RELSHAPE: PROGRAM-MEDIATED GENERATION OF REALISTIC 3D ASSETS WITH CONTROLLABLE RELATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Text- and image-conditioned 3D generation could supply games, simulation and design with objects on demand. Yet 3D generative models yield realistic assets either as monolithic shapes or as parts with no relations between them, while program-based methods expose structure but rarely realism or reliable variation, so neither suits workflows that revise and vary assets. We therefore change what 3D generation produces. Program-mediated generation, a paradigm for creating realistic 3D assets with controllable relations, writes a procedural model of the object from the input and materializes every executed instance into a complete asset with a pretrained 3D generator. To make the relations controllable, we represent the procedural model as a Relational Shape Program (RSP), which assembles parts through interfaces computed from their geometry and derives part parameters from object-level controls. To synthesize an RSP that varies correctly from a single input, a 3D coding agent designs its variation before its shape: it declares the object-level controls with the effects and invariants each must satisfy, and keeps only those that executions at non-default values confirm. Each asset keeps the part structure and configuration of its instance, so the assets are realistic, explicitly structured and reliably variable. Experiments show that our assets rival 3D generative models and surpass program-based methods, while following their controls and edits at unseen settings.

1 INTRODUCTION

3D assets populate games, simulation and product design, and text- and image-conditioned generation promises to supply them on demand. 3D generative models now produce detailed geometry and appearance (Xiang et al., 2025; 2026; Tencent Hunyuan3D Team, 2025). Yet an asset rarely leaves generation in its final form. A designer resizes a generated object or adds parts to it, and a game needs many variants of the same object. These revisions act on the *relations* among the object's parts: how they attach and which dimensions depend on which.

Methods that generate 3D objects largely fall into two families, which treat these relations differently. 3D generative models reach high realism, but they represent an object as a single shape or as parts generated side by side (Lin et al., 2025; Yang et al., 2025), so they record no explicit relations between its parts. Program-based methods draw on procedural modeling, which represents an object by the program that constructs it, making such relations explicit (Sun et al., 2024; Pearl et al., 2025). They have LLMs write such programs from text or images (Zhang et al., 2025b;a; Gao et al., 2026; Le et al., 2025), but their programs are checked mainly at the input's setting. Such revisions need three properties at once: *realistic* complete assets, *explicit relations* among their parts, and *reliable variation*, so that each revision yields another valid object. 3D generative models provide the first and program-based methods the second, while the procedural models that provide all three, such as those of Infinigen (Raistrick et al., 2024), are written by hand, one category at a time.

We therefore make such a procedural model the output of generation. We call this paradigm *program-mediated generation*: what it generates is not a shape but the program that constructs one. The procedural model is synthesized from a single text or image, and a pretrained 3D generator (Xiang et al., 2026) materializes every executed instance into a complete asset that keeps the instance's

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

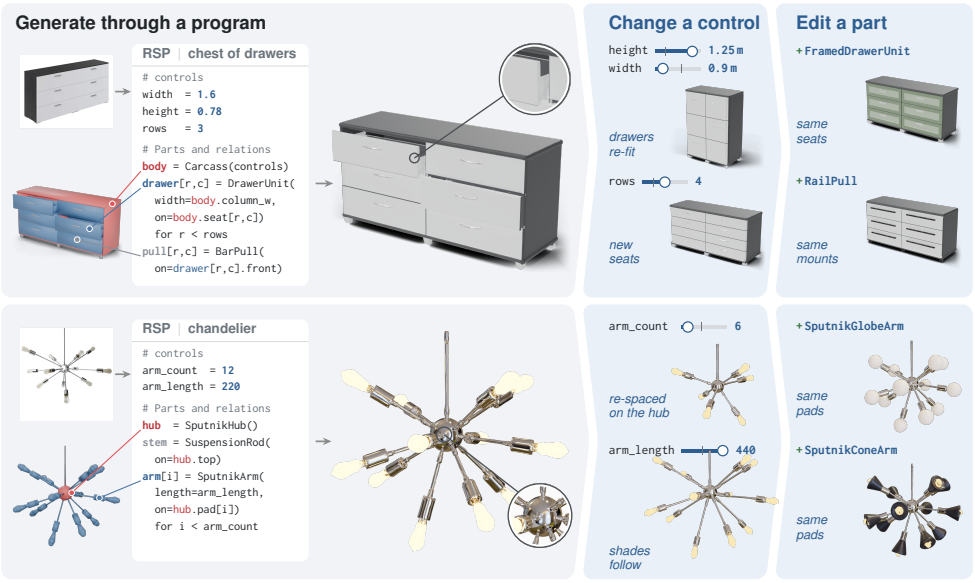

Figure 1: **RelShape generates a realistic 3D asset through a program whose controls keep its relations.** Left: from an input photo, our agent writes a Relational Shape Program (RSP); executing it gives an instance built from parts (colored by role; lines link parts to the code that builds them), which conditions a pretrained generator to produce the complete asset (center; insets magnify a drawer on its seat and the arms on the hub). Right: changing a control rebuilds the affected parts and places them through the same relations.

parts and configuration and takes its part labels from the program. Revising or varying the asset then changes a control, and the new instance is materialized the same way (Figure 1). The assets are only as reliable as this model, and synthesizing it is the hard part: many procedural models reproduce the single instance in the input yet vary incorrectly; for example, a chest whose drawers have a fixed width looks correct until it is narrowed. Reliable variation needs relations that the program states explicitly and controls whose effects hold beyond the input's setting; we provide the first with a representation and the second with an agent that synthesizes it.

We represent the procedural model as a *Relational Shape Program* (RSP), which separates how each part is built from how it is assembled. Each part is constructed by its own code and exposes interfaces computed from its geometry, and assembly places parts only through these interfaces. Explicit dependencies derive the parameters of every part from a few object-level controls. Changing a control therefore rebuilds the affected parts and reassembles them through unchanged relations.

A 3D coding agent synthesizes the RSP *variation-first*. Before writing any construction code, it declares the object-level controls the RSP will expose, with what each must change and what it must keep. It then builds the RSP, checks the default object against the input with a visual critic, and executes the RSP at non-default values, keeping only the controls whose declared effects hold. The model is thus tested at settings the input never shows.

Together, these components target the three properties at once, and we evaluate each of them on 64 text descriptions (Sinha et al., 2025) and 32 product images (Collins et al., 2022): our assets rival those of 3D generative models and surpass those of program-based methods, and more of the program's structure survives our materialization than a program-to-generator pipeline's. At unseen settings, our RSPs also follow their controls more reliably than those programs, and ablations attribute this to the RSP's explicit dependencies and interfaces and to declaring the controls before construction and executing them at non-default values.

Our contributions are as follows.

- **Program-mediated generation.** We propose a new paradigm that shifts the output of 3D generation from a static shape to a procedural model. Materializing its instances with

a pretrained 3D generator, it unifies the generation and structural editing of realistic 3D assets.

- **Relational Shape Programs (RSPs).** We introduce a novel procedural representation that joins parts through interfaces computed from their own geometry, not through coordinates or bounding boxes. Part parameters derive from object-level controls, so the assembly remains intact as controls change or parts are replaced.
- **A variation-first 3D coding agent.** We design a 3D coding agent that plans an object's variation before its structure and verifies both by execution: it tests every control beyond the input's setting and keeps only those that pass, so its programs vary reliably.

2 Related Work

3D generative models. Native 3D generative models such as TRELLIS, TRELLIS.2 and Hunyuan3D 2.1 learn realistic geometry and materials from large asset collections (Xiang et al., 2025; 2026; Tencent Hunyuan3D Team, 2025), and LLaMA-Mesh writes meshes as text (Wang et al., 2024). Part-aware models output an object as separate parts, generated jointly in PartCrafter (Lin et al., 2025) or from planned part boxes in OmniPart (Yang et al., 2025), and generated assets can be edited through multi-view images (Erkoç et al., 2025). In their output, how the parts attach and which dimensions depend on which stay implicit. We use TRELLIS.2 to materialize our assets and take these relations from the program.

Spatial control of pretrained generators. Coin3D (Dong et al., 2024) guides generation with a proxy assembled from basic shapes, and SpaceControl (Fedele et al., 2026) adds test-time control from geometry ranging from primitives to detailed meshes. In these methods the condition is a given geometry. In program-mediated generation, the condition is each instance that a synthesized procedural model executes, so generating, revising and varying an asset pass through one procedural model, and each asset keeps the instance's parts and configuration.

Procedural shape representations. Procedural models represent an object by the program that constructs it, which makes its parts, relations and variation explicit. Such programs are written by hand for each category, as in Infinigen (Raistrick et al., 2024), organized around shared concepts (Sun et al., 2024; Sun & Lu, 2025), inferred from data (Jones et al., 2020; 2024; 2026), or recovered for existing programs and shapes (Pearl et al., 2025; Ganeshan et al., 2024), and Procedura (Lin et al., 2026) places parts through typed mates. RSPs are designed to be synthesized from a single input, without a template for its category. Each part exposes interfaces computed from its geometry, assembly places parts only through them, and explicit dependencies derive part parameters from object-level controls.

Program-based methods and 3D coding agents. Program-based methods write 3D programs from text or images (Zhang et al., 2025b;a), including CAD models (Gong et al., 2026) and articulated objects (Gao et al., 2026; Le et al., 2025); others expose procedural graphs or named parts for editing (Raji et al., 2026; Noor et al., 2026). Concurrent work (Wang et al., 2026) pairs a shape language with relational placement operators and a plan–execute–critic agent, and materializes its output with SpaceControl. In these methods, programs are judged by what they build at the input's setting, by how their joints move or by edits after generation. Our agent designs the variation of an RSP before its shape and keeps only the controls whose declared effects hold at settings the input never shows.

3 Method

Given a single text description or image of an object, RelShape performs program-mediated generation: it synthesizes a procedural model of the object and materializes the instances it executes into complete assets (Figure 2). The procedural model is a Relational Shape Program (RSP) P with object-level controls of two kinds: *design controls* d set dimensions and member counts, and *configuration controls* q set joint states, such as the opening angle of a door. Executing P at a setting

yields an *instance*, and the instances form the *family*

$$\mathcal{F}(P) = \{\mathcal{E}(P; d, q) \mid (d, q) \in \Omega_P\}, \tag{1}$$

where Ω_P is the domain that P declares for its controls and $\mathcal{E}$ builds and assembles the parts; $G_t = \mathcal{E}(P; d_t, q_t)$ denotes the instance at a setting t. A revision within the family changes only (d, q); a revision beyond it, such as adding a part, is an edit of P.

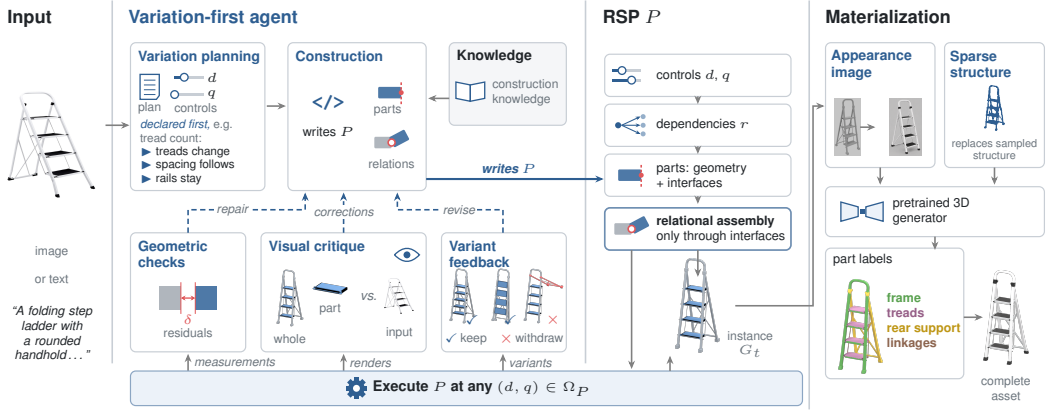

Figure 2: Program-mediated generation, shown on a folding step ladder. From a single text or image, the agent (Section 3.2) writes an RSP P and declares its controls before construction; it revises P from the measurements, renders and variants that executing P returns (dashed arrows) and keeps only the controls whose declared effects hold. Executing P at a setting yields an instance G_t (Section 3.1), which materialization (Section 3.3) turns into a complete asset whose layout is the instance's own.

3.1 Relational Shape Programs

An RSP separates how each part is built from how the parts are assembled, and assembles them only through interfaces that each part computes from its own geometry. Changing a control therefore rebuilds the affected parts, and the same relations reassemble them. A fence with a gate serves as the running example (Figure 3).

Parts and interfaces. A *part* is a unit with one rigid pose, such as the gate; its constructor C_i may contain many members, such as repeated planks. Besides the part's geometry G_i, the constructor computes its *interfaces* F_i from that geometry: surface regions, member ends, joint axes and mounting regions, expressed in the part's own frame (Figure 3a). Explicit rules r derive the parameters $p = (p_1, \ldots, p_n)$ of all parts from the design controls:

$$p = r(d), \qquad (G_i, F_i) = C_i(p_i), \qquad \Pi = a(p, q), \tag{2}$$

where a states the assembly *relations* Π. A *dependency* stated in r holds at every setting in Ω_P, so one control resizes every part that depends on it.

Relational assembly. A root part anchors the object, and every other part receives its rigid transform T_i from exactly one relation over current interfaces, such as a surface mated to a surface or a revolute joint between two axes whose angle comes from q. Each relation is evaluated once the parts whose interfaces it uses are in place and computes its part's pose directly from them:

$$\{T_i\} = \mathrm{Assemble}(\Pi, \{F_i\}), \qquad \mathcal{E}(P; d, q) = \mathrm{Compose}\big(\{T_i G_i\}\big). \tag{3}$$

Coordinates fixed at the default setting (Zhang et al., 2025a) still point to where a rebuilt neighbor's surface used to be, and bounding-box anchors (Wang et al., 2026) need not lie on the geometry that carries a joint or a contact, such as a hinge barrel; an interface is that geometry itself.

Controls and part swaps. Because relations refer only to interfaces, one set of relations serves every setting of the controls and every compatible construction of a part (Figure 3b–e). A design control rebuilds the parts that depend on it, and each relation places a rebuilt part from its recomputed interface. A count control changes the members inside a part or the number of copies of a part, and a relation of the same form places each copy on its own interface. A configuration control only sets the angle of a joint relation, so it moves rigid parts without reshaping them. Replacing a part's construction needs an edit of its constructor alone: a new construction that computes a compatible interface is placed by the existing relation and follows the existing controls. Appendix A gives the program semantics and this example.

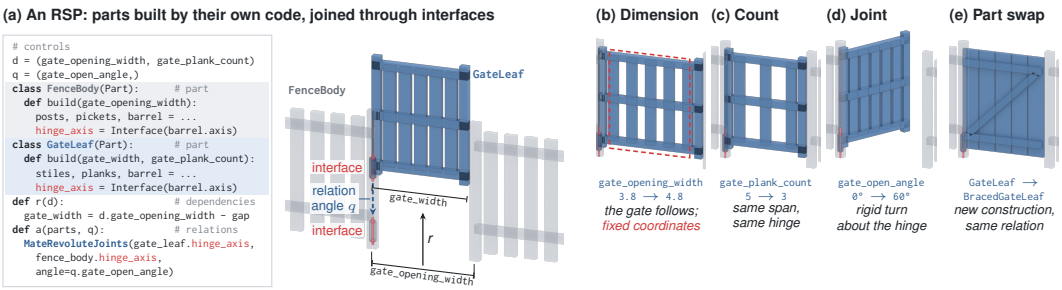

Figure 3: An RSP builds each part with its own code and joins the parts only through interfaces computed from their geometry. (a) The fence-and-gate RSP (abridged) beside the parts it builds, with the gate lifted along its hinge axis: interfaces in red, the relation in blue and the dependency r from the opening to the gate width. (b–d) The same program at other settings; in (b), the red outline marks where coordinates fixed at the default would leave the gate. (e) A gate with a new construction, hung by the unchanged relation.

3.2 A Variation-First 3D Coding Agent

An LLM coding agent writes the RSP as Python code against a geometry runtime, designing the object's variation before its shape. It observes the object only by executing the program, through renders and geometric measurements, and verifies both variation and shape in three stages (Algorithm 1).

Variation planning. Before any whole object is executed, the agent declares the controls the RSP will expose. A declaration states a control's domain and three kinds of effect: its *targets*, which must change; its *propagation*, the dimensions and placements that must follow; and its *preservation*, the attachments, counts and shapes that must stay fixed. The domain is the range the object's function allows, such as a hinge's full opening, not the value depicted in the input. Declared first, the controls become requirements of the construction, so parts, dependencies and joints are built to deliver them; controls chosen afterwards only describe what the finished program happens to allow.

Construction and visual critique. The agent then writes the constructors, dependencies and relations and executes the program. A visual critic compares renders of the whole object and of every isolated part with the input and returns its findings to the agent. The critic does not see the agent's reasoning, so it judges what was built rather than what was intended.

Variant feedback. Finally, the agent executes every declared control at non-default values, including values near both ends of its domain. Each test changes one control from the default and measures its declared effects on the resulting geometry, such as extents, part counts and distances between interfaces. Every part that a joint moves must also keep its shape. The agent revises and retests until each control's declared effects hold and keeps only the controls that pass, so every control the RSP exposes is confirmed.

3.3 MATERIALIZATION

Program geometry makes the structure explicit but has no realistic detail or materials. A pretrained 3D generator (Xiang et al., 2026) supplies them, yet conditioned on text or an image alone it does not keep the counts, dimensions and joint states that an instance fixes. Materialization therefore conditions it, without changing its weights, on the executed instance G_t at each requested setting t, through two signals.

Appearance image. An image-editing model, guided by the input, turns a render of G_t into an appearance image that keeps the render's counts, layout, configuration and viewpoint.

Sparse structure. The instance takes the place of the first of TRELLIS.2's three stages, which generate a sparse structure of occupied voxels, a shape latent on those voxels and a texture latent. The instance, made watertight and normalized as $\widetilde{G}_t$, is encoded from its O-Voxel representation by the TRELLIS.2 shape encoder E_{shape}, and the occupied coordinates of the result,

$$\mathcal{S}_t = \text{coords}\Big(E_{\text{shape}}\Big(\text{OVoxel}(\widetilde{G}_t)\Big)\Big)\,, \tag{4}$$

replace the sampled sparse structure; the shape and texture stages sample on them, conditioned on the appearance image. The asset thus inherits the instance's layout, while surface detail and physically based rendering (PBR) materials come from the generator; guidance that re-noises an encoded structure trades this faithfulness against realism through a strength parameter (Fedele et al., 2026). Appendix C gives the sampling settings and the image-editing model.

Part labels. The asset's parts come from the program: every face takes the label of the nearest part of the instance, and the asset is exported with one node per part (Figure 2, right) without a segmentation model.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Tasks and data. To test generation from both kinds of input, text-to-3D uses 64 objects from 32 categories, each described by the MARVEL caption (Sinha et al., 2025) of an Objaverse object with PartNeXt part annotations (Deitke et al., 2023; Wang et al., 2025). Image-to-3D uses 32 ABO products (Collins et al., 2022), one RGB image each; the ground-truth mesh is used only for evaluation. To test controls, 16 of the text objects receive two goals each, such as a shelf count or a coupled width, with three test values per goal (96 settings). Goals and the editing requests of Section 4.3 were written from the input before any output was seen, and the goals were never shown to the agent.

Baselines. To compare with the closest methods, we use published program-synthesis methods with public code, ShapeCraft (Zhang et al., 2025a), Scene Language (Zhang et al., 2025b) and Graph-CAD (Gong et al., 2026), and the concurrent aDSL (Wang et al., 2026); except Graph-CAD, which uses its released fine-tuned models and is evaluated on generation only, they run on the same LLM as our agent, DeepSeek-V4.1-Flash. Direct 3D generators serve as unranked references: TRELLIS-text (Xiang et al., 2025), LLaMA-Mesh (Wang et al., 2024), TRELLIS.2 (Xiang et al., 2026) and Hunyuan3D 2.1 (Tencent Hunyuan3D Team, 2025) for text, and TRELLIS.2, Hunyuan3D 2.1 and PartCrafter (Lin et al., 2025) for images.

Metrics. The metrics test the three properties of Section 1: realistic assets that match the input, explicit structure and reliable variation. VQAScore (Lin et al., 2024) and Long-CLIP (Zhang et al., 2024) measure text alignment and CLIP-I (Radford et al., 2021) image alignment; on ABO, Chamfer distance (CD) and F-score measure surface accuracy against the ground-truth mesh. Structural fidelity (Struct.) is the fraction of input-grounded questions about parts, counts, shapes and relations that a VLM judge answers correctly (Hu et al., 2023; Cho et al., 2024; Zhang et al., 2025a). GPTEval3D (Wu et al., 2024) rates 3D plausibility as Elo with Qwen3-VL (Bai et al., 2025) as the judge, and Valid is the fraction of tasks with a usable mesh. For controls, Coverage is the fraction of goals with an existing control, Target the fraction of settings that reach the goal value, Success the fraction

that also keep every required relation, and Rel. acc. the relation-state accuracy of ParSEL (Ganeshan et al., 2024).

4.2 Text- and Image-to-3D Generation

To test whether materialization keeps the program's structure, Table 1 scores two stages of the same runs: *geometry* (the executed program, or a method's native mesh) and *complete assets* (each method's final output).

Table 1: Text- and image-to-3D generation. VQA (VQAScore), Long-CLIP, CLIP-I ($\times 100$); CD ($\times 10^3$); F, Struct., Valid (%). Elo is fitted separately for geometry and for complete assets. Reference generators (grey) are not ranked; bold marks the best program-based result.

(a) Text-to-3D, 64 objects

	Geometry					Complete asset				
Method	VQA↑	L-CLIP↑	Struct.↑	Elo↑	Valid↑	VQA↑	L-CLIP↑	Struct.↑	Elo↑	Valid↑
ShapeCraft / +FlashTex*	57.4	21.6	74.5	1062	98	61.4	22.1	76.0	687	98
Scene Language	59.7	22.0	85.2	1163	**100**	–	–	–	–	–
Graph-CAD	49.3	20.5	58.6	961	98	–	–	–	–	–
aDSL§ / +SpaceControl*	63.5	22.4	84.9	1196	**100**	73.9	23.4	77.2	1009	**100**
Ours	**66.6**	**22.9**	**87.9**	**1317**	**100**	**76.0**	**24.0**	**90.1**	**1180**	**100**
TRELLIS-text	68.1	22.6	66.4	1130	100	75.6	23.6	65.3	988	100
TRELLIS.2†	70.7	22.5	83.9	1408	100	82.0	24.7	81.1	1323	100
Hunyuan3D 2.1†	–	–	–	–	–	78.9	24.2	80.8	1175	100
LLaMA-Mesh	38.2	18.2	36.4	604	92	–	–	–	–	–

(b) Image-to-3D, 32 ABO objects

	Geometry						Complete asset					
Method	CLIP-I↑	CD↓	F↑	Struct.↑	Elo↑	Valid↑	CLIP-I↑	CD↓	F↑	Struct.↑	Elo↑	Valid↑
Scene Language	80.6	16.82	11.9	89.6	907	**100**	–	–	–	–	–	–
aDSL§ / +SpaceControl*	81.7	14.10	17.1	89.6	927	**100**	85.6	18.40	15.9	90.6	904	**100**
Ours	**82.7**	**12.69**	**20.0**	**90.6**	**1062**	**100**	**87.5**	**11.45**	**20.0**	**93.8**	**1093**	**100**
TRELLIS.2	84.0	7.91	30.1	94.8	1094	100	88.3	7.91	30.1	95.8	1086	100
Hunyuan3D 2.1	81.7	27.08	21.6	82.3	980	100	86.8	27.03	21.2	85.4	998	100
PartCrafter‡	81.5	17.44	14.6	83.3	–	97	–	–	–	–	–	–

*Complete assets: FlashTex (Deng et al., 2024) textures ShapeCraft geometry, whose painting stage is unreleased, and SpaceControl (Fedele et al., 2026) materializes aDSL geometry, following aDSL. †From a Qwen-Image (Qwen Team, 2025) image of the description. §Concurrent work. ‡With a VLM-selected part count; geometry only. –: not produced or not evaluated.

Program-based methods. The program-based methods test what program mediation adds to program synthesis: among them, RelShape is best on every metric for both inputs, as geometry and as complete assets. Materialization keeps the program's structure, since the asset is sampled on the instance's own occupied voxels (Section 3.3): 95.8% of the structural questions answered correctly on the geometry stay correct on the asset, so the lead in structure widens from geometry to complete asset.

Reference generators. The direct generators test whether the assets rival 3D generative models: they return no program, yet RelShape's assets answer more structural questions on text and receive the highest Elo of all methods on images. The RSP states the counts and sides that the input describes, and materialization keeps them in the asset (Figure 4).

4.3 Structural Control and Editing

To test whether the programs vary reliably, we measure the controls a program already contains at held-out settings and show revisions on requests fixed before any edit.

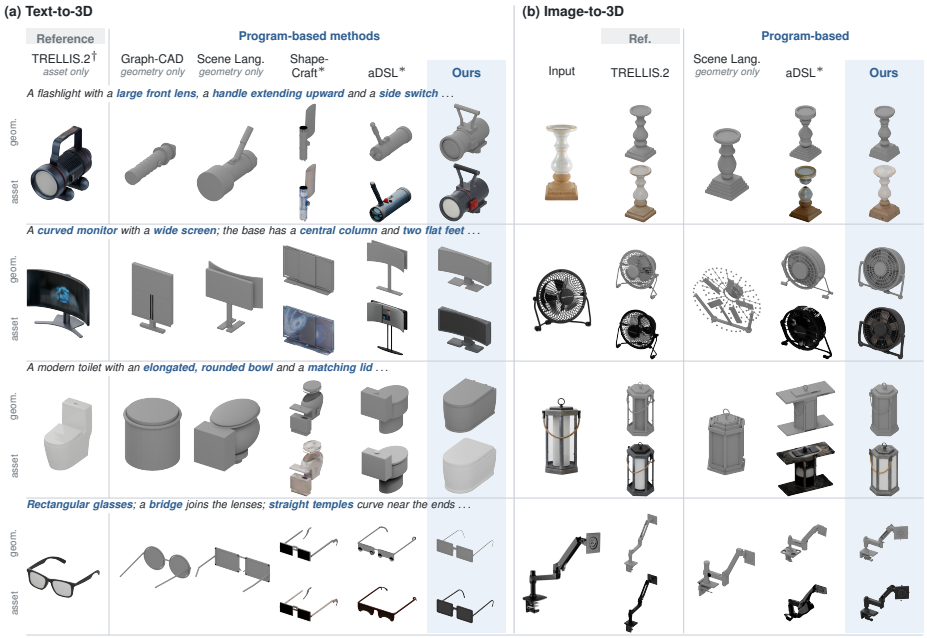

Figure 4: Qualitative comparison on (a) text and (b) image inputs. For each object, the upper row shows geometry and the lower row the complete asset; methods marked *only* produce a single stage, shown across both rows. Highlighted phrases name the structure the prompt asks for. †, *: as in Table 1. More examples are in Appendix D.

Held-out settings. To test the controls beyond the agent's own checks, we freeze each generated program and set only its existing controls, with no new code or LLM calls. The values are ones the agent never executed, chosen regardless of the domain Ω_P it declared (Eq. 1). Like our agent, the baselines never see the goals: their programs run as published, with their parameters bound to the same goals.

Table 2: Controls at held-out settings on 16 text objects, 32 goals, 96 settings (%). All columns count every setting, and a goal without a control or an invalid output counts as a failure; Rel. acc. counts all 261 assigned relation checks.

Method	Coverage↑	Success↑	Target↑	Rel. acc.↑	Time / variation↓
ShapeCraft (as published)	9.4	3.1	3.1	5.7	44.9 s
Scene Language (as published)	56.3	49.0	53.1	51.0	189.1 s
aDSL (as published)	62.5	40.6	43.8	55.9	77.9 s
Ours	**93.8**	**71.9**	**77.1**	**78.5**	**24.5 s**

Table 2 tests whether the programs follow their controls, and RelShape leads on every measure. Its RSPs expose a control for nearly every goal, including every angle goal and most count goals, which need a joint that turns a rigid part or members generated and attached by rule. Most dimension settings succeed because an explicit dependency carries the change to every part that shares the value. Figure 5(d–e) shows such settings: dependent parts are reshaped, and parts that only move stay rigid. A variation only re-executes the program, so it takes seconds instead of the 22 minutes of generating the object again.

Instruction-guided edits. To test revision beyond the family, we give the agent requests that need a control the program lacks; it revises the RSP and keeps its interfaces and controls, so the edited object remains a controllable family. In Figure 5(b–c), only the edited parts change, and the original control still drives the edited program (d–e).

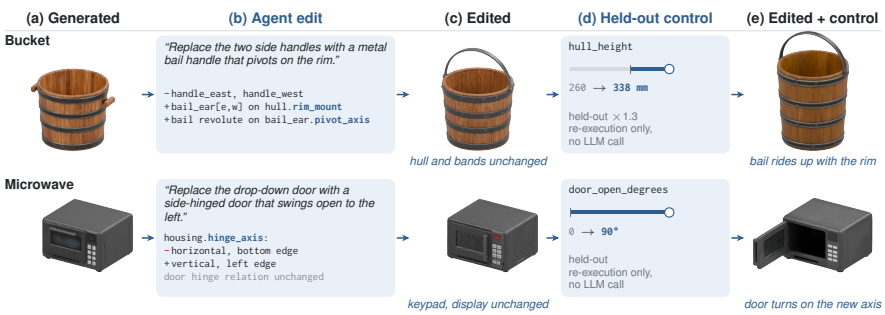

Figure 5: Edits keep a generated RSP a working family. (a) The generated asset. (b) The agent edits the RSP for the quoted request; the lines shown are the changed parts and relations, with the interfaces they use in blue. (c) The edited asset. (d) An original control is set to a held-out value by re-execution alone, and (e) it drives the edited parts. Appendix E gives more examples.

4.4 Ablation Study

To test which elements make the controls hold, we remove one element of the representation or the agent per row on the 16 objects of Section 4.3, with one run per object and the same system version, tools, budget and backbone in every row (Table 3).

Table 3: Ablation of the representation and the agent on 16 objects (%). Struct., VQA and Valid score default geometry; the other columns follow Table 2.

Variant	Valid↑	Struct.↑	VQA↑	Coverage↑	Success↑	Target↑	Rel. acc.↑	Tokens
Full	100	89.3	71.0	93.8	71.9	77.1	78.5	1.10M
Representation								
w/o geometric interfaces	100	93.9	70.6	90.6	62.5	68.8	69.0	0.79M
w/o explicit dependencies	100	88.2	66.9	71.9	53.1	54.2	53.3	0.98M
Agent								
w/o variation planning	100	89.5	66.3	75.0	45.8	49.0	56.3	0.74M
w/o visual critique	100	88.9	67.6	81.3	62.5	67.7	71.3	0.94M
w/o variant feedback	100	90.8	67.9	78.1	51.0	53.1	58.6	0.93M

Representation. The first two rows test the representation's two elements, and both make the controls hold (Table 3). Without geometric interfaces, parts sit at fixed numeric poses, so a change can separate a part from what it should touch, as when a count adds chandelier arms. Without explicit dependencies, each control sets a single part, so shared widths and repeated counts lose control.

Agent. The last three rows test the agent's stages: each contributes, and designing controls first and testing them by execution contribute most (Table 3). Without variation planning, controls are chosen after the object is built, and coverage and success fall furthest, since declared controls become requirements of the construction (Section 3.2). Without variant feedback, the agent is still told to design controls but never executes them at non-default values, and fewer controls reach the held-out settings, so the gain comes from execution rather than the instruction. The visual critic raises both alignment with the input and control success.

5 Conclusion

We presented program-mediated generation, which turns a single text or image into a Relational Shape Program whose instances materialize into complete assets; with explicit relations and controls verified by execution, one generated program serves many related assets and later edits. Families are bounded by the agent's choice of parts, relations and control domains, and single views leave hidden geometry ambiguous.

AI USE STATEMENT

In this work, we used generative AI tools for research ideation and execution, including implementing methods, designing and running experiments and analyzing their results, and for the retrieval and discovery of related work. Additionally, we used generative AI tools to draft sections of the paper and to aid and polish its writing. We have reviewed all AI-assisted work: AI-generated code was run and its outputs checked, the reported results come from recorded experiment logs, and related work found with AI tools was checked against the original papers. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sections 3 and 4.1 describe the method and the experimental setup, and Appendices A–C give the RSP semantics with the program of Figure 3, the agent's stages, instructions and budget, and the materialization settings. Supplementary material is available at https://anonymous.4open.science/r/relshape-supp-D44C.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL Technical Report. arXiv preprint arXiv:2511.21631, 2025. URL https://arxiv.org/abs/2511.21631.

Jaemin Cho, Yushi Hu, Roopal Garg, Peter Anderson, Ranjay Krishna, Jason Baldridge, Mohit Bansal, Jordi Pont-Tuset, and Su Wang. Davidsonian Scene Graph: Improving Reliability in Fine-grained Evaluation for Text-to-Image Generation. In *International Conference on Learning Representations*, 2024. doi: 10.48550/ARXIV.2310.18235. URL https://arxiv.org/abs/2310.18235.

Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F. Yago Vicente, Thomas Dideriksen, Himanshu Arora, Matthieu Guillaumin, and Jitendra Malik. ABO: Dataset and Benchmarks for Real-World 3D Object Understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21126–21136, June 2022.

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 (CVPR)*, pp. 13142–13153, June 2023.

Kangle Deng, Timothy Omernick, Alexander Weiss, Deva Ramanan, Jun-Yan Zhu, Tinghui Zhou, and Maneesh Agrawala. FlashTex: Fast Relightable Mesh Texturing with LightControlNet. In *European Conference on Computer Vision (ECCV)*, 2024. URL https://arxiv.org/abs/2402.13251.

Wenqi Dong, Bangbang Yang, Lin Ma, Xiao Liu, Liyuan Cui, Hujun Bao, Yuewen Ma, and Zhaopeng Cui. Coin3D: Controllable and Interactive 3D Assets Generation with Proxy-Guided Conditioning. In *ACM SIGGRAPH 2024 Conference Papers*, 2024. URL https://arxiv.org/abs/2405.08054.

Ziya Erkoç, Can Gümeli, Chaoyang Wang, Matthias Nießner, Angela Dai, Peter Wonka, Hsin-Ying Lee, and Peiye Zhuang. PrEditor3D: Fast and precise 3d shape editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 640–649, 2025. doi: 10.1109/CVPR52734.2025.00068.

Elisabetta Fedele, Francis Engelmann, Ian Huang, Or Litany, Marc Pollefeys, and Leonidas Guibas. SpaceControl: Introducing Test-Time Spatial Control to 3D Generative Modeling. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=mEqsCVI5sN.

Aditya Ganeshan, Ryan Huang, Xianghao Xu, R. Kenny Jones, and Daniel Ritchie. ParSEL: Parameterized Shape Editing with Language. *ACM Transactions on Graphics*, 43(6):1–14, November 2024. ISSN 1557-7368. doi: 10.1145/3687922.

Yipeng Gao, Yunhao Ge, Peilin Cai, Daniel Seita, and Laurent Itti. LAM: Language Articulated Object Modelers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16010–16020, June 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Gao_LAM_Language_Articulated_Object_Modelers_CVPR_2026_paper.html.

Shengjie Gong, Wenjie Peng, Hongyuan Chen, Gangyu Zhang, Yunqing Hu, Huiyuan Zhang, Shuangping Huang, and Tianshui Chen. Learning Hierarchical and Geometry-Aware Graph Representations for Text-to-CAD. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=oKMomThD6n.

Yushi Hu, Benlin Liu, Jungo Kasai, Yizhong Wang, Mari Ostendorf, Ranjay Krishna, and Noah A. Smith. TIFA: Accurate and Interpretable Text-to-Image Faithfulness Evaluation with Question Answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 20406–20417, October 2023.

R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. ShapeAssembly: learning to generate programs for 3D shape structure synthesis. *ACM Transactions on Graphics*, 39(6):1–20, November 2020. ISSN 1557-7368. doi: 10.1145/3414685.3417812.

R. Kenny Jones, Siddhartha Chaudhuri, and Daniel Ritchie. Learning to Infer Generative Template Programs for Visual Concepts. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 22465–22490. PMLR, 2024. URL https://proceedings.mlr.press/v235/jones24a.html.

R. Kenny Jones, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. ShapeLib: Designing a library of programmatic 3D shape abstractions with Large Language Models. *ACM Transactions on Graphics*, 2026. doi: 10.1145/3840293. URL https://doi.org/10.1145/3840293.

Long Le, Jason Xie, William Liang, Hung-Ju Wang, Yue Yang, Yecheng Jason Ma, Kyle Vedder, Arjun Krishna, Dinesh Jayaraman, and Eric Eaton. Articulate-Anything: Automatic Modeling of Articulated Objects via a Vision-Language Foundation Model. In *International Conference on Learning Representations*, volume 2025, pp. 17578–17602, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/2c8047bf3ed8ef6905351608d641f02f-Abstract-Conference.html.

Youtian Lin, Yikang Yang, Zhanpeng Hu, Mengqi Zhou, Feihu Zhang, Xun Cao, Jiaheng Liu, and Yao Yao. Procedura: Agentic 3D Modeling with Procedural Control. arXiv preprint arXiv:2608.26238, 2026. URL https://arxiv.org/abs/2608.26238.

Yuchen Lin, Chenguo Lin, Panwang Pan, Honglei Yan, Yiqiang Feng, Yadong Mu, and Katerina Fragkiadaki. PartCrafter: Structured 3D Mesh Generation via Compositional Latent Diffusion Transformers. In *Advances in Neural Information Processing Systems*, volume 38, pp. 35387–35415, 2025. doi: 10.52202/085713-1189. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/32cc61322f1e2f56f989d29ccc7cfbb7-Abstract-Conference.html.

Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. Evaluating Text-to-Visual Generation with Image-to-Text Generation. In *Computer Vision – ECCV 2024*, pp. 366–384. Springer Nature Switzerland, October 2024. ISBN 9783031726736. doi: 10.1007/978-3-031-72673-6_20.

Nimra Noor, Muhammad Bilal, Abdullah Hussain, and Hassan Baig. Nova3D: Code-Native Generation of Programmable 3D Assets. arXiv preprint arXiv:2607.22738, 2026. URL https://arxiv.org/abs/2607.22738.

Ofek Pearl, Itai Lang, Yuhua Hu, Raymond A. Yeh, and Rana Hanocka. GeoCode: Interpretable Shape Programs. *Computer Graphics Forum*, 44(1), February 2025. ISSN 1467-8659. doi: 10.1111/cgf.15276.

Qwen Team. Qwen-Image-2512. Model release, 2025. URL https://huggingface.co/Qwen/Qwen-Image-2512.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, Zeyu Ma, and Jia Deng. Infinigen Indoors: Photorealistic Indoor Scenes using Procedural Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21783–21794, June 2024. doi: 10.1109/CVPR52733.2024.02058. URL https://openaccess.thecvf.com/content/CVPR2024/html/Raistrick_Infinigen_Indoors_Photorealistic_Indoor_Scenes_using_Procedural_Generation_CVPR_2024_paper.html.

Fadlullah Raji, Stefano Petrangeli, Matheus Gadelha, Yu Shen, Uttaran Bhattacharya, and Gang Wu. Proc3D: Procedural 3D Generation and Parametric Editing of 3D Shapes with Large Language Models. arXiv preprint arXiv:2601.12234, 2026. URL https://arxiv.org/abs/2601.12234.

Sankalp Sinha, Mohammad Sadil Khan, Muhammad Usama, Shino Sam, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. MARVEL-40M+: Multi-Level Visual Elaboration for High-Fidelity Text-to-3D Content Creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8105–8116, June 2025.

Jianhua Sun and Cewu Lu. Digital Gene: Learning about the Physical World through Analytic Concepts. arXiv preprint arXiv:2504.04170, 2025. URL https://arxiv.org/abs/2504.04170.

Jianhua Sun, Yuxuan Li, Longfei Xu, Nange Wang, Jiude Wei, Yining Zhang, and Cewu Lu. ConceptFactory: Facilitate 3D Object Knowledge Annotation with Object Conceptualization. In *Advances in Neural Information Processing Systems 37*, NeurIPS 2024, pp. 75454–75467. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.52202/079017-2402.

Tencent Hunyuan3D Team. Hunyuan3D 2.1: From Images to High-Fidelity 3D Assets with Production-Ready PBR Material. arXiv preprint arXiv:2506.15442, 2025.

Penghao Wang, Yiyang He, Xin Lv, Yukai Zhou, Lan Xu, Jingyi Yu, and Jiayuan Gu. PartNeXt: A Next-Generation Dataset for Fine-Grained and Hierarchical 3D Part Understanding. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-2030. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/579636b69390b41fa2d39b744593bfbe-Paper-Datasets_and_Benchmarks_Track.pdf.

Rui-Huan Wang, Si-Tong Wei, Jia-Qi He, Heng-Yi Wei, Baoquan Chen, and Peng-Shuai Wang. aDSL: Agentic 3D Creation via Joint Agent-Program Design. arXiv preprint arXiv:2608.17975, 2026. URL https://arxiv.org/abs/2608.17975.

Zhengyi Wang, Jonathan Lorraine, Yikai Wang, Hang Su, Jun Zhu, Sanja Fidler, and Xiaohui Zeng. LLaMA-Mesh: Unifying 3D Mesh Generation with Language Models. arXiv preprint arXiv:2411.09595, 2024. URL https://arxiv.org/abs/2411.09595.

Tong Wu, Guandao Yang, Zhibing Li, Kai Zhang, Ziwei Liu, Leonidas Guibas, Dahua Lin, and Gordon Wetzstein. GPT-4V(ision) is a human-aligned evaluator for text-to-3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3D Latents for Scalable and Versatile 3D Generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21469–21480. IEEE, June 2025. doi: 10.1109/cvpr52734.2025.02000.

Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, and Jiaolong Yang. Native and Compact Structured Latents for 3D Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Xiang_Native_and_Compact_Structured_Latents_for_3D_Generation_CVPR_2026_paper.html.

Yunhan Yang, Yufan Zhou, Yuan-Chen Guo, Zi-Xin Zou, Yukun Huang, Ying-Tian Liu, Hao Xu, Ding Liang, Yan-Pei Cao, and Xihui Liu. OmniPart: Part-Aware 3D Generation with Semantic Decoupling and Structural Cohesion. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*. Association for Computing Machinery, 2025. doi: 10.1145/3757377.3763872. URL https://doi.org/10.1145/3757377.3763872.

Beichen Zhang, Pan Zhang, Xiaoyi Dong, Yuhang Zang, and Jiaqi Wang. Long-CLIP: Unlocking the Long-Text Capability of CLIP. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, volume 15109 of *Lecture Notes in Computer Science*, pp. 310–325. Springer, 2024. doi: 10.1007/978-3-031-72983-6_18. URL https://doi.org/10.1007/978-3-031-72983-6_18.

Shuyuan Zhang, ChenHan Jiang, Zuoou Li, and Jiankang Deng. ShapeCraft: LLM Agents for Structured, Textured and Interactive 3D Modeling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 65116–65144. Curran Associates, Inc., 2025a. doi: 10.52202/085713-2181. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/5e2217482fa75556f1970be809acd3f8-Paper-Conference.pdf.

Yunzhi Zhang, Zizhang Li, Matt Zhou, Shangzhe Wu, and Jiajun Wu. The scene language: Representing scenes with programs, words, and embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025b. URL https://ai.stanford.edu/~yzzhang/projects/scene-language/.

A RSP SEMANTICS AND A COMPLETE EXAMPLE

This section defines the interfaces, relations and controls of an RSP and gives the program of Figure 3.

Interfaces. A constructor computes every interface of its part from the part's own geometry and expresses it in the part's frame, so each rebuild of the part recomputes its interfaces. There are four kinds: a surface region with its outward normal; a member end, or a group of member ends, with its position and direction; a joint axis with a reference direction for its angle; and a mounting region that serves as a reference for placement.

Relations and checks. A root part anchors the object, and every other part has exactly one relation, so the relations form a tree rooted at the root part. A relation is evaluated once the parts whose interfaces it uses are placed and computes its part's pose directly (Table 4); the same setting therefore always yields the same object. Checks measure contacts between surfaces and shared axes that no relation places, such as a second hinge on the same door, and report them without moving any part.

Table 4: Relations of an RSP. Each places one part from interfaces computed at the current setting.

Relation	Interfaces used	Pose of the placed part
Placement	a mounting region of another part	a pose given in the frame of that region
Surface mate	a surface region on each part	its surface is brought into contact with the other surface, normals opposed
Member-end attach	member ends of the placed part and a surface region of the other part	its member ends are placed on that surface
Revolute joint	a joint axis on each part	the two axes coincide, and the part is turned about them by an angle from q

Controls and dependencies. Design controls d are named values with declared domains. The rules r compute the parameters of every part from d, so a dependency such as the gate width following the opening holds at every setting. A count control changes the number of members inside a constructor or the number of copies of a part, and each copy is placed by its own relation. Configuration controls q set the angles of revolute relations.

The fence-and-gate program. The program of Figure 3 has two parts, the fence as root and the gate (below, with the member construction elided). Each constructor computes its hinge axis from its own hinge barrel, one revolute relation joins the two axes at the opening angle, and one dependency sets the gate width from the opening. The three controls of Figure 3b–d are the opening width (design), the number of gate planks (count) and the opening angle (configuration). The braced gate of Figure 3e replaces `GateLeaf` with a constructor that builds boards, ledges and a diagonal brace and computes the same `hinge_axis` from its own barrel, so the unchanged relation hangs it and the three controls still apply.

```
class FenceBody(ConceptTemplate):                        # root part
    def __init__(self, fence_width, fence_height, gate_opening_width,
                 fence_plank_count, ...):
        components = self._build_components(...)         # posts, rails, planks, barrel
        self._finalize_mesh(components)
        barrel = next(c for c in components if c.name == "fixed_hinge_barrel")
        self.expose_revolute_joint_port("hinge_axis", barrel, "axis")

class GateLeaf(ConceptTemplate):
    def __init__(self, gate_width, gate_height, gate_plank_count, ...):
        components = self._build_components(...)         # stiles, rails, planks, barrel
        self._finalize_mesh(components)
        barrel = next(c for c in components if c.name == "gate_hinge_barrel")
        self.expose_revolute_joint_port("hinge_axis", barrel, "axis")

def resolve_part_parameters(*, design):                  # dependencies r
    gate_width = design["gate_opening_width"] - design["gate_clearance"]
    gate_height = 0.86 * design["fence_height"]
    hinge_height = 0.14 * design["fence_height"]
    return {"fence_body": {...}, "gate_leaf": {"gate_width": gate_width, ...}}
```

```
def build_assembly_plan(*, parts, configuration):      # relations
    hinge = MateRevoluteJoints(
        moving="gate_leaf", moving_port="hinge_axis",
        fixed="fence_body", fixed_port="hinge_axis",
        angle_degrees=configuration["gate_open_angle"])
    return AssemblyPlan(root="fence_body", relations=(hinge,),
                        checks=(CheckRevoluteJoint(...),))
```

B THE VARIATION-FIRST AGENT

This section gives the agent's stages (Algorithm 1), what it observes and its budget.

Contexts. Variation planning, construction and visual critique run in separate contexts of the same LLM and exchange only the current plan, program and observations.

Declarations. A declaration names a control, its kind (design, count or configuration) and its domain, and lists three kinds of effect: targets, which must change; propagation, the dimensions and placements that must follow; and preservation, the attachments, counts and shapes that must stay fixed. Each effect is stated as a measurement that execution returns: the extent of a part along an axis, the number of instances of a part, the distance between two interfaces, the rotation of one part relative to another, or the local shape change of a part.

Observations. Execution returns renders of the whole object and of every isolated part, the report of every relation and check, and the requested measurements. During construction the agent can also read part guides, which describe how common parts are built, and the source of library parts.

Visual critique and variant feedback. The critic receives the input, the program, and the renders of the whole object and of each part, but not the agent's reasoning. It makes at most two critiques, each naming at most two major corrections. Variant feedback executes each declared control in a separate copy of the program at non-default values, including values near both ends of its domain, and changes one control from the default in each test. A control whose declared effects still fail after revision is withdrawn.

Backbone and budget. All contexts use DeepSeek-V4.1-Flash at high reasoning effort and share one budget of 2M weighted tokens per object, in which cached input counts one tenth; Table 3 reports the tokens used.

Instruction excerpts. Each context receives a stage instruction, followed by the runtime contract and, for construction and variant feedback, manuals on part construction, assembly and verification; these appear below as bracketed placeholders, and elisions as [...].

Variation planning. "Analyze the original input before any modeling. Produce a concrete spatial construction plan for an executable RSP template. Distinguish semantic rigid/moving Parts from internal construction helpers. [...] Include supported local details and useful design/state freedoms from the outset. [...] For motion, list every joint the input depicts (hinge, pivot, fold, lid, door, drawer, swivel): each is configuration state driving a joint relation between rigid Parts, with the moving Parts, the axis and its functional range, not the depicted angle." [RUNTIME CONTRACT]

Construction. "Construct the executable RSP template from the original input and concrete plan, or repair the supplied execution/visual defect. Complete all Part classes, resolver and AssemblyPlan; reuse internal functions for repeated construction. Main geometry and attached details derive from the same current parameterized paths/surfaces. [...] Propose tentative controls with declare_controls before the first whole build. [...] A separate context performs that critique." [RUNTIME CONTRACT] [PART AND ASSEMBLY MANUALS]

Visual critique. "Inspect the original input, actual current whole/all-Part images, and current source. Judge silhouette, proportions, spatial relationships, negative spaces, and local visible construction. The plan and valid code are hypotheses, not visual truth. [...] Check every joint the input depicts: it must be configuration state moving rigid Parts through a joint relation. [...] Request at most two highest-impact corrections with visible evidence and affected construction."

Variant feedback. "For each chosen operation, derive the necessary changes and preserved properties from its meaning, including interfaces and attachments. [...] Choose discriminating increases and decreases where the declared domain permits, including count changes. After failed tests, fix the causal code dependency, rebuild and inspect the default again, then re-probe every affected accepted control. Inspect actual target, necessary propagation, preserved properties and connections; a changed parameter or whole bbox alone is insufficient." [RUNTIME CONTRACT] [PART, ASSEMBLY AND VERIFICATION MANUALS]

Algorithm 1 The variation-first agent

Require: Input x (a text description or an image)
1: **Variation planning:** declare controls with domains, targets, propagation and preservation from x
2: **Construction:** write constructors, dependencies r and relations of P that deliver the declared controls; execute P and act on its relation and check reports
3: **for** at most two visual critiques **do**
4: Compare renders of the whole object and of each part of $\mathcal{E}(P; d_0, q_0)$ with x in a separate context
5: **if** no major correction is named **then**
6: **break**
7: **end if**
8: Revise P and execute it
9: **end for**
10: **while** a declared control is untested or failing, and budget remains **do**
11: **Variant feedback:** execute P with one control at a non-default value of its domain
12: Measure its targets, propagation and preservation; revise P where an effect fails
13: **end while**
14: Withdraw controls whose effects do not hold; return P with its confirmed controls

C MATERIALIZATION DETAILS

This section gives the settings of materialization and how the parts and materials of an asset are assigned.

Figure 6 shows the stages for one RSP at two settings.

Appearance image. GPT Image 2 (quality low, 2048×2048) edits a render of G_t in two passes: it first adds appearance, conditioned on an appearance brief derived once from the input and on a structural summary of the instance, and then refines local detail under the same conditions. For image inputs, the input photograph is also given as an appearance reference.

Sparse structure and sampling. With mesh preprocessing W and normalization N_t, the encoder input in Eq. 4 is $\widetilde{G}_t = N_t(W(G_t))$: W builds a watertight mesh at resolution 512, and N_t centers and scales it into the generator's frame. TRELLIS.2 samples with its 512-to-1024 cascade and seed 42, and the export applies N_t^{-1} to recover the input frame and scale.

Part labels and materials. Every face of the asset takes the label of the nearest part of the instance, the labels are smoothed over adjacent faces, and the asset is exported with one node per part. Figure 7 shows assets colored by these labels. The generator predicts albedo, metallic and roughness for every voxel from the conditioning image alone, so shading in the image can make a fabric part metallic or shift its color. One call to the agent's LLM per object receives the appearance brief, the conditioning image and the part names of the program, and assigns every part one of 14 material classes (for example fabric, lacquered wood, glossy plastic, painted metal or polished metal) and a color. Every voxel takes the part whose surface is nearest to its center. The class sets the metallic value (0 or 1; painted metal keeps the predicted value up to 0.5) and clamps roughness to the class's range, and when the median chroma of a part's voxels (CIELAB a^*, b^*) lies more than 15 units from the part's color, the chroma is shifted to it, keeping the predicted lightness and texture detail.

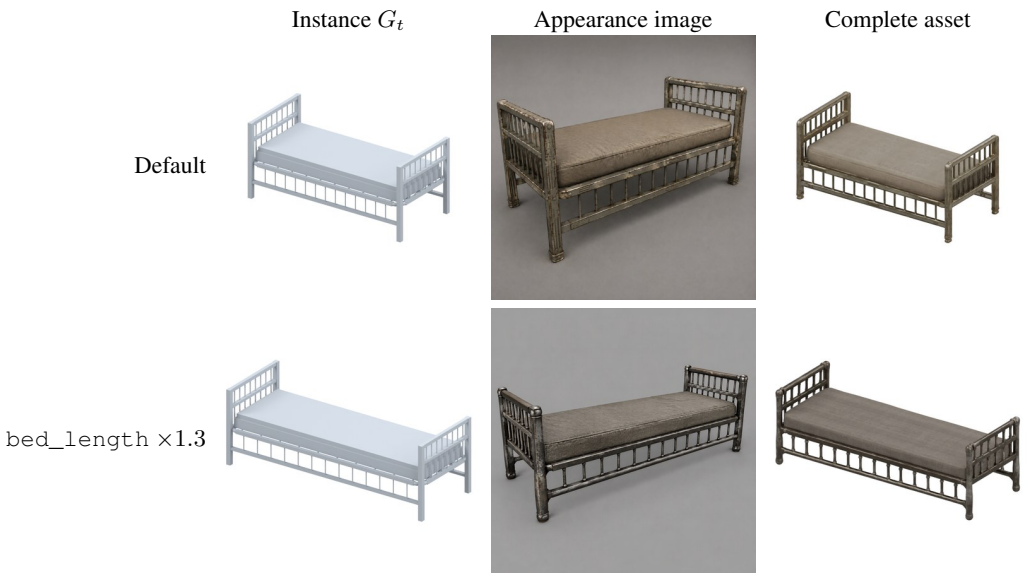

Figure 6: Materialization of one RSP at its default and at a held-out setting of its length control. The generator samples on the occupied voxels of the executed instance (left), conditioned on its appearance image (middle), so the asset (right) keeps the instance's bars, rails and length.

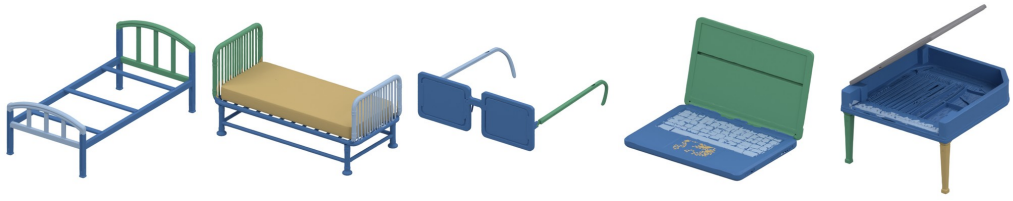

Figure 7: Materialized assets colored by part label, one color per part of the program: two beds, glasses, a laptop and a grand piano. Each asset is exported with one node per part.

D Additional Qualitative Comparisons

Figures 8–11 extend Figure 4 to 24 text and 24 image inputs and to every method of Table 1. The objects were chosen from the inputs alone: one prompt per text category and the first 24 image inputs in dataset order, leaving out the objects and categories of the main-text figures and, for text, pens, knives and swords, which are unreadable at this size.

E Additional Controls and Edits

This section extends Figure 5 to further objects of Table 2. Figure 12 sets one count or angle goal to a held-out value on the program of every method, by re-execution alone. Figure 13 shows three more edits: the agent adds armrests to a chair and, in two objects, replaces the construction of one part; the existing relations place the new parts, and an original control still drives each edited program.

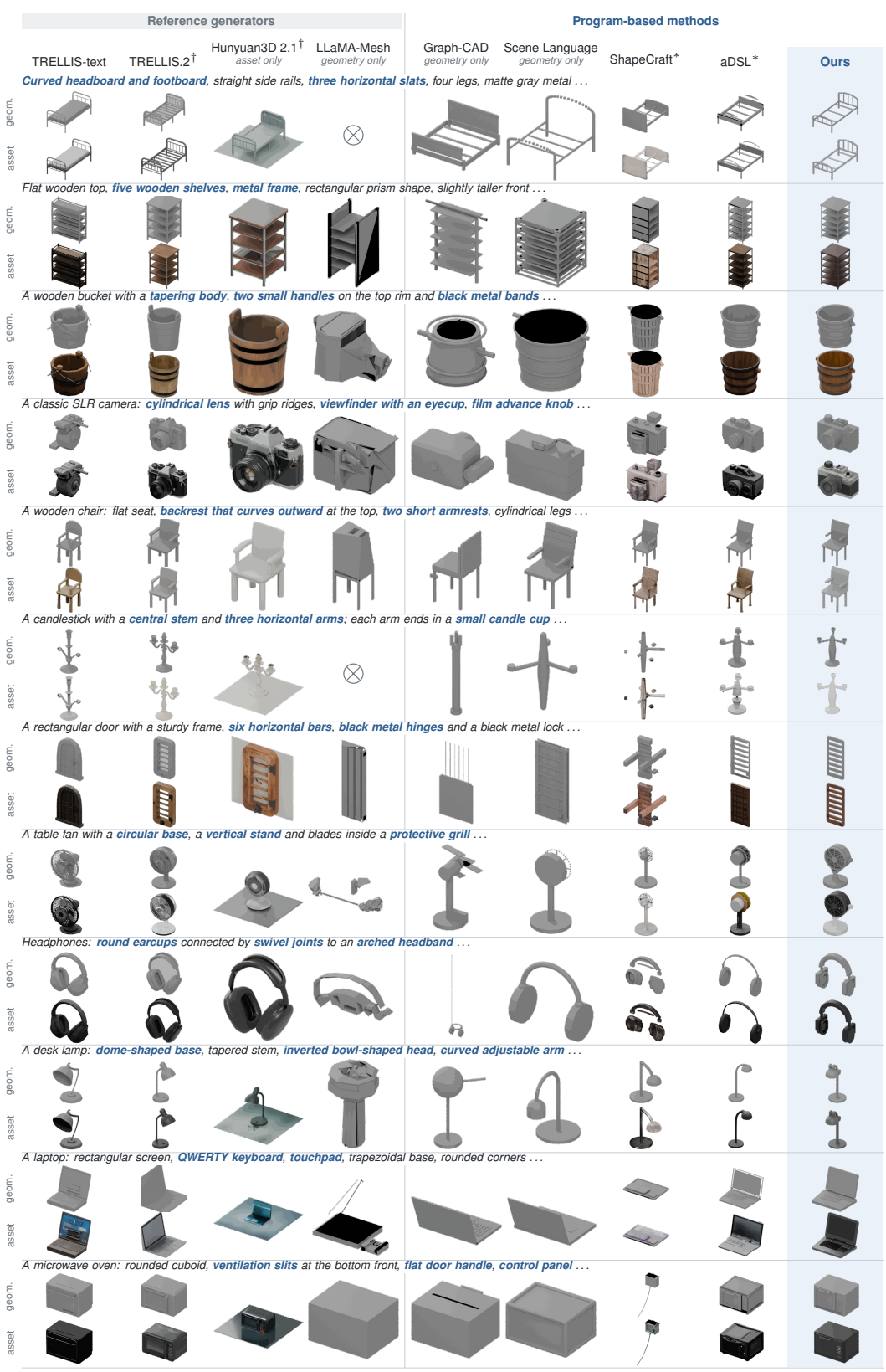

Figure 8: Additional text-to-3D comparisons (1/2), read as Figure 4(a). A crossed circle marks a run without valid output. †From a Qwen-Image image of the description. *As in Figure 4.

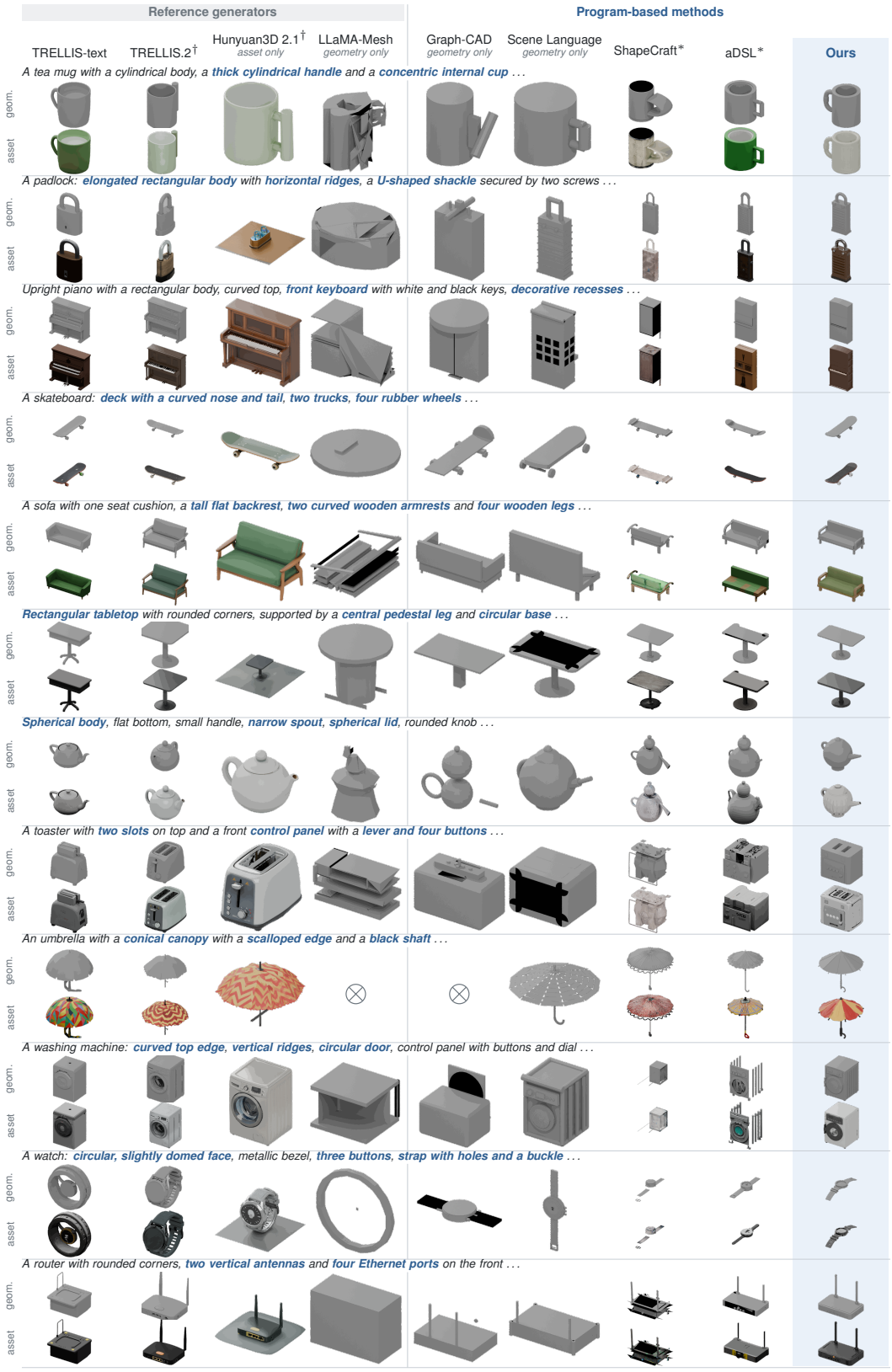

Figure 9: Additional text-to-3D comparisons (2/2), continued from Figure 8.



Figure 10: Additional image-to-3D comparisons (1/2), read as Figure 4(b). A crossed circle marks a run without valid output. *As in Figure 4.

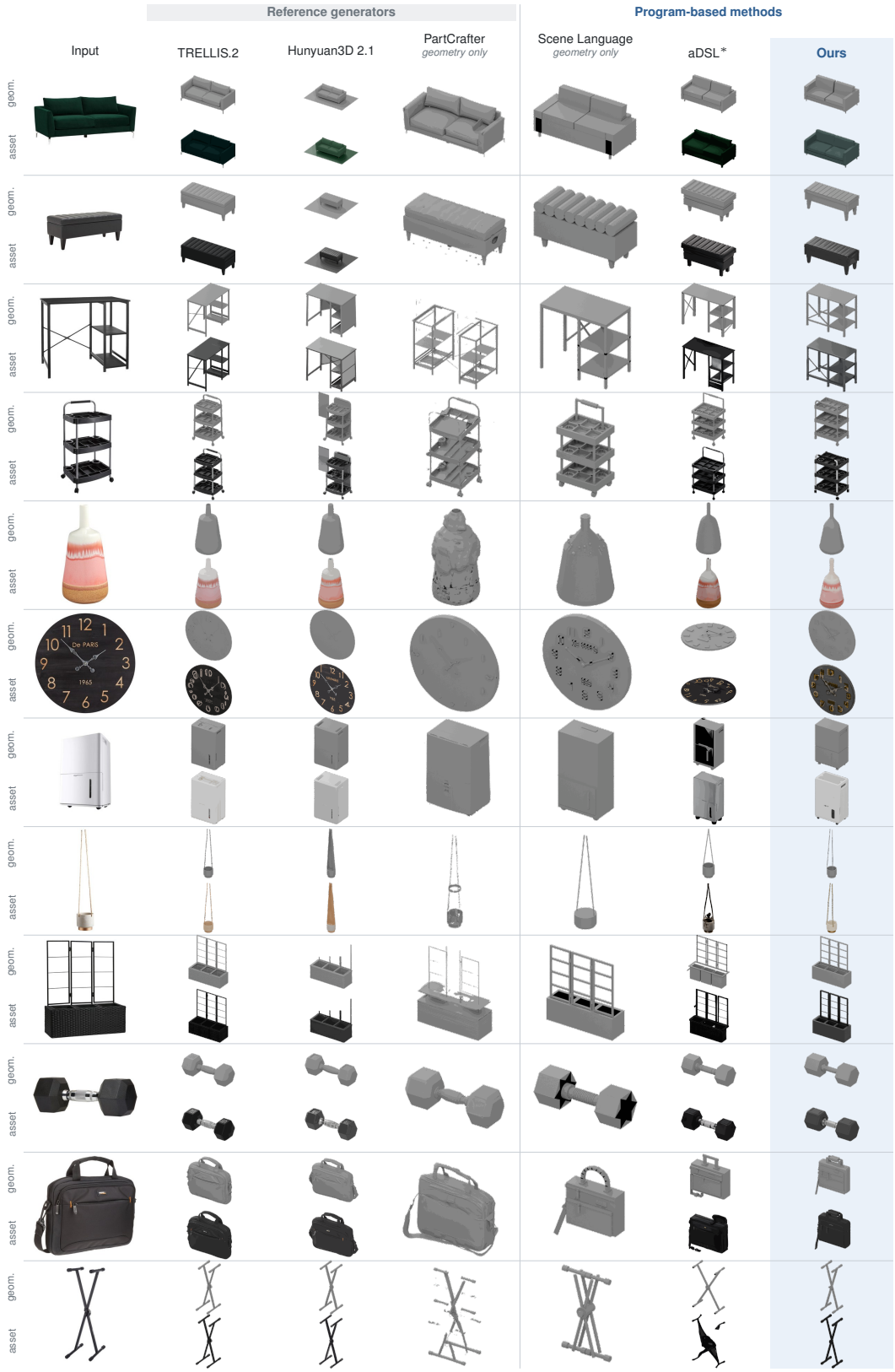

Figure 11: Additional image-to-3D comparisons (2/2), continued from Figure 10.



Figure 12: Held-out control examples. Each row is one count or angle goal of Table 2 at a held-out value, executed on each method's program; left of each arrow the default, right the setting. Program geometry of every method is rendered alike.

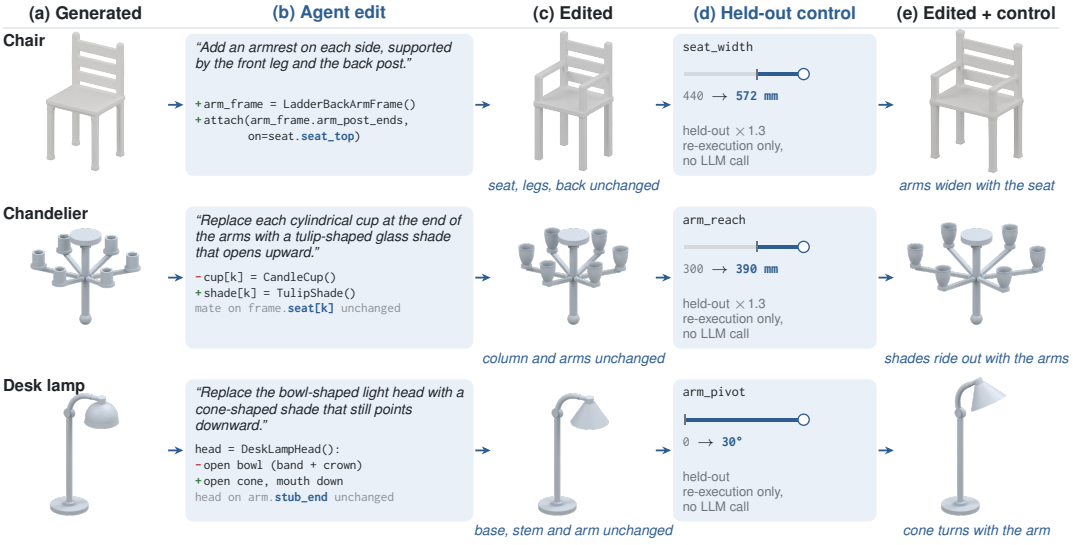

Figure 13: More edits, read as Figure 5: an added part (chair) and two replaced constructions. The existing relations place the new parts, and an original control, set to a held-out value by re-execution alone, still drives each edited program.