Programming as Dialogue

LLMs and Designers in Collaborative 3D Modeling

Antonio Sitong Li, Lydia Chilton

LAIDLAW FOUNDATION

Introduction:

Generative AI has recently expanded into three-dimensional design, yet **most methods still rely on diffusion models that produce noisy point clouds or meshes**. Such outputs are difficult to edit and poorly suited for professional workflows that require precision and parametric control.

This paper proposes a programming-centered alternative: **using large language models (LLMs) to generate parametric code** in familiar languages like Python, which can be run inside popular 3D modeling software such as Rhinoceros 3D through a Model Context Protocol (MCP) connection. Making code the central representation ensures models remain editable, customizable, and aligned with existing design practices. In doing so, it lays the groundwork for more natural human–AI collaboration, allowing designers and AI to work iteratively within familiar modeling environments and **positioning AI as a true copilot in the design process.** Beyond collaboration, this framework ties generative 3D modeling directly to advances in code generation, enabling improvements in **generative programming to directly translate into improvement for generative 3D design.**

Approach:

Framework Structure: The system links **Cursor IDE**, Rhino MCP Server, and **Rhino 3D** into a pipeline where natural-language prompts are converted into Python scripts, executed as 3D models, and iteratively refined through re-prompting, error correction, and direct editing.

Self Improving Framework: By utilizing a code-based system, the framework builds a growing repository of generated scripts that serve as self-references. This repository enables multi-shot learning, allowing the model to improve over time by drawing on prior generations.

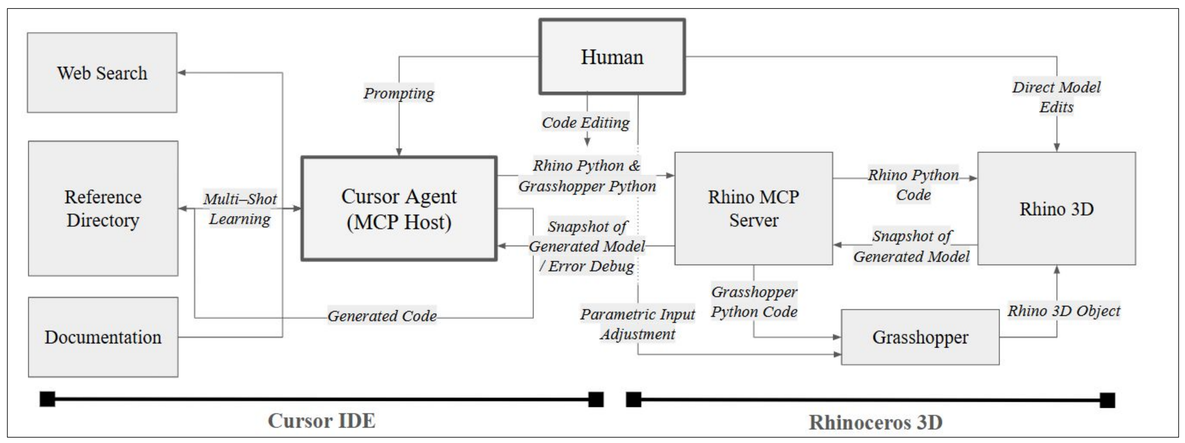


Diagram of Framework Structure

Programmatic Representation:

The framework organizes design elements into a clear hierarchy, where subdirectories represent projects, files correspond to individual parts, and functions define specific objects or operations. This programmatic mapping makes models inherently modular and reusable, while also supporting the scalable assembly of complex designs from simpler components.

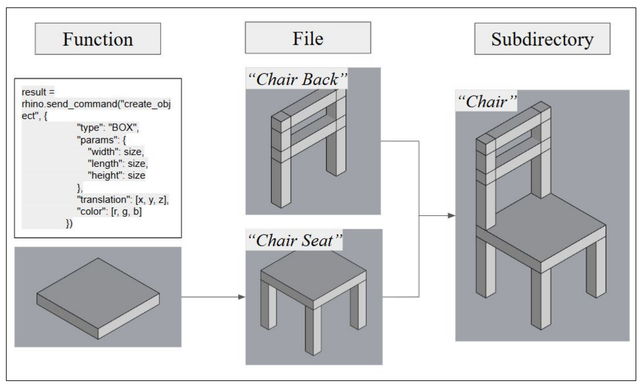


Diagram of Link Between Code Repo Structure and 3D Model

Direct Generation Structure:

1: Prompting

- User enters a natural-language request in Cursor IDE.

2: Code Generation

- Cursor agent generates a Python file using RhinoPython or Grasshopper Python.
- File is exported via MCP to Rhino.

3: 3D Object Generation

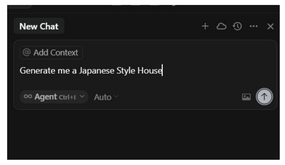
- Rhino executes the code to create geometry.
- Errors trigger diagnostics sent back to Cursor for auto-fix and retry.

4: Re-Prompting

- User gives a new prompt to edit or extend the model.
- Cursor updates existing code or creates a new script referencing prior outputs.

5: Direct Editing

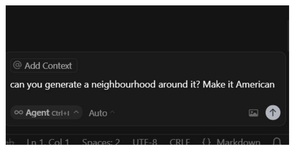
- User refines results inside Rhino.
- RhinoPython outputs are editable with exiting Rhino modeling tools



Cursor Prompt: "Generate me a Japanese Style House"

Sample of Generated Code

Photo of Generated House



Cursor Re-prompt: "Generate me a neighbourhood around it"

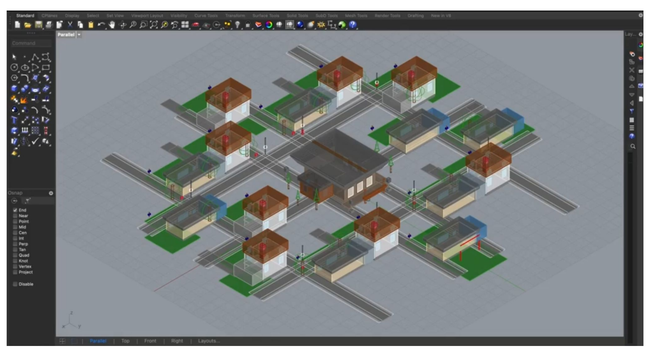
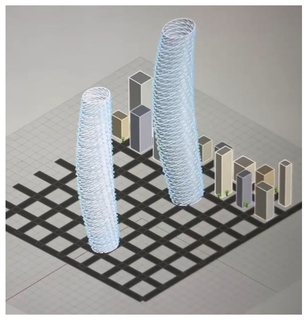


Photo of Final Generated Model

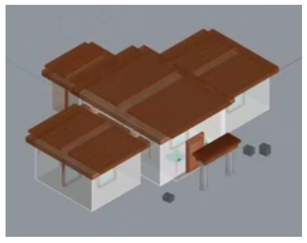
Results:

The framework generates coherent, editable models that improve with iteration, capture stylistic cues, and extend prior outputs into larger assemblies, supporting part-based workflows.

Limitations include occasional overlaps and poor performance on complex surfaces, underscoring the need for stronger spatial reasoning and geometry handling.



Prompt 1: "Make me two parametric skyscrapers"
Prompt 2: "Create a city around it"



Prompt: "Generate me a Chinese Style House"

Conclusion:

This work introduces a programming-centered framework linking LLMs with Rhino 3D via MCP to generate editable, parametric models from natural-language prompts. Results highlight its potential for coherent, stylistically informed, and extensible design, while limitations remain in handling complex geometry. Future work should pursue true bidirectionality, where edits in 3D propagate back into code, and incorporate user studies with quantitative evaluation to assess effectiveness in practice.

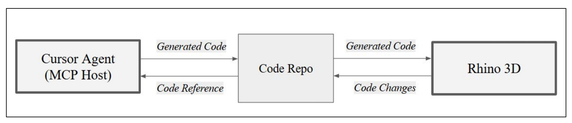


Diagram of Bidirectional Framework

References:

- https://github.com/jingcheng-chen/rhinomcp
- https://cursor.com/agents
- https://developer.rhino3d.com/en/guides/rhinopython/
- https://doi.org/10.48550/arXiv.2009.08026