Residual Primitive Fitting of 3D Shapes with SuperFrusta
— Supplementary —

Aditya Ganeshan* Matheus Gadelha† Thibault Groueix† Zhiqin Chen†
Siddhartha Chaudhuri† Vladimir G. Kim† Wang Yifan† Daniel Ritchie*

*Brown University †Adobe Research

adityaganeshan@gmail.com

Supplementary Overview

This document provides additional qualitative results, detailed technical explanations, and extended experiments supporting the main paper. For ease of navigation, we list the major sections below.

1. **Qualitative Examples** (Sec. 1)
   Expanded visualizations of primitive assemblies on Toys4K, Part-Objaverse, and 3D GenPrim.
2. **Understanding SuperFrusta** (Sec. 2)
   Expanded formulation, parameterization, and projection operators.
3. **ResFit Details** (Sec. 3)
   High-level design, iterative loop structure, and full algorithmic flow.
4. **Decomposition & Initialization** (Sec. 4)
   Detailed description of our volumetric partitioning method.
5. **Optimization Details** (Sec. 5)
   Two-phase optimization, stochastic preconditioning, curvature weighting, and hyperparameters.
6. **Applications** (Sec. 6)
   Textured primitive assemblies, canonical CSG inference, and semantic refinement.
7. **Additional Experiments** (Sec. 7)
   Further evaluations, ablations, and extended comparisons.
8. **Failure Cases** (Sec. 8)
   Representative scenarios where our method struggles and directions for improvement.

1. Qualitative Examples

Qualitative results on diverse 3D shapes. Figures 1 and 2 present qualitative reconstructions produced by our method on assets sourced from two datasets: Toys4K [10] and Part-Objaverse [13]. Across both collections, by combining ResFit with SuperFrustum, we successfully models a broad range of object categories and geometric shapes. The inferred assemblies remain both *accurate* and *parsimonious*, capturing thin features, complex curvature, while partitioning the shape (often) along semantically meaningful partitions. These examples highlight the versatility of our primitive design: despite its compact parameterization, it adapts fluidly to highly heterogeneous shapes, producing coherent decompositions across diverse objects.

Comparison to prior work. We additionally compare our method against Primitive Anything [14] (PA) (with test-time optimization) and Marching Primitives [4] (MPS) (at 128 resolution) on representative meshes from the 3D GenPrim dataset [12]. As shown in Figures 3, our method achieves noticeably higher reconstruction fidelity while using substantially fewer primitives. The performance gap is visually pronounced: prior methods tend to either over-fragment shapes or underfit curved and intricate structures, whereas our approach consistently yields clean, coherent assemblies that more closely match the target geometry.

2. Understanding SuperFrusta

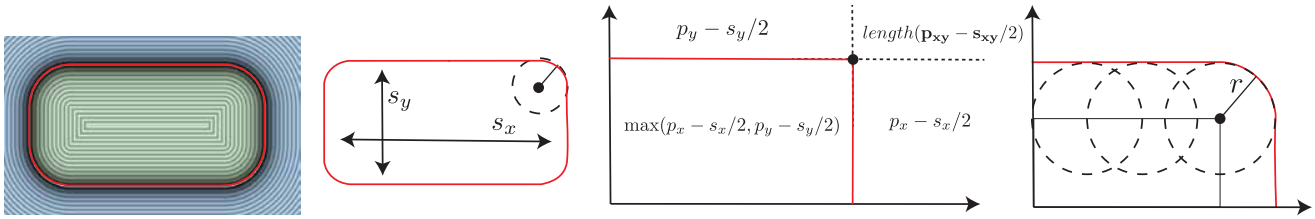
As discussed in the main paper, SuperFrusta is a convex primitive defined by an exact signed–distance function f with six interpretable controls: *size*, *dilation*, *roundness*, *taper*, *curvature*, and *onion* (shell thickness). We group them as $\theta = \{s_x, s_y, s_z, r, d, t, c, o\}$, where "size" counts as one control though it has three components.

$$\text{sdf}_{\mathbf{p}} \;=\; f(\mathbf{p};\theta), \qquad \theta = \{s_x, s_y, s_z,\; r,\; d,\; t,\; c,\; o\}.$$

Figure 1. Qualitative results on the Toys4K dataset. Each example shows the input mesh (gray, bottom row) and the corresponding primitive assembly inferred by ResFit. These examples highlight our method's ability to capture fine geometric detail and overall structure using compact, interpretable assemblies across a wide variety of toy categories.

Figure 2. Primitive assemblies inferred on samples from Toys4K and Part-Objaverse. The top two rows show reconstructions on the diverse, real-world objects of Part-Objaverse, while the bottom rows show additional Toys4K results. Across both datasets, ResFit consistently produces compact assemblies that faithfully follow complex, varied geometries; gray rows show the corresponding input meshes.

Figure 3. Comparison with PA (TTO) [14] and MPS [4] on examples from the 3D GenPrim dataset [12]. Our method achieves higher geometric fidelity while using far fewer primitives.



a) Level sets of RoundedRect2D b) Parameterization c) SDF by region in the quarter rectangle d) Morphological dilation

Figure 4. RoundedRect2D. From left to right: level sets of the primitive; parameterization with size (s_x, s_y) and radius r; SDF for each of the four regions of an origin-centered rectangle (by symmetry in x and y we reason in the top-right quadrant); morphological dilation to achieve a rounded rectangle with corner radius r, formed by shrinking to $\frac{1}{2}(s_x, s_y) - (r, r)$ and taking the r-level set. In all panels, the 0-level set is shown in red.

Construction by composition. Let $\mathbf{p} = (x, y, z)$ and define the stages

$$\begin{aligned} \mathbf{p}^* &= \text{DomainCurve3D}(\mathbf{p};\ s_z,\ c)\,, \\ \mathbf{s}_{2D} &= \text{RoundedRect2D}\big(\mathbf{p}^*_{xy};\ s_x, s_y,\ r\big)\,, \\ s^* &= \text{Trapezoid2D}((\,\mathbf{s}_{2D},\ \mathbf{p}^*_z\,);\ s_z,\ t,\ o)\,, \\ f(\mathbf{p};\theta) &= s^* - d. \end{aligned}$$

We start by defining the 2D SDF primitives RoundedRect2D and Trapezoid2D, as well as their respective parameters. Second, we discuss how to compose these two 2D SDFs to form a 3D SDF, with Trapezoid2D($(\,\mathbf{s}_{2D},\ \mathbf{p}^*_z\,)$; s_z, t, o). Third, we describe the curving function DomainCurve3D that produces the bulge effect. Last, we discuss the related work that informed this formulation.

2.1. RoundedRect2D

In this section, we describe the analytic formulation for RoundedRect2D, parameterized by s_x, s_y, r. See our Shadertoy example for an illustration, as well as panels (a)–(b) of Figure 4 for a visualization of the level sets and the parameters.

Symmetry preamble. Consider an origin–centered, axis–aligned rectangle. The shape is invariant under the reflections $(x, y) \mapsto (\pm x, \pm y)$, and the oriented (inside–negative) distance to its boundary is unchanged by these reflections. It therefore suffices to derive the expression for the top–right quadrant $x \geq 0$, $y \geq 0$; the formulas for the other quadrants follow by replacing x, y with $|x|, |y|$. (This symmetry relies on the rectangle being centered and axis–aligned.)

We start by defining the SDF for a 2D rectangle, then generalize to RoundedRect2D.

Axis-aligned rectangle. Let $p_{xy} = (x, y) \in \mathbb{R}^2$. We consider a rectangle centered at the origin, with half-sizes $\mathbf{h} = (h_x, h_y) = (\frac{s_x}{2}, \frac{s_y}{2})$. Define $\mathbf{q} = (q_x, q_y) = \big(|x| - h_x,\ |y| - h_y\big)$. The signed distance to the rectangle (negative inside) is obtained by studying four cases (outside–corner, outside–near a vertical edge, outside–near a horizontal edge, inside); see panel (c) of Figure 4:

$$d_\square(p_{xy}; \mathbf{h}) = \begin{cases} \|\mathbf{q}\|_2, & q_x > 0,\ q_y > 0, \\ q_x, & q_x > 0 \geq q_y, \\ q_y, & q_y > 0 \geq q_x, \\ \max(q_x, q_y), & q_x \leq 0,\ q_y \leq 0. \end{cases}$$

The above can be rewritten more compactly:

$$\begin{aligned} \mathbf{q} &= \big|\mathbf{p}_{xy}\big| - \mathbf{h}, \\ \text{sd}_\square(p_{xy}; \mathbf{h}) &= \big\|\max(\mathbf{q}, \mathbf{0})\big\|_2 + \min\big(\max(q_x, q_y), 0\big). \end{aligned} \tag{1}$$

Rounded rectangle via dilation by radius r. To form a rounded rectangle, given a corner radius $r \geq 0$, we first shrink the half-extents, $\mathbf{h}' = \mathbf{h} - (r, r)$ (assume $r \leq \min(h_x, h_y)$), then take the r-level set (a morphological dilation / Minkowski sum with a disk of radius r). See panel (d) of Figure 4. In signed-distance terms this is

$$\begin{aligned} \text{sd}_{\text{rrect}}(\mathbf{p}_{xy};\ s_x, s_y,\ r) &= \text{sd}_\square\big(p_{xy}; \mathbf{h}'\big) - r, \\ \mathbf{h}' &= \tfrac{\mathbf{s}_{xy}}{2} - (r, r), \end{aligned} \tag{2}$$

where sd_{rrect} is an appreviation for the RoundedRect2D function. Expanding with the rectangle one-liner gives the standard shader form:

$$\begin{aligned} \mathbf{q} &= \big|\mathbf{p}_{xy}\big| - \big(\mathbf{h} - (r, r)\big), \\ \text{sd}_{\text{rrect}}(\mathbf{p}_{xy};\ s_x, s_y,\ r) &= \big\|\max(\mathbf{q}, \mathbf{0})\big\|_2 \\ &\quad + \min\big(\max(q_x, q_y), 0\big) - r. \end{aligned} \tag{3}$$

2.2. Trapezoid2D

We now describe the analytic formulation for Trapezoid2D, parameterized by s_z, t, o (height, taper, and onion factor), visualized in Figure 5 and Shadertoy.

Trapezoid geometry. Let $p = (x, y) \in \mathbb{R}^2$. We work with the intermediary parameters $H > 0$, $\texttt{inner} \geq 0$, and $x_3 \in \mathbb{R}$ (derived from t, s_x, s_y, s_z), and set $y_B = -H$, $y_T = +H$. The trapezoid is defined by its four vertices:

$$\begin{aligned} L_t &= (-\texttt{inner},\, y_T), \qquad & R_t &= (x_3,\, y_T), \\ L_b &= (-\texttt{inner},\, y_B), \qquad & R_b &= (0,\, y_B). \end{aligned} \tag{4}$$

Its four edges are: $[L_b, L_t]$ (left), $[R_b, R_t]$ (right), $[L_b, R_b]$ (bottom), $[L_t, R_t]$ (top). The right-edge direction is $e_S = (x_3,\, 2H)$; the left edge is vertical.

Here, the intermediary parameters are computed based on the parameters t and $\mathbf{s}$ as follows:

$$\begin{aligned} \texttt{inner} &= \min(s_x, s_y) \times 0.5, \\ x_3 &= -(1 - \mathrm{t})\, \texttt{inner}, \\ H &= 0.5 \times s_z. \end{aligned} \tag{5}$$

The distance function is the minimum distance from p to the four edges:

$$\mathrm{sd}_{\mathrm{trap}}(p) \;=\; \pm\, \min\big\{d_{\mathrm{left}}(p),\, d_{\mathrm{right}}(p),\, d_{\mathrm{bottom}}(p),\, d_{\mathrm{top}}(p)\big\},$$

with negative sign inside the trapezoid and positive outside. We now detail the distance to one edge (the right slanted edge); the other edges follow analogously.

Distance to one segment via projection. Consider the segment $[A, B] = [R_b, R_t]$ with

$$\begin{aligned} A &= R_b = (0,\, -H), \\ B &= R_t = (x_3,\, +H), \\ e &= B - A = (x_3,\, 2H). \end{aligned} \tag{6}$$

Define the (unclamped) line-projection parameter

$$t^\star_{\mathrm{line}}(p) \;=\; \frac{\langle\, p - A\,,\; e\,\rangle}{\|e\|^2} \quad \text{with} \quad \|e\|^2 = x_3^2 + (2H)^2.$$

The *segment* projection clamps this to $[0, 1]$:

$$t^\star(p) \;=\; \mathrm{clamp}\big(t^\star_{\mathrm{line}}(p),\, 0,\, 1\big), \qquad q(p) \;=\; A + t^\star(p)\, e.$$

The exact Euclidean distance from p to the segment is

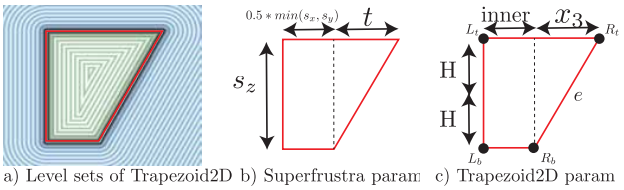
$$d_{\mathrm{right}}(p) \;=\; \|\, p - q(p)\,\|\,.$$

Sign from an oriented half-edge. For an oriented edge $[A, B]$ with direction $e = B - A$, define

$$c_{A\to B}(p) \;=\; \det\big(p - A,\, e\big) \;=\; p - A_x\, e_y - p - A_y\, e_x.$$

Geometrically, $\frac{c_{A\to B}(p)}{\|e\|}$ is the *signed* distance from p to the *infinite line* through $[A, B]$, positive on the left side of the oriented edge and negative on the right. To make the polygon interior correspond to "non-positive", we orient edges accordingly and, if needed, flip the sign (multiply by -1).

We collect the four signs; p is inside the trapezoid iff all four signs are non-positive.



a) Level sets of Trapezoid2D b) Superfrustra param c) Trapezoid2D param

Figure 5. Trapezoid2D. Level sets and parameterization: s_z controls the height, while t controls the amount of tapering (*i.e.*, how slanted the right edge is).

Magnitude. We compute the four segment distances via projection:

$$\begin{aligned} d_{\mathrm{right}}(p) &= d\big(p,\, [R_b, R_t]\big), \\ d_{\mathrm{left}}(p) &= d\big(p,\, [L_b, L_t]\big), \\ d_{\mathrm{bottom}}(p) &= d\big(p,\, [L_b, R_b]\big), \\ d_{\mathrm{top}}(p) &= d\big(p,\, [L_t, R_t]\big), \\ d_{\min}(p) &= \min\{d_{\mathrm{left}}, d_{\mathrm{right}}, d_{\mathrm{bottom}}, d_{\mathrm{top}}\}. \end{aligned} \tag{7}$$

Sign. With the orientations/sign flips fixed above, set

$$\begin{aligned} &\mathrm{inside}(p) \iff C(p) \\ &\quad C(p) = \max\{c_L(p), c_S(p), c_B(p), c_T(p)\} \leq 0. \end{aligned}$$

Final expression.

$$\mathrm{sd}_{\mathrm{trap}}(p) \;=\; \mathrm{sign}(-C(p)) \;\cdot\; d_{\min}(p).$$

This yields the exact signed distance for the convex trapezoid (negative inside, positive outside). The same projection-and-half-space procedure applies *identically* to each edge.

2.3. Combining two 2D SDFs to define a 3D SDF

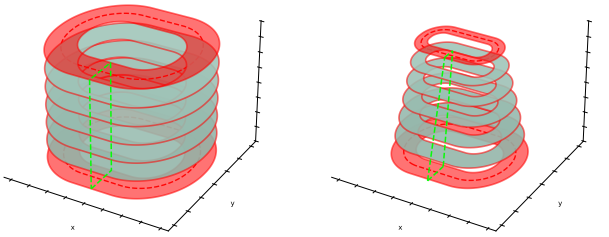
Remarkably, 2D signed–distance functions (SDFs) can be composed to form a 3D SDF, as in our parameterization:

$$\begin{aligned} \mathbf{s}_{2\mathrm{D}} &= \mathrm{RoundedRect2D}\big(\mathbf{p}^*_{xy};\; s_x, s_y,\; r\big)\,, \\ s^* &= \mathrm{Trapezoid2D}\big((\,\mathbf{s}_{2\mathrm{D}},\, \mathbf{p}^*_z\,);\; s_z,\; t,\; o\big)\,. \end{aligned}$$

We now provide an intuitive visualization of how this composition operates.

A simple example with two 2D rectangles. We begin with a toy example to introduce the concept of composing two 2D SDFs. Define *two* rectangles: (i) a *base* rectangle in the (x, y)-plane that defines the 2D signed distance (dashed red in Figure 6):

$$s(x, y) = \mathrm{sd}_{\square}\big((x, y);\; b_x, b_y\big), \quad \text{with half-sizes } (b_x, b_y),$$



(a) rectangle ∘ rectangle (b) Trapezoid2D ∘ RoundedRect2D

Figure 6. **Composition of 2D SDFs to form a 3D SDF.** The 0-level sets of the two 2D primitives are shown as dashed curves: red for the *base* XY primitive, and green for the vertical *gate* primitive. For each z-slice, the vertical primitive defines a band on the red primitive (constant here for simplicity). The 0-level set of the resulting 3D primitive is drawn in solid red (line on slices; filled on top and bottom caps).

and (ii) a *gate* rectangle in the (s, z)-plane, namely $[a, b] \times [-H, H]$, which selects a distance band $a \le s \le b$ and a vertical slab $|z| \le H$ (dashed green in Figure 6).

The 3D SDF for this volume is obtained by composing the two 2D SDFs:

$$\mathrm{sd}(p) = \mathrm{sd}_{\square}\big(\,(\,s(x,y) - m,\; z\,);\; w,\; H\big),$$

$$m = \tfrac{a+b}{2}, \quad w = \tfrac{b-a}{2}.$$

One can verify that $\mathrm{sd}(p) \le 0$ iff $a \le s(x,y) \le b$ and $|z| \le H$, with $\mathrm{sd}(p) = 0$ on the faces $s = a$, $s = b$, and $z = \pm H$.

Composing a rounded rectangle with a trapezoid. We now generalize the rectangle–rectangle composition in two ways. First, the *base* SDF in the (x, y)-plane can be any 2D primitive; in SuperFrusta, we use a rounded rectangle because it is instrumental for modeling conic shapes:

$$s(x,y) \;=\; \mathrm{sd}_{\mathrm{rrect}}\big((x,y);\; \mathbf{b},\; r\big), \qquad \mathbf{b} = (w,h).$$

Second, the *gate* in the (s, z)-plane is no longer a constant band $[a, b] \times [-H, H]$, but a *trapezoid* whose horizontal bounds vary linearly with height z. Let $H > 0$ denote the vertical half-extent, $\texttt{inner} \ge 0$ the bottom-left extent in s-space, $x_3 \in \mathbb{R}$ the top-right offset, and $\gamma \in [0, 1]$ the onion-thinning factor. With normalized height

$$t \;=\; \frac{z+H}{2H} \in [0,1] \quad (t = 0 \text{ bottom},\; t = 1 \text{ top}),$$

the z-dependent band becomes

$$\begin{cases} a(z) = L_\gamma(z) = -\texttt{inner}\,(1-\gamma) + (\gamma\, x_3)\, t, \\ b(z) = R_\gamma(z) = x_3\, t. \end{cases}$$

Each slice thus enforces

$$\text{inside} \iff a(z) \le s(x,y) \le b(z) \quad \text{and} \quad |z| \le H,$$

with slice width $b(z) - a(z) = (1-\gamma)\big(\texttt{inner} + x_3\, t\big)$. The resulting 3D SDF is obtained by evaluating the exact trapezoid SDF in (s, z):

$$\mathrm{sd}(p) \;=\; \mathrm{sd}_{\mathrm{trap}}\big(\,(\,s(x,y),\; z\,);\; \texttt{inner},\; H,\; x_3,\; \gamma\big).$$

2.4. Onioning.

In signed–distance modeling, onioning extracts a finite–thickness shell by thresholding an SDF symmetrically about its zero set. Let $d : \mathbb{R}^n \to \mathbb{R}$ be an exact SDF for $\Gamma = \{x \mid d(x) = 0\}$ and let $o > 0$ be a half–thickness. Define

$$d^{\mathrm{on}}(x) \;=\; |d(x)| - o. \tag{8}$$

This operator assigns a negative SDF to points *between the two isosurfaces* $\{x \mid d(x) = \pm o\}$.

Topology under onioning. Shadertoy. Consider a disk of radius $R > 0$ with signed distance $d(x) = \|x\| - R$. For half–thickness $o > 0$, the onioned band is

$$B_o \;=\; \{\,x :\, |d(x)| \le o\,\} \;=\; \{\,x :\, R-o \le \|x\| \le R+o\,\}.$$

There is a critical value $o^\star = R$ at which the band changes topology. If $o \ge R$, the inner bound collapses and

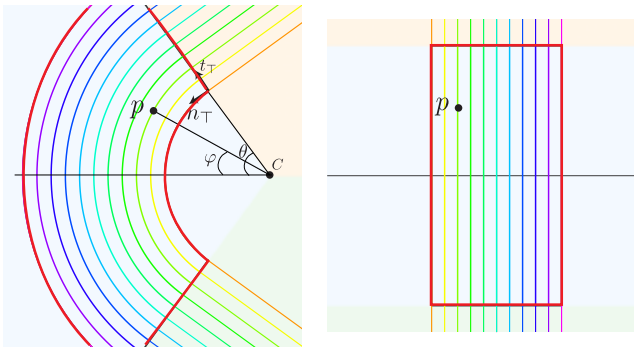
$$B_o \;=\; \{\,x :\, \|x\| \le R+o\,\},$$

i.e., a pure morphological dilation of the original disk (genus 0). If $0 < o < R$, then B_o is an annulus with inner radius $R-o$ and outer radius $R+o$ (a hole appears, genus 1). This illustrates that onioning can change the topology of a base shape, controlled continuously by o—a convenient parameterization that is harder to achieve with explicit primitives.

A naive approach to achieve the onion effect is to simply apply the onion computation on the 2D rounded rectangle sdf value, i.e.

$$\mathrm{sd}^{\mathrm{on}}_{\mathrm{rrect}} = |\,\mathrm{sd}_{\mathrm{rrect}}(\mathbf{p}_{xy};\; s_x, s_y,\; r)\,| - o \tag{9}$$

Since this 2D sdf function is used along along with the trapezoid function to create a 3D surface, the resulting sdf after this onion operation is very inaccurate – to the extent that sphere tracing with such a primitive definition results wrong renders. Furthermore, this design of the onion operator couples the inner "hole" creation with outer rounding operation. That is, the inner hole boundary ($-d(x) + o = 0$) exists in tandem with a dilation of the outer surface ($d(x) + o = 0$). Therefore, we instead have a formulation where a) the onion effect is achieved by



a) Source domain of the mapping b) target domain

Figure 7. DomainCurve3D. **(a)** *Source domain.* The sector is parameterized around the circle center C, with polar angle $\varphi \in [-\theta, \theta]$. Inside the sector, iso–offset curves (colored) are circular arcs about C; at the boundaries $\varphi = \pm\theta$ they meet the tangent frames with unit directions $\mathbf{t}_\top = (\sin\theta, \cos\theta)$ and normals $\mathbf{n}_\top$. **(b)** *Target domain.* Under $\mathbf{q} = \text{mapArcBulge}(\mathbf{p})$, concentric arcs map to vertical lines inside the strip $x' \in [-U, U]$, $y' \in [-\frac{z}{2}, \frac{z}{2}]$, while the tangent continuations remain straight outside the strip. Colors are preserved to track corresponding curves across panels. The red iso-surface of the bulged rectangle on the left maps to the rectangle on the right.

modifying the trapezoid parameterization, and (b) its only performed for the inner hole creation without affecting the positive sdf outside:

$$\begin{aligned} x_L &= -\,\texttt{inner}\,(1-o), \\ x_{TL} &= -\,\texttt{inner} + (x_3 + \texttt{inner})\,o, \\ y_B &= -h, \qquad y_T = h, \\ L_b &= (x_L,\, y_B), \qquad L_t = (x_{TL},\, y_T), \\ R_b &= (0,\, y_B), \qquad R_t = (x_3,\, y_T). \end{aligned} \tag{10}$$

2.5. Adding curvature

We now explain the last parameter, the curvature c, used to bend primitives with DomainCurve3D. The function is parameterized by the primitive height s_z and by the bulging intensity c.

The goal of the `mapArcBulge` transformation is to construct a smooth bijection between a *rectangular chart* in canonical coordinates (x', y') and a *bulged cap* region in the original domain (x, y). Intuitively, the mapping unwraps a circular arc into a straight vertical strip. Concentric circles around the arc's center correspond to vertical iso-x' lines, while the angular coordinate φ along the arc maps linearly to the vertical coordinate y'. Outside the sector delimited by the arc, the mapping transitions continuously to tangent-plane coordinates, ensuring differentiability across the boundary. In practice, the shader implements the *inverse transformation*: given a point $\mathbf{p} = (x, y)$ in the bulged domain, it computes its corresponding chart-space coordinate $\mathbf{p}' = (x', y')$.

Construction of the reference arc. Let $h = z/2$ be half the height and define the endpoints $E_\top = (0, +h)$ and $E_\perp = (0, -h)$. The circular arc through these points has center $C = (x_c, 0)$ on the x-axis and radius $R = \sqrt{x_c^2 + h^2}$. We set the half-opening angle

$$\theta \;=\; \max\!\big(c \times \tfrac{\pi}{2},\, \varepsilon\big),$$

where $c \in [0, 1]$ controls curvature: $c = 0$ yields a flat cap ($\theta \to 0$), while $c = 1$ produces a semicircular cap ($\theta = \pi/2$). By trigonometry,

$$x_c \;=\; \frac{h}{\tan\theta}.$$

The circular sector bounded by $\varphi = \pm\theta$ is the interior of the bulged region.

Inside-sector mapping. For points whose polar angle around C satisfies $|\varphi| \le \theta$,

$$\begin{aligned} y' &= \frac{h}{\theta}\,\text{clamp}\Big(\frac{\varphi}{\theta}, -1, 1\Big)\,, \\ x' &= \|\mathbf{p} - C\| - R. \end{aligned}$$

The first line linearly reparameterizes angle into vertical coordinate (clamped to $y' = \pm h$ at the endpoints), and the second converts radial offset to x', so the reference arc maps to $x' = 0$. Hence concentric arcs map to vertical iso-x' lines.

Outside-sector mapping. For $\varphi > \theta$ or $\varphi < -\theta$, we switch to a local tangent frame anchored at the top or bottom endpoint. At the top, with unit tangent and normal

$$t_\top = (\sin\theta, \cos\theta), \qquad n_\top = (-\cos\theta, \sin\theta),$$

(and analogously $t_\perp = (-\sin\theta, \cos\theta)$ at the bottom), we project

$$\text{along} = \langle \mathbf{p} - E, t \rangle, \qquad \text{perp} = \langle \mathbf{p} - E, n \rangle,$$

and define

$$(x', y') = \begin{cases} (\text{perp},\; +h + \text{along}), & \varphi > \theta, \\ (\text{perp},\; -h + \text{along}), & \varphi < -\theta. \end{cases}$$

This continues the mapping outside the arc by following tangent rays: horizontal displacements measure perpendicular offsets to the tangent, while the vertical coordinate advances along the tangent direction.

2.6. Our Contributions.

Prior "super primitives" have appeared in the demoscene and Shadertoy community [5–8], where the central goal is to achieve a large variety of analytic shapes using compact expressions suitable for real-time rendering. Our work shows that these formulations are not only expressive under this original objective, but—crucially—can also serve as *differentiably optimizable* primitives. We demonstrate that, with appropriate reformulation, this family of primitives becomes highly effective for inverse modeling, enabling stable gradient-based fitting while retaining compactness and editability.

Size-relative parameterization. A key contribution of our work is the introduction of *size-relative* parameterization for the taper, rounding, and onion factors. In prior formulations, these parameters are specified in absolute units; while this is acceptable for manual authoring, gradient descent during inverse modeling can drive the parameters toward values where the SDF becomes inaccurate or even ill-defined, causing optimization instability. Our relative formulation guarantees geometric validity across all optimization iterates, ensuring that the solid remains well-behaved and sphere-traceable for all parameter updates. This modification is critical for reliable differentiable fitting.

Arc-based bending via domain reparameterization. We introduce an analytic bending operation based on an *arc-based domain reparameterization* (Section 2.5). To our knowledge, no prior Shadertoy super primitive incorporates this arc-based mapping. This bending operation substantially expands the shape family expressible by a single primitive—enabling smooth axial curvature without sacrificing differentiability.

Lower-error onion operator. We also introduce a revised formulation of the onion (shell) operator that preserves the *accuracy* of the signed distance field (Section 2.4). Conventional onioning introduces SDF distortions that lead to rendering artifacts, incorrect gradients, and inconsistencies in the constructed surface—issues that become critical during inverse modeling. Our alternative formulation maintains correct distance values across the entire shell region, ensuring stable gradient flow and producing reliable geometry under subsequent constructive operations such as onioning and dilation.

On SDF accuracy and exact formulations. Our super primitive yields a accurate but not mathematically exact signed distance field. In practice this leads to only mild rendering artifacts, and these can be further suppressed by restricting the parameter ranges to regions where the SDF approximation remains most reliable.

For completeness, we explored an exact formulation in which the SDF is computed as the minimum distance to analytically partitioned surface patches. However, the tapered–rounded corner regions form canal surfaces whose exact distance queries require solving a quartic equation, making the method significantly more expensive. Moreover, this exact variant does not naturally accommodate onioning or bending. Given these limitations, we adopt the approximate—but efficient and stable—formulation presented in the main paper.

3. Residual Primitive Fitting (ResFit) Details

High-level overview. ResFit addresses the gap between top–down geometric analysis and bottom–up primitive optimization. Purely optimization-based methods tend to produce entangled assemblies, while analysis-only methods partition shapes without regard to the expressive limits of the primitive family. ResFit bridges these extremes by alternating between residual-based shape analysis and joint primitive optimization: each analysis step extracts regions from the unexplained residual, and each optimization step fits the growing assembly to maximize $\mathcal{O}$. By allowing these two phases to inform each other, the method produces assemblies that remain both compact and geometrically faithful, correcting over- and under-parameterization as the reconstruction progresses.

Procedure. At iteration k, the current residual volume is decomposed into regions using morphological shape decomposition (MSD). Each region seeds a new primitive, which are merged with the existing assembly. All primitives are then optimized together to maximize $\mathcal{O}$, with each primitive influenced by its local support. A pruning step then remove redundant primitives whose presence lowers $\mathcal{O}$. Finally, the pruned assembly defines a new residual, and the cycle repeats until the objective saturates or no further valid partitions can be extracted. This interleaved process enables the system to refine earlier primitives as new ones are added, yielding a coherent and compact final assembly.

Figure 8 visualizes ResFit sequences. The iterative loop progressively improves reconstruction quality while keeping the assembly compact: each round refines the residual, introduces new primitives only where needed, and re-optimizes the entire assembly. Importantly, the sequence does *not* form a hierarchy—primitives introduced early are re-adjusted whenever new primitives are added. This continual re-optimization enables the method to self-correct and converge toward a coherent, parsimonious assembly.

4. Decomposition & Initialization

4.1. Morphological Shape Decomposition (MSD)

MSD identifies thickness–homogeneous regions by analyzing progressively less–eroded level sets of the target SDF. Let $m = \min(T)$ denote the most negative SDF value, and let $\ell_1 < \ell_2 < \cdots < \ell_L$ be a linear sequence from m to 0. For each erosion level ℓ_j, we form the eroded

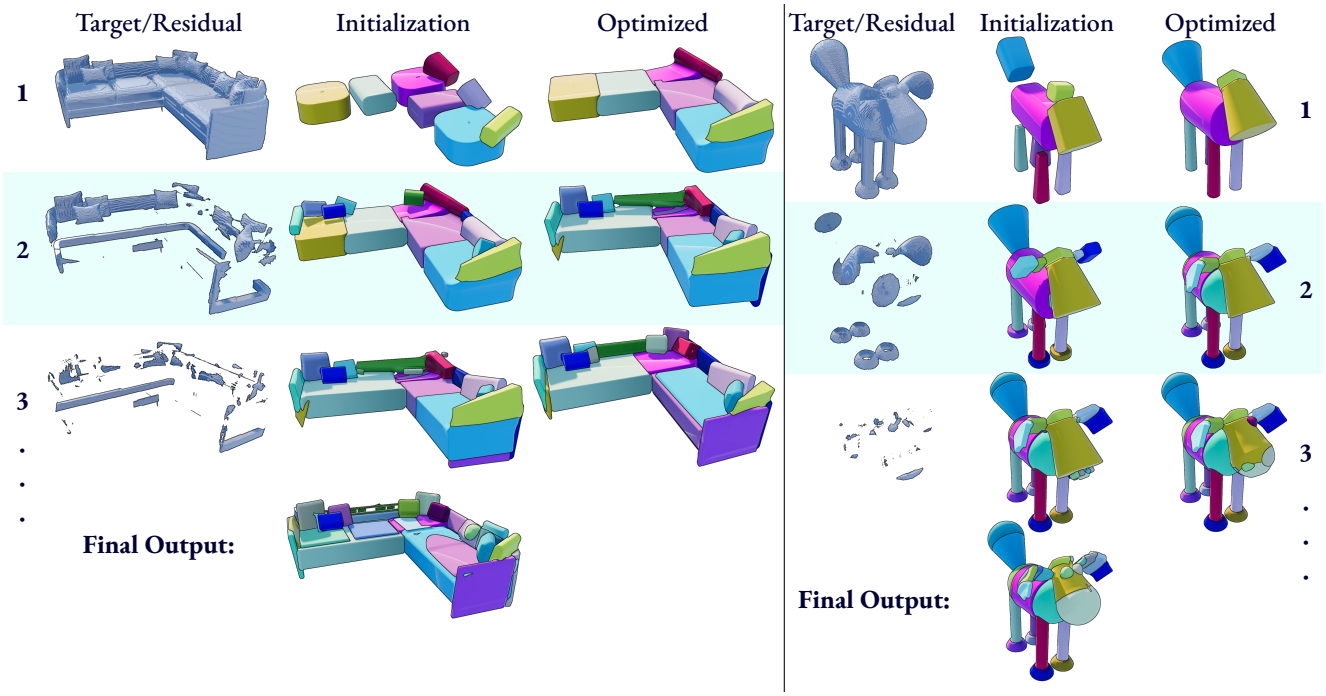


Figure 8. ResFit reconstruction sequences for a couch and a dog toy. Each row corresponds to a single round of ResFit. Each initialization phase introduces primitives required for the residual region, and each optimization phase jointly re-optimizing *all* existing primitives. This full-assembly refinement enables ResFit to correct early under-fitting errors, progressively improving fidelity while carefully increasing the number of primitives. This process consequently converges to a accurate yet compact assembly.

Algorithm 1: Residual Primitive Fitting (ResFit): Interleaved Decomposition and Optimization

Input: Target SDF grid T; max iterations K; MSD thresholds $(v_{\min}, e_{\min})$; parsimony weight λ
Output: Final primitive assembly P

$P \leftarrow \emptyset$
$R \leftarrow T$
$\mathcal{O}_{\text{prev}} \leftarrow -\infty$
for $k \leftarrow 1$ **to** K **do**
  $\mathcal{C} \leftarrow \text{MSDDECOMPOSE}(R, v_{\min}, e_{\min})$
  if $\mathcal{C} = \emptyset$ **then**
    return P
  $P \leftarrow P \cup \text{INITPRIMITIVES}(\mathcal{C})$
  $P \leftarrow \text{OPTIMIZEPROGRAM}(P, T, \lambda)$
  $P \leftarrow \text{PRUNE}(P, T, \lambda)$
  $\mathcal{O} \leftarrow \text{OBJECTIVE}(P, T, \lambda)$
  if $\mathcal{O} \leq \mathcal{O}_{prev}$ **then**
    return P
  $\mathcal{O}_{\text{prev}} \leftarrow \mathcal{O}$
  $R \leftarrow \text{RESIDUAL}(P, T)$
return P

mask $M_{\ell_j} = \{x : R(x) \leq \ell_j\}$ and extract its connected components $\{C_i\}$. Components whose eroded support satisfies $|C_i| \geq S_{\text{erode}}$ are then dilated back toward the zero level of R to obtain morphological–opening candidates D_i; components meeting the post–dilation size requirement $|D_i| \geq S_{\text{part}}$ are retained as MSD parts. Once such a batch is obtained, these parts are subtracted from the current target, the SDF is redistanced using a fast–sweep [11] to restore sdf correctness, finally a small morphological opening removes residual artifacts. This procedure, detailed in Algorithm 2, is repeated until no further valid parts can be extracted, yielding a compact set of coherent regions that serve as initialization seeds for primitive fitting.

4.2. Initializing SuperFrusta

Once MSD yields volumetric partitions $\{V_i\}$, we initialize one SuperFrustum per partition as part of the INITPRIMITIVES$(\mathcal{C})$ step in Algorithm 1. For each V_i, we first sample points from its interior and compute a centroid-aligned PCA frame. Each candidate direction (in both orientations) is evaluated as a potential primitive axis.

For a selected axis direction Z, we project all points onto Z to obtain axial coordinates t, and slice the point set along this axis. For each slice, we compute the 2D covariance of the points in the plane orthogonal to Z, whose eigenvalues (σ_u^2, σ_v^2) represent cross-sectional variances along two orthogonal in-plane directions. These statistics yield three

Algorithm 2: MSDDecompose: Morphological Shape Decomposition

Input: Target SDF T;
n. erosion levels L; max iterations K;
min eroded size S_{erode}; min part size S_{part}.
Output: Set of MSD parts $\mathcal{P}$

$\mathcal{P} \leftarrow \emptyset$
$R \leftarrow T$
$m \leftarrow \min(T)$
$\{\ell_j\}_{j=1}^{L} \leftarrow \text{linspace}(m, 0, L)$
for $k \leftarrow 1$ **to** K **do**
  $\mathcal{D} \leftarrow \emptyset$
  for $j \leftarrow 1$ **to** L **do**
    $M_{\ell_j} \leftarrow \{x : R(x) \le \ell_j\}$
    $\{C_i\} \leftarrow$ connected components of M_{ℓ_j}
    `// Eroded size condition`
    $\{C_i\} \leftarrow \{C_i : |C_i| \ge S_{\text{erode}}\}$
    if $\{C_i\} \neq \emptyset$ **then**
      `// Dilate parts`
      $\mathcal{D} \leftarrow$ dilations of $\{C_i\}$ toward R
      `// Part size condition`
      $\mathcal{D} \leftarrow \{D_i \in \mathcal{D} : |D_i| \ge S_{\text{part}}\}$
      if $\mathcal{D} \neq \emptyset$ **then**
        $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{D}$
        break
  if $\mathcal{D} = \emptyset$ **then**
    return $\mathcal{P}$
  `// Residual update`
  $R \leftarrow \textsc{SubtractParts}(T, \mathcal{P})$
  $R \leftarrow \textsc{RenormalizeFastSweep}(R)$
  $R \leftarrow \textsc{CleanUpMorphOpen}(R)$
return $\mathcal{P}$

scalar measures: (1) *stability* (S_{stab}), capturing how consistent the cross-sections are along Z; (2) *circularity* (S_{circ}), capturing how similar the in-plane variances are (i.e., how round the section is); and (3) *elongation* (S_{elong}), capturing the normalized length of the region along Z. Given slice-averaged variances σ_u^2, σ_v^2 and axial range $t_{\min}, t_{\max}$, the cylindricity score for axis Z is:

$$
\begin{aligned}
S_{\text{stab}} &= 1 - \frac{\text{std}(\sigma_u^2, \sigma_v^2)}{\text{mean}(\sigma_u^2, \sigma_v^2) + \varepsilon}, \\
S_{\text{circ}} &= 1 - \frac{|\sigma_u^2 - \sigma_v^2|}{\sigma_u^2 + \sigma_v^2 + \varepsilon}, \\
S_{\text{elong}} &= \frac{t_{\max} - t_{\min}}{\sqrt{\sigma_u^2 + \sigma_v^2} + \varepsilon}, \\
\text{S} &= S_{\text{stab}} + S_{\text{circ}} + 0.1\, S_{\text{elong}}.
\end{aligned}
\tag{11}
$$

The axis with the highest score is chosen as the principal axis of the initialized SuperFrustum.

Once the principal axis is selected, the primitive's translation, rotation, and in–plane size parameters are obtained directly from the centroid and PCA frame of the partition's point cloud. We additionally estimate the remaining shape parameters using simple statistics computed in the local (u, v, t) coordinate system aligned with the chosen axis. *Roundness* is initialized from the uniformity of in–plane radii across cross–sections, giving higher values when the region is consistently circular. *Taper* is inferred from the relative change in in–plane extent between the lower and upper portions of the volume, capturing whether the shape widens or narrows along the axis. Finally, the *Onion* parameter is estimated from the ratio between inner and outer radii measured in narrow bands around the coordinate axes, providing an initial guess for how much the shape contains an inner "core" structure. Other parameters such as *Bulge*, *Dilation* and smooth union amount are initialized with a fixed initial value. We emphasize that all the parameters are continuously adjusted using gradient descent during optimization. Please refer to the code for more details.

4.3. Additional Remarks

We highlight two details that clarify how MSD interacts with the fitting loop.

MSD can yield multiple seeds per round. Each MSD iteration extracts all connected components that satisfy the thickness criteria for the current erosion level. Consequently, a single round may produce several volumetric partitions if multiple regions share similar thickness in the residual. Thus running N rounds of MSD does *not* imply obtaining exactly N primitives; the method can expand the seed set adaptively when the residual contains many similarly thick, disconnected regions. This behavior is essential for modeling complex shapes: even with as few as 5–10 rounds, ResFit can introduce dozens of primitives when needed (see the grapes example in Fig. 1), allowing reconstruction quality to scale with shape complexity without requiring many outer-loop iterations.

Relation to Marching Primitives [4] The initialization strategy used in Marching Primitives (MPS) can be interpreted as a restricted, single-iteration instance of MSD. At each round, MPS selects the deepest erosion level whose eroded region exceeds a minimum volume and directly initializes primitives from that eroded component. In MSD terms, this corresponds to taking a single erosion threshold, extracting the qualifying eroded region, and *not* performing the dilation step that restores the component's full morphological extent. Because MPS does not decompose the shape into multiple connected components at that erosion level, it behaves more like an optimization-based method driven by

residual depth. Nonetheless, its seeding process aligns with a simplified form of MSD.

5. Optimization Details

We provide additional clarification on the components of our decomposition-aware optimization strategy.

Two-phase stochastic dropout. Our optimization proceeds in two phases distinguished by the temperature parameter of the Gumbel–Softmax existence distribution. During the first phase, the temperature is held at 1.0, yielding a smooth relaxation that allows primitives to remain active while the assembly adjusts globally. Once progress saturates, we enter a second phase in which the temperature is gradually annealed toward 0.1, sharpening the distribution and forcing the optimizer to make discrete keep-or-delete decisions for each primitive.

Stochastic preconditioning. At the beginning of each phase, we apply stochastic preconditioning [2]. For the first third of the iterations of that phase, we add diminishing Gaussian noise to the input sample points, reducing it linearly to zero by the one-third mark. This strategy improves convergence by smoothing the loss landscape early on and reducing local trapping, particularly in the presence of complex residual geometry.

Periodic surface resampling. To ensure fresh and accurate near-surface supervision, we resample both surface points and surface-adjacent perturbed points every $k = 100$ iterations. Occupancy queries for these points are computed using a BVH-accelerated signed-distance evaluation [9], providing robust ground-truth supervision throughout the optimization trajectory.

In total our loss for reconstruction can be written as:

$$\mathcal{L}_R = \mathcal{L}_{\text{vol}} + \mathbf{w} * \mathcal{L}_{\text{surf}} + \mathbf{w} * \mathcal{L}_{\text{adj}}, \tag{12}$$

where, $\mathcal{L}_{\text{vol}}$ stands for occupancy loss on points sampled uniformly in $\mathbb{R}^3$ (for each voxel, typically in 64^3 resolution from -1.0 to 1.0, $\mathcal{L}_{\text{surf}}$ stands for loss on points on the surface of the target mesh, where the loss forces the signed distance value to be equal to 0, and $\mathcal{L}_{\text{adj}}$ stands for occupancy loss on points sampled around the mesh surface. Additionally, $\mathbf{w}$ stands for the mesh curvature based weights.

Curvature-aware reconstruction weighting. To emphasize thin and high-curvature structures during optimization, each surface sample $\mathbf{p}$ is assigned a curvature weight $w(\mathbf{p})$ derived from the principal curvatures of the target mesh. We first estimate a multiscale curvedness value $C(\mathbf{p})$ by smoothing the raw per-vertex curvedness through heat diffusion at several scales and averaging the results. This produces a stable curvature estimate that suppresses spurious high-frequency spikes. To bound the influence of curvature and avoid extreme weights, we map $C(\mathbf{p})$ through a sigmoid function:

$$w(\mathbf{p}) = 1 + \sigma(\alpha\,(C(\mathbf{p}) - C_0))\,,$$

where $\sigma(\cdot)$ is the logistic sigmoid, α controls the sharpness of the transition, and C_0 sets the curvature threshold at which weighting begins to increase. Thus $w(\mathbf{p}) \in [1, 2]$, providing a modest but reliable emphasis on geometrically salient regions. The same weighting is used both in the reconstruction loss and in the reconstruction component of the overall objective $\mathcal{O}$.

Bound-preserving reparameterization. Rather than optimizing primitive parameters directly, we optimize unconstrained variables and map them to valid parameter ranges via a smooth reparameterization. For a parameter with bounds $[p_{\text{low}}, p_{\text{high}}]$, we write

$$p = \tanh(x)\,R + C,$$

where $R = \frac{1}{2}(p_{\text{high}} - p_{\text{low}})$ and $C = \frac{1}{2}(p_{\text{high}} + p_{\text{low}})$. This guarantees that parameters remain within their prescribed domain throughout optimization and improves numerical stability.

Further details. We refer readers to our code release for complete reproducibility.

5.1. Hyperparameters

For all experiments, we use a consistent set of hyperparameters governing optimization, sampling, and temperature scheduling. We optimize with a learning rate of 0.01 and run a maximum of 1600 iterations, with a base budget of 400 iterations and a saturation patience of 100 iterations for both triggering the second (temperature-decay) phase and finishing optimization. The Gumbel–Softmax temperature is fixed at $\tau_{\max} = 1.0$ during Phase 1 and annealed toward $\tau_{\min} = 0.1$ during Phase 2. To stabilize occupancy predictions, we apply a scale-factor schedule increasing smoothly from 10 to 15 over the course of optimization. The keep the losses local, we use loss banding, that is, the loss is only evaluated on points x where $\mathcal{F} < 0.05$.

Stochastic preconditioning uses an initial noise magnitude of $\sqrt{3} \times 0.02$, linearly reduced to zero over the first third of each phase. Every 100 iterations we resample 10^5 surface points and corresponding perturbed surface-adjacent points from the target mesh. Volumetric supervision is provided by uniformly sampling the SDF on a 128^3 grid.

All results reported in the paper and supplementary use these settings without per-shape or per-category tuning.

5.2. Pruning Routine

Our pruning strategy follows the general variant-sampling approach used in [1, 3]. After differentiable optimization converges, we generate candidate program variants and select the one that maximizes the objective $\mathcal{O}$.

Sampling via stochastic existence variables. Because each primitive has an associated Gumbel–Softmax existence variable, we can sample discrete program variants by adding fresh Gumbel noise to the optimized existence logits and converting the resulting probabilities into one-hot indicators (primitive kept or removed). We draw $K = 100$ such samples, evaluate $\mathcal{O}$ for each resulting assembly, and retain the best-scoring variant. This captures stochastic uncertainty from the optimization phase and can remove primitives that are only marginally supported.

Deterministic leave-one-out pruning. When the existence distributions have already sharpened, stochastic sampling alone is often insufficient. We therefore complement it with a deterministic leave-one-out procedure: for each primitive, we evaluate $\mathcal{O}$ on the assembly formed by removing that primitive. If any removal improves $\mathcal{O}$, we accept the corresponding reduced assembly. This process is applied recursively—each accepted deletion triggers a new pass over the remaining primitives—until no further improvement is possible.

Final selection. The highest-scoring assembly identified by either stochastic sampling or recursive leave-one-out pruning is returned as the final pruned program.

6. Applications

6.1. Inferring Canonical CSG Programs

We describe how the inferred primitive assembly can be adapted to recover programs composed only of canonical solids such as cuboids, cylinders, cones, and spheres. Directly constraining each primitive to one of these forms would disrupt the continuous optimization process that is central to our representation. Instead, we introduce a *Solid Super–Primitive*: a variant of SuperFrustum augmented with four additional logits that define a distribution over the canonical forms. During optimization, the primitive behaves as a probabilistic mixture of these canonical shapes, and gradually collapses to one of them when the temperature is annealed.

Canonical parameterizations. Note that SuperFrustum is parameterized with 8 parameters $\theta = \{s_x, s_y, s_z, r, d, t, c, o\}$ Each canonical solid is expressed as a special case of SuperFrustum by fixing or tying its parameters appropriately:

$$\begin{aligned}
\mathcal{F}_{\text{cube}} &= \mathcal{F}(\mathbf{p}; (s_x, s_y, s_z,\ 0, 0, 1, 0, o)),\\
\mathcal{F}_{\text{cone}} &= \mathcal{F}(\mathbf{p}; (s_x, s_y, s_z,\ 1, 0, 0, 0, 0)),\\
\mathcal{F}_{\text{cyl}} &= \mathcal{F}(\mathbf{p}; (s_x, s_y, s_z,\ 1, 0, 1, 0, o)),\\
\mathcal{F}_{\text{sphere}} &= \mathcal{F}(\mathbf{p}; (0, 0, 0,\ r, d, 1, 0, 0)).
\end{aligned}$$

Notably, allowing nonzero onion amounts for cubes and cylinders enables axis-aligned subtractive effects, which commonly arise in mechanical designs.

Solid Super–Primitive. Let $p_{\text{cube}}, p_{\text{sphere}}, p_{\text{cyl}}, p_{\text{cone}}$ be soft selection weights (from a Gumbel–Softmax distribution) over the four canonical solids, and let $\mathbf{s} = (s_x, s_y, s_z)$, r, d, t, c, and o denote the base size, roundness, dilation, tapering, bulge ratio, and onion ratio parameters of SuperFrustum. Instead of evaluating all four SDFs and mixing them in execution space, we construct a single *super–primitive* by blending these parameters in parameter space.

We first form a cylindrical size profile by averaging the in-plane extents,

$$s_{xy} = \tfrac{1}{2}(s_x + s_y), \qquad \mathbf{s}_{\text{cyl}} = (s_{xy}, s_{xy}, s_z),$$

and then define the blended parameters

$$\begin{aligned}
\mathbf{s}' &= p_{\text{cube}}\,\mathbf{s} + (p_{\text{cyl}} + p_{\text{cone}})\,\mathbf{s}_{\text{cyl}},\\
r' &= p_{\text{cyl}} + p_{\text{cone}},\\
d' &= p_{\text{sphere}}\,d,\\
t' &= p_{\text{cube}} + p_{\text{sphere}} + p_{\text{cyl}},\\
c' &= 0,\\
o' &= (p_{\text{cube}} + p_{\text{cyl}} + p_{\text{cone}})\,o.
\end{aligned} \tag{13}$$

The Solid Super–Primitive is then evaluated as a single instance of SuperFrustum with parameters $(\mathbf{s}', r', d', t', c', o')$. This parameter-space blending approximates a mixture-of-SDF formulation $p_{\text{cube}}\mathcal{F}_{\text{cube}} + p_{\text{sphere}}\mathcal{F}_{\text{sphere}} + p_{\text{cyl}}\mathcal{F}_{\text{cyl}} + p_{\text{cone}}\mathcal{F}_{\text{cone}}$, but avoids the $4\times$ cost of evaluating all four SDFs separately, making canonical-program inference computationally practical.

Temperature coupling and canonical collapse. The soft-selection distribution is governed by a temperature variable that we tie to the per-primitive existence temperature used in our two-phase optimization. During Phase 1, this temperature is held at 1.0, allowing the super-primitive to explore continuous interpolations among the canonical

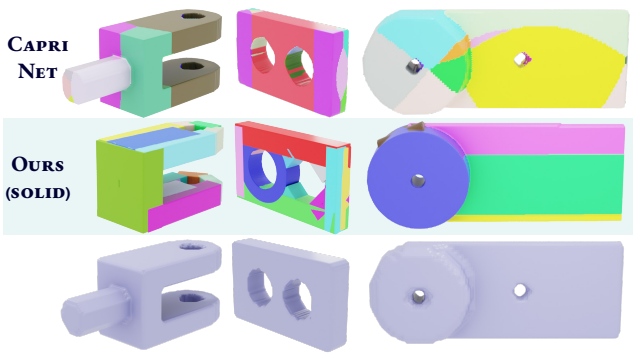


Figure 9. Failure cases arising from the lack of explicit subtractive operations in our assembly. Certain shapes—such as parts with cavities, through-holes, or cutouts—require true CSG subtraction to be represented exactly. As illustrated, ResFit approximates these structures using positive volumes only, which can reproduce the outer form but cannot capture internal voids or Boolean detail.

families. During Phase 2, it is annealed toward 0.1, forcing the mixture to collapse to a single canonical solid. Consequently, the final assembly contains only the four target canonical shapes, even though the underlying primitive family is far more expressive.

Limitations and subtractive structures. Because our assembly does not include explicit subtraction, the inferred canonical programs cannot express full CSG trees with complex Boolean structure. As shown in Fig. 9, parts with true subtractive operations may therefore be approximated but not reproduced exactly. Extending our framework to handle subtractive compositions is an exciting direction for future work.

6.2. Semantic Segmentation Enrichment

Most shape datasets provide only coarse semantic labels, leaving fine-grained part annotation both time-consuming and difficult for users. ResFit offers a practical alternative: by intersecting the coarse labels from PartObjVerse [13] with our inferred primitive assemblies (Fig. 10), we automatically obtain a set of fine, functionally meaningful subcomponents that remain strictly within each coarse semantic region. These automatically generated subparts can then be presented directly to annotators, reducing their task to simply *naming* or *grouping* the already-proposed segments rather than drawing masks. This illustrates how analytic decomposition can serve as a powerful structural prior for scalable, fine-grained part annotation.

6.3. Editable & Deployable Assets

Each inferred primitive serves not only as a geometric element but can also server as a texturable, editable primitive.

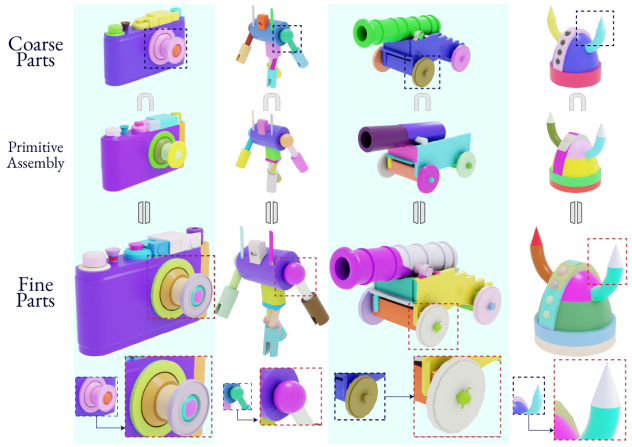


Figure 10. ResFit enables finer, semantically consistent part segmentation. Row 1 shows the coarse semantic regions provided in PartObjVerse. Row 2 shows the primitive assemblies inferred by ResFit. Intersecting each coarse region with its corresponding assembly (Row 3) yields meaningful sub-parts—capturing functional structure while remaining strictly within the original semantic boundaries.

To this end, every primitive $\mathcal{F}_i$ is assigned a *single 2D texture map* T_i defined in spherical coordinates. During sphere tracing, when a surface point $\mathbf{p}$ is hit on primitive i, we convert its local coordinates $\mathbf{p}_{\text{local}}$ into spherical angles and perform a texture lookup:

$$r = \|\mathbf{p}_{\text{local}}\|, \qquad \mathbf{d} = \frac{\mathbf{p}_{\text{local}}}{r},$$

$$\theta = \operatorname{atan2}(d_z, d_x), \qquad \phi = \arccos(d_y),$$

$$\mathbf{u} = \left(\frac{\theta}{2\pi} + \tfrac{1}{2}, \frac{\phi}{\pi}\right), \qquad \text{albedo} = T_i(\mathbf{u}).$$

This yields a stable, distortion-minimal parameterization for arbitrary primitive shapes, enabling consistent texture authoring and editing.

Materials under smooth union. While the geometry of two primitives is blended with a smooth-union operator, we do *not* blend their textures. Given two primitives with SDF outputs $f_1(\mathbf{p})$ and $f_2(\mathbf{p})$, the smooth union computes a blended SDF

$$f_{\text{su}}(\mathbf{p}) \;=\; \text{SmoothUnion}(f_1, f_2, \beta),$$

but the material is taken from whichever primitive attains the smaller SDF value:

$$\text{material}(\mathbf{p}) = \begin{cases} \text{mat}_1(\mathbf{p}) & \text{if } f_1(\mathbf{p}) < f_2(\mathbf{p}), \\ \text{mat}_2(\mathbf{p}) & \text{otherwise.} \end{cases}$$

This choice avoids texture mixing across primitives, producing cleaner, more interpretable textures while retaining smooth geometric transitions.

Together, these mechanisms allow the inferred primitive assemblies to act as lightweight, editable, and visually rich assets that can be deployed directly in real-time sphere-traced renderers. Next, we describe how these per-primitive texture maps are *inferred* by optimization to match a target textured mesh.

Texture inversion for deployable assets. Given a textured input mesh, we first run ResFit to obtain a geometric primitive assembly. We then equip each primitive with a learnable $128 \times 128 \times 3$ texture map. To supervise these textures, we mesh the primitive assembly via dual contouring on its SDF and uniformly sample points on the reconstructed surface. For each sampled point on primitive i, we find the closest point on the original textured mesh and assign its color as the target albedo for that location.

During optimization, the primitive geometry remains fixed; only the per-primitive textures are updated. Each surface point $\mathbf{p}_j$ is evaluated through the textured assembly, producing a predicted color $\hat{\mathbf{c}}_j$ via spherical texture lookup (Sec. 6.3). Let $\mathbf{c}_j$ denote the ground-truth color transferred from the mesh. Both are converted to Oklab using a fixed mapping $g(\cdot)$, and we minimize an ℓ_2 color loss:

$$\mathcal{L}_{\text{tex}} = \frac{1}{N} \sum_{j=1}^{N} \left\| g(\hat{\mathbf{c}}_j) - g(\mathbf{c}_j) \right\|_2^2.$$

To encourage spatial coherence and suppress high-frequency artifacts, we apply a standard total-variation penalty $\mathrm{TV}(T)$ on the texture maps, yielding the final objective

$$\mathcal{L}_{\text{mat}} = \mathcal{L}_{\text{tex}} + \lambda_{\text{tv}} \, \mathrm{TV}(T).$$

We optimize the texture parameters using the same stochastic-preconditioning framework as in primitive fitting, but with all geometric parameters frozen. This produces a textured primitive assembly that matches the appearance of the original asset while remaining compact, sphere-traceable, and fully editable.

7. Additional Experiments

7.1. Primitive Design Ablation

In the main paper we evaluated several alternative primitive families without smooth union. Here we extend this study by repeating the ablation with smooth union enabled for all families (Table 1). This isolates the geometric expressiveness of each primitive class from the benefits provided by smooth blending. The results confirm two trends observed earlier: (1) smooth union improves reconstruction fidelity and reduces overlap across *all* primitive families, and (2) even under this more favorable setting, SuperFrustum remains the strongest representation, achieving the highest reconstruction accuracy with competitive parsimony and low overlap.

	S. Union	IOU	CD	#Prims	Overlap
Cuboid	✗	82.81	0.208	24.26	0.333
S Q	✗	76.61	0.954	18.79	0.293
S P	✗	86.30	0.214	23.93	0.340
S F	✗	87.69	0.206	20.55	0.299
Cuboid	✓	84.46	0.128	23.88	0.340
S Q	✓	78.94	0.601	**17.96**	**0.171**
S P	✓	89.06	0.152	22.59	0.192
S F	✓	**89.97**	**0.144**	24.01	0.214

Table 1. We compare our full SuperFrustum against Cuboids, Superquadrics (SQ), and variants of the SuperPrimitive (SP). **SP** corresponds to SuperFrustum with tapering and bending disabled. Results are shown both with and without smooth union. Across all settings, SuperFrustum achieves the best reconstruction fidelity, while smooth union consistently improves performance for every primitive family.

7.2. ResFit Ablation

We now perform ablations to verify design decisions. We evaluate the variations in this sub-section on a subset of 250 Shapes from Toys4k dataset. As discussed in the paper, several design choices ensure that ResFit iterative loop can correct both over-parameterization, that is having more primitives than required, and under-parameterization, that is, having fewer primitives than required.

Table 3 isolates the impact of these design choices in ResFit. To avoid *underparameterization*, two components are essential. First, *local optimization* ensures that newly inserted primitives focus on the region that seeded them, leaving clean, fillable residuals for subsequent iterations. Removing this ("No re-opt") substantially lowers reconstruction due to primitives competing for the same geometry. Second, *global re-optimization* refines all existing primitives at every iteration instead of freezing earlier ones; omitting this step ("No freeze", shown as "Global Opt" versus the baseline) limits the system's ability to correct early mistakes and reduces overall fidelity.

To avoid *overparameterization*, we use a decomposition-aware objective that balances reconstruction with redundancy penalties. If we maximize reconstruction alone ("No prune"), the system indeed attains the highest IoU, but only by introducing nearly 30% more primitives—yielding assemblies that are far less compact. Our full method strikes the intended middle ground: high reconstruction quality while maintaining strong parsimony.

Effect of the number of MSD–seeded primitives per round. A key mechanism preventing over-

MSD Steps	Reconstruction Quality				Program Quality			
Per Round	IOU (↑)	BiSurfIOU (↑)	CD (↓)	EMD (↓)	#Prims (↓)	Overlap (↓)	IntraPrim (↓)	InterPrim (↑)
2	86.59	79.85	0.246	0.072	**17.49**	**0.155**	0.248	**0.278**
5	87.82	81.32	0.244	0.069	19.83	0.175	0.233	0.235
7	**90.26**	85.87	**0.148**	**0.066**	24.09	0.214	0.219	0.200
10	88.77	82.55	0.262	0.066	22.63	0.220	0.220	0.203
20	88.38	**86.24**	0.185	0.068	21.91	0.216	0.215	0.197
∞	88.46	82.41	0.318	0.073	28.49	0.242	**0.211**	0.184

Table 2. Ablation on the number of MSD-seeded primitives per round. Seeding too few primitives (e.g., 2 or 5) yields compact programs but can prematurely terminate the fitting loop, reducing reconstruction quality. Seeding too many (e.g., /∞) improves some reconstruction metrics but substantially worsens program quality due to over-parameterization. Moderate seeding (7 to 20) provides the best balance, achieving high fidelity while maintaining compact, low-overlap assemblies.

	IOU	#P	Overlap	IntraPrim	InterPrim
No re-opt	84.18	23.66	0.220	0.231	0.207
Global Opt	87.82	**19.16**	**0.165**	0.229	**0.222**
No Prune	**91.13**	29.02	0.213	**0.211**	0.182
Ours	89.92	23.98	0.213	0.220	0.201

Table 3. Ablation of the key components of ResFit. Local optimization and global re-optimization are both crucial for preventing underparameterization and achieving strong reconstruction. Conversely, removing decomposition-aware pruning ("No prune") increases IoU but at the cost of substantially more primitives, indicating overparameterization. Our full method provides the best balance of accuracy and parsimony.

parameterization in ResFit is that each MSD round contributes only a small number of new primitives rather than instantiating all detected volumetric parts at once. Table 2 evaluates this design by varying the number of primitives seeded per round. The results reveal a clear trade-off: seeding too *few* primitives (e.g., MSD–2 or MSD–5) produces more compact programs but can prematurely halt the fitting loop, leading to underfitting and lower reconstruction quality. Conversely, seeding too *many* primitives (e.g., MSD–40) improves reconstruction in some cases but yields substantially worse program quality—especially larger primitive counts and higher overlap—indicating over-parameterization. Moderate seeding levels (MSD–7 to MSD–20) strike the best balance, achieving strong reconstruction while maintaining compact and well-behaved assemblies.

MSD vs. CoACD Initialization. We compare our morphological shape decomposition (MSD) seeding strategy with CoACD—a widely used convex decomposition method—under matched budgets of decomposition steps (Table 4). Across all settings, both approaches yield broadly

	N Steps	IOU	CD	#Prims	Overlap
MSD	5	87.55	**0.249**	**19.67**	**0.174**
CoACD	5	**88.47**	0.307	21.32	0.224
MSD	10	**88.45**	0.260	**22.35**	**0.218**
CoACD	10	88.24	**0.243**	25.11	0.229
MSD	20	89.37	**0.182**	**21.94**	**0.215**
CoACD	20	**89.95**	0.216	25.14	0.231
MSD	∞	88.46	0.318	**28.49**	**0.242**
CoACD	∞	**89.61**	**0.242**	30.83	0.273

Table 4. Comparison of MSD and CoACD as decomposition-based initializers under matched step counts. Both methods achieve similar reconstruction accuracy, but MSD consistently produces more compact assemblies with fewer primitives and lower overlap. This supports MSD as a more structurally appropriate prior for primitive-based reconstruction.

similar reconstruction fidelity; however, MSD consistently produces *significantly more parsimonious* assemblies. At 5, 10, and 20 decomposition steps, MSD achieves comparable or superior Chamfer distance and markedly lower overlap while using 10–20% fewer primitives. These results highlight that MSD is a better structural prior for primitive-based reconstruction: it extracts coherent, thickness-homogeneous regions that align naturally with our primitives, whereas CoACD tends to over-fragment the shape. Overall, MSD offers a more compact, semantically aligned initialization without sacrificing reconstruction accuracy.

7.3. Optimization Loss Ablation

Losses on Program Length and Program Quality. We first ablate the two decomposition–aware terms that discourage over–parameterization: the program–length loss $\mathcal{L}_{\text{P}}$ and the primitive–quality loss $\mathcal{L}_{\text{Q}}$. Removing $\mathcal{L}_{\text{P}}$ yields

Method	Reconstruction Quality				Program Quality			
	IOU (↑)	BiSurfIOU (↑)	CD (↓)	EMD (↓)	#Prims (↓)	Overlap (↓)	IntraPrim (↓)	InterPrim (↑)
- $\mathcal{L}_{\text{Q}}$	89.63	85.00	0.145	0.066	20.45	0.248	0.226	0.208
- $\mathcal{L}_{\text{P}}$	**91.13**	**87.51**	**0.143**	**0.064**	29.05	0.213	**0.211**	0.182
- $\mathcal{L}_{\text{surf}}$	83.03	76.07	0.212	0.073	23.640	**0.161**	0.232	0.210
$w(p) = 1$	87.81	81.34	0.295	0.068	**20.01**	0.189	0.228	**0.222**
+ $\mathcal{L}_{\text{Tversky}}$	87.97	81.93	0.322	0.067	23.48	0.221	0.221	0.196
- Prune	89.73	85.16	0.151	0.066	22.74	0.194	0.223	0.219
Ours	89.92	85.66	0.154	0.067	23.67	0.208	0.221	0.202

Table 5. Ablation of our loss components. Removing program–length or program–quality losses dramatically alters the parsimony and coherence of the inferred assemblies (top block). Ablations of the reconstruction losses (middle block) show the importance of surface supervision, curvature weighting, and balanced inner/outer penalties. Removing hard pruning (bottom block) causes early saturation and weaker reconstructions. Overall, the full loss design achieves the best balance between fidelity and program quality.

	IOU	#P	Overlap	IntraPrim	InterPrim
No	89.49	28.56	0.294	0.219	0.187
Low	89.36	28.09	0.278	0.219	0.194
Med	**89.93**	24.00	0.213	0.219	0.201
High	83.97	**16.80**	**0.094**	0.240	**0.258**

Table 6. Ablation of jointly scaling the program–length and program–quality loss weights. Excessively small weights reduce program quality, while excessively large weights severely hurt reconstruction by over–regularizing the assembly. The medium setting (our default) provides the best balance between fidelity and structural compactness.

the highest reconstruction accuracy in the table, but does so by inflating the primitive count by nearly 30%, confirming that this term is essential for maintaining parsimony. Conversely, removing $\mathcal{L}_{\text{Q}}$ produces assemblies with significantly higher primitive overlap and weaker geometric coherence, even though the primitive count remains low. Together, these losses jointly regulate the structure of the inferred program, trading a small amount of reconstruction fidelity for a substantially cleaner and more compact assembly.

Losses Governing Reconstruction. We next ablate components of the reconstruction loss. Eliminating the surface–point loss $\mathcal{L}_{\text{surf}}$ leads to a sharp drop in both IOU and BiSurfIOU, showing that explicit surface supervision is critical for convergence. Replacing curvature–aware sample weights $w(p)$ with a constant $w(p) = 1$ also hurts performance—especially surface IOU—demonstrating that curvature weighting effectively guides optimization toward challenging geometric regions. We additionally evaluate a Tversky–style outer–vs.–inner reweighting (heavier penalties for false positives), similar to LightSQ, but find that it degrades performance: overly emphasizing exterior errors can prematurely shrink primitives before they have adapted to their local neighborhoods.

Pruning. Finally, we ablate the hard–pruning step used after optimization. Removing pruning does not degrade parsimony in the way one might expect; instead, it leads to *lower* reconstruction accuracy with slightly fewer primitives. This occurs because without pruning the objective saturates sooner, causing ResFit to terminate earlier and reducing the number of refinement rounds. Thus, pruning acts as a stabilizer that prolongs meaningful improvement and contributes to the overall reconstruction–quality–vs.–parsimony balance achieved by our full method.

Joint Scaling of Program–Length and Program–Quality Losses. We jointly vary the loss weights for program length (λ_{count}) and program quality (λ_{qual}), where λ_{qual} controls both the overlap and unoverlap terms. In the main paper we report $\lambda_{\text{count}} = 10^{-3}$ and $\lambda_{\text{qual}} = 10^{-2}$; the actual coefficients used during optimization are twice these values due to symmetric penalties applied to inside–outside consistency. Here, we sweep four regimes: *No* ($\lambda_{\text{count}} = \lambda_{\text{qual}} = 0$), *Low*, *Med* (our default), and *High*, corresponding to weight configurations ranging from 0 up to 2×10^{-2} and 2×10^{-1} for λ_{count} and λ_{qual}, respectively. This experiment evaluates how the relative strength of these two regularizers shapes the reconstruction–parsimony tradeoff.

Increasing these weights too aggressively (*High*) yields extremely compact assemblies with very low overlap, but at the cost of a large drop in reconstruction quality. At the opposite extreme, eliminating both terms (*No*) surprisingly does not improve reconstruction, while significantly degrading program quality—indicating that a mod-

Method	Reconstruction Quality				Program Quality			
	IOU (↑)	BiSurfIOU (↑)	CD (↓)	EMD (↓)	#Prims (↓)	Overlap (↓)	IntraPrim (↓)	InterPrim (↑)
Run 1	88.10	81.70	0.243	0.067	21.18	0.193	0.227	0.215
Run 2	88.03	81.67	0.248	0.067	20.98	0.186	0.227	0.223
Run 3	87.91	81.62	0.250	0.068	20.98	0.195	0.227	0.217
Avg	88.01	81.66	0.247	0.067	21.05	0.191	0.227	0.218
Std	0.078	0.032	0.003	0.000	0.094	0.004	0.0	0.003

Table 7. Statistical significance of evaluation metrics. We report reconstruction and program-quality scores across three independent runs, along with their mean and standard deviation. The extremely small variance across all metrics confirms that the quantitative results presented in the main paper are statistically reliable and not sensitive to run-to-run randomness.

Method	Reconstruction Quality			Program Quality				Timing
	IOU (↑)	BiSurfIOU (↑)	CD (↓)	#Prims (↓)	Overlap (↓)	IntraPrim (↓)	InterPrim (↑)	
MPS 64	71.46	51.68	2.207	9.338	0.472	0.355	0.332	7.9
MPS 128	80.60	72.75	1.147	30.62	0.588	0.245	0.201	37.9
MPS 256	86.30	79.93	0.740	85.76	0.738	0.196	0.103	194.6
Ours 1	81.13	73.83	0.798	9.354	0.113	0.288	0.326	84.8
Ours 2	86.55	80.62	0.231	15.54	0.145	0.241	0.248	184.1
Ours 3	88.28	83.08	0.167	19.19	0.168	0.230	0.223	290.6
Ours 5	89.67	85.25	0.149	23.01	0.203	0.222	0.204	551.2
Ours 10	89.91	85.68	0.147	23.93	0.213	0.220	0.201	692.6

Table 8. Comparison with Marching Primitives (MPS) across voxel resolutions (64, 128, 256). ResFit achieves a substantially better reconstruction–parsimony tradeoff: even with only two refinement iterations, our method attains reconstruction quality comparable to MPS–256 while requiring 5–6× fewer primitives and similar runtime. With additional iterations, ResFit surpasses all MPS variants across all reconstruction metrics while maintaining compact assemblies.

erate amount of structure regularization is beneficial for fitting. Our default *Med* setting achieves the best overall balance: strong reconstruction accuracy, reasonable primitive counts, and clean low–overlap assemblies.

7.4. Others

Statistical significance of reported metrics. Because our fitting procedure includes stochastic components—sampling of Gumbel variables, point–resampling, and stochastic preconditioning—it is important to verify that the evaluation metrics reported in the main paper are statistically meaningful rather than artifacts of a single run. To assess this, we execute the full pipeline three times on the same inputs and report the per-run scores together with their mean and standard deviation (Table 8). To lower the computational cost of this experiment, we conduct volumetric point sampling at 64^3 resolution instead of 128^3 resolution used for other experiments (consequentely, all the three runs have slightly lower reconstruction accuracy). Across all reconstruction metrics (IoU, BiSurfIoU, CD, EMD) and program-quality measures (#Prims, Overlap, IntraPrim, InterPrim), the variance is extremely small, indicating that the differences we observe across methods and ablations in the main paper are well above the noise floor. Thus, the metrics used throughout our comparisons can be interpreted as statistically stable and representative of the method's true performance.

Comparison to Marching Primitives Across Resolutions. Table 8 compares our method to Marching Primitives (MPS) evaluated at three voxel resolutions (64, 128, 256). As the MPS resolution increases, reconstruction improves but at the cost of substantially more primitives and rapidly growing runtime. In contrast, our ResFit iterations trace out a significantly better Pareto frontier: for a comparable number of primitives, our reconstructions exhibit markedly higher IOU, lower Chamfer distance, and stronger surface consistency. Notably, our 2–iteration setting (*Ours 2*) nearly matches or exceeds the reconstruction quality of MPS at *256* resolution while using *5–6× fewer primitives* and similar runtime. Our full 10–iteration model further pushes reconstruction quality well beyond MPS at any tested resolution. This highlights the efficiency and effectiveness of decomposition–aware fitting: high fidelity does not require ever-increasing voxel grids, but instead a sequence of informed primitive refinements.

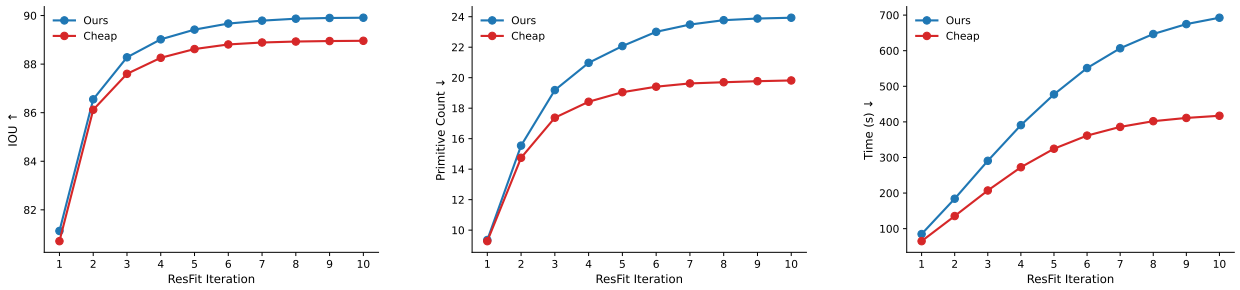


Figure 11. Comparison of our default fitting schedule (*Ours*) against the accelerated configuration (*Cheap*) across ResFit iterations. Left: IOU steadily improves under both settings, with our configuration achieving higher accuracy. Middle: Primitive count grows more aggressively under the full schedule, supporting higher reconstruction fidelity. Right: The accelerated regime reduces runtime substantially while maintaining competitive quality.

Method	Reconstruction Quality			Program Quality				Timing (↓)
	IOU (↑)	BiSurfIOU (↑)	CD (↓)	#Prims (↓)	Overlap (↓)	IntraPrim (↓)	InterPrim (↑)	
Cheap	89.05	84.14	0.160	**19.71**	0.222	0.227	**0.221**	**417.2**
Ours	**90.02**	**85.74**	**0.146**	23.87	**0.214**	**0.220**	0.199	692.6

Table 9. Effect of a cheaper optimization configuration. Reducing the number of iterations, increasing the learning rate, and using lower-resolution volumetric sampling significantly decreases runtime (from 692 s to 417 s) while maintaining comparable reconstruction and program-quality metrics. This shows that meaningful speed–accuracy trade-offs are achievable without major degradation in assembly quality.

Cheaper optimization regime. We also investigate whether the fitting procedure can be accelerated by relaxing several optimization hyperparameters. Specifically, we reduce the number of optimization iterations (from 400 to 250), lower the maximum iteration cap (from 1600 to 1200), increase the learning rate (from 0.01 to 0.015), and perform volumetric sampling at 32^3 resolution instead of 64^3. As shown in Table 9, these modifications reduce runtime by more than $1.6\times$ while preserving most of the reconstruction fidelity and structural quality of the inferred assemblies. Although the "Cheap" configuration yields slightly weaker performance than the full model, the gap remains small across all metrics, indicating that substantially faster fitting is possible with modest trade-offs. Exploring more principled acceleration strategies—such as adaptive schedules, coarse-to-fine decomposition, or learned initialization—presents a promising direction for future work.

To better understand how the accelerated configuration behaves over the course of the ResFit loop, Fig. 11 plots the evolution of IOU, primitive count, and runtime across iterations. Both schedules follow the same qualitative trajectory, but the cheaper regime progresses more quickly and saturates earlier, yielding slightly lower reconstruction accuracy and fewer primitives. Importantly, ResFit does not force a fixed number of refinement rounds: the loop continues only while the objective improves, and we merely raise the *maximum* iteration limit rather than enforcing all iterations. This highlights the trade-off—our full schedule achieves higher accuracy, while the accelerated variant offers a controlled speed–quality compromise for.

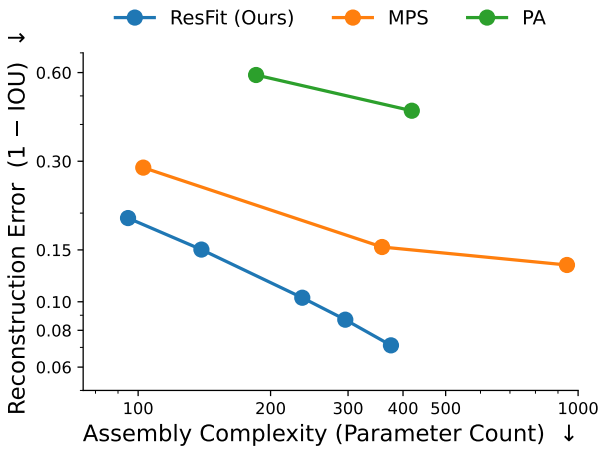


Figure 12. Reconstruction IOU versus total parameter count for Primitive Anything (PA), Marching Primitives (MPS), and our method. Despite using a more complex primitive, our approach yields the best accuracy for a given parameter budget, resetting the accuracy–parsimony Pareto frontier.

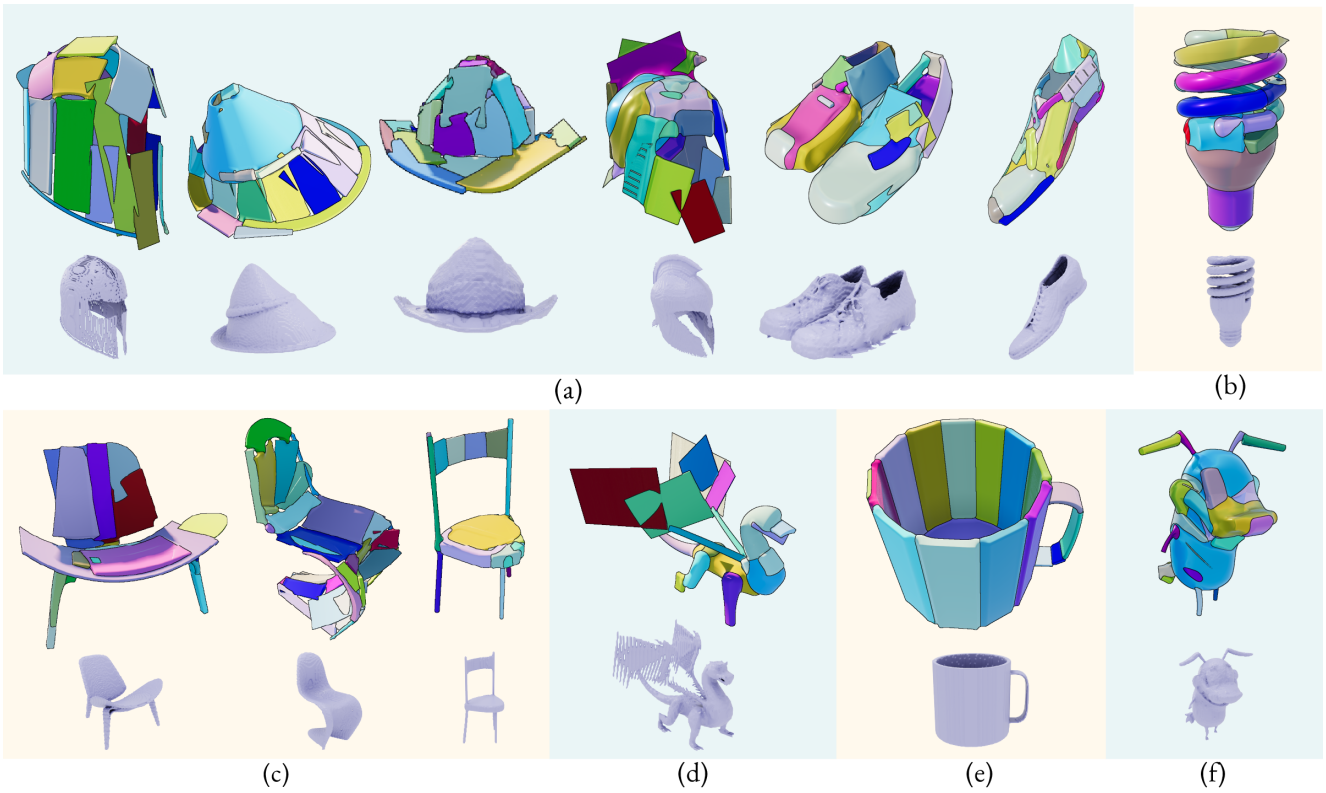


Figure 13. **Common Failure Cases** (a) Shell-like or hollow shapes (e.g., helmets, shoes) cannot be represented without explicit negative operations, leading to many primitives. (b) Multi-turn curved structures (e.g., LED filaments) fragment due to the single-arc primitive design. (c) Some complex shapes (e.g., chairs) require overlapping primitives when local geometry exceeds the primitive's expressiveness. (d) Extremely thin curved surfaces yield weak gradients and underfit geometry. (e) Irregular thickness can cause MSD to decompose hollow containers into thin slats, rather than a hollow cylinder. (f) In rare cases, primitives spill across semantic boundaries when needed to maintain fidelity.

Accuracy–Parameter Tradeoff. In the main paper we compare reconstruction accuracy against primitive count, but different primitive families vary substantially in the number of parameters they require: since Marching primitives (MPS) uses super quadrics, its primitives require 5 parameters each, PA uses 2–3 depending on type, while our SuperFrustum uses 8. To provide a fair cross-method comparison, Fig. 12 plots IOU against the *total parameter count* of each inferred program. Even here, our method establishes a new Pareto frontier: for any given parameter budget, SuperFrustum assemblies achieve higher fidelity than both MPS and PA, highlighting that our primitive is not only expressive but also parameter-efficient.

8. Failure Cases

While ResFit performs reliably across a broad range of shapes, we observe several characteristic failure modes (Fig. 13).

(a) Non uniform hollow or shell-like structures. Our primitive family cannot represent arbitrary hollow surfaces. Shapes such as helmets, hats, and shoes contain thin shells enclosing large voids; without explicit negative-space operations, ResFit approximates these with many primitives, producing assemblies that are geometrically valid but semantically unsatisfying. Richer support for subtractive geometry would address many of these cases, though deformable materials (e.g., cloth) may remain difficult regardless.

(b) Multi-arc or winding structures. Because each primitive contains only a single bending arc, it cannot capture long, continuously curved trajectories. In shapes such as LED filament spirals, the desired geometry has a multi-turn curve, but MSD splits it into multiple short thickness-consistent segments; the optimizer faithfully fits each, yielding many fragmented primitives. A multi-arc or curve-swept primitive family would resolve this limitation.

(c) Overlapping assemblies in complex but common categories. Even in standard categories such as chairs, we occasionally observe assemblies that achieve reasonable reconstruction but require a large number of overlapping primitives. This arises when local curvature cannot be matched well by the current primitive's shape space, suggesting the need for a more expressive base primitive.

(d) Extremely flat curved surfaces. When a part is both highly curved and extremely thin, the gradients associated with roundness, scale, or taper parameters become very small, leading to weak updates. As a result, surfaces such as thin bent panels such as the wings of the dragon may be improperly fit. Sampling points directly from a dual-contoured reconstruction of the target mesh would provide higher-quality supervision in such cases.

(e) Irregular thickness in hollow containers. We observe that for some hollow shapes, the ground truth contains inconsistent wall thickness. In such caese, MSD decomposes the shape into thin slats instead of recovering a coherent onioned cylinder. This occurs because thickness irregularity perturbs the morphological openings. Increasing MSD's robustness to noisy or highly variable thickness fields is an important future improvement.

(f) Semantic spillover across part boundaries. In a few cases, achieving high fidelity requires primitives whose spatial support crosses semantic boundaries—for example, a primitive intended for the cheek region may extend into the eye area. These issues could be mitigated by incorporating segmentation cues from multi-view renderings during optimization, nudging primitives to remain consistent with part structure.

9. Frequently Asked Questions (FAQs)

10. FAQ

Q. How does texture mapping work for onioned primitives?
For the texture-mapping example, we use spherical parameterization, indexing a 2D texture map by the polar angles (θ, ϕ). Under onioning, the inner and outer surfaces therefore share the same texture coordinates, and hence the same texture appearance. This simple scheme does not support assigning different textures or materials to the two sides of an onioned primitive.

Q. What smooth union operator do you use, and does the order matter?
We use the quadratic smooth minimum popularized by Inigo Quilez as our smooth union operator.

Q. How are the primitive assemblies rendered?
All geometry in our figures is rendered from the optimized smooth-union SDF; we do not use hard Boolean union for the surface itself. For visualization, however, we color each surface point using the ID of its closest primitive and add primitive-boundary outlines for clarity. Most results are rendered directly by sphere tracing the implicit field in GLSL. For cross-method comparisons, we instead extract a mesh from the zero level set and render it in Blender.

Q. Can a low-poly mesh be extracted directly from the primitive representation?
This is only straightforward when primitives are unsmoothed and non-dilated. In that setting, the geometry remains analytically sharp and can be converted more directly into a compact polygonal form. Once smooth union or dilation is introduced, the surface becomes a general implicit iso-surface, and meshing requires standard iso-surface extraction methods such as dual contouring.

References

[1] Aditya Ganeshan, R. Kenny Jones, and Daniel Ritchie. Improving unsupervised visual program inference with code rewriting families. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2023. 13

[2] Selena Ling, Merlin Nimier-David, Alec Jacobson, and Nicholas Sharp. Stochastic preconditioning for neural field optimization. *ACM Trans. Graph.*, 44(4), 2025. 12

[3] Hsueh-Ti Derek Liu, Maneesh Agrawala, Cem Yuksel, Tim Omernick, Vinith Misra, Stefano Corazza, Morgan Mcguire, and Victor Zordan. A unified differentiable boolean operator with fuzzy logic. In *ACM SIGGRAPH 2024 Conference Papers*, New York, NY, USA, 2024. Association for Computing Machinery. 13

[4] Weixiao Liu, Yuwei Wu, Sipu Ruan, and Gregory Chirikjian. Marching-primitives: Shape abstraction from signed distance function. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 1, 4, 11

[5] paniq. sdsuperprim. Shadertoy shader, 2016. https://www.shadertoy.com/view/Xdy3Rm. 9

[6] paniq. sdsuperprim. Shadertoy shader, 2016. https://www.shadertoy.com/view/MsVGWG.

[7] paniq. sdsuperpill. Shadertoy shader, 2016. https://www.shadertoy.com/view/MdK3RW.

[8] Inigo Quilez. 3d sdfs — distance functions. https://iquilezles.org/articles/distfunctions/, n.d. 9

[9] Alex Yu Shen. cubvh: Cuda-accelerated bounding volume hierarchy. https://github.com/ashawkey/cubvh, 2023. Accessed: 2025-02-18. 12

[10] Stefan Stojanov, Anh Thai, and James M. Rehg. Using shape to categorize: Low-shot learning with an explicit shape bias. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1798–1808, 2021. 1

[11] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable signed distance function rendering. *Transactions on Graphics (Proceedings of SIGGRAPH)*, 41(4):125:1–125:18, 2022. 10

[12] Yuhan Wang, Weikai Chen, Zeyu Hu, Runze Zhang, Yingda Yin, Ruoyu Wu, Keyang Luo, Shengju Qian, Yiyan Ma, Hongyi Li, Yuan Gao, Yuhuan Zhou, Hao Luo, Wan Wang, Xiaobin Shen, Zhaowei Li, Kuixin Zhu, Chuanlang Hong, Yueyue Wang, Lijie Feng, Xin Wang, and Chen Change Loy. Light-sq: Structure-aware shape abstraction with superquadrics for generated meshes. In *SIGGRAPH Asia*, 2025. 1, 4

[13] Yunhan Yang, Yukun Huang, Yuan-Chen Guo, Liangjun Lu, Xiaoyang Wu, Edmund Y Lam, Yan-Pei Cao, and Xihui Liu. Sampart3d: Segment any part in 3d objects. *arXiv preprint arXiv:2411.07184*, 2024. 1, 14

[14] Jingwen Ye, Yuze He, Yanning Zhou, Yiqin Zhu, Kaiwen Xiao, Yong-Jin Liu, Wei Yang, and Xiao Han. Primitiveanything: Human-crafted 3d primitive assembly generation with auto-regressive transformer, 2025. 1, 4