Arbor: Explicit Geometric Conditioning for Controllable 3D Asset Generation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Text and image conditioned 3D models now generate convincing assets, but they still offer little direct control over the space an object should occupy or avoid. In authoring, this spatial intent is often known before generation starts. A chair should fit a seating envelope, a prop should leave clearance for motion, or a part should expose a contact surface. Prompts and image views are poor carriers for such constraints, requiring the need for an explicit control interface.

We present Arbor[1], a trainable attachment for text conditioned latent 3D generation. Arbor introduces constraint meshes as a native 3D control interface. The interface uses hull regions where geometry should exist, avoidance regions that should remain empty, and touch regions the object should contact. Unlike completion or whole object scaffold control, these meshes are not target evidence. They are local typed requirements and can include regions where no surface should appear. Arbor keeps this signal as geometry by converting constraint meshes into tokens and learning a routed attachment inside a frozen denoiser. Each latent region can therefore receive the part of the constraint that matters for its spatial location.

We evaluate Arbor on automatic and artist curated control benchmarks with hull, avoidance, and touch constraints, and compare the metric trends to a user preference study. Even without dedicated compliance losses, Arbor improves constraint obedience while preserving object quality and variation under fixed constraints.

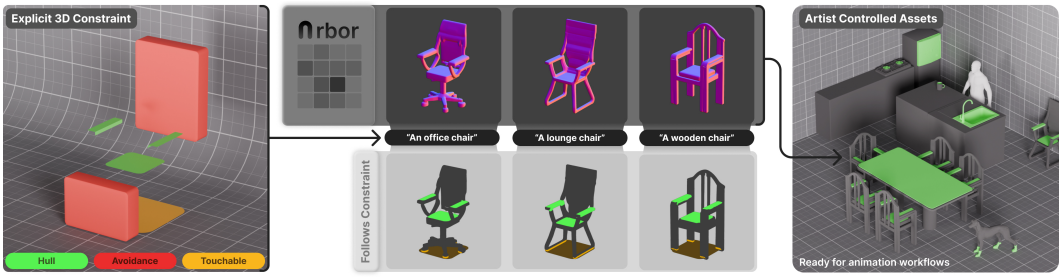

Figure 1: **Arbor overview.** Arbor turns simple 3D control objects into an explicit constraint signal for text-conditioned 3D generation. Hull regions mark where generated geometry should exist, touch regions mark contact patches, and avoidance regions mark free space that should remain empty. This enables artist to co-author the generation process, making asset generation more reliable and therefore more likely to be used in production.

[1]Named after an arched support trellis to guide plant growth.

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

1 Introduction

3D asset creation begins with an idea, but must often align with precise spatial requirements for the object to be production ready. A chair should fit a seating template. A handle should stay within reach. A kit bash part should expose a clean attachment region. A prop should leave room for motion, gameplay, or neighboring objects. For an artist, these requirements are easier to express with explicit geometry than with words, but current 3D generators do not yet offer a strong interface for this. The result is slot machine behavior: the artist tries many prompts, never lands on the exact constraint, and ends up doing manual cleanup.

3D generation from text prompts and from images has advanced quickly, from early optimization pipelines to modern feed forward and latent 3D priors [27, 13, 11, 37, 2, 36, 43, 33, 7], and now produces convincing geometry from text, images, or partial observations. What these methods do not provide is explicit author control over occupied and empty space when free volume starts to matter. Prompts describe semantics, not space. Image conditions are tied to viewpoint and appearance, making spatially absolute constraints inconvenient to encode. Arbor adds this missing interface, providing explicit volumetric control over 3D asset generation without interfering with the generator's variability in shape creation.

We introduce a novel control mechanism which tackles this attachment and interface problem. This is essential for game production as animations are often reused for multiple assets. Hence they require absolute accurate interaction surfaces or the animation will not interact with the object properly (clip through the surface, attach to the wrong place, etc.). While 3D control methods exist, they tackle different problems on editing [6, 14], generation under global structural priors [8, 31, 29], completion of partial geometry [3, 34, 25], or steering inference at sampling time [9]. For us the goal is to generate a full object from a text prompt while respecting a dense local geometric hull that marks semantically meaningful volumes for the object (handles, seats, wheels). Recent shape guidance methods focus mainly on regions where geometry should appear. Arbor also specifies regions that must stay empty (holes, gaps, clearance). We further show that the condition space can define custom constraints such as touch, which marks surfaces where the asset should make contact without extending to much past them (for example where legs meet the ground plane). Our constraint pipeline combines hull, avoidance, and touch signals as meshes that an artist prepares in a familiar modeling workflow before controlled generation, see Fig. 1.

Enforcing these constraints inside an existing 3D generator is not straightforward. Modern generators operate on a compressed 3D latent [36, 33, 43] while the constraint meshes are dense. Two sub-problems follow. We need to turn dense meshes into compact tokens that preserve both local detail and typed signals, and we need to inject those tokens into the latent regions where they matter. For encoding, Arbor reuses frozen geometric encoders and repurposes their material channels to carry the typed signals for hull, touch, and avoidance. For injection, a geometry router groups latent queries by region, retrieves the local constraint evidence relevant to each group, summarizes the full control object into a small set of global tokens, and injects both through lightweight residual branches into the frozen generator. Training this adapter alongside the frozen backbone on the original generation objective ties the constraint latents to the generator's own latent space to obtain semantically and geometrically correct objects.

In summary, Arbor lets artists steer generation by defining explicit constraint meshes the generator must obey, acting as an adapter that maps those constraints into the generator's latent space. We make three contributions.

- **Explicit geometric control interface.** We introduce a unified set of typed regions (hull, touch, avoidance) that users specify directly as 3D meshes.
- **Encoded geometry as condition.** We show that frozen geometric encoders can be repurposed to turn constraint meshes and typed signals into compact latent tokens that preserve local structure and serve as a model input.
- **Geometry router and adapter.** We introduce a router that assigns local constraint evidence to the latent regions where it matters, and a residual branch that injects this evidence into a frozen 3D generator.

We evaluate Arbor against the backbone without geometry and against baselines that steer during sampling. Because this setting has to assess both constraint adherence and generation quality, we introduce Ctrl Score and compare its trends to a user preference study.

2 Related work

3D generation has advanced along three practical lines. Optimization based methods lift text or image supervision into explicit 3D forms such as NeRFs or Gaussian splats [16, 24, 21, 27, 28]. They are flexible, but often slow and sensitive to the optimization objective. Reconstruction based methods infer 3D from one or a few images and are often used for generation by first producing an image and then recovering 3D [13, 30, 37, 2, 15, 7]. Native latent 3D models such as Direct3D [33], TRELLIS [36], and Hunyuan3D 2.0 [43] instead denoise compact 3D states directly. A related family of generators produces structured outputs such as parts, layouts, assemblies, or CAD primitives [12, 18, 39, 19, 42, 38, 20], where authoring happens by editing or completing the structured output after generation. Arbor instead brings the artist into the generation step itself, giving native latent generators the direct interface they currently lack.

ControlNet and Adapter methods showed that a pretrained text to image diffusion model can absorb new spatial conditions through a control branch trained alongside the frozen backbone [41, 22, 40]. The 3D version of the problem is harder, because a condition must remain meaningful across viewpoints, compressed latent states, and the final output representation. Arbor inherits the attached control intuition, but rebuilds it for native 3D generation where the condition is itself geometry rather than a 2D map.

Editing methods receive a complete asset or scene up front and modify it in a chosen region. SIGNeRF edits NeRF scenes with depth conditioned reference sheets [6]. Instant3dit and ObjFiller-3D inpaint 3D objects through multiview image edits before mapping the result back to 3D [1, 10], while Easy3E performs feed forward asset editing directly in a voxel flow [14]. Each method requires the user to bring the asset they want to modify. Arbor instead conditions a generator before any asset exists, with only typed local constraint regions on top of a text prompt.

Reconstruction methods condition on an input that already describes the asset's global structure. Several methods rely on a 3D structural prior. SK-Adapter encodes a skeleton into tokens injected into a frozen TRELLIS backbone [31]. Coin3D conditions on a coarse primitive proxy through a 3D adapter [8]. Others retrieve a 3D reference shape as a similarity prior [32]. Other methods rely on an image. SPAR3D conditions on an image plus an editable point cloud [15]. Hunyuan3D-Omni combines an image with several geometric modalities including points, boxes, voxels, and skeletons. It accepts partial observations but still requires the image to anchor the asset [29]. Each input fixes the asset's global structure ahead of generation, whether as a skeleton, a proxy, a reference, or an image. Arbor specifies only the parts that matter and leaves the rest to the prompt and the generator, allowing different assets to satisfy the same constraint set.

Inpainting and completion methods start from sparse or partial geometry and grow it into a finished asset. DiffComplete and Points-to-3D condition a 3D diffusion model on partial geometry and complete the rest around the input surface [3, 34], while Spice-E uses cross entity attention between a coarse guidance shape and the noisy sample [25]. SpaceControl steers a pretrained 3D denoiser during sampling without extra training, exposing a global tradeoff between constraint fidelity and generative variation [9]. None of these distinguish typed roles for the input geometry, treating it as a single signal to complete, match, or steer toward. Arbor uses geometry as a typed specification. Hull regions indicate where the asset should exist, touch regions mark contact surfaces, and avoidance regions are defined precisely by not becoming surface. In contrast to methods that prefill the latent or attend to a coarse proxy, Arbor guides denoising with explicit structure while the artist specifies volumes and signals up front.

3 Method

The main goal of Arbor is the creation of explicit geometry control methods for 3D generation. Given a text prompt y and a constraint object C, the goal is to sample a 3D asset $x \sim p(x \mid y, C)$.

3.1 Overview

Arbor builds on existing 3D generative models from the TRELLIS family. We use Trellis 1 (T1) [36] as this strong text-conditioned backbone for 3D geometry generation. To impose surface constraints, we leverage TRELLIS.2 (T2) [35], whose direct surface encoding mechanism provides a flexible

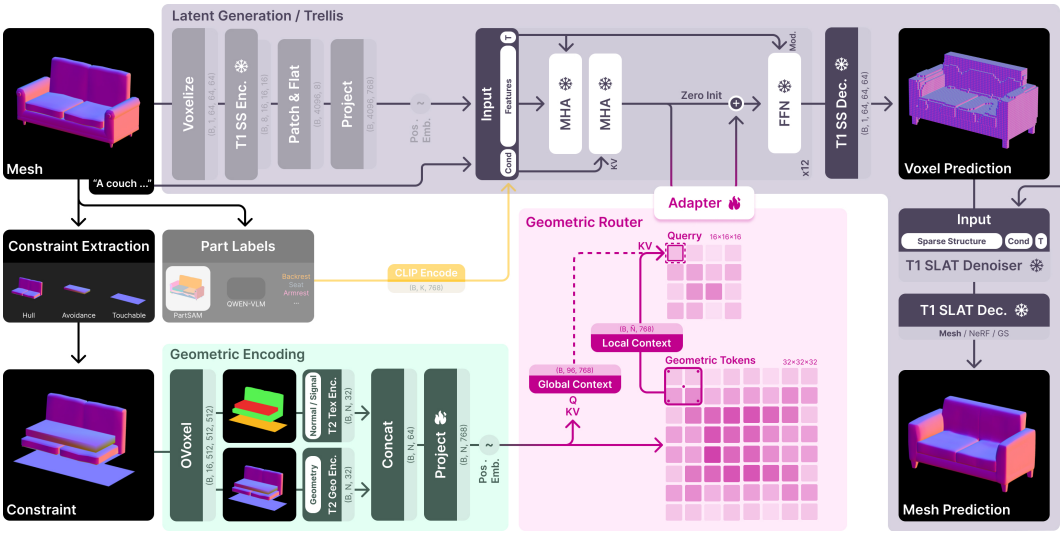

Figure 2: **Constraint conditioning pipeline.** Arbor converts a typed constraint object into TRELLIS.2 OVoxels, encodes geometry and signal attributes with frozen encoders (Sec. 3.2), aligns the resulting latents into geometry tokens, and routes those tokens into the TRELLIS sparse structure denoiser (Sec. 3.3). Local routing gives each query group the nearby constraint evidence it needs, while learned global summaries provide object scale context. (Part Labels and yellow path are for Arbor Semantics A.3)

interface for conditioning generation on explicit geometric signals. Both models follow a common compression generation design: raw 3D representations are first mapped to compact latent states, and then generation is performed in the compressed domain.

T2 also supports auxiliary per-surface channels, originally intended for material parameters used in rendering. We repurpose these channels to encode constraint information as binary flags, where each flag indicates the presence or absence of a specific surface constraint. For example, channel (c) 0-2 encode normals, c3 hull, c4 avoid, and c5 touch. We then inject them through a learned routed residual branch as shown in Fig. 2.

TRELLIS 1 - Text conditioned 3D generation T1 is a text-conditioned 3D generator that separates generation into a sparse-structure stage followed by latent refinement. Its sparse-structure stage operates on a 16^3 latent grid with 8 channels per lattice cell and is trained with flow matching. We use this stage as the prompt-conditioned backbone in Arbor, since it determines the coarse object structure and exposes a text-conditioned denoising process.

At each transformer block, the latent state is updated by self-attention, text cross-attention, and a feed-forward network. Omitting residual connections, normalization, and adaptive time modulation, one block can be written as

$$z_{\text{text}}^{(\ell)} = \text{CrossAttn}_{\text{text}}^{(\ell)}\Big(\text{SelfAttn}^{(\ell)}(z^{(\ell-1)}), y\Big), \qquad z^{(\ell)} = \text{FFN}^{(\ell)}\big(z_{\text{text}}^{(\ell)}\big). \tag{1}$$

Here, ℓ indexes the transformer layer, $z^{(\ell-1)}$ is the incoming sparse-structure state, $z_{\text{text}}^{(\ell)}$ is the state after cross-attention to the text prompt, and y denotes the encoded text context. The diffusion or flow time t enters through adaptive modulation, which is omitted from Eq. (1) for clarity.

During inference, T1 denoises an initial noise latent under the prompt conditioning y and time t. The resulting sparse structure is decoded into a 64^3 occupancy grid and passed to a second latent refinement stage. This refinement stage operates on active voxels and produces the final representation, which can be decoded as a mesh, radiance field, or 3D Gaussian representation.

TRELLIS 2 - Constraint Encoding T2 is designed around a direct sparse voxel representation, OVoxels, which jointly represents geometry and aligned surface attributes. While T2 is image-conditioned as a generator, Arbor uses only its frozen encoder stack. This makes T2 suitable as a constraint encoder without changing Arbor's prompt-driven generation setting.

Given a mesh M with aligned surface attributes A, T2 first converts the input into aligned shape and material fields on a 512^3 voxel grid:

$$\mathrm{OVoxelize}(M, A) = \big(\widetilde{M}_{\mathrm{shape}}, \widetilde{M}_{\mathrm{mat}}\big). \tag{2}$$

The frozen T2 shape and material encoders then map these fields to compact sparse latents,

$$z_{\mathrm{shape}} = E_{\mathrm{T2,shape}}(\widetilde{M}_{\mathrm{shape}}), \qquad z_{\mathrm{mat}} = E_{\mathrm{T2,mat}}(\widetilde{M}_{\mathrm{mat}}). \tag{3}$$

Each encoder reduces spatial resolution by a factor of 16 and produces sparse 32-dimensional tokens.

Arbor repurposes this T2 encoding path to represent geometric constraints. In particular, we encode constraint signals through the auxiliary attribute channels normally used for material parameters. These channels provide a direct, surface-aligned interface for injecting binary constraint flags, while keeping the T1 text-conditioned generation backbone unchanged.

3.2 Constraints

Constraints in Arbor are 3D meshes that specify desired spatial behavior. We use hull, avoidance, and touch constraints, and encode them into compact latent tokens for the text conditioned generator.

Types Hull constraints define regions where generated geometry should exist, e.g., the seat of a chair. Avoidance constraints define regions where generated geometry should not exist, e.g., empty space above the seat. Touch constraints define surfaces the object should contact, e.g., chair legs touching the ground. A.1 describes constraint creation. Here, we focus on how constraints are introduced into the model.

Encoding The natural first option would be to encode constraints with the native T1 encoder, since T1 is the generator we want to influence. This representation, however, is poorly suited to model interactive regional control. T1 encoding was built for training the latent space, not for interactive conditioning. A constraint would need to be voxelized, rendered from 150 views, and projected with DINOv2 [23] features. The process is slow, expects complete objects, and has no direct place for typed control signals. We therefore encode constraints with the frozen T2 stack instead. Arbor fuses the separate constraint meshes into a single mesh C_{mesh} with surface normals C_{normals} and signal channels C_{signal} that encode the constraint type. Applying Eq. 2 to this constraint object yields the OVoxel shape field $\widetilde{C}_{\mathrm{shape}}$ and aligned fields $\widetilde{C}_{\mathrm{normals}}$ and $\widetilde{C}_{\mathrm{signal}}$. The shape encoder receives $\widetilde{C}_{\mathrm{shape}}$, while the attribute encoder is repurposed. Instead of the original 6 material channels, it receives 3 voxel aligned normal channels $\widetilde{C}_{\mathrm{normals}}$ and 3 binary control channels for typed signals $\widetilde{C}_{\mathrm{signal}}$. Eq. 4 is the Arbor version of the T2 encoder call in Eq. 3.

$$c_{\mathrm{shape}} = E_{\mathrm{T2,shape}}\big(\widetilde{\boldsymbol{C}}_{\mathbf{shape}}\big), \qquad c_{\mathrm{signal}} = E_{\mathrm{T2,mat}}\big(\widetilde{\boldsymbol{C}}_{\mathbf{normals}}\,;\,\widetilde{\boldsymbol{C}}_{\mathbf{signal}}\big). \tag{4}$$

The semicolon denotes channel-wise concatenation. We concatenate both latent streams into the geometry memory $c_{\mathrm{geo}} = \big(c_{\mathrm{shape}}\ c_{\mathrm{signal}}\big)$. Each geometry token keeps its OVoxel position p_i. While this encoding is simple and effective, two issues remain. The tokens are not native to the T1 latent space, and they live on a finer 32^3 grid than the T1 16^3 state. We map every geometry token to the T1 model width with a learned projection and add a learned 3D positional embedding from p_i. We keep the notation c_{geo} for these prepared tokens below. The router handles the resolution mismatch by choosing which geometry tokens each local group of T1 queries receives.

3.3 Control

Generated geometry should obey the constraint, but still follow the text prompt. These goals can conflict. A model can improve hull overlap by overfilling the guide while losing prompt fit, structure, or variation. This is why sampling time steering [9], including our Gradient baseline (Appendix A.3), are an important but incomplete alternative. Arbor instead learns the attachment inside the generator. An adapter injects constraint tokens, and a router decides which tokens each latent region receives (Fig. 2, pink path). Each SS block carries one hidden query token for every cell of the 16^3 lattice. Arbor partitions this query lattice into 64 groups indexed by g, each covering a $4 \times 4 \times 4$ block of neighboring queries. For each group, the router builds a geometry context from the projected constraint tokens. The adapter uses the current T1 hidden states as queries, attends to this context, and writes the result back as a residual update.

Adapter Arbor injects geometry with a separate grounding branch inside each T1 block, after frozen text cross attention and before the feed forward (FFN) update. Eq. 5 modifies the T1 block from Eq. 1 by inserting a geometry residual before the original FFN. For the queries in group g at block ℓ, with router context $c_{\text{ctx}}^{(g)}$ defined below, we write

$$\Delta \mathbf{z}_{\text{geo}}^{(\ell,g)} = \text{CrossAttn}_{\text{geo}}^{(\ell)}\Big(\mathbf{z}_{\text{text}}^{(\ell,g)}, c_{\text{ctx}}^{(g)}\Big), \qquad \mathbf{z}^{(\ell,g)} = \text{FFN}^{(\ell)}\Big(\mathbf{z}_{\text{text}}^{(\ell,g)} + W_{\text{geo}}^{(\ell)} \Delta \mathbf{z}_{\text{geo}}^{(\ell,g)}\Big). \tag{5}$$

Here $\mathbf{z}_{\text{text}}^{(\ell,g)}$ denotes the subset of text conditioned hidden states in query group g, $\Delta \mathbf{z}_{\text{geo}}^{(\ell,g)}$ is the geometry update, and $W_{\text{geo}}^{(\ell)}$ is zero initialized. $\text{CrossAttn}_{\text{geo}}^{(\ell)}$ is one packed attention pass over local tokens and global summary tokens. This placement keeps the pretrained text path intact while allowing geometry to affect the state before the FFN. Empirically, later insertion, routing geometry through the text conditioning path, or unfreezing larger parts of the pretrained model led to weaker results. The remaining challenge is scale, since Arbor conditions on dense encoded geometry rather than the small global token set used by skeleton adapters such as SK-Adapter [31]. Attending from all T1 queries to all constraint tokens would be expensive and would require truncation.

Router The router makes dense geometry memory usable by the SS denoiser. Routing is separate from the position embeddings introduced above. The embedding marks where each geometry token lies, while routing chooses which tokens are read by each query group. Before denoising, Arbor prepares the projected geometry memory c_{geo}, token positions p_i, and the 64 fixed query groups, which depend only on the constraint object and the T1 lattice. At block ℓ, the original T1 stream supplies the attention queries, namely the text conditioned hidden states $\mathbf{z}_{\text{text}}^{(\ell,g)}$ in Eq. 5. The router supplies the keys and values by retrieving local geometry for each query group and appending a compact global context. All queries in one group share the context,

$$c_{\text{ctx}}^{(g)} = \big[c_{\text{local}}^{(g)}\,;\, c_{\text{global}}\big]. \tag{6}$$

For local routing, each group uses its center and eight corners as routing anchors R_g. We score every geometry token by its distance to the nearest anchor and keep the $K = 2048$ nearest tokens,

$$c_{\text{local}}^{(g)} = \text{TopK}_{2048}\Big(\{(c_{\text{geo}}^{(i)}, p_i)\}_{i=1}^{N}, \min_{r \in R_g} \|p_i - r\|_2\Big). \tag{7}$$

Here, p_i is the 3D position of geometry token $c_{\text{geo}}^{(i)}$, N is the number of geometry tokens, and r ranges over the anchors in R_g. This TopK step is not attention. It selects a bounded local memory for Eq. 5, keeping cost fixed while tying the context to the relevant constraint region. To retain object level information that local routing can miss, Arbor also summarizes the full geometry memory with 96 learned summary tokens,

$$c_{\text{global}} = \text{MLP}\Big(\text{CrossAttn}\big(S_{\text{global}}, \{c_{\text{geo}}^{(i)}\}_{i=1}^{N}\big)\Big), \tag{8}$$

where S_{global} is the learned summary set. These tokens capture broader signals such as extent, touch direction, and the relation between occupied and forbidden regions.

Training. We train only the geometry facing modules. These are the geometry projection and position embedding, the global summary modules, the semantic part token modules, and the grounding adapters. The T1 self attention, text cross attention, and feed forward weights remain frozen. Optimization uses the standard SS flow matching objective of the T1 backbone. No explicit constraint compliance loss is used in this run family. Text and geometry conditions are dropped independently for classifier free guidance. Training examples are built by sampling an object, constructing or selecting a typed constraint set, encoding it as above, and using the original T1 sparse structure latent as the flow matching target. Details, schedules, and implementation settings are deferred to A.1.

4 Evaluation

We evaluate Arbor as a control interface for text-conditioned 3D generation. The goal is not to copy a guide into the output, but to integrate constraint meshes while still producing a plausible object that follows the prompt. We test control over the same backbone without geometry, comparison to geometry guided baselines, and variation under fixed constraints.

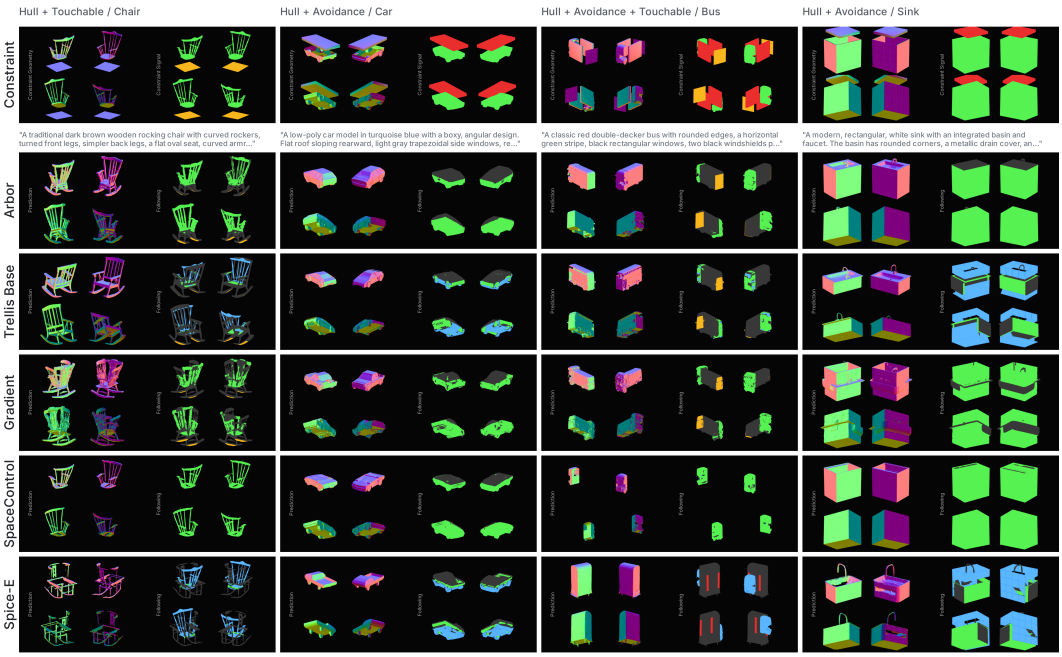

Figure 3: **Controlled generation comparison.** Each column shows one prompt and constraint object. The constraint is rendered as normal shaded geometry with signal regions colored **hull**, **touch**, and **avoidance**. Rows compare predictions and their constraint following. Here, green indicates a hull match and blue indicates missing hull. Arbor keeps readable objects while following local roles.

Models. Arbor is the model from Sec. 3. Both the TRELLIS [36] text generator and the TRELLIS.2 [35] OVoxel encoders stay frozen and only the Arbor geometry modules are trained. We also report Arbor Semantics, which adds stronger per query semantic text cues, and Arbor Compliance, which finetunes Arbor with explicit hull, avoidance, and touch losses (Appendix A.3). We compare against TRELLIS, our Gradient baseline at sampling time (Appendix A.3), SpaceControl [9], and Spice-E [25] for controlled generation. The variation track adds Point-E, SPAR3D [15], and Hunyuan3D-Omni [29], which receive image cues that the others do not.

Datasets. Arbor is trained on roughly 50k objects sampled from ABO [4], HSSD [17], and a Sketchfab subset of Objaverse-XL [5], which is about 10% of the training volume reported by TRELLIS [36]. Evaluation uses Toys4K [26], the dataset on which TRELLIS reports its official numbers. We construct two control benchmarks on Toys4K with hull, avoidance, and touch signals. The automatic split contains 128 procedurally generated constraints; the manual split contains 32 hand authored constraints that are not sampled by the training program (Appendix A.1).

Metrics. We report Hull Hit, Avoidance Violation (Avoid Viol.), Touch Hit, Volume Match (Vol. Match), multiview CLIP (MV-CLIP), and Control Score (Ctrl. Scr.), our new addition (Appendix A.5). All geometry terms use a shared 64^3 voxel grid, matching the sparse structure resolution used by the backbone. Ctrl. Scr. is a per sample harmonic mean over the terms where higher values are better (Hull Hit, Touch Hit, $1-$Avoid Viol., Vol. Match, MV-CLIP), so a method has to do well on all of them at once. Vol. Match is a coarse size guard that prevents overfilled outputs from looking artificially complete.

4.1 Controlled generation

Fig. 3 and Tab. 1 show the main result. TRELLIS keeps the object prior strong, but has no mechanism for the typed geometry and only matches the guide by chance. Gradient and SpaceControl move mass toward the control object, yet this often comes at the cost of noisy geometry, missing structure, or a collapsed shape. Spice-E can preserve recognizable form, but treats the guide as a shape signal and does not reliably separate hull, avoidance, and touch roles. Arbor keeps both requirements visible. The outputs remain readable assets, and the following views show local roles respected in the intended regions.

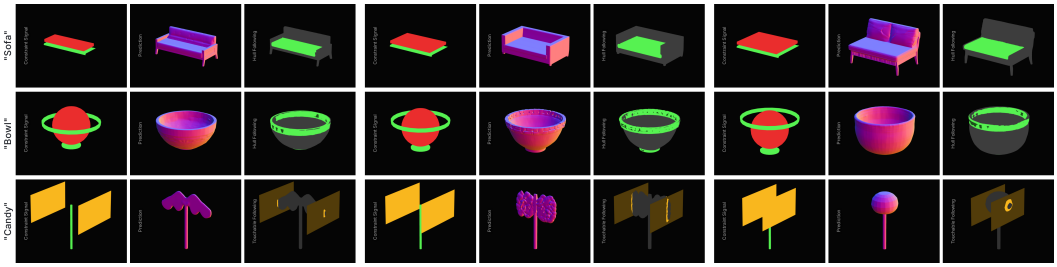

Figure 4: **Constraint sweeps.** The prompt is fixed and the constraint region is continuously moved, scaled, or rotated. Arbor follows the deformation without snapping to a small set of canonical layouts.

Table 1: **Controlled generation benchmark.** Manual (n=32) and automatic (n=128) splits. Pref. is the pairwise user study win rate over 404 trials from 27 participants, with the parenthesized value merging the three Arbor variants. Geometry metrics use 64^3 voxels and Ctrl. Scr. is defined in Appendix A.5.

Method	User Study	Manual (n=32)						Auto (n=128)					
	Pref. (%)↑	Ctrl. Scr.↑	Hull Hit↑	Avoid Viol.↓	Touch Hit↑	Vol. Match↑	MV-CLIP↑	Ctrl. Scr.↑	Hull Hit↑	Avoid Viol.↓	Touch Hit↑	Vol. Match↑	MV-CLIP↑
Arbor	**45.9 (59.2)**	**0.402**	0.714	0.025	0.857	0.469	0.229	**0.472**	0.786	0.006	0.984	0.487	0.240
Arbor Semantics	27.7	0.355	0.600	0.019	0.714	0.481	0.231	0.467	0.775	0.005	0.984	0.485	0.238
Arbor Compliance	29.9	0.401	0.772	0.015	0.857	0.441	0.233	0.466	0.802	0.003	0.968	0.481	0.240
Trellis	33.3	0.253	0.283	0.026	0.714	0.423	0.235	0.267	0.269	0.019	0.667	0.340	0.245
Gradient	18.2	0.347	0.690	0.046	0.571	0.471	0.230	0.436	0.775	0.019	0.833	0.474	0.236
SpaceControl	9.6	0.151	0.823	0.000	0.000	0.188	0.220	0.214	0.729	0.001	0.492	0.166	0.234
Spice-E	5.2	0.151	0.108	0.085	0.286	0.469	0.228	0.227	0.196	0.031	0.508	0.465	0.239

The table mirrors this qualitative behavior. Arbor and Compliance are essentially tied on the manual split, and all Arbor variants separate clearly from the non-Arbor baselines. The Arbor family wins 59.2% of pairwise user choices in our 27 participant, 404 trial study, the single Arbor row is preferred most often before merging variants. TRELLIS is the next strongest human baseline because it produces clean objects, even though it misses many constraints. For this reason, Ctrl. Scr. combines control adherence, volume agreement, and MV-CLIP instead of reporting geometry overlap alone.

4.2 Variation under fixed constraints

The prompt steerable baselines above either miss the control object or degrade the sample. We therefore also compare to recent reconstruction models with explicit 3D conditioning. These methods solve a different task because they receive an image input, but test whether stronger visual evidence is enough to combine control and variation. We fix one control object and vary the seed, while keeping the image input fixed for Point-E, SPAR3D, and Hunyuan3D-Omni. In Fig. 5, Arbor changes the truck back and tires while respecting hull and avoidance, and it builds new sofa cushions and surrounding frame geometry around the controlled seating area. Point-E reaches similar raw variation, but the following views show that much of this variation is unstable and leaves the hull. SPAR3D and Hunyuan3D-Omni stay closer to the image and therefore vary less. Tab. 2 confirms the tradeoff. Arbor reaches the highest variation and Ctrl. Scr. even without an image anchor, while image bound models either lose control or lose generative range.

4.3 Ablations

Routing. We probe the conditioning path at a fixed checkpoint to identify which signal Arbor actually uses. Tab. 3 reports constraint IoU retained relative to the full model. Local routing alone keeps 84.3% of the constraint IoU, showing that routed local evidence is the main mechanism. The gap to the full model shows that global summaries still add useful object level context, while global summaries alone are not sufficient. Removing the signal stream is most damaging because the model still sees geometry but no longer knows which regions should be filled, touched, or avoided. The shape and normal rows show that Arbor uses more than token location. Encoded shape carries substantial information, and normals provide a smaller but still visible gain.

Sweeps. Fig. 4 tests whether this control stays smooth outside the discrete benchmark. The prompt is fixed while a single hull region moves through position, scale, and orientation. The generated asset follows the moving constraint without snapping to a small set of layouts or losing object identity. The

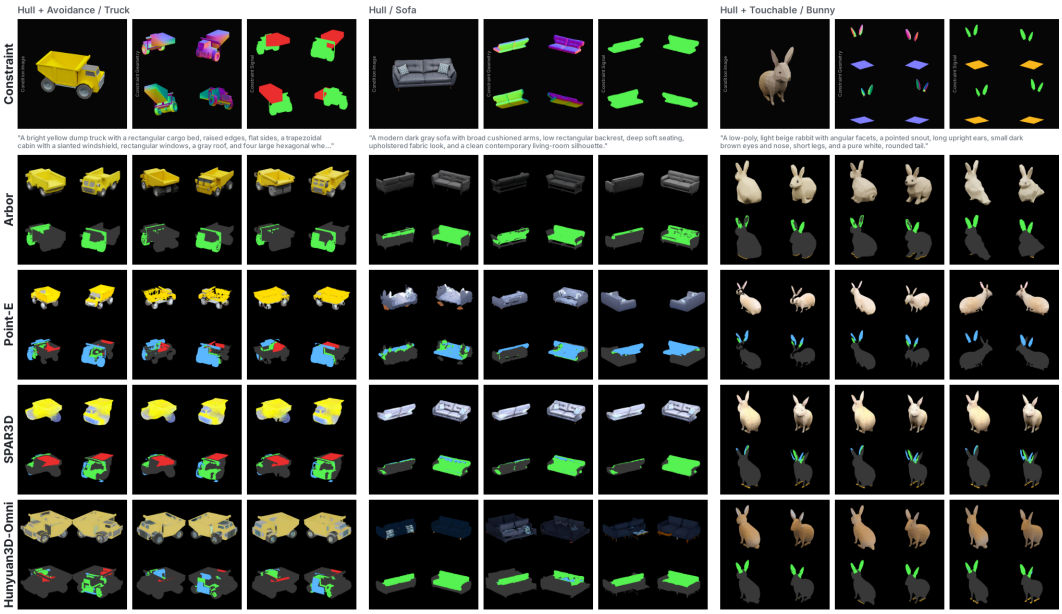

Figure 5: **Variation under a fixed constraint.** Each block keeps the hull fixed and varies the seed; image conditioned baselines also fix the input image. Arbor changes details and proportions across seeds while still satisfying the constraint, where the image anchored methods stay close to their input.

Method	Var.↑	Ctrl. Scr.↑	Hull Hit↑	Avoid Viol.↓	Vol. Match↑	MV-CLIP↑
Arbor	0.740	**0.361 ± 0.010**	0.707 ± 0.044	0.016 ± 0.010	0.460 ± 0.024	0.229 ± 0.002
Point-E	0.731	0.105 ± 0.009	0.100 ± 0.014	0.089 ± 0.015	0.364 ± 0.023	0.224 ± 0.002
SPAR3D	0.141	0.162 ± 0.004	0.182 ± 0.001	0.198 ± 0.003	0.565 ± 0.002	0.224 ± 0.000
Hunyuan3D-Omni	0.526	0.318 ± 0.034	0.533 ± 0.028	0.040 ± 0.027	0.534 ± 0.023	0.230 ± 0.001

Table 2: **Variation under a fixed constraint.** Var. is mean pairwise 1−IoU across three seeds; other columns report mean ± standard deviation. Arbor keeps high variation while preserving control.

Ablation type	IoU retained↑
Only local routing	84.3%
Only global summary	27.0%
No shape stream	54.9%
No signal stream	39.2%
No normals	83.8%

Table 3: **Conditioning ablation.** Constraint IoU retained relative to full Arbor.

sweep grid also isolates hull, touch, and avoidance controls, showing that Arbor reacts to each signal rather than only to a single combined constraint. We include additional sheets and video sweeps in the supplemental material.

5 Conclusion

Arbor introduces an explicit geometric conditioning interface for a frozen text conditioned 3D generator. Constraint meshes are turned into compact tokens by frozen 3D encoders and injected through a routed residual branch into the denoising blocks, so that hull, touch, and avoidance regions become part of the generator's input rather than only a sampling time correction. On our Toys4K benchmarks, Arbor improves constraint adherence over the backbone without geometry, sampling time baselines, and trained baselines, while preserving the variation of the underlying prior under fixed constraints.

The current system still exposes two limits. First, constraint regions carry geometry and typed signals, but not full semantic function. A seat region gives the generator a volume and an orientation, but not a guarantee that the volume will be used as a seat when the prompt conflicts with the constraint. Our semantic variant did not yet outperform the routed geometry path itself. Second, Arbor acts only at the sparse structure stage and does not directly control later refinement, where surface detail and material attributes enter. These limits point to the next step: richer part labels and the same routed attachment extended to later stages, so that structure, detail, and material can follow one explicit geometric specification. Further limitations are recorded in A.6.

References

[1] Amir Barda, Matheus Gadelha, Vladimir G. Kim, Noam Aigerman, Amit H. Bermano, and Thibault Groueix. Instant3dit: Multiview inpainting for fast editing of 3D objects. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16273–16282, 2025. URL https://arxiv.org/abs/2412.00518.

[2] Mark Boss, Zixuan Huang, Aaryaman Vasishta, and Varun Jampani. SF3D: Stable fast 3D mesh reconstruction with UV-unwrapping and illumination disentanglement. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16240–16250, 2025. URL https://arxiv.org/abs/2408.00653.

[3] Ruihang Chu, Enze Xie, Shentong Mo, Zhenguo Li, Matthias Nießner, Chi-Wing Fu, and Jiaya Jia. DiffComplete: Diffusion-based generative 3D shape completion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2306.16329.

[4] Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F. Yago Vicente, Thomas Dideriksen, Himanshu Arora, Matthieu Guillaumin, and Jitendra Malik. ABO: Dataset and benchmarks for real-world 3D object understanding. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. URL https://arxiv.org/abs/2110.06199.

[5] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. Objaverse-XL: A universe of 10M+ 3D objects. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2307.05663.

[6] Jan-Niklas Dihlmann, Andreas Engelhardt, and Hendrik Lensch. SIGNeRF: Scene integrated generation for neural radiance fields. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. doi: 10.1109/CVPR52733.2024.00638. URL https://doi.org/10.1109/CVPR52733.2024.00638.

[7] Jan-Niklas Dihlmann, Mark Boss, Simon Donne, Andreas Engelhardt, Hendrik P. A. Lensch, and Varun Jampani. ReLi3D: Relightable multi-view 3D reconstruction with disentangled illumination. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=BlSKgQb3Vd.

[8] Wenqi Dong, Bangbang Yang, Lin Ma, Xiao Liu, Liyuan Cui, Hujun Bao, Yuewen Ma, and Zhaopeng Cui. Coin3D: Controllable and interactive 3D assets generation with proxy-guided conditioning. In *ACM SIGGRAPH*, 2024. doi: 10.1145/3641519.3657425. URL https://doi.org/10.1145/3641519.3657425.

[9] Elisabetta Fedele, Francis Engelmann, Ian Huang, Or Litany, Marc Pollefeys, and Leonidas Guibas. SpaceControl: Introducing test-time spatial control to 3D generative modeling. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=mEqsCVI5sN.

[10] Haitang Feng, Jie Liu, Jie Tang, Gangshan Wu, Beiqi Chen, Jianhuang Lai, and Guangcong Wang. ObjFiller-3D: Consistent multi-view 3D inpainting via video diffusion models. *arXiv preprint*, 2025. URL https://arxiv.org/abs/2508.18271.

[11] Zexin He and Tengfei Wang. OpenLRM: Open-source large reconstruction models. https://github.com/3DTopia/OpenLRM, 2023. URL https://github.com/3DTopia/OpenLRM. GitHub repository; open-source implementation of LRM, not a primary paper.

[12] Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. SPAGHETTI. *ACM Transactions on Graphics (TOG)*, 2022. doi: 10.1145/3528223.3530084. URL https://doi.org/10.1145/3528223.3530084.

[13] 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. URL https://openreview.net/forum?id=sllU8vvsFF.

[14] Shimin Hu, Yuanyi Wei, Fei Zha, Yudong Guo, and Juyong Zhang. Easy3E: Feed-forward 3D asset editing via rectified voxel flow. *arXiv preprint*, 2026. URL https://arxiv.org/abs/2602.21499. CVPR 2026.

[15] Zixuan Huang, Mark Boss, Aaryaman Vasishta, James Matthew Rehg, and Varun Jampani. SPAR3D: Stable point-aware reconstruction of 3D objects from single images. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16860–16870, 2025. URL https://arxiv.org/abs/2501.04689.

[16] Ajay Jain, Ben Mildenhall, Jonathan T. Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 857–866, 2022. doi: 10.1109/CVPR52688.2022.00094. URL https://doi.org/10.1109/CVPR52688.2022.00094.

[17] Mukul Khanna, Yongsen Mao, Hanxiao Jiang, Sanjay Haresh, Brennan Schacklett, Dhruv Batra, Alexander Clegg, Eric Undersander, Angel X. Chang, and Manolis Savva. Habitat synthetic scenes dataset (HSSD-200): An analysis of 3D scene scale and realism tradeoffs for ObjectGoal navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024. URL https://arxiv.org/abs/2306.11290.

[18] Juil Koo, Seungwoo Yoo, Minh Hieu Nguyen, and Minhyuk Sung. SALAD: Part-level latent diffusion for 3D shape generation and manipulation. In *International Conference on Computer Vision (ICCV)*, 2023. URL https://arxiv.org/abs/2303.12236.

[19] Juil Koo, Wei-Tung Lin, Chanho Park, Chanhyeok Park, and Minhyuk Sung. BoxSplitGen: A generative model for 3D part bounding boxes in varying granularity. In *Winter Conference on Applications of Computer Vision (WACV)*, pages 1777–1787, 2026. URL https://arxiv.org/abs/2602.20666.

[20] Mingi Lee, Dongsu Zhang, Clément Jambon, and Young Min Kim. BrepDiff: Single-stage b-rep diffusion model. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, SIGGRAPH Conference Papers '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715402. doi: 10.1145/3721238.3730698. URL https://doi.org/10.1145/3721238.3730698.

[21] 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 *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 300–309, 2023. doi: 10.1109/CVPR52729.2023.00037. URL https://doi.org/10.1109/CVPR52729.2023.00037.

[22] Chong Mou, Xintao Wang, Liangbin Xie, Yanze Wu, Jian Zhang, Zhongang Qi, and Ying Shan. T2I-adapter: Learning adapters to dig out more controllable ability for text-to-image diffusion models. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 4296–4304, 2024. doi: 10.1609/AAAI.V38I5.28226. URL https://doi.org/10.1609/AAAI.V38I5.28226.

[23] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research (TMLR)*, 2024. URL https://openreview.net/forum?id=a68SUt6zFt.

[24] 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. URL https://openreview.net/forum?id=FjNys5c7VyY.

[25] Etai Sella, Gal Fiebelman, Noam Atia, and Hadar Averbuch-Elor. Spice·e: Structural priors in 3D diffusion using cross-entity attention. In *ACM SIGGRAPH*, pages 1–11, 2024. URL https://arxiv.org/abs/2311.17834.

[26] Stefan Stojanov, Anh Thai, and James M. Rehg. Using shape to categorize: Low-shot learning with an iterative categorization-discrimination loop. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. URL https://arxiv.org/abs/2104.07371.

[27] Jingxiang Sun, Bo Zhang, Ruizhi Shao, Lizhen Wang, Wen Liu, Zhenda Xie, and Yebin Liu. DreamCraft3D: Hierarchical 3D generation with bootstrapped diffusion prior. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=DDX1u29Gqr.

[28] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. DreamGaussian: Generative gaussian splatting for efficient 3D content creation. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=UyNXMqnN3c.

[29] Team Hunyuan3D. Hunyuan3D-omni: A unified framework for controllable generation of 3D assets. *arXiv preprint*, 2025. URL https://arxiv.org/abs/2509.21245.

[30] Dmitry Tochilkin, David Pankratz, Zexiang Liu, Zixuan Huang, Adam Letts, Yangguang Li, Ding Liang, Christian Laforte, Varun Jampani, and Yan-Pei Cao. TripoSR: Fast 3D object reconstruction from a single image. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2403.02151.

[31] Anbang Wang, Yuzhuo Ao, Shangzhe Wu, and Chi-Keung Tang. SK-adapter: Skeleton-based structural control for native 3D generation. *arXiv preprint*, 2026. URL https://arxiv.org/abs/2603.14152.

[32] Zhenwei Wang, Tengfei Wang, Zexin He, Gerhard Hancke, Ziwei Liu, and Rynson W. H. Lau. Phidias: A generative model for creating 3D content from text, image, and 3D conditions with reference-augmented diffusion. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/50ca96a1a9ebe0b5e5688a504feb6107-Abstract-Conference.html.

[33] 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)*, volume 37, pages 121859–121881, 2024. doi: 10.52202/079017-3873. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/dc970c91c0a82c6e4cb3c4af7bff5388-Abstract-Conference.html.

[34] Jiatong Xia, Zicheng Duan, Anton van den Hengel, and Lingqiao Liu. Points-to-3D: Structure-aware 3D generation with point cloud priors. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. URL https://jiatongxia.github.io/points2-3D/.

[35] 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. *arXiv preprint*, 2025. URL https://arxiv.org/abs/2512.14692.

[36] 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 *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21469–21480, 2025. URL https://arxiv.org/abs/2412.01506.

[37] Jiale Xu, Weihao Cheng, Yiming Gao, Xintao Wang, Shenghua Gao, and Ying Shan. InstantMesh: Efficient 3D mesh generation from a single image with sparse-view large reconstruction models. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2404.07191.

[38] Xiang Xu, Joseph G. Lambourne, Pradeep Kumar Jayaraman, Zhengqing Wang, Karl D.D. Willis, and Yasutaka Furukawa. BrepGen: A b-rep generative diffusion model with structured latent geometry. *ACM Transactions on Graphics (TOG)*, 43(4):1–14, 2024. doi: 10.1145/3658129. URL https://brepgen.github.io/.

[39] Yunhan Yang, Yufan Zhou, Yuan-Chen Guo, Zi-Xin Zou, Yukun Huang, Ying-Tian Liu, Hao Xu, Ding Liang, Yan-Pei Cao, and Xihui Liu. OmniPart: Part-aware 3D generation with semantic decoupling and structural cohesion. *arXiv preprint*, 2025. URL https://arxiv.org/abs/2507.06165. SIGGRAPH Asia 2025.

[40] Hu Ye, Jun Zhang, Sibo Liu, Xiao Han, and Wei Yang. IP-adapter: Text compatible image prompt adapter for text-to-image diffusion models. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2308.06721.

[41] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *International Conference on Computer Vision (ICCV)*, 2023. URL https://arxiv.org/abs/2302.05543.

[42] Wang Zhao, Yan-Pei Cao, Jiale Xu, Yuejiang Dong, and Ying Shan. Assembler: Scalable 3D part assembly via anchor point diffusion. *arXiv preprint*, 2025. URL https://arxiv.org/abs/2506.17074. SIGGRAPH Asia 2025.

[43] Zibo Zhao, Zeqiang Lai, Qingxiang Lin, Yunfei Zhao, Haolin Liu, Shuhui Yang, Yifei Feng, Mingxin Yang, Sheng Zhang, Xianghui Yang, Huiwen Shi, Sicong Liu, Junta Wu, Yihang Lian, Fan Yang, Ruining Tang, Zebin He, Xinzhou Wang, Jian Liu, Xuhui Zuo, Zhuo Chen, Biwen Lei, Haohan Weng, Jing Xu, Yiling Zhu, Xinhai Liu, Lixin Xu, Changrong Hu, Shaoxiong Yang, Song Zhang, Yang Liu, Tianyu Huang, Lifu Wang, Jihong Zhang, Meng Chen, Liang Dong, Yiwen Jia, Yulin Cai, Jiaao Yu, Yixuan Tang, Hao Zhang, Zheng Ye, Peng He, Runzhou Wu, Chao Zhang, Yonghao Tan, Jie Xiao, Yangyu Tao, Jianchen Zhu, Jinbao Xue, Kai Liu, Chongqing Zhao, Xinming Wu, Zhichao Hu, Lei Qin, Jianbing Peng, Zhan Li, Minghui Chen, Xipeng Zhang, Lin Niu, Paige Wang, Yingkai Wang, Haozhao Kuang, Zhongyi Fan, Xu Zheng, Weihao Zhuang, YingPing He, Tian Liu, Yong Yang, Di Wang, Yuhong Liu, Jie Jiang, Jingwei Huang, and Chunchao Guo. Hunyuan3D 2.0: Scaling diffusion models for high resolution textured 3D assets generation. *arXiv preprint*, 2025. URL `https://arxiv.org/abs/2501.12202`.

[44] Zhe Zhu, Le Wan, Rui Xu, Yiheng Zhang, Honghua Chen, Zhiyang Dou, Cheng Lin, Yuan Liu, and Mingqiang Wei. PartSAM: A scalable promptable part segmentation model trained on native 3D data. *arXiv preprint*, 2026. URL `https://arxiv.org/abs/2509.21965`. ICLR 2026.

A Supplementary Material

This supplement adds the details that support the experimental claims but would interrupt the flow of the main paper. Section A.1 covers data creation, typed constraint synthesis, PartSAM preprocessing, and semantic annotation generation. Section A.2 adds qualitative coverage beyond the main figures. Section A.3 summarizes the secondary Arbor variants and the internal design choices that did not become the final method. Section A.4 records the paper configuration. Section A.5 defines the evaluator protocol and the metrics used in the review.

A.1 Data Creation

As mentioned in Sec. 4, Arbor is trained on about 50k objects from ABO [4], HSSD [17], and Objaverse XL [5]. This is a large corpus, but still much smaller than the pretraining scale of TRELLIS [36]. During development we also saw similar behavior when training only on ABO and HSSD, which reduces the corpus to roughly 10k objects. In this section we describe constraint creation, evaluation data, part segmentation, and semantic extraction. Table 4 gives a compact overview of the datasets and their role in training.

Table 4: **Dataset Overview:** Distribution of data used and their corresponding role in training.

	ABO [4]	HSSD [17]	Objaverse XL [5]	Toys4K [26]
Role	train	train	train	benchmark / eval
Segmented count	4,484	6,660	39,846	3,225

Constraint creation. While the main paper describes constraints as artist authored meshes, there is a practical gap: there is no dataset of hull, avoidance, and touch meshes that can be used directly for training. We therefore spent a large part of the project on an automatic constraint creation system. We call this system the orchestra of constraints. It is used online during data loading, so every training batch can sample fresh control geometry. For evaluation, we freeze the exported constraints and store them in benchmark manifests.

The system is organized into three families shown in Fig. 6: hull in green, avoidance in red, and touch in yellow. Each family contains several fast geometry samplers. For hull constraints, one important option is Part Union Hull. Here we use the offline PartSAM [44] segmentation, pick one or more semantic parts, and merge them into a positive region. This gives meaningful hulls such as chair seats, back rests, or lamp heads. Another option is a random surface patch, where we sample connected triangles directly on the mesh and convert them into a positive control region. This produces less semantic but more diverse support geometry. We also use section based hulls and simple proxy shapes when we want broad geometric coverage.

Avoidance constraints are sampled differently because they describe free space. One common case is a layout blocker, where we place a forbidden region in a location that should stay empty, for example above a seat or inside the opening of a shelf. Another case is a surface clearance region, where we offset a region away from the mesh and ask the model not to generate into that space. These negative constraints are important because they force Arbor to reason not only about where geometry should exist, but also where it should not.

Touch constraints couple a small support region with a forbidden half space behind it. The support region marks where contact should happen, while the forbidden side prevents the model from simply filling both sides of the plane. This is useful for object placement such as feet on the ground or attachment points on walls and support surfaces.

The most useful training rows are often combinations. For example, we may place an avoidance region above a hull region, or sample a touch region together with a nearby hull patch. This is what turns the constraint object into a more realistic authoring signal instead of a single isolated mask. Figure 6 shows the full family set. The exact sampling balance of the paper model is given in Appendix A.4.

Evaluation data. As mentioned in Sec. 4, we evaluate on Toys4K [26], the same object set used by TRELLIS [36]. This gives a fair base corpus with broad category coverage and existing prompts.

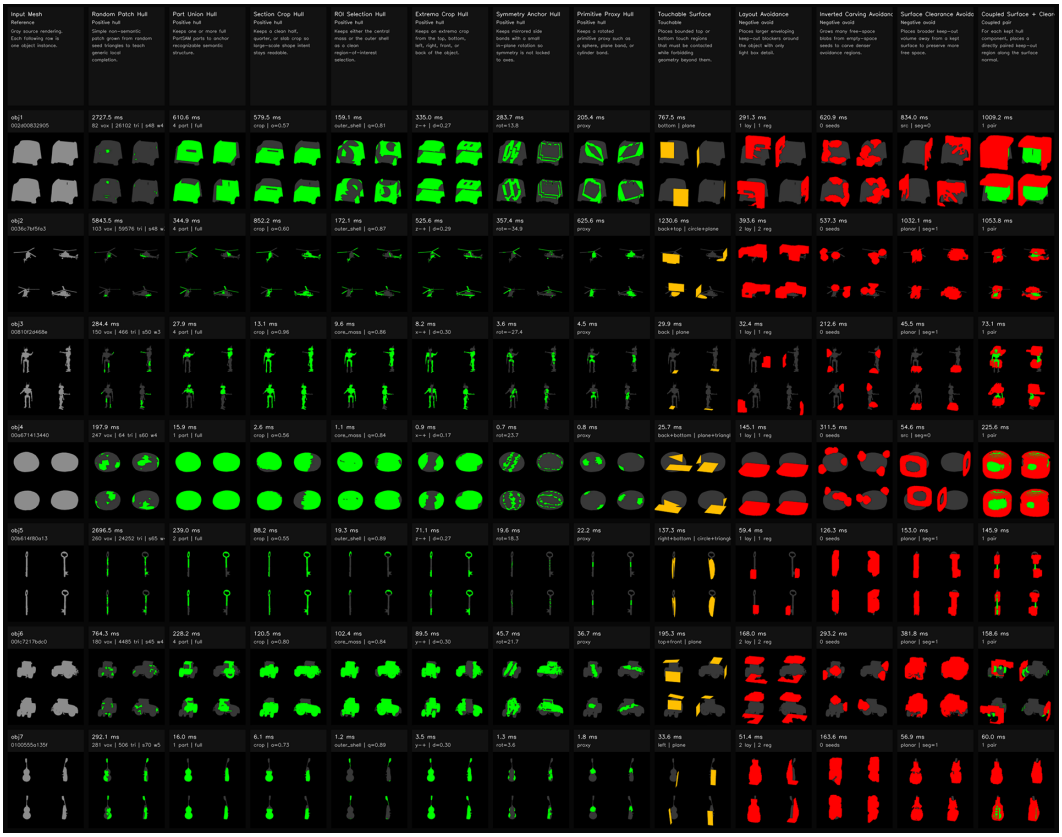

Figure 6: **Automatic constraint families used by Arbor.** The figure shows the concrete generators that make up Arbor's typed constraint program. Green columns are positive hull families, yellow columns are touch/contact families, and red columns are avoidance families. Each column lists the family intent at the top and example outputs on several objects below. These families are sampled online during training, while benchmark manifests freeze their exported meshes and metadata for evaluation.

However, Toys4K does not provide hull, avoidance, or touch constraints. We therefore had to build our own benchmark.

We created two benchmark splits. The first is an automatic split, which samples 128 rows from the same constraint families that we also use during training. We call this split auto. The second is a manual split with 32 rows. Here we tried to work with artist intent instead of the training program. We loaded reference meshes in Blender, removed parts, added support regions, or blocked space in the way a human author would reason about the object. This is more time consuming, but it is also more important, because it gives us a cleaner out of distribution benchmark.

For the qualitative figures in the main paper we only use the manual split. It is the strongest test of whether Arbor can follow human intent rather than only replaying the program it saw during training. In the appendix we also include examples from the automatic split to show broader coverage. We keep the split sizes fixed because several baselines require more than ten minutes per object, so larger suites would make the comparison depend more on compute budget than on method behavior. We plan to release the benchmark manifests and typed control meshes where redistribution is allowed, so that later work can compare against the same control setup.

Part segmentation. As mentioned above, we use PartSAM [44] to extract semantically meaningful object parts. This helps us sample better hull regions and later connect geometry to semantic labels. Part segmentation is a difficult problem on its own, so we treat it as offline preprocessing. The practical benefit is that the expensive step only happens once per object.

PartSAM is a feed forward method. In our pipeline the mesh is encoded once, then many part masks are predicted from cached features instead of re encoding the mesh for every part query. Optional

graph cut refinement improves difficult boundaries, and the final labels are written back to the original mesh. In practice this takes about 12 seconds per object and is therefore feasible at our scale. We use these segments both for constraint selection and for semantic extraction.

Semantic extraction. As already mentioned in the limitations and future work of the main paper, we believe that semantic labels attached to hull parts are one of the most promising next steps for constrained control. This is why we also report Arbor Semantics in Appendix A.3. Since there is no public 3D dataset with both part segments and usable part labels, we built this data ourselves.

The final production path stays fully offline and uses a local Qwen3-VL image prompting workflow. For each object we keep the raw PartSAM RGB regions, load four whole-object context renders, and select a subset of visible regions that covers most of the segmented area. The first Qwen3-VL prompt receives the captions together with the context views and proposes a compact object-specific vocabulary of candidate part labels, usually about 10–24 short noun phrases with a few aliases.

We then issue a second prompt over lettered region cards. Each card shows one raw PartSAM region highlighted in up to two views, while the context renders stay visible on the same page. The prompt forces the VLM to choose exactly one label from the candidate vocabulary or return null, rather than inventing a free-form phrase. This constrained vocabulary step matters in practice: it keeps the label set stable across objects and reduces the generic responses that appear when the model is asked to name parts without a shared candidate list. If a page fails to parse or the confidence stays low, we re-query that region with a smaller single-region prompt instead of accepting the ambiguous answer.

The stored annotation for each confident region includes the selected label, short variants, confidence, a brief visual-evidence note, supporting views, and the original region geometry statistics. Weak or ambiguous regions remain unlabeled. We write the result as an offline sidecar JSON together with resumable CSV and diagnostics files, so the semantic branch stays out of the training-time dataloader.

These labels are the basis of Arbor Semantics, where labeled queries receive an additional semantic text signal.

A.2 Extended Results

We provide more qualitative results for Arbor in Fig. 7. While the main paper highlights the manual benchmark, this figure adds more rows from both the manual and automatic Toys4K splits. One can see that Arbor generalizes across different object categories and across different mixes of hull, avoidance, and touch constraints.

We also provide more sweep states in the supplementary media. These are easier to judge in motion than in still frames, because one can directly see whether the object moves smoothly with the constraint or starts to collapse.

A.3 Arbor Variants

In the main paper we chose Arbor as the final method. The variants in this section explain what we tried next, what each change was supposed to solve, and why Arbor remained the strongest overall choice. They are useful because they show which parts of the control problem are still open.

Compliance. Arbor Compliance starts from a trained Arbor checkpoint and adds explicit decoded constraint losses during finetuning. The idea was simple: if Arbor already knows how to use the constraint, perhaps a small amount of direct pressure on the decoded occupancy could improve difficult cases such as weak hulls or small touch regions.

We decode the geo conditioned sparse structure prediction $\hat{z}_{\mathrm{geo}}$ and evaluate it against the hull, avoidance, and touch targets. Using the notation from Sec. 3, the added objective is

$$\mathcal{L}_{\mathrm{comp}} = \mathcal{L}_{\mathrm{flow}} \; + \; \lambda_{\mathrm{hull}}\mathcal{L}_{\mathrm{hull}} \; + \; \lambda_{\mathrm{avoid}}\mathcal{L}_{\mathrm{avoid}} \; + \; \lambda_{\mathrm{touch}}\mathcal{L}_{\mathrm{touch}} \; + \; \lambda_{\mathrm{margin}}\mathcal{L}_{\mathrm{margin}}.$$

The reported run uses weight 0.10 for hull, 0.06 for avoidance, 0.06 for touch support, 0.06 for touch forbidden space, and 0.04 for the margin term against the no geometry counterfactual. The margin term is important because it asks the geo conditioned branch to be better than its own no geometry prediction, rather than only to minimize an absolute occupancy error.

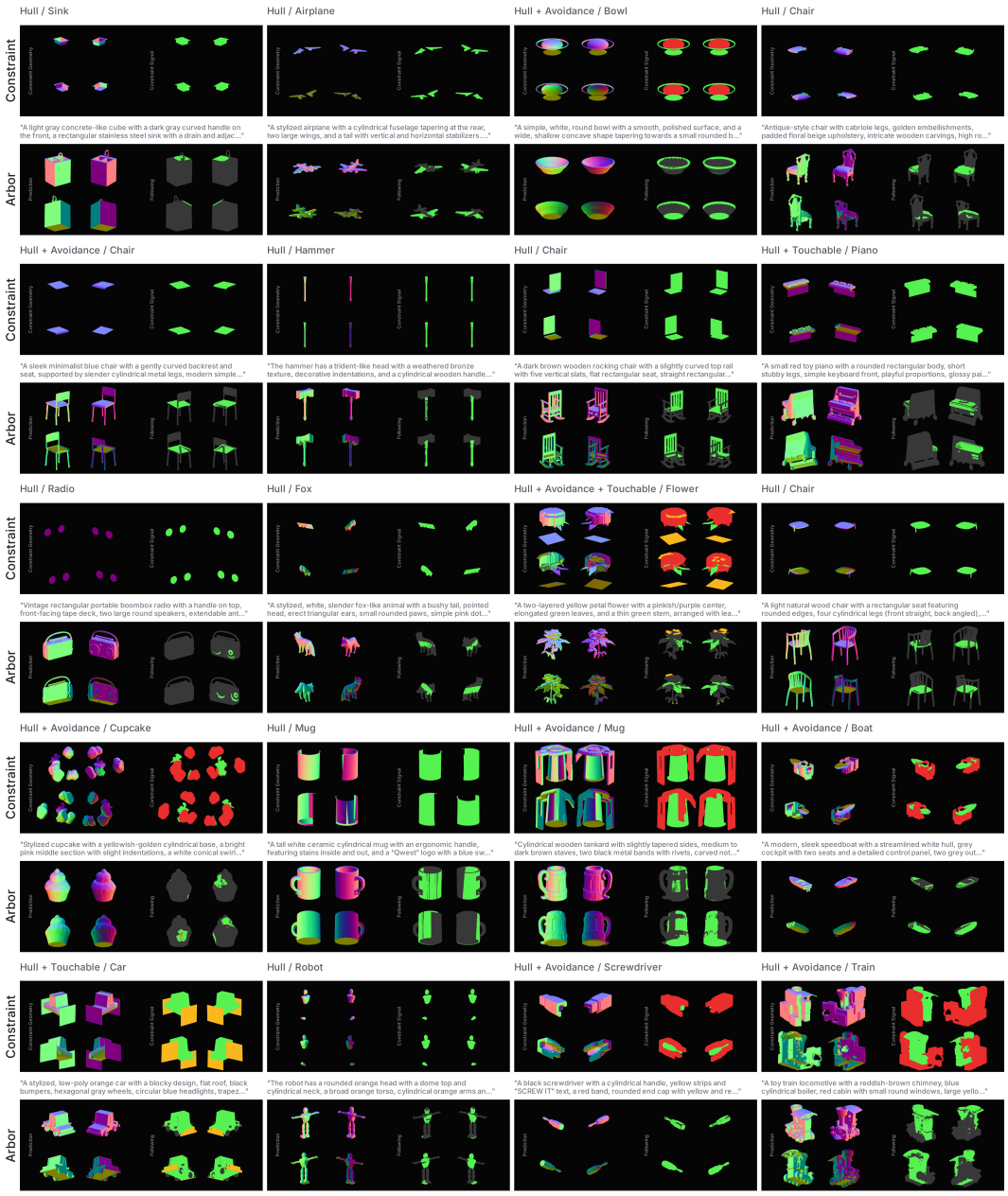

Figure 7: **Additional Arbor results on selected Toys4K constraints.** Showing manual and automatic benchmark cases.

In practice, this variant was not the best solution. It does improve direct constraint pressure, but it also moves the model toward a failure mode where following the constraint becomes easier than generating a plausible object from the prompt. This is close to the behavior seen in SpaceControl, where the guide itself can become the dominant object. From both the experiment section and the user study we therefore conclude that Compliance is useful as a probe, but too brittle to be the default method.

Semantics. Arbor Semantics keeps the geometry path of Arbor and changes only the text path. The motivation was that semantics should probably not be treated as just another geometry token. Instead, they should help the model understand what a local region means. For example, it is more useful to tell the network that a region is the seat of a chair than to append a weak semantic vector to the routed geometry memory.

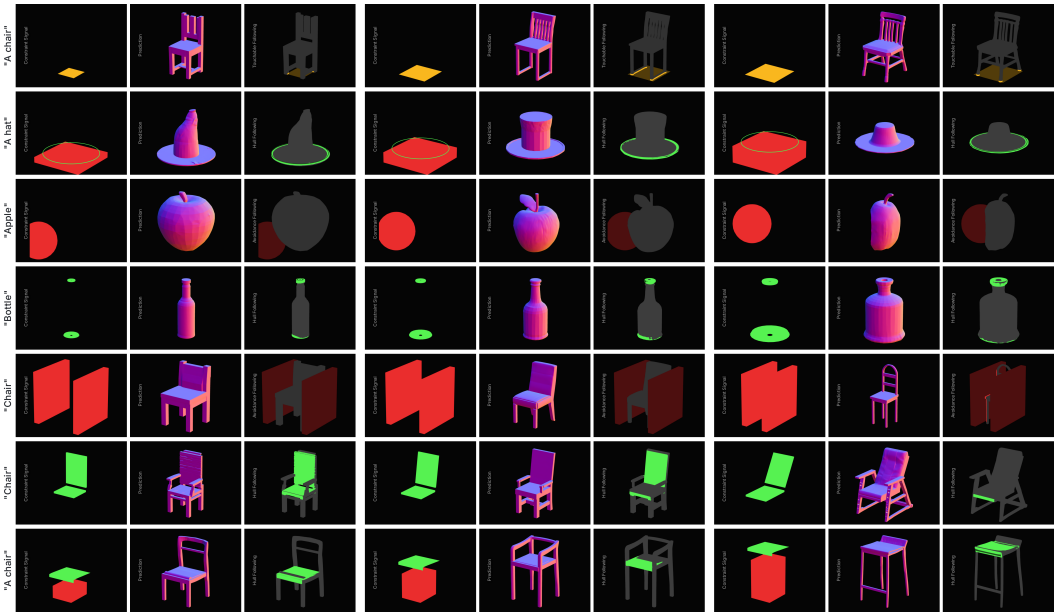

Figure 8: **Extended constraint sweeps.** Additional sweep selections using the same rendering language as Fig. 4, collected in the appendix to show further controlled variations beyond the main paper examples.

For a labeled query q with label set S_q and prompt y, we render a semantic first prompt

$$\widetilde{y}_q = \mathrm{render}(S_q, y), \qquad \widetilde{c}_q = E_{\mathrm{text}}(\widetilde{y}_q),$$

and replace the normal text context $E_{\mathrm{text}}(y)$ with $\widetilde{c}_q$ only for those queries. The routed geometry update from Eq. 5 stays unchanged. Only the text cross attention and its input normalization are reopened. Self attention, FFN, and time modulation remain frozen.

This is a more natural semantic injection site, because it touches the part of the model that already turns language into structure. We also tested semantic labels inside the routed geometry path, but those signals were mostly ignored. The text route is more promising. Still, the current Arbor Semantics model is not yet preferred over Arbor in the user study, and it remains less validated than the main model. We therefore present it as the clearest future direction, not as the final method.

Gradient baseline. The Gradient method is a separate idea that never becomes part of Arbor itself. It asks whether control can be injected only at inference time by modifying the denoising trajectory, instead of being learned during training. We keep TRELLIS frozen, decode the current sparse structure state, evaluate a hull loss on the occupied target region H, and backpropagate that loss into the latent:

$$g_k = \nabla_{x_k} \mathcal{L}_{\mathrm{hull}}(x_k), \qquad x_k \leftarrow x_k - \eta_k \frac{g_k}{\|g_k\|_2}.$$

In the reported setup this guidance is applied only during the last denoising steps.

This method does steer the sample toward the constraint, and it often does so better than SpaceControl. However, it also introduces new artifacts. The model can walk in different directions at once, partially reconstruct the guide, or copy in the constraint while damaging the rest of the object. This is why the Gradient row is useful as a test time baseline, but not as the main answer to the control problem.

Variant comparison. As mentioned in the method section, we also experimented with the placement of the adapter and with deeper interference into the TRELLIS block. Table 5 compares the final Arbor model against the two strongest alternatives that were fully evaluated. Late Adapter moves the geometry branch later in the block. Fused Attention+FFN merges text and geometry into one main conditioning site and also reopens the following FFN. Both variants can work, but Arbor remains the best balanced solution.

We also experimented with different encoding structures. One idea was to use O voxels directly and reduce them with a learned convolutional encoder. This failed badly, because the model first had to

Table 5: **Arbor and its two strongest internal alternatives under the full Toys4K protocol.** Manual (n=32) and automatic (n=128) splits. Late Adapter moves the geometry branch later in the sparse structure block. Fused Attention+FFN replaces Arbor's separate geometry branch with one retrained joint conditioning site followed by an unfrozen feed forward block. Values are reported under the same evaluator and control score definition as Tab. 1.

Method	Manual ($n = 32$)						Auto ($n = 128$)					
	Ctrl. Scr.↑	Hull Hit↑	Avoid Viol.↓	Touch Hit↑	Vol. Match↑	MV-CLIP↑	Ctrl. Scr.↑	Hull Hit↑	Avoid Viol.↓	Touch Hit↑	Vol. Match↑	MV-CLIP↑
Arbor	**0.4021**	**0.7143**	0.0245	**0.8571**	0.4685	**0.22947**	**0.4718**	**0.7860**	0.0055	**0.9841**	0.4866	**0.24020**
Late Adapter	0.3828	0.6299	0.0257	**0.8571**	**0.4818**	0.22933	0.4707	0.7827	**0.0049**	0.9762	**0.4873**	0.24019
Fused Attention+FFN	0.3372	0.6099	**0.0161**	0.5714	0.4578	0.22675	0.4574	0.7497	0.0102	0.9603	0.4695	0.24017

learn its own control representation before it could even start to use the signal. We also tried a coarse 16^3 control field that matches the TRELLIS lattice. This was easy for the network to detect, but too coarse to provide meaningful control at the final 64^3 output scale. In another version the network learned to copy the coarse signal instead of integrating it into the object.

Taken together, these experiments explain why the TRELLIS.2 encoding [35] is the right choice even if it looks unusual at first. It gives Arbor a strong frozen geometry representation with explicit signal channels. This lets the control path learn how to use geometry, instead of first having to invent the geometry representation itself.

A.4 Implementation Details

Paper model. Arbor is built on TRELLIS [36] sparse structure generation from text. The denoiser has 12 blocks, width 768, and operates on a 16^3 latent lattice with 8 channels per site. Constraint meshes are voxelized at 512^3, encoded by the frozen TRELLIS.2 [35] shape and attribute encoders into sparse 32^3 tokens, projected into model width, and capped at 2048 local geometry tokens per query group. The router partitions the sparse-structure lattice into 64 query groups of size $4 \times 4 \times 4$. Each group also receives 96 learned global summary tokens.

Trainable scope. Only the modules that face geometry are trained in the final Arbor run: the geometry projection, geometry position embedding, routed grounding adapters, global summary modules, and the small semantic part layers that attach confident labels to selected geometry regions. The pretrained TRELLIS self attention, text cross attention, and feed forward weights remain frozen. This keeps the base text prior intact and makes Arbor a real conditioning attachment rather than a full backbone finetune.

Training recipe. The paper model is trained on 8 GPUs with batch size 4 per GPU, AdamW at learning rate 10^{-4}, EMA rate 0.9999, fp16 training, and adaptive gradient clipping. The frozen paper checkpoint was reached after roughly 80 hours of training. Classifier free dropout is applied independently to text and geometry with probabilities 0.1 and 0.1. No explicit compliance loss is used in the final Arbor run. Progression boards use a fixed evaluation subset of 72 samples balanced across ABO, HSSD, and ObjaverseXL Sketchfab, with fixed prompts, fixed constraints, and fixed noise. This recipe was selected because it remained stable across long multi GPU runs without reopening the frozen TRELLIS backbone.

Constraint family balance. Every training sample contains one hull family. Avoidance and touchable families are activated independently with probability 0.5 each. Within the hull sampler, part based hulls dominate the mass with weight 0.6, while random patch hulls, section crops, planar patches, support patches, symmetry anchors, and primitive proxies each carry weight 0.0667. Within avoidance, layout blockers and inverted carvings each carry weight 0.35, surface clearance regions carry 0.2, and coupled keep out patterns carry 0.1. Touchable uses one family but varies the support side, patch shape, and forbidden half space.

Constraint source geometry. The paper run uses reduced segmented meshes for part based constraint creation, but preserves exact PartSAM identities for semantic joins. This choice matters for both speed and data quality: Arbor keeps the geometry source compact enough for stable training while still letting the semantic annotations refer to the same part identities.

A.5 Evaluation Details

Evaluator protocol. All paper benchmarks are frozen manifests. Each row fixes the prompt, typed constraint meshes, and the benchmark canonical frame. All compared methods are first aligned to one common evaluation interface and always return a mesh, regardless of their native internal representation. The evaluator then voxelizes or rerenders those meshes in one shared protocol. This is what makes the 64^3 control metrics comparable across latent generators, training-free guidance baselines, and modality-mismatched reconstruction models.

Metric definitions. All control metrics use a shared 64^3 voxel grid. This matches the sparse structure stage and gives one common grid for all methods, but very thin contact errors still require qualitative inspection. Hull Hit is hull support recall, i.e. the fraction of required hull support surface reached by the prediction. Avoid Viol. is the occupied fraction inside forbidden avoidance volume. Touch Hit is the hit rate over touch constraints: a touchable region counts as satisfied if the prediction reaches it anywhere. Volume Match is a bounded occupancy count agreement term,

$$\mathrm{VolMatch} = \frac{\min\big(|V_{\text{pred}}|,\ |V_{\text{gt}}|\big)}{\max\big(|V_{\text{pred}}|,\ |V_{\text{gt}}|\big)},$$

where V_{gt} is the source Toys4K asset occupancy used to construct the benchmark row. This is not a reconstruction metric; it is only a coarse size guard that prevents methods that simply overfill the guide from looking artificially complete. MV-CLIP is computed from canonical semantic render views and acts as a coarse prompt semantic check rather than as a realism metric.

Ctrl. Scr. Ctrl. Scr. is computed per sample as a harmonic mean over the positive terms that apply to that sample. Let

$$\mathcal{S}_i = \{\mathrm{HullHit}_i,\ \mathrm{VolMatch}_i,\ 1 - \mathrm{AvoidViol}_i,\ \mathrm{MVCLIP}_i\},$$

and when touch constraints are present, append $\mathrm{TouchHit}_i$. The sample score is

$$\mathrm{CtrlScr}_i = \frac{|\mathcal{S}_i|}{\sum_{s \in \mathcal{S}_i} \frac{1}{\max(s, \varepsilon)}},$$

and the table reports the mean over the split. The harmonic mean is deliberate: a method cannot hide a severe control failure behind one strong metric. It is a compact summary, not a replacement for the component columns reported in Tab. 1.

Fair comparison policy. The main controlled generation table is restricted to methods that can reasonably be evaluated as text plus geometry generation under the same benchmark protocol. Image conditioned or point conditioned methods are moved to the fixed hull variation track rather than being mixed into the main control table with extra cues. In that track, the required image or point set condition is derived from the same benchmark hull rather than giving those methods additional authored supervision. Internally we also keep track of whether a comparison row is official, surrogate, modality mismatched, or an internal baseline, so those distinctions stay explicit during evaluation. We also keep diagnostic GT metrics such as Chamfer and ICP aligned scores in the evaluator, but we do not surface them in the main paper table because the paper claim is controlled generation, not reconstruction accuracy.

User study. We compare the table trends with a small preference study of 404 unlabeled pairwise choices from 27 participants. Each trial shows the prompt, the constraint render, and candidate outputs without method names. Participants are asked to choose the preferred result under the combined criterion of control following and object plausibility. The reported percentages are pairwise win rates over the trials in which a method appears, not a single multinomial distribution over methods. For this study only, Arbor, Arbor Semantics, and Arbor Compliance are merged into one Arbor family, because the question is whether the Arbor conditioning approach is preferred overall, not which small internal variant wins within that family. That is why the preference percentage in Tab. 1 is shown both for the base Arbor row and for the merged Arbor family number in parentheses.

Responsible research notes. The user study is a low risk visual preference task over rendered 3D objects. The interface shows the prompt, the constraint image, and anonymized method outputs, and asks participants to select the result that best balances constraint following with object plausibility. The instruction shown to participants is: choose the output that best follows the shown constraint while remaining a plausible object for the prompt. No paid crowd work dataset was collected for this study. No personal or sensitive participant data is used in the model or benchmark. All training and evaluation assets are drawn from cited 3D datasets and models, and their original sources are credited in the paper. Our new assets are the Arbor method, the typed constraint generation pipeline, the Toys4K control benchmark manifests, and the evaluation code. The code and data package is not part of the initial submission. The planned public release will include source citations, license notes, configuration files, benchmark manifests, and evaluation scripts after review and packaging checks. The main risks are the same as for other 3D asset generators: generated assets may be low quality, misleading, or incompatible with source asset licenses if used without review. We therefore treat Arbor as an authoring aid whose outputs should remain subject to human inspection and license checks before use.

A.6 Extended Limitations

The main paper already states the most important limitations, but three additional points matter for interpretation. First, Arbor still operates only at the sparse structure stage, so later SLAT refinement and decoding can soften very local geometric details even when the coarse support is correct. Second, constraint meshes can conflict sharply with the text prior: very small hulls, weak semantic cues, or aggressive keep out regions can force the model into compromises where either prompt following or local obedience becomes visibly worse. Third, the benchmark remains heterogeneous because nearby methods come from editing, completion, sampling time steering, and image conditioned reconstruction rather than from one clean text plus geometry benchmark family. We therefore keep modality specific methods in the track where their inputs are honest, and we do not claim that Arbor solves every form of 3D control. Semantic labels remain the clearest next step, but the current semantic variants still do not match routed geometry as the main control carrier.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and Sec. 1 state the interface, encoding, routing, and adapter claims. The evaluation in Sec. 4, the user study described in A, and the limitations folded into Sec. 5 together with A.6 state the tested scope and current boundaries.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: Sec. 5 discusses Arbor's geometry without roles limitation and its sparse structure stage focus. A.6 extends this with prompt and constraint conflict regimes, the weak semantic extension, and benchmark heterogeneity. A.3 reports the gradient, compliance, and semantic Arbor variants.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A].

Justification: The paper does not present theoretical results or proofs. The equations in Sec. 3 define the architecture and training objective.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes].

Justification: Sec. 3, Sec. 4, A.1, and A.5 describe the architecture, trainable modules, loss, benchmark splits, baselines, and metrics used for the main claims. Some release details such as public code packaging are handled separately in the code access question below.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No].

   Justification: The paper discloses the method, data construction, benchmarks, metrics, and implementation details in Sec. 3, Sec. 4, Appendix A.1, and Appendix A.5, but the code and data package is not part of the initial submission. We plan a public release after review and packaging checks.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: Sec. 4 specifies the training mixture, benchmark sizes, baselines, and metric protocol. A.1, A.3, and A.5 provide additional data, baseline, and metric details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No].

   Justification: Tab. 2 reports mean and standard deviation across seeds for the fixed constraint variation track, but the main controlled generation table reports deterministic benchmark means over the frozen manual and automatic splits without confidence intervals as computation is expensive for these methods such as Spice-E and to not clutter the table even more, with the broad analyisis we already make in the paper.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: Appendix A.4 reports the main Arbor training setup, including the eight GPU run, batch size, optimizer, precision, and approximate training time. Baseline and ablation compute is smaller in scope and follows the evaluation protocol described in Appendix A.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work focuses on controllable generation of object assets, uses cited 3D datasets and models, and reports a low risk preference study over rendered outputs. Appendix A.5 notes that no personal or sensitive participant data is used in the model or benchmark.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes].

Justification: Appendix A.5 discusses the positive impact of a more controllable 3D authoring interface and the negative risks of low quality, misleading, or rights infringing 3D assets. It also states that generated outputs should remain subject to human review and source license checks.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [Yes].

Justification: Appendix A.5 states that released assets should include source citations, license notes, documentation, and human review before use. The submission does not release an unchecked scraped dataset dump.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes].

Justification: Existing models, baselines, and datasets are credited through citations in the main paper and supplement. Appendix A.5 states that released assets should include source citations and license notes before use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [No].

Justification: The paper introduces Arbor and two Toys4K control benchmarks, and documents their construction, constraint semantics, evaluation protocol, and metrics in Sec. 3, Sec. 4, Appendix A.1, and Appendix A.5. The assets themselves are not included with the initial submission; documentation will accompany the later public release.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [Yes].

Justification: Appendix A.5 describes the participant task, trial content, and preference criterion. The study is a small preference study over rendered examples rather than a paid crowd work dataset collection.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [Yes].

    Justification: Appendix A.5 describes the study as a low risk visual preference task over rendered 3D outputs and states that no personal or sensitive participant data is used in the model or benchmark. The study followed the applicable internal process for such low risk preference judgments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes].

    Justification: Appendix A.1 describes the offline semantic extraction pipeline, where a local Qwen model proposes candidate part phrases for optional semantic annotations. These labels support secondary semantic variants and are not the central mechanism behind the main Arbor result.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.