Computational Design of Shape-Aware Sieves

DAVID CHA, University of Southern California, USA
ODED STEIN, University of Southern California, USA

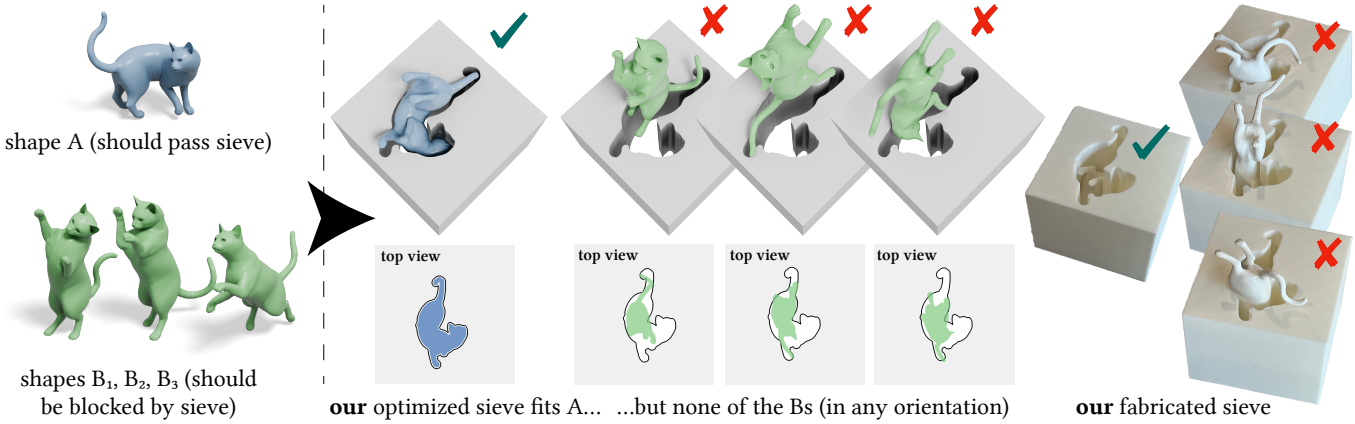


Fig. 1. Given a shape A that should pass through a sieve, and shapes B_1, B_2, B_3 that should be blocked by it, we compute a sieve hole that admits A and blocks the Bs. A ✓ means the sieve lets the shape pass, and a × means the shape does not fit. We fabricate the sieves to verify their properties (cf. supplemental video). Our method can also handle multiple A_is and accounts for various fabrication considerations.

We introduce mathematical tools to describe the geometric problem of *sieves*, two-dimensional holes that admit certain three-dimensional objects to pass through them, but block others. This is achieved by formulating the sieve design problem as a two-player game where both players (the one that wants to pass, and the one that wants to block) try to find a set of rigid transformations to achieve their objective. We also introduce an algorithm for solving this game by solving a global optimization problem employing both differentiable rendering with gradient-based optimization as well as particle swarm optimization. Our procedure accounts for real-world manufacturing concerns, and we fabricate a variety of examples demonstrating the practical viability of our sieves. Our implementation takes advantage of GPUs and does not rely on any clean or manifold input geometry as long as it is a triangle mesh. We can produce intricate sieves that block an arbitrary set of shapes $\mathcal{B}$ but admit another arbitrary set of shapes $\mathcal{A}$ (if finding a solution is possible for our method).

CCS Concepts: • **Computing methodologies** → *Computer graphics*; **Shape modeling**; • **Mathematics of computing** → *Mathematical optimization*; **Nonconvex optimization**.

Additional Key Words and Phrases: sieves,fabrication,global optimization

ACM Reference Format:
David Cha and Oded Stein. 2025. Computational Design of Shape-Aware Sieves. In *SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25), December 15–18, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3757377.3763875

Authors' Contact Information: David Cha, University of Southern California, Los Angeles, California, USA; Oded Stein, University of Southern California, Los Angeles, California, USA.

SA Conference Papers '25, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-2137-3/2025/12
https://doi.org/10.1145/3757377.3763875

1 Introduction

Sieves are an old tool for separating objects: Some objects pass through the sieve, and some do not. They are an important utensil in the modern world, from laypeople's kitchens to a variety of industrial applications [Advantech Manufacturing 2001], and the problem of designing sieves is both geometrically intriguing as well as practically relevant. In this article, we introduce a method for designing sieves that can be used to filter arbitrary objects using only the shapes' geometry. Leveraging the power of computational design, we can use sieves to distinguish between shapes of intricate geometries.

In its simplest form, the problem of designing a sieve is the problem of designing a two-dimensional hole that will let certain shapes pass in some correct orientation, and will not let other shapes pass no matter what orientation they are in. For industrial settings such as mining, users might focus on simple shapes and sizes for holes to separate particles by size and weight, combined with shaking techniques and other physics considerations. We focus purely on the *geometric* problem: what are the properties of a hole that certain shapes can be translated through, but others not? It turns out that there is an interesting mathematical theory hiding behind this simple question, which we study in depth in this article.

Actually solving the sieve problem raises many interesting algorithmic as well as practical questions. We propose a method for constructing a valid sieve that admits a set of arbitrary shapes $A_1, \ldots, A_m$ but will block arbitrary shapes $B_1, \ldots, B_n$ if such a configuration can exist. This algorithm works by solving a 2-player

game between player $\mathcal{A}$ and player $\mathcal{B}$, formulated as a global optimization problem. In an iterative process, the B_is use gradient-based optimization starting from a variety of initial configurations to optimize an overlap loss computed through differentiable rendering in order to orient themselves in a way that will fit through the hole. In an outer loop, the A_is use a particle swarm optimization routine to find a hole that will let them pass, but that the B_is cannot crack.

Our method specifically incorporates practical fabrication considerations like 3D printing tolerances in order to achieve a viable sieve that will perform in the real world (Fig. 1). We show the utility of our method by computing a wide array of specialized and geometrically intricate sieves (Figs. 5, 9, 8), and we fabricate sieves to show their viability (Fig. 15). Since it is built on differentiable rendering, our method is extraordinarily robust to bad input geometry. We can compute sieve holes for any triangle mesh that can be rendered; there is no requirement on manifoldness, connectedness, or watertightness.

2 Related Works

While sieves are widely used in engineering applications, and there exists practical research in designing sieves for filtering objects by simple shapes and sizes, the design of sieves is not well-studied in the geometry processing community. Mechanical engineers' study of sieves is concerned with the physical aspects of sieving and how they are affected by object size [Liu 2009; Liu et al. 2019] or the necessary motions of a sieving machine for optimal performance [Modrzewski et al. 2022]; see the literature review of Sanchez-Suarez et al. [2022] for an overview of sieving research in that field. By contrast, our work focuses purely on the geometry of the shape and the sieve hole, and on the question whether there is any orientation under which a shape can translate through a hole.

Shadow art. Designing sieves is closely related to the problem of *shadow art*: given one or more 2D outlines, find a 3D shape that, when projected onto a plane from a certain direction (i.e., casting a shadow), produces the 2D outlines.

Shadow art was studied, e.g., by Mitra and Pauly [2009], who use a voxel-based discretization of 3D shapes, and more recently by Sadekar et al. [2022]; Wang and Deng [2024], who use differentiable rendering. Our problem of finding sieve holes is related: We want to find a 2D outline so that certain 3D shapes fit through the 2D outline when projected (player $\mathcal{A}$), and certain shapes will never fit no matter how they are rotated before they are projected (player $\mathcal{B}$). Much like these recent approaches, we use differentiable rendering to optimize through an orthographic projection.

Further shadow art methods like Hsiao et al. [2018]'s focus on one-dimensional curves generated by wires (where other works in this section are two-dimensional). Min et al. [2017] introduce a method for grayscale shadow art (where other works produce solid shadows generated through full occlusion only). Other shadow art methods analyze anamorphic 3D shapes [Debnath et al. 2025] or shadows generated by hands [Gangopadhyay et al. 2023].

Optimizing for rigid transformation. Our problem is related to the packing problem, where shapes are rigidly transformed so they can fit in a certain bounded volume (in our case, so their projection can fit through the sieve hole). This is a well-studied problem in many disciplines, and we refer to various relevant surveys for an overview of the literature [Ali et al. 2022; Guo et al. 2022]. Of specific interest to us is the Matryoshka packing of Jacobson [2017]; we employ a similar particle swarm global optimization to find results in the complex and highly nonconvex space of orientations for our shapes $\mathcal{A}$. Other examples of recent geometry processing approaches to the packing problem are the approaches of Attene [2015]; Cui et al. [2023]; Xue et al. [2023]; Yang et al. [2023].

Another closely related field is path planning; finding a series of rigid transforms that can maneuver an object through certain obstacles without collisions. Examples of recent geometry processing work in this area are the methods of Joho et al. [2024]; Sellán et al. [2021]; Wang et al. [2024]; Zhang et al. [2020]; Zhao et al. [2023]; general surveys in robotics and continuous collision detection can be found in the works of Guo et al. [2023]; Nie et al. [2020]. We reduce the pathing in our sieve problem strictly to translations along the z-axis only—no complicated pathing is allowed.

Katz and Tal [2025], extending earlier works on point cloud visibility computation [Katz and Tal 2015; Katz et al. 2007; Mehra et al. 2010], optimize through their visibility method to find a rigid transform for an optimal viewpoint, similar to how we find an optimal rigid transform for fitting through a sieve hole.

Pure math. Pure mathematics has a variety of interesting subfields connected with our sieve problem; we want to highlight two examples here. Geometric tomography studies how to reconstruct shapes (or properties of shapes) from *tomographic* data such as projections and cross-sections [Gardner 2006]. Computational geometers study the polygon containment problem (whether one polygon can fit inside another) [Goodman et al. 2017, Chapter 30].

Differentiable rendering. In recent years, powerful differentiable renderers [Jakob et al. 2022; Jatavallabhula et al. 2019; Laine et al. 2020; Ravi et al. 2020] have appeared that can be easily integrated into standard machine learning and optimization environments (see the survey of Kato et al. [2020] for a more complete overview). We use Kaolin [Jatavallabhula et al. 2019] and Pytorch [Paszke et al. 2019] to compute orthographic projections of objects, and we differentiate through this projection to optimize our objective.

A popular use of differentiable rendering in geometry processing is to use an image-based loss based on a rendered image, and then differentiate that image with respect to some deformation of a mesh. This approach has been used in generative modeling of 3D shapes [Dinh et al. 2025; Gao et al. 2023; Kim et al. 2025; Poole et al. 2023; Wang et al. 2022], and of course in the generation of shadow art [Sadekar et al. 2022]. There are countless other uses of differentiable rendering in geometry and graphics, such as 3D reconstruction [Kerbl et al. 2023; Lombardi et al. 2019; Mildenhall et al. 2020], inverse rendering [Cole et al. 2021; Nicolet et al. 2021; Vicini et al. 2022; Zhu et al. 2022], and many more.

3 The geometry of sieves

We first define the basic language used to describe sieves.

Definition 3.1. A sieve hole is a simply connected region $H \subseteq \mathbb{R}^2$.

Definition 3.2. Let $M \subseteq \mathbb{R}^3$ be a solid object. We refer to its orthographic projection along the z-axis onto $\mathbb{R}^2$ as $\text{proj}(M)$.

A sieve hole H *admits* M if there exists some rigid transformation $R \in \text{SO}(3)$, $\mathbf{t} \in \mathbb{R}^2$ such that $\text{proj}(RM) + \mathbf{t} \subseteq H$, denoted as $M \in H$. If no such transformation exists, we say that H *blocks* M, denoted as $M \notin H$.

Admission and blocking can be used to define a mathematical relation.

Definition 3.3. Let $A, B \subseteq \mathbb{R}^3$ be two shapes. We define

$$A \preceq B \tag{1}$$

to mean that, for any sieve hole H with $B \in H$, we also have $A \in H$.

If $A \preceq B$ and $B \preceq A$, then we say $A \sim B$.

Proposition 3.4. *$\preceq$ is a partial order with $\sim$ as the equivalence relation.*

Proof. If $A \preceq B$ and $B \preceq C$, for any $C \in H$ we must have that $B \in H$ and hence $A \in H$, so $A \preceq C$, thus $\preceq$ is transitive.

Trivially $A \preceq A$, so $\preceq$ is reflexive. By definition of $\sim$, $\preceq$ is antisymmetric. □

Crucially, $\preceq$ is not a total (or complete) order, as there are examples where neither $A \preceq B$ nor $B \preceq A$. Since $\sim$ is reflexive, symmetric, and transitive, it is an equivalence relation. The relation $\sim$ exhibits a useful property:

Proposition 3.5. *$A \sim B$ iff the two shapes have the same set of projections across all orientations.*

Proof. See Supplemental Material A for the proof. □

However, $A \sim B$ does not imply A and B are the same shapes, as shown by the counterexample in the inset: The (convex) icosahedron and the (non-convex) hollowed-out icosahedron are different shapes, but look the same when projected onto $\mathbb{R}^2$ from any direction.

4 The Sieve Game

Having defined sieves, admissibility, and their relations to each other, we now define the central game that is the focus of this article: A contest between two players seeking to block and pass through a sieve hole, respectively.

Definition 4.1 (Sieve Game). Given a set of shapes $\mathcal{A} = \{A_1, ..., A_m\}$ that we wish to allow through a sieve and another set $\mathcal{B} = \{B_1, ..., B_n\}$ that we do not, the game objectives of the Sieve Game are:

- For player $\mathcal{A}$, to find a sieve hole H that admits all A_i while blocking all B_i.
- For player $\mathcal{B}$, to find a rigid transform that will allow passage of any B_i through the sieve hole H found by $\mathcal{A}$.

If such an H exists, we say that $\mathcal{A}$ *wins* and $\mathcal{B}$ *loses*, and vice versa if it does not exist.

Two-player games are sometimes formulated as maximin optimization problems [Nisan et al. 2007], and our game is particularly amenable to this. We start with $\mathcal{B}$s goal. Consider the expression

$$\text{area}(\text{proj}(M)) - \text{area}(\text{proj}(M) \cap H) \tag{2}$$

which equals the area of the projection of a 3D shape M not contained in the sieve hole H. The condition $\text{proj}(M) \subseteq H$ is equivalent to $\text{area}(\text{proj}(M)) - \text{area}(\text{proj}(M) \cap H) = 0$, and $\text{proj}(M) \nsubseteq H$ is equivalent to $\text{area}(\text{proj}(M)) - \text{area}(\text{proj}(M) \cap H) > 0$.

From now on, let M' refer to the shape M rigidly transformed by $R \in SO(3)$ and $\mathbf{t} \in \mathbb{R}^2$ (we ignore the technicality of $\mathbf{t}$'s dimension, since the shape will always be projected onto $\mathbb{R}^2$). For a given sieve hole H, the problem of finding a rigid transformation such that the projection of $B' = RB + \mathbf{t}$ is contained in H can be formulated as the minimization problem

$$\min_{R,\mathbf{t}} \quad \text{area}(\text{proj}(B')) - \text{area}(\text{proj}(B') \cap H)\,, \tag{3}$$

where, if the optimum of zero is attained, then B is admitted, and otherwise the optimum is positive, meaning B is blocked. Player $\mathcal{B}$ must thus solve the optimization problem (3) for all B_i.

What does player $\mathcal{A}$ do? They must predict any move by $\mathcal{B}$, and find a sieve hole that all the A_i can pass through, while the B_i are blocked; i.e., they want a hole H so that the result of 3 is positive for all B_i. This can be written as the *maximization* problem

$$\begin{aligned} \max_{H,R_A,\mathbf{t}_A} \min_{R_B,\mathbf{t}_B} \quad & \big(\text{area}(\text{proj}(B')) - \text{area}(\text{proj}(B') \cap H)\big) \\ \text{s.t.} \quad & \text{area}(\text{proj}(A')) = \text{area}(\text{proj}(A') \cap H) \end{aligned} \tag{4}$$

for only one shape A and one shape B, or, for the general case,

$$\boxed{\begin{aligned} \max_{H,\mathcal{R}_A,\mathcal{T}_A} \min_{\mathcal{R}_B,\mathcal{T}_B} \min_{i=1}^{n} & \big(\text{area}(\text{proj}(B_i')) - \text{area}(\text{proj}(B_i') \cap H)\big) \\ \text{s.t. } \forall i = 1,\dots,m \quad & \text{area}(\text{proj}(A_i')) = \text{area}(\text{proj}(A_i') \cap H)\,, \end{aligned}} \tag{5}$$

where $\mathcal{R}_A = (R_{A,i})_i$, $\mathcal{T}_A = (\mathbf{t}_{A,i})_i$, $\mathcal{R}_B = (R_{B,i})_i$, $\mathcal{T}_B = (\mathbf{t}_{B,i})_i$, $\text{proj}(A_i') = \text{proj}(R_{A,i}A_i) + \mathbf{t}_{A,i}$, and $\text{proj}(B_i') = \text{proj}(R_{B,i}B_i) + \mathbf{t}_{B,i}$. Our Sieve Game can be practically reduced to the maximin problem (5). $\mathcal{A}$ wins if the optimization result is positive, and $\mathcal{B}$ wins if it is 0.

The Sieve Game is a two-player, zero-sum game with infinite strategy sets where player $\mathcal{A}$ wants to maximize an objective value, and player $\mathcal{B}$ wants to minimize it [Nisan et al. 2007]. For our purposes, we are only interested in pure strategies, and not mixed strategies where probabilities are assigned to the possible decisions for each player: a straightforward global optimization of (5).

5 Constructing sieves

Solving (5) for arbitrary holes H is a difficult problem. We will now simplify the problem in steps to make it practically solvable, and to actually present an algorithm that can be used to solve some version of the Sieve Game. We will begin incorporating practical considerations.

5.1 One shape in $\mathcal{A}$ and one shape in $\mathcal{B}$

For the simpler case where $\mathcal{A} = \{A\}$ and $\mathcal{B} = \{B\}$, we start by simplifying $\mathcal{A}$s strategy to always set $H = \text{proj}(A')$. This automatically satisfies the constraint $\text{area}(\text{proj}(A_i')) = \text{area}(\text{proj}(A_i') \cap H)$. We are left with the unconstrained optimization problem

$$\max_{R_A} \min_{R_B,\mathbf{t}_B} \quad \text{area}(\text{proj}(B')) - \text{area}(\text{proj}(B') \cap \text{proj}(A'))\,. \tag{6}$$

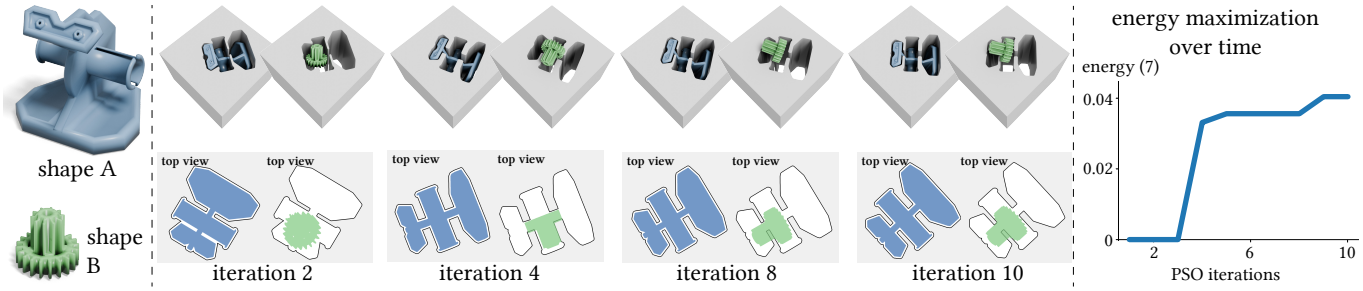


Fig. 2. The progression of our optimization procedure over time in an example with one shape in $\mathcal{A}$ and one shape in $\mathcal{B}$. The current best sieve at different iterations of our PSO procedure is on the *left*, and the energy (7) at the end of each PSO step is on the *right*.

Note that we no longer need the translation vector $\mathbf{t}_A$ for A due to the translation vector for B. Thus, in the simplified case it is just a matter of finding a rotation of A such that B is not admitted by its projection.

We find that the argument of the optimization problem is sensitive to the relative sizes of A and B: Larger shapes lead to larger projections, and hence larger objective values. This was particularly problematic in practice in the case where we have multiple shapes in $\mathcal{B}$. To accommodate for this, we normalize the argument by $\text{area}(\text{proj}(B'))$, so that it is always in $[0, 1]$:

$$\max_{R_A} \min_{R_B, \mathbf{t}_B} \quad \beta_{A'}(R_B, \mathbf{t}_B) \,, \tag{7}$$

where

$$\beta_{A'}(R_B, \mathbf{t}_B) = \frac{\text{area}(\text{proj}(B')) - \text{area}(\text{proj}(B') \cap \text{proj}(A'))}{\text{area}(\text{proj}(B'))} \,.$$

5.2 One shape in $\mathcal{A}$ and many shapes in $\mathcal{B}$

We now move on to the more complicated (but still not general) case where $\mathcal{A} = \{A\}$ and $\mathcal{B} = \{B_1, \ldots, B_n\}$. This can be handled with a simple extension of the inner optimization function β:

$$\beta^i_{A'}(R_{B,i}, \mathbf{t}_{B,i}) = \frac{\text{area}(\text{proj}(B'_i)) - \text{area}(\text{proj}(B'_i) \cap \text{proj}(A'))}{\text{area}(\text{proj}(B'_i))} \,. \tag{8}$$

As in (5), the introduction of many B_i turns our optimization into a minimization over the B_i. Using $\beta^i_{A'}(R_{B,i}, \mathbf{t}_{B,i})$, our simplification of (5) becomes

$$\max_{\mathcal{R}_A} \min_{\mathcal{R}_B, \mathcal{T}_B} \quad \beta_{A'}\left(\mathcal{R}_B, \mathcal{T}_B\right) \tag{9}$$

where

$$\beta_{A'}\left(\mathcal{R}_B, \mathcal{T}_B\right) = \min_{i=1}^{n} \beta^i_{A'}(R_{B_i}, t_{B_i}) \,, \tag{10}$$

and, as before, $\mathcal{R}_B = (R_{B,i})_i$ and $\mathcal{T}_B = (\mathbf{t}_{B,i})_i$.

In (9) an optimum of zero means A loses, otherwise the optimum is positive and A wins.

5.3 Many shapes in $\mathcal{A}$ and many shapes in $\mathcal{B}$

At last, we deal with the case where $\mathcal{A} = \{A_1, \ldots, A_m\}$ and $\mathcal{B} = \{B_1, \ldots, B_n\}$. A straightforward solution would be to use the technique for a single shape in $\mathcal{A}$ and many shapes in $\mathcal{B}$, and if a solution A'_i exists for each A_i, we can simply create a sieve hole that is the disjoint union of all the projections of the A'_i (a valid solution for the Sieve Game we devised). This is possible to do by simply optimizing (9). In this section, we will try to find more interesting holes H that are a single, connected silhouette. Note that this is not as simple as merely creating an overlapping union of the projections of the A'_i, as the result could be a shape that admits some of the B_i.

We start by noting that, regardless of how the sieve hole H is generated, finding the matching transformations for the B_i is similar to before (just now with a generic sieve hole H), by minimizing

$$\beta_H(\mathcal{R}_\mathcal{B}, \mathcal{T}_\mathcal{B}) = \min_{i=1}^{n} \beta^i_H(R_{B,i}, \mathbf{t}_{B,i}) \,, \tag{11}$$

where

$$\beta^i_H(R_{B,i}, \mathbf{t}_{B,i}) = \frac{\text{area}(\text{proj}(B'_i)) - \text{area}(\text{proj}(B'_i) \cap H)}{\text{area}(\text{proj}(B'_i))} \,. \tag{12}$$

We choose to construct H by finding optimal transformations for the shapes $\mathcal{A}$, $\mathcal{R}_\mathcal{A} = \{R_{A,1}, ..., R_{A,m}\}$ and $\mathcal{T}_\mathcal{A} = \{\mathbf{t}_{A,2}, ..., \mathbf{t}_{A,m}\}$ (we do not keep track of a translation for A_1, as this would simply translate the final sieve hole H itself). H is then defined as the union of the projection of A_is,

$$H = \bigcup_{i=1}^{m} \text{proj}(A'_i) \,. \tag{13}$$

Simply optimizing (5) with this hole H without any further constraints, $\max_{\mathcal{R}_\mathcal{A}, \mathcal{T}_\mathcal{A}} \min_{\mathcal{R}_\mathcal{B}, \mathcal{T}_\mathcal{B}} \beta_H(\mathcal{R}_\mathcal{B}, \mathcal{T}_\mathcal{B})$, leaves too much freedom and does not work well with our optimization infrastructure. We deal with this by maximizing an energy that promotes overlaps by adding a term to the energy that wants to minimize the area of H, and by adding an activation function that helps overlapping the different $\text{proj}(A'_i)$s:

$$\max_{\mathcal{R}_\mathcal{A}, \mathcal{T}_\mathcal{A}} \min_{\mathcal{R}_\mathcal{B}, \mathcal{T}_\mathcal{B}} \frac{\tau}{2} \left[\tanh \rho \left(\beta_H(\mathcal{R}_\mathcal{B}, \mathcal{T}_\mathcal{B}) - \eta\right)\right] - \alpha \, \text{area}\left(H\right) \,. \tag{14}$$

We choose $\tau = 10$, $\rho = 200$, $\eta = 0.02$, $\alpha = 5$. The $\text{area}(H)$ term promotes a more compact sieve hole, while the tanh activation function allows a trade-off between blocking the B_is and minimizing the area of H: After enough of B is blocked, no more can be gained by maximizing the blocked area, and it is easier to maximize the objective by reducing the area of H.

5.4 Fabrication considerations

The Sieve Game (5) and the different strategies for solving it discussed in Section 5 so far have completely ignored the practical need to fabricate H in a way so that shapes actually can pass through it

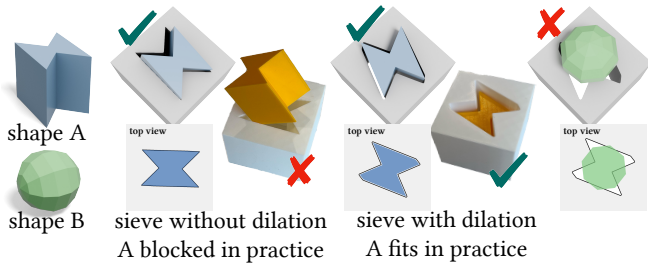


Fig. 3. Because of real-world properties like 3D printer manufacturing tolerance and printers, a perfect theoretical fit for A does not mean that A will actually fit in practice, hence we demand the sieve hole fits a dilated A. Cf. supplemental video.

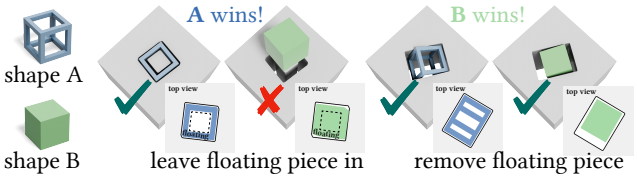


Fig. 4. We remove floating pieces in the sieve to make sure the sieve hole is a single simply connected shape that can be physically fabricated. This has effects on the game; in this example, A could win if we left the floating piece in, but cannot if the piece is removed.

and be blocked by it. Here, we quickly address modifications to our method that are necessary to accommodate the real world.

Offsetting the hole geometry. Even though in an ideal world A can perfectly pass through the hole $\text{proj}(A)$, this is not true in practice, because of friction, as well as manufacturing tolerances. Hence we *enlarge* H to allow for the A_i to physically pass through it. At first glance, it might seem like we only need to dilate the hole H resulting from any of the optimizations in this section. This naive approach could, however, lead to this now dilated hole admitting a shape B_i that it is supposed to block. Instead we dilate the shapes A_i at the start of any optimization, and search for holes that can admit the dilated shapes. That way, the optimization accounts for fabrication tolerances. By default, we dilate by 1 millimeter. Fig. 3 shows the effect of our dilation procedure and how it works.

Simple connectedness of sieve holes. We have, so far, ignored the statement of Definition 3.1 that the sieve hole H be simply connected. This property is vital for fabrication: We cannot produce a sieve hole that has a floating solid piece inside it, not connected to the walls of the sieve. We account for this during our method by, wherever $\text{proj}(A)$ is mentioned, filling in all disconnected outside components as part of the projection. Note that this might make certain games unwinnable for $\mathcal{A}$ that are winnable without this. Fig. 4 shows how our hole filling procedure works.

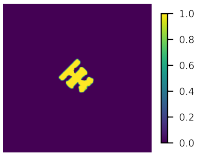
Accounting for complex motions. We assume that shapes are only translated along the z-axis in a fixed orientation through the hole and do not account for complex motions involving other translations and rotations. It may seem as if some blocked shapes can actually be wriggled through the sieve hole via complex motions, but making the sieve sufficiently thick (at least the diameter of the circumsphere of the shape) prevents this, as proven in Supplemental Material B (for space reasons, the full depth of the sieves is not always displayed in the figures).

6 Optimization methods

Our method, as described in Section 5, requires us to solve a variety of minimization and maximization problems. These problems are highly nonconvex and difficult to solve in general. Maximin and minimax optimization problems are often the focus of study in game theory [Nisan et al. 2007], but we do not use such existing algorithms as the literature is often focused on finding optimal probabilities for strategies to find the mixed Nash equilibrium of a game. Instead, we want to find out whether $\mathcal{A}$ can win and, if possible, what the winning configuration is.

We separate the optimization problems of Section 5 into two problems: the *inner minimization problem*, which we solve using gradient-based optimization, and the *outer maximization problem*, which we solve with a zero-order global optimization method.

6.1 Gradient-based optimization



We represent the A_is and the B_is by triangle meshes in $\mathbb{R}^3$. The projection operator proj is implemented via differentiable rendering. We render an orthographic raster of each mesh projection using binary values for the pixels. However, to guarantee the projection operator's differentiability for optimization, the actual rendering routine renders a soft mask smoothly transitioning from 0 to 1 at the boundaries (the inset shows an example of such a soft mask). We use a function in the differentiable rendering library Kaolin [Jatavallabhula et al. 2019] to render soft masks as defined by Chen et al. [2019]. All rasters were set to a resolution of 256×256 pixels. Computing $\text{area}(H)$ is then achieved by summing all pixel values. Terms that involve the intersection of two regions G and H, such as $\text{area}(G \cap H)$, are implemented as an elementwise multiplication of the per-pixel values of the binary rasters. These actions are all easily differentiable.

We use Adam [Kingma and Ba 2017] with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ at a learning rate of 0.05 for 100 iterations to find a minimum for all the inner minimization problems in Section 5. In order to make sure the minimization is not caught in local minima, we initialize $\mathcal{R}_B, \mathcal{T}_B$ with 10 random values drawn from a uniform distribution of $SO(3)$ and $[-0.1, 0.1]^2$ respectively. While more random values increases the chances of finding a global minimum, the algorithm has no theoretical guarantees, so when $\mathcal{B}$ can win, it may be incorrectly reported that it loses. But note that when it cannot win, it will never lead to incorrectly reporting that it does since an objective value of zero cannot be attained. We define the solution of this gradient-based minimization to be

$$f(\mathcal{R}_\mathcal{A}, \mathcal{T}_\mathcal{A}) = \begin{cases} \min\limits_{R_B, \mathbf{t}_B} \beta_{A'}(R_B, \mathbf{t}_B) \\ \min\limits_{\mathcal{R}_B, \mathcal{T}_B} \beta_{A'}(\mathcal{R}_B, \mathcal{T}_B) \\ \min\limits_{\mathcal{R}_\mathcal{B}, \mathcal{T}_\mathcal{B}} \frac{\tau}{2} \left[\tanh \rho \left(\beta_H(\mathcal{R}_\mathcal{B}, \mathcal{T}_\mathcal{B}) - \eta\right)\right] - \alpha \text{area}(H) \end{cases}, \quad (15)$$

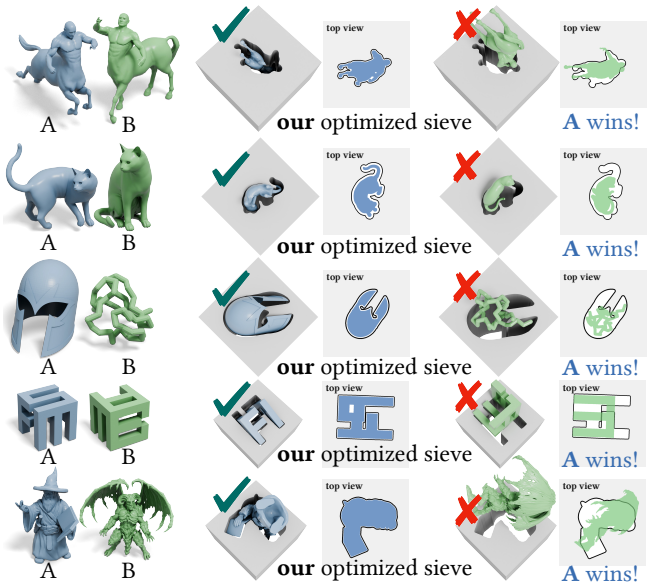


Fig. 5. Our sieves for the scenario with one shape in $\mathcal{A}$ and one in $\mathcal{B}$. A ✓ means the sieve lets the shape pass, and a × means the shape is blocked.

depending on which of the 3 problems (one shape in $\mathcal{A}$, one shape in $\mathcal{B}$; one shape in $\mathcal{A}$, many shapes in $\mathcal{B}$; many shapes in $\mathcal{A}$, many shapes in $\mathcal{B}$) we are solving. While these objective values are computed using soft masks at each iteration during optimization, the final value, which does not need differentiation, is computed using binary rasters as they give the exact projection.

6.2 Particle swarm optimization

It remains to solve the outer maximization problem by maximizing f, which we do using particle swarm optimization (PSO) [Poli 2008] with 10 particles, 10 iterations, inertia weight $\omega = 0.25$, and acceleration coefficients $\psi_1 = 0.25$, $\psi_2 = 0.25$. Fig. 2 shows how H evolves during the optimization.

For the case where $\mathcal{A}$ contains only one shape, straightforward PSO with no additional modifications can be used. If $\mathcal{A}$ contains multiple shapes, we employ the following approach to find a solution with sufficient overlap between the projection of all the A_i's:

- For each $A_i \in \mathcal{A}$, solve $\max_{R_{A,i}, \mathbf{t}_{A,i}} f\left(R_{A,i}, \mathbf{t}_{A,i}\right)$ as if we were solving the one-shape-in-$\mathcal{A}$-many-shapes-in-$\mathcal{B}$ case.
- Sort the A_i by the area of $\text{proj}(A_i')$, from largest to smallest.
- Set $H_1 = \text{proj}(A_1')$.
- For $i = 2, \ldots, m$:
  - Initialize 10 random translation vectors and rotation angles for transformations in $\mathbb{R}^2$ to apply to and, for each, use gradient-based optimization to maximize the area of the projection's overlap with H.
  - Using PSO initialized with particles from last step, solve
    $$\max_{R_{A,i}, \mathbf{t}_{A,i}} f\left(\{R_{A,1}, \ldots, R_{A,i}\}, \{\mathbf{t}_{A,2}, \ldots, \mathbf{t}_{A,i}\}\right) .$$
    to get an optimized transformed projection $\text{proj}(A_i^*)$ and then set $H_i = H_{i-1} \cup \text{proj}(A_i^*)$.

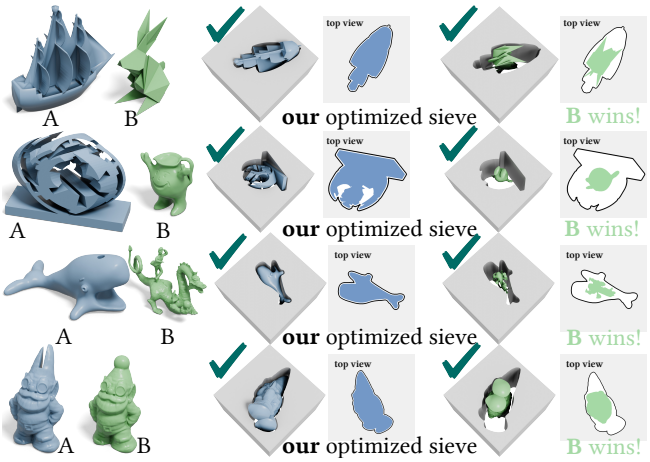


Fig. 6. Examples with one shape in $\mathcal{A}$ and one in $\mathcal{B}$, but where $\mathcal{A}$ cannot find a winning sieve. A ✓ means the sieve lets the shape pass.

- Finally, set $H = H_m$.

For the intermediate PSO steps, we use the same parameters as elsewhere, except for $\psi_1 = 0.05$, $\psi_2 = 0.05$.

This method computes the optimal orientations for the A_i one after the other. If, at the end, $f(\mathcal{R}_\mathcal{A}, \mathcal{T}_\mathcal{A}) = 0$, $\mathcal{B}$ wins. Else if $f(\mathcal{R}_\mathcal{A}, \mathcal{T}_\mathcal{A}) > 0$, $\mathcal{A}$ wins. By successively optimizing each orientation of A_i, we are able to ensure the simply connectedness of each intermediate sieve hole H_i and hence the final sieve hole H much better than optimizing all orientations at once.

Declaring a winner. After PSO is finished, we run the gradient-based inner optimization again to see how well $\mathcal{B}$ does against the sieve hole H chosen $\mathcal{A}$, but using 100 random initial orientations for $\mathcal{B}$ and 500 Adam iterations for significantly greater accuracy. If none of the B_i achieve a final energy of exactly zero (which is valid since there is no floating point error due to it being computed from binary rasters), then $\mathcal{A}$ wins.

7 Implementation & fabrication

We implement our method in Python using Pytorch [Paszke et al. 2019] and Kaolin [Jatavallabhula et al. 2019] for the gradient-based optimization and differentiable rendering. We use Gpytoolbox [Sellán et al. 2025] for common geometry processing tasks, particularly for dilating meshes by taking the signed distance field of a mesh on a grid and using marching cubes to reconstruct the offset surface. Our code and result files including all STL files of the input and output meshes are available at https://github.com/David-Cha/shape-aware-sieves.

To create the meshes for fabrication in a 3D printer, we take the binary raster of the output sieve hole H, and stack it to make a 3D raster of a prism with H as its base; this prism is at least as deep as the diameter of the largest circumsphere of all A_is and B_is, in order to make sure that shapes can only pass through H by translation along the z-axis, and not by rotation or xy translation. We then mesh the raster with marching cubes [Lorensen and Cline 1987], and do a mesh boolean between a rectangular block and the hole prism to

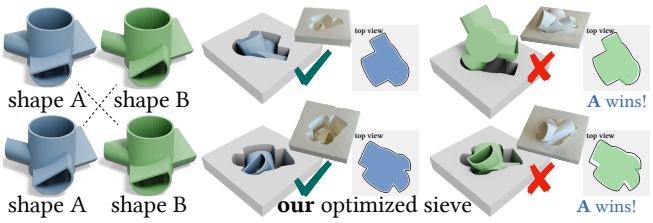


Fig. 7. Constructing sieves for two very similar input shapes alternately in the role of $\mathcal{A}$ and $\mathcal{B}$. Even though the two shapes are very similar, our method manages to construct sieves distinguishing between the two no matter which one is $\mathcal{A}$ and which one is $\mathcal{B}$.

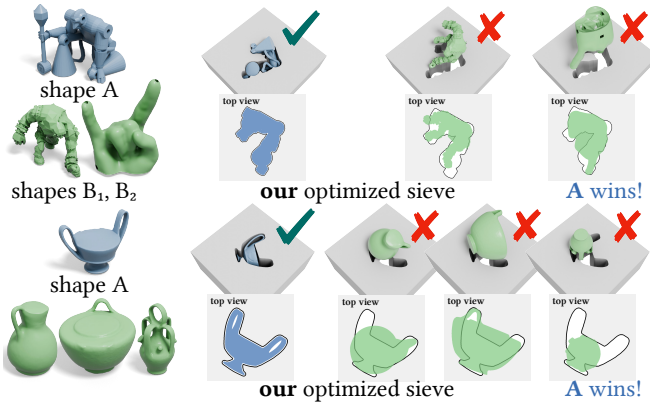


Fig. 8. Sieves produced with our method that try to admit a mesh A and block multiple meshes B_i. A ✓ means the sieve lets the shape pass, and a × means the shape does not fit.

get a mesh of the sieve object that can be printed in a 3D printer. We print both the shapes as well as the sieves using an UltiMaker S5 with PLA material or a Stratasys F370 with ABS material.

Further implementation details, such as parameter choices for certain experiments that deviate from the defaults, can be found in the supplemental material.

8 Results

We use our method to generate a large variety of sieves for diverse input geometries. We display the input shapes, the sieve geometry (both in 3D and 2D from above), the orientation for the A_is to pass through the sieve, and the orientation for the B_is that achieves the lowest energy on the inner minimization.

Figs. 5 and 6 show sieves computed for the case with one shape in $\mathcal{A}$ and one shape in $\mathcal{B}$. Our method determines that, in some cases, $\mathcal{A}$ can win (and produces the corresponding sieve), and that, in some bases, $\mathcal{A}$ cannot win (and the best possible sieve is produced). In Fig. 7 we demonstrate that our method can generate sieves to distinguish even extremely similar geometry: Of the two very similar-looking machine parts, our sieve only admits one, and blocks the other (for each choice of object we want to admit). Figs. 1 and 8 showcase our results for the scenario where we have one shape in $\mathcal{A}$ and multiple shapes in $\mathcal{B}$. Fig. 9 illustrates that the one-$\mathcal{A}$-many-$\mathcal{B}$ scenario is not the same as merely solving for the individual $B_i \in \mathcal{B}$,

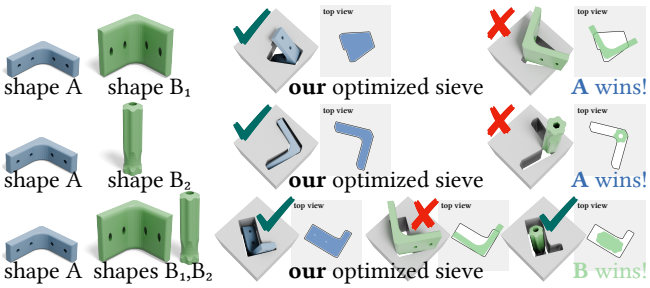


Fig. 9. Our method can generate a sieve admitting A and blocking B_1 by itself, blocking B_2 by itself, but not a sieve blocking both $\mathcal{B} = \{B_1, B_2\}$ at the same time (B_2 can pass).

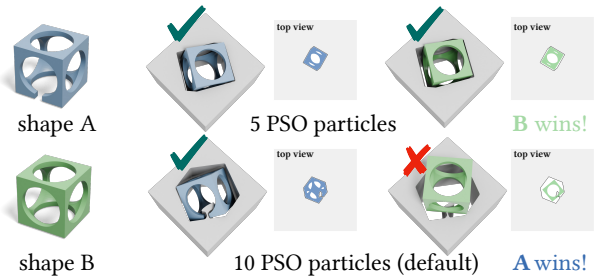


Fig. 10. Our PSO needs enough particles to find a good global optimum. By using 5 instead of than the default value of 10 for an example where $\mathcal{A}$ wins, it leads to $\mathcal{B}$ winning instead.

as here $\mathcal{A}$ could win against each of the meshes in $\mathcal{B}$ individually, but cannot block both of them at the same time. It is not enough to merely solve the pairwise problem of deciding whether a mesh A can block every mesh B individually. Our fabricated sieves can be seen in Figs. 1 and 15, and contain scenarios with one shape in $\mathcal{A}$ and one shape in $\mathcal{B}$, as well as multiple shapes in $\mathcal{A}$ and multiple shapes in $\mathcal{B}$. We verify in practice that the sieves admit the shapes they should be admitting, and block the shapes that they should be blocking. The examples in Fig. 15 with multiple shapes in $\mathcal{A}$ and multiple shapes in $\mathcal{B}$ show where our method truly shines: We can generate intricate sieve holes that pass many different shapes of arbitrary geometry, but block others.

Figs. 10, 11, 12, and 13 show the effect of varying the hyperparameters of our method. In Fig. 2 we show how the energy decreases over multiple iterations of PSO. Supplemental Material C features further runtime and convergence analysis of the method. A detailed table with runtime statistics for all figures in the article can be found in the supplemental material.

9 Limitations

Neither our inner gradient-based optimization (Figs. 11, 13) nor our outer PSO (Figs. 10, 12) guarantee that a global optimum is actually found, and thus we cannot guarantee that our method always correctly solves the Sieve Game. This can be sometimes remedied by increasing the various hyperparameters for our method: number

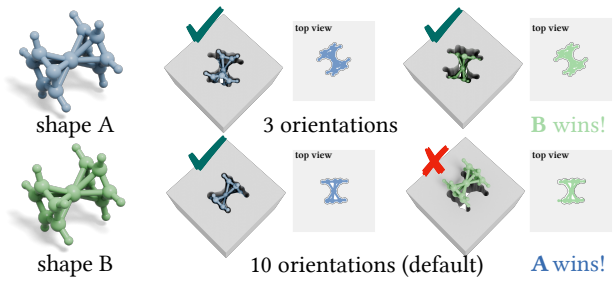


Fig. 11. Our inner optimization for $\mathcal{B}$ needs to be initialized with enough different initial orientations to find a correct solution. Using 3 instead of the default 10 for an example where $\mathcal{A}$ can win, leads to a $\mathcal{B}$ win instead.

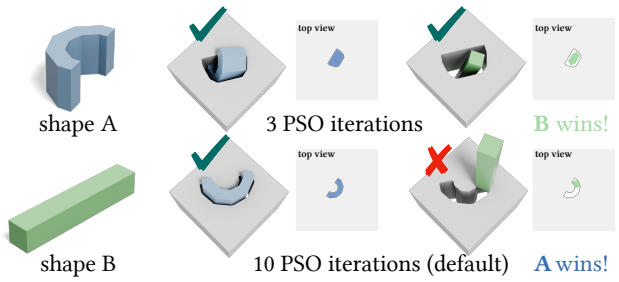


Fig. 12. The PSO in our method needs to be run for enough iterations. By using 3 instead of the default value of 10 for an example where $\mathcal{A}$ wins, it leads to $\mathcal{B}$ winning instead.

of particles, number of PSO iterations, number of $\mathcal{B}$ initializations, and number of Adam iterations.

Our procedure for declaring a winner of the game can result in a wrong call. In Fig. 14, we declare A a winner, even though B can pass through the sieve in practice. A more principled approach, taking into account further real-world tolerances and optimizing with enough power to achieve certainty is required to remedy this.

10 Conclusion

Our method can find holes for arbitrary configurations of $\mathcal{A}$ and $\mathcal{B}$ in many situations, enabling the fabrication of holes to filter shapes using geometry alone. By creating and solving an optimization formulation of this problem via differentiable rendering, we are able to robustly handle arbitrary triangle meshes as inputs. Moreover, we demonstrate the practicality of our method by accounting for fabrication considerations and physical manufacturing of sieves.

Our method cannot yet guarantee that the solution it finds blocks all shapes $B \in \mathcal{B}$, given the nonconvex nature of our optimization in $\mathcal{B}$. It is an interesting challenge for future work to combine our method with an algorithm that can prove efficiently that a certain shape cannot fit through a hole in *any* configuration. Potential future directions include partitioning the space of $SO(3)$ in a way that some optimization algorithm is guaranteed to find the global optimum or incorporating well-established geometry processing algorithms with theoretical guarantees like the iterative closest point algorithm [Besl and McKay 1992].

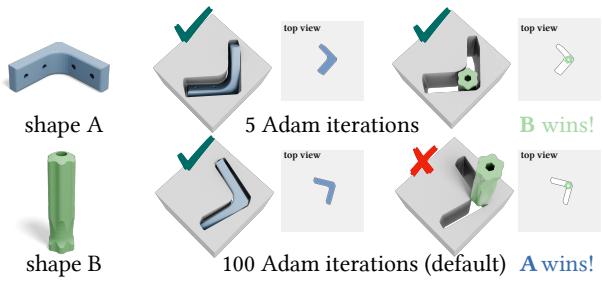


Fig. 13. The gradient-based Adam optimization needs to run for a certain minimum number of iterations. By using 5 instead of the default value of 100 for an example where $\mathcal{A}$ wins, it leads to $\mathcal{B}$ winning instead.

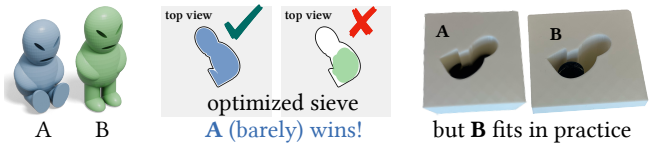


Fig. 14. It can happen that we determine that A has won, but B is blocked by such a tiny amount that, when printed, it still passes the sieve. This can sometimes be overcome by increasing the rasterization resolution or (similarly) scaling the meshes.

In this work we have also assumed that shapes can only be translated along the z-axis through the sieve hole, which is only valid when the sieve is very thick. If the sieve is thin enough, then shapes reported to be blocked may actually be able to pass via complex motions involving translations in the xy plane and rotations, so one has to account for more possibilities for shapes to fit through holes (similar to the work of Zhang et al. [2020]), an interesting direction for future work.

Lastly, our method is only applicable to the case where the shapes $A \in \mathcal{A}$ and $B \in \mathcal{B}$ can reliably attain their optimal orientation for fitting through the sieve hole. In sieves, as used in industrial settings or in the kitchen, objects are not precisely aligned with holes; rather, the sieve is shaken, each object makes contact with the sieve in a random orientation, and then has to pass through the hole in that orientation. Accounting for the physics and the orientation probabilities of the shaking process is another promising avenue for future research.

Acknowledgments

We thank Jernej Barbič, Huanyu Chen, Silvia Sellán, Ryan Mei, Bingjian Huang, and the USC Maker Space team for helping with 3D prints. We thank Silvia Sellán, Yingying (Samara) Ren, and Eitan Grinspun for scientific discussions. We thank Pranav Jain for insight on finding interesting combinations of meshes by suggesting the aliens and machine parts examples. We are grateful to Leticia Mattos Da Silva, Dylan Rowe, Pranav Jain, Letao Chen, and Alice Wei for proofreading. USC's Geometry and Graphics Group is supported by NSF grant 2335493 and a gift by Adobe Inc. We thank the creators of assets used in this article: the COSEG dataset [Wang 2012], the TOSCA dataset [Bronstein et al. 2025], and various Thingiverse

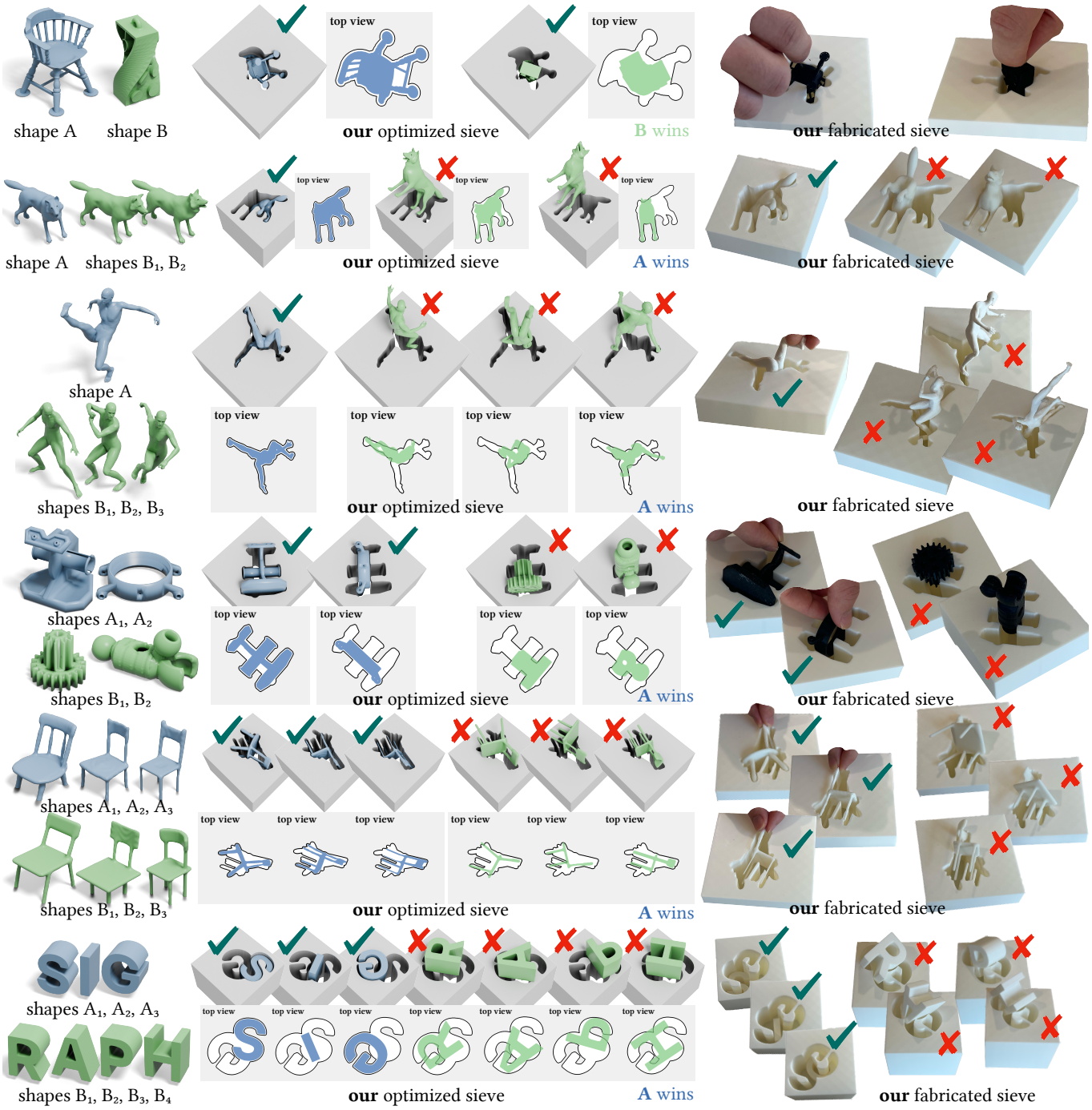


Fig. 15. We fabricate our sieves with a 3D printer to demonstrate the practical use of our method, and verify manually that the sieves admit the A_is they should be admitting / block the B_is they should be blocking. A ✓ means the sieve lets the shape pass, and a × means the shape does not fit.

meshes obtained from Thingi10K [Zhou and Jacobson 2016], [blincoln 2013; bohnded 2011; cerberus333 2012; clintkc 2012; craigmclark 2012; etrohn 2012; gpvillamil 2011; Hotproceed 2012; hugolours 2020; Inorganic 2012; joeyC 2012; KingRahl 2013; kwalus 2013; Landru 2011; leemorton123 2011; Lenbok 2011; MakerBlock 2011; MakerBot 2012; Mecano 2011; Mirice 2014; Padamsky_Miniatures 2023; pmoews 2011; PrettySmallThings 2012; SIMPAD17 2014; spadehand 2012; Sybren 2011; tarturo 2011; TheGoofy 2014].

References

Advantech Manufacturing. 2001. *Test Sieving: Principles and Procedures*. New Berlin, Wisconsin.

Sara Ali, António Galrão Ramos, Maria Antónia Carravilla, and José Fernando Oliveira. 2022. On-line three-dimensional packing problems: A review of off-line and on-line solution approaches. *Computers & Industrial Engineering* 168 (2022).

Marco Attene. 2015. Shapes In a Box: Disassembling 3D Objects for Efficient Packing and Fabrication. *Comput. Graph. Forum* 34, 8 (2015), 13 pages.

P.J. Besl and Neil D. McKay. 1992. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14, 2 (1992), 239–256.

blincoln. 2013. *Twisted Brick*. https://www.thingiverse.com/thing:49080

bohnded. 2011. *Yet Another Entry for the MakerBot Mascot*. https://www.thingiverse.com/thing:11625

Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. 2025. TOSCA dataset. originally in Numerical Geometry of Non-Rigid Shapes (2009), obtained via https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/datasets/tosca.html.

cerberus333. 2012. *Kool aid man*. https://www.thingiverse.com/thing:26249

Wenzheng Chen, Jun Gao, Huan Ling, Edward J. Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. 2019. *Learning to predict 3D objects with an interpolation-based differentiable renderer*.

clintkc. 2012. *Ball in cube (or Rolling Cube)*. https://www.thingiverse.com/thing:19780

Forrester Cole, Kyle Genova, Avneesh Sud, Daniel Vlasic, and Zhoutong Zhang. 2021. Differentiable Surface Rendering via Non-Differentiable Sampling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 6088–6097.

craigmclark. 2012. *Whoofle Magic Dragon*. https://www.thingiverse.com/thing:27969

Qiaodong Cui, Victor Rong, Desai Chen, and Wojciech Matusik. 2023. Dense, Interlocking-Free and Scalable Spectral Packing of Generic 3D Objects. *ACM Trans. Graph.* 42, 4, Article 141 (2023).

Soumyaratna Debnath, Ashish Tiwari, Kaustubh Sadekar, and Shanmuganathan Raman. 2025. RASP: Revisiting 3D Anamorphic Art for Shadow-Guided Packing of Irregular Objects. arXiv:2504.02465 [cs.GR]

Nam Anh Dinh, Itai Lang, Hyunwoo Kim, Oded Stein, and Rana Hanocka. 2025. Geometry in Style: 3D Stylization via Surface Normal Deformation. arXiv:2503.23241

etrohn. 2012. *Dualstrusion Ball in Cube*. https://www.thingiverse.com/thing:22506

Aalok Gangopadhyay, Prajwal Singh, Ashish Tiwari, and Shanmuganathan Raman. 2023. Hand Shadow Art: A Differentiable Rendering Perspective. In *Pacific Graphics Short Papers and Posters*, Raphaëlle Chaine, Zhigang Deng, and Min H. Kim (Eds.). The Eurographics Association.

William Gao, Noam Aigerman, Thibault Groueix, Vova Kim, and Rana Hanocka. 2023. TextDeformer: Geometry Manipulation using Text Guidance. In *ACM SIGGRAPH 2023 Conference Proceedings*. Article 82.

Richard J. Gardner. 2006. *Geometric Tomography: Projections and projection functions*. Cambridge University Press, 97–140.

Jacob E. Goodman, Joseph O'Rourke, , and Csaba D. Tóth. 2017. *Handbook of Discrete and Computational Geometry* (3 ed.). CRC Press.

gpvillamil. 2011. *Little Green Men (flying saucer pilots)*. https://www.thingiverse.com/thing:11810

Baosu Guo, Yu Zhang, Jingwen Hu, Jinrui Li, Fenghe Wu, Qingjin Peng, and Quan Zhang. 2022. Two-dimensional irregular packing problems: A review. *Frontiers in Mechanical Engineering* 8 (2022).

Huihui Guo, Fan Wu, Yunchuan Qin, Ruihui Li, Keqin Li, and Kenli Li. 2023. Recent Trends in Task and Motion Planning for Robotics: A Survey. *ACM Comput. Surv.* 55, 13s, Article 289 (2023).

Hotproceed. 2012. *Sold Holder*. https://www.thingiverse.com/thing:21704

Kai-Wen Hsiao, Jia-Bin Huang, and Hung-Kuo Chu. 2018. Multi-view wire art. *ACM Trans. Graph.* 37, 6, Article 242 (2018).

hugolours. 2020. *Gargoyle/ Demon*. https://www.thingiverse.com/thing:4594515

Inorganic. 2012. *Ferrocene*. https://www.thingiverse.com/thing:19622

Alec Jacobson. 2017. Generalized Matryoshka: Computational Design of Nesting Objects. *Computer Graphics Forum* 36, 5 (2017).

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. 2022. *Mitsuba 3 renderer*. https://mitsuba-renderer.org.

Krishna Murthy Jatavallabhula, Edward Smith, Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. 2019. Kaolin: A PyTorch Library for Accelerating 3D Deep Learning Research. arXiv:1911.05063

joeyC. 2012. *Parametric Magnetic Driver Bit Handle*. https://www.thingiverse.com/thing:34852

Dominik Joho, Jonas Schwinn, and Kirill Safronov. 2024. Neural Implicit Swept Volume Models for Fast Collision Detection. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 15402–15408.

Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. 2020. Differentiable Rendering: A Survey. arXiv:2006.12057

Sagi Katz and Ayellet Tal. 2015. On the Visibility of Point Clouds. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 1350–1358.

Sagi Katz and Ayellet Tal. 2025. HPRO: Direct Visibility of Point Clouds for Optimization. *Computer Graphics Forum* (2025), e70046.

Sagi Katz, Ayellet Tal, and Ronen Basri. 2007. Direct visibility of point sets. *ACM Trans. Graph.* 26, 3 (2007), 24–es.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. arXiv:2308.04079

Hyunwoo Kim, Itai Lang, Noam Aigerman, Thibault Groueix, Vladimir G. Kim, and Rana Hanocka. 2025. MeshUp: Multi-Target Mesh Deformation via Blended Score Distillation. arXiv:2408.14899

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

KingRahl. 2013. *Gnome Chess*. https://www.thingiverse.com/thing:151265

kwalus. 2013. *A Balloon Powered Helicopter*. https://www.thingiverse.com/thing:152804

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular primitives for high-performance differentiable rendering. *ACM Trans. Graph.* 39, 6, Article 194 (2020).

Landru. 2011. *Magneto Helmet: X-men First Class*. https://www.thingiverse.com/thing:12006

leemorton123. 2011. *3D Perspective Illusion Generator(ish)*. https://www.thingiverse.com/thing:13963

Lenbok. 2011. *SOTC Model Cleanup Test*. https://www.thingiverse.com/thing:11710

KeShun Liu. 2009. Some factors affecting sieving performance and efficiency. *Powder Technology* 193, 2 (2009), 208–213.

Yu Liu, Xiaodong Zhou, Zhanping You, Biao Ma, and Fangyuan Gong. 2019. Determining Aggregate Grain Size Using Discrete-Element Models of Sieve Analysis. *International Journal of Geomechanics* 19, 4 (2019), 04019014.

Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural Volumes: Learning Dynamic Renderable Volumes from Images. *ACM Trans. Graph.* 38, 4, Article 65 (2019).

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. 163–169.

MakerBlock. 2011. *OpenSCAD Pirate Ship*. https://www.thingiverse.com/thing:12856

MakerBot. 2012. *Escape From Leviathan*. https://www.thingiverse.com/thing:27065

Mecano. 2011. *Air 2*. https://www.thingiverse.com/thing:14204

Ravish Mehra, Pushkar Tripathi, Alla Sheffer, and Niloy J. Mitra. 2010. Visibility of noisy point cloud data. *Computers & Graphics* 34, 3 (2010), 219–230.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. arXiv:2003.08934

Sehee Min, Jaedong Lee, Jungdam Won, and Jehee Lee. 2017. Soft shadow art. In *Proceedings of the Symposium on Computational Aesthetics*. Article 3.

Mirice. 2014. *Origamix_Rabbit*. https://www.thingiverse.com/thing:600429

Niloy J. Mitra and Mark Pauly. 2009. Shadow art. *ACM Trans. Graph.* 28, 5 (2009).

Remigiusz Modrzewski, Andrzej Obraniak, Adam Rylski, and Krzysztof Siczek. 2022. A Study on the Dynamic Behavior of a Sieve in an Industrial Sifter. *Applied Sciences* 12, 17 (2022).

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. 2021. Large steps in inverse rendering of geometry. *ACM Trans. Graph.* 40, 6, Article 248 (2021).

Quan Nie, Yingfeng Zhao, Li Xu, and Bin Li. 2020. A Survey of Continuous Collision Detection. In *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*. 252–257.

Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani (Eds.). 2007. *Algorithmic Game Theory*. Cambridge University Press.

Padamsky_Miniatures. 2023. *Gnome Wizard*. https://www.thingiverse.com/thing:5763365

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. *PyTorch: an imperative style, high-performance deep learning library*.

pmoews. 2011. *Protein Models*. https://www.thingiverse.com/thing:12283

Riccardo Poli. 2008. Analysis of the Publications on the Applications of Particle Swarm Optimisation. *Journal of Artificial Evolution and Applications* 2008, 1 (2008), 685175.

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

PrettySmallThings. 2012. *Three 1:24 Windsor Chairs*. https://www.thingiverse.com/thing:21999

Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. 2020. Accelerating 3D Deep Learning with PyTorch3D. arXiv:2007.08501

Kaustubh Sadekar, Ashish Tiwari, and Shanmuganathan Raman. 2022. Shadow Art Revisited: A Differentiable Rendering Based Approach. In *2022 IEEE/CVF Winter*

Conference on Applications of Computer Vision (WACV). 628–636.
Nevardo Sanchez-Suarez, Gina Lia Orozco-Mendoza, Jhon Wilder Zartha-Sossa, Delcy Camila Gafaro-Garcés, Lourdes Gladys Melchor-Cahuana, and Cristian Gonzalez-Tovar. 2022. Trends in Sieving and Its Applications in Cereals. A Literature Review. *Frontiers in Sustainable Food Systems* 6 (2022).
Silvia Sellán, Noam Aigerman, and Alec Jacobson. 2021. Swept volumes via spacetime numerical continuation. *ACM Trans. Graph.* 40, 4, Article 55 (2021).
Silvia Sellán, Oded Stein, et al. 2025. gptyoolbox: A Python Geometry Processing Toolbox. https://gpytoolbox.org/.
SIMPAD17. 2014. *Omnis Terra.* https://www.thingiverse.com/thing:492429
spadehand. 2012. *Watch Case for 6498 Manual Wind Movement.* https://www.thingiverse.com/thing:21853
Sybren. 2011. *Enlightened Horns.* https://www.thingiverse.com/thing:11599
tarturo. 2011. *3D Initials Logo - Make your own.* https://www.thingiverse.com/thing:12354
TheGoofy. 2014. *3D printed mechanical Clock with Anchor Escapement.* https://www.thingiverse.com/thing:328569
Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022. Differentiable signed distance function rendering. *ACM Trans. Graph.* 41, 4, Article 125 (2022).
Caoliwen Wang and Bailin Deng. 2024. Neural Shadow Art. arXiv:2411.19161 [cs.CV]
Haochen Wang, Xiaodan Du, Jiahao Li, Raymond A. Yeh, and Greg Shakhnarovich. 2022. Score Jacobian Chaining: Lifting Pretrained 2D Diffusion Models for 3D Generation. arXiv:2212.00774
Jingping Wang, Tingrui Zhang, Qixuan Zhang, Chuxiao Zeng, Jingyi Yu, Chao Xu, Lan Xu, and Fei Gao. 2024. Implicit Swept Volume SDF: Enabling Continuous Collision-Free Trajectory Generation for Arbitrary Shapes. *ACM Trans. Graph.* 43, 4, Article 110 (2024).
Yunhai Wang. 2012. COSEG dataset. https://irc.cs.sdu.edu.cn/~yunhai/public_html/ssl/ssd.htm.
Tianyang Xue, Mingdong Wu, Lin Lu, Haoxuan Wang, Hao Dong, and Baoquan Chen. 2023. Learning Gradient Fields for Scalable and Generalizable Irregular Packing. In *SIGGRAPH Asia 2023 Conference Papers*. Article 105, 11 pages.
Zeshi Yang, Zherong Pan, Manyi Li, Kui Wu, and Xifeng Gao. 2023. Learning Based 2D Irregular Shape Packing. *ACM Trans. Graph.* 42, 6, Article 266 (2023).
Xinya Zhang, Robert Belfer, Paul G. Kry, and Etienne Vouga. 2020. C-Space tunnel discovery for puzzle path planning. *ACM Trans. Graph.* 39, 4, Article 104 (2020).
Hang Zhao, Zherong Pan, Yang Yu, and Kai Xu. 2023. Learning Physically Realizable Skills for Online Packing of General 3D Shapes. *ACM Trans. Graph.* 42, 5, Article 165 (2023).
Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).
Jingsen Zhu, Fujun Luan, Yuchi Huo, Zihao Lin, Zhihua Zhong, Dianbing Xi, Rui Wang, Hujun Bao, Jiaxiang Zheng, and Rui Tang. 2022. Learning-based Inverse Rendering of Complex Indoor Scenes with Differentiable Monte Carlo Raytracing. In *SIGGRAPH Asia 2022 Conference Papers*. Article 6, 8 pages.

Supplemental Material

A Proof for similarity proposition

Here we supply the proof for Prop. 3.5 from the article.

Proposition A.1. *$A \sim B$ iff the two shapes have the same set of projections across all orientations.*

Proof. We first prove a related *Claim*: $A \preceq B \iff \forall R_B \in SO(3)\, \exists R_A \in SO(3), \mathbf{t}_A \in \mathbb{R}^2$ such that $\mathrm{proj}(R_A A) + \mathbf{t}_A \subseteq \mathrm{proj}(R_B B)$

($\Rightarrow$): For any $R_B \in SO(3)$, trivially $B \in \mathrm{proj}(R_B B)$. Since $A \preceq B$, we must also have $A \in \mathrm{proj}(R_B B)$, which means there must exist $R_A \in SO(3), \mathbf{t}_A \in \mathbb{R}^2$ such that $\mathrm{proj}(R_A A) + \mathbf{t}_A \subseteq \mathrm{proj}(R_B B)$.

($\Leftarrow$): Suppose H is a sieve hole such that $B \in H$. Then there exists $R_B \in SO(3)$ and $\mathbf{t}_B \in \mathbb{R}^2$ such that $\mathrm{proj}(R_B B) + \mathbf{t}_B \subseteq H$. By the assumption, $\exists R_A \in SO(3), \mathbf{t}_A \in \mathbb{R}^2$ such that

$$\mathrm{proj}(R_A A) + \mathbf{t}_A + \mathbf{t}_B \subseteq \mathrm{proj}(R_B B) + \mathbf{t}_B \subseteq H$$

So $A \in H$, and thus $A \preceq B$.

Now we prove the two directions of the proposition.

($\Rightarrow$): Since $A \preceq B$,

$$\{\mathrm{proj}(RA) + \mathbf{t} \mid R \in SO(3), \mathbf{t} \in \mathbb{R}^2\} \subseteq \{\mathrm{proj}(RB) + \mathbf{t} \mid R \in SO(3), \mathbf{t} \in \mathbb{R}^2\},$$

and since $B \preceq A$, the converse also holds. Thus the two sets of projections are equal.

($\Leftarrow$): The equality of the two sets implies

$$\{\mathrm{proj}(RA) + \mathbf{t} \mid R \in SO(3), \mathbf{t} \in \mathbb{R}^2\} \subseteq \{\mathrm{proj}(RB) + \mathbf{t} \mid R \in SO(3), \mathbf{t} \in \mathbb{R}^2\},$$

as well as the converse, and thus $A \preceq B$ and $B \preceq A$ respectively, so $A \sim B$. □

B Proof for blocking complex motions

Proposition B.1. *Define the sieve pipe P to be the prism of height height h with sieve hole H as the base, which represents the shape of the 3D tunnel of our sieve. If h is at least the diameter of the circumsphere of a blocked shape B, then there cannot exist a trajectory for this shape to pass through the sieve.*

Proof. For the sake of contradiction, suppose otherwise. Then due the thickness of h, at some point along the trajectory, B must be fully contained in P at some orientation $(R, \mathbf{t})$ where $R \in SO(3)$ and $\mathbf{t} \in \mathbb{R}^3$. This implies $\mathrm{proj}(RB + \mathbf{t}) \subseteq H$, a contradiction. □

C Mesh & runtime statistics

Tab. 1 features runtime statistics for each experiment performed. Tab. 2 contains vertex and face counts for the meshes used. The mesh names reference either TOSCA [Bronstein et al. 2025], COSEG [Wang 2012], Thingi10k [Zhou and Jacobson 2016], or very simple meshes created by the authors.

Fig 17 features a convergence plot of the inner Adam optimization. Fig. 16 shows how the runtime of our method scales with the number of input shapes.

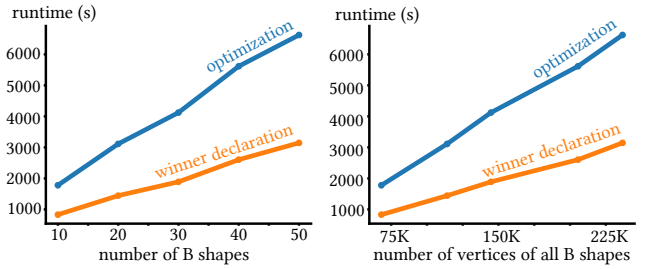


Fig. 16. Runtimes of the optimization and winner declaration stages of our method with one shape in $\mathcal{A}$ and a varying number of shapes in $\mathcal{B}$, where all shapes were chosen randomly from Thingi10K. For these experiments, we used a raster resolution of 128×128 during optimization as the default value led to exceeding the GPU memory.

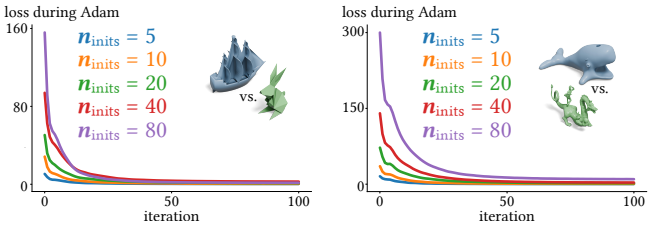


Fig. 17. Evolution of the loss in the first case of (15) during Adam optimization for varying numbers of different initializations of the state of $\mathcal{B}$.

D Implementation details

For any figures of results that were produced using non-default parameters, we report the values below:

- Fig. 5, first row: 20 random $\mathcal{B}$ initial orientations
- Fig. 5, fourth row: 40 random $\mathcal{B}$ initial orientations
- Fig. 7, second row: 30 random $\mathcal{B}$ initial orientations
- Fig. 15, second row: 20 random $\mathcal{B}$ initial orientations

Meshes of cats and humans from the TOSCA dataset used here were modified to be watertight and have been renamed by appending `_fixed` to each STL filename. While our algorithm can robustly handle non-watertight meshes, they are unsuitable for our 3D printers.

E Figure details

To render images featuring a 3D object inside a sieve, we preprocess the sieve mesh by slightly smoothing the hole geometry to remove jagged edges from the surface reconstruction. This does not matter for the 3D printing process, since the tolerances are larger than the jaggedness.

Figure	$\mathcal{A}$ meshes	$\mathcal{B}$ meshes	Optimization runtime (s)	Winner declaration runtime (s)	Final $\mathcal{B}$ losses	$\mathcal{A}$ wins
Fig. 8 (top)	37111.stl	37009.stl, 37415.stl	902.0	415.9	0.168, 0.077	Y
Fig. 14	37731.stl	37730.stl	317.1	146.5	0.001	N
Fig. 5 (third)	38631.stl	39880.stl	273.2	122.0	0.037	Y
Fig. 5 (fourth)	39930.stl	39925.stl	222.9	19.4	0.044	Y
Fig. 6 (first)	41729.stl	964933.stl	54.1	16.2	0.000	N
Fig. 6 (second)	45550.stl	79239.stl	1691.3	754.6	0.000	N
Fig. 13 (bottom)	46665.stl	100139.stl	78.7	31.4	0.020	Y
Fig. 13 (top)	46665.stl	100139.stl	11.2	60.2	0.000	N
Fig. 9 (top)	46665.stl	46665_long.stl	54.9	19.6	0.270	Y
Fig. 9 (bottom)	46665.stl	46665_long.stl, 100139.stl	121.4	54.0	0.108, 0.000	N
Fig. 11 (bottom)	62526_cut.stl	62526.stl	352.1	149.3	0.030	Y
Fig. 11 (top)	62526_cut.stl	62526.stl	115.0	151.6	0.000	N
Fig. 4 (left)	62860.stl	cube.stl	30.4	10.0	0.345	Y
Fig. 4 (right)	62860.stl	cube.stl	27.6	9.8	0.000	N
Fig. 2	67223.stl	591211.stl	63.6	27.7	0.017	Y
Fig. 15 (first)	67856.stl	131500.stl	845.7	367.1	0.000	N
Fig. 10 (bottom)	69079_cut.stl	69079.stl	176.7	45.9	0.031	Y
Fig. 10 (top)	69079_cut.stl	69079.stl	53.7	48.5	0.000	N
Fig. 6 (third)	80597.stl	83229.stl	1805.5	788.9	0.000	N
Fig. 6 (fourth)	293457.stl	293453.stl	1118.6	494.9	0.000	N
Fig. 7 (top)	296802.stl	296803.stl	148.7	44.4	0.013	Y
Fig. 7 (bottom)	296803.stl	296802.stl	233.2	36.8	0.006	Y
Fig. 12 (bottom)	arch.stl	rect_prism.stl	23.4	10.3	0.021	Y
Fig. 12 (top)	arch.stl	rect_prism.stl	9.4	9.9	0.000	N
Fig. 3 (right)	bowtie.stl	sphere.stl	29.8	11.5	0.193	Y
Fig. 3 (left)	bowtie.stl	sphere.stl	24.4	11.5	0.212	Y
Fig. 1	cat9_fixed.stl	cat1_fixed.stl, cat3_fixed.stl, cat4_fixed.stl	3956.2	1778.5	0.012, 0.030, 0.053	Y
Fig. 5 (second)	cat9_fixed.stl	cat6_fixed.stl	1315.8	613.8	0.007	Y
Fig. 5 (first)	centaur5.stl	centaur4.stl	1551.3	332.0	0.084	Y
Fig. 15 (fifth)	coseg_chairs_102.stl, coseg_chairs_103.stl, coseg_chairs_111.stl	coseg_chairs_112.stl, coseg_chairs_116.stl, coseg_chairs_120.stl	9410.5	878.9	0.081, 0.049, 0.046	Y
Fig. 8 (bottom)	coseg_vases_364.stl	coseg_vases_361.stl, coseg_vases_365.stl, coseg_vases_808.stl	1978.2	891.9	0.100, 0.361, 0.280	Y
Fig. 15 (third)	david6_fixed.stl	david10_fixed.stl, david11_fixed.stl, david13_fixed.stl	7547.2	3267.4	0.141, 0.153, 0.107	Y
Fig. 15 (fourth)	67523.stl, 67223.stl	822070.stl, 591211.stl	2037.7	271.5	0.032, 0.037	Y
Fig. 15 (sixth)	thick_S.stl, thick_I.stl, thick_G.stl	thick_R.stl, thick_A.stl, thick_P.stl, thick_H.stl	1981.6	118.1	0.080, 0.063, 0.050, 0.074	Y
Fig. 5 (fifth)	wizard.stl	gargoyle.stl	9334.0	4126.7	0.203	Y
Fig. 15 (second)	wolf2.stl	wolf0.stl, wolf1.stl	878.3	204.8	0.036, 0.036	Y

Table 1. Runtime statistics for experiments in this article. "Final $\mathcal{B}$ losses" is the energy (9) at the end of $\mathcal{B}$'s gradient-based optimization step from the winner declaration procedure.

Mesh filename	Number of vertices	Number of faces
100139.stl	1112	2224
131500.stl	17426	34882
293453.stl	23739	47474
293457.stl	23773	47542
296802.stl	1156	2320
296803.stl	1120	2248
37009.stl	12002	23909
37111.stl	44004	88106
37415.stl	7853	13340
37730.stl	5212	10512
37731.stl	5480	11048
38631.stl	8059	16728
39880.stl	4953	9902
39925.stl	418	844
39930.stl	514	1024
41729.stl	2655	5330
45550.stl	1595	3222
46665.stl	376	764
46665_long.stl	376	764
591211.stl	680	1380
62526.stl	7818	13152
62526_cut.stl	7480	12560
62860.stl	40	96
67223.stl	1030	2064
67523.stl	1592	3204
67856.stl	5647	11350
69079.stl	1470	2956
69079_cut.stl	1478	2968
79239.stl	36164	72328
80597.stl	21999	43994
822070.stl	11213	22450
83229.stl	37988	76026
964933.stl	148	292
arch.stl	28	52
bowtie.stl	16	28
cat1_fixed.stl	27896	55788
cat3_fixed.stl	27896	55788
cat4_fixed.stl	27896	55788
cat6_fixed.stl	27896	55788
cat9_fixed.stl	27896	55788
centaur4.stl	15768	31528
centaur5.stl	15768	31532
coseg_chairs_102.stl	15724	31456
coseg_chairs_103.stl	9652	19304
coseg_chairs_111.stl	8050	16100
coseg_chairs_112.stl	13628	27256
coseg_chairs_116.stl	13463	26926
coseg_chairs_120.stl	10121	20242
coseg_vases_361.stl	14859	29734
coseg_vases_364.stl	13548	27100
coseg_vases_365.stl	13514	27028
coseg_vases_808.stl	13172	26344
cube.stl	8	12
david10_fixed.stl	52568	105132
david11_fixed.stl	52568	105132
david13_fixed.stl	52569	105134
david6_fixed.stl	52569	105134
gargoyle.stl	172687	347360
rect_prism.stl	8	12
sphere.stl	50	96
thick_A.stl	680	1360
thick_G.stl	774	1544
thick_H.stl	622	1240
thick_I.stl	282	560
thick_P.stl	564	1128
thick_R.stl	692	1384
thick_S.stl	738	1472
wizard.stl	308544	616940
wolf0.stl	4344	8684
wolf1.stl	4344	8684
wolf2.stl	4344	8684

Table 2. Details for the meshes used in experiments in this article.