PROCEDURA: AGENTIC 3D MODELING WITH PROCEDURAL CONTROL

Anonymous authors
Paper under double-blind review



Figure 1. Procedura turns a text prompt into an editable procedural assembly, a parametric program whose named parts are joined by typed mates, written with a frozen LLM and no 3D training. Top left: the compiled geometry. Top right: further objects coloured by their named modules, a part decomposition that comes free with the representation. Bottom: further objects with the per-part materials **Procedura** assigns from the reference image.

ABSTRACT

Native 3D generators now recover impressive mesh geometry from a single image. However, a dense mesh stays soft where a machined object should be sharp, it carries no part decomposition, and it exposes no parameter a user could edit. To address this, we explore the paradigm of 3D shape as code, leveraging and scaling the coding ability of an LLM for 3D modeling. We introduce **Procedura**, a novel 3D modeling agent framework that writes an object as a procedural assembly, a parametric program whose named parts are joined by typed, machine-checkable mates. From a text prompt, the agent plans the object as an assembly graph and writes the program part by part, solving each placement from the mated frames rather than guessing it, and admitting a part only once compile, mate, and connectivity checks pass. A decoupled vision critic then refines the assembly one diagnosed fix at a time. Moreover, the same graph carries per-part materials and a simulator-validated articulation. We evaluate on P3D-Bench under its assembly judge, and on MechBench-36, our hard-surface benchmark, under the same blind judging protocol. On both, **Procedura** outperforms state-of-the-art native 3D generators and every prior 3D-code agent on judged quality, produces the sharpest edges of any method we evaluate, and is the only one whose output is an editable, part-structured program.

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.

1 Introduction

Machines, vehicles, and tools are the objects that fill games, simulators, and factory floors, and generating them is a fundamental problem at the intersection of computer vision and graphics. Native 3D generators have advanced rapidly and now recover impressive mesh geometry from a single image (Wu et al., 2024a; Xiang et al., 2025; Team Hunyuan3D, 2025; Jia et al., 2025). However, neither of the two dominant generator families turns this fidelity into a usable production asset. Native generators produce the shape as an SDF or a voxel grid, whereas autoregressive generators write the mesh vertex by vertex and face by face (Nash et al., 2020; Siddiqui et al., 2024; Chen et al., 2025). Both output raw triangles with no structure attached. Surfaces decoded from a grid stay soft and scan-like where a machined object demands sharp edges and exact faces, and meshes written token by token hold only as much detail as the token budget allows. Nothing in the mesh says which triangles belong to which part, so a part decomposition must come from a separate segmentation model applied afterwards (Yang et al., 2024b; Liu et al., 2025; Yang et al., 2026b), and even then the cuts follow surface geometry rather than meaning, so we still cannot reliably tell which piece is, say, the left finger of a robot hand. Editing is also hard, since the mesh exposes no parameter to adjust, which shuts out the non-expert user. Taken together, prior methods fall short on high-precision geometry, meaningful part decomposition, and editability.

Writing the shape as code offers another route, since a program is parametric and part-structured by construction. State-of-the-art LLMs such as GPT and Gemini (OpenAI, 2023; Gemini Team, 2023) now write high-quality code, and this ability already reaches 3D, from Three.js scenes to Blender scripts (Makatura et al., 2023; Hu et al., 2024). Yet a 3D shape that an LLM writes as code still comes out toy-like, extremely low-poly and without fine detail. A single pass must commit every part, dimension, and placement at once, with no view of what its code builds, and even recent agentic systems do not close this gap (Fan et al., 2026; Adam CAD, 2026; Zhou et al., 2026). Can we use the coding ability of an LLM to generate detailed, high-quality 3D shapes that meet the three requirements above?

To answer this question, we design **Procedura** (Fig. 1), an agentic 3D modeling framework that leverages the code generation ability of an LLM and scales it to 3D shapes with high-grade detail and a native, meaningful part decomposition. **Procedura** lets a frozen LLM write the shape in constructive solid geometry (CSG), where every face is an exact plane or cylinder, so edges come out sharp where a generated mesh stays soft. Our key idea is to cast the code as a procedural assembly, a parametric program whose named parts are joined by typed, machine-checkable mates. This structure turns one large blind generation into many small, checkable steps. Every part is a named, editable module that the model writes in a single call, each placement is solved from the mated frames rather than guessed, and compile, mate, and connectivity checks admit every commit, so the build grows into a detailed, many-part object without drifting. A decoupled vision critic then refines the result (Fig. 2).

We evaluate **Procedura** on P3D-Bench (Yang et al., 2026a) under its assembly judge, and on MechBench-36, our hard-surface benchmark, under the same blind protocol, against native generators, prior 3D-code agents, and single-shot LLM baselines. On both benchmarks, **Procedura** outperforms all of them on judged quality. Our contributions are as follows:

- **A 3D-shape-as-code modeling agent.** We explore the paradigm of 3D shape as code and introduce **Procedura**, an agentic framework that scales the coding ability of a frozen LLM to detailed, many-part 3D shapes from a text prompt with no 3D training.
- **Procedural assembly with typed mates.** We represent an object as a parametric CSG program whose named parts are joined by typed, machine-checkable mates, so placement is solved, every commit is verified, and every part stays editable by name. The same graph further carries per-part materials and a simulator-validated articulation.
- **State-of-the-art results.** On P3D-Bench and on MechBench-36, **Procedura** outperforms state-of-the-art native 3D generators and all prior 3D-code agents, with the sharpest edges and the only editable, part-structured output.

2 Related Work

Native 3D generation. Native generators synthesize a shape directly as continuous geometry. They distill a 2D diffusion prior (Poole et al., 2023; Lin et al., 2023; Wang et al., 2023), map a prompt

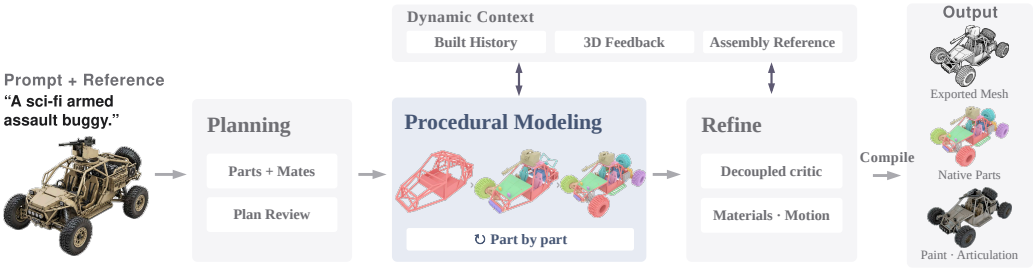


Figure 2. Overview of Procedura. From a text prompt, **Procedura** synthesizes a reference view, plans an assembly graph of parts joined by typed mates, builds the program part by part under solved placement and compile, mate, and connectivity checks, and refines it with a decoupled vision critic. The same graph carries per-part materials and a simulator-validated articulation.

or an image to a point cloud, an implicit field, or Gaussians in one pass (Nichol et al., 2022; Jun & Nichol, 2023; Zhao et al., 2023; Hong et al., 2024; Tang et al., 2024), or, with native 3D diffusion, recover mesh geometry of striking fidelity (Wu et al., 2024a; 2025; Xiang et al., 2025; Team Hunyuan3D, 2025; Jia et al., 2025), while autoregressive models write the mesh token by token (Nash et al., 2020; Siddiqui et al., 2024; Chen et al., 2025). However, all of them optimize fidelity to the input view, so they emit dense, soft meshes with no named parts or editable dimensions. Post-hoc segmentation (Yang et al., 2024b; Liu et al., 2025; Yang et al., 2026b) names parts afterwards but attaches no dimensions or relations to them. We instead generate the assembly program that builds the object, which makes parts, dimensions, and relations explicit.

Program-based 3D generation. Early work induces CSG trees (Sharma et al., 2018; Kania et al., 2020) or assembly programs (Jones et al., 2020), CAD-native methods learn sketch-extrude sequences from engineering datasets (Wu et al., 2021; Willis et al., 2021; Xu et al., 2022), and text-conditioned models write parametric 3D programs or construction commands (Makatura et al., 2023; Khan et al., 2024) and label the parts of existing ones (Yuan et al., 2024). Production CAD treats assembly as a first-class graph of typed mates, following the design-for-assembly practice of matched male–female pairs that share one nominal and a fit (Boothroyd et al., 2010), and learned-assembly methods predict mates or joints between existing parts (Jones et al., 2021; Willis et al., 2022). However, no generative system authors this structure, and single-shot program writers miscount and misplace the parts of many-part objects (Yang et al., 2026a). **Procedura** generates the parts and their mate graph together, so the structure that assembly research takes as input becomes our output.

Agentic 3D-code generation. Recent agents wrap an LLM in a draft-execute-revise loop over 3D code (Fan et al., 2026; Barkley et al., 2026; Shui et al., 2026; Zhou et al., 2026; Adam CAD, 2026), domain-tuned coders fine-tune a model to emit CAD code (Guan et al., 2025; Kolodiazhnyi et al., 2025), and related pipelines articulate generated assets (Zhou et al., 2026; Shui et al., 2026) or existing meshes (Le et al., 2025) or compose scenes (Feng et al., 2023; Yang et al., 2024a; Hu et al., 2024; Deitke et al., 2022; Raistrick et al., 2023). These systems place parts by frames or transforms the model authors, with no check on the compiled mesh that two parts register, and they fit joint axes to finished surfaces. In **Procedura**, every relation is a typed mate measured on the compiled mesh, so placement is solved and connectivity is enforced during generation instead of repaired afterwards (Attene et al., 2013; Huang et al., 2020), and a hinge is the very mate that seated its two parts. Its critic is also independent of the coding model, whereas self-refinement lets a model review its own output (Madaan et al., 2023; Shinn et al., 2023; Yang et al., 2024c), a reviewer that often misses its own errors (Huang et al., 2024).

3 Method

Given a text prompt, our goal is a 3D model that meets the three requirements of Sec. 1: high-precision geometry, a meaningful part decomposition, and editability. **Procedura** therefore generates the program that builds the mesh. One pass cannot write it, since every dimension and placement of a many-part object must agree, so **Procedura** commits one part at a time and checks every commit. It represents the object as a procedural assembly of typed mates (Sec. 3.1), plans the assembly graph (Sec. 3.2), builds the program under solved placement and verification (Sec. 3.3), refines it with a decoupled critic (Sec. 3.4), and adds per-part materials and articulation (Sec. 3.5).

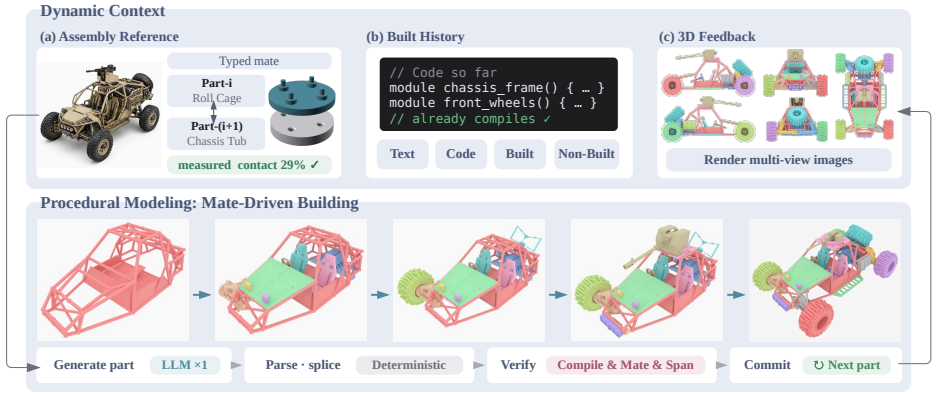


Figure 3. Mate-driven building. Top: the dynamic context of one generation call, (a) assembly reference, (b) built history, (c) 3D feedback. Bottom: the assault buggy growing one verified part at a time (2, 10, 18, 24, 28 parts).

3.1 Procedural Assembly Representation

Procedura represents an object as a procedural assembly $A = (P, C)$, where $P = \{p_1, \ldots, p_n\}$ is an ordered set of parts, each a named parametric CSG module, and C is a set of mates, each a typed relation between the mate frames of two parts; the shipped geometry is the compiled mesh. This representation answers the three requirements by construction. Geometry is precise since a program's faces are planes and its bores are cylinders, so its edges stay sharp at any resolution. The part decomposition is the module list itself, so it is exact and needs no separate segmentation model. Editing is a parameter change, since every dimension is a named parameter bound to a named part. Mates also record how two parts join, so placement can be solved and verified and the same relation later drives articulation.

Typed mates. A mate $c = (i, j, \text{type}, F_i, F_j, d, \phi)$ joins a frame F_j published by the new part p_j to a frame F_i published by an earlier part p_i. Its type is drawn from a closed vocabulary of nine static mating features (Fig. 6) and three kinematic ones (revolute, prismatic, and spherical). The two halves share one nominal dimension d, and a fit class $\phi \in \{\text{clearance, location, press, snap}\}$ resolves to a signed per-side offset in millimetres. Two rules follow. Both halves of a mate derive from the same parameter d plus a fit offset, never from two literals, so an edit to one side can never orphan the other; and each type fixes a degree-of-freedom signature, so a static mate fully locates its part while a kinematic mate leaves exactly the intended motion free.

3.2 Assembly-Graph Planning

The plan fixes the parts and their order before any geometry is written, since the build never revisits the list. Given the prompt, **Procedura** first synthesizes a reference view that grounds every later stage. A single vision call then decomposes the object into an ordered list of named parts. Each part carries a module name, a detail level, and a description that fixes its relative size, aspect, and pose against its neighbours. Each part also declares its typed mates to parts placed earlier, naming the partner, mate type, fit class, and feature count; the shared nominal is read at build time from the partner's parameter. The list is a topological order of the assembly graph, so the build seats each module on existing geometry. A second vision call reviews the plan against the reference and may revise it in only one direction. It may add a missing part or sharpen a description, but never merge, remove, rename, or reorder, since each part is exactly one generation call and merging two parts removes detail; we enforce the rule in code. Removing the plan is the costliest ablation (Sec. 4.4).

3.3 Mate-Driven Building

One part per call. A single call can write the whole program for a simple shape, but a detailed, many-part object forces it to fix every part, count, and placement at once, and the model loses track, leaving misplaced sub-assemblies and floaters. To address this, the build grows the program one module at a time in plan order, so each call only adds a single part to a model that already exists.

The call emits only the new part, in a strict five-block format of parameters, helpers, body, interface frames, and placement intent. A one-shot mode that writes the whole program in one call is our experimental control (Sec. 4).

Dynamic context. Each generation call sees three grounded sources (Fig. 3). The assembly reference is the reference image, the text prompt, and the plan slice, namely the part to add, its mates, and the parts not yet built. The built history is the full accumulated program, passed as text, and because it compiles, the model reads exact parameters instead of guessing the scale. The 3D feedback is a parts-colour render of the build from up to twelve viewpoints, where each committed part carries a distinct colour so the model can see where its part attaches (ablated in Sec. 4.4).

Solved placement. The model authors the part's geometry and its mate frames but not the assembly transform. For a static mate joining the new part's frame F_j to the placed partner frame F_i, the placement is the rigid transform that aligns them:

$$T_j \;=\; F_i \cdot \Delta(\phi) \cdot F_j^{-1}, \tag{1}$$

where $\Delta(\phi)$ denotes the signed fit offset along the mate axis. Solving T_j in code removes the misplaced parts that an estimated transform causes in prior 3D-code systems (Yang et al., 2026a), since the model never emits a transform it read off a render. The five blocks are then spliced into the program at fixed markers, so earlier parts are never re-emitted or corrupted.

Verification stack. A spliced part must clear three deterministic checks before it joins the build. First, a compile gate feeds any compiler error back and regenerates the part. Second, a mate gate measures each of the part's static mates on the compiled mesh, testing the registration area $a(c)$ of the mated surfaces and the penetration depth $\delta(c)$ between the two bodies:

$$a(c) \geq \tau_a \, d^2 \quad \text{and} \quad \delta(c) \leq \delta_{\max}(\phi), \tag{2}$$

where d denotes the mate's nominal and $\delta_{\max}$ the tolerance its fit class allows, so a part that merely hovers near its partner, or gouges into it, is rejected with the measured values returned in the feedback. Third, a connectivity gate rejects the part only if it raises the count of visible floaters (Sec. 3.4) over the running baseline, so a part that connects late is never blocked; a part whose mate has registered is exempt.

3.4 Decoupled-Critic Refinement

A refine loop then polishes the draft. Each cycle renders the build, has a critic diagnose it against the reference, and fixes the top-ranked defect. The defining choice is to decouple the critic from the fixer. The critic is a single-turn vision call on the same model with its own prompt and context, never a turn of the coding agent, and returns a prioritized issue list that names the responsible module and a fix direction for each issue. We decouple the two because a model that judges its own output carries the very assumptions that produced the error, because the diagnosis then becomes a persisted artifact rather than hidden reasoning, and because a history-aware critic that receives the trail of prior diagnoses can verify that earlier fixes landed and flag regressions (App. C.2). In code, each diagnosis sets a one-shot latch that exactly one edit consumes, and the critic refuses stale views (App. A.3). Since abutting parts compile to disconnected solids that look joined in a render, `finish(ok)` is refused while any visible floater remains, that is, any component besides the main body whose bounding-box span reaches $\tau = 0.01$ of the model's; span, unlike volume, catches thin detached panels (App. A.2).

3.5 Materials and Articulation

Geometry alone does not make a production asset, so the same graph also carries a physically-based material for every part and a kinematic tree over its mates (Apps. A.4 and A.5). Materials are read from the reference into a compact library, assigned per named part, and corrected by a separate material critic, and a sub-part pass that splits coarse modules can never alter the shape. Articulation needs no new structure, since the mate network is already the kinematic graph; measured symmetry axes and contacts seed each joint, and the exported OpenUSD articulation and URDF are validated headless in Isaac Sim.

Table 1. MechBench-36 (Figs. 4 and 5). Every method is re-rendered through one rig and scored by the same judge and CLIP backbone, so the only difference between rows is the method. Geo, Aes, and Sem are the judge's geometry, aesthetics, and semantic axes, mapped per case to 0–1 as in Tab. 2, and Overall is their mean. Shrp_{60} is total sharp-edge length above a 60° dihedral and Dih_{95} the 95th-percentile dihedral, both on the shipped mesh. A case with no shipped mesh scores zero and stays in the mean. Single-shot rows are one call with no plan, build, or refine stage. Codex agent is a general-purpose coding agent, the Codex CLI on GPT-6-astra, given the same prompt, reference view, and an OpenSCAD binary. Best per column in bold.

	Judge					Edge structure	
Method	Geo↑	Aes↑	Sem↑	Overall↑	CLIP↑	Shrp_{60} ↑	Dih_{95} ↑
Procedura (GPT-6-astra)	0.673	0.694	**0.716**	**0.694**	82.50	176.23	99.59
Procedura (Gemini 3.7 Flash)	**0.679**	**0.698**	0.694	0.690	**82.67**	158.08	97.97
Procedura (GPT-5.6-sol)	0.673	0.685	0.701	0.686	81.76	**185.18**	**105.28**
Agentic 3D-code							
Codex agent (GPT-6-astra)	0.673	0.685	0.698	0.685	82.38	109.11	92.74
Adam CAD (GPT-5.6-sol) (Adam CAD, 2026)	0.642	0.642	0.701	0.662	81.26	57.35	90.41
Articraft (GPT-5.6-sol) (Zhou et al., 2026)	0.614	0.512	0.685	0.604	79.12	41.31	94.74
CAD-Coder (Guan et al., 2025)	0.194	0.265	0.188	0.216	60.62	7.63	92.91
cadrille (Kolodiazhnyi et al., 2025)	0.179	0.179	0.176	0.178	50.65	19.58	97.18
Single-shot LLM							
GPT-5.6-sol	0.660	0.623	0.704	0.663	80.91	60.39	89.76
Gemini 3.7 Flash	0.651	0.630	0.701	0.660	80.86	63.54	91.35
Gemini 3.1 Pro	0.617	0.568	0.694	0.627	79.94	57.81	90.05
Native 3D generation							
TRELLIS.2 (Xiang et al., 2026)	0.642	0.660	0.698	0.667	77.51	134.02	73.11
UltraShape (Jia et al., 2025)	0.627	0.614	0.694	0.645	75.04	8.58	33.06
Hunyuan3D (Team Hunyuan3D, 2025)	0.580	0.525	0.685	0.597	74.62	6.41	35.12
Direct3D-S2 (Wu et al., 2025)	0.540	0.500	0.673	0.571	74.90	35.17	52.78

4 Experiments

Our experiments answer three questions. Does mate-driven shape-as-code make a frozen LLM a state-of-the-art 3D generator against native generators, prior 3D-code agents, and single-shot prompting of the same models (Secs. 4.2 and 4.3)? What do the pipeline's stages contribute (Sec. 4.4)? And does the program deliver a full production asset (Sec. 4.5)?

4.1 Setup and Protocol

Benchmarks. We evaluate on two benchmarks. MechBench-36, our hard-surface benchmark, is a frozen set of 36 deliberately many-part objects, rovers, mechs, cranes, excavators, engines, and landing gear, chosen because they stress part counts, placement, and connectivity. The assembly task of P3D-Bench (Yang et al., 2026a) has 203 text-and-image cases scored by the benchmark's own assembly judge. For evaluation we lock the prompt and the synthesized reference view and hand the same pair to every method, so image synthesis cannot confound the comparison.

Blind judging protocol. Every shipped model is re-rendered from one fixed view set under one grey-clay rig, assigned an opaque hash, and judged blind, so method names never reach the judge. A vision LLM (Wu et al., 2024b) scores each render with one shared prompt on three axes, semantic fidelity, geometric quality, and aesthetics, the last rating the object as an engineered part. Alongside the judge we report CLIP (Radford et al., 2021), the image-text alignment between the prompt and the shared renders, and two edge measurements on the shipped mesh, sharp-edge length and the 95th-percentile dihedral, all under the one shared rig (App. B.2).

Models and implementation. Unless noted, planning, building, and the critic all use one frozen deep-reasoning LLM, Gemini 3.7 Flash; we also report GPT-5.6-sol and GPT-6-astra variants of the full pipeline. On MechBench-36 the judge is Gemini 3.8 Flash in its high-reasoning mode, blinded by hash; P3D-Bench keeps its own assembly judge. The program is realized as OpenSCAD and compiled by the OpenSCAD engine; beyond the compiler, the only external binaries are Blender for renders and a headless Isaac Sim for physical validation (App. B.1).

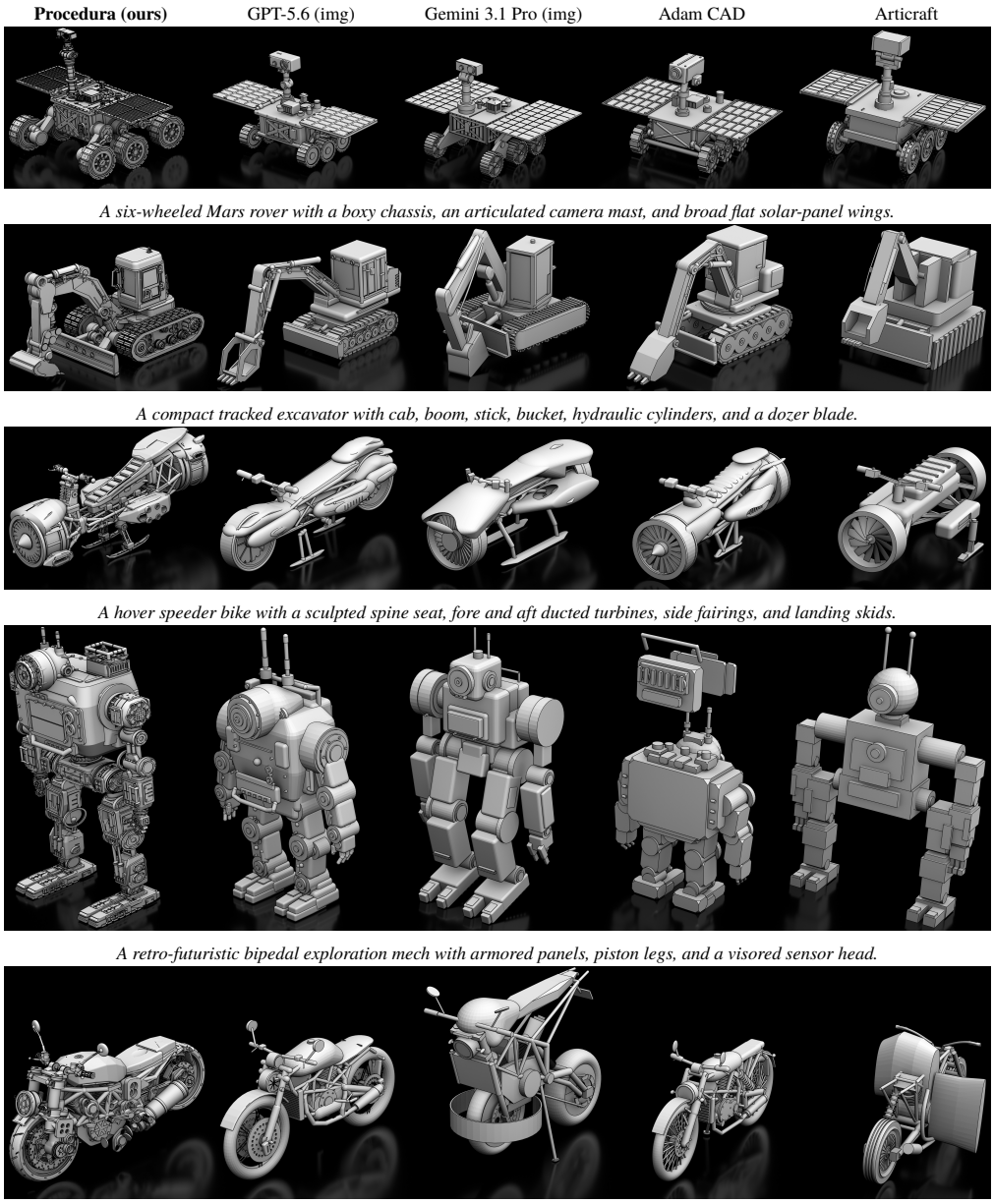


Figure 4. Qualitative comparison against LLM-driven 3D generation. Prior agents keep the silhouette but collapse the mechanism into a few boxes; **Procedura** models each part separately (all six in Fig. 14).

4.2 Results on MechBench-36

As shown in Tab. 1, the three **Procedura** arms take the top three judge composites, 0.694 with GPT-6-astra, 0.690 with Gemini 3.7 Flash, and 0.686 with GPT-5.6-sol; the Gemini 3.7 Flash arm leads geometry, aesthetics, and CLIP, and every arm ships a mesh for every case. The Codex agent on GPT-6-astra follows at 0.685, and the strongest remaining methods are TRELLIS.2 at 0.667, single-shot GPT-5.6-sol at 0.663, and Adam CAD at 0.662. Semantic barely separates the field, spanning 0.673 to 0.716 across the methods that work, since it mostly asks whether the object is of the right kind, whereas geometry and aesthetics span 0.540 to 0.679 and 0.500 to 0.698, and **Procedura** tops both.

Comparison with native generators. The strongest native generator, TRELLIS.2, trails by 5.2 CLIP points, and Direct3D-S2 and Hunyuan3D fall behind on aesthetics by 0.20 and 0.17, their soft surfaces fusing fine features into the body (Fig. 5).

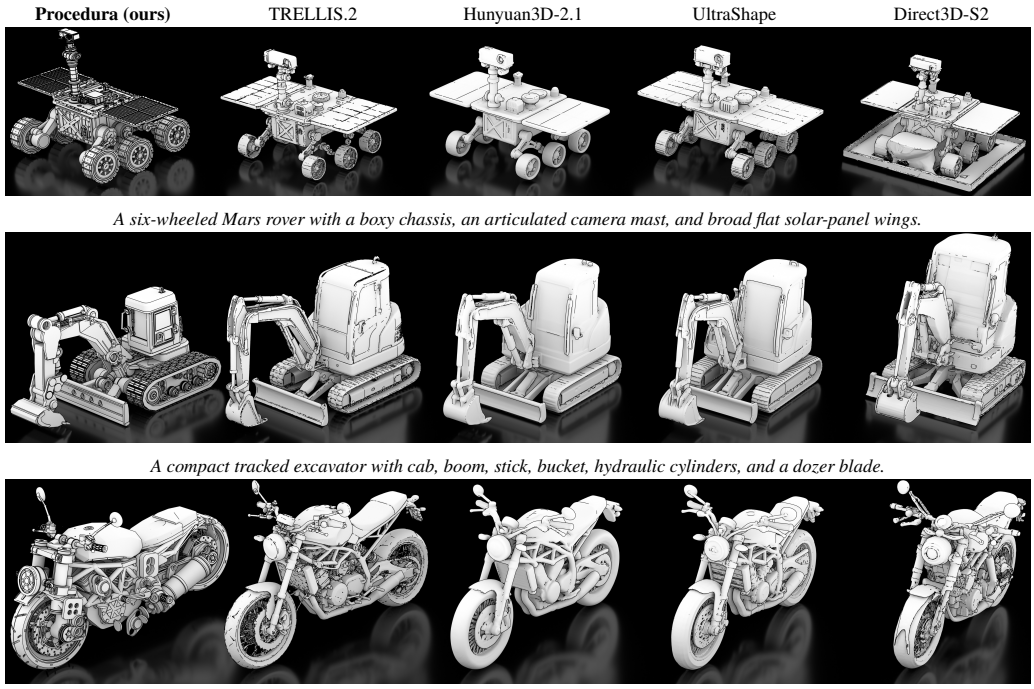


Figure 5. Qualitative comparison against native 3D generators. Three objects (all six in Fig. 15), each generated by four native 3D diffusion models and by **Procedura** (left) from the same locked reference image and rendered with the same rig. The natives reproduce the silhouette, but their soft surfaces fuse bolt heads, vents, and panel lines into one skin with no part boundary, whereas **Procedura**'s compiled CSG keeps edges sharp and every part separable.

Comparison with prior 3D-code agents. The general-purpose Codex agent on GPT-6-astra comes closest, at 0.685 overall and 82.38 CLIP, but writes fewer sharp features (Tab. 1, Shrp_{60}; side by side in Fig. 16). The strongest prior 3D-code agent, Adam CAD on GPT-5.6-sol, reaches 0.642 aesthetics and 81.26 CLIP. The domain-tuned coders CAD-Coder and cadrille collapse on many-part objects, scoring 0.265 and 0.179 aesthetics and shipping no mesh on 1 and 5 cases, since a model trained to emit one sketch-extrude body cannot express an assembly (Tab. 1).

Edge structure. **Procedura** leads both edge columns. Our programs carry 158.1, 185.2, and 176.2 units of sharp edge above a 60° dihedral with Gemini 3.7 Flash, GPT-5.6-sol, and GPT-6-astra, 1.4–1.7× the Codex agent's 109.1 and 2.5–2.9× every other code-emitting method, at 95th-percentile dihedrals of 98.0° to 105.3°. The native generators sit at 33–73°, since a regressed surface has no notion of a face or a bore; TRELLIS.2's comparable sharp-edge length of 134.0 comes with a 73.1° dihedral, far shallower creases than ours, whose edges are exact intersections of analytic surfaces.

Paired gain over the base model. Paired per object against a single call to the same frozen model, the pipeline improves aesthetics by +0.068 on Gemini 3.7 Flash (18 wins, 0 losses, sign test $p < 0.001$) and by +0.062 on GPT-5.6-sol (14 wins, 0 losses, $p < 0.001$), and the judge composite by +0.030 ($p < 0.001$) and +0.024 ($p = 0.012$). CLIP improves by +1.80 ($p = 0.011$) and +0.85 respectively, the latter not significant (App. B.4).

4.3 Results on P3D-Bench

We re-render and re-judge every method under the shared harness, so every row of Tab. 2 is directly comparable. **Procedura** solves all 203 cases with either base model and leads every column. The composite reaches 0.590 on Gemini 3.7 Flash and 0.575 on GPT-5.6-sol, against 0.524 for the strongest entry that also covers the full benchmark, a margin several times the 95% intervals. The lead is largest on geometry, 0.490 against 0.392, where the mate-driven build acts, since solved placement and verified connectivity are geometric properties. Prompting our two base models once

Table 2. P3D-Bench assembly results. Every method is re-rendered from the same rig and scored by the benchmark's assembly judge on its three axes, normalized to 0–1; Judge (Overall) is their mean, the benchmark's composite, with 95% intervals. Baselines are the benchmark's entries, re-run under the same harness, plus single-shot prompting of our two base models; App. B.3 lists the four weaker entries omitted here.

Method	Semantic↑	Geometry↑	Aesthetics↑	Judge (Overall)↑
Procedura (Gemini 3.7 Flash)	**0.666**	**0.490**	**0.616**	**0.590** ±0.009
Procedura (GPT-5.6-sol)	0.661	0.460	0.604	0.575 ±0.010
Gemini 3.7 Flash	0.662	0.458	0.577	0.566 ±0.016
GPT-5.6-sol	0.662	0.444	0.584	0.563 ±0.015
GPT-5.5	0.643	0.392	0.537	0.524 ±0.012
Gemini 3.1 Pro	0.639	0.390	0.508	0.513 ±0.012
Opus 4.6	0.572	0.327	0.446	0.448 ±0.016
Kimi K2.6	0.523	0.294	0.372	0.396 ±0.014

Table 3. Ablations on MechBench-36. All variants use the same base model and are judged in the same single session as Tab. 1, with the same normalization ($n = 36$). Ablated variants are evaluated at the draft stage, before refine, so the planning and 3D-feedback deltas also include the absence of the refine stage.

Variant	Geo↑	Aes↑	Sem↑	Judge↑	CLIP↑
Procedura (full)	0.679	0.698	0.694	0.690	82.67
w/o refine	0.664	0.694	0.704	0.687	81.84
w/o planning	0.664	0.670	0.701	0.678	82.36
w/o 3D feedback	0.657	0.682	0.707	0.682	81.96

already reaches 0.566 and 0.563 on the sets those runs cover, so part of the margin over published rows reflects base-model strength; the agent supplies the rest.

4.4 Ablation Study

Tab. 3 removes the refine loop, the assembly planning, or the per-part 3D feedback and re-judges the 36 cases under the same regime as Tab. 1. Every removal lowers geometry, aesthetics, the composite, and CLIP, and the full pipeline is the best variant on all four; semantic stays within 0.694–0.707 and does not separate the variants. Dropping the refine loop lowers geometry from 0.679 to 0.664, the cleanest single-component read, since that variant differs from the full pipeline by refine alone; the composite moves less, to 0.687, consistent with refine correcting shape rather than content. Dropping the planning stage costs the most, 0.678 overall with the largest aesthetics drop, from 0.698 to 0.670, so the assembly graph the build consumes is load-bearing. Dropping the per-part 3D feedback costs the most geometry, 0.657. App. C.2 traces the refine loop on one of the hardest cases, and App. C.1 varies the base model's reasoning effort.

4.5 Materials and Articulation

Every named module carries its own material, painted from the reference alone (Figs. 8 and 12); we report no paint metric, since no ground-truth material map exists. Articulation is checked in the physics simulator, where 14 of the 18 objects we articulated clear every dynamics phase and 9 also clear the advisory asset rules (Figs. 9 and 13).

5 Limitations and Conclusion

Three limitations remain. First, the agent perceives the build only through rendered images, so occluded interiors and fine contact geometry can hide from every camera; depth maps or direct mesh queries would strengthen the loop. Second, CSG captures mechanical, hard-surface objects precisely, whereas organic, freeform surfaces are expressed less naturally than in a neural field. Third, the mate vocabulary covers rigid joints only, and a mate the plan never declares cannot be verified.

We presented **Procedura**, an agent that writes a 3D object as a parametric program whose named parts are joined by typed, machine-checkable mates, using a frozen LLM with no 3D training. On P3D-Bench and MechBench-36, judged blind, it leads every native generator and prior 3D-code agent on judged quality, has the sharpest edges of any method we measure, and is the only one whose output is an editable, part-structured program.

Reproducibility statement

Sec. 3 specifies the representation, the mate vocabulary, the placement solve, and every gate with its threshold, and App. A gives the remaining procedures: the fit offsets of the mate library, the span-based connectivity gate, the refinement tools, and the materials and articulation stages. Sec. 4.1 and App. B describe both benchmarks, the blind judging protocol, the normalization of judge scores, the edge metrics, the paired tests, and the exact base and judge models, so that every number in Tabs. 1 to 3 can be recomputed from the shipped meshes under the stated rig. The single-shot controls are defined in Tab. 1, and the ablation protocol, including the draft-stage caveat, in Tab. 3.

Ethics statement

This work generates synthetic 3D assets of machines, vehicles, and tools from text prompts. It involves no human subjects, no personal data, and no scraped 3D datasets, since the method requires no 3D training. Some benchmark prompts describe fictional, game-style military vehicles; the outputs are coarse, part-level CSG props with no functional detail, and the same representation serves ordinary robotics and manufacturing assets. The blind judging protocol was designed to remove evaluator bias, and we report the cases where our method does not separate from its controls.

AI use statement

Large language models are the object of study in this work: a frozen LLM is the generator inside **Procedura**, and a vision LLM serves as the blind judge, both described in Sec. 4.1. We have not used generative AI tools to implement the method, generate synthetic data sets, design or provide feedback on research methodology or experiments, propose or refine hypotheses, or interpret results, and developing theoretical models or conceptual frameworks, formulating mathematical claims or proofs, translation, dataset cleaning, and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to edit the paper for readability, to help write and debug evaluation, rendering, and figure-generation code, to create or modify figures, and to check references against the cited papers. All AI-assisted code was reviewed and tested by the authors, all AI-edited text was checked for technical accuracy, and every reported number was recomputed from the original artifacts. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

References

Adam CAD. Adam: AI CAD copilot for hardware teams. `https://adam.new/`, 2026. Commercial product; accessed 2026-07-19.

Marco Attene, Marcel Campen, and Leif Kobbelt. Polygon mesh repairing: An application perspective. *ACM Computing Surveys*, 45(2), 2013.

Jesse Barkley, Rumi Loghmani, and Amir Barati Farimani. CADSmith: Multi-agent CAD generation with programmatic geometric validation. *arXiv preprint arXiv:2603.26512*, 2026.

Geoffrey Boothroyd, Peter Dewhurst, and Winston A. Knight. *Product Design for Manufacture and Assembly*. CRC Press, 3rd edition, 2010.

Yiwen Chen, Tong He, Di Huang, Weicai Ye, Sijin Chen, Jiaxiang Tang, Zhongang Cai, Lei Yang, Gang Yu, Guosheng Lin, and Chi Zhang. MeshAnything: Artist-created mesh generation with autoregressive transformers. In *International Conference on Learning Representations (ICLR)*, 2025.

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

Fengxiao Fan, Jingzhe Ni, Xiaolong Yin, Sirui Wang, Xingyu Lu, Qiang Zou, Ruofeng Tong, Min Tang, and Peng Du. CADDesigner: Conceptual CAD model generation with a general-purpose agent. *Computer-Aided Design*, 198:104087, 2026.

Weixi Feng, Wanrong Zhu, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. LayoutGPT: Compositional visual planning and generation with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Gemini Team. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. CAD-Coder: Text-to-CAD generation with chain-of-thought and geometric reward. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. LRM: Large reconstruction model for single image to 3D. In *International Conference on Learning Representations (ICLR)*, 2024.

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 (ICML)*, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024.

Jingwei Huang, Yichao Zhou, and Leonidas Guibas. ManifoldPlus: A robust and scalable watertight manifold surface generation method for triangle soups. *arXiv preprint arXiv:2005.11621*, 2020.

Tanghui Jia, Dongyu Yan, Dehao Hao, Yang Li, Kaiyi Zhang, Xianyi He, Lanjiong Li, Yuhan Wang, Jinnan Chen, Lutao Jiang, Qishen Yin, Long Quan, Ying-Cong Chen, and Li Yuan. UltraShape 1.0: High-fidelity 3D shape generation via scalable geometric refinement. *arXiv preprint arXiv:2512.21185*, 2025.

Benjamin Jones, Dalton Hildreth, Duowen Chen, Ilya Baran, Vladimir G. Kim, and Adriana Schulz. AutoMate: A dataset and learned model for fixturing and mating in CAD assemblies. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 40(6), 2021.

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 (SIGGRAPH Asia)*, 39(6), 2020.

Heewoo Jun and Alex Nichol. Shap-E: Generating conditional 3D implicit functions. *arXiv preprint arXiv:2305.02463*, 2023.

Kacper Kania, Maciej Zięba, and Tomasz Kajdanowicz. UCSG-Net: Unsupervised discovering of constructive solid geometry tree. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multi-modal CAD reconstruction with reinforcement learning. *arXiv preprint arXiv:2505.22914*, 2025.

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 (ICLR)*, 2025.

Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3D: High-resolution text-to-3D content creation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Minghua Liu, Mikaela Angelina Uy, Donglai Xiang, Hao Su, Sanja Fidler, Nicholas Sharp, and Jun Gao. PartField: Learning 3D feature fields for part segmentation and beyond. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Liane Makatura, Michael Foshey, Bohan Wang, Felix Hähnlein, Pingchuan Ma, Bolei Deng, Megan Tjandrasuwita, Andrew Spielberg, Crystal Elaine Owens, Peter Yichen Chen, et al. How can large language models help humans in design and manufacturing? *arXiv preprint arXiv:2307.14377*, 2023.

Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. PolyGen: An autoregressive generative model of 3D meshes. In *International Conference on Machine Learning (ICML)*, 2020.

Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-E: A system for generating 3D point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. In *International Conference on Learning Representations (ICLR)*, 2023.

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 *International Conference on Machine Learning (ICML)*, 2021.

Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, Alejandro Newell, Hei Law, Ankit Goyal, Kaiyu Yang, and Jia Deng. Infinite photorealistic worlds using procedural generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: Neural shape parser for constructive solid geometry. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

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.

Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner. MeshGPT: Generating triangle meshes with decoder-only transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Jiaxiang Tang, Zhaoxi Chen, Xiaokang Chen, Tengfei Wang, Gang Zeng, and Ziwei Liu. LGM: Large multi-view Gaussian model for high-resolution 3D content creation. In *European Conference on Computer Vision (ECCV)*, 2024.

Team Hunyuan3D. Hunyuan3D 2.1: From images to high-fidelity 3D assets with production-ready PBR material. *arXiv preprint arXiv:2506.15442*, 2025.

Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. ProlificDreamer: High-fidelity and diverse text-to-3D generation with variational score distillation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 Gallery: A dataset and environment for programmatic CAD construction from human design sequences. *ACM Transactions on Graphics (SIGGRAPH)*, 40(4), 2021.

Karl D. D. Willis, Pradeep Kumar Jayaraman, Hang Chu, Yunsheng Tian, Yifei Li, Daniele Grandi, Aditya Sanghi, Linh Tran, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. JoinABLe: Learning bottom-up assembly of parametric CAD joints. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: A deep generative network for computer-aided design models. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Jingxi Xu, Philip Torr, Xun Cao, and Yao Yao. Direct3D: Scalable image-to-3D generation via 3D latent diffusion transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a.

Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Yikang Yang, Yajie Bao, Jiachen Qian, Siyu Zhu, Xun Cao, Philip Torr, and Yao Yao. Direct3D-S2: Gigascale 3D generation made easy with spatial sparse attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024b.

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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. SkexGen: Autoregressive generation of CAD construction sequences with disentangled codebooks. In *International Conference on Machine Learning (ICML)*, 2022.

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*, 2026a.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3D embodied AI environments. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024a.

Yunhan Yang, Yukun Huang, Yuan-Chen Guo, Liangjun Lu, Xiaoyang Wu, Edmund Y. Lam, Yan-Pei Cao, and Xihui Liu. SAMPart3D: Segment any part in 3D objects. *arXiv preprint arXiv:2411.07184*, 2024b.

Yunhan Yang, Yuan-Chen Guo, Yukun Huang, Zi-Xin Zou, Zhipeng Yu, Yangguang Li, Yan-Pei Cao, and Xihui Liu. HoloPart: Generative 3D part amodal segmentation. In *International Conference on Learning Representations (ICLR)*, 2026b.

Zhengyuan Yang, Jianfeng Wang, Linjie Li, Kevin Lin, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. Idea2Img: Iterative self-refinement with GPT-4V for automatic image design and generation. In *European Conference on Computer Vision (ECCV)*, 2024c.

Haocheng Yuan, Jing Xu, Hao Pan, Adrien Bousseau, Niloy J. Mitra, and Changjian Li. CADTalk: An algorithm and benchmark for semantic commenting of CAD programs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Zibo Zhao, Wen Liu, Xin Chen, Xianfang Zeng, Rui Wang, Pei Cheng, Bin Fu, Tao Chen, Gang Yu, and Shenghua Gao. Michelangelo: Conditional 3D shape generation based on shape-image-text aligned latent representation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

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.

A Method Details

A.1 The Mate Library

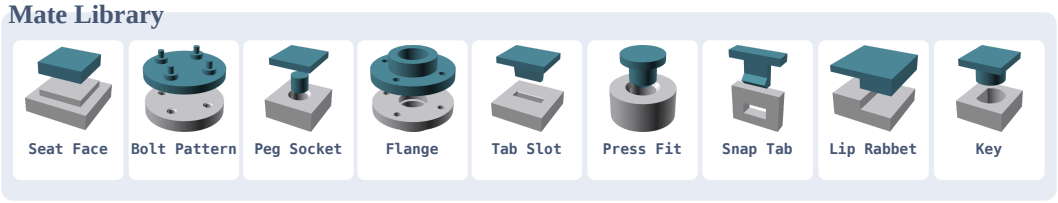


Figure 6. The mate library. The nine static mating features a part may declare, each rendered from the compiler macros the build emits. Teal is the male half added to the new part and grey the female half cut from its partner; both derive from one shared nominal dimension and a signed fit offset.

Each static mate kind of Sec. 3.1 is backed by a matched pair of compiler macros, a male feature that is unioned onto the new part and a female feature that is subtracted from its partner (Fig. 6). Both macros take the same nominal d and the fit class ϕ, which resolves to a signed per-side offset: clearance $+0.25$ mm, location $+0.15$ mm, press -0.05 mm, and snap $+0.20$ mm, tuned for the tolerances of an FDM-printed part. Every peg and every hole mouth carries a lead-in chamfer so a part self-centres as it seats. Locating and fastening are separate jobs: a snug register fixes position while looser clearance holes clamp, and a part that can only go one way is keyed by a small asymmetry so it cannot seat backwards. The seat-face, flange, lip-rabbet, and key kinds have no helper macro and are modelled by hand under the same shared-nominal rule. The generator may also model an interface by hand, but the five-block format and the shared-nominal rule are unchanged.

A.2 Span-Based Connectivity Gate

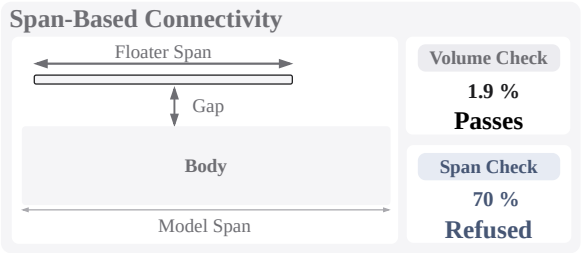


Figure 7. Span-based connectivity. A thin detached panel has near-zero enclosed volume, so a volume check passes and the floater ships; its bounding-box span covers most of the model and the defect is plainly visible. **Procedura** thresholds on span (Eq. (4)). Drawn to the worst case from our audit of a pre-gate pipeline: 70% span, 1.9% volume.

Let M be the compiled mesh. We weld coincident vertices and partition its triangles into connected components by union-find over shared edges:

$$C = \{c_1, \dots, c_K\} = \textsc{components}(M), \qquad \rho_i = \frac{\text{span}(c_i)}{\text{span}(M)}, \tag{3}$$

where $\text{span}(\cdot)$ is the largest side of the axis-aligned bounding box. Let the body be the largest component, $b = \arg\max_i \text{vol}(c_i)$. A component c_i with $i \neq b$ is a visible floater if its span fraction ρ_i reaches a threshold τ, and the gate passes only when none remain:

$$N_{\text{vis}} = |\{\, i \neq b : \rho_i \geq \tau \,\}|, \qquad \textsc{gate}(M) = [\, N_{\text{vis}} = 0 \,], \tag{4}$$

with $\tau = 0.01$. A volume-based criterion instead misses the floaters that occur in practice. A thin, flat panel has near-zero enclosed volume yet a large span, so it passes a volume check while remaining clearly detached (Fig. 7). We measure by span so the metric tracks what a viewer and the critic actually see. During the build (Sec. 3.3) the same measure is applied delta-aware: a part is rejected only when it raises N_{vis} over the running baseline, so a part that legitimately connects late is never blocked.

A.3 Refinement Loop Details

Agent-chosen context. The agent selects which views to render and how many (1 to 20) from a fixed catalog of orthographic faces, isometric corners, eye-level diagonals, and front tilts. On a fresh model it surveys broadly so no side hides a defect; once a problem is localized it renders the focused subset that best shows it. A refine step is bounded by a single module edit, and only edits count against the budget. After the edit, the agent compiles and, when the compile flags floaters, checks connectivity within the same cycle, so a broken edit is caught before it wastes the next review. The agent ends with `ok` when the critic reports no high-priority issue, or `give_up` when several edits fail to shrink the issue list.

Critic output. The critic returns a one-line `SUMMARY` and a prioritized `ISSUES` list, each issue tagged `[HIGH|MED|LOW]` and naming the responsible module with a one-sentence fix direction. It receives the trail of prior diagnoses, so it verifies that earlier fixes landed and flags regressions.

Gated finish. The terminal action `finish(ok)` is refused while $N_{\text{vis}} > 0$. Since the agent may edit after its last compile, the committed program is recompiled when stale, so the gate grades the mesh the user actually receives. When it refuses, it names the worst offenders with their bounding box so the next cycle has a concrete target. The `give_up` verdict is never gated, allowing a genuinely intentional separation to terminate.

A.4 Materials

Material library and assignment. Paint runs once the geometry is frozen and changes only colour and surface; the whole stage is four one-shot vision calls with no agent loop. **Procedura** first reads the reference image and catalogues every distinct surface as a compact material library, typically one to two dozen physically-based rendering (PBR) entries with albedo, class, roughness, metalness, and wear. It then assigns one library material to each named part, so the palette is grounded in the reference and shared across parts made of the same thing. As with geometry, a separate vision critic compares a render of the painted model to the reference and returns a corrected assignment.

Sub-part colour blocks. One material per module is often too coarse, since one wheel module carries a rubber tread, a painted rim, and a cast hub. **Procedura** therefore rewrites a module into sibling colour blocks so a sub-feature takes its own material, recompiling each rewrite and rejecting any whose triangle count moves beyond a fixed tolerance, so colour can never alter the shape. The stage emits a colour-tagged program, a multi-group `OBJ+MTL` with per-part PBR fields, and a Blender-Cycles studio render (Fig. 8); every artifact stays parts-addressable.

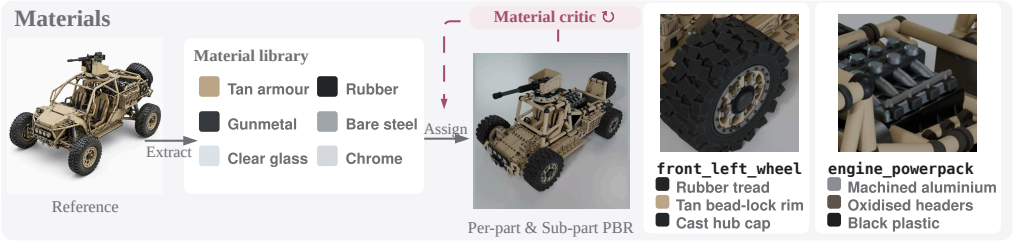


Figure 8. Per-part PBR materials from the reference. Procedura extracts a compact material library from the reference (25 entries for the assault buggy, six shown), assigns one entry to each of the 41 named parts, and a decoupled material critic corrects the assignment against the reference. A sub-part pass then splits a module into sibling colour blocks where one material is too coarse. Renders are Blender-Cycles PBR.

A.5 Articulation

Turning the same parts into a jointed, physically validated asset requires no new structure, since the mate network is already the kinematic graph. Static mates group parts into rigid links. Each kinematic mate, whether revolute, prismatic, or spherical, names its joint, axis, and anchor, and is either authored at plan time or promoted later from geometric evidence.

Geometric evidence. **Procedura** measures the geometry before any reasoning about motion. For each part it computes two kinds of evidence: rotational-symmetry axes, taken from the area-weighted covariance of the part's surface triangles, and contact regions between touching parts. Both are deterministic and yield concrete axis and anchor candidates, so a wheel's spin axis or a hinge line starts from measured geometry.

Joint planning and validation. A vision call then completes the articulation. It groups modules into links, builds the parent–child tree over the kinematic mates, and assigns each joint its axes, limits, and drives. The plan is refined against deterministic feedback and exported as an OpenUSD articulation and a URDF, so the same object drops into a physics simulator or a robotics stack. **Procedura** then validates the plan headless in Isaac Sim through a fixed battery: a dynamic settle, per-joint actuation sweeps, a rest-contact scan, a wheeled mobility test, and a URDF re-import. A flagged fault feeds up to eight rendered simulation frames back to the planner for one refine, so the critic can see a wrong axis (Fig. 9).

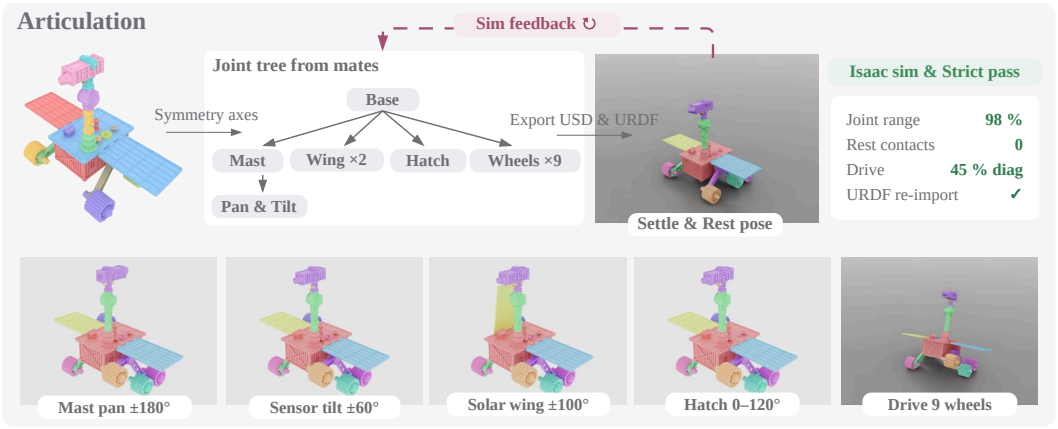


Figure 9. A sim-ready articulation validated in Isaac Sim. From the rover's named parts, **Procedura** measures geometric evidence (the rotational-symmetry axes and contact regions) and plans the joint tree over the kinematic mates (17 links, 16 joints). It exports an OpenUSD articulation and a URDF and validates them headless in Isaac Sim; a failed verdict feeds rendered simulation frames back to the planner for one refine. Every panel is a real Isaac frame in per-link colours: the settled rest pose, four per-joint actuation sweeps, and the rover after an all-wheel drive.

B Experimental Details

B.1 Implementation

Procedura is built on an in-house LLM harness. The program is realized as OpenSCAD and compiled by the OpenSCAD engine, with module parsing, splicing, the mesh parser, and the union-find connectivity analysis in process. Beyond the compiler, the only external binaries are Blender for renders and a headless Isaac Sim for physical validation. Each part receives up to three generation attempts in the build, and the refine loop is bounded by a fixed cycle budget; a case that exhausts the budget ships its last compiling program.

B.2 Judge Normalization and Edge Metrics

On MechBench-36 the judge sees no ground truth, since none exists for a prompt; on P3D-Bench it follows the benchmark's protocol. The judge scores each axis on a 1–10 scale; we map every per-case score to 0–1 by $\mathrm{clamp}((v-1)/9)$ and average over cases, so Tabs. 1 to 3 share one scale, and Overall is the mean of the three normalized axes. A case whose method ships no mesh scores zero and stays in the mean, which is why the domain-tuned coders sit low in Tab. 1. Sharp-edge length sums the length of mesh edges whose dihedral angle exceeds 60°; the threshold is deliberately steeper than the common 30°, since a shallower one also counts the ridges of a regressed surface and the cracks between fragments, so that only genuine machined creases survive. Both edge metrics are computed on the shipped mesh and are invariant to the render rig. All judge scores in Tab. 1 and Tab. 3 were

produced in a single Gemini 3.8 Flash judging session, since absolute scores from a served judge model are only comparable within one session; we re-judged every row together when the served Gemini 3.7 Flash model no longer reproduced its earlier scores. On the thirteen methods both judges scored, the two rankings agree closely (Spearman $\rho = 0.97$).

B.3 Full P3D-Bench Table

Table 4 lists every entry of Tab. 2, including the four weakest published entries omitted from the main text for space.

Table 4. P3D-Bench assembly results, all entries. Same protocol and normalization as Tab. 2.

Method	Semantic↑	Geometry↑	Aesthetics↑	Judge (Overall)↑
Procedura (Gemini 3.7 Flash)	**0.666**	**0.490**	**0.616**	**0.590** ± 0.009
Procedura (GPT-5.6-sol)	0.661	0.460	0.604	0.575 ± 0.010
Gemini 3.7 Flash	0.662	0.458	0.577	0.566 ± 0.016
GPT-5.6-sol	0.662	0.444	0.584	0.563 ± 0.015
GPT-5.5	0.643	0.392	0.537	0.524 ± 0.012
Gemini 3.1 Pro	0.639	0.390	0.508	0.513 ± 0.012
Opus 4.6	0.572	0.327	0.446	0.448 ± 0.016
Kimi K2.6	0.523	0.294	0.372	0.396 ± 0.014
Qwen 3.6	0.451	0.264	0.322	0.346 ± 0.014
Doubao	0.408	0.253	0.302	0.321 ± 0.014
GLM-5V Turbo	0.402	0.249	0.300	0.317 ± 0.013
MiMo Omni	0.379	0.242	0.294	0.305 ± 0.012

B.4 Paired Analysis

Because every row of Tab. 1 uses a frozen model, the single-shot rows serve as paired controls for the Gemini 3.7 Flash and GPT-5.6-sol pipeline arms on the same 36 cases. We report the mean per-case difference, the win/loss split over non-tied cases, and a two-sided sign test. Both arms improve aesthetics (+0.068, 18/0; +0.062, 14/0; both $p < 0.001$) and the judge composite (+0.030, 20/3, $p < 0.001$; +0.024, 16/4, $p = 0.012$). Geometry separates on Gemini 3.7 Flash only (+0.028, 8/1, $p = 0.039$; +0.012, 8/5, $p = 0.58$), semantic separates on neither, and CLIP improves on both (+1.80, $p = 0.011$; +0.85, $p = 0.41$).

C Additional Results

C.1 Reasoning Effort

Table 5. Reasoning effort on MechBench-36. The full pipeline on GPT-5.6-sol at three reasoning efforts, all from one code version with otherwise identical settings, judged by Gemini 3.8 Flash under the protocol and normalization of Tab. 1, all three rows in one judging session. That code version is later than the one behind the GPT-5.6-sol row of Tab. 1, so these rows compare efforts with each other rather than with that row. †35 of 36 cases.

Effort	Geo↑	Aes↑	Sem↑	Overall↑	CLIP↑	Shrp_{60} ↑	Dih_{95} ↑
low	0.590	0.590	0.688	0.622	77.40	94.26	102.39
medium	0.586	0.608	0.682	0.626	**79.10**	109.32	104.84
xhigh†	**0.613**	**0.644**	**0.689**	**0.649**	78.25	**125.84**	**109.50**

Table 5 varies GPT-5.6-sol's reasoning effort. Raising it from low to xhigh lifts geometry from 0.590 to 0.613, aesthetics from 0.590 to 0.644, the composite from 0.622 to 0.649, and sharp-edge length from 94.3 to 125.8, and the dihedral rises with it. Medium improves on low in aesthetics, sharp-edge length, and CLIP, and is on par with it on the judge composite (0.626 against 0.622).

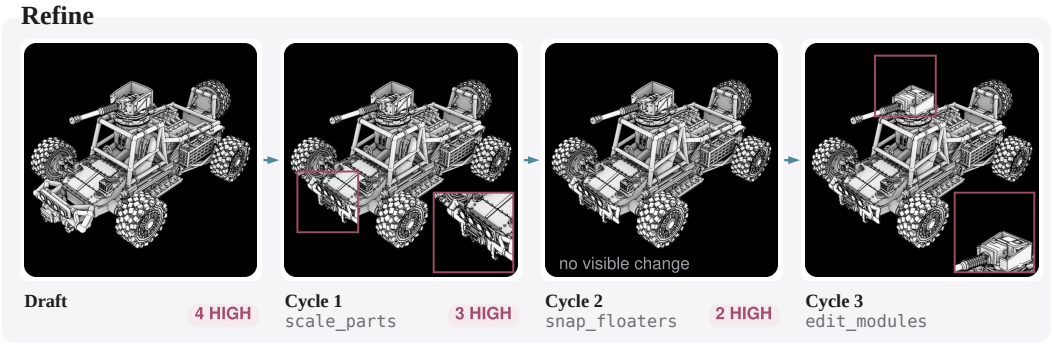


Figure 10. What each gated edit changes. The assault buggy, refined from the draft the build stage produced. Every frame is rendered from one fixed camera, so a difference between frames is a difference in geometry; the inset magnifies the densest region of pixels that cycle's edit changed, and the outline on the model marks where the inset is taken from. Each cycle is one fresh diagnosis followed by one gated edit; under every state we print the tool that produced it and the number of `HIGH` issues the critic reported for that state. The count falls $4 \to 3 \to 2$; the final state carries no count because the cycle budget ends before a further diagnosis. Cycle 2 is a sub-millimetre contact snap that changes connectivity but nothing a reader could see, and the frame says so.

C.2 Refinement Case Study

Fig. 10 traces the refine loop on one of the hardest benchmark cases. The critic's complaints are quantitative and structural. It reports the front winch 1.25× a road-tire diameter ahead of the front axle against roughly 0.6–0.7× in the reference, and a gun receiver at 0.19× the tire diameter against 0.30–0.35×, and the agent answers each with a single tool call on the named modules. Over three cycles the `HIGH` list shrinks monotonically, and the history-aware critic carries context forward, with cycle 3 opening on the rear equipment stack that "remains too low". The case ends on `max-steps` rather than converging, since one gated fix per fresh diagnosis makes progress linear in the cycle budget. One of the three edits is a `snap_floaters` call that closes sub-millimetre gaps, changing connectivity without changing the silhouette, which is why the cycle-2 frame looks unchanged and why we report the issue list rather than a pixel difference as the outcome.

C.3 Further Results Across the Benchmark

Figure 11 shows the part decomposition that the representation carries for free, Fig. 12 the texturing stage across ten benchmark objects, and Fig. 13 six articulated objects validated in Isaac Sim.

C.4 Full Qualitative Comparisons

Figures 14 and 15 show all six benchmark objects of the main-text comparisons against LLM-driven 3D generation and native 3D generators, each with the prompt printed beneath its row.

C.5 GPT-6-astra: Pipeline versus Codex Agent

Figure 16 places **Procedura** and the general-purpose Codex agent side by side, both on GPT-6-astra and given the same prompt and reference view. The two score within 0.01 of each other on the judge (Tab. 1), yet **Procedura**'s programs carry 1.6× the sharp-edge length (176.2 against 109.1), visible here as the denser small features of each assembly.

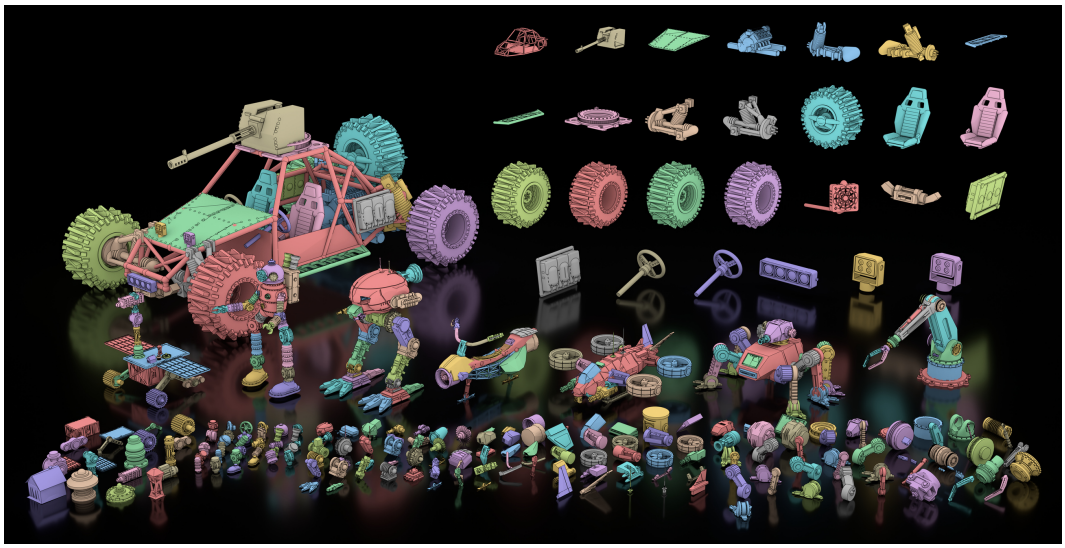


Figure 11. A shape written as code is natively part-structured. One named module per physical part, so the model decomposes into editable segments with no segmentation step. The assault buggy (left) is fanned out into its named parts, alongside other objects and a library of parts.



Figure 12. Texturing across the benchmark. Ten of the 36 benchmark objects, geometry and materials both authored by **Procedura**. Every image is a Blender-Cycles render of the shipped program, with per-part materials assigned to the named modules.

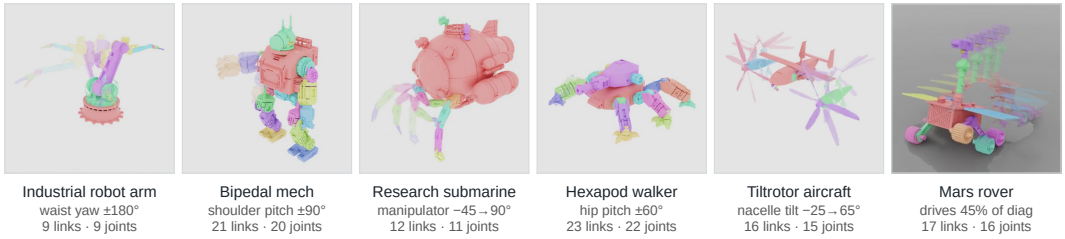


Figure 13. Articulation across the benchmark. Six objects, each planned, exported to OpenUSD and URDF, and validated headless in Isaac Sim. Every panel is composited from that object's own simulation clip, with the swept poses ghosted and the final pose solid. The first five hold one commanded joint traversing its range; the last shows the wheeled base driving on the ground plane. All six pass every dynamics phase, with each driven joint traversing $\geq 97\%$ of its commanded range, no non-adjacent links touching at rest, and a clean URDF re-import; five also clear the asset rules outright, and the tiltrotor trips only the advisory drive-gain rule.

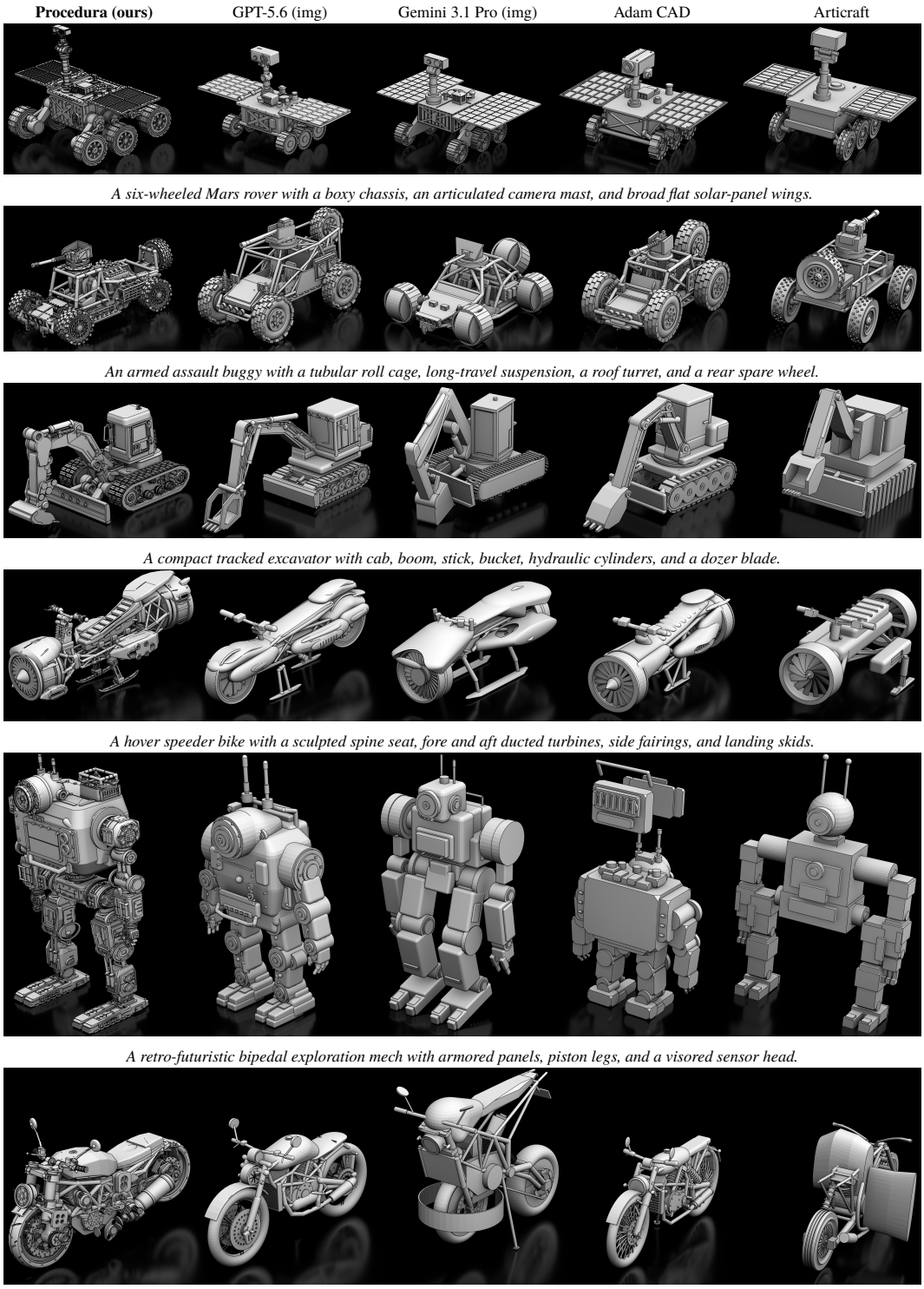


Figure 14. Full qualitative comparison against LLM-driven 3D generation (all six objects of Fig. 4). One object per row, each generated from the prompt printed beneath it; every method receives the same reference image and render rig. Prior agents recover the rough silhouette but collapse the mechanism into a handful of boxes, whereas **Procedura** resolves each part as its own solid.

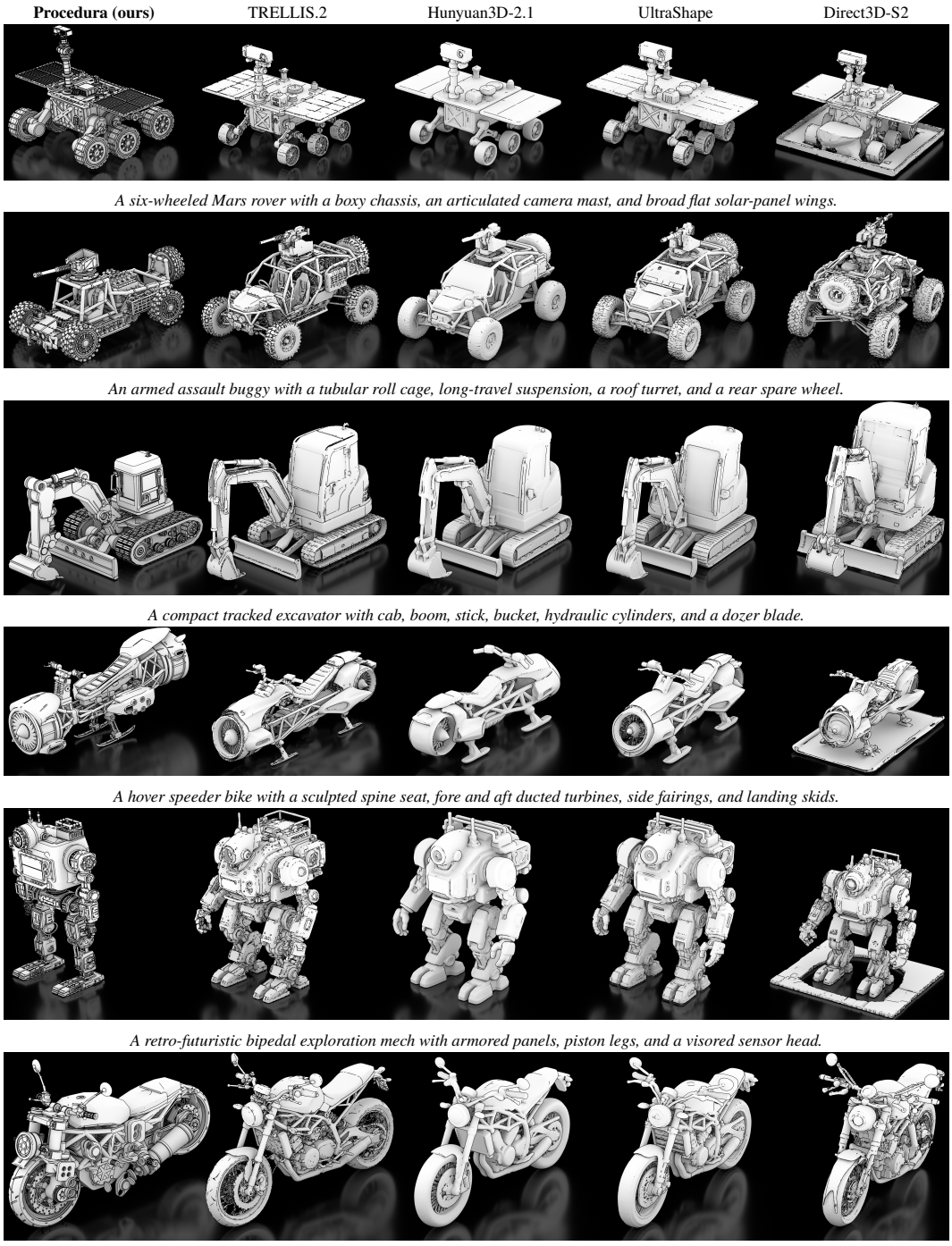
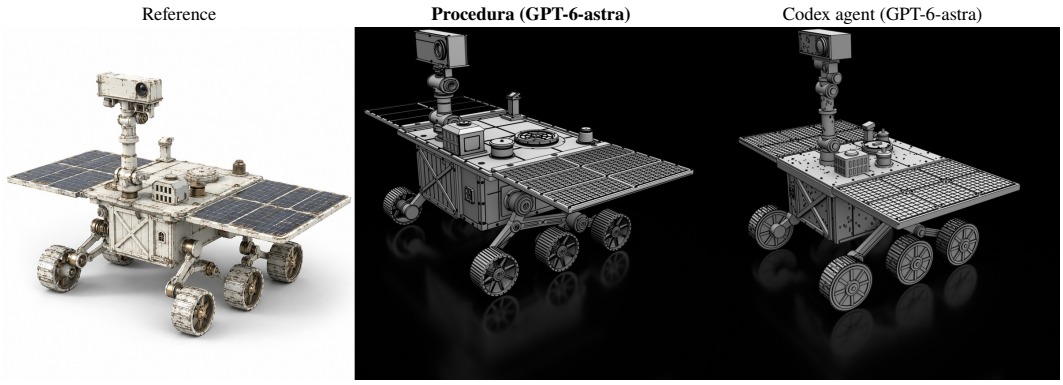
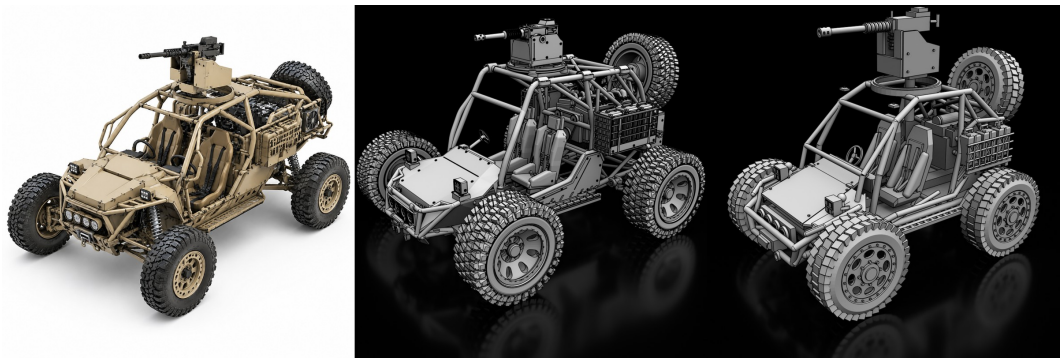


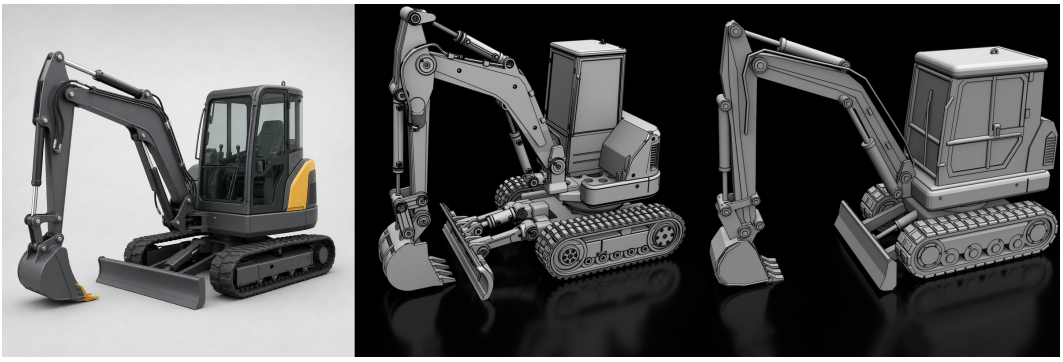
Figure 15. Full qualitative comparison against native 3D generators (all six objects of Fig. 5). The same six objects, one per row with its prompt printed beneath it, generated by four native 3D diffusion models and by **Procedura** (left column), all conditioned on the same locked reference image and re-rendered from the same camera and shading rig. The natives reproduce the silhouette faithfully, since they are trained end-to-end to do so, but their surfaces are soft, so bolt heads, vents, and panel lines fuse into one skin and no part boundary survives. **Procedura** compiles exact CSG solids, so edges stay sharp and every part remains separable.



A six-wheeled Mars rover with a boxy chassis, an articulated camera mast, and broad flat solar-panel wings.

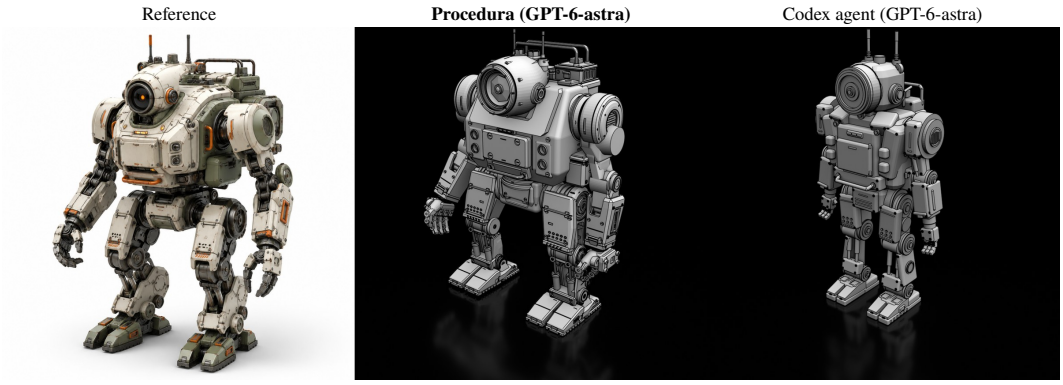


An armed assault buggy with a tubular roll cage, long-travel suspension, a roof turret, and a rear spare wheel.



A compact tracked excavator with cab, boom, stick, bucket, hydraulic cylinders, and a dozer blade.

Figure 16. Procedura versus the Codex agent, both on GPT-6-astra. Five of the objects of Fig. 14, one per row: the reference view both methods received (left), then each method's model under the shared grey-clay rig. Each model is turned to the heading of its reference before rendering; the heading was chosen by silhouette match against the reference and checked by eye. Continued on the next page.



A retro-futuristic bipedal exploration mech with armored panels, piston legs, and a visored sensor head.



A motorcycle rolling chassis with trellis frame, V-twin engine, fork, swingarm, spoked wheels, and exhaust.

Figure 16. Procedura versus the Codex agent, both on GPT-6-astra (continued).