ARTICULATECODE: FUNCTIONAL CONTRACTS AND PHYSICAL BAKING FOR ARTICULATED 3D ASSET GENERATION

Anonymous authors
Paper under double-blind review

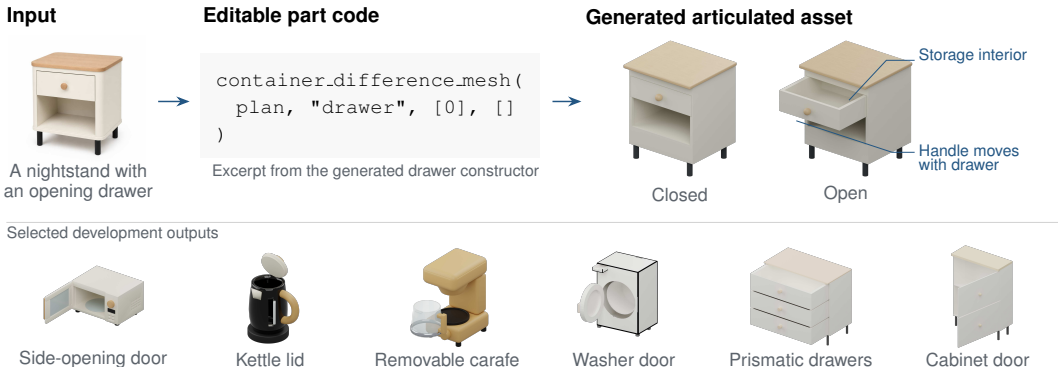


Figure 1: **ArticulateCode generates editable articulated assets through executable part programs.** Top: a reference nightstand, an excerpt of the generated program that builds its drawer, and the closed and open outputs. Bottom: generated assets with varied mechanisms, including a washer with its door open and detergent drawer extended. Views are scaled independently for visibility.

ABSTRACT

Training embodied agents at scale calls for varied furniture and appliances that can be opened, grasped, and used for storage in simulation. Generating their appearance and joint motion is only part of the task: a drawer pull must belong to the drawer, and its collision shapes must leave room for an object inside. We present ARTICULATECODE, which turns a reference image and an interaction request into executable programs for named parts and a physically compiled articulated asset. A functional contract specifies moving parts, joints, handle ownership, and usable interiors. It guides part construction and remains attached to the rigid bodies and collision shapes produced by physical baking. When collision shapes are rebuilt on identical generated meshes, component-aware shapes clear every sampled joint sweep and all loaded-motion cycles, whereas per-body convex decomposition introduces 3.08–18.13 mm of interference. A geometry-level audit localizes three misplaced repeated parts in two generated assets, and a single part-level code revision repairs each. ARTICULATECODEBENCH, our companion evaluation protocol, compares final assets from direct generators, feedback agents, and full systems under shared interaction requests.

1 INTRODUCTION

An agent asked to put a cup away needs more than a cabinet-shaped object. It must locate a handle, open the door or drawer, and place the cup in a space that the simulator also recognizes as empty. Large collections of such assets would make household simulation more varied and useful (Xiang et al., 2020; Deitke et al., 2022; Wang et al., 2024). Building each asset by hand, however, means designing its visible parts, their motion, and the physical shapes used for contact.

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.

Program-based generation makes the visible construction editable: a model can write code for individual panels, drawers, handles, and joints (Wang et al., 2026b; Shui et al., 2026; Zhou et al., 2026). The hard part is keeping these pieces consistent as the object moves into simulation. A handle modeled in the right place may be attached to the stationary cabinet. A visually hollow drawer may receive a solid collision hull. Both errors appear only when someone tries to use the asset. The relevant unit is therefore the *interaction*: which part moves, what moves with it, and which space remains available throughout that motion.

We develop ARTICULATECODE around this unit. Given a reference image, dimensions, and an interaction request, it first constructs a functional contract describing named parts, joints, handle ownership, and protected interiors. Executable programs then generate the parts under stable names. After evaluating the resulting geometry, the system binds parts to rigid bodies and builds body-specific physical properties and collision shapes. The same contract guides checks on the exported asset: handles must follow their movers, moving parts must clear their surroundings, and storage interiors must remain accessible. Measured geometric errors can be returned to the program for the affected part without regenerating the entire object.

The experiments isolate the contribution of these choices. Contract checks identify missing and ambiguous relations that a mover-name check misses. With generated geometry and joints held fixed, component-aware collision construction preserves sampled motion and loaded operation, while per-body convex decomposition introduces interference. An assembly study shows that measured part bounds pinpoint and repair misplaced repeated parts. We also introduce ARTICULATECODEBENCH, a shared-input protocol comparing final assets from direct models, feedback agents, and generation systems. Across its comparisons, delivering an importable package proves far easier than clean motion: full-sweep success stays at or below two of seven interactions for every system.

2 RELATED WORK

Articulated-object generation. SINGAPO, S2O, and ArtLLM share the goal of assets whose parts move, but they draw the articulation from different sources: image evidence (Liu et al., 2025), an existing static mesh (Iliash et al., 2024), and an LLM-issued part–joint blueprint (Wang et al., 2026a). Their evaluations stop on the visible side of the asset—part segmentation, joint parameters, and renders. We start from a reference image plus an explicit interaction request and carry the request through compilation: handle ownership and usable interiors become properties of the exported rigid bodies and collision shapes, which is the stage these works leave unmeasured.

Programs as an asset representation. Executable code makes geometry editable at the part level, and program representations increasingly drive 3D generation: ShapeAssembly composes shapes from part programs (Jones et al., 2020); 3D-GPT and SceneCraft synthesize programs from language (Sun et al., 2025; Hu et al., 2024); SceneCode, ArtiCAD, and Articraft generate per-part construction programs for articulated objects, adding assembly connectors and motion tests (Wang et al., 2026b; Shui et al., 2026; Zhou et al., 2026). A program that renders correctly, however, can still compile into an asset that fails to function. We keep the program as the editable source and make it answerable to the interaction: every part enters physics under its contract identity, so a failed check names the program to revise.

Physical compilation and evaluation. Contact simulation requires convex proxies, and convex decomposition such as CoACD (Wei et al., 2022) is the standard route to them; for an articulated container, that decomposition also decides whether a drawer still opens and whether storage space survives. PhyScene checks collision and reachability at scene scale (Yang et al., 2024), whereas the corresponding questions here live inside a single object and follow its joints. Program- and CAD-generation benchmarks score execution and geometric quality (Gao et al., 2026; Yang et al., 2026; Dong et al., 2026); ARTICULATECODEBENCH extends final-asset evaluation to part correspondence, motion, and interior access, which is what the physical compiler's claims require.

3 ARTICULATECODE

ArticulateCode takes a reference image, metric dimensions, and a requested interaction, denoted together by x, and returns a physically compiled articulated asset. The pipeline is $x \mapsto (\pi, \rho) \mapsto (C, \{G_p\}_{p \in P}) \mapsto \mathcal{A}$: a plan π and ownership map ρ turn the request into named obligations, model-generated source C produces named geometry G_p, and physical baking produces the asset $\mathcal{A}$, which a final audit checks against the same plan (Figure 2). Algorithm 1 shows where failed checks return to source code or physical compilation.

One abstraction drives all four stages. Section 3.1 states the obligations; Sections 3.2 and 3.3 preserve their identities through geometry generation and physical compilation; Section 3.4 re-checks them on the compiled asset; Section 3.5 converts failures into local repairs.

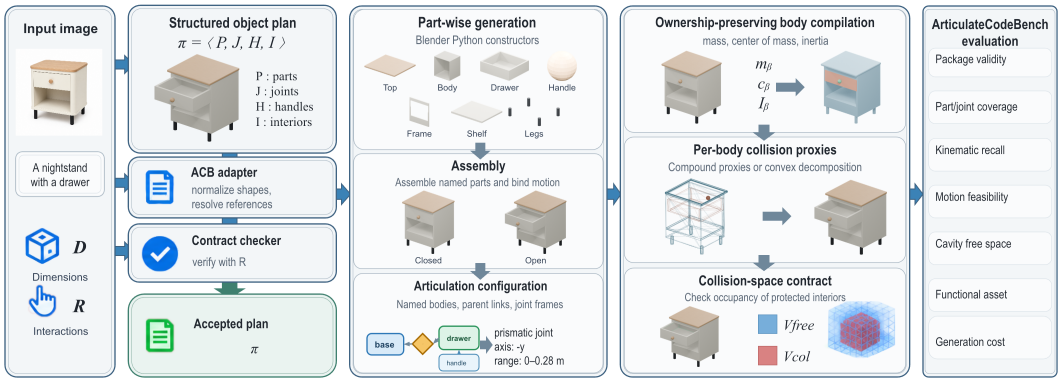


Figure 2: **From interaction request to physical asset.** A functional contract specifies parts, joints, handle ownership, and interior space. Executable programs build named geometry; physical baking groups parts by rigid owner and constructs inertial and collision records. The exported asset is checked for motion and interior access. Colors distinguish bodies and protected space; parts are enlarged for visibility.

3.1 FROM REQUEST TO OBLIGATIONS

Functional contract. The model translates x into an object plan $\pi = \langle P, J, H, I \rangle$ in a shared object frame (meters, with width, depth, and height along x, y, z). Each named part in P has dimensions, a pose, and a geometric role. A joint in J identifies its moving child, parent, type, axis, frame, and limits. The handle map H contains pairs (h, m) linking a graspable part h to its requested mover m, and I records each storage region, its owner, and its opening direction. The plan is deliberately small: it states what must move and what must remain usable, and leaves how the parts look to the next stage.

Ownership. The map $\rho : P \to \mathcal{B}$ assigns visible parts to rigid bodies, and the contract requires, for every requested handle–mover pair,

$$(h, m) \in H \quad \Longrightarrow \quad \rho(h) = \rho(m). \tag{1}$$

A drawer pull and its drawer therefore share one owner, while the cabinet frame has another. A storage region stays in its owner's local frame even when a different part controls its opening: a pot retains its interior when its lid moves. The contract resolves all references before geometry generation, so an ambiguous assignment returns to planning instead of surfacing later as a silent attachment error.

3.2 GEOMETRY FROM PROGRAMS, UNDER NAMES

Part-wise program synthesis. For each part p, its record π_p specifies the intended shape, dimensions, pose, material, and role in the interaction. In the native profile, the model receives π_p together with the reference input x and writes an executable Blender program c_p; evaluating the program yields the named mesh $G_p = \mathrm{Eval}(c_p)$. The model call produces source, and only program execution

produces geometry, so every later stage operates on evaluated meshes. Movable parts stay separate objects, and substructures that move rigidly with a part are constructed under that part's identity, which keeps later correction local to the affected program.

Program representation. The typed profile restricts generation to solid, shell, and cavity operations, allowing one program to define several named parts. It retains the positive geometric components used to form each part—the input that Section 3.3 needs to build collision pieces under the correct rigid owner.

Source-geometry validation. Both profiles validate geometry rather than source: the typed profile checks each returned solid for closedness, winding, volume, and bounds, and the native profile executes its scripts and measures the resulting meshes.

Geometry feedback. An envelope check verifies that part p occupies the region the plan assigns it: the plan defines a world-space envelope E_p, including declared mirror and array transforms, and $\widehat{E}_p$ is measured from the vertices of G_p. A center or extent beyond tolerance returns only the source of p for one bounded revision, with π and all other parts fixed. This is what catches a seed translation applied twice in a repeated array (Appendix E). The checked geometry and its owner, $(G_p, \rho(p))$, feed the next stage.

3.3 Physical Baking: Names to Bodies

Body assembly. Physical baking turns named geometry into actuated, contact-bearing bodies. It groups parts with the same owner into one rigid body, estimates mass and inertia from geometry and material density priors, and assigns contact parameters. Equation 1 places a drawer and its pull in the same body frame, so their collision proxies follow the same joint motion after export.

Collision proxies. How each body's collision shapes are built decides whether the compiled asset still works: a single hull around a drawer erases its interior. The typed profile therefore prefers an explicit compound or supported open shell. Let $\mathrm{Comp}^{+}(G_p)$ be the positive source components retained with evaluated part G_p; the proxies of body b are

$$\mathcal{C}_b = \left\{ \mathrm{Hull}(g) : p \in P,\ \rho(p) = b,\ g \in \mathrm{Comp}^{+}(G_p) \right\}. \tag{2}$$

The sides and floor of a drawer become separate convex proxies under one rigid owner, and the source-level gap between them survives compilation. When explicit components are unavailable, CoACD (Wei et al., 2022) decomposes each body under a hull budget and a geometric-fit check.

The compiler exports $\mathcal{A}$ as a URDF package or native MuJoCo profile with meshes, joints, inertial records, and contact parameters (Appendix A). The exported proxies, rather than the visible mesh, determine contact in the simulator.

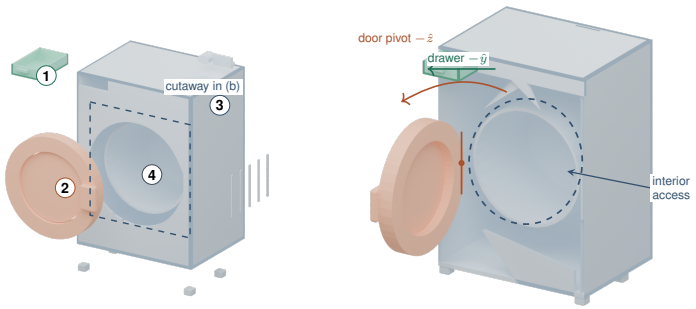
Figure 3 shows why ownership and openings must travel together: the washer door and its handle form one moving body, the detergent drawer forms another, and the fixed enclosure surrounds an interior that must stay reachable.

3.4 Auditing the Compiled Asset

Interior occupancy and access. The audit asks whether the compiled asset still honors the contract. For each declared region $r \in I$, let Ω_r be its owner-local region, V_{vis} the owner's visual material, and V^r_{col} the exported collision shapes. The space that appears empty is $F_r = \Omega_r \setminus V_{\mathrm{vis}}$, and its occupancy and access from the specified opening o_r are

$$B_r = \frac{|F_r \cap V^r_{\mathrm{col}}|}{|F_r|}, \qquad A_r = \frac{|\,\mathrm{Reach}_{o_r}(F_r \setminus V^r_{\mathrm{col}})|}{|F_r|}. \tag{3}$$

The pair matters because the two failures it separates call for different repairs: B_r detects a filled cavity, while A_r detects a thin barrier across its entrance. A declared region passes when $B_r \le 0.15$, $A_r \ge 0.85$, and visual occupancy is at most 0.15.



(a) Rest pose, separated rigid-body owners (b) Open-joint pose and front cutaway

1 Detergent drawer 2 Door and handle 3 Fixed enclosure 4 Interior opening

Figure 3: **Part ownership and accessible interior in a generated washer.** The blue outline in (a) identifies the panel cut away in (b). At the recorded joint endpoints, the door rotates 1.57 rad and the detergent drawer extends 0.12 m; the ellipse marks the main-body opening. Appendix H analyzes its interior.

Handle co-motion and joint clearance. The motion check verifies the exported articulation against the ownership and joint specification. Equation 1 identifies which handle should follow each mover. With $\mathcal{Q}_j$ the sampled poses of its joint, q_0 the rest pose, $T_m(q)$ the mover's world transform, and $v_h(q; v)$ the world position of handle vertex v at pose q, the maximum drift in the mover's local frame is

$$D_{h,m} = \max_{q \in \mathcal{Q}_j,\, v \in G_h} \left\| T_m(q)^{-1} v_h(q; v) - T_m(q_0)^{-1} v_h(q_0; v) \right\|_2 . \tag{4}$$

Drift above 10^{-5} m breaks the requested ownership relation. Each joint is also swept with the compiled collision shapes engaged. Together these checks produce a diagnostic report η naming the relation or geometry to revisit.

3.5 Bounded Functional Corrections

Scoped feedback. A failed check names its cause, so repair stays local. An envelope failure returns the offending part source for one bounded LLM revision. An interaction failure in η instead triggers a supported source-level correction: moving a hinged closure's pivot within its budget, cutting a swept opening in the identified fixed panel, or adjusting explicit cavity geometry to satisfy the support check. The opening cut is confined to the panel slab when its depth is known (Appendix H); the support adjustment keeps the exterior and joint travel fixed (Appendix D). After either change, the affected geometry is evaluated, baked, and audited again; checks that still fail remain in η, so the asset ships with its diagnosis. Algorithm 1 summarizes both feedback paths.

Algorithm 1 Contract-guided construction with bounded correction

Require: Input x; per-part code budget b_p; physical-correction budget b_c
1: $(\pi, \rho) \leftarrow \mathrm{Plan}(x)$; $C \leftarrow \mathrm{LLMCode}(x, \pi)$; $\{G_p\} \leftarrow \mathrm{Eval}(C)$
2: Revise each envelope-failing part within b_p attempts; re-evaluate its G_p
3: **if** required geometry remains invalid, **return** unresolved report
4: $\mathcal{A} \leftarrow \mathrm{Bake}(\{G_p\}, \rho, J, I)$; $\eta \leftarrow \mathrm{Audit}(\mathcal{A}, \pi)$
5: Apply supported source corrections, re-evaluate, bake, and audit at most b_c times
6: **return** $(\mathcal{A}, \eta)$, retaining any unresolved checks

4 Experiments

4.1 Evaluation Setting

ArticulateCodeBench evaluates articulated household objects from a reference image, metric dimensions, and a requested interaction, and scores the delivered asset rather than a ground-truth mesh. The protocol contains 31 requests over 21 objects with 34 joint obligations; its Hard-12 subset

supports the direct-model and standardized-agent comparison, and the Hard-6 subset supports the complete-system comparison with seven requested interactions and four storage cases. All methods in a comparison receive the same reference and request.

The experiments ask three successive questions. Section 4.2 tests whether the contract and collision-space audit identify known relation and obstruction errors. Section 4.3 then holds generated geometry fixed while changing collision construction, so differences in motion and payload behavior can be attributed to that physical representation. Section 4.4 compares final outputs from Direct Code, SceneCode, and ArticulateCode with their native prompts and call budgets. The two studies are reported separately.

Final-asset measures. For request i, package validity $P_i \in \{0, 1\}$ indicates whether its export contains importable geometry and the required articulation records; package rate is $\sum_i P_i/N$ over N issued requests. Joint-child coverage $R_J = N_{\text{matched}}/N_{\text{requested}}$ counts requested movers with a matching child mesh. Let $c_t(q)$ indicate collision-free motion at sampled pose q of trajectory t. Full-sweep success is $S_t = \min_{q \in Q_t} c_t(q)$, whereas clear-state fraction is $C_t = |Q_t|^{-1} \sum_{q \in Q_t} c_t(q)$. Maximum penetration is $M_t = \max_{q \in Q_t} \delta_t(q)$, where $\delta_t(q)$ is the largest overlap depth. Motion rates average S_t or C_t over requested trajectories. The end-to-end functional indicator F_i is one only when all evaluator-required tests for request i pass; both P_i and F_i average over all issued requests, with missing outputs scoring zero.

Space and physical-use measures. For a declared interior, occupancy B_r and opening-side access A_r are calculated by Eq. 3. Payload-entry rate $D = n_{\text{settled}}/n_{\text{drops}}$ requires the payload to enter, settle, and be supported on the original floor. Loaded-motion rate $L = n_{\text{successful}}/n_{\text{placements}}$ tests actuation with the payload already inside. Thus D concerns the approach path and support surface, while L concerns retaining a payload during motion. The system evaluator additionally reports the grid-sampled free-interior fraction U_r, which complements the opening-side access A_r (Appendix A).

4.2 Contract, Ownership, and Collision-Space Validation

Part relations. Across six object types, we vary handle and mover relations to obtain 30 faulty and 18 valid or uniquely recoverable examples. Every example names a mover, so a mover-only check accepts all 48. The contract identifies 18 of the 30 relation defects while retaining all 18 valid examples (Table 1).

Table 1: **Relation and interior-space checks.** Panel-preserving proxies pass all 18 payload drops; filled or obstructed interiors fail. Relation tests include 30 defects and 18 valid examples.

A. Relation screening

Check	Rejected	Retained
Target only	0/30	18/18
Contract adapter	18/30	18/18

B. Protected collision space

Proxy	Occupancy B_r	Access A_r	Accepted	Drops passed
Preserved panels	0.000	1.000	6/6	18/18
Solid envelope	1.000	0.000	0/6	0/18
Bottom 25% filler	.237–.263	.737–.763	0/6	0/18
Top 10% barrier	.079–.091	0.000	0/6	0/18

Ownership after export. Across seven generated asset packages, ten required handle relations undergo complementary ownership checks. Five relations resolve to their requested movers through fixed-joint chains; the other five remain fixed in the mover's local frame after re-import across 61 poses, within 10^{-5} m. A wrong-owner control still moves visibly while drifting 0.20 m relative to the intended mover, so the audit measures attachment in the mover's frame rather than world displacement (Appendix B).

Interior access. We vary collision shapes while holding each container's visual geometry fixed, then drop a 0.1 kg cube at three positions. Occupancy detects solid and bottom-filled interiors. Opening-side reachability additionally detects a thin barrier that blocks entry despite occupying little volume. The combined test accepts the six panel-preserving containers, which pass all 18 drops (Table 1). Appendix Figure 5 sketches how the four proxy variants differ.

4.3 Functional Outcomes on Generated Assets

We test a nightstand, rice cooker, and kitchen cabinet with generated geometry held fixed across collision conditions (Table 2). Motion and physical outcomes are measured from their compiled packages.

Table 2: **Selected generated-asset validation.** Sweeps include each joint separately and one synchronized trajectory, at 241 poses each. Interior counts concern all-body closed-pose checks. Loaded trials use a 50 g payload at nine grid positions with bias-compensated local actuation; the kitchen tests only its drawer floor.

Asset	Triangles	Joints	Clear sweeps	Clear interiors	Loaded trials
Nightstand	3,132	1	2/2	1/1	9/9
Rice cooker	3,480	1	2/2	1/1	9/9
Kitchen cabinet	2,644	3	4/4	3/3	9/9

Joint placement. The rice cooker's initial hinge penetrates the enclosure by 9.71 mm in a 241-pose sweep. Moving its origin 10 mm to the measured closure boundary removes the penetration without changing vertices, joint axis, or range.

Collision representation. With meshes, body assignments, joints, and inertial properties fixed, we replace component-aware shapes with per-body CoACD under the same 64-hull cap. The replacements introduce 3.08–18.13 mm of maximum penetration at the 0.05 threshold across the three objects; component-aware shapes clear every sampled sweep (Table 3). A tighter 0.01 threshold completes only for the kitchen within the set budget and still gives 14.84 mm of penetration (Appendix D).

Component-aware shapes also pass 6/9, 9/9, and 6/9 payload-entry tests for the nightstand, rice cooker, and kitchen, compared with 0/9, 1/9, and 0/9 under CoACD. The rice-cooker difference depends on the 2 mm floor tolerance: at 5 mm both conditions pass 9/9. These tests measure entry and support, distinct from the loaded-motion trials that begin with the payload inside (Appendix D).

Multiple joints. The kitchen program constructs one drawer and two doors from nine valid solids. Individual and synchronized sweeps, three interior checks, and re-imported handle co-motion pass. Loaded placements target the drawer floor; Appendix D gives the asset-level test conditions.

From image to functional asset. In the raw-input setting, Qwen3.8-Max predicts the plan and part roles from the image. Two nightstand construction conditions pass drawer sweeps, handle co-motion, and nine loaded cycles after one handle revision; constraint projection also recovers the requested dimensions (Appendix D). On a three-drawer cabinet, a support-constrained cavity adjustment changes the estimated margin from -22.9 to $+10.0$ mm and raises loaded-cycle success from 0/27 to 27/27 at the same starts, with exterior dimensions and joint travel fixed (Appendix D).

Part-level code repair. For a generated office chair and gas cooktop, the evaluated-geometry guard identifies three misplaced arrays among 15 part programs. One revision per affected program brings all three within their expected envelopes. Re-import recovers all 15 meshes and both required joints; the plans and the other 12 programs remain unchanged (Appendix E).

Table 3: **Collision construction on identical generated meshes.** Component-aware proxies and per-body CoACD (threshold 0.05) share body assignments, joints, inertial properties, and a 64-hull cap per body. Component-aware maximum penetration is zero in all sampled sweeps. Drops count payloads entering and settling on the original floor under a 2 mm tolerance. Full settings and sensitivity analysis appear in Appendix D.

Asset	Comp. sweeps	CoACD sweeps	Max. pen. (mm)	Comp. drops	CoACD drops
Nightstand	2/2	0/2	12.37	6/9	0/9
Rice cooker	2/2	0/2	3.08	9/9	1/9
Kitchen cabinet	4/4	0/4	18.13	6/9	0/9

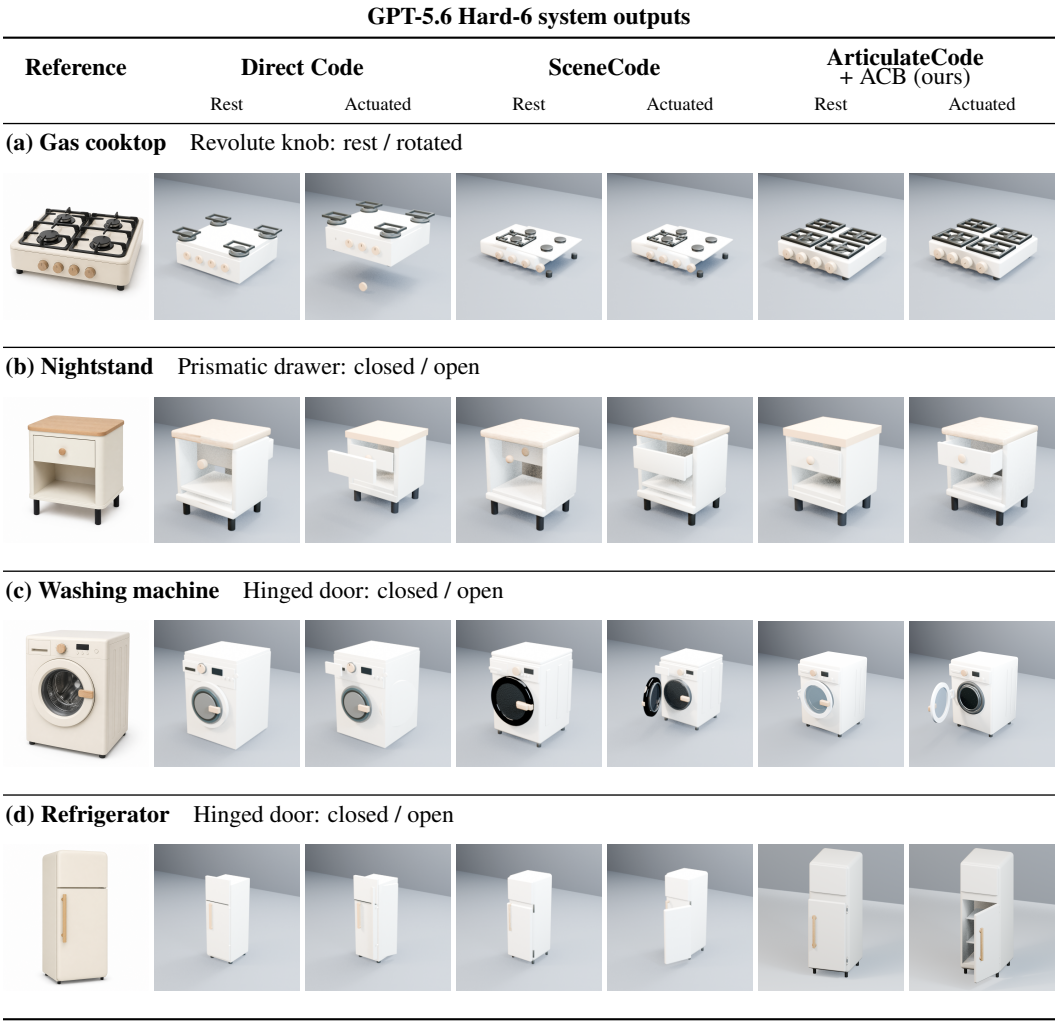


Figure 4: **Final assets from the complete-system comparison.** Four shared Hard-6 requests are shown for Direct Code, SceneCode, and the native ArticulateCode configuration, at rest and at 82% of each submitted joint range. Each row uses the same reference input.

4.4 Complete-System Comparison

Table 4 compares the final Hard-6 packages produced by Direct Code, SceneCode, and the native ArticulateCode configuration under three model/provider anchors. Each system retains its own prompts and call budget; this comparison measures full systems, whereas Section 4.3 holds geometry fixed to isolate physical compilation. The full evaluator and cost breakdown appear in Appendix F (Table 11).

Table 4: **Hard-6 complete-system outcomes under three model/provider anchors.** Package uses six requests; joint-child and full-sweep coverage use seven requested interactions. Missing packages score zero. The detailed cavity, functional, and cost measurements are in Appendix Table 11.

Anchor	System	Package	Joint child	Full sweep
Qwen3-VL-235B HF–Novita	Direct Code	4/6	5/7	2/7
	SceneCode	4/6	5/7	1/7
	ArticulateCode	1/6	1/7	0/7
Qwen3.8-Max Aliyun	Direct Code	6/6	7/7	2/7
	SceneCode	3/6	4/7	1/7
	ArticulateCode	0/6	0/7	0/7
GPT-5.6 Luna OpenRouter	Direct Code	6/6	7/7	2/7
	SceneCode	5/6	6/7	1/7
	ArticulateCode	5/6	6/7	0/7

Motion is the shared bottleneck. Delivering an importable package proves far easier than moving cleanly: under Qwen3.8-Max and GPT-5.6 Luna, Direct Code resolves all seven requested movers, yet only two clear their full sweeps, and full-sweep success stays at or below 2/7 for every system in Table 4. The ArticulateCode nightstand keeps 88% of its interior free but clears only 10 of 21 drawer poses, which is why interior volume and motion are scored separately.

Interface sensitivity. The contract interface is model-sensitive: in the earlier configuration it yields five packages under GPT-5.6 Luna and none under Qwen3.8-Max, while the part-level repair and typed construction studies use a revised interface and are reported separately. On four Hard-12 routes, feedback lifts package yield from 36/48 to 43/48 while geometry-valid yield stays at 1/48: retries repair packaging, not geometry (Appendix F). Figure 4 compares delivered rest and actuated states for shared references.

5 Conclusion

ArticulateCode carries functional requirements from part programs into rigid-body and collision representations. The contract names what must move together and which interior space must stay usable; physical baking preserves those identities, and the export audit checks them on the compiled asset. This coupling yields collision shapes that clear joint sweeps and interiors, and it localizes repair: a failed check names the part program to revise.

AI Use Statement

The authors used AI-based writing assistance for language editing and manuscript formatting. All research content—the audit design, detector and review protocol, contract construction, experiments, statistical analyses, and all reported numbers—was produced and verified by the authors, who take full responsibility for the content of this submission.

Ethics Statement

This work evaluates software-generated 3D assets and does not involve human participants, human-subject data, or animal experiments.

Reproducibility Statement

The benchmark requests, evaluation metrics, model and provider settings, and physical validation procedures are described in Section 4.1 and Appendix A. Metric denominators and the treatment of missing outputs are specified with the corresponding measures and results.

REFERENCES

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-scale embodied ai using procedural generation. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Xiaoyu Dong, Zhi Li, and Xiao-Ming Wu. MUSE: Benchmarking manufacturable, functional, and assemblable text-to-CAD generation. *arXiv preprint arXiv:2605.28579*, 2026. URL https://arxiv.org/abs/2605.28579v2.

Yipeng Gao, Lei Shu, Genzhi Ye, Xi Xiong, Ameesh Makadia, Meiqi Guo, Laurent Itti, and Jindong Chen. 3DCodeBench: Benchmarking agentic procedural 3d modeling via code. *arXiv preprint arXiv:2606.01057*, 2026.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. SceneCraft: An llm agent for synthesizing 3d scenes as blender code. In *International Conference on Machine Learning*, 2024.

Denys Iliash, Hanxiao Jiang, Yiming Zhang, Manolis Savva, and Angel X. Chang. S2O: Static to openable enhancement for articulated 3d objects. *arXiv preprint arXiv:2409.18896*, 2024.

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), 2020.

Jiayi Liu, Denys Iliash, Angel X. Chang, Manolis Savva, and Ali Mahdavi-Amiri. SINGAPO: Single image controlled generation of articulated parts in objects. In *International Conference on Learning Representations*, 2025.

Yuan Shui, Yandong Guan, Zhanwei Zhang, Juncheng Hu, Jing Zhang, Dong Xu, and Qian Yu. ArtiCAD: Articulated CAD assembly design via multi-agent code generation. *arXiv preprint arXiv:2604.10992*, 2026. URL https://arxiv.org/abs/2604.10992v2.

Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3D-GPT: Procedural 3d modeling with large language models. In *International Conference on 3D Vision*, pp. 1253–1263, 2025.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. GenSim: Generating robotic simulation tasks via large language models. In *International Conference on Learning Representations*, 2024.

Penghao Wang, Siyuan Xie, Hongyu Yan, Xianghui Yang, Jingwei Huang, Chunchao Guo, and Jiayuan Gu. ArtLLM: Generating articulated assets via 3D LLM. *arXiv preprint arXiv:2603.01142*, 2026a. URL https://arxiv.org/abs/2603.01142v2.

Puyi Wang, Yuhao Wang, Linjie Li, Zhengyuan Yang, Kevin Qinghong Lin, Yangguang Li, and Yu Cheng. SceneCode: Executable world programs for editable indoor scenes with articulated objects. *arXiv preprint arXiv:2605.19587*, 2026b.

Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. Approximate convex decomposition for 3D meshes with collision-aware concavity and tree search. *ACM Transactions on Graphics*, 41(4): 1–18, 2022. URL https://colin97.github.io/CoACD/.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. SAPIEN: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11097–11107, 2020.

Yandan Yang, Baoxiong Jia, Peiyuan Zhi, and Siyuan Huang. PhyScene: Physically interactable 3d scene synthesis for embodied ai. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16262–16272, 2024.

Yikang Yang, Zhanpeng Hu, Youtian Lin, Mengqi Zhou, Jingxi Xu, Feihu Zhang, Jiaheng Liu, and Yao Yao. P3D-Bench: Benchmarking MLLMs for parametric 3D generation and structural reasoning. *arXiv preprint arXiv:2606.11152*, 2026. URL https://arxiv.org/abs/2606.11152v2.

Matt Zhou, Ruining Li, Xiaoyang Lyu, Zhaomou Song, Zhening Huang, Chuanxia Zheng, Christian Rupprecht, Andrea Vedaldi, and Shangzhe Wu. Articraft: An agentic system for scalable articulated 3D asset generation. *arXiv preprint arXiv:2605.15187*, 2026. URL https://arxiv.org/abs/2605.15187v1.