

Zen AI Model Family

Zen-3D

Image-to-3D Generation via a Redistribution of Microsoft TRELLIS

Technical Report v2025.01

Hanzo AI Research Team* Zoo Labs Foundation[†]

January 2025

Abstract

Zen-3D is a packaging and integration of **TRELLIS**—the open-source 3D generation system released by Microsoft Research [1]—into the Zen model stack. It is *not* a from-scratch model: Zen-3D redistributes the publicly available TRELLIS code and pretrained weights (most notably the image-conditioned **JeffreyXiang/TRELLIS-image-large** checkpoint, $\approx 1.2\text{B}$ parameters) under their original **MIT license**, wrapped behind the Zen SDK and serving interfaces. TRELLIS generates high-quality 3D assets from a single image or a text prompt using a unified *Structured LATent* (SLAT) representation and rectified-flow transformers, and can decode the same latent into 3D Gaussian splats, radiance fields, or textured meshes. This report describes what TRELLIS actually is and how it works, documents which artifacts Zen-3D redistributes and under what license, and explains the integration surface (API, output formats, deployment) that the Zen stack adds on top. All architectural details and any reported results are attributed to the original TRELLIS authors; we do not claim a novel architecture and report no benchmarks of our own.

Contents

1	Introduction	3
1.1	What Zen-3D is, and what it is not	3
1.2	Why TRELLIS	3
1.3	Redistributed Artifacts	3
2	TRELLIS Architecture	4
2.1	Structured LATent (SLAT) Representation	4
2.2	SLAT Variational Autoencoder and Decoders	4
2.3	Two-Stage Rectified-Flow Generation	4
2.4	Conditioning	5
2.5	Model Sizes and Training Data (as reported upstream)	5
3	Reported Results (Upstream)	5
4	Zen Integration	5
4.1	What Zen-3D Provides	5
4.2	Output Formats	6
4.3	Python API	6
4.4	Deployment Notes	6

*research@hanzo.ai

[†]foundation@zoo.ngo

5	Licensing and Attribution	7
6	Conclusion	7

1 Introduction

1.1 What Zen-3D is, and what it is not

Zen-3D is the Zen stack’s image-to-3D and text-to-3D capability. Concretely, it is a redistribution and integration of **TRELLIS** (“Structured 3D Latents for Scalable and Versatile 3D Generation”), an open-source system from Microsoft Research presented at CVPR 2025 [1]. The TRELLIS code and pretrained weights are released under the **MIT license** [2], which permits redistribution and commercial use with attribution and license retention.

To be explicit about provenance:

- Zen-3D does **not** introduce a new model architecture. There is no “from-scratch” Zen 3D network and no proprietary 3D training run behind this report.
- The model, weights, and generation algorithm are those of TRELLIS, authored by Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang [1].
- Zen-3D’s contribution is purely at the *packaging and integration* layer: pinning a known-good TRELLIS checkpoint, exposing it through the Zen SDK, normalizing its output formats, and providing serving/deployment glue.
- All quantitative results belong to the original work; this report reproduces no benchmark numbers of its own and omits any figure we cannot attribute.

1.2 Why TRELLIS

Single-image and text-conditioned 3D asset generation is a frontier capability for content creation, simulation, and AR/VR. Among openly licensed options, TRELLIS is notable for (1) a permissive MIT license on both code and weights, (2) a unified latent representation that can be decoded into multiple downstream 3D formats (Gaussian splats, radiance fields, meshes) from a single generation pass, and (3) strong reported quality at the 1–2B parameter scale [1]. These properties make it a practical foundation to integrate rather than a model to reinvent.

1.3 Redistributed Artifacts

Property	Value
Upstream project	TRELLIS (Microsoft Research) [1, 2]
Task	Image-to-3D and text-to-3D generation
Representation	Structured LATent (SLAT)
Generative model	Rectified-flow transformers (two-stage)
Primary weights	JeffreyXiang/TRELLIS-image-large ($\approx 1.2\text{B}$)
Other public weights	TRELLIS-text-base/large/xlarge (342M / 1.1B / 2.0B)
Output formats	3D Gaussian splats, radiance fields, textured meshes
License (code + weights)	MIT [2]
Training data (upstream)	TRELLIS-500K ($\approx 500\text{K}$ curated 3D assets)
Zen-3D additions	SDK wrapper, serving, output normalization

Table 1: Zen-3D: artifacts redistributed from TRELLIS and the integration layer added by Zen. Parameter counts and dataset sizes are as reported by the TRELLIS authors [1].

2 TRELIS Architecture

This section summarizes the TRELIS method as described by its authors [1]. It is included for completeness; none of it is original to Zen-3D.

2.1 Structured LATent (SLAT) Representation

The cornerstone of TRELIS is the *Structured LATent* (SLAT) representation: a set of local latent vectors attached to the active (occupied) voxels of a sparse 3D grid. Formally, an asset is encoded as

$$\mathbf{z} = \{(\mathbf{z}_i, \mathbf{p}_i)\}_{i=1}^L, \quad \mathbf{p}_i \in \{0, \dots, N-1\}^3, \quad (1)$$

where each $\mathbf{p}_i$ is the integer coordinate of an active voxel and $\mathbf{z}_i$ is its local latent vector. The active voxels $\mathbf{p}_i$ outline the coarse geometry of the asset, while the latents $\mathbf{z}_i$ capture finer structural and textural detail. By default the grid resolution is $N = 64$, which yields roughly $L \approx 20,000$ active voxels per asset on average [1].

The per-voxel features used to fit the latents are obtained from 2D vision features rather than from raw geometry. TRELIS renders an asset from many camera views sampled on a sphere, extracts feature maps with a pretrained **DINOv2** encoder, projects each active voxel into those multiview feature maps, and averages the retrieved features to form the voxel feature $\mathbf{f}_i$ [1]. This couples geometry (the sparse grid) with appearance (the aggregated visual features) in a single structured latent.

2.2 SLAT Variational Autoencoder and Decoders

A transformer-based VAE encodes the voxel features into SLAT and reconstructs 3D outputs from it. Voxel tokens are serialized with sinusoidal positional encodings derived from their coordinates. A distinctive property of TRELIS is that the *same* SLAT can be decoded into several 3D output formats by separate, format-specific decoders [1]:

- **3D Gaussians.** Each latent decodes to a small set of Gaussians with position offsets, colors, scales, opacities, and rotations. Positions are anchored to the voxel and perturbed by a bounded offset, e.g. $\mathbf{x}_i^k = \mathbf{p}_i + \tanh(\mathbf{o}_i^k)$. The encoder and the Gaussian decoder are trained end-to-end.
- **Radiance fields.** Each latent decodes to a local radiance volume represented with a CP-decomposition (following Strivec), enabling neural volumetric rendering.
- **Meshes.** Each latent decodes to FlexiCubes parameters and signed-distance values, from which a textured surface mesh is extracted (upsampled to a higher resolution).

The non-Gaussian decoders are trained separately on top of a frozen encoder [1].

2.3 Two-Stage Rectified-Flow Generation

TRELIS generates SLAT with two **rectified-flow transformers** trained with a conditional flow-matching objective, run in sequence [1]:

1. **Sparse-structure stage.** A flow transformer $\mathcal{G}_S$ generates the sparse occupancy structure—i.e. which voxels are active. The binary grid is first compressed by a VAE into a low-resolution dense feature grid, and the flow model operates in that space before decoding back to the occupied voxel set.
2. **Structured-latent stage.** A second flow transformer $\mathcal{G}_L$ generates the local latents $\mathbf{z}_i$ for the active voxels produced by stage 1, operating over the sparse set of occupied cells.

Both transformers use attention adapted to 3D sparsity (e.g. serialization of active voxels and shifted-window attention in 3D); $\mathcal{G}_S$ and $\mathcal{G}_L$ are trained separately [1].

2.4 Conditioning

TRELLIS supports both image and text conditioning, injected into the flow transformers via cross-attention [1]:

- **Image conditioning** uses DINOv2 visual features of the input image. This is the path exercised by the TRELLIS-image-large checkpoint that Zen-3D primarily redistributes.
- **Text conditioning** uses CLIP text features, corresponding to the text-conditioned TRELLIS checkpoints.

2.5 Model Sizes and Training Data (as reported upstream)

The TRELLIS authors report training models at three scales—**342M** (Base), **1.1B** (Large), and **2.0B** (X-Large)—on **TRELLIS-500K**, a curated set of roughly 500K 3D assets sourced from Objaverse(XL), ABO, 3D-FUTURE, HSSD, and Toys4k and filtered by aesthetic score [1, 2]. The image-conditioned checkpoint redistributed by Zen-3D, JeffreyXiang/TRELLIS-image-large, is the ≈ 1.2 B-parameter image variant.

Public checkpoint	Parameters	Conditioning
TRELLIS-image-large	≈ 1.2 B	Image
TRELLIS-text-base	342M	Text
TRELLIS-text-large	1.1B	Text
TRELLIS-text-xlarge	2.0B	Text

Table 2: Publicly released TRELLIS checkpoints, as listed by the upstream project [2]. Zen-3D primarily integrates the image-conditioned model.

3 Reported Results (Upstream)

We do not run our own evaluation, and we do not restate numerical benchmarks here because we cannot independently verify them in this report. The TRELLIS paper reports that the method produces diverse, high-quality 3D assets and compares favorably against prior and contemporaneous 3D generators at similar scale; for the exact quantitative comparisons, evaluation protocol, and qualitative galleries, we refer the reader to the original publication and project page [1, 2]. Any number not attributable to that source should be considered absent rather than implied.

4 Zen Integration

Everything in this section is the integration layer that Zen-3D adds; it does not modify the TRELLIS model itself.

4.1 What Zen-3D Provides

- **Pinned redistribution.** A fixed, known-good TRELLIS checkpoint and code revision, repackaged under the Zen distribution while retaining the upstream MIT license and attribution (see Section 5).

- **SDK surface.** A thin Python wrapper that loads the upstream `TrellisImageTo3DPipeline` and exposes a uniform Zen API for image-to-3D (and text-to-3D where the corresponding checkpoint is used).
- **Output normalization.** Helpers to export the decoded SLAT into the format a caller wants—3D Gaussian splats, a radiance field, or a textured mesh (e.g. GLB/OBJ)—using TRELLIS’s own decoders.
- **Serving glue.** Deployment configuration for running the pipeline behind the Zen serving stack.

4.2 Output Formats

Because a single SLAT can be decoded multiple ways, the same generation can yield different deliverables without re-running the flow models:

Output	Decoder (TRELLIS)	Typical use
3D Gaussian splats	Gaussian decoder	Fast novel-view rendering
Radiance field	CP-decomposition volume	Volumetric rendering
Textured mesh	FlexiCubes + SDF	DCC tools, game/CAD pipelines

Table 3: TRELLIS output formats exposed through the Zen-3D API. Decoders are those of the upstream model [1].

4.3 Python API

The Zen-3D wrapper mirrors the upstream TRELLIS pipeline. Usage is illustrative:

```

1 from zen import Zen3D
2 from PIL import Image
3
4 # Loads the redistributed TRELLIS-image-large checkpoint (MIT licensed)
5
6 model = Zen3D.from_pretrained("zenlm/zen-3d")
7
8 image = Image.open("input.png")
9 outputs = model.generate(image)
10
11 # Decode the same SLAT into the format you need.
12 outputs.gaussians.save("asset.ply")      # 3D Gaussian splats
13 outputs.mesh.export("asset.glb")        # textured mesh (FlexiCubes)
# outputs.radiance_field for volumetric rendering

```

Listing 1: Zen-3D image-to-3D (wrapping TRELLIS)

4.4 Deployment Notes

TRELLIS depends on differentiable rendering submodules (e.g. a differentiable octree renderer and a modified FlexiCubes) for training and for certain decoding paths; these submodules carry their own licenses (see Section 5) and have their own build requirements. Inference for the redistributed image model runs on a single modern GPU; for exact hardware requirements, dependency versions, and build steps, follow the upstream project documentation [2].

5 Licensing and Attribution

- **TRELLIS code and weights: MIT.** The TRELLIS repository and the released checkpoints (including TRELLIS-image-large) are MIT licensed [2]. Zen-3D’s redistribution retains the upstream MIT license text and copyright notice, and credits Microsoft Research and the TRELLIS authors.
- **Submodule exceptions.** Some optional submodules are not MIT: the differentiable octree renderer (`diffoctreerast`) and the modified FlexiCubes are licensed under their own terms [2]. Deployments that build these components must comply with those licenses.
- **Upstream dependencies.** The vision/text encoders used for conditioning (DINOv2, CLIP) and the data sources behind TRELLIS-500K carry their own licenses and usage terms, which downstream users should review independently.
- **No relicensing of capabilities.** Zen-3D claims no ownership of the TRELLIS model or its outputs’ underlying method. Where this report describes architecture or results, the source of record is the TRELLIS paper and repository [1, 2].

6 Conclusion

Zen-3D is an honest packaging of Microsoft’s TRELLIS into the Zen stack: an MIT-licensed, image-conditioned (and optionally text-conditioned) 3D generator built on the Structured Latent representation and two-stage rectified-flow transformers, capable of decoding to 3D Gaussian splats, radiance fields, or meshes. The model, its architecture, and any reported quality are the work of the TRELLIS authors [1]; Zen-3D adds only redistribution, an SDK wrapper, output normalization, and serving integration. We make no from-scratch architecture or benchmark claims, and we encourage users to consult the upstream project for authoritative technical detail and evaluation.

References

- [1] J. Xiang, Z. Lv, S. Xu, Y. Deng, R. Wang, B. Zhang, D. Chen, X. Tong, and J. Yang, “Structured 3D Latents for Scalable and Versatile 3D Generation,” CVPR, 2025 (arXiv:2412.01506). <https://arxiv.org/abs/2412.01506>
- [2] Microsoft Research, “TRELLIS,” GitHub repository (MIT license; weights JeffreyXiang/TRELLIS-image-large). <https://github.com/microsoft/TRELLIS>