessary for accurate object repositioning. Leveraging the zero-shot learning capabilities of pre-trained LLMs, our framework translates natural language instructions into CGA operations, facilitating exact spatial transformations within 3D scenes without the need for specialized pre-training.

To accurately assess the impact of CGA, we benchmark against robust Euclidean-based baselines, evaluating both latency and accuracy. Comparative performance evaluations indicate that our framework significantly reduces LLM response times by 16% and boosts success rates by 9.6% on average compared to traditional methods.

These advancements underscore our framework's potential to democratize 3D scene generation and editing, enhancing accessibility and fostering innovation across sectors such as education, entertainment, and virtual reality.

# Δημιουργία και Επεξεργασία Τρισδιάστατων Σκηνών χρησιμοποιώντας Μεγάλα Μοντέλα Όρασης-Γλώσσας και Γεωμετρική Άλγεβρα

## Περίληψη

Στον κόσμο του Embodied AI, η δημιουργία 3D simulated environments κρατά πρωτεύουσα σημασία, ωστόσο συχνά απαιτεί εξειδικευμένη εμπειρία και σημαντικό χειρονακτικό έργο, περιορίζοντας συνεπώς την ποικιλία και την εκτεταμένη χρήση τους.

Σε αυτήν τη διατριβή, πρώτον, παρουσιάζουμε ένα νέο σύστημα που σχεδιάστηκε για να αντιμετωπίσει αυτό το περιορισμό. Το σύστημα αυτό διευκολύνει την πλήρως αυτοματοποιημένη δημιουργία 3D περιβαλλόντων που προσαρμόζονται σε παραμέτρους που καθορίζει ο χρήστης. Το σύστημά μας επιδεικνύει ευελιξία στη δημιουργία ποικίλων σκηνών. Κεντρικό στην προσέγγισή μας είναι η χρήση ενός Μεγάλου Γλωσσικού Μοντέλου (LLM), το οποίο εμπνέει το σύστημα με κοινή λογική γνώση για να φανταστεί πιθανές διαμορφώσεις σκηνών. Επιπλέον, αξιοποιούμε μια τεράστια συλλογή 3D μοντέλων για να γεμίσουμε τις σκηνές με μια πλούσια γκάμα αντικειμένων. Επιπλέον, ενισχύουμε το σύστημα μας ενσωματώνοντας έναν πράκτορα ικανό να παρέχει αυτο-επιβλεπόμενα σχόλια στη διαδικασία δημιουργίας. Αυτός ο πράκτορας, κινούμενος από το GPT-4V, λειτουργεί ως rewarding agent, καθοδηγώντας τη δημιουργία προς τα επιθυμητά αποτελέσματα. Επίσης, εκμεταλλευόμαστε τις δυνατότητες του RAG (Retrieval Augmented Generation) για να εμπλουτίσουμε περαιτέρω τη διαδικασία δημιουργίας. Επιπρόσθετα, ενσωματώνουμε τη χρήση μιας εικόνας αναφοράς στη συνολική διαδικασία, αξιοποιώντας την προηγμένη οπτική κατανόηση του GPT-4V.

Επιπλέον, προτείνουμε έναν καινοτόμο αλγόριθμο που συνδυάζει Μεγάλα Γλωσσικά Μοντέλα (LLMs) με την Σύμμορφη Γεωμετρική Άλγεβρα (CGA) για την επεξεργασία 3D σκηνών, ιδίως για εργασίες αναδιάταξης αντικειμένων. Οι συμβατικές μέθοδοι υποφέρουν συνήθως από την εξάρτηση από μεγάλα σύνολα δεδομένων εκπαίδευσης ή την έλλειψη μιας τυποποιημένης γλώσσας για ακριβείς επεξεργασίες. Χρησιμοποιώντας το (CGA) ως ένα τυποποιημένο γλωσσικό σύστημα, το σύστημά μας μοντελοποιεί με ακρίβεια τις χωρικές μετατοπίσεις που απαιτούνται για ακριβή αναδιάταξη αντικειμένων. Αξιοποιώντας τις ικανότητες μηδενικής εκπαίδευσης των προεκπαιδευμένων LLMs, το σύστημά μας μεταφράζει φυσικές γλωσσικές οδηγίες σε πράξεις (CGA) χωρίς την ανάγκη για εξειδικευμένη προεκπαίδευση. Για την ακριβή αξιολόγηση της επίδρασης του (CGA), κάνουμε μια σύγκριση με υλοποιήσεις βασισμένες στην ευκλείδεια γεωμετρία, αξιολογώντας τόσο την καθυστέρηση όσο και την ακρίβεια. Οι συγκρίσεις δείχνουν ότι το σύστημά μας μειώνει σημαντικά τους χρόνους απόκρισης των LLM κατά 16% και αυξάνει τις επιτυχίες κατά 9,6% κατά μέσο όρο σε σύγκριση με τις παραδοσιακές μεθόδους.





## Acknowledgments

I would like to express my deepest gratitude to Prof. George Papagiannakis for his invaluable guidance, mentorship, and unwavering support throughout this journey. His encouragement, and constructive feedback have been instrumental in shaping the direction and quality of this work.

I am also indebted to Dr. Prodromos Kolyvakis, for his continuous support, insightful discussions, and dedication to fostering my growth as a researcher. His expertise, mentorship and encouragement have been invaluable in navigating the challenges of this thesis.

I would like to extend my sincere appreciation to Manos Kamarianakis, whose expertise and insights have greatly enriched this work.

I am grateful to my friends and colleagues for their encouragement, support, and camaraderie throughout this endeavor. Their friendship, discussions, and shared experiences have provided much-needed motivation and inspiration during challenging times.

Finally, I want to thank my family for their unconditional love, encouragement, and understanding. Their unwavering support and belief in me have been the bedrock of my journey, and I am profoundly grateful for their presence in my life.



*Explore, Experiment, Learn*



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>State of the Art</b>	<b>5</b>
2.1	Large Language Model for Scene Design . . . . .	5
2.2	Text-driven 3D Generation . . . . .	6
2.3	Agent-Driven Object Rearrangement . . . . .	7
2.4	Machine Learning Applications of Geometric Algebra . . . . .	8
<b>3</b>	<b>3D Scene Generation</b>	<b>11</b>
3.1	Prompt Design . . . . .	11
3.2	Floor & Wall Module . . . . .	12
3.2.1	Output Format . . . . .	12
3.2.2	Material Selection . . . . .	13
3.2.3	Wall Height . . . . .	13
3.3	Doorway & Window Module . . . . .	13
3.3.1	Door Selection . . . . .	14
3.3.2	Window Selection . . . . .	14
3.4	Object Selection Module . . . . .	15
3.4.1	Object Selection Module . . . . .	15
3.4.2	Leveraging Objaverse Assets . . . . .	15
3.5	Constraint-based Layout Design Module . . . . .	17
3.5.1	Spatial Relational Constraints . . . . .	17
3.5.2	Constraint Satisfaction . . . . .	18
3.6	Feedback Module . . . . .	18
3.6.1	Input Data . . . . .	19
3.6.2	Processing and Analysis . . . . .	19
3.6.2.1	Quality Assessment . . . . .	19
3.6.2.2	Scoring Mechanism . . . . .	19
3.6.3	Feedback Generation . . . . .	19
3.6.4	Iterative Improvement . . . . .	20
3.7	Real-life Image Integration . . . . .	20
3.7.1	Input Data . . . . .	20
3.7.2	Feedback Generation . . . . .	21

3.8	Retrieval Augmented Generation (RAG) Integration . . . . .	21
3.8.1	Database of Previous Generations . . . . .	22
3.8.2	Query Handling . . . . .	22
3.8.2.1	Query Analysis . . . . .	22
3.8.2.2	Reference Retrieval . . . . .	22
3.8.3	Incorporation into Scene Generation . . . . .	22
3.8.3.1	Guided Generation . . . . .	22
3.8.4	Evaluation and Benefits . . . . .	23
<b>4</b>	<b>3D Scene Editing</b>	<b>25</b>
4.1	Conformal Geometric Algebra . . . . .	26
4.2	LLM Processing &CGA Formulation . . . . .	28
4.2.1	Implementation Details . . . . .	29
4.3	Collision Detection Module . . . . .	29
4.4	Retrieval Augmented Generation &Caching Mechanism Integration	30
4.4.1	Retrieval Augmented Generation . . . . .	30
4.4.2	Caching Mechanism . . . . .	30
4.5	Universal Scene Description . . . . .	31
4.5.1	Benefits of Using USD for Dataset Management . . . . .	31
4.5.2	Implementation Details . . . . .	32
<b>5</b>	<b>Evaluation</b>	<b>33</b>
5.1	System Configuration . . . . .	33
5.2	System-based evaluation . . . . .	33
5.2.1	Execution Speed . . . . .	33
5.2.2	Memory Efficiency . . . . .	33
5.2.3	Modularity and Extendability . . . . .	34
5.2.4	3D Models Dataset Enhancement . . . . .	34
5.2.5	Advanced Features and Rendering Quality . . . . .	35
5.2.6	Summary . . . . .	35
5.3	3D Scene Generation . . . . .	35
5.3.1	Experimental Setup . . . . .	36
5.3.2	Performance Metrics . . . . .	36
5.3.2.1	RAG and Feedback Module . . . . .	36
5.3.2.2	Reference Image . . . . .	36
5.3.3	Results &Discussion . . . . .	36
5.3.3.1	Feedback Module . . . . .	37
5.3.3.2	Retrieval-Augmented Generation (RAG) . . . . .	37
5.3.3.3	Reference Image . . . . .	38
5.3.3.4	Discussion . . . . .	39
5.3.3.5	Examples of Generated Scenes . . . . .	40
5.4	3D Scene Editing . . . . .	45
5.4.1	Experimental Setup . . . . .	45
5.4.2	Benchmarking . . . . .	46

5.4.2.1	Baseline Systems . . . . .	46
5.4.2.2	Performance Metrics . . . . .	47
5.4.2.3	Results & Discussion . . . . .	47
5.4.3	Scene Editing Examples . . . . .	52
<b>6</b>	<b>Conclusion</b>	<b>57</b>
6.1	Summary . . . . .	57
6.2	Limitations & Future Work . . . . .	58
6.2.1	3D Scene Generation . . . . .	58
6.2.2	3D Scene Editing . . . . .	58
	<b>Bibliography</b>	<b>61</b>
<b>A</b>	<b>Appendices</b>	<b>73</b>
A.1	Conformal Geometric Algebra . . . . .	73
A.1.1	Vector Objects of $\mathbb{R}_{4,1}$ . . . . .	73
A.1.2	Products in $\mathbb{R}_{4,1}$ . . . . .	74
A.1.3	Dual Objects . . . . .	75
A.1.4	Rotations, Translations and Dilations . . . . .	76
A.1.5	Interpolation of Multivectors . . . . .	77
A.2	3D Scene Generation . . . . .	77
A.2.1	Prompts . . . . .	77
A.3	3D Scene Editing . . . . .	85
A.3.1	Ours, Euclidean & Omniverse Prompts . . . . .	85





# List of Tables

2.1	Comparison of main features across relevant papers with our system.	10
5.1	Comparison of module average execution time between our system and Holodeck . . . . .	34
5.2	Results across three different scenes for each prompt and for each system. . . . .	54
5.3	Results across three different scenes for each prompt and for each system. . . . .	55



# List of Figures

2.1	Top: Scene generated using Holodeck. Bottom: Scene generated using LayoutGPT. . . . .	6
2.2	Scene generated using Text2Room and Neural Descriptor Fields. . . . .	7
2.3	Object Rearrangement through code generation. Left: LLMR. Right: Cook2LTL . . . . .	8
3.1	Floor & Wall Module Overview . . . . .	12
3.2	<b>Material Customizability.</b> Our framework can select appropriate floor and wall materials to make the scenes more realistic. . . . .	13
3.3	Doorway & Window Module Overview . . . . .	14
3.4	Object Selection Module Overview . . . . .	15
3.5	<b>Objects Customizability.</b> Our framework can select and place appropriate objects based on the input, such as “living room” and “wine cellar”. . . . .	16
3.6	Constraint Layout Design Module Overview . . . . .	17
3.7	Feedback Module Overview . . . . .	18
3.8	Example of real-life images used as references for scene generation. . . . .	21
4.1	Scene editing component overview: It generates a CGA transformation after <i>templating</i> the query and adds a $\delta$ upward translation for a hypothetical Z-axis collision. . . . .	25
4.2	Examples of conformal geometric algebra equations generated for the queries <i>Move X0 to the right</i> and <i>Move X0 on top of X1</i> . . . . .	26
5.1	Comparison of scene snapshots: Our system (left) within TDW and Holodeck (right) within AI2THOR. . . . .	35
5.2	Results from participants’ choice between Holodeck and our generated scenes based on feedback. . . . .	37
5.3	Results from participants’ choice between Holodeck and our RAG-generated scenes. . . . .	38
5.4	Results of participants’ agreement on resemblance between reference images and generated scenes. . . . .	39
5.5	On the left the scene was generated using the query <i>a living room</i> while on the right the prompt <i>a wine cellar</i> . . . . .	40

5.6	On the left the scene was generated using the query <i>an operating room</i> while on the right the prompt <i>a medieval dungeon</i> . . . . .	41
5.7	On the left the scene was generated using the query <i>a library room</i> while on the right the prompt <i>a warehouse</i> . . . . .	41
5.8	On the left we can see the picture of a living room provided as reference for the generation and on the right the final generated scene. . . . .	42
5.9	Two scenes generated using RAG. On the left the query is <i>A living room</i> and on the right <i>A doctor's office</i> . . . . .	43
5.10	On the left we can see the picture of the initially generated living room and on the right the final generated living room based on the feedback generated from our Feedback Module. . . . .	43
5.11	The left chart compares the average success rates, and the right chart compares the average LLM response times for three different systems—Omniverse, Euclidean, and CGA—across five categories of queries: Simple Queries, Compositional Queries, Fuzzy Queries, Compositional Fuzzy Queries, and Hard Queries. Success rates are measured on a scale from 0.0 to 1.0. Horizontal dashed lines indicate the overall mean recall and response times for each system. . . . .	48
5.12	Query resolution performance for different prompts—Omniverse, Euclidean, and CGA—across five query categories: Simple, Compositional, Fuzzy, Compositional Fuzzy, and Hard. Success rate is assessed for each templated query over ten different scenes, with five query variations per scene. . . . .	49
A.1	Prompt used for the floor module. . . . .	78
A.2	Prompt used for the wall module. . . . .	79
A.3	Prompt used for the door module. . . . .	80
A.4	Prompt used for the window module. . . . .	81
A.5	Prompt used for the object selection module. . . . .	82
A.6	Prompt used for the object selection module if in the first try we didn't gather enough objects to fill the room. . . . .	83
A.7	Prompt used for the feedback module. . . . .	84
A.8	Template for Conformal Geometric Algebra Prompts Used by our framework. (1/2) . . . . .	86
A.9	Template for Conformal Geometric Algebra Prompts Used by our framework. (2/2) . . . . .	87
A.10	Euclidean Prompt template for the Scene Editing Agent. . . . .	88
A.11	Omniverse Prompt template for the Scene Editing Agent Part 1. . . . .	89
A.12	Omniverse Prompt template for the Scene Editing Agent Part 2. . . . .	90
A.13	Omniverse Prompt template for the Scene Editing Agent Part 3. . . . .	91

## Our Publications Related to this Work

- Progressive tearing and cutting of soft-bodies in high-performance virtual reality Kamarianakis et al. [2022], Manos Kamarianakis, Antonis Protopsaltis, Dimitris Angelis, Michail Tamiolakis, George Papagiannakis, *ICAT-EGVE 2022*
- MAGES 4.0: Accelerating the world’s transition to VR training and democratizing the authoring of the medical metaverse Zikas et al. [2023], Paul Zikas, Antonis Protopsaltis, Nick Lydatakis, Mike Kentros, Stratos Geronikolakis, Steve Kateros, Manos Kamarianakis, Giannis Evangelou, Achilleas Filipidis, Eleni Grigoriou, Dimitris Angelis, Michail Tamiolakis, Michael Dodis, George Kokiadis, John Petropoulos, Maria Pateraki, George Papagiannakis, *IEEE*
- Project Elements: A computational entity-component-system in a scene-graph pythonic framework, for a neural, geometric computer graphics curriculum Papagiannakis et al. [2023], George Papagiannakis, Manos Kamarianakis, Antonis Protopsaltis, Dimitris Angelis, Paul Zikas, *EuroGraphics 2023*
- Geometric Algebra Meets Large Language Models: Instruction-Based Transformations of Separate Meshes in 3D, Interactive and Controllable Scenes, Dimitris Angelis, Prodromos Kolyvakis, Manos Kamarianakis, George Papagiannakis, (submitted)



# Chapter 1

## Introduction

The creation of 3D scenes holds profound importance in contemporary digital landscapes, transcending traditional 2D representations to offer immersive, interactive environments across various domains. From entertainment and gaming to education, simulation, and beyond, 3D scenes serve as powerful tools for storytelling, communication, and exploration. Their ability to simulate real-world environments with high fidelity and interactivity opens up avenues for applications such as architectural visualization, virtual reality experiences, and scientific data analysis. As technology advances, the role of 3D scenes continues to expand, shaping the way we interact with digital content and enabling new possibilities for creative expression and problem-solving.

Generating realistic, diverse, and interactive 3D environments is pivotal to the success of various applications. However, existing environment creation methods, including manual design, 3D scanning, and procedural generation with hard-coded rules, are labor-intensive and limited in their scope. They require significant human effort to design complex layouts, select compatible assets, and ensure semantic consistency among scene elements. Consequently, previous research has primarily focused on generating specific types of environments. To overcome these limitations, recent studies have leveraged 2D foundational models to generate 3D scenes from text Zhang et al. [2023], Höllein et al. [2023], Fridman et al. [2023]. While promising, these models often produce scenes with notable artifacts, such as mesh distortions, and lack interactivity. Other models are tailored for specific tasks, like floor plan generation Hu et al. [2020], Shabani et al. [2023] or object arrangement Wei et al. [2023], Paschalidou et al. [2021]. Furthermore, language-guided scene generation systems that utilize multiple large language models (LLMs) to create scenes sequentially have been proposed Yang et al. [2023]. Nevertheless, these systems lack essential features for more interactive scene creation, including self-evaluation and improvement, integration of vision models to enhance generation, and are restricted to specific frameworks and asset datasets.

Additionally, the editing of 3D scenes represents a crucial aspect of content creation, allowing creators to refine, customize, and optimize virtual environments

to meet specific requirements and objectives. Whether adjusting object placements, fine-tuning lighting effects, or refining textures and materials, the editing process empowers creators to craft scenes that accurately convey their vision and narrative. Beyond aesthetic enhancements, 3D scene editing plays a vital role in fields such as architectural design, where precise modifications are essential for visualizing proposed structures and evaluating spatial relationships. Moreover, in interactive applications such as video games and virtual simulations, real-time editing capabilities enable dynamic storytelling and user engagement, fostering immersive experiences that respond to user input and interactions.

Traditionally, 3D scene editing has required intensive manual effort and specialized knowledge, thereby restricting its efficiency and accessibility. Recent advancements in Foundational Models Bommasani et al. [2021] signal a paradigm shift, suggesting that complex scene editing could become more intuitive and accessible through simple text-based instructions. However, existing machine learning-based scene representation techniques, such as Neural Radiance Fields (NeRFs) Mildenhall et al. [2020] and Gaussian Splatting Kerbl et al. [2023], present substantial challenges for precise object repositioning due to their holistic nature, which often obscures individual object details and limits model generalization due to scene diversity Yu et al. [2020]. In contrast, using separate mesh scene representations provides a viable solution by granting direct access to individual mesh components, facilitating intuitive interactions and precise alignments tailored to specific editing requirements Zhai et al. [2024]. The overarching question remains: *How effectively can machine learning interact with these separate mesh components for editing?*

The advent of Large Language Models (LLMs) Ramesh et al. [2021], Touvron et al. [2023], Jiang et al. [2023] has substantially advanced human-computer interaction by utilizing linguistic proxies to facilitate instruction-based applications Hong et al. [2023a], De La Torre et al. [2024], Gong et al. [2024], Durante et al. [2024], Yang et al. [2023]. Furthermore, LLMs have significantly contributed to the field of Neurosymbolic AI Sheth et al. [2023], Garcez and Lamb [2023], enhancing our ability to transform linguistic instructions into precise symbolic representations that bridge the gap between LLMs' comprehensive understanding abilities and the precision required for complex spatial transformations Dziri et al. [2023], Jignasu et al. [2023], Hong et al. [2023b]. This integration prompts a critical inquiry into the optimal symbolic notation that encapsulates geometric transformations in a manner that LLMs can readily interpret. Geometric Algebra (GA), also known as Clifford Algebra, provides a robust mathematical structure ideal for managing transformations and interactions of geometric objects, which has been widely applied in Computer Graphics Papagiannakis [2013], Papaefthymiou et al. [2016], Gunn and De Keninck [2019], Hildenbrand and Rockwood [2022].

We introduce a language-guided framework built upon ThreeDWorld (TDW) framework Gan et al. [2021] to generate diverse, customized, and interactive 3D environments from textual descriptions. Our framework selects from over 50,000 high-quality 3D assets from Objaverse Deitke et al. [2023] to satisfy a wide range of environment descriptions. Using a Large Language Model (GPT-4), our framework



designs floor plans, assigns suitable materials, installs doorways and windows, and arranges 3D assets coherently in the scene using constraint-based optimization. Leveraging the emergent abilities of Large Language Models (LLMs), our framework exploits the commonsense priors and spatial knowledge inherently present in LLMs. Beyond object selection and layout design, our framework demonstrates its versatility in style customization by applying textures and designs to the scene and its objects. Additionally, our framework showcases its proficiency in spatial reasoning, such as devising floor plans and arranging objects regularly in scenes. Furthermore, our framework features a self-improvement pipeline, where an agent based on a Large Vision-Language Model (GPT-4V) provides feedback on generations and iteratively improves the scene. We also support visual guidance by providing images of real-life examples of the desired scenes. Finally, we employ RAG (Retrieval Augmented Generation) in our pipeline to help each module consider high-quality scenes.

Additionally, we employ Conformal Geometric Algebra (CGA) to effectively integrate intuitive linguistic instructions with precise geometric operations, offering a more accessible method for editing separate mesh 3D scenes. This method utilizes the zero-shot learning capabilities of Large Language Models (LLMs), which allows for adaptability to new 3D environments without the requirement for scene-specific training. On top of that, we employ caching and RAG (Retrieval Augmented Generation) capabilities, to either use already existing equations for existing instructions or provide our agent with more examples that would help the generation of the final equation. Our contribution is the development of a system tailored for separate mesh scene editing, particularly designed for object repositioning through textual descriptions. Integrated within the ThreeDWorld (TDW) framework Gan et al. [2021], our framework supports the Unity3D Engine and enhances VR-ready 3D scene interaction capabilities. Through rigorously designed experiments, we show that our framework markedly surpasses existing LLM-based alternatives, including those used in NVIDIA’s Omniverse, in terms of object repositioning within 3D and interactive scenes. A critical aspect of our work is the detailed examination of the limitations currently faced by LLM solutions in object repositioning tasks. Furthermore, we illustrate that LLMs can proficiently execute and manipulate CGA operations with minimal input.

Ultimately, our research marks a significant advancement, potentially democratizing the creation and manipulation of digital environments by simplifying the complex technicalities traditionally involved in 3D scene editing.



# Chapter 2

## State of the Art

### 2.1 Large Language Model for Scene Design

Numerous approaches to scene design have been proposed in literature, ranging from learning spatial priors from existing 3D scene databases Chang et al. [2017], Tan et al. [2019], Ma et al. [2018], Tang et al. [2023], Zhao et al. [2023], Wang et al. [2021b], Wei et al. [2023] to leveraging user input for iterative refinement of 3D scenes Chang et al. [2014], Cheng et al. [2019]. However, the reliance on datasets with limited categories, such as 3D-FRONT Fu et al. [2021], constrains the applicability of these methods. Recently, the integration of Large Language Models (LLMs) has emerged as a promising avenue for generating 3D scene layouts Feng et al. [2023], Lin et al. [2023b](as illustrated in Figure 2.1). Nevertheless, existing approaches often suffer from limitations such as directly outputting numerical values by LLMs, which can result in layouts that defy physical plausibility (e.g., overlapping assets). A more comprehensive methodology was introduced by Yang et al. [2023](as illustrated in Figure 2.1), which effectively leveraged LLMs for 3D scene generation. However, this approach fell short in fully utilizing certain features, such as RAG and the incorporation of vision modalities. In contrast, our proposed framework utilizes LLMs to sample spatial relational constraints, subsequently optimized by a solver to ensure physically plausible scene arrangements. Results from our human study demonstrate a preference for layouts generated by our framework over those produced solely by LLMs in an end-to-end manner.



Figure 2.1: Top: Scene generated using Holodeck. Bottom: Scene generated using LayoutGPT.

## 2.2 Text-driven 3D Generation

Early efforts in 3D generation primarily revolved around learning the distribution of 3D shapes and textures from datasets specific to certain categories Wu et al. [2016], Yang et al. [2019], Zhou et al. [2021], Henzler et al. [2019], Nguyen-Phuoc et al. [2019]. Subsequently, the emergence of large vision-language models like CLIP Radford et al. [2021] facilitated zero-shot generation of 3D textures and objects Huang et al. [2023], Mildenhall et al. [2021], Poole et al. [2022], Metzger et al. [2023], Lin et al. [2023a], Gu et al. [2023]. While these advancements excel at generating individual 3D objects, they often struggle to produce complex 3D scenes. More recently, emerging approaches have attempted to generate 3D scenes by integrating pre-trained text-to-image models with depth prediction algorithms, yielding either textured meshes, Neural Radiance Fields (NeRFs) or Gaussian Splatting Fridman et al. [2023], Höllein et al. [2023], Zhang et al. [2023], Kerbl et al. [2023]. However, these methods often result in 3D representations lacking modular composability and interactive affordances, thereby limiting their suitability for embodied AI applications. In contrast, our framework leverages a comprehensive 3D asset database to generate semantically precise, spatially efficient, and interactive 3D environments.



Figure 2.2: Scene generated using Text2Room and Neural Descriptor Fields.

## 2.3 Agent-Driven Object Rearrangement

In the domain of object rearrangement, the quest to comprehend reusable abstractions through geometric goal specifications spans from simple coordinate transformations to intricate multi-object scenarios [Batra et al. 2020]. Chang et al. [2023] tackle rearrangement as an offline goal-conditioned reinforcement learning problem, orchestrating actions to reposition objects from an initial setup in an input image to align with criteria defined by a goal image. Kapelyukh and Johns [2023] propose learning a cost function via an energy-based model to favor object arrangements reminiscent of human behavior. Simeonov et al. [2023] address rearrangement tasks employing Neural Descriptor Fields [Simeonov et al. 2022], assigning consistent local coordinate frames to task-relevant object parts, localizing these frames on unseen objects, and aligning them through executed actions. Furthermore, Zhai et al. [2024] integrates scene graphs with diffusion processes for editable generative models, albeit requiring extensive training and access to complete scene graphs, contrasting with the efficiency of our localized transformation approach.

Diverging from methods reliant on extensive scene-specific training, our approach harnesses the generalization capabilities of Large Language Models (LLMs) to simplify object repositioning tasks. In contrast to Kwon et al. [2024], who confine LLM applications to Euclidean spaces for predicting end-effector poses, and Mavrogiannis et al. [2023] (as illustrated in Figure 2.3), who necessitate numerous and cumbersome predefined predicates for translating cooking instructions into Linear Temporal Logic, our method exploits Conformal Geometric Algebra

(CGA) and decomposes repositioning tasks into a minimal set of primitive transformations. Unlike the LLMR approach De La Torre et al. [2024], which employs specialized LLMs to generate C# Unity code for scene creation and editing (as illustrated in Figure 2.3), our method circumvents direct code generation, ensuring precise scene manipulation without the intricacies of code authoring and debugging. Finally, building on exploration of natural language in virtual environment creation Manesh et al. [2024], our work advances further by addressing complex, multi-object repositioning tasks, thus significantly propelling the realm of future VR scene manipulation.

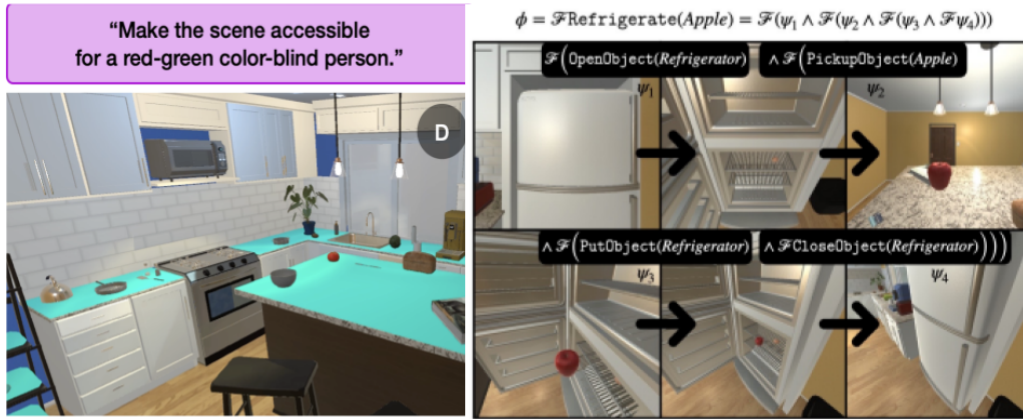


Figure 2.3: Object Rearrangement through code generation. Left: LLMR. Right: Cook2LTL

## 2.4 Machine Learning Applications of Geometric Algebra

The integration of Geometric Algebra (GA) into neural computation was first introduced in Pearson and Bisset [1994], with subsequent developments introducing multivector-valued neurons for radial basis function networks Corrochano et al. [1996], multilayer perceptrons (MLPs) Buchholz [2000], Buchholz and Sommer [2001], and various neural network architectures Buchholz and Le Bihan [2008], Buchholz and Sommer [2008], Buchholz et al. [2007], Bayro-Corrochano [2001]. GA-based neural networks have found applications across diverse domains, including signal processing Buchholz and Le Bihan [2008], robotics Bayro-Corrochano et al. [2018], partial differential equation modeling Brandstetter et al. [2022], fluid

dynamics Ruhe et al. [2023], and particle physics Ruhe et al. [2024]. Additionally, novel GA-based architectures such as multivector-valued convolutional neural networks (CNNs) Li et al. [2022], Wang et al. [2021a], recurrent neural networks Kuroe [2011], Zhu and Sun [2016], and transformer networks Liu and Cao [2022], Brehmer et al. [2024] have been introduced, showcasing the versatility and efficacy of GA in enhancing neural network capabilities for geometrically oriented tasks.

Unlike these approaches, our work does not directly incorporate geometric algebraic components within deep learning architectures. Instead, we utilize geometric algebra as a communicative mediator between the Large Language Model (LLM) and our object rearrangement application. This approach underscores the use of geometric algebra primarily as a tool to enhance the interaction and translation of complex geometrical tasks into understandable formats for the LLM, thereby fostering more effective problem-solving capabilities in practical applications. Recently, Wang et al. [2023] fine-tuned ChatGPT with a large curated collection of textual documents on Geometric Algebra, aimed at developing customized learning plans for students in diverse fields. In contrast, our work diverges from the fine-tuning approach of LLMs and instead investigates whether LLMs can effectively utilize geometric algebra for object rearrangement tasks with minimal prompting.

Relevant Methods	Text-Guided Scene Generation	Image-Guided Scene Generation	Text-Guided Scene Editing	RAG	Caching	Automated Feedback
Yang et al. [2023]	x					
Höllein et al. [2023]	x					
Feng et al. [2023]	x		x			
De La Torre et al. [2024]			x			
Mavrogiannis et al. [2023]			x		x	
Our Method	x	x	x	x	x	x

Table 2.1: Comparison of main features across relevant papers with our system.



## Chapter 3

# 3D Scene Generation

In this chapter, we introduce the scene generation component of our framework, a promptable system built upon the foundation of ThreeDWorld Gan et al. [2021] and enriched with an extensive asset library from Objaverse Deitke et al. [2023]. This system is designed to produce diverse, customizable, and interactive environments guided by large language models.

Our framework is inspired by Yang et al. [2023] and adopts a systematic approach to scene construction, leveraging a series of specialized modules: (1) the *Floor & Wall Module* which generates floor plans, constructs wall structures, and selects appropriate materials for floors and walls; (2) the *Doorway & Window Module* responsible for integrating doorways and windows into the environment; (3) the *Object Selection Module* tasked with retrieving suitable 3D assets from Objaverse; (4) the *Constraint-based Layout Design Module* which arranges assets within the scene by employing spatial relational constraints to ensure realistic object placement; and (5) the *Feedback Module*, providing iterative feedback based on the design and layout of the generated scene to facilitate continuous improvement in subsequent generations.

The main differences from Yang et al. [2023] (as illustrated in Table 2.1) are the addition of the Feedback Module, the inclusion of vision capabilities in the pipeline, and the integration of RAG (Retrieval Augmented Generation). Furthermore, our framework is built on top of ThreeDWorld Gan et al. [2021], offering greater modularity in rendering capabilities and features. This foundation also facilitates the expansion of the library of 3D assets and materials beyond Objaverse Deitke et al. [2023].

### 3.1 Prompt Design

Each module retrieves information from a language model and translates it into components incorporated into the final layout. For every module, a Large Language Model (LLM) prompt is constructed, comprising three fundamental elements: (1) *Task Description*, outlining the context and goals of the task; (2)

*Output Format*, defining the expected structure and format of outputs; and (3) *One-shot Example*, presenting a specific instance to assist the LLM in comprehending the task. The text within the Figures of each module that are following, provides simplified examples of prompts<sup>1</sup>. Responses from LLMs to these prompts are processed at a high level and utilized as input parameters for modules to generate detailed specifications of the scene.

## 3.2 Floor & Wall Module

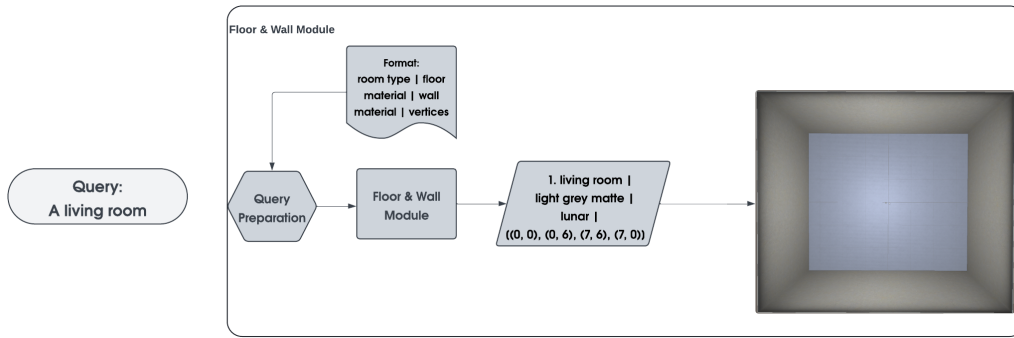


Figure 3.1: Floor & Wall Module Overview

The **Floor & Wall Module**, depicted in Figure 3.1, is responsible for generating floor plans, constructing wall structures, and selecting materials for floors and walls. Each room is represented as a rectangle, defined by four tuples that specify the coordinates of its corners.

GPT-4 provides direct coordinates for room placement and suggests realistic dimensions and connectivity for these rooms.

Furthermore, this module selects materials for the floors and walls, a critical aspect in enhancing the realism of the environments. Our framework matches LLM proposals to one of 538 materials, enabling semantic customization of scenes. As illustrated in Figure 3.2, our framework can create scenes with appropriate materials tailored to the specific scene type, thereby augmenting the authenticity and immersion of the generated environments.

### 3.2.1 Output Format

In the LLM outputs for the Floor & Wall Module, the following details are provided for each room:

- **Room Type:** The room’s name, e.g., kitchen, bedroom.

<sup>1</sup>Complete prompts (available in the appendices) include additional guidance to aid LLMs and mitigate common errors



Figure 3.2: **Material Customizability.** Our framework can select appropriate floor and wall materials to make the scenes more realistic.

- **Floor Material:** A description of the floor’s appearance.
- **Wall Material:** A description of the wall’s appearance.
- **Vertices:** Four tuples  $\{(x_i, y_i), i \in [1, 2, 3, 4]\}$ , representing the coordinates of the room’s corners.

### 3.2.2 Material Selection

We have an image representation for each of 538 materials. Using CLIP<sup>2</sup> Radford et al. [2021], we calculate the similarity between the material descriptions provided by the Large Language Model (LLM) and these images. The material with the highest similarity score is selected.

### 3.2.3 Wall Height

The LLM suggests a suitable wall height based on the user’s input. For example, it may recommend a high ceiling for spaces like museums.

## 3.3 Doorway & Window Module

The **Doorway & Window Module**, depicted in Figure 3.3, oversees the suggestion of room connections and windows within the scene. Each of these components is independently queried from the LLM. The LLM is capable of suggesting doorways and windows, which can be adjusted based on various properties such as size, height, and quantity. This module plays a crucial role in ensuring that the placement and attributes of doorways and windows contribute to the overall coherence and realism of the generated 3D environments.

<sup>2</sup>We employ OpenCLIP Ilharco et al. [2021] with ViT-L/14, trained on the LAION-2B dataset Schuhmann et al. [2022], for all CLIP-related components in this thesis.

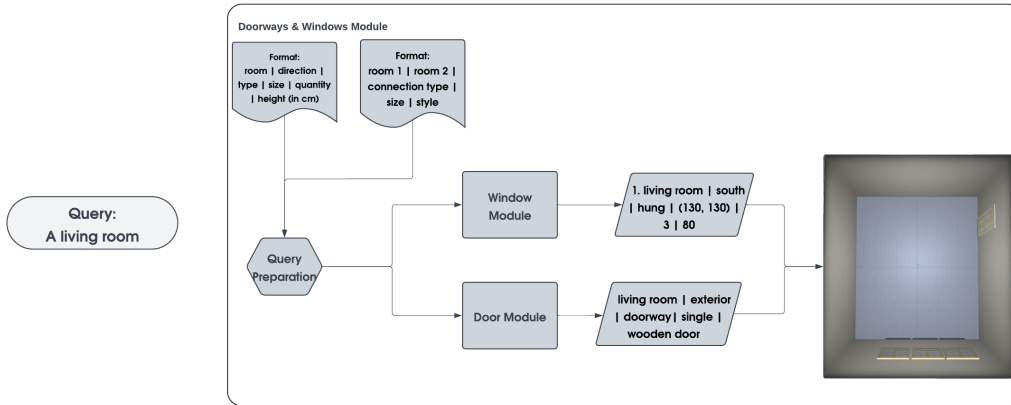


Figure 3.3: Doorway &amp; Window Module Overview

### 3.3.1 Door Selection

The LLM provides essential information to aid in the selection of doors:

- **Room 1 & Room 2:** The two rooms connected by the door, for example, bedroom and kitchen.
- **Connection Type:** One of the three connection types: *doorframe* (frame without a door), *doorway* (frame with a door), and *open* (no wall separating the rooms).
- **Size:** The size of the door: *single* (one meter in width) or *double* (two meters in width).
- **Door Style:** A description of the door’s appearance.

We have an image for each door, and we utilize CLIP to select the door that most closely matches the description.

### 3.3.2 Window Selection

The LLM provides the following data about windows:

- **Room Type:** The room where the window will be installed.
- **Direction:** The wall’s direction (*south*, *north*, *east*, or *west*) where the window will be placed.
- **Type:** One of the three window types: *fixed*, *slider*, or *hung*.
- **Size:** The width and height of the window.
- **Quantity:** The number of windows installed on each wall.
- **Height:** The distance from the floor to the window’s base.

## 3.4 Object Selection Module

### 3.4.1 Object Selection Module

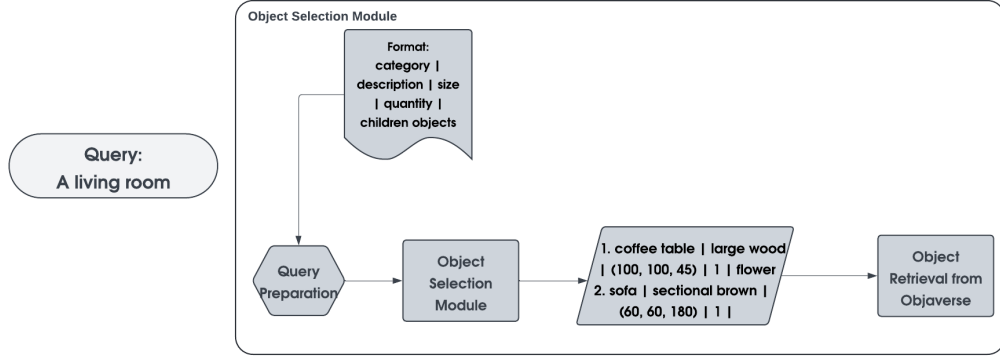


Figure 3.4: Object Selection Module Overview

The **Object Selection Module**, illustrated in Figure 3.4, enables the LLM to suggest objects for inclusion in the layout. Utilizing the extensive Objaverse asset collection, our framework can source and place a diverse array of objects within the scene. Queries are generated using descriptions and dimensions proposed by the LLM, such as "coffee table — large wood — (100, 100, 45)," allowing for the retrieval of the most suitable asset from Objaverse.

The retrieval function<sup>3</sup> considers both visual and textual similarities, as well as dimensional aspects, to ensure the selected assets match the intended design. Figure 3.5 demonstrates our framework’s ability to customize and position a wide variety of objects, whether on the floor, walls, atop other items, or even on the ceiling, ensuring the creation of cohesive and contextually relevant scenes.

### 3.4.2 Leveraging Objaverse Assets

Our framework supports the creation of diverse and tailored scenes. We source assets from Objaverse 1.0 Deitke et al. [2023], annotated with details including textual descriptions, scale, and canonical views as provided by Yang et al. [2023].

To integrate Objaverse assets into ThreeDWorld Gan et al. [2021], we use their pipeline for mesh optimization and conversion into a compatible format.

In Objaverse, each 3D asset  $o \in \mathcal{O}$  is associated with the following metadata: a textual description of the asset  $t$ , the 3D bounding box size of the asset  $(w, d, h)$ , and a set of 2D images  $I$  captured from three different angles (0, 45, and  $-45$ ). For each object proposed by the LLM  $o'$ , we obtain a detailed description of the object  $(t')$  and its 3D bounding box size  $(w', d', h')$  for retrieval purposes.

<sup>3</sup>We use CLIP Radford et al. [2021] for visual similarity, Sentence-BERT Reimers and Gurevych [2019] for textual similarity, and 3D bounding box sizes for dimensionality considerations.



Figure 3.5: **Objects Customizability.** Our framework can select and place appropriate objects based on the input, such as “living room” and “wine cellar”.

To evaluate the similarity between a candidate 3D asset in the repository  $o = (t, (w, d, h), I)$  and the object proposed by the LLM  $o' (t', (w', d', h'))$ , we use three metrics:

- **Visual Similarity** ( $\mathcal{V}$ ) measures the CLIP similarity between the 2D renderings of the candidate asset and the textual description of the LLM-proposed object:  $\mathcal{V}(o, o') = \max_{i \in I} \text{CLIP}(i, t')$ .
- **Textual Similarity** ( $\mathcal{T}$ ) measures the similarity between the textual description of the candidate 3D asset and the textual description of the LLM-proposed object. This metric is crucial in improving retrieval accuracy, ensuring that we select assets within the correct category. We use the sentence transformer (SBERT) Reimers and Gurevych [2019] with all-mpnet-base-v2 checkpoint to calculate the scores:  $\mathcal{T} = \text{SBERT}(t, t')$ .
- **Size Discrepancy** ( $\mathcal{S}$ ) measures the difference in the 3D bounding box size between the candidate asset and the LLM-proposed object. Since the size of objects is important in designing scenes, e.g., a larger sofa for a large living room, the size matching score is computed as:

$$\mathcal{S}(o, o') = (|w - w'| + |h - h'| + |d - d'|) / 3.$$

Two objects of similar size will have a smaller value of  $\mathcal{S}$ .

The overall matching score  $\mathcal{M}(o, o')$  is a weighted sum of the above metrics:

$$\mathcal{M}(o, o') = \alpha \cdot \mathcal{V}(o, o') + \beta \cdot \mathcal{T}(o, o') - \gamma \cdot \mathcal{S}(o, o') \quad (3.1)$$

with weights  $\alpha = 100$ ,  $\beta = 1$ , and  $\gamma = 10$ . The asset with the highest matching score is selected.

### 3.5 Constraint-based Layout Design Module

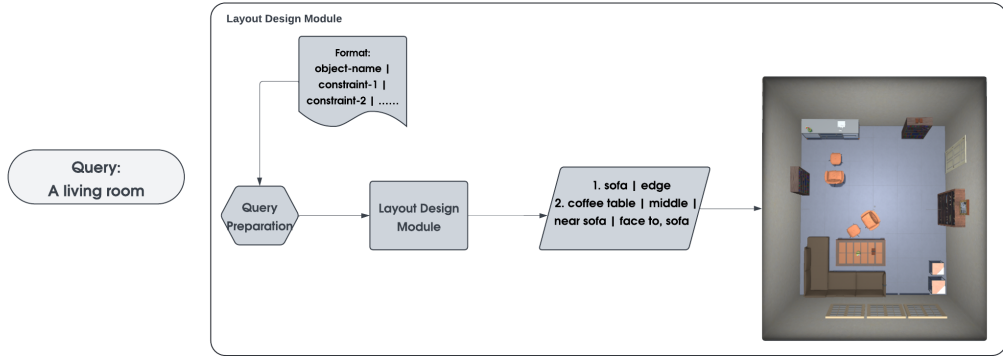


Figure 3.6: Constraint Layout Design Module Overview

The **Constraint-based Layout Design Module**, depicted in Figure 3.6, is responsible for determining the positioning and orientation of objects within the scene. While previous research Feng et al. [2023] has shown that LLMs can directly provide absolute values for an object’s bounding box, this approach often leads to errors such as out-of-boundary placements and object collisions, particularly in complex environments with diverse assets. To overcome these challenges, we adopt a constraint-based strategy inspired by Yang et al. [2023]. This approach involves the LLM generating spatial relationships between objects, such as “coffee table, in front of, sofa,” and then optimizing the layout based on these constraints. Leveraging the probabilistic nature of LLMs, our framework can generate multiple valid layouts from the same prompt, enhancing the diversity and adaptability of the resulting scenes.

#### 3.5.1 Spatial Relational Constraints

In alignment with Yang et al. [2023], we define ten types of constraints, categorized into five groups: (1) Global: *edge, middle*; (2) Distance: *near, far*; (3) Position: *in front of, side of, above, on top of*; (4) Alignment: *center aligned*; and (5) Rotation: *face to*. The LLM selects a subset of these constraints for each object, forming a scene graph for the room. These constraints are treated flexibly, allowing for certain violations when finding a layout that satisfies all constraints is not possible. Additionally, we enforce hard constraints to prevent object collisions and ensure all objects remain within the room’s boundaries.

### 3.5.2 Constraint Satisfaction

We begin by translating the spatial relational constraints into precise mathematical conditions. To generate layouts adhering to the selected constraints, we employ an optimization algorithm that positions objects in an autoregressive manner. Initially, the algorithm identifies an anchor object and explores potential placements based on LLM guidance. It then employs Depth-First Search (DFS) to find valid positions for the remaining objects, ensuring adherence to hard constraints. Soft constraints are considered with flexibility, allowing for some violations in pursuit of a feasible layout.

For floor objects, constraints include global constraints (e.g., edge, middle), distance constraints (e.g., near, far), position constraints (e.g., in front of, side of), alignment constraints (e.g., center align with), and rotation constraints (e.g., face to). The LLM combines these constraints to form a constraint list for each object, guiding the DFS Solver in positioning objects within the scene. The DFS Solver explores different placements while optimizing object positioning to meet the most constraints.

Wall objects are placed based on attributes such as their relationship with floor objects (e.g., above) and specified height. Small surface objects are positioned on larger surfaces using the DFS Solver and constraining the 2D grid on the surface of the larger objects.

## 3.6 Feedback Module

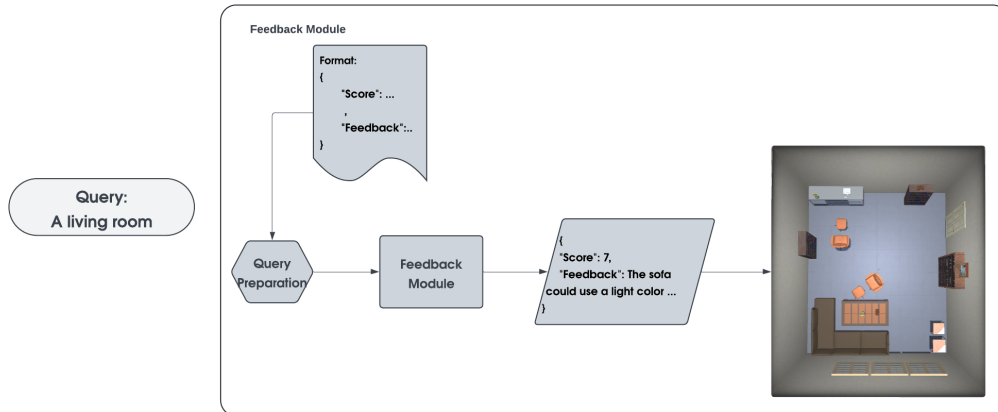


Figure 3.7: Feedback Module Overview

The **Feedback Module** is a crucial component of our framework, designed to enhance the quality and realism of generated 3D scenes through iterative improvement. This module leverages GPT-4 Vision to analyze various perspectives of the scene and provide detailed feedback and scoring.



### 3.6.1 Input Data

The Feedback Module takes as input:

- **Top-down View:** A comprehensive overhead view of the entire generated scene.
- **First-person Views:** Four first-person perspective images for each room, captured from the angles of 0°, 90°, 180°, and 270°.

These inputs ensure that the module has a thorough visual representation of the scene, capturing both the overall layout and the detailed aspects from multiple angles.

### 3.6.2 Processing and Analysis

Upon receiving the input images, the Feedback Module processes them through the following steps:

#### 3.6.2.1 Quality Assessment

- **Aesthetic Evaluation:** The module assesses the aesthetic aspects of the scene, such as balance, symmetry, and overall visual appeal.
- **Functionality Check:** It evaluates the functionality and usability of the scene, ensuring that the layout is practical and objects are placed in a manner that supports intended interactions.

#### 3.6.2.2 Scoring Mechanism

The module assigns a score to the scene based on several criteria, including but not limited to spatial accuracy, object placement, aesthetic appeal, and functionality. Each criterion is weighted to reflect its importance in the overall scene quality.

### 3.6.3 Feedback Generation

Based on the analysis, the Feedback Module generates detailed feedback that includes:

- **Score:** A numerical score that reflects the overall quality of the scene.
- **Feedback:**
  - Highlights of the aspects that are well-executed in the scene.
  - **Object Placement:** Suggestions on adjusting the placement of objects to improve spatial relationships and enhance usability.

- **Layout Adjustments:** Recommendations for modifying the layout to better align with design constraints and improve the overall flow of the scene.
- **Aesthetic Enhancements:** Advice on aesthetic improvements, such as balancing object distribution, improving symmetry, and selecting more suitable materials.

### 3.6.4 Iterative Improvement

The feedback is used to iteratively improve the scene generation process. The iterative loop involves:

1. **Feedback Integration:** Incorporating the detailed feedback into the scene generation pipeline. Adjustments are made to object placement, layout configurations, and aesthetic choices based on the module’s recommendations.
2. **Regeneration:** The scene is regenerated with the incorporated feedback, creating an improved version of the initial scene.
3. **Re-evaluation:** The updated scene is re-evaluated by the Feedback Module to ensure that the suggested improvements have been effectively implemented.

This iterative process continues until the generated scene meets the desired quality standards, ensuring that each iteration progressively enhances the scene’s realism, usability, and visual appeal.

By utilizing GPT-4 Vision in this feedback loop, our framework achieves a high level of refinement in scene generation, producing environments that are not only realistic and functional but also visually compelling and contextually appropriate.

## 3.7 Real-life Image Integration

The **Real-life Image Integration** feature enhances the realism and specificity of generated 3D scenes by incorporating real-life images as input. This feature is applied across all modules, leveraging GPT-4 Vision to analyze real-life photos and textual guidance, providing a more structured and specific approach to scene generation (as illustrated in Figure 3.8).

### 3.7.1 Input Data

The Real-life Image Integration feature takes as input:

- **Real-life Images:** Photos of actual rooms or environments that serve as references for the generated scene. Thanks to GPT-4’s exceptional visual capabilities, each module can process images ranging from low to very high quality.

- **Text Guidance:** Descriptive text providing context and specific details about the desired elements and layout of the scene.

### 3.7.2 Feedback Generation

Based on the analysis, the feature generates detailed feedback that includes:

- **Comparative Analysis:** The feature compares the generated scene with the real-life images to evaluate how well the scene matches the desired characteristics and spatial configurations.
- **Detailed Feedback:** Specific suggestions on adjustments needed to more closely align the generated scene with the real-life references, focusing on aspects such as object placement, materials, and overall layout.

This process ensures that the generated scenes not only adhere to abstract design principles but also closely mimic real-world environments, enhancing the authenticity and practical relevance of the generated scenes.



Figure 3.8: Example of real-life images used as references for scene generation.

## 3.8 Retrieval Augmented Generation (RAG) Integration

The **Retrieval Augmented Generation (RAG)** feature enhances the scene generation process by leveraging previous successful generations stored in a database. This integration allows the framework to use the most relevant past examples as

references for new queries, providing the Large Language Model (LLM) with valuable context and examples to improve the quality and coherence of the generated scenes.

### 3.8.1 Database of Previous Generations

The RAG feature maintains a comprehensive database containing successful scene generations from each module. This database includes:

- **Scene Descriptions:** Detailed textual descriptions of the generated scenes, including context and specific design elements.
- **Generated Outputs:** The final outputs of each module, such as floor plans, object placements, and material selections.

### 3.8.2 Query Handling

When a new query is received, the RAG feature processes it through the following steps:

#### 3.8.2.1 Query Analysis

The query is compared against the database of previous generations to identify the most relevant examples. This comparison is based on textual similarity, design elements, and overall scene context.

#### 3.8.2.2 Reference Retrieval

The most relevant previous generations are retrieved from the database. These references include previous generations that share similar design requirements or contexts with the new query.

### 3.8.3 Incorporation into Scene Generation

The retrieved references are incorporated into the scene generation process as follows:

#### 3.8.3.1 Guided Generation

- **Contextual Guidance:** The LLM uses the retrieved references to understand the context and design elements that led to successful previous generations.
- **Example-Based Prompts:** The references are used to construct example-based prompts for the LLM, providing concrete instances of successful designs to guide the generation process.

### 3.8.4 Evaluation and Benefits

The integration of RAG offers several key benefits:

- **Enhanced Coherence:** By using successful past examples as references, the RAG feature helps maintain consistency and coherence across different scene generations.
- **Improved Quality:** The incorporation of high-quality references ensures that new generations meet established design standards and exhibit refined aesthetic qualities.
- **Increased Efficiency:** The RAG feature streamlines the generation process by providing the LLM with concrete examples, reducing the time and effort required to achieve desirable results.

The RAG integration significantly enhances the capability of our framework to generate diverse, high-quality 3D scenes by leveraging the knowledge and success of previous generations, thereby facilitating a more effective and efficient design process.



## Chapter 4

# 3D Scene Editing

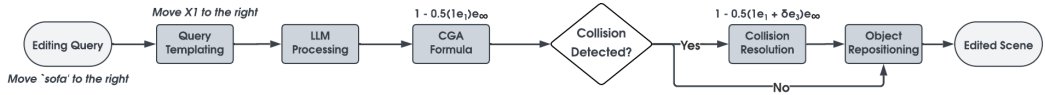


Figure 4.1: Scene editing component overview: It generates a CGA transformation after *templating* the query and adds a  $\delta$  upward translation for a hypothetical Z-axis collision.

In this chapter, we introduce the scene editing component of our framework, which facilitates dynamic and precise 3D scene manipulation. As depicted in Figure 4.1, the system processes a user query, such as *move the sofa to the right*, by first converting it into a standardized templated form, like *move X1 to the right*.

Our framework achieves this conversion by searching for exact matches within the query and sequentially introducing variables for each identified match. This templating process allows the Large Language Model (LLM) to perform symbolic reasoning effectively. Once the query is templated, the system populates it with the user’s input and generates the corresponding Conformal Geometric Algebra (CGA) transformations to achieve the desired scene modifications.

It is important to note that our framework operates at a local level, focusing on individual objects without a comprehensive understanding of other objects in the scene. This localized approach enhances processing efficiency but may lead to potential collisions with other objects in the environment.

To address the possibility of collisions, our framework leverages the inherent flexibility of queries, such as *move the sofa to the right*, which often allow for multiple valid solutions. Upon detecting a collision, the system explores alternative perturbed transformations to resolve the issue. For instance, it may apply a small  $\delta$  upward translation along the Z-axis to avoid overlapping objects, ensuring minimal deviation from the intended initial transformation.

As a final remark, the proposed scene editing system is integrated within the ThreeDWorld (TDW) framework Gan et al. [2021]. This integration supports the Unity3D Engine, facilitating the creation of VR-ready 3D scenes and enhancing

Move X0 to the right	Move X0 on top of X1
$1 - 0.5 \cdot (1 \cdot e_1) \cdot e_\infty$	$1 - 0.5 \left( \begin{array}{l} \left( \left( \frac{X_1^{\max} + X_1^{\min}}{2}   e_1 \right) \cdot e_1 - \left( \frac{X_0^{\max} + X_0^{\min}}{2}   e_1 \right) \cdot e_1 \right) \\ + \\ \left( \left( \frac{X_1^{\max} + X_1^{\min}}{2}   e_3 \right) \cdot e_3 - \left( \frac{X_0^{\max} + X_0^{\min}}{2}   e_3 \right) \cdot e_3 \right) \\ + \\ \left( \left( X_1^{\max}   e_2 \right) \cdot e_2 + \left( \left( X_0^{\max} - X_0^{\min} \right)   e_2 \right) \cdot e_2 - \left( X_0^{\min}   e_2 \right) \cdot e_2 \right) \end{array} \right) \cdot e_\infty$

Figure 4.2: Examples of conformal geometric algebra equations generated for the queries *Move X0 to the right* and *Move X0 on top of X1*.

human interaction within virtual reality environments. By leveraging the power of CGA and the flexibility of templated queries, our framework offers a robust solution for precise and efficient 3D scene editing.

## 4.1 Conformal Geometric Algebra

In his seminal 1872 Erlangen Programme, Felix Klein introduced the revolutionary idea that algebraic structures govern geometric shapes, asserting that geometry is best understood through algebra Hawkins [1984]. This concept laid the foundation for modern geometric algebra (GA), which unifies various algebraic systems, including vector algebra, complex numbers, and quaternions, under the broader framework of *multivectors*. Unlike traditional methods that treat scalars, vectors, and higher-dimensional entities separately, GA provides a cohesive structure for uniformly representing and manipulating these elements.

At the core of GA is the *geometric product*, which generalizes the dot product and the cross product within Euclidean spaces. Since all other products can be derived from the geometric product, it suffices to use the geometric product, along with addition, scalar multiplication, and conjugation, for any multivector operation.

In this work, we employ Conformal Geometric Algebra (CGA), a 32-dimensional extension of quaternions and dual-quaternions Rooney [2007]. CGA enables the uniform representation of various geometric entities such as vertices, spheres, planes, as well as transformations including rotations, translations, and dilations,



all expressed as multivectors Kamarianakis and Papagiannakis [2021].

For example, using the standard CGA basis elements  $\{e_1, e_2, e_3, e_4, e_5\}$ , a CGA basis comprises all 32 ( $2^5$ ) combinations of these elements through the geometric product, i.e.,  $\{1, e_i, e_i e_j, e_i e_j e_k, e_i e_j e_k e_l, e_1 e_2 e_3 e_4 e_5$  for  $1 \leq i < j < k < l \leq 5\}$ . For convenience, we define the vectors  $e_o = 0.5(e_5 - e_4)$  and  $e_\infty = e_4 + e_5$ , and use subscripts to denote products of basic elements, e.g.,  $e_{ijk} := e_i e_j e_k$ . Using this notation, a sphere  $s$  centered at  $x = (x_1, x_2, x_3)$  with radius  $r$  is represented by the CGA multivector:

$$S = x_1 e_1 + x_2 e_2 + x_3 e_3 + \frac{1}{2}(x_1^2 + x_2^2 + x_3^2 - r^2)e_\infty + e_o. \quad (4.1)$$

Notably, setting  $r = 0$  yields the respective multivector for the point  $x$ , and given  $S$ , both  $x$  and  $r$  can be extracted.

In CGA, a translation by  $(t_1, t_2, t_3)$  is represented as:

$$T = 1 - 0.5(t_1 e_1 + t_2 e_2 + t_3 e_3)e_\infty, \quad (4.2)$$

with the inverse being:

$$T^{-1} = 1 + 0.5(t_1 e_1 + t_2 e_2 + t_3 e_3)e_\infty. \quad (4.3)$$

As noted in Kamarianakis and Papagiannakis [2021], a rotation typically expressed by the unit quaternion:

$$q := a - d\mathbf{i} + c\mathbf{j} - b\mathbf{k}, \quad (4.4)$$

can be represented by the corresponding *rotor*:

$$R = a + b e_{12} + c e_{13} + d e_{23}. \quad (4.5)$$

The inverse of  $R$  is given by:

$$R^{-1} = a - b e_{12} - c e_{13} - d e_{23}. \quad (4.6)$$

Additionally, the multivector:

$$D = 1 + \frac{1-d}{1+d} e_{45}, \quad (4.7)$$

corresponds to a dilation with scale factor  $d > 0$  with respect to the origin, and its inverse is:

$$D^{-1} = \frac{(1+d)^2}{4d} + \frac{d^2-1}{4d} e_{45}. \quad (4.8)$$

In the literature,  $e_{45}$  may be replaced by the equivalent  $e_\infty \wedge e_o$ , where  $\wedge$  denotes the *wedge* (or outer) product.

To apply transformations  $M_1, M_2, \dots, M_n$  to an object, we define the multivector  $M := M_n M_{n-1} \cdots M_1$ , where all intermediate products are geometric. Our framework can identify and generate the intermediate transformations  $M_i$  to be

applied to an object, yielding  $M$ . As a composition of rigid body transformations and dilations, the resulting multivector is equivalent to the product  $M = TRD$  (translation  $T$ , rotation  $R$ , and dilation  $D$ ). By extracting  $T$ ,  $R$ , and  $D$ , we can express  $M$  as a translation vector, unit quaternion, and scale factor, which can be visualized in TDW and Unity.

Extracting  $T$ ,  $R$ , and  $D$  from  $M = TRD$  is complex and undocumented in current literature. To achieve this, we apply  $M$  to a sphere  $C$  centered at the origin with radius 1. The transformed sphere's multivector  $C'$  is evaluated using the *sandwich product*  $C' = MCM^{-1}$ . From  $C'$ , we can extract its center  $x$  and radius  $r$ . Since  $C'$  results from applying  $M = TRD$  to  $C$ , its radius is the scaling factor of  $D$ , and its center is the translation vector of  $T$ . Knowing  $T$  and  $D$ , we can evaluate  $R := T^{-1}MD^{-1}$  and determine the corresponding unit quaternion using (4.4) and (4.5).

## 4.2 LLM Processing & CGA Formulation

Our framework utilizes Conformal Geometric Algebra (CGA) to handle spatial transformations within a 3D environment, leveraging the precision and rigor of this mathematical framework. This approach allows us to accurately represent and manipulate the spatial properties of objects.

To begin, our framework converts user queries into a templated format, which is crucial for abstracting the user's intent and preparing the data for algebraic processing. Each object  $X_i$  in the scene is represented by an axis-aligned bounding box, defined by two points,  $X_i^{\max}$  and  $X_i^{\min}$ , encapsulating the object's spatial extent. This bounding box model simplifies the calculation of movements and rotations by providing clear, definable limits to each object's position.

Through carefully designed prompts, we instruct the Language Model (LLM) on the key concepts and operations of CGA (The complete prompt template is provided in the Appendices Section). Below are some key remarks about the prompt structure:

- **Coordinate Extraction:** We explain how to use the inner product  $|$  operation to precisely isolate spatial coordinates. For instance,  $(X_1^{\max}|e_2)$  corresponds to the  $y$  coordinate of  $X_1^{\max}$ .
- **Outer (Wedge) Product:** This operation is defined to establish planes for rotational operations, detailing the mechanisms for both translation and rotation rotors. These rotors are individually defined and can be combined through rotor composition, enabling complex sequential transformations essential for accurate scene manipulation.

To effectively guide the LLM in using the rotation and translation rotors, we provided five illustrative examples:

- **Basic Examples:** One example each of rotation and translation to demonstrate fundamental movements.
- **Compositional Example:** An example that integrates both types of transformations, illustrating how to combine rotors.
- **Complex Queries:** Two additional examples address more intricate queries, such as *move on top of* and *move next to* another object.

This diverse set of examples ensures the LLM effectively understands and implements the necessary algebraic operations for object repositioning. Despite the limited number of examples, the LLM demonstrates the ability to generalize to more complex queries.

Finally, the LLM responds by generating a JSON output, which includes the specified rotors for composition to be applied to each object (as illustrated in Figure 4.2). This JSON output details the necessary transformations, allowing for precise and accurate adjustments within the 3D scene.

#### 4.2.1 Implementation Details

The scene editing agent processes user queries like *move sofa to the right*, converting them into a templated format such as "move X1 to the right." This involves matching specific phrases in the query and assigning variables accordingly. We then make an API call to OpenAI's *gpt-4-1106-preview* model via LangChain<sup>1</sup>. The model's JSON response maps edited object variables (e.g., 'X1') to CGA transformations (e.g., 'X1': 'T(-1 \* e1)'), where 'T' stands for translation and 'R' for rotation. Next, we transform this symbolic output into executable Python code using the Clifford<sup>2</sup> library, replacing variables like 'X' with actual object names and converting 'T' and 'R' into their respective functions. For example, the response might be transformed to 'sofa': 'generate\_translation\_rotor(-1 \* e1)'. These equations are then applied to the corresponding objects in the scene, implementing the specified transformations.

### 4.3 Collision Detection Module

Our framework features a robust Collision Detection Module designed to manage potential collisions following scene edits. This module leverages the bounding box information of each object to ensure seamless integration and spatial coherence.

Upon receiving a scene editing query, the module first identifies the involved target objects and checks them for potential collisions. It constructs an axis-aligned 3D grid around all objects, incorporating a fixed buffer zone to accommodate space requirements during object placement. This buffer zone helps prevent objects from being placed too closely together, reducing the likelihood of collisions.

<sup>1</sup><https://github.com/langchain-ai/langchain>

<sup>2</sup><https://github.com/pygae/clifford>

Drawing inspiration from Yang et al. [2023], our collision detection strategy utilizes a Depth-First Search (DFS) solver. This solver systematically explores various configurations to ensure that no objects overlap. Each object is described by six parameters:  $(x, y, z)$  for the center coordinates, and  $w, d, h$  for the width, depth, and height. The process begins with initial placements for the target objects, followed by systematic optimization for arranging the remaining objects.

The solver operates within a predefined time frame (e.g., 0.5 or 1 second) to explore multiple configurations. It selects the optimal arrangement based on a distance-based grading system, which prioritizes placements that keep objects as close to their original positions as possible. This approach minimizes disruptions and preserves the spatial integrity of the scene.

Once the optimal arrangement is determined, the module applies all necessary transformations to the objects, ensuring that the final scene configuration is collision-free and maintains the intended spatial relationships. This method ensures a dynamic yet coherent scene layout, enhancing the overall user experience.

## 4.4 Retrieval Augmented Generation & Caching Mechanism Integration

Our framework leverages Retrieval Augmented Generation (RAG) and a caching mechanism to enhance the efficiency and accuracy of the Large Language Model (LLM) in understanding and generating Conformal Geometric Algebra (CGA) transformations for 3D scene editing.

### 4.4.1 Retrieval Augmented Generation

RAG is utilized to improve the LLM’s comprehension of CGA by providing relevant editing queries as examples. When a new query is received, the system retrieves similar past queries from a pre-existing database. These examples serve as references to guide the LLM, allowing it to generate more accurate and contextually appropriate transformations. By leveraging past knowledge, RAG helps the LLM to better understand complex spatial relationships and transformation rules inherent in CGA.

This approach not only enhances the LLM’s performance but also reduces the computational overhead by narrowing down the search space to a set of pertinent examples.

### 4.4.2 Caching Mechanism

To further optimize the query processing, our framework employs a caching mechanism. Given that the queries are templated, this mechanism allows the system to fetch already existing queries and their corresponding transformations without needing to invoke the LLM each time. This significantly speeds up the processing time and reduces the computational resources required.

The caching mechanism operates as follows:

1. **Query Templating:** Incoming queries are converted into a standardized templated form.
2. **Cache Lookup:** The system checks if the templated query exists in the cache.
3. **Cache Retrieval:** If a match is found, the corresponding CGA transformations are retrieved from the cache.
4. **LLM Invocation:** If no match is found, the LLM processes the query to generate the required transformations, which are then stored in the cache for future use.

By integrating RAG and the caching mechanism, our framework ensures that the LLM is provided with high-quality examples, enhancing its understanding and generation of CGA transformations. Simultaneously, the caching mechanism streamlines query processing, making the system more efficient and responsive.

This dual approach not only improves the accuracy and efficiency of 3D scene editing but also ensures that the system can handle a large volume of queries with minimal latency, making it robust and scalable for real-time applications.

## 4.5 Universal Scene Description

In the construction of our 3D asset dataset, we made a strategic decision to leverage the Universal Scene Description (USD) format as the backbone for organizing and storing metadata associated with each asset. This integration of USD offers several advantages in terms of data management, organization, and interoperability.

### 4.5.1 Benefits of Using USD for Dataset Management

#### 1. Rich Metadata Representation:

- USD provides a robust framework for representing rich metadata associated with each 3D asset. This includes attributes such as object descriptions, materials, textures, scale, orientation, and more. By encoding this metadata directly into the USD files, we ensure that all relevant information is encapsulated within each asset, facilitating efficient retrieval and manipulation.

#### 2. Hierarchical Scene Composition:

- Leveraging USD’s hierarchical scene composition capabilities, we can organize our dataset in a structured manner. Each USD file represents a single asset, with the ability to nest objects within a scene hierarchy. This hierarchical structure allows for logical grouping of assets and supports the organization of complex datasets.

### 3. Layering and Versioning:

- USD’s support for layering and versioning enables us to manage different versions of assets and apply edits or variations without modifying the original files. This non-destructive workflow ensures data integrity and provides flexibility in dataset maintenance and evolution over time.

### 4. Interoperability with 3D Software:

- One of the key strengths of USD is its interoperability with a wide range of 3D software tools and pipelines. USD files can be seamlessly imported and exported across different applications, ensuring compatibility and facilitating collaboration among users with diverse workflows.

### 5. Efficient Data Compression:

- USD employs a highly efficient data compression scheme, resulting in compact file sizes without compromising on data quality. This is particularly advantageous for large-scale datasets containing a significant amount of metadata, textures, and geometric information.

## 4.5.2 Implementation Details

We leverage the Universal Scene Description (USD) format to store and organize metadata for our dataset of 3D assets. USD is an open-source, interchange file format developed by Pixar Animation Studios, designed for efficient and scalable representation of complex 3D scenes and assets.

All metadata and information associated with our 3D assets are compressed and stored within a database of USD files. These files contain structured data representations of each asset, including attributes such as geometry, materials, textures, animations, and other relevant properties. By utilizing the USD format, we can efficiently organize and manage our dataset, enabling seamless retrieval and manipulation of asset information.

USD provides significant value in our workflow, particularly during similarity searches and retrieval operations. Its optimized file structure and data organization facilitate fast and easy loading of specific attributes or metadata associated with each asset. This enables swift comparisons and processing of large volumes of data, enhancing the efficiency of our similarity search operations.

Moreover, USD’s hierarchical nature and layering capabilities offer a structured framework for storing and organizing metadata, making it convenient to navigate and locate specific information within the dataset.

By integrating USD into our 3D asset dataset management pipeline, we benefit from a versatile and scalable framework for organizing, storing, and accessing metadata associated with each asset. The rich feature set provided by USD, combined with its interoperability and efficient data compression, makes it an ideal choice for managing large-scale 3D datasets across diverse domains and applications.

# Chapter 5

## Evaluation

### 5.1 System Configuration

We employed OpenAI’s GPT-4 model, specifically the *gpt-4-1106-preview* variant, to process queries. The evaluation was done on a laptop with Ubuntu 22.04.4 LTS and AMD Ryzen™ 9 4900HS Mobile Processor (8-core/16-thread, 12MB Cache, 4.3 GHz max boost), NVIDIA® GeForce RTX™ 2060 with Max-Q Design 6GB GDDR6 and 16GB DDR4 RAM at 3200MHz. Rendering was performed using Unity™ 2022.3.9f1.

### 5.2 System-based evaluation

Our system offers several key improvements over the Holodeck framework, particularly in terms of execution speed, memory efficiency, modularity, and extendability. Below, we discuss these enhancements in detail.

#### 5.2.1 Execution Speed

Our system takes approximately 82 seconds to generate a single room while Holodeck takes approximately 142 seconds (as illustrated in Table 5.1). Our implementation demonstrates a significant advantage in the efficiency of the constraint solvers. Our system generates approximately **10 times more** solutions for object placements compared to Holodeck, highlighting the superior efficiency of our constraint-solving approach. Also, we optimized object retrieval speed by leveraging Huggingface datasets and pyVespa for efficient and fast retrieval.

#### 5.2.2 Memory Efficiency

Memory efficiency is a critical aspect of large-scale 3D scene generation. Holodeck loads all the necessary metadata, including BERT and CLIP embeddings for all 3D models in Objaverse, into memory. This approach can lead to substantial memory consumption. In contrast, our system incorporates a Huggingface dataset within

Modules	Ours	Holodeck
Room Module	5.12s	18s
Door Module	1.12s	5s
Wall Module	0.49s	0.52s
Window Module	1.12s	3.2s
Object Selector Module	25s	56s
Floor Constraint Module	27s <sup>1</sup>	26s
Wall Constraint Module	3s <sup>2</sup>	2.05s
Small Object Selector	19.8s	32s
<b>Total</b>	<b>82.65s</b>	<b>142.77s</b>

Table 5.1: Comparison of module average execution time between our system and Holodeck

the pipeline, along with a dockerized pyVespa application. This setup limits RAM usage and offloads a significant amount of information to disk, while maintaining high performance.

To evaluate the memory usage, we employed the `memory_profiler`<sup>3</sup> library in Python, collecting memory usage data at 0.1-second intervals. Throughout the generation pipeline, our system consumes at most around 4.2 GBs of RAM, whereas Holodeck consumes approximately 4.5 GBs. This gives us an edge of about **6.7%** decrease in terms of memory consumption.

### 5.2.3 Modularity and Extendability

One of the standout features of our system is its modularity. Unlike Holodeck, which has a rigid architecture that complicates extensions, our system is designed to be easily extendable. This is evidenced by the seamless integration of multiple features such as Retrieval-Augmented Generation (RAG), feedback modules, and reference image functionalities. Our modular architecture allows for the straightforward addition of new components and enhancements, making it a more versatile and adaptable framework.

### 5.2.4 3D Models Dataset Enhancement

Our system is built on top of the ThreeDWorld (TDW) framework, which provides a streamlined process for converting 3D models to a TDW-compatible format. This ease of conversion facilitates the expansion of our 3D models dataset. In contrast, Holodeck is built on AI2THOR Kolve et al. [2017], which lacks comprehensive documentation or support for creating compatible assets. This limitation restricts the flexibility and scalability of Holodeck in terms of incorporating new 3D models.

<sup>3</sup>[https://github.com/pythonprofilers/memory\\_profiler](https://github.com/pythonprofilers/memory_profiler)



### 5.2.5 Advanced Features and Rendering Quality

TDW offers a broader range of features and superior rendering quality compared to AI2THOR. This includes advanced physics simulations, higher fidelity visual rendering, and more detailed environmental interactions. These enhancements contribute to the creation of more realistic and visually appealing 3D scenes in our system (as illustrated in Figure 5.1).



Figure 5.1: Comparison of scene snapshots: Our system (left) within TDW and Holodeck (right) within AI2THOR.

### 5.2.6 Summary

In summary, our system provides a faster, more memory-efficient, modular, and extendable solution for 3D scene generation compared to Holodeck. The use of TDW further enhances the capabilities of our framework, allowing for easy expansion of the 3D models dataset and superior rendering quality. These improvements collectively demonstrate the robust and adaptable nature of our system, positioning it as a significant advancement in the field of 3D scene generation and editing.

## 5.3 3D Scene Generation

The 3D scene generation component of our framework closely follows the methodology presented in Yang et al. [2023], achieving comparable results. Therefore, the evaluation focused primarily on the novel features introduced in our pipeline, including Retrieval Augmented Generation (RAG), reference image and feedback integration, as well as system implementation.

### 5.3.1 Experimental Setup

This evaluation involved five participants who assessed unique 150 generated scenes—50 generated with feedback, 50 with a reference image, and 50 with Retrieval Augmented Generation (RAG). Each scene was evaluated using top-down views and a 360-degree display, comparing them to baseline scenes generated using the approach described in Yang et al. [2023]. The scenes were categorized into three main types: *living rooms*, *kitchens*, and *operating rooms*. Additionally, iterative generations based on feedback were limited to five iterations.

### 5.3.2 Performance Metrics

In evaluating the performance of our framework, we employed different metrics tailored to the specific capabilities of the RAG, feedback module, and reference image components. These metrics were designed to capture the effectiveness and user satisfaction with the generated scenes.

#### 5.3.2.1 RAG and Feedback Module

For both the Retrieval-Augmented Generation (RAG) and feedback module components, the primary performance metric was participant preference. This metric involved a comparative analysis where participants were presented with scenes generated by our framework and those generated by the baseline approach described in Yang et al. [2023]. Participants assessed 150 generated scenes, divided into three categories: *living rooms*, *kitchens*, and *operating rooms*. For each category, 50 scenes were generated using feedback, 50 using RAG, and 50 using the baseline approach.

Participants were shown top-down views and 360-degree displays of each scene and asked to choose their preferred scene.

#### 5.3.2.2 Reference Image

For the reference image component, the performance metric focused on resemblance. Participants were provided with a reference image and the corresponding scene generated by our framework. They were asked a simple yes or no question: "Does the generated scene resemble the reference image?" This binary metric allowed us to quantify the accuracy of our framework in replicating specific visual characteristics and layouts from the reference images.

### 5.3.3 Results & Discussion

In this section, we present the results of our evaluation and discuss the findings for the three components of our 3D scene generation framework: feedback module, RAG, and reference image.

### 5.3.3.1 Feedback Module

The feedback module was designed to enhance the scene generation process by iteratively refining scenes based on a feedback agent. To evaluate its effectiveness, participants were asked to choose between scenes generated using our feedback module and those generated by the baseline approach from Yang et al. [2023]. As

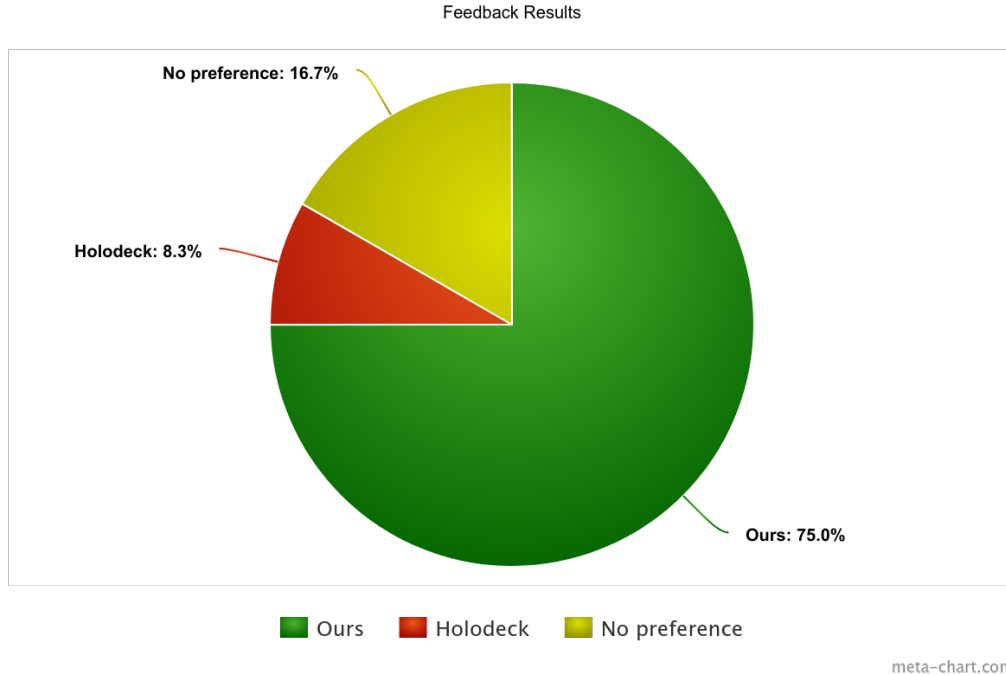


Figure 5.2: Results from participants’ choice between Holodeck and our generated scenes based on feedback.

depicted in Figure 5.2, the results showed a strong preference for our feedback-generated scenes, with 75% of participants favoring them over the baseline scenes. This significant preference indicates that the iterative refinement process enabled by the feedback module successfully enhances the quality and desirability of the generated scenes. The feedback mechanism allows the model to better align the generated scenes with design expectations laid out by the Multimodal Feedback agent, leading to more satisfying outcomes.

### 5.3.3.2 Retrieval-Augmented Generation (RAG)

The RAG component was evaluated by comparing scenes generated using RAG with those from the baseline approach. Participants were presented with 50 scenes generated by RAG and 50 baseline scenes across three categories: living rooms, kitchens, and operating rooms. They were asked to select their preferred scene.

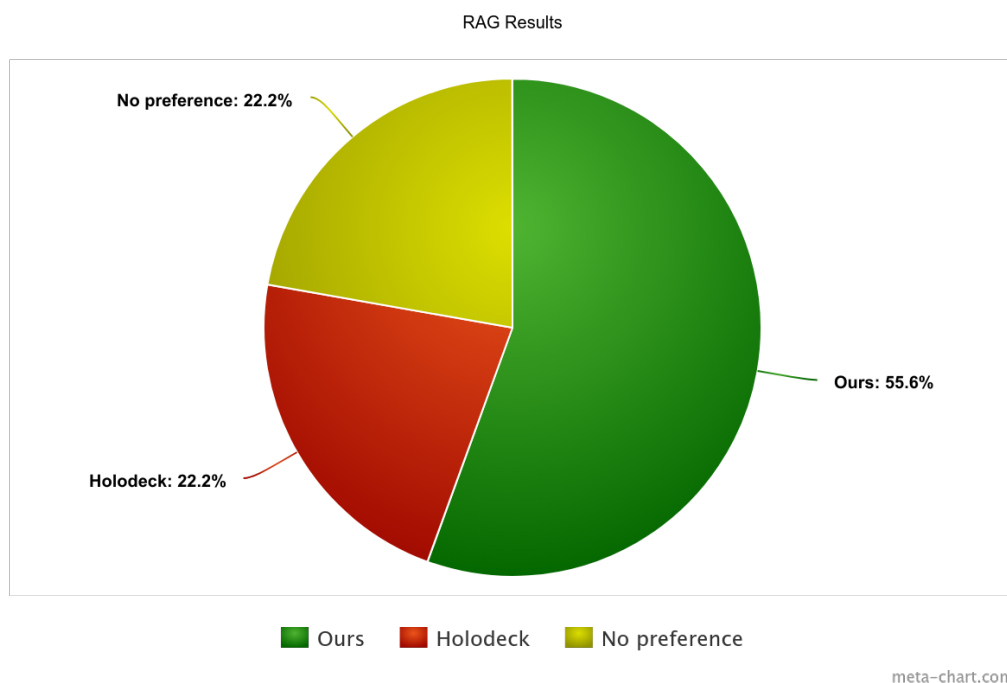


Figure 5.3: Results from participants' choice between Holodeck and our RAG-generated scenes.

The results, illustrated in Figure 5.3, revealed that 55.6% of participants preferred the RAG-generated scenes over the baseline. Although the preference for RAG-generated scenes is not as pronounced as for the feedback module, it still indicates a notable enhancement in scene quality. The use of RAG allows the generation process to incorporate a wider range of relevant information, leading to more contextually rich and accurate scenes. However, the relatively lower preference compared to the feedback module suggests that while RAG improves scene generation, there is still room for further refinement to better meet user expectations.

### 5.3.3.3 Reference Image

The reference image component was assessed based on the participants' agreement on whether the generated scenes resembled the provided reference images. Participants were shown a reference image and the corresponding generated scene and were asked to answer yes or no to whether the generated scene accurately resembled the reference image.

As shown in Figure 5.4, 83.3% of participants agreed that the generated scenes resembled the reference images. This high level of agreement underscores the effectiveness of our framework in capturing and replicating the visual and spatial characteristics of reference images. The ability to accurately generate scenes that

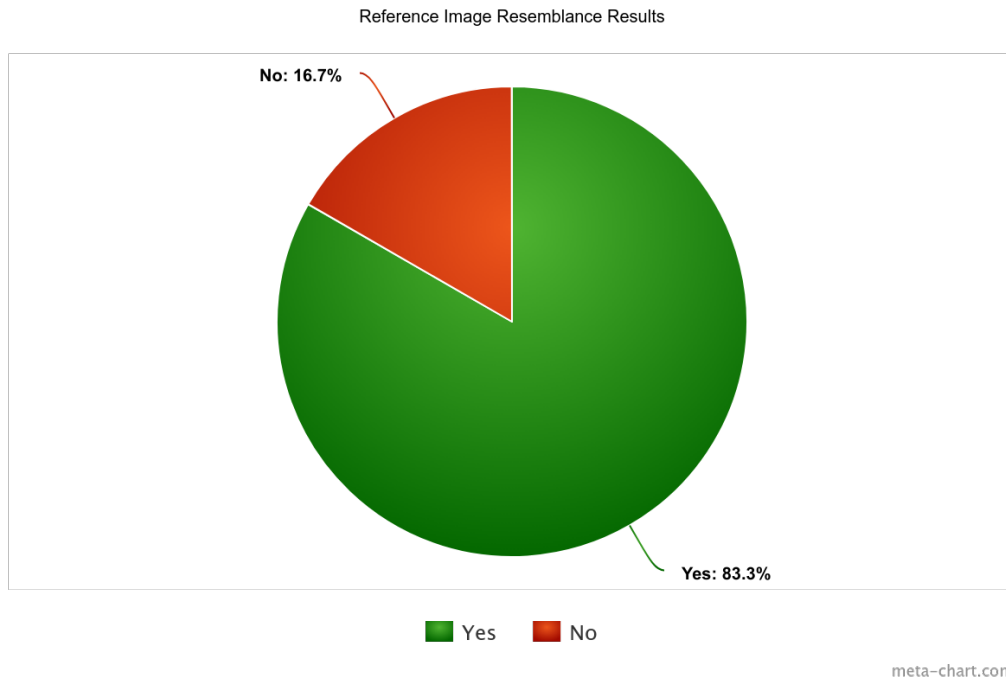


Figure 5.4: Results of participants’ agreement on resemblance between reference images and generated scenes.

closely match reference images is crucial for applications requiring precise visual fidelity, such as interior design and virtual staging.

#### 5.3.3.4 Discussion

The results from the evaluations provide several key insights into the performance and user satisfaction with our framework:

1. **Feedback Module:** The significant preference for feedback-generated scenes highlights the importance of iterative refinement in scene generation. By continuously incorporating feedback, the framework can produce scenes that more closely align with the expectations, resulting in higher satisfaction.

2. **RAG:** While the preference for RAG-generated scenes is positive, it is less pronounced than for the feedback module. This suggests that while RAG enhances the generation process by incorporating relevant information, it may not always capture the nuanced preferences of users as effectively as iterative feedback.

3. **Reference Image:** The high agreement rate for the reference image component demonstrates the framework’s capability to accurately replicate specific visual inputs. This ability is particularly valuable for applications where visual fidelity to a reference image is critical.

Overall, our framework demonstrates substantial improvements over the baseline, with notable enhancements in user satisfaction and scene quality across all

components. The combination of feedback, RAG, and reference image capabilities allows for versatile and robust 3D scene generation, catering to a wide range of user needs and preferences.

### 5.3.3.5 Examples of Generated Scenes



Figure 5.5: On the left the scene was generated using the query *a living room* while on the right the prompt *a wine cellar*

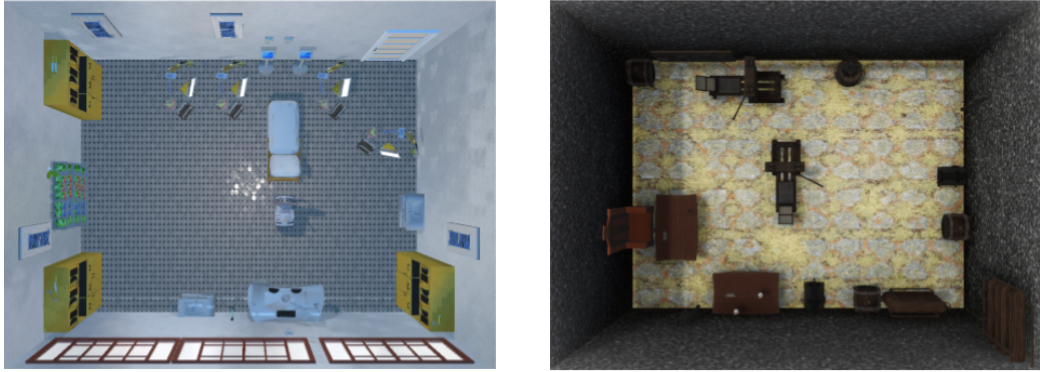


Figure 5.6: On the left the scene was generated using the query *an operating room* while on the right the prompt *a medieval dungeon*



Figure 5.7: On the left the scene was generated using the query *a library room* while on the right the prompt *a warehouse*



Figure 5.8: On the left we can see the picture of a living room provided as reference for the generation and on the right the final generated scene.



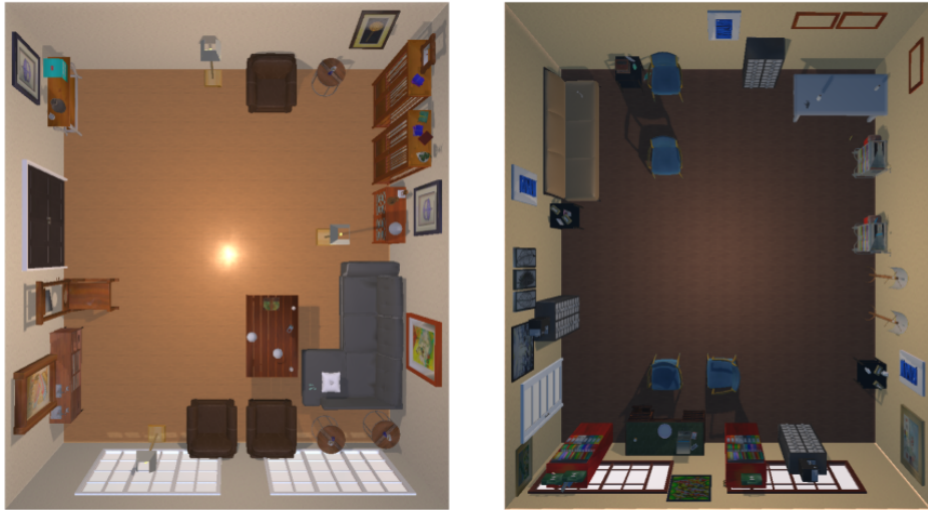


Figure 5.9: Two scenes generated using RAG. On the left the query is *A living room* and on the right *A doctor's office*.



Figure 5.10: On the left we can see the picture of the initially generated living room and on the right the final generated living room based on the feedback generated from our Feedback Module.

In Figure 5.10 we can see the generation of a scene based on the following feedback:

The overall design lacks cohesion and could benefit from a more thoughtful arrangement. The sectional sofa is quite large but appears to be missing from the images provided. The current sofa in the images does not facilitate conversation and feels disconnected from the other elements. The coffee table placement does not effectively complement the seating arrangement. The bookshelf and console table are suitably placed, but the room could be better organized to create a more inviting space. The wall art is appropriately placed, adding aesthetic value, but could be better aligned with the furniture. The placement of floor lamps seems arbitrary and could be positioned to better define different areas of the room. The side table placement next to the accent chairs is not shown, but their purpose should be to enhance function and aesthetics. The ottoman can be used as a functional piece between seating to improve utility and design. To improve the room, ensure the sofa is central to the layout, the coffee table is within easy reach of all seating, and ambient lighting is thoughtfully positioned. The design elements should reflect a warm and inviting atmosphere, similar to the reference image, which shows a well-coordinated space with harmonious furniture placement, clear walking paths, and a balanced look.

## 5.4 3D Scene Editing

To evaluate our framework’s capability in object repositioning within 3D scenes, we conducted comprehensive human evaluations involving 20 participants: 5 3D designers, 5 game programmers, and 10 individuals from non-gaming and non-3D design backgrounds. This study focused on our framework’s effectiveness in editing a variety of scenes.

Our findings from these user studies demonstrate that our framework, leveraging Conformal Geometric Algebra (CGA) in its prompting strategies, significantly outperforms baseline alternatives, including those created with NVIDIA Omniverse (see Section 5.4.2.1).

### 5.4.1 Experimental Setup

For human evaluation, we generated ten diverse scenes, encompassing various settings including living rooms, wine cellars, kitchens, and medical operating rooms.

For each scene, we created five variations of templated queries, populating them with objects randomly selected from within the scene. This resulted in 50 distinct queries per scene, amounting to a total of 2,500 scenes (with corresponding prompts) for human assessment. Each query was used to compare the initial scene with the resulting scene after processing by the two baseline systems and our framework.

For each edited scene, we displayed top-down view images and 360-degree video views, asking annotators to assess the accuracy of the editing performed. Each object repositioning query was evaluated by five annotators, with a result considered valid only if all annotators unanimously agreed on the assessment. The evaluation queries are displayed on the y-axis of Figure 5.12.

It is important to note that all preliminary experiments on dilation-related queries with our framework and other baselines achieved a perfect success rate, leading us to omit these queries. The queries are categorized into five groups, each containing ten queries that evaluate the system’s performance across a progressively increasing difficulty gradient. The group categories used in our analysis are detailed below:

- **Simple Queries:** Basic actions such as rotations and movements along specified axes or planes of a single object.
- **Compositional Queries:** Combinations of relative movements and rotations among two or more objects.
- **Fuzzy Queries:** Interpretation and execution of spatial and orientation-specific actions, such as proximity adjustments and directional alignments.
- **Compositional Fuzzy Queries:** Multiple elements combining aspects from both compositional and fuzzy queries.

- **Hard Queries:** The most challenging scenarios, testing the limits of each system’s processing capabilities.

## 5.4.2 Benchmarking

In this section, we assess the performance of our system compared to established baselines. In the following, we provide an overview of the baseline systems against which our solution is measured, the performance metrics that form the criteria for comparison, and a detailed discussion of the results.

### 5.4.2.1 Baseline Systems

For comparison, one of our baseline systems utilizes a publicly available prompt from NVIDIA’s Omniverse<sup>4</sup>. This prompt generates a JSON output detailing object placements within a 3D space, similar to our system. Unlike our framework, the Omniverse prompt does not impose constraints on the reasoning scheme the LLM should follow, requesting only the final positions of the objects. It accepts input specifications for each object, including its name, dimensions along the  $X$ ,  $Y$ , and  $Z$  axes, and a centrally located origin point.

To ensure comparability with our system, we include only points of interest relevant to the scene editing query, rather than the entire scene description as initially done. This refined input approach significantly impacts the LLM’s response time. Our experiments demonstrate that providing only the relevant points of interest leads to a substantial decrease in response time—specifically, an average of  $3.3 \pm 0.1$  times faster—without compromising overall accuracy. Due to these findings, we report results exclusively from this optimized methodology. Finally, we extended the prompt by incorporating Euler angles for object orientation.

Our second baseline model is inspired by recent work showing that LLMs can predict a dense sequence of end-effector poses for manipulation tasks Kwon et al. [2024]. We extend this concept to object repositioning in 3D scenes, treating it as a form of zero-shot trajectory generation. In this context, we task the LLM with constructing the necessary translation and rotation matrices to define the final trajectory, adapting the underlying techniques to suit scene manipulation objectives. This baseline model closely aligns with our approach, as it guides the LLM to operate using templated functions corresponding to translation and rotation functions within the Euclidean space. This method parallels our use of structured prompts that direct the LLM to generate specific geometric transformations necessary for scene editing tasks.

To ensure a fair comparison, we appended five specific examples to both the baseline prompts and the final guidelines (refer to Figure A.8). This inclusion is based on findings that demonstrated significant benefits in enhancing the LLM’s

---

<sup>4</sup><https://github.com/NVIDIA-Omniverse/kit-extension-sample-airoomgenerator/blob/main/exts/omni.example.airoomgenerator/omni/example/airoomgenerator/prompts.py>

effectiveness in performing scene editing tasks. The exact baseline prompts can be found in the *Appendices*.

#### 5.4.2.2 Performance Metrics

To evaluate the system’s performance, we calculated the success rate per query across the ten different scenes, each with five templated variations. The success rate  $S$  for each scene editing query is calculated as follows:

$$S = \frac{1}{M} \sum_{i=1}^M \chi_{\text{correct}}(i)$$

where  $\chi_{\text{correct}}(i)$  is the characteristic function that equals 1 if the editing query was performed correctly for the  $i$ -th query, and 0 otherwise. Here,  $M$  represents the total number of queries, calculated as  $M = N \times k$ , where  $N$  is the number of different scenes considered (10 in our case), and  $k$  is the number of variations per scene (5 in our case).

Similarly, we compared the average response time across the ten different scenes and their respective variations. The average response time  $T$  is calculated by:

$$T = \frac{1}{M} \sum_{i=1}^M t_i$$

where  $t_i$  represents the response time for the  $i$ -th query.

#### 5.4.2.3 Results & Discussion

In this section, we present a comprehensive analysis that spans overall performance metrics, detailed evaluations by query group, and granular analyses of individual query performances. These discussions aim to highlight significant findings, interpret the implications of the results, and explore potential areas for further improvement.

In all reported figures, we refer to the modified baseline prompts for scene editing as *Omniverse* and *Euclidean*, respectively derived from NVIDIA’s Omniverse usage examples and methods akin to zero-shot trajectory generation. We name the latter *Euclidean* because it closely aligns with our method’s approach in guiding the LLM to generate transformations such as rotations and translations within Euclidean space, in contrast to our use of CGA. Finally, we refer to our framework’s prompt as *CGA*.

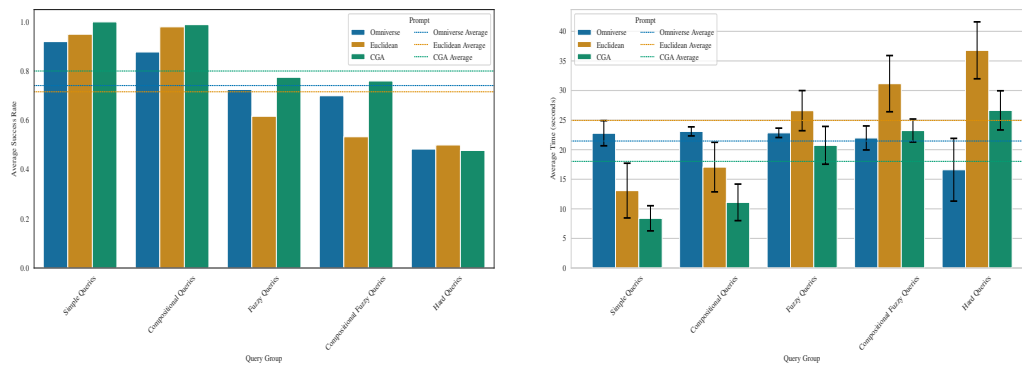


Figure 5.11: The left chart compares the average success rates, and the right chart compares the average LLM response times for three different systems—Omniverse, Euclidean, and CGA—across five categories of queries: Simple Queries, Compositional Queries, Fuzzy Queries, Compositional Fuzzy Queries, and Hard Queries. Success rates are measured on a scale from 0.0 to 1.0. Horizontal dashed lines indicate the overall mean recall and response times for each system.

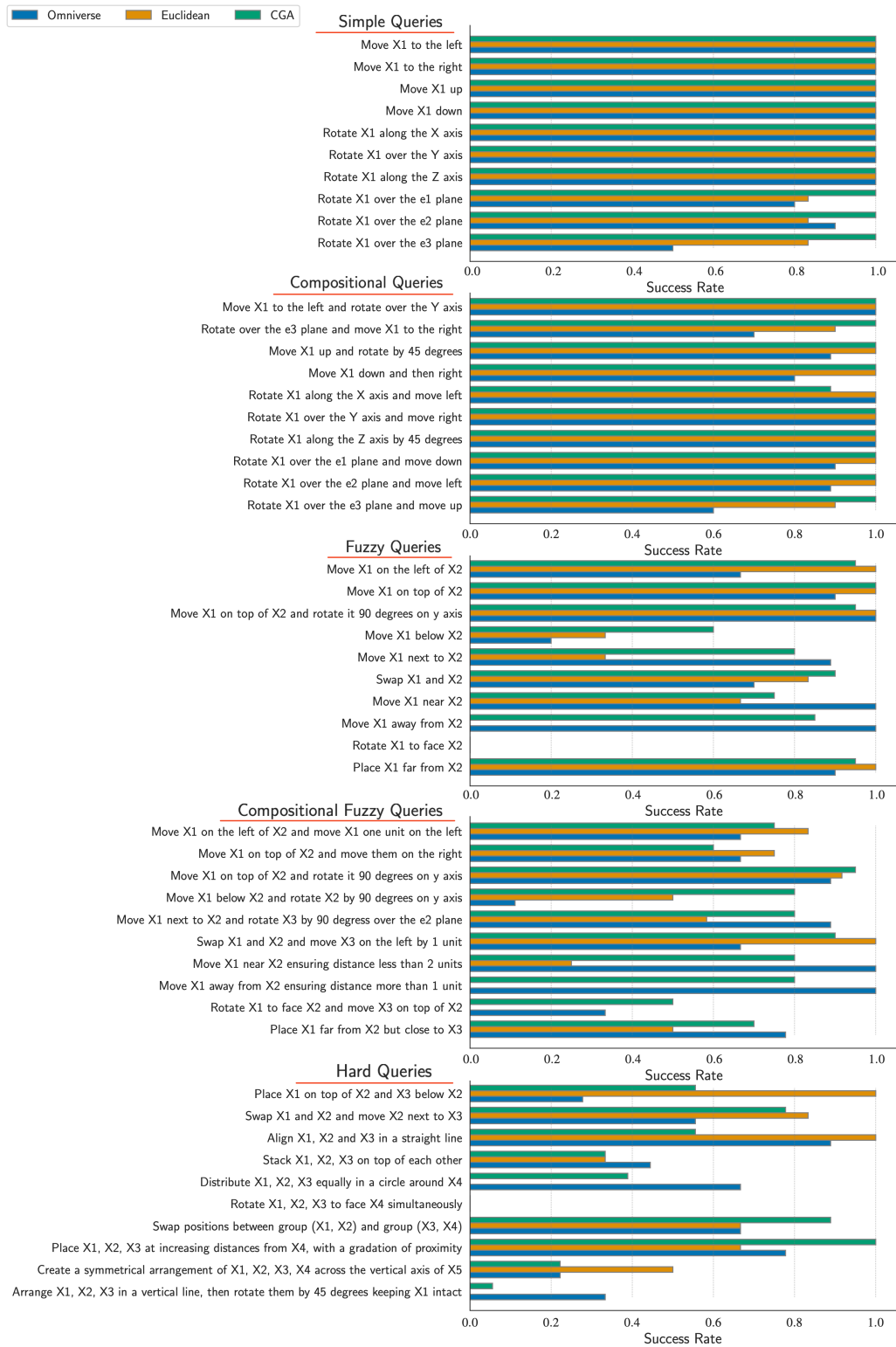


Figure 5.12: Query resolution performance for different prompts—Omniverse, Euclidean, and CGA—across five query categories: Simple, Compositional, Fuzzy, Compositional Fuzzy, and Hard. Success rate is assessed for each templated query over ten different scenes, with five query variations per scene.

**Performance Analysis by Query Group** Figure 5.11 presents a detailed evaluation of the prompts’ performance across different groups of queries. Each group’s results are analyzed to highlight specific strengths and weaknesses of the system in handling varying complexities. Specifically, the effectiveness and efficiency of the CGA, Omniverse, and Euclidean prompts were evaluated across the five categories of queries: Simple, Compositional, Fuzzy, Compositional Fuzzy, and Hard. We focused on two key performance metrics: average success rate and average response time, providing insights into each system’s scene editing capabilities.

Overall, CGA achieves the highest average success rate of 0.80 and the lowest average response time of 18 seconds ( $p < 0.05$ ), with the results being statistically significant. Interestingly, all prompts exhibited the same overall standard deviation, both in terms of success rate and response time, indicating consistent variability across the different editors. To assess the differences between the systems, we performed t-tests. In comparison, the Omniverse and Euclidean prompts achieve success rates of 0.74 and 0.72, and average response times of 21.5 seconds and 24.5 seconds, respectively. While the differences in success rates among Omniverse and Euclidean prompts are not statistically significant, the variations in average response times are.

We conjecture that the lack of statistical significance in terms of success rates among the systems can be attributed to the nature of the reasoning methods employed. Both the Omniverse and Euclidean prompts rely on classical geometrical reasoning, which may lead to similar levels of performance. In contrast, our framework utilizes a different approach to reasoning, which not only reflects in its superior performance but also in its statistical significance. This suggests that the distinct reasoning method employed by our system may contribute to its enhanced effectiveness.

Focusing on the success rate metric, CGA consistently exhibits high success rates across all query types, particularly excelling in complex scenarios such as Compositional Fuzzy queries, indicating robust scene understanding. Given that the success rates of the Euclidean and Omniverse systems are not statistically significant, it is evident that CGA provides a relative boost of 9.6% over their average performance. In contrast, Omniverse demonstrates variable performance, experiencing a drop in simpler queries but excelling in fuzzy queries. Conversely, the Euclidean system performs well in simpler queries but shows significant weaknesses in both simple and complex scenarios. CGA appears to integrate the advantages of both systems, performing exceptionally well across simple, fuzzy, and compositional queries. However, it is important to note that with all prompts, ChatGPT-4 shows average performance on hard queries, suggesting potential deficiencies in spatial reasoning.

Regarding response time, Omniverse exhibits consistent average response times across various query groups, regardless of query difficulty, with the notable exception of hard queries, which show the slowest response times. Importantly, CGA demonstrates a 16% relative decrease in response time compared to Omniverse, which is the fastest among the baseline systems. In contrast, the Euclidean prompt



demonstrates an increasing trend in response times correlating with query difficulty, peaking with fuzzy, compositional fuzzy, and hard queries. This trend is noteworthy as it suggests that response times increase as query complexity rises. The Euclidean prompt consistently records the longest response times across the complex queries, posing challenges in time-sensitive applications. Meanwhile, the CGA prompt also displays an increasing trend in response times with escalating query difficulty. Notably, it presents slower response times for simple and simple compositional queries despite achieving a 100% success rate. Overall, CGA maintains competitive response times suitable for practical applications, with its response times scaling appropriately with the complexity of queries and achieving the best overall response performance.

**Detailed Performance by Individual Query** Figure 5.12 delves into the performance metrics for each individual query within the groups. This detailed breakdown provides insights into the prompts’ consistency, efficiency, and accuracy in executing scene editing tasks with varying degrees of complexity, facilitating a granular understanding of their operational effectiveness.

Regarding the Simple Queries, all systems generally exhibit high success rates for straightforward tasks, even in geometrically nuanced rotations such as those over specific planes, e.g., *Rotate X1 over the e1 plane*, with CGA consistently achieving perfect scores. This highlights CGA’s superior handling of precise geometric transformations.

For the Compositional Queries, which involve a combination of movements and rotations, CGA and Euclidean perform robustly, indicating effective integration of complex instructions. Omniverse, however, shows variability, particularly underperforming in scenarios like *Rotate over the e3 plane and move X1 to the right*, where it achieves a 0.7 success rate compared to 1.0 by CGA.

In dealing with spatially ambiguous commands, i.e., Fuzzy Queries, Omniverse and CGA outperform Euclidean, especially notable in queries like *Move X1 away from X2* where Euclidean drops to 0 against Omniverse’s perfect score. Interestingly, the Omniverse prompt struggles with the seemingly simple query, *Move X1 on the left of X2*, displaying a stark contrast to the near-perfect performance of the other systems. Although Omniverse is not presented with a specific *move below* example (only *on top* is provided in their prompts), all systems show generalization capabilities, with CGA achieving the highest success rate of 0.6. Similar generalization is observed with the *Swap X1 and X2* query, where all systems are expected to perform well given their operational logic at the coordinate level; however, Omniverse shows surprisingly lower performance. In contrast, while the Euclidean and CGA systems perform less effectively on the *Move X1 near X2* query, Omniverse excels with perfect success rates. Lastly, all prompts consistently fail the *Rotate X1 to face X2* query. Intriguingly, despite being equipped with Euler angles, Omniverse also fails, suggesting a significant challenge area for current scene editing technologies.

The Compositional Fuzzy Queries category presents a notable challenge, combining fuzzy directives with multiple actions. In this category, Omniverse and CGA show comparable performances, indicating their effective handling of combined directives. However, it is important to note discrepancies in the Euclidean system’s performance. While it achieves good results on individual components of these queries, it performs poorly when actions are combined, as seen in queries like *Move X1 next to X2 and rotate X3 by 90 degrees over the e2 plane* and *Swap X1 and X2 and move X3 on the left by 1 unit*. This inconsistency highlights potential limitations in the Euclidean prompt’s ability to effectively integrate multiple spatial manipulations within a single query.

All systems exhibit challenges with the most demanding Hard Queries, which require intricate arrangements and precise manipulations. Notably, the Euclidean prompt displays extremely inconsistent behavior: it achieves perfect success rates on complex queries such as *Place X1 on top of X2 and X3 below X2* and *Align X1, X2 and X3 in a straight line*, where other systems face difficulties. Conversely, it presents a zero success rate on tasks like *Distribute X1, X2, X3 equally in a circle around X4* or *Arrange X1, X2, X3 in a vertical line, then rotate them by 45 degrees keeping X1 intact*, where other systems manage average performances. This variability suggests that while the Euclidean prompt excels in certain types of spatial reasoning, it may lack robustness in scenarios requiring dynamic spatial transformations or uniform object distribution.

These findings underscore the need for further optimization and testing of the Euclidean system to enhance its consistency across a broader range of complex scene editing tasks. In summary, while CGA emerges as the most capable across a broad range of tasks, challenges remain, particularly in Hard and Compositional Fuzzy queries, underscoring the need for further advancements in LLM reasoning capabilities.

### 5.4.3 Scene Editing Examples

Table 5.2 and 5.3 present qualitative results that enhance understanding of the user experience quality, illuminate the evaluation scheme, and clarify the advantages and disadvantages of different prompts. As detailed in Section 5.4 of the thesis, each object repositioning query is evaluated by five annotators. A result is considered valid only if there is unanimous agreement among all annotators on the assessment. The first column of Table 5.2 displays the performance of different prompts—CGA, Euclidean, and Omniverse—respectively, for the query *move ‘chair 1’ away from the table*. Our annotators unanimously agreed that the Omniverse prompt accurately aligns with the requested query. The CGA prompt correctly interprets the query but leaves the chair relatively close to the table, which is not considered a correct result. In contrast, the Euclidean prompt does not result in any transformation. In the second column, for the second prompt, i.e., *Rotate X1 over the e3 plane and move up*, both CGA and Euclidean adeptly rotate and elevate a book on the top right side of the bookcase, while Omniverse

incorrectly rotates it along the wrong axis. Finally, in the third column of Table 5.2, both CGA and Omniverse successfully relocate the patient table, while Euclidean fails to do so. It can be observed that the Euclidean prompt either fails to alter object relations significantly or misinterprets the queries, even simple ones.

In the first query of Table 5.3, both CGA and Euclidean accurately position the orange bottle and the vase. In contrast, Omniverse misinterprets the z-axis as the y-axis, resulting in the vase being placed behind the table. In the second column of Table 5.3, when examining a wine cellar scene, only CGA accurately repositions a mask frame below a city frame. Conversely, the Euclidean approach incorrectly places it adjacent to the other frame, while Omniverse incorrectly identifies the correct axis for movement. Last but not least, we see that for the last query on Table 5.3, i.e., *move 'barrel 1' far from 'barrel 2' but close to 'barrel 3'*, both Euclidean and Omniverse do not provide appropriate solutions. Importantly, CGA provides a valid solution.













Query	Move 'chair 1' away from table	Rotate 'book 1' over e3 plane and move up	Move 'tools table' away from 'patient' ensuring a distance of at least 1 unit
Initial Scene			
CGA			
Euclidean			
Omniverse			

Table 5.2: Results across three different scenes for each prompt and for each system.













Query	Place 'orange bottle' on top of 'tools table' and 'vase' below 'tools table'	Move 'mask frame' below 'city frame'	Move 'barrel 1' far from 'barrel 2' but close to 'barrel 3'
Initial Scene			
CGA			
Euclidean			
Omniverse			

Table 5.3: Results across three different scenes for each prompt and for each system.



# Chapter 6

## Conclusion

### 6.1 Summary

The objectives of this thesis were to develop a comprehensive framework with the following key features:

1. **Modular and User-Friendly Text-to-3D Scene Generation:** The framework aims to provide a highly modular and user-friendly component for generating 3D scenes from textual descriptions. This component includes extensive features such as:
  - **Retrieval Augmented Generation (RAG):** Enhancing the generation process by retrieving relevant examples to guide the LLM.
  - **Reference Image Integration:** Allowing users to provide reference images that the system can use to incorporate additional context and detail into the generated scenes.
  - **Feedback-Based Iterative Generation:** Enabling iterative improvements to the generated scenes based on user feedback or self-evaluating LLM agents, refining the results through multiple iterations.
2. **Novel 3D Scene Editing Component:** This component integrates Conformal Geometric Algebra (CGA) with Large Language Models (LLMs) to facilitate advanced scene editing capabilities. The editing component is further enhanced by:
  - **Retrieval Augmented Generation (RAG):** Similar to the generation component, RAG is used to retrieve relevant editing queries, providing examples that improve the LLM's understanding and execution of scene edits.
  - **Caching Mechanism:** To optimize performance, a caching mechanism is implemented. This mechanism stores templated queries and their corresponding results, allowing the system to quickly fetch and apply pre-existing queries without requiring real-time processing by the LLM.

## 6.2 Limitations & Future Work

### 6.2.1 3D Scene Generation

While the 3D scene generation component of our framework offers significant advancements and features, several limitations need to be acknowledged:

1. **Dependency on Pre-Existing Data:** The quality and variety of generated scenes are inherently dependent on the pre-existing dataset of 3D assets. Limitations in the dataset can directly affect the richness and accuracy of the generated scenes. Future work will focus on creating a more diverse dataset that includes a wide range of 3D models to cover a greater variety of possible scenes.
2. **Evaluation Metrics:** The evaluation of generated scenes is inherently subjective. While human evaluations provide valuable insights, they can be influenced by individual biases and preferences. To address this, we plan to develop more objective and standardized evaluation metrics. For instance, we envision using CLIP to compare generated scenes against baseline scenes or employing multimodal agents to assess the quality of generations.
3. **Constraint Solver:** Currently, constraints between objects are generated in a templated format and solved using a depth-first search (DFS) solver. This approach can be time-consuming, may not always produce the best results, and can be complex. Future work will explore more efficient optimization solvers and "smarter" solutions, such as agent-based placement strategies.

Addressing these limitations will guide future research and development efforts to enhance the capabilities and applicability of the 3D scene generation component.

### 6.2.2 3D Scene Editing

While the 3D scene editing component offers innovative features and significant advancements, several limitations should be considered:

1. **Object Representations:** Currently, our system utilizes multiple intermediate representations of objects, including textual, templated, and bounding box references, which require exact object names. To enhance this, we plan to implement semantic similarity measures at the token level or more advanced similarity searches using distributed representations. Additionally, enriching the system with further object information, such as orientation details, will improve its ability to handle sophisticated queries involving relative rotations. By improving the system's grasp of spatial relationships, we expect better performance in complex scenarios.



2. **System Optimizations:** We envision transforming the collision module into a multi-modal agent, enabling nuanced handling of complex scenarios. This agent could be provided with a top-down view of the scene to facilitate finding better resolutions. Lastly, we plan to integrate speech interaction capabilities into our system, making it more user-friendly and accessible.
3. **Spatial Reasoning:** Our work sheds light on the spatial reasoning capabilities and limitations of LLMs. Although better and curated prompting could improve performance on more challenging queries, we conjecture that more specialized agents are needed. Future work will explore multimodal alternatives that incorporate top-down views of the scene along with linguistic queries or LLMs fine-tuned on geometric algebra (GA) Wang et al. [2023]. This approach aims to enhance the spatial reasoning and manipulation capabilities of the system.

Addressing these limitations and pursuing the outlined future work will significantly advance the state-of-the-art in both 3D scene generation and editing, making these systems more robust, efficient, and versatile.



# Bibliography

- Dhruv Batra, Angel X. Chang, Sonia Chernova, Andrew J. Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Motlaghi, Manolis Savva, and Hao Su. Rearrangement: A challenge for embodied ai, 2020.
- Eduardo Bayro-Corrochano, Luis Lechuga-Gutiérrez, and Marcela Garza-Burgos. Geometric techniques for robotics and hmi: Interpolation and haptics in conformal geometric algebra and control using quaternion spike neural networks. *Robotics and Autonomous Systems*, 104:72–84, 2018. ISSN 0921-8890. doi: <https://doi.org/10.1016/j.robot.2018.02.015>. URL <https://www.sciencedirect.com/science/article/pii/S0921889017303317>.
- Eduardo Jose Bayro-Corrochano. Geometric neural computing. *IEEE Transactions on Neural Networks*, 12(5):968–986, 2001.
- Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kudipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R’e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram

- Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *ArXiv*, 2021. URL <https://crfm.stanford.edu/assets/report.pdf>.
- Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K Gupta. Clifford neural layers for pde modeling. *arXiv preprint arXiv:2209.04934*, 2022.
- Johann Brehmer, Pim De Haan, Sönke Behrends, and Taco S Cohen. Geometric algebra transformer. *Advances in Neural Information Processing Systems*, 36, 2024.
- Sven Buchholz. Quaternionic spinor mlp. In *Proc. European Symposium on Artificial Neural Networks, 2000*, pages 377–382, 2000.
- Sven Buchholz and Nicolas Le Bihan. Polarized signal classification by complex and quaternionic multi-layer perceptrons. *International journal of neural systems*, 18(02):75–85, 2008.
- Sven Buchholz and Gerald Sommer. Clifford algebra multilayer perceptrons. In *Geometric Computing with Clifford Algebras: Theoretical Foundations and Applications in Computer Vision and Robotics*, pages 315–334. Springer, 2001.
- Sven Buchholz and Gerald Sommer. On clifford neurons and clifford multi-layer perceptrons. *Neural Networks*, 21(7):925–935, 2008.
- Sven Buchholz, Kanta Tachibana, and Eckhard MS Hitzer. Optimal learning rates for clifford neurons. In *Artificial Neural Networks–ICANN 2007: 17th International Conference, Porto, Portugal, September 9–13, 2007, Proceedings, Part I 17*, pages 864–873. Springer, 2007.
- Angel Chang, Manolis Savva, and Christopher D Manning. Interactive learning of spatial knowledge for text to 3d scene generation. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 14–21, 2014.
- Angel X Chang, Mihail Eric, Manolis Savva, and Christopher D Manning. Scene-seer: 3d scene design with natural language. *arXiv preprint arXiv:1703.00050*, 2017.
- Michael Chang, Alyssa Li Dayan, Franziska Meier, Thomas L. Griffiths, Sergey Levine, and Amy Zhang. Hierarchical abstraction for combinatorial generalization in object rearrangement. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=fGG6vHp3W9W>.

- Yu Cheng, Yan Shi, Zhiyong Sun, Dezhi Feng, and Lixin Dong. An interactive scene generation using natural language. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6957–6963. IEEE, 2019.
- E Bayro Corrochano, Sven Buchholz, and Gerald Sommer. Selforganizing clifford neural network. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 120–125. IEEE, 1996.
- Fernanda De La Torre, Cathy Mengying Fang, Han Huang, Andrzej Banburski-Fahey, Judith Amores Fernandez, and Jaron Lanier. Llmr: Real-time prompting of interactive worlds using large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. ACM, 2024.
- Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023.
- Zane Durante, Bidipta Sarkar, Ran Gong, Rohan Taori, Yusuke Noda, Paul Tang, Ehsan Adeli, Shrinidhi Kowshika Lakshmikanth, Kevin Schulman, Arnold Milstein, Demetri Terzopoulos, Ade Famoti, Noboru Kuno, Ashley Llorens, Hoi Vo, Katsu Ikeuchi, Li Fei-Fei, Jianfeng Gao, Naoki Wake, and Qiuyuan Huang. An interactive agent foundation model, 2024.
- Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang (Lorraine) Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Roman Le Bras, Jena Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 70293–70332. Curran Associates, Inc., 2023. URL [https://proceedings.neurips.cc/paper\\_files/paper/2023/file/deb3c28192f979302c157cb653c15e90-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2023/file/deb3c28192f979302c157cb653c15e90-Paper-Conference.pdf).
- 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. *arXiv preprint arXiv:2305.15393*, 2023.
- Rafail Fridman, Amit Abecasis, Yoni Kasten, and Tali Dekel. Scenescape: Text-driven consistent scene generation. *arXiv preprint arXiv:2302.01133*, 2023.
- Huan Fu, Bowen Cai, Lin Gao, Ling-Xiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao, et al. 3d-front: 3d furnished rooms with layouts and semantics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10933–10942, 2021.

- Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, Kuno Kim, Elias Wang, Michael Lingelbach, Aidan Curtis, Kevin Tyler Feigelis, Daniel Bear, Dan Gutfreund, David Daniel Cox, Antonio Torralba, James J. DiCarlo, Joshua B. Tenenbaum, Josh Mcdermott, and Daniel LK Yamins. ThreeDWorld: A platform for interactive multimodal physical simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL <https://openreview.net/forum?id=db1InWAwW2T>.
- Artur d’Avila Garcez and Luís C. Lamb. Neurosymbolic AI: the 3rd wave. *Artificial Intelligence Review*, 56(11):12387–12406, November 2023. ISSN 1573-7462. doi: 10.1007/s10462-023-10448-w. URL <https://doi.org/10.1007/s10462-023-10448-w>.
- Steven Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-chun Zhu, Demetri Terzopoulos, Feifei Li, and Jianfeng Gao. Mindagent: Emergent gaming interaction. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL) 2024*, jan 2024.
- Jiatao Gu, Alex Trevithick, Kai-En Lin, Joshua M Susskind, Christian Theobalt, Lingjie Liu, and Ravi Ramamoorthi. Nerfdiff: Single-image view synthesis with nerf-guided distillation from 3d-aware diffusion. In *International Conference on Machine Learning*, pages 11808–11826. PMLR, 2023.
- Charles G. Gunn and Steven De Keninck. Geometric algebra and computer graphics. In *ACM SIGGRAPH 2019 Courses*, SIGGRAPH ’19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450363075. doi: 10.1145/3305366.3328099. URL <https://doi.org/10.1145/3305366.3328099>.
- Thomas Hawkins. The erlanger programm of felix klein: Reflections on its place in the history of mathematics. *Historia Mathematica*, 11(4):442–470, 1984. ISSN 0315-0860. doi: [https://doi.org/10.1016/0315-0860\(84\)90028-4](https://doi.org/10.1016/0315-0860(84)90028-4). URL <https://www.sciencedirect.com/science/article/pii/0315086084900284>.
- Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Escaping plato’s cave: 3d shape from adversarial rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9984–9993, 2019.
- Dietmar Hildenbrand and Alyn Rockwood. Geometric algebra computing for computer graphics using gaalop. In *ACM SIGGRAPH 2022 Courses*, SIGGRAPH ’22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393621. doi: 10.1145/3532720.3535655. URL <https://doi.org/10.1145/3532720.3535655>.

- Lukas Höllein, Ang Cao, Andrew Owens, Justin Johnson, and Matthias Nießner. Text2room: Extracting textured 3d meshes from 2d text-to-image models. *arXiv preprint arXiv:2303.11989*, 2023.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2023a.
- Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3d-LLM: Injecting the 3d world into large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL <https://openreview.net/forum?id=YQA28p7qNz>.
- Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. Graph2plan: Learning floorplan generation from layout graphs. *ACM Transactions on Graphics (TOG)*, 39(4):118–1, 2020.
- Ian Huang, Vrishab Krishna, Omoruyi Atekha, and Leonidas Guibas. Aladdin: Zero-shot hallucination of stylized 3d assets from abstract scene descriptions. *arXiv preprint arXiv:2306.06212*, 2023.
- Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. Openclip, July 2021. URL <https://doi.org/10.5281/zenodo.5143773>. If you use this software, please cite it as below.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Léo Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.
- Anushrut Jignasu, Kelly Marshall, Baskar Ganapathysubramanian, Aditya Balu, Chinmay Hegde, and Adarsh Krishnamurthy. Towards foundational ai models for additive manufacturing: Language models for g-code debugging, manipulation, and comprehension, 2023.
- Manos Kamarianakis and George Papagiannakis. An all-in-one geometric algorithm for cutting, tearing, and drilling deformable models. *Advances in Applied Clifford Algebras*, 31(3):58, 2021.
- Manos Kamarianakis, Antonis Protopsaltis, Dimitris Angelis, Michail Tamiolakis, and George Papagiannakis. Progressive tearing and cutting of soft-bodies in high-performance virtual reality, 2022.

- Ivan Kapelyukh and Edward Johns. Scenescore: Learning a cost function for object arrangement. In *CoRL 2023 Workshop on Learning Effective Abstractions for Planning (LEAP)*, 2023. URL <https://openreview.net/forum?id=DVKEwUfKQA>.
- Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. URL <https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/>.
- Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.
- Yasuaki Kuroe. Models of clifford recurrent neural networks and their dynamics. In *The 2011 international joint conference on neural networks*, pages 1035–1041. IEEE, 2011.
- Teyun Kwon, Norman Di Palo, and Edward Johns. Language models as zero-shot trajectory generators. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024. URL <https://openreview.net/forum?id=5fEHN44e3k>.
- Yanping Li, Yue Wang, Rui Wang, Yi Wang, Kaili Wang, Xiangyang Wang, Wenming Cao, and Wei Xiang. Ga-cnn: Convolutional neural network based on geometric algebra for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–14, 2022.
- Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 300–309, 2023a.
- Yiqi Lin, Hao Wu, Ruichen Wang, Haonan Lu, Xiaodong Lin, Hui Xiong, and Lin Wang. Towards language-guided interactive 3d generation: Llms as layout interpreter with generative feedback. *arXiv preprint arXiv:2305.15808*, 2023b.
- Qifan Liu and Wenming Cao. Geometric algebra graph neural network for cross-domain few-shot classification. *Applied Intelligence*, 52(11):12422–12435, 2022.
- 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.



- Setareh Aghel Manesh, Tianyi Zhang, Yuki Onishi, Kotaro Hara, Scott Bateman, Jiannan Li, and Anthony Tang. How people prompt to create interactive vr scenes. *ArXiv*, abs/2402.10525, 2024. URL <https://api.semanticscholar.org/CorpusID:267740699>.
- Angelos Mavrogiannis, Christoforos Mavrogiannis, and Yiannis Aloimonos. Cook2ltl: Translating cooking recipes to ltl formulae using large language models. *arXiv preprint arXiv:2310.00163*, 2023.
- Gal Metzer, Elad Richardson, Or Patashnik, Raja Giryes, and Daniel Cohen-Or. Latent-nerf for shape-guided generation of 3d shapes and textures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12663–12673, 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 *The European Conference on Computer Vision (ECCV)*, 2020.
- 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. *Communications of the ACM*, 65(1):99–106, 2021.
- Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7588–7597, 2019.
- Margarita Papaefthymiou, Dietmar Hildenbrand, and George Papagiannakis. An inclusive conformal geometric algebra gpu animation interpolation and deformation algorithm. *The Visual Computer*, 32:751–759, 2016.
- George Papagiannakis. Geometric algebra rotors for skinned character animation blending. In *SIGGRAPH Asia 2013 Technical Briefs*, pages 1–6. 2013.
- George Papagiannakis, Manos Kamarianakis, Antonis Protopsaltis, Dimitris Angelis, and Paul Zikas. Project elements: A computational entity-component-system in a scene-graph pythonic framework, for a neural, geometric computer graphics curriculum, 2023.
- 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.
- JK Pearson and DL Bisset. Neural networks in the clifford domain. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 3, pages 1465–1469. IEEE, 1994.

- Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.
- 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.
- Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/ramesh21a.html>.
- Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL <https://arxiv.org/abs/1908.10084>.
- Joe Rooney. William kingdon clifford (1845–1879). In *Distinguished Figures in Mechanism and Machine Science: Their Contributions and Legacies Part 1*, pages 79–116. Springer, 2007.
- David Ruhe, Jayesh K Gupta, Steven De Keninck, Max Welling, and Johannes Brandstetter. Geometric clifford algebra networks. In *International Conference on Machine Learning*, pages 29306–29337. PMLR, 2023.
- David Ruhe, Johannes Brandstetter, and Patrick Forré. Clifford group equivariant neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.
- Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade W Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa R Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5b: An open large-scale dataset for training next generation image-text models. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL <https://openreview.net/forum?id=M3Y74vmsMcY>.
- Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5466–5475, 2023.
- A. Sheth, K. Roy, and M. Gaur. Neurosymbolic artificial intelligence (why, what, and how). *IEEE Intelligent Systems*, 38(03):56–62, may 2023. ISSN 1941-1294. doi: 10.1109/MIS.2023.3268724.

- Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B. Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann. Neural descriptor fields:  $Se(3)$ -equivariant object representations for manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6394–6400, 2022. doi: 10.1109/ICRA46639.2022.9812146.
- Anthony Simeonov, Yilun Du, Yen-Chen Lin, Alberto Rodriguez Garcia, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Pulkit Agrawal.  $Se(3)$ -equivariant relational rearrangement with neural descriptor fields. In Karen Liu, Dana Kulis, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 835–846. PMLR, 14–18 Dec 2023. URL <https://proceedings.mlr.press/v205/simeonov23a.html>.
- Fuwen Tan, Song Feng, and Vicente Ordonez. Text2scene: Generating compositional scenes from textual descriptions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6710–6719, 2019.
- Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Scene graph denoising diffusion probabilistic model for generative indoor scene synthesis. *arXiv preprint arXiv:2303.14207*, 2023.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.
- Jian Wang, Ziqiang Wang, Han Wang, Wen Luo, Linwang Yuan, Guonian Lü, and Zhaoyuan Yu. Large language model for geometric algebra: A preliminary attempt. In *Advances in Computer Graphics: 40th Computer Graphics International Conference, CGI 2023, Shanghai, China, August 28 – September 1, 2023, Proceedings, Part IV*, page 237–249, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-50077-0. doi: 10.1007/978-3-031-50078-7\_19. URL [https://doi.org/10.1007/978-3-031-50078-7\\_19](https://doi.org/10.1007/978-3-031-50078-7_19).

- Rui Wang, Miaomiao Shen, Xiangyang Wang, and Wenming Cao. Rga-cnns: convolutional neural networks based on reduced geometric algebra. *Sci. China Inf. Sci.*, 64(129101):1–129101, 2021a.
- 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, 2021b.
- Qihong Anna Wei, Sijie Ding, Jeong Joon Park, Rahul Sajjani, Adrien Poulenard, Srinath Sridhar, and Leonidas Guibas. Lego-net: Learning regular rearrangements of objects in rooms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19037–19047, 2023.
- Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems*, 29, 2016.
- Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019.
- 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. *arXiv preprint arXiv:2312.09067*, 2023.
- Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. <https://arxiv.org/abs/2012.02190>, 2020.
- Guangyao Zhai, Evin Pinar Örnek, Dave Zhenyu Chen, Ruotong Liao, Yan Di, Nassir Navab, Federico Tombari, and Benjamin Busam. Echoscene: Indoor scene generation via information echo over scene graph diffusion. *arXiv preprint arXiv:*, 2024.
- Jingbo Zhang, Xiaoyu Li, Ziyu Wan, Can Wang, and Jing Liao. Text2nerf: Text-driven 3d scene generation with neural radiance fields. *arXiv preprint arXiv:2305.11588*, 2023.
- Yiqun Zhao, Zibo Zhao, Jing Li, Sixun Dong, and Shenghua Gao. Roomdesigner: Encoding anchor-latents for style-consistent and shape-compatible indoor scene generation. *arXiv preprint arXiv:2310.10027*, 2023.
- Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5826–5835, 2021.

Jingwen Zhu and Jitao Sun. Global exponential stability of clifford-valued recurrent neural networks. *Neurocomputing*, 173:685–689, 2016.

Paul Zikas, Antonis Protopsaltis, Nick Lydatakis, Mike Kentros, Stratos Geronikolakis, Steve Kateros, Manos Kamarianakis, Giannis Evangelou, Achilleas Filippidis, Eleni Grigoriou, Dimitris Angelis, Michail Tamiolakis, Michael Dodis, George Kokiadis, John Petropoulos, Maria Pateraki, and George Papagiannakis. Mages 4.0: Accelerating the world’s transition to vr training and democratizing the authoring of the medical metaverse, 2023.



# Appendix A

## Appendices

### A.1 Conformal Geometric Algebra

The Conformal Geometric Algebra (CGA) used in this paper can be seen as another algebra containing dual-quaternions which allows round elements such as spheres to be represented as objects of this algebra, i.e., as *multivectors*. To be more precise, CGA is the lowest possible extension where this is possible. Being able to represent round elements in conjunction with the ability to reflect on objects using the so-called *sandwich operation* presented in the following sections, CGA is also able to represent dilators (uniform scaling) as multivectors. Therefore, CGA is a geometric algebra where round elements (points, spheres, circles), flat elements (lines, planes, point pairs) and all basic deformations (translations, rotations, dilations) can be expressed explicitly in multivector form.

In order to create the model of 3D CGA, we extend the basis  $\{e_1, e_2, e_3\}$  of the original Euclidean space  $\mathbb{R}^3$  by two elements  $e_+$  and  $e_-$ . These elements have positive and negative signature respectively, i.e., it holds that  $e_+^2 = -e_-^2 = 1$ . The resulting non-Euclidean space is usually denoted as  $\mathbb{R}^{4,1}$  while the Clifford (geometric) algebra of  $\mathbb{R}^{4,1}$  is denoted as  $\mathbb{R}_{4,1}$  or  $\mathcal{G}(4, 1)$ .

It is convenient to define a *null* basis given by the original basis vectors  $e_1, e_2, e_3$  of  $\mathbb{R}^3$  and

$$e_o = \frac{1}{2}(e_- - e_+), \quad e_\infty = e_- + e_+. \quad (\text{A.1})$$

The elements  $e_o$  and  $e_\infty$  are called *null* vectors because  $e_o^2 = e_\infty^2 = 0$ , where the operation implied is the geometric product described in the following sections.

#### A.1.1 Vector Objects of $\mathbb{R}_{4,1}$

A generic vector  $Y$  of  $\mathbb{R}_{4,1}$  is a linear combination of the basis elements  $\{e_1, e_2, e_3, e_\infty, e_o\}$ , i.e.,

$$Y = y_1 e_1 + y_2 e_2 + y_3 e_3 + y_\infty e_\infty + y_o e_o, \quad y_i \in \mathbb{R}. \quad (\text{A.2})$$

Note that CGA is a projection space where the elements  $Y$  and  $Z$  are equivalent if and only if there is a  $\lambda \in \mathbb{R}$  such that  $Y = \lambda Z$ . Due to this equivalence, we

usually assume, without loss of generality, that the coordinate of  $e_o$  is either 0 or 1. In this algebra, points, spheres and planes are easily represented as vector objects of the space, as described below.

**Points** A point  $x = (x_1, x_2, x_3) = x_1e_1 + x_2e_2 + x_3e_3$  of  $\mathbb{R}^3$  is *up-projected* into the conformal vector

$$\begin{aligned} X &= x + \frac{1}{2}x^2e_\infty + e_o \\ &= x_1e_1 + x_2e_2 + x_3e_3 + \frac{1}{2}(x_1^2 + x_2^2 + x_3^2)e_\infty + e_o. \end{aligned} \quad (\text{A.3})$$

**Spheres** A sphere  $s$  of the  $\mathbb{R}^3$ , centered at  $x = (x_1, x_2, x_3)$  with radius  $r$  is *up-projected* into the conformal vector

$$\begin{aligned} S &= X - \frac{1}{2}r^2e_\infty \\ &= x_1e_1 + x_2e_2 + x_3e_3 + \frac{1}{2}(x_1^2 + x_2^2 + x_3^2 - r^2)e_\infty + e_o, \end{aligned} \quad (\text{A.4})$$

where  $X$  is the image of  $x$  in  $\mathbb{R}_{4,1}$ .

**Planes** A plane  $\pi$  of the original space, with Euclidean distance  $d$  from the origin, perpendicular to the *normal* vector  $\vec{n} = (n_1, n_2, n_3)$  is *up-projected* into the conformal vector

$$\Pi = \vec{n} + de_\infty = n_1e_1 + n_2e_2 + n_3e_3 + de_\infty. \quad (\text{A.5})$$

### A.1.2 Products in $\mathbb{R}_{4,1}$

There are three major products in  $\mathbb{R}_{4,1}$ : the inner, the outer and the geometric. Each of these products is initially defined among the vectors  $e_1, e_2, e_3, e_-, e_+, e_o, e_\infty$ . The respective definition is then extended to any element (a *multivector*) of the space. Below we present some of the basic properties of these products.

**Inner** The inner product (denoted by  $\cdot$ ) of the basis elements is defined as follows:

- $e_i \cdot e_j := \delta_{ij}$  for  $i, j \in \{1, 2, 3, +\}$ ,
- $e_- \cdot e_- := -1$ ,
- $e_- \cdot e_j := 0$  for  $j \in \{1, 2, 3, +\}$ ,
- $e_o \cdot e_o := e_\infty \cdot e_\infty = 0$ ,
- $e_o \cdot e_\infty := -1$ ,
- $e_i \cdot e_j := 0$  for  $i \in \{1, 2, 3, +\}$  and  $j \in \{o, \infty\}$ .



**Outer** The outer product of the basis elements  $e_i$  and  $e_j$  is denoted as  $e_i \wedge e_j$ . The outer product is an associative operation that can be applied to more than two elements, e.g.,  $e_i \wedge e_j \wedge e_k$  and  $e_i \wedge e_j \wedge e_k \wedge e_\infty$  are properly defined. The outer product of  $k$  basis vectors is called a  $k$ -blade and  $k$  is usually referred to as the *grade* of this blade. A sum of  $k$ -blades is called a  $k$ -vector and the addition of  $k$ -vectors of different grades is a *multivector*.

The importance of the outer product derives from the fact that it allows us, in certain cases, to obtain the intersection of two objects by simply evaluating their outer product. Specifically, a circle (resp. line) can be seen as the intersection - outer product of two spheres (resp. planes). The outer product of a circle with an intersecting sphere or equivalently, the outer product of three intersecting spheres represent a set of two points, usually referred to as a *point pair*.

**Geometric** The most important product in  $\mathbb{R}_{4,1}$  is the so-called *geometric* product. For the basis vectors  $e_i$  and  $e_j$ , their geometric product  $e_i e_j$  is defined as the addition of the outer and inner product of the elements, i.e.,

$$e_i e_j := e_i \wedge e_j + e_i \cdot e_j.$$

Note that, by the definition,  $e_i e_j = e_i \wedge e_j$  for every  $i, j \in \{1, 2, 3, \infty, o\}$  such that  $i \neq j$  and  $\{i, j\} \neq \{\infty, o\}$ .

### A.1.3 Dual Objects

First, let us denote the *pseudoscalar*  $I$  of  $\mathbb{R}_{4,1}$ ,

$$I := e_1 \wedge e_2 \wedge e_3 \wedge e_+ \wedge e_- = e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \wedge e_o. \quad (\text{A.6})$$

Using  $I$ , we may define the dual object  $m^*$  of a multivector  $m$  is to be

$$m^* := -mI, \quad (\text{A.7})$$

where the operation between  $m$  and  $I$  is the geometric product. Notice that it holds that  $(m^*)^* = -m$  and therefore we can easily obtain the normal form  $m$  of an object from it's dual form  $m^*$  and vice versa.

The dual form of certain objects holds strong geometric meaning, as described below.

- The outer product of 4 non-coplanar points yields the dual form of the sphere defined by these points.
- The outer product of 3 non-collinear points and  $e_\infty$  yields the dual form of the plane defined by these points.
- The outer product of 3 non-coplanar points yields the dual form of the circle defined by these points.
- The outer product of 2 points and  $e_\infty$  yields the dual form of the line defined by these points.

### A.1.4 Rotations, Translations and Dilations

So far we have shown that objects (or their duals) such as points, planes, circles, spheres, lines and point pairs are represented as multivectors. However, the beauty and versatility of this algebra comes from its ability to also represent rotations, translations and dilations as multivectors as described below.

**Rotation.** A rotation in CGA is encapsulated in a multivector

$$R := \exp\left(-b\frac{\phi}{2}\right) = \exp\left(-I_3u\frac{\phi}{2}\right) = \cos\left(\frac{\phi}{2}\right) - uI_3\sin\left(\frac{\phi}{2}\right), \quad (\text{A.8})$$

where  $\phi$  is the angle of the rotation,  $b$  is the normalized plane of the rotation,  $u$  is the normalized axis of the rotation and  $I_3 := e_1e_2e_3$ . All products are geometric products and  $\exp(\cdot)$  denotes the exponential function. The inverse multivector of  $R$  is

$$R^{-1} := \exp\left(b\frac{\phi}{2}\right) = \exp\left(I_3u\frac{\phi}{2}\right) = \cos\left(\frac{\phi}{2}\right) + uI_3\sin\left(\frac{\phi}{2}\right). \quad (\text{A.9})$$

**Translation.** The multivector

$$T := \exp\left(-\frac{1}{2}te_\infty\right) = 1 - \frac{1}{2}te_\infty, \quad (\text{A.10})$$

where  $t = t_1e_1 + t_2e_2 + t_3e_3$  is a euclidean vector, represents a translation by  $t$  in CGA. The inverse multivector of  $T$  is

$$T^{-1} := \exp\left(\frac{1}{2}te_\infty\right) = 1 + \frac{1}{2}te_\infty. \quad (\text{A.11})$$

**Dilation.** The multivector

$$D = 1 + \frac{1-d}{1+d}e_\infty \wedge e_o \quad (\text{A.12})$$

corresponds to a dilation of scale factor  $d > 0$  with respect to the origin. The inverse of  $D$  is given by the expression

$$D^{-1} = \frac{(1+d)^2}{4d} + \frac{d^2-1}{4d}e_\infty \wedge e_o. \quad (\text{A.13})$$

An interesting remark is that, for  $d = 0$ , it holds that  $D = 1 + e_\infty \wedge e_o = 1 + e_+e_-$ , which is clearly not invertible in  $\mathbb{R}_{4,1}$  as  $(e_+e_-)^2 = 1$ .

The conformal space model allows us to apply any or multiple of the operations above not only to a point but also to any object  $O$  that was previously defined. Let  $M_i$ , for  $i = 1, \dots, n$ , be either a rotation, a translation or a dilation as defined above. To apply the transformations  $M_1, M_2, \dots, M_n$  (in this order), to an object  $O$ , we first define the multivector  $M := M_n M_{n-1} \cdots M_1$ , where all in-between products are geometric. The object

$$O' := MOM^{-1} \quad (\text{A.14})$$

represents the final form of  $O$  after all transformations are applied.

### A.1.5 Interpolation of Multivectors

Interpolation of data is an essential part for Computer Graphics as it is needed in every animation procedure of a rigged model. The poses of the model with respect to time are not stored in a continuous manner but rather at discrete time-steps. If additional intermediate frames are demanded, we have to interpolate the animation data between two provided keyframes.

As in the case of matrix or (dual) quaternion interpolation, a blending of two multivectors can be accomplished in various ways, yielding different results. Choosing a proper interpolation technique is not a simple task as it may depend on the model or other factors. However, two methods remain dominant in analogue with the quaternion case: the linear and the logarithmic blending.

Linear blending of the multivectors  $m_1$  and  $m_2$ , which, in a model animation context, may represent translations, rotations or dilations, is done by evaluating  $(1 - \alpha)m_1 + \alpha m_2$ , for  $\alpha \in [0, 1]$ . Another blending method is the so-called logarithmic interpolation where we evaluate  $m_1 \exp(\alpha \log(m_1^{-1}m_2))$ , for  $\alpha \in [0, 1]$ , where the exponential and logarithmic function of a multivector  $m$  are approximated in our case using the respective Taylor series expansion. Notice that  $m_1$  is either a rotator, a translator, a dilator with  $d > 0$  or a geometric product of such multivectors and therefore is invertible. Although not evident, one can prove that the logarithmic interpolation method is symmetric if we interchange  $m_1$  and  $m_2$  as well as  $\alpha$  and  $1 - \alpha$ , by using basic exponential and logarithmic properties.

## A.2 3D Scene Generation

### A.2.1 Prompts

You are an experienced room designer. Please assist me in crafting a floor plan. Each room is a rectangle. You need to define the four coordinates and specify an appropriate design scheme, including each room's color, material, and texture. Assume the wall thickness is zero. Please ensure that all rooms are connected, not overlapped, and do not contain each other. Note: the units for the coordinates are meters.

For example:

living room | maple hardwood, matte | light grey drywall, smooth | [(0, 0), (0, 8), (5, 8), (5, 0)] kitchen | white hex tile, glossy | light grey drywall, smooth | [(5, 0), (5, 5), (8, 5), (8, 0)]

Here are some guidelines for you:

1. A room's size range (length or width) is 3m to 8m. The maximum area of a room is  $48 \text{ m}^2$ . Please provide a floor plan within this range and ensure the room is not too small or too large.
2. It is okay to have one room in the floor plan if you think it is reasonable.
3. The room name should be unique.

Now, I need a design for {input}.

Additional requirements: {additional\_requirements}.

Also, you will be given a list of similar designs to take into account for your design. These designs can be used as a reference but you don't need to copy them. You can use them to get inspiration for your design.

Additional designs: {additional\_designs}

Your response should be direct and without additional text at the beginning or end.

Figure A.1: Prompt used for the floor module.

I am now designing (input). Please help me decide the wall height in meters. Answer with a number, for example, 3.0. Do not add additional text at the beginning or in the end.

Figure A.2: Prompt used for the wall module.

I need assistance in designing the connections between rooms.

The connections could be of three types:

- doorframe (no door installed)
- doorway (with a door)
- open (no wall separating rooms).

The sizes available for doorframes and doorways are:

- single (1m wide)
- double (2m wide).

Ensure that the door style complements the design of the room.

The output format should be:

**room 1 | room 2 | connection type | size | door style.**

For example:

exterior | living room | doorway | double | dark brown metal door

living room | kitchen | open | N/A | N/A

living room | bedroom | doorway | single | wooden door with white frames

The design under consideration is {input}, which includes these rooms: {rooms}. The length, width and height of each room in meters are:{room\_sizes} Certain pairs of rooms share a wall: {room\_pairs}. There must be a door to the exterior. Adhere to these additional requirements: {additional\_requirements}.

Also, you will be given a list of similar designs to take into account for your design. These designs can be used as a reference but you don't need to copy them. You can use them to get inspiration for your design.

Additional designs: {additional\_designs}

Provide your response succinctly, without additional text at the beginning or end.

Figure A.3: Prompt used for the door module.

Guide me in designing the windows for each room. The window types are:

- fixed
- hung
- slider.

The available sizes (width x height in cm) are:

fixed: (92, 120), (150, 92), (150, 120), (150, 180), (240, 120), (240, 180)

hung: (87, 160), (96, 91), (120, 160), (130, 67), (130, 87), (130, 130)

slider: (91, 92), (120, 61), (120, 91), (120, 120), (150, 92), (150, 120)

Your task is to determine the appropriate type, size, and quantity of windows for each room, bearing in mind the room's design, dimensions, and function.

Please format your suggestions as follows:

**room | wall direction | window type | size | quantity | window base height (cm from floor).**

For example:

living room | west | fixed | (130, 130) | 1 | 50

I am now designing {input}. The wall height is {wall\_height} cm. The walls available for window installation (direction, width in cm) in each room are: {walls} Please note: It is not mandatory to install windows on every available wall. Within the same room, all windows must be the same type and size. Also, adhere to these additional requirements: {additional\_requirements}. Take into account this additional feedback from an experienced designer: {additional\_feedback}.

Also, you will be given a list of similar designs to take into account for your design. These designs can be used as a reference but you don't need to copy them. You can use them to get inspiration for your design.

Additional designs: {additional\_designs}

Provide a concise response, omitting any additional text at the beginning or end.

Figure A.4: Prompt used for the window module.

You are an experienced room designer, please assist me in selecting large **floor/wall** objects and small objects on top of them to furnish the room. You need to select appropriate objects to satisfy the customer's requirements. If a specific object is mentioned in the design, please include it.

You must provide a description and desired size for each object since I will use it to retrieve object. If multiple identical items are to be placed in the room, please indicate the quantity and variance type (same or varied).

Present your recommendations in JSON format:

```
object_name:{
  "description": a short sentence describing the object,
  "location": "floor" or "wall",
  "size": the desired size of the object, in the format of a list of three numbers,
  [length, width, height] in centimeters,
  "quantity": the number of objects (int),
  "variance_type": "same" or "varied",
  "objects_on_top": a list of small children objects (can be empty) which are placed *on
  top of* this object. For each child object, you only need to provide the object name,
  quantity and variance type. For example, {"object_name": "book", "quantity": 2,
  "variance_type": "varied"}
}
```

For example:

```
"sofa": {
  "description": "modern sectional, light grey sofa",
  "location": "floor",
  "size": [200, 100, 80],
  "quantity": 1,
  "variance_type": "same",
  "objects_on_top": [
    {"object_name": "news paper", "quantity": 2, "variance_type": "varied"},
    {"object_name": "pillow", "quantity": 2, "variance_type": "varied"},
    {"object_name": "mobile phone", "quantity": 1, "variance_type": "same"}
  ]
},
"tv stand": {
  "description": "a modern style TV stand",
  "location": "floor",
  "size": [200, 50, 50],
  "quantity": 1,
  "variance_type": "same",
  "objects_on_top": [
    {"object_name": "49 inch TV", "quantity": 1, "variance_type": "same"},
    {"object_name": "speaker", "quantity": 2, "variance_type": "same"},
    {"object_name": "remote control for TV", "quantity": 1, "variance_type": "same"}
  ]
}
```

Currently, the design in progress is *INPUT*, and we are working on the *ROOM\_TYPE* with the size of *ROOM\_SIZE*.

Please also consider the following additional requirements: *REQUIREMENTS*.

Take into account this additional feedback from an experienced designer: *FEEDBACK*.

Also, you will be given a list of similar designs to take into account for your design. These designs can be used as a reference but you don't need to copy them. You can use them to get inspiration for your design.

Additional designs: *ADDITIONAL\_DESIGNS*

Here are some guidelines for you:

1. Provide reasonable type/style/quantity of objects for each room based on the room size to make the room not too crowded or empty.
2. Do not provide rug/mat, windows, doors, curtains, and ceiling objects which have been installed for each room.
3. I want more types of large objects and more types of small objects on top of the large objects to make the room look more vivid.

Please first use natural language to explain your high-level design strategy for *ROOM\_TYPE*, and then follow the desired JSON format **strictly** (do not add any additional text at the beginning or end).

Figure A.5: Prompt used for the object selection module.



User: {object\_selection\_prompt\_new\_1}  
Agent: {object\_selection\_1}  
User: Thanks! After following your suggestions to retrieve objects, I found the *{room}* is still too empty. To enrich the *{room}*, you could:

1. Add more *floor* objects to the *{room}* (excluding rug, carpet, windows, doors, curtains, and *ignore ceiling objects*).
2. Increase the size and quantity of the objects.
3. Add more *types* of small objects on top of the large objects. Could you update the entire JSON file with the same format as before and answer without additional text at the beginning or end?

Agent:

Figure A.6: Prompt used for the object selection module if in the first try we didn't gather enough objects to fill the room.

You are an experienced room designer. The design in progress is {input} and we are working on the {room\_type}. Please help me evaluate the design of the {room\_type}. Focus on the layout of the room, the placement of the objects, the selected objects, and the overall design of the room.\

I will be providing you with a series of images of the room as well as a description of the room in json format. Also, I'm going to provide a real image that I want you to use as a reference for the design. Take it into account for your evaluation.

Description: {description}

Provide a score for the design of the room from 1 to 10. If the design is not good, provide what should be changed, added or removed.

Please your response in a JSON format as follows:

```
{
  "score": 10,
  "comments": "The design is good."
}
```

Your response should be precise, without additional text at the beginning or end. Follow the desired JSON format **strictly**

Figure A.7: Prompt used for the feedback module.

## A.3 3D Scene Editing

### A.3.1 Ours, Euclidean & Omniverse Prompts

As discussed in Section 5.4 of the thesis, Figure A.10 displays the Euclidean prompt. Additionally, Figures A.11 and A.12 showcase the prompts from NVIDIA’s Omniverse<sup>1</sup>. For a fair comparison, both sets of prompts are presented with the main guidelines and examples, similar to the CGA prompt.

---

<sup>1</sup><https://github.com/NVIDIA-Omniverse/kit-extension-sample-airroomgenerator/blob/main/exts/omni.example.airroomgenerator/omni/example/airroomgenerator/prompts.py>

cga\_prompt.md

2024-05-23

**Objective:**

You MUST calculate the appropriate transformation for objects in a 3D space using templated variables (X1, X2, etc.). You MUST only use a valid combination of the defined algebraic operations.

**Coordinate System:**

- **X-axis (e1):** Translates the object LEFT (-) or RIGHT (+).
- **Y-axis (e2):** Translates the object BOTTOM (-) or TOP (+) height wise.
- **Z-axis (e3):** Translates the object BACK (-) or FORWARD (+).

**Defined Algebraic Operations:**

1. **Rotation** The rotation rotor  $R(\theta, u, v)$  rotates a vector by an angle  $\theta$  in the plane defined by  $u \wedge v$ .
2. **Translation** The translation rotor  $T(x * e1 + y * e2 + z * e3)$  translates an object along the vector  $x * e1 + y * e2 + z * e3$ .
3. **Coordinate Extraction** Use the inner product  $(x1 | e3) * e3$  to extract the coordinate along the Y-axis.
4. **Outer Product** For independent vectors,  $u \wedge v$  is non-zero and represents an oriented plane.
5. **Commutation:** Translation operators commute,  $T(v) * T(w) = T(w) * T(v)$ .
6. **Rotor Composition:** Rotors can be combined by multiplication, representing sequential rotations.

**Task Instructions:**

1. APPLY the specified T for translation and R for rotation operations, adhering to the defined algebraic operations.
2. DO NOT introduce any variables or constants not explicitly mentioned in the input.
3. USE ONLY the variables names and e1, e2, e3 on the operations.
4. OUTPUT the transformations in JSON format, using key-value pairs for each object and its transformation.
5. USE ONLY the max and min points of an object's bounding box (e.g., X1.max and X1.min) for all calculations.
6. For coordinate extraction, employ the inner product operator |. For instance, use  $(X1.max | e2) * e2$  to extract the Y- axis coordinate.
7. CONSIDER the current position of objects when moving one in relation to another. Calculate transformations using the initial locations derived from their bounding box points.

**Example Inputs, Results in JSON format**

1. **Input:** 'Translate X1 by 1 unit to the left.' **Output:** {'X1': 'T(-1 \* e1)'} **Explanation:** The bounding box of X1 is defined by the points X1.max and X1.min. To translate X1 by 1 unit to the left, we must subtract 1 from the X-axis coordinate of both points. Therefore, the transformation is  $T(-1 * e1)$ .
2. **Input:** 'Rotate X1 by 90 degrees around the z-axis.' **Output:** {'X1': 'R(np.pi/2, e1, e2)'} **Explanation:** The rotation rotor  $R(\theta, u, v)$  rotates a vector by an angle  $\theta$  in the plane defined by  $u \wedge v$ . In this case, the rotation is 90 degrees in the z-axis is defined as the rotation around e1, e2 plane. Therefore, the transformation is  $R(np.pi/2, e1, e2)$ .

Figure A.8: Template for Conformal Geometric Algebra Prompts Used by our framework. (1/2)

cga\_prompt.md

2024-05-23

3. **Input:** 'Translate X2 by 1 unit to the right and rotate it by 90 degrees around the z-axis.' **Output:** {'X2': 'T(1 \* e1) \* R(np.pi/2, e1, e2)'} **Explanation:** The bounding box of X2 is defined by the points X2.max and X2.min. To translate X2 by 1 unit to the right, we must add 1 to the X-axis coordinate of both points. Therefore, the translation is T(1 \* e1). The rotation rotor R(theta, u, v) rotates a vector by an angle theta in the plane defined by u ^ v. In this case, the rotation is 90 degrees in the z-axis is defined as the rotation around e1, e2 plane. Therefore, the rotation is R(np.pi/2, e1, e2). Since the rotation must be applied after the translation, the transformation is T(1 \* e1) \* R(np.pi/2, e1, e2).
4. **Input:** 'Move X1 on top of X0.' **Output:** {'X1': 'T((X0.max+X0.min)/2 - (X1.max+X1.min)/2 + ((X0.max - X0.min) | e2) \* e2)'} **Explanation:** The bounding box of X1 is defined by the points X1.max and X1.min. The bounding box of X0 is defined by the points X0.max and X0.min. To move X1 on top of X0, we must translate X1 by the difference between the center of X0 and X1 and the Y-axis coordinate of X0. Therefore, the transformation is T((X0.max+X0.min)/2 - (X1.max+X1.min)/2 + ((X0.max - X0.min) | e2) \* e2).
5. **Input:** 'Move X1 next to X0.' **Output:** {'X1': 'T((X0.max+X0.min)/2 - (X1.max+X1.min)/2 + (X0.max + (X0.max - X0.min) | e1) \* e1)'} **Explanation:** The bounding box of X1 is defined by the points X1.max and X1.min. The bounding box of X0 is defined by the points X0.max and X0.min. To move X1 next to X0, we must translate X1 by the difference between the center of X0 and X1 and the X-axis coordinate of X0 and then add an offset along the e1 plane in order for the objects to not collide with each other. Therefore, the transformation is T((X0.max+X0.min)/2 - (X1.max+X1.min)/2 + ((X0.max - X0.min) | e1) \* e1).

**Output Format:**

The output MUST be formatted as a SINGLE JSON object containing all transformations for each object.

**Additional Guidance for LLM:**

- ADHERE strictly to the defined algebraic operations.
- DOCUMENT Each Step: For every action you perform, you MUST provide a clear and numbered list of operations used during your reasoning.
- REVISE any transformation that does not comply with these principles.
- It's **IMPORTANT** to take into consideration all axis when performing relative transformations. Focus on the X **AND** Z axis when moving objects around the room. Make changes in the Y axis when a vertical movement is required. So, for example, when moving an object next to another, you should move it in both the X and Z axis, and when moving an object on top of another, you should move it in the X and Z axis, and also in the Y axis.
- When no axis is defined for rotations, the default axis is the Y-axis which means a rotation in the e1, e3 planes.
- When you want to move or rotate an object along an axis you must use the inner product operator |. For example, to move an object along the Y-axis you must use (X1.max | e2) \* e2.
- When no unit is specified for the translation, the default unit is 1 and for the rotation, the default unit is 90 degrees.

Figure A.9: Template for Conformal Geometric Algebra Prompts Used by our framework. (2/2)

You are a sentient AI capable of manipulating the placement of objects within a scene through matrix calculations. These calculations produce transformation matrices in Euclidean 3D space that conform to user commands.\

#### AVAILABLE FUNCTIONS:

You MUST remember that this conversation is a monologue, with you in control. I am unable to assist with any questions.

You MUST independently produce the final output using available information, common sense, and general knowledge.

You MUST use the following details for each object to generate the output:

1. **TRS\_matrix**: A 4x4 transformation matrix that represents the translation, rotation, and scaling of an object in 3D space.
2. **Box**: A list of two vectors, where the first vector represents the minimum point of the bounding box and the second vector the maximum point. Use `X1.Box[0]` to access the minimum point and `X1.Box[1]` for the maximum point. For coordinates, `X1.Box[0][0]` accesses the x-coordinate of the minimum point, etc.

#### General Functions:

1. **translation\_matrix(x, y, z)**: Generates a translation matrix for the specified x, y, and z values.
2. **rotation\_matrix(x, y, z)**: Produces a rotation matrix for the given x, y, and z values.
3. **scaling\_matrix(x, y, z)**: Creates a scaling matrix for the specified x, y, and z values.

#### ENVIRONMENT SET-UP:

The 3D coordinate system is defined as follows:

1. The x-axis extends horizontally to the right.
2. The y-axis extends vertically towards the ceiling.
3. The z-axis also extends vertically, but upwards.

#### COLLISION AVOIDANCE:

For tasks involving multiple objects, you MUST ensure no collisions by considering each object's bounding box.

#### INITIAL PLANNING:

You MUST detail the initial planning and reasoning for the task before generating calculations.

#### OUTPUT FORMAT:

YOU MUST provide a sequence of calculations for generating the transformation matrix for each object in the scene in JSON format.

#### EXAMPLES:

1. **Input**: "Translate X1 2 units to the right."  
**Output**: `{'X1': 'X1.TRS_matrix @ translation_matrix(2, 0, 0)'}`  
**Explanation**: The translation matrix is applied to the object X1 to move it 2 units to the right.
2. **Input**: "Rotate X1 90 degrees around the z-axis."  
**Output**: `{'X1': 'X1.TRS_matrix @ rotation_matrix(0, 0, 90)'}`  
**Explanation**: The rotation matrix is applied to the object X1 to rotate it by 90 degrees around the z-axis.
3. **Input**: "Translate X2 1 unit to the right and rotate it by 90 degrees around the z-axis."  
**Output**: `{'X2': 'X2.TRS_matrix @ translation_matrix(1, 0, 0) @ rotation_matrix(0, 0, 90)'}`  
**Explanation**: The translation matrix is applied to the object X2 to move it 1 unit to the right, followed by a rotation of 90 degrees around the z-axis.
4. **Input**: "Move X1 to the right of X2."  
**Output**: `{'X1': 'X1.TRS_matrix @ translation_matrix(X2.Box[1][0] - X1.Box[0][0], 0, 0)'}`  
**Explanation**: The translation matrix is applied to the object X1 to move it to the right of object X2.
5. **Input**: "Move X1 on top of X2."  
**Output**: `{'X1': 'X1.TRS_matrix @ translation_matrix((X2.Box[1][0] - X1.Box[0][0])/2, X2.Box[1][1] - X1.Box[0][1], (X2.Box[1][2] - X1.Box[0][2])/2)'}`  
**Explanation**: The translation matrix is applied to the object X1 to move it on top of object X2.
6. **Input**: "Rotate X1 by 90 degrees around the x-axis and move it by 2 to the right."  
**Output**: `{'X1': 'X1.TRS_matrix @ translation_matrix(2, 0, 0) @ rotation_matrix(90, 0, 0)'}`  
**Explanation**: The translation matrix is applied to the object X1 to move it 2 units to the right, followed by a rotation of 90 degrees around the x-axis.

#### Additional Guidance for LLM:

- ADHERE strictly to the defined algebraic operations.
- DO NOT introduce any variables or constants not explicitly mentioned in the input.
- DOCUMENT each step: For every action you perform, you MUST provide a clear and numbered list of operations used during your reasoning.
- REVISE any transformation that does not comply with these principles.
- It is **IMPORTANT** to take into consideration all axis when performing relative transformations. Focus on the X **AND** Z axis when moving objects around the room. Make changes in the Y axis when a vertical movement is required. So, for example, when moving an object next to another, you should move it in both the X and Z axis, and when moving an object on top of another, you should move it in the X and Z axis, and also in the Y axis.
- When no axis is defined for rotations, the default axis is the Y-axis.
- When no unit is specified for the translation, the default unit is 1 and for the rotation, the default unit is 90 degrees.
- MOVE ONLY the objects that are mentioned in the query. Do not move any other objects in the scene.

Figure A.10: Euclidean Prompt template for the Scene Editing Agent.

## Objective:

You MUST calculate the appropriate transformation for objects in a 3D space.

## Coordinate System:

- **X-axis (e1):** Translates the object LEFT (-) or RIGHT (+).
- **Y-axis (e2):** Translates the object BOTTOM (-) or TOP (+) height wise.
- **Z-axis (e3):** Translates the object back (-) or forward (+).

You receive from the user the query to move objects in the 3D space. You must reason about the query in Euclidean Geometry and provide the transformation for each object in the scene.

You answer by only generating JSON files that contain the following information:

For each object you need to store:

- object\_name: name of the object
- P\_X: position of the object on X axis
- P\_Y: position of the object on Y axis
- P\_Z: position of the object on Z axis
- R\_X: rotation of the object on X axis
- R\_Y: rotation of the object on Y axis
- R\_Z: rotation of the object on Z axis
- Length: dimension in cm of the object on X axis
- Width: dimension in cm of the object on Y axis
- Height: dimension in cm of the object on Z axis

Keep in mind, objects should be disposed in the area to create the most meaningful layout possible, and they shouldn't overlap.

Also keep in mind that the objects should be disposed all over the area in respect to the origin point of the area, and you can use negative values as well to display items correctly, since origin of the area is always at the center of the area.

## Example Inputs, Results in JSON format

The current scene is:

```
[
  {
    "object_name": "Parts_Pallet_1",
    "P_X": -150,
    "P_Y": 0.0,
    "P_Z": 250,
    "R_X": 0.0,
    "R_Y": 0.0,
    "R_Z": 0.0,
    "Length": 100,
    "Width": 100,
    "Height": 10,
  },
  {
    "object_name": "Boxes_Pallet_2",
    "P_X": -150,
    "P_Y": 0.0,
    "P_Z": 150,
    "R_X": 0.0,
    "R_Y": 0.0,
    "R_Z": 0.0,
    "Length": 100,
    "Width": 100,
    "Height": 10,
  }
]
```

1. **Input:** "Move the Parts\_Pallet\_1 to the right by 50 units."

**Output:**

```
{
  "object_name": "Parts_Pallet_1",
  "P_X": -100,
  "P_Y": 0.0,
  "P_Z": 250,
  "R_X": 0.0,
  "R_Y": 0.0,
  "R_Z": 0.0,
  "Length": 100,
  "Width": 100,
  "Height": 10,
}
```

Figure A.11: Omniverse Prompt template for the Scene Editing Agent Part 1.

2. **Input:** "Rotate the Parts\_Pallet\_1 by 90 degrees around the z-axis."

**Output:**

```
{
  "object_name": "Parts_Pallet_1",
  "P_X": -150,
  "P_Y": 0.0,
  "P_Z": 250,
  "R_X": 0.0,
  "R_Y": 0.0,
  "R_Z": 90.0,
  "Length": 100,
  "Width": 100,
  "Height": 10,
}
```

3. **Input:** "Translate the Parts\_Pallet\_1 by 100 units to the right rotate it by 90 degrees around the z-axis."

**Output:**

```
{
  "object_name": "Parts_Pallet_1",
  "P_X": -50,
  "P_Y": 0.0,
  "P_Z": 250,
  "R_X": 0.0,
  "R_Y": 0.0,
  "R_Z": 90.0,
  "Length": 100,
  "Width": 100,
  "Height": 10,
}
```

4. **Input:** "Move the Parts\_Pallet\_1 on top of the Boxes\_Pallet\_2."

**Output:**

```
{
  "object_name": "Parts_Pallet_1",
  "P_X": -150,
  "P_Y": 10.0,
  "P_Z": 150,
  "R_X": 0.0,
  "R_Y": 0.0,
  "R_Z": 0.0,
  "Length": 100,
  "Width": 100,
  "Height": 10,
}
```

5. **Input:** "Move the Parts\_Pallet\_1 next to the Boxes\_Pallet\_2."

**Output:**

```
{
  "object_name": "Parts_Pallet_1",
  "P_X": -250,
  "P_Y": 0.0,
  "P_Z": 150,
  "R_X": 0.0,
  "R_Y": 0.0,
  "R_Z": 0.0,
  "Length": 100,
  "Width": 100,
  "Height": 10,
}
```

Figure A.12: Omniverse Prompt template for the Scene Editing Agent Part 2.



**Output Format:**

The output **MUST** be formatted as a list of JSON objects, each containing the transformation for each object in the scene.

**Additional Guidance for LLM:**

- **ADHERE** strictly to the defined algebraic operations.
- **DOCUMENT Each Step:** For every action you perform, you **MUST** provide a clear and numbered list of operations used during your reasoning.
- **REVISE** any transformation that does not comply with these principles.
- It's **IMPORTANT** to take into consideration all axis when performing relative transformations. Focus on the X **AND** Z axis when moving objects around the room. Make changes in the Y axis when a vertical movement is required. So, for example, when moving an object next to another, you should move it in both the X and Z axis, and when moving an object on top of another, you should move it in the X and Z axis, and also in the Y axis.
- When no axis is defined for rotations, the default axis is the Y-axis.
- When no unit is specified for the translation, the default unit is 1 and for the rotation, the default unit is 90 degrees.
- **MOVE ONLY** the objects that are mentioned in the query. Do not move any other objects in the scene.

Figure A.13: Omniverse Prompt template for the Scene Editing Agent Part 3.