Putting Rigid Bodies to Rest

HOSSEIN BAKTASH, Carnegie Mellon University, USA
NICHOLAS SHARP, NVIDIA Research, USA
QINGNAN ZHOU, Adobe Research, USA
KEENAN CRANE, Carnegie Mellon University, USA
ALEC JACOBSON, University of Toronto & Adobe Research, Canada

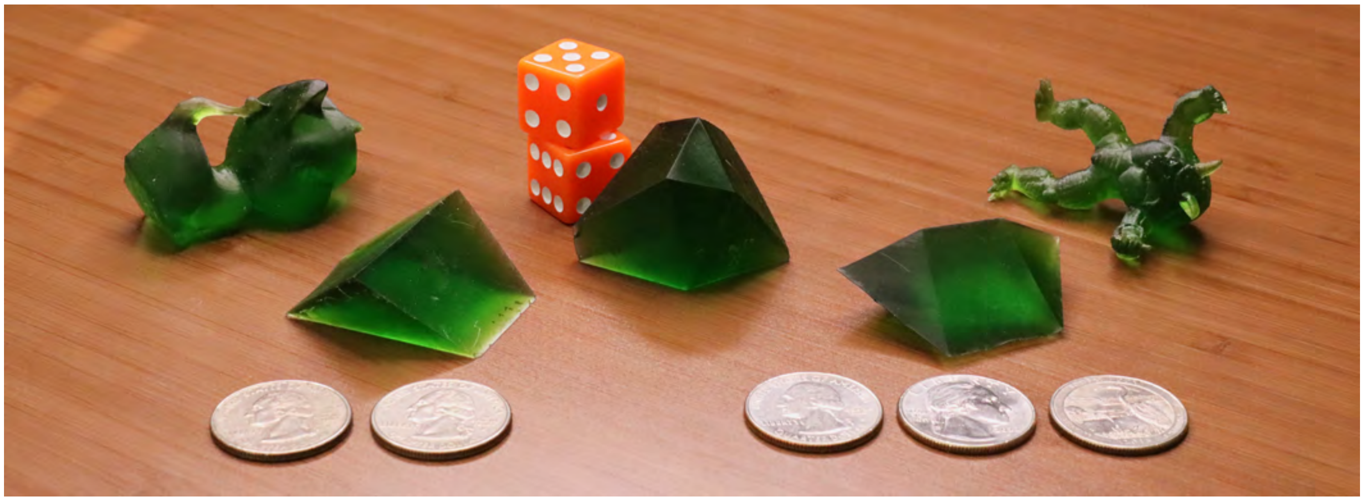


Fig. 1. We introduce a geometric model for the resting equilibria of rigid bodies. This model supports fast differential evaluation, enabling tasks such as the inverse design of dice with prescribed probabilities. As shown in the photograph above, we design single dice that capture the statistics of other objects such as the sum of a pair of ordinary dice (middle), or the number of heads from flipping two or three fair coins (front). We can also deform nonconvex shapes such that their stable configurations exhibit dice-like probabilities; here, the kitten and armadillo models both have just three stable configurations with equal probabilities.

This paper explores the analysis and design of the resting configurations of a rigid body, without the use of physical simulation. In particular, given a rigid body in $\mathbb{R}^3$, we identify all possible stationary points, as well as the probability that the body will stop at these points, assuming a random initial orientation and negligible momentum. The forward version of our method can hence be used to automatically orient models, to provide feedback about object stability during the design process, and to furnish plausible distributions of shape orientation for natural scene modeling. Moreover, a differentiable inverse version of our method lets us design shapes with target resting behavior, such as dice with target, nonuniform probabilities. Here we find solutions that would be nearly impossible to find using classical techniques, such as dice with additional unstable faces that provide more natural overall geometry.

From a technical point of view, our key observation is that rolling equilibria can be extracted from the *Morse-Smale complex* of the *support function* over the Gauss map. Our method is hence purely geometric, and does not make use of random sampling, or numerical time integration. Yet surprisingly, this purely geometric model makes extremely accurate predictions of rest behavior, which we validate both numerically, and via physical experiments. Moreover, for computing rest statistics, it is orders of magnitude faster than state of the art rigid body simulation, opening the door to inverse design—rather than just forward analysis.

Authors' Contact Information: Hossein Baktash, Carnegie Mellon University, Pittsburgh, USA, hbaktash@andrew.cmu.edu; Nicholas Sharp, NVIDIA Research, Seattle, USA, nmwsharp@gmail.com; Qingnan Zhou, Adobe Research, New York, USA, qzhou@adobe.com; Keenan Crane, Carnegie Mellon University, Pittsburgh, USA, keenanc@andrew.cmu.edu; Alec Jacobson, University of Toronto & Adobe Research, Toronto, Canada, alecjacobson@adobe.com.

ACM 1557-7368/2025/8-ART
https://doi.org/10.1145/3731203

CCS Concepts: • **Computing methodologies → Shape analysis**; Physical simulation.

Additional Key Words and Phrases: Rigid Body, Static Equilibrium, Dice

ACM Reference Format:
Hossein Baktash, Nicholas Sharp, Qingnan Zhou, Keenan Crane, and Alec Jacobson. 2025. Putting Rigid Bodies to Rest. *ACM Trans. Graph.* 44, 4 (August 2025), 16 pages. https://doi.org/10.1145/3731203

1 Introduction

"Let it roll, baby, roll." —Jim Morrison, *Roadhouse Blues*

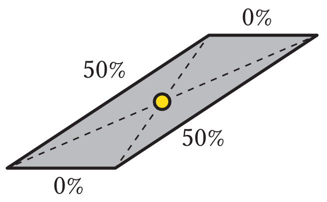
Where will a rolling object come to rest? The equilibrium distribution of orientation for rigid bodies is central to diverse physical, functional, and aesthetic problems, yet in all its simplicity, computing this distribution with general-purpose simulation tools is not straightforward. Even enumerating a complete set of equilibrium states is challenging for conventional, simulation-based methods, since it is prone to sampling error: equilibria which are improbable or not well-separated may be overlooked when running random trials. In general, analyzing the *static* behavior of a rigid body by running a full *dynamical* simulation is time-intensive, prone to numerical errors, and requires delicate parameter tuning (Section 4.1). Obtaining a reliable distribution of equilibria is even harder, requiring numerous trials just to obtain approximate statistics.

We adopt the philosophy that, from a statistical perspective, the rest behavior of a rolling object is largely a function of its geometry and little else—not so different from properties like center of mass or moments of inertia.

In particular we assume that momentum is negligible, and that dynamics hence follow the gradient of a gravitational potential. This assumption enables us to analyze rolling via techniques from computational geometry, rather than those from physical simulation. Unlike a dynamic simulation, our method does not predict the final resting pose of an object from a given initial height, orientation, and momentum. Instead, it estimates the probability distribution over stable resting configurations, based purely on geometry.

Of course, just as the rigidity assumption ignores certain physical effects (such as the influence of elastic deformation), our geometric analysis ignores the influence of dynamical effects on the distribution of equilibrium states. However, in situations where the distribution of initial configurations and/or velocities is unknown (or hard to measure), this purely geometric analysis is the only one possible: one simply does not have the data needed to obtain more accurate statistics via dynamical simulation. Moreover, we find in practice that our geometric analysis is often highly predictive of statistics obtained via full-blown rigid body simulation—at a fraction of the computational cost (Section 4.1). This is especially useful for analyzing complex and dense shapes, where a robust simulator can take a very long time, and a fast simulator can produce unreliable or physically implausible results (Section 4.1).

From a purely geometric point of view, one way to reason about the probabilities of stopping at a certain face is to simply consider the (normalized) solid angles of each face of the convex hull. Although intuitive, the solid angle-based approach is only accurate for a subset of shapes whose convex hull faces are all stable (Section 2.4.1). A 2D counterexample is shown on the right: a parallelogram that has zero probability of resting on its top or bottom sides without tipping over, despite those sides having non-zero associated solid angles.



Overall, the speed of our method makes it an effective design tool, where a user can get real time feedback on the equilibrium distribution of an object while modeling or editing a shape. For instance, we can show the most likely orientation, or quantify the probability that an object is stable in a configuration of interest. Since analysis is nearly instantaneous even when computed from scratch, users are free to perform arbitrary modeling operations. Moreover, since we make no assumption on the input geometry (*e.g.*, it need not be closed, manifold, *etc.*), the method can easily be integrated with many existing modeling tools.

We can easily differentiate the computed probabilities with respect to the input geometry, with any assumption about the mass density, and design objects with arbitrary resting probabilities (Section 5). We fabricated such objects with various 3D printers and experimented with them (see Section 6).

1.1 Related work

The rolling behavior of irregularly-shaped objects has been studied from antiquity to modern times. For instance, irregular sheep knuckle bones (with unequal resting probabilities) have long been used as dice in cultures across the globe [Mazzorin and Minniti 2013]. More recently, mathematical analysis of rolling stability reveals remarkable depth—even for simple questions. For instance, as recently as 2006 an open question was whether there is a 3D shape with only two equilibria (one stable, one unstable), finally leading to construction of an object called the *Gömböc* which answers this question in the affirmative [Várkonyi and Domokos 2006b,a].

1.1.1 Computational Fabrication. Our work follows the spirit of research from computational fabrication and inverse design that analyze and optimize physical properties of rigid solids. For instance, *Make It Stand* [Prévost et al. 2013] optimizes the internal weight distribution to ensure stability of a fabricated shape at a given orientation. Subsequent work has likewise studied buoyancy [Wang and Whiting 2016; Prévost et al. 2016], and spinning [Bächer et al. 2014; Hafner et al. 2024]. This stream of work leads to the creation of unified weight distribution frameworks [Wu et al. 2016; Musialski et al. 2015, 2016] to optimize a shape to stand, spin and float along a chosen orientation. Additional properties such as structural weakness [Stava et al. 2012; Zhou et al. 2013; Sellán et al. 2022] and strength-to-weight ratio [Lu et al. 2014] have also been studied in depth. We refer interested readers to [Livesu et al. 2017] for additional studies in computational fabrication. Closer to our work, [Fu et al. 2008; Liu et al. 2016] predict a plausible upright orientation for man-made objects, using a data-driven approach. In another work, [Hurst and Tandiman 2024], solid angle of each face is computed and used to fabricate unfair dice, but the work is only limited to rectangular prisms. Earlier work such as [Ngoi and Lim 1996] and [Boothroyd and Ho 1977], inspired by studying how parts rest on a manufacturing pipeline, relates resting probabilities (also under momentum-less assumption) to the solid angle of faces of a shape. We see in Section 2.4.1 that these solid angles correspond to resting probabilities of faces only for a very limited subset of shapes. Our work identifies *all* stable configurations of arbitrary shapes, and enables fully and semi automatic design/modification of shapes with all of their resting probabilities in mind. This leads to designing asymmetric shapes that behave like fair die, and shapes with non-uniform resting probabilities (Section 5).

1.1.2 Rigid Body Simulation. Rigid body simulation has been a useful tool for a variety of tasks ranging from bin packing [Xu et al. 2023] to generating contact force distribution for dropped objects

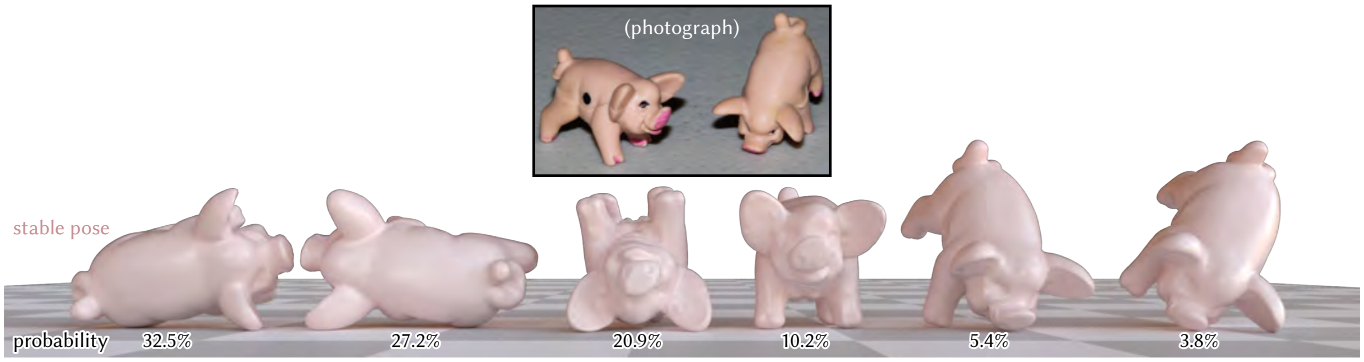


Fig. 2. Our algorithm efficiently and robustly computes the probability of all resting configurations (in 3 *ms*) of the pig model from the popular game "Pass the Piggies" [Moffatt 1977]. In particular, our predictions match the experimental data from [Kern 2006] up to an optimal transport distance of 0.09.

[Langlois et al. 2016]. Accurately predicting the probability of rest poses is challenging for simulators, as it typically requires running a large number of simulations with randomized initial drop configurations. Beyond the inefficiency of this approach, rigid body simulators can be difficult to control [Popović et al. 2000], and are highly sensitive to both physical parameters (such as coefficients of friction and restitution) and simulation parameters (time step, convergence tolerance, etc.). While modern physics-simulation engines such as Bullet [Coumans and Bai 2016] and Jolt [Rouwe 2024] are quite *efficient*, achieving statistically significant, physically plausible results can require numerous trials and data-specific parameter tuning, and may not be possible for all problems. More recent rigid body simulators, e.g. Rigid IPC [Ferguson et al. 2021], are far more reliable and always produce physically plausible outcomes; though this comes with a computational cost and ignoring bounces (coefficient of restitution) which is essential in many real-world scenes. We use both Rigid IPC and Bullet to validate our results in Section 4.1. Remarkably, we find a strong correspondence between the probabilities predicted by our model and those obtained from momentum-preserving simulations. These comparisons also offer insights that guide our shape optimization process in Section 3.2, enabling us to design objects that perform as intended in the presence of momentum, especially after fabrication.

1.1.3 Rolling Analysis. Rolling trajectories were analyzed by Sobolev et al. [2023], who designed objects that roll naturally along a given periodic path along the ground. They assume that the center of mass stays at a fixed height—hence, gravity has no effect on the trajectory, and objects simply roll like a wheel, given an initial rotation velocity. In contrast, we do not consider the translation of the objects caused by rolling, and specifically focus on the rotational trajectory induced by a gravitational potential.

1.1.4 Morse-Smale Complex. To study the rolling and resting behavior of rigid bodies, we construct a *potential* function defined along the Gauss map (Section 2), and analyze its *Morse-Smale complex (MS complex)*. In the past, the MS complex has been applied to a variety of problems in graphics and computational geometry, for tasks like shape simplification and quadrilateral remeshing [Gyulassy and Natarajan 2005; Dong et al. 2006; Bauer et al. 2012; Weinkauf et al. 2010]. To our knowledge, however, the Morse-Smale complex has never been used to numerically analyze the rolling of rigid bodies.

2 Algorithm

Input. The input to our algorithm is a 3D shape $\Omega \subset \mathbb{R}^3$ (encoded by a mesh, a point cloud, *etc.*) with center of mass $\mathbf{c} \in \mathbb{R}^3$. We assume Ω has a polyhedral convex hull $\mathcal{H} \supseteq \Omega$, and make a general position assumption that each face f on the boundary of $\mathcal{H}$ is a triangle. The shape Ω need not have a homogeneous mass density, and hence $\mathbf{c}$ may not coincide with the geometric barycenter. However, we do assume nonnegative density so that $\mathbf{c}$ lies within the convex hull $\mathcal{H}$.

Output. The output of our computation is a set of probabilities $p_f \in [0,1]$ assigned to each face of $\mathcal{H}$, with $\sum_{f\in\mathcal{H}} p_f = 1$. The quantity p_f gives the probability that Ω will roll to rest on face f, after starting from an initial orientation chosen uniformly at random.

An example shape with all its stable orientations and their corresponding probabilities is shown in Figure 2.

2.1 Rolling as Energy Minimization

Our analysis assumes the rigid object Ω translates and rotates in response to a gravitational potential, but does not exhibit any momentum, coming to rest as soon as the potential is locally minimized. These modeling assumptions are essentially equivalent to a high-friction scenario, where kinetic energy is rapidly dissipated due to a rough surface (*e.g.*, rolling across a thick carpet) or a high-viscosity assumption, where kinetic energy is dissipated due to drag (*e.g.*, submerged in a viscous liquid).

More explicitly, the object follows gradient trajectories of the following optimization problem:

$$\underset{\mathbf{R}\in SO(3),\mathbf{t}\in\mathbb{R}^3}{\text{minimize}} \int_\Omega \rho\left(\mathbf{R}\mathbf{x}+\mathbf{t}\right)\cdot\hat{\mathbf{z}}\,dV \tag{1}$$

$$\text{subject to: } \mathbf{x}_z \geq 0\,\forall\mathbf{x}\in\Omega. \tag{2}$$

Here $\mathbf{R}$ and $\mathbf{t}$ represent the rotation and translation of the object, *resp.*, $\rho : \Omega \to \mathbb{R}_{\geq 0}$ is the mass density, the vector $\hat{\mathbf{z}} = (0,0,1)$ is the opposite direction of gravity, and the constraint in Equation 2 prohibits the object from falling through the ground floor ($z = 0$).

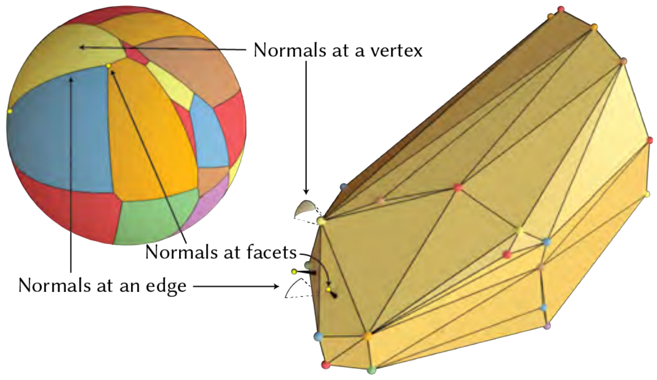


Fig. 3. For a convex polyhedron (right), the Gauss map partitions the sphere (left) into regions corresponding to normals spanned at vertices, separated by arcs corresponding to normals spanned at edges, connected to isolated points corresponding to the normals of faces.

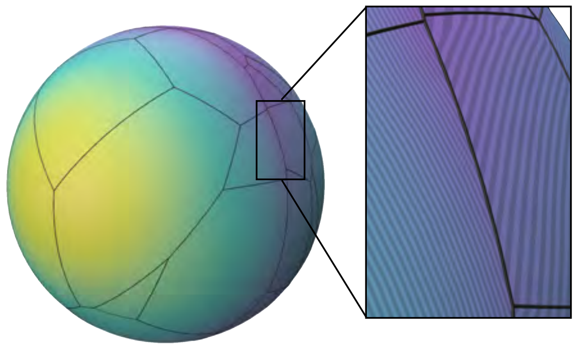


Fig. 4. The height of the center of mass is a continuous function (U) with respect to the downward normal direction. Visualized in pseudocolor over the Gauss map we can see that its gradient $\partial U/\partial \hat{\mathbf{n}}$ can be discontinuous across dual edges.

Denoting $m = \int_\Omega \rho \, dV$ (total mass) and $\mathbf{c} = \frac{1}{m}\int_\Omega \rho \, \mathbf{x} \, dV$ (center of mass), by linearity the integral in Equation 1 can be simplified to $(m\mathbf{R}\mathbf{c} + m\mathbf{t}) \cdot \hat{\mathbf{z}} = m(\mathbf{R}\mathbf{c} + \mathbf{t}) \cdot \hat{\mathbf{z}}$. Without loss of generality we can assume $m = 1$, then we are simply left with the z-coordinate or "height" of the transformed center of mass $(\mathbf{R}\mathbf{c} + \mathbf{t}) \cdot \hat{\mathbf{z}}$. Moreover, for any fixed rotation $\mathbf{R}$, the corresponding optimal translation $\mathbf{t}$ is trivial: move the object Ω downward starting at $z = +\infty$ until some vertex hits the ground. Hence, we need only consider $\mathbf{R}$ directly.

For any rotation $\mathbf{R} \in SO(3)$ of the object, the direction $\hat{\mathbf{n}} = \mathbf{R}^{-1}\hat{\mathbf{z}}$ is a unit vector representing the opposite gravity direction in the object's local coordinate frame. It can be associated with a face, edge or vertex of the convex hull, $\mathcal{H}$, based on its location in $\mathcal{H}$'s Gauss Map. Since $\mathcal{H}$ is a convex triangulated polyhedron, its Gauss Map is a dual mesh, where original faces, edges and vertices map to points, great arcs, and spherical polygons on the Gauss sphere. Together, they form a partition of the unit sphere (Figure 3). We refer to the (in general) set of normals corresponding to any hull simplex i as $\mathcal{N}_i \subset S^2$

Rotating the object around the z-axis does not change its gravitational potential: the height of the center of mass $\mathbf{c}$ does not change. When a hull vertex with position $\mathbf{x}_j \in \mathbb{R}^3$ is resting on the ground, and the upward direction, $\hat{\mathbf{z}}$, is obtained by rotating a normal direction $\hat{\mathbf{n}}$ by a rotation $\mathbf{R}$, then the height of the center of mass $\mathbf{c}$ is a simple smooth function (ignoring new collisions with the ground):

$$U_j(\hat{\mathbf{n}}) = \left(\mathbf{c} - \mathbf{x}_j\right) \cdot \hat{\mathbf{n}} \tag{3}$$

If the shape is lowered towards the ground in the $-\hat{\mathbf{n}}$ direction, there will be a unique vertex, edge, or face that hits the ground first. We can write the height of the center of mass as a piecewise-smooth function:

$$U(\hat{\mathbf{n}}) = \sum_j \delta_j(\hat{\mathbf{n}}) U_j(\hat{\mathbf{n}}), \tag{4}$$

where $\delta_j(\hat{\mathbf{n}})$ equals 1 if $\hat{\mathbf{n}}$ lies in the interior of vertex j's spherical polygon on the Gauss map, $\frac{1}{2}$ if it lies on j's boundary arc, $\frac{1}{3}$ if it lies on j's boundary vertex, and 0 otherwise. This allows us to rewrite the minimization in Equation 1 directly over $\hat{\mathbf{n}}$:

$$\underset{\hat{\mathbf{n}} \in S^2}{\text{minimize}} \quad U(\hat{\mathbf{n}}) \tag{5}$$

.

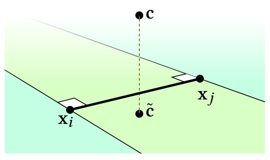
2.2 Tracing a single path

Given some initial upward orientation $\hat{\mathbf{n}}_0$, a momentumless simulation of rolling amounts to rotating in the direction that most decreases U until it is locally minimized, resulting in a stable orientation. Note that we always assume that the inner product on S^2 is induced by the Euclidean metric.

2.2.1 Tracing in a vertex's Gauss image. The gradient $\partial U/\partial \hat{\mathbf{n}}$ is well defined when $\hat{\mathbf{n}}$ falls in the Gauss image of some hull vertex i. We can avoid numerical gradient integration and use the fact that pathlines of the $\partial U/\partial \hat{\mathbf{n}}$ vector field inside a vertex i's dual image are great arcs on S^2 emanating from a unique point $\hat{\mathbf{n}}_i^\star$ (see Sec. 2.3.2). To precisely integrate along $\partial U/\partial \hat{\mathbf{n}}$, we can find the point of intersection between the great arc "ray" departing from the current normal $\hat{\mathbf{n}}$ in the direction of $-\partial U/\partial \hat{\mathbf{n}}$ and the dual edges (also great arcs) bounding the dual image of vertex i.

2.2.2 Tracing in an edge's Gauss image. If $\hat{\mathbf{n}}$ falls along the Gauss image of some hull edge ij, then its integration requires a bit of care. While U is a continuous function over all S^2, its full gradient is not well defined along Gauss images of hull edges. These points correspond to "cusps" akin to the sharp crease formed between unioned spheres. An example of this type of discontinuity is shown in figure 4.

Nevertheless, for any normal $\hat{\mathbf{n}}$ along a hull edge's dual image, the direction of steepest descent is always well defined. Edges can be categorized into two cases, readily understood in the primary domain $\mathcal{H}$ (see inset). When $\hat{\mathbf{n}}$ lies on the dual image of edge ij, it means that the edge is lying on the ground. In general position, this leads to two possible cases. In case (E1) the object Ω rotates like



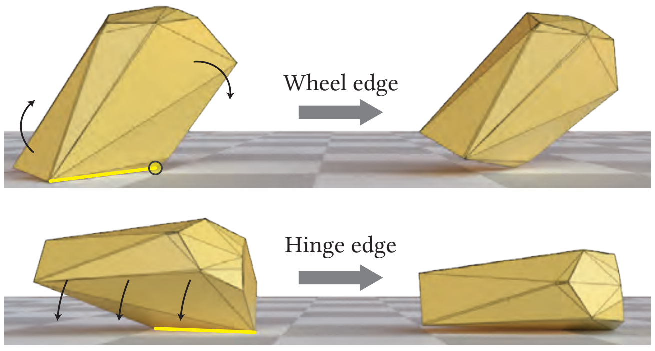


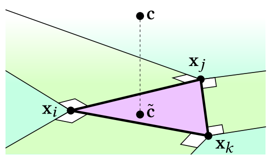
Fig. 5. Convex hull edges can be classified by their relationship with the center of mass: the object either rotates end-of-end like a cartwheel (top) or around the edge as an axis (bottom).

a cartwheel onto one of its incident vertices, based on the steepest descent at $\hat{\mathbf{n}}$ agreeing with the gradient of the contribution for either incident vertex $\partial U_i/\partial\hat{\mathbf{n}}$ or $\partial U_j/\partial\hat{\mathbf{n}}$. Alternately, in case (E2) the object Ω rotates around the edge (like a hinge) onto one of its incident faces, based on the steepest descent at $\hat{\mathbf{n}}$ agreeing with $\partial U/\partial\hat{\mathbf{n}}$ *restricted* to the Gauss image of the edge ij. Each edge ij falls into one case regardless of the particular evaluation normal $\hat{\mathbf{n}}$ in its Gauss image. We can identify edges precisely in the primary domain with the following steps:

(1) Take an arbitrary plane L passing through $\mathbf{x}_i, \mathbf{x}_j$,
(2) Divide L into three sections by drawing two parallel lines from $\mathbf{x}_i$ and $\mathbf{x}_j$, both orthogonal to the segment $[\mathbf{x}_i, \mathbf{x}_j]$,
(3) Let $\tilde{\mathbf{c}}$ be the projection of the center of mass $\mathbf{c}$ onto L,
(4) If $\tilde{\mathbf{c}}$ is not in the middle section, then e is a cartwheel edge (E1) rolling onto the closest vertex to $\tilde{\mathbf{c}}$,
(5) Otherwise e is a hinge edge (E2).

For case E1, the next step of rolling continues with tracing within the identified vertex's Gauss image (see Sec. 2.2.1). For case E2, the next step of rolling continues to an incident face, identified by checking the direction of $\partial U/\partial\hat{\mathbf{n}}$ restricted along the edge's Gauss image evaluated at the current normal $\hat{\mathbf{n}}$. These two types of behaviour are shown in the primary domain in figure 5.

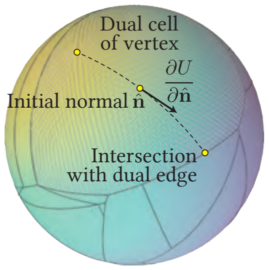
2.2.3 Tracing in a face's Gauss image. The situation is similar if $\hat{\mathbf{n}}$ is equal to a face f's normal (laying *in* its Gauss image by nature of being the same isolated point). A face is either *stable* (F0, rolling stops), or rolling continues onto a vertex (F1, like a cartwheel) or onto an edge (F2, like a hinge). We can precisely classify each case in the primary domain with the following steps:



(1) Split the plane of this face L into 7 regions: interior of the triangle, three "stripe" regions formed by each edge, and three cone regions at every vertex (see inset);
(2) Let $\tilde{\mathbf{c}}$ be the projection of the center of mass $\mathbf{c}$ onto L,
(3) If $\tilde{\mathbf{c}}$ is in a stripe region, then the next step in rolling is onto the edge corresponding to that stripe;
(4) If $\tilde{\mathbf{c}}$ is in a cone region, then the next step is onto the vertex corresponding to that cone;
(5) Otherwise $\tilde{\mathbf{c}}$ lies inside of the interior of the triangle and the face is stable (F0).

Before conducting any tracing we can classify all hull edges (as E1 or E2) and hull faces (as F0, F1, or F2). For E1 and F1 types we only need to determine the vertex that they roll onto. For E2 and F2 types we determine the face they roll onto; note that F2 faces roll onto an edge that is an E2 and rolls on to a face. This information is sufficient for doing a complete trace (see inset).

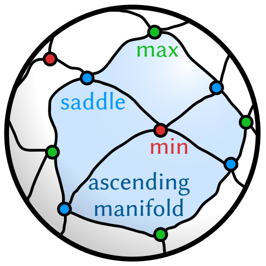
2.2.4 Complete trace. Then, for any given input downward orientation $\hat{\mathbf{n}}$, we can complete its trace to a stable face by iteratively rolling from one element (vertex, edge or face) to the next. Due to strict convexity, each step decreases U and the process is guaranteed to terminate. Note that after identifying which vertex's Gauss image $\hat{\mathbf{n}}$ lays at, all the subsequent steps consist of only local operations. A pseudo-code of this tracing procedure is given in Appendix A, algorithm 1.


Dual cell of vertex
$\frac{\partial U}{\partial \hat{\mathbf{n}}}$
Initial normal $\hat{\mathbf{n}}$
Intersection with dual edge

This is already a very efficient and robust method for laying objects to rest from an initial orientation. Examples of a tracing process on the Gauss map are shown in figure 7. A tracing process example in the ambient domain is also shown in figure 6. This task is found in the GRABCAD software to assist user's placement of 3D printed parts stably onto the build platform. Our proposed method is 400 to 60 times faster than a single bullet simulation (more complex geometry leads to larger speed-ups) and does not rely on tolerances or finicky collision detection/handling to determine its final orientation. In our comparisons with physics engines, we noticed that for some initial orientations $\mathcal{H}$ comes to rests on unstable faces, and even more surprisingly on edges; likely due to numerical issues and collision detection limits.

2.3 Categorizing all paths by constructing the Morse-Smale complex

The Morse-Smale complex of a scalar function over a 2D domain (such as the Gauss sphere) partitions into cells restricted upon which the function is monotonic [Morse 1934; Smale 1961]. The cells are bounded by separatrix curves connecting extrema to saddle points (see inset).



The "ascending manifold" for a local minimum corresponds to the union of cells containing all points that flow to that minimum. The area of the ascending manifold around the Gauss image of a stable face's $\hat{\mathbf{n}}_f$ is precisely proportional to the resting probability p_f. Our goal is to construct the Morse-Smale Complex *atop* the Gauss map of the input shapes convex hull $\mathcal{H}$, so that we can simply read off these areas.

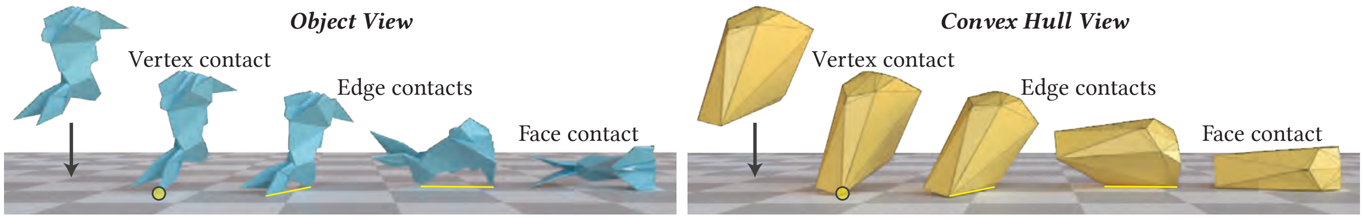


Fig. 6. Imagine an object falls from well above the floor in a general arbitrary orientation. Translating the object downward along the z direction decreases the gravitational potential until the object makes contact with the floor at some vertex on its convex hull. The center of mass will not lie directly above the vertex, causing a rotation around the vertex until another hull vertex meets the ground: a hull edge now lies on the ground. Once again, $\mathbf{c}$ lies off center from the edge causing rotation either over the new vertex or around the edge as a hinge. A hinge edge rotation stops when another vertex hits the ground: a face of the hull. This process may repeat: passing through additional vertices, edges and faces, but due to our general-position assumption it always stops when the center of mass lies directly above some *stable face* i.

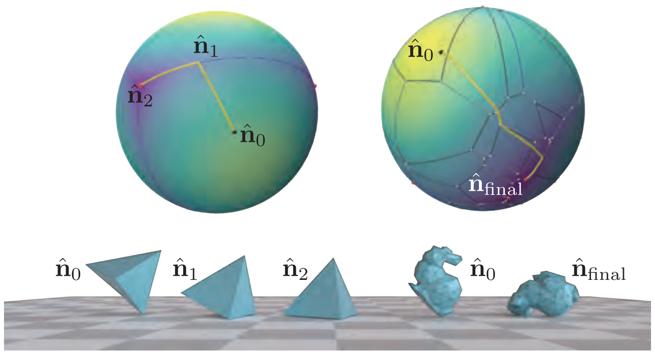


Fig. 7. We follow the gradient of U from an orientation $\hat{\mathbf{n}}_0$ to a local minimum of U for two examples: a tetrahedron and a bunny. Yellow curves show the orientation trajectories: piecewise-great-arc curves on the Gauss image.

2.3.1 *Local Minima.* Local minima occur at stable hull faces: the normal $\hat{\mathbf{n}}_i$ for every face i classified as a F0 (see Sec. 2.2.3) is recorded as a minimum.

2.3.2 *Local Maxima.* Local maxima occur at unstable hull vertices. For each vertex, we can identify the normal direction $\hat{\mathbf{n}}^\star_j$ which maximizes the height of the center of mass according to that vertex's function contribution U_j:

$$\hat{\mathbf{n}}^\star_j = \operatorname*{argmin}_{\hat{\mathbf{n}}} U_j(\hat{\mathbf{n}}) = \widehat{\mathbf{x}_j - \mathbf{c}}. \tag{6}$$

This direction does not necessarily result in vertex j touching the ground first, so a vertex j is a maximum vertex if and only if its maximizing normal is in its Gauss image ($\hat{\mathbf{n}}^\star_j \in \mathcal{N}_j$).

2.3.3 *Saddle points.* Saddle points only occur at type E2 "hinge" edges (see Sec. 2.2.2). When edge ij lies on the ground, consider rotating about the edge as a hinge until the center of mass lies directly above:

$$\hat{\mathbf{n}}^\star_{ij} = \widehat{\mathbf{c} - \mathbf{c}_{ij}} \tag{7}$$

$$\mathbf{c}_{ij} = \mathbf{x}_i + \frac{(\mathbf{c} - \mathbf{x}_i) \cdot \mathbf{v}_{ij}}{\mathbf{v}_{ij} \cdot \mathbf{v}_{ij}} \cdot \mathbf{v}_{ij} \tag{8}$$

where $\mathbf{c}_{ij} \in \mathbb{R}^3$ is the projection of the center of mass $\mathbf{c}$ onto the edge ij and $\mathbf{v}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ is the edge vector. This normal is a saddle point if and only if it lies in the Gauss image of the type E2 edge ($\hat{\mathbf{n}}^\star_{ij} \in N_{ij}$).

2.4 Tracing a Separatrix on the Gauss Map

A separatrix is a curve connecting a saddle point to a local minimum or maximum. Similar to tracing rolling paths in the Gauss map (see 2.2.4), we can avoid numerical integration by precisely constructing each segment of the separatrix. A separatrix is itself a gradient flow line [Smale 1961], so it is also a piecewise-great-arc curve on the Gauss map. Starting at a saddle point on an edge ij we follow the steepest descent/ascent directions (according to $\partial U_i/\partial\hat{\mathbf{n}}$ or $\partial U_j/\partial\hat{\mathbf{n}}$), passing through intersection points with Gauss images of other edges (also great arcs), until reaching a local minimum/maximum (pre-identified in a vertex's Gauss image). This algorithm requires only: the pre-identification of maximum vertices, saddle edges, and minimum (stable) faces, which can be done by a single pass over them; and a subroutine for intersecting great-arcs. Pseudo-code for the subroutine and the full procedure is given in Appendix A, algorithms 2 and 3. Some separatrix examples are shown for various shapes in figure 8. This procedure is very fast, as it can be seen as tracing gradients from a set of initial orientations (saddle points), which can only be as large as the number of edges of $\mathcal{H}$. Finally, we compute the area A_f of the ascending manifold for each stable faces (local minimum point of the Morse-Smale complex):

$$p_f = \frac{A_f}{4\pi}. \tag{9}$$

Each ascending manifold is a spherical polygon given by a set of unit vectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k$, we can also compute A_f as a sum over signed spherical triangle areas:

$$A_f = \sum_i \Omega(\hat{\mathbf{n}}, \mathbf{u}_i, \mathbf{u}_{i+1}) \tag{10}$$

where $\Omega(\hat{\mathbf{n}}, \mathbf{u}_i, \mathbf{u}_{i+1})$ is signed area of the spherical triangle with vertices $\{\hat{\mathbf{n}}, \mathbf{u}_i, \mathbf{u}_{i+1}\}$, vector $\hat{\mathbf{n}}$ (treated like the origin) can be taken to be any point on the sphere. When computing the area of a ascending manifold corresponding to face f, we take $\hat{\mathbf{n}}$ to be the normal of face f. This signed area can also be interpreted as the solid angle of a planar triangle with vertices at $\hat{\mathbf{n}}, \mathbf{u}_i, \mathbf{u}_{i+1}$; see [Van Oosterom and Strackee 1983] for details.

2.4.1 Special Cases. The procedure explained earlier can be simplified for certain class of shapes. If all faces of $\mathcal{H}$ are stable, and all vertices of $\mathcal{H}$ are unstable equilibria ($\hat{\mathbf{n}}_j^\star \in \mathcal{N}_j, \forall j$), then p_f for every face f is simply the normalized solid angle of face f seen from the center of mass of $\mathcal{H}$. This fact has lead to the solid angle heuristics used in earlier work (such as [Ngoi and Lim 1996], [Boothroyd and Ho 1977]), but these assumptions are extremely limiting and almost never hold for general shapes. Evaluation of our method on symmetric shapes, *e.g.* platonic solids, also reduces to computing solid angles of faces and results in uniform probabilities for all faces. We know by symmetry that this is indeed correct.

3 Manipulating the probabilities

In this section, we leverage the fast and differentiable nature of our algorithm to design new shapes or deform existing ones to achieve a desired resting probability distribution.

Starting from an initial convex shape $\mathcal{H}$, we first compute its resting probabilities $\{p_f\}$ and their gradients with respect to the vertex positions of $\mathcal{H}$. Using these gradients, we iteratively modify the shape until we find a convex surface $\mathcal{H}^*$ that matches the target resting probabilities, as described in Section 3.2. When the goal is to design a non-convex shape with specified resting probabilities, we begin with an initial non-convex surface and deform it so that its convex hull matches $\mathcal{H}^*$, as detailed in Section 3.4.

3.1 Differentiation

In order to build an ascending manifold and compute its area in section Section 2.4, we need to compute all separatrixes on its boundary. Each is a path $\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_k\}$ starting at a saddle point $\mathbf{u}_0 \in S^2$ of U, and ending at a local maximum or minimum $\mathbf{u}_k$. Every point $\mathbf{u}_i$ in this piecewise geodesic path is directly computed from the previous point $\mathbf{u}_{i-1}$, and the Gauss image of the edges along the path. In addition, $\mathbf{u}_0$ is simply expressed in terms of its corresponding edge position and the given center of mass. Therefore, all involved quantities can be computed locally, which makes computing gradients efficient with automatic differentiation. We use the resting probabilities and their gradients in the next sections to build models for designing arbitrary dice.

3.2 Dice Energy

In our design problems, the goal is to take a target discrete distribution $Q = \{q_1, \ldots, q_k\}$ and find a convex surface $\mathcal{H}^*$ with associated probabilities $\{p_f\}$, where the nonzero p_f's are a permutation of the q_i's.

We model this task by defining and optimizing an energy functional over convex shapes. Given a convex shape $\mathcal{H}$ with center of mass at $\mathbf{c}$, we define the energy:

$$E_{\text{diceSimple}}(\mathcal{H}, \mathbf{c}) = \sum_{f \in F(\mathcal{H})} \Big(p_f - q(f)\Big)^2 \tag{11}$$

Here, $q\colon F(\mathcal{H}) \to Q \cup \{0\}$ assigns target probabilities to faces of $\mathcal{H}$: exactly one face should be assigned to each $q_i \in Q$, while other faces are assigned zero probability.

A natural approach to minimizing $E_{\text{diceSimple}}$ and finding $\mathcal{H}^*$ is to start from an initial convex surface $\mathcal{H}^0$ and use a first-order method to deform $\mathcal{H}^0$ along the gradient of $E_{\text{diceSimple}}$, producing a sequence of surfaces $\{\mathcal{H}^t\}$ until the desired distribution is achieved.

The main difficulty in optimizing energies involving p_f lies in the topological changes that occur in the sequence $\{\mathcal{H}^t\}$ or the Morse–Smale complex of their potential functions. These changes can arise either from change in the mesh connectivity (which is determined uniquely by the vertex positions) or from changes in the Morse complex of the function U. Small perturbations to the geometry can cause critical events such as the splitting of a large stable face into multiple smaller ones, or the merging of multiple stable faces into a single one.

Because of these instabilities, $E_{\text{diceSimple}}$ is only piecewise continuous, and the energy landscape becomes difficult to navigate. Furthermore, maintaining a consistent assignment of target probabilities to faces becomes challenging at every step t. To address these issues, we instead take k normal directions $\mathbf{u}_i \in S^2$ to which we assign the target probabilities q_i ($1 \le i \le k$). These normals are not necessarily aligned with the face normals of $\mathcal{H}$. We then cluster stable faces based on proximity to the $\mathbf{u}_i$, with each cluster denoted by F_i (i.e., the set of faces whose normals lie in the Voronoi cell of $\mathbf{u}_i$).

This leads to a better-behaved energy:

$$E_{\text{dice}}(\mathcal{H}, \mathbf{c}) = \sum_i \left(\left(\sum_{f \in F_i} p_f \right) - q_i \right)^2 \tag{12}$$

With this new energy, events such as face splits or merges within a cluster have a much softer effect on E_{dice}, toning down discontinuities and improving optimization robustness. In practice, E_{dice} often converges to zero for most inputs, whereas $E_{\text{diceSimple}}$ tends to stall near discontinuities.

Moreover, this formulation allows for a simple policy for updating the directions $\mathbf{u}_i$: at each deformation step, $\mathbf{u}_i$ is updated to the normalized average of the stable face normals in cluster F_i.

While the original goal is to assign each probability q_i to exactly one stable face, minimizers of E_{dice} can occasionally produce clusters containing multiple stable faces whose combined probability matches q_i. This can be addressed by introducing a regularizer that softly encourages the merging of stable faces within each cluster. Additional regularizers are also used to enforce aesthetic and physically motivated constraints.

Center of Mass. When optimizing energies involving p_f's, we must decide whether we are designing a shape assuming uniform mass density (that uniquely determines the center of mass), or targeting a fixed center of mass $\mathbf{c}_0$ without considering the internal mass distribution. This choice affects the gradient of the dice energy with respect to the vertex positions of $\mathcal{H}$. Fortunately, either variant can be incorporated seamlessly into the algorithm using automatic differentiation.

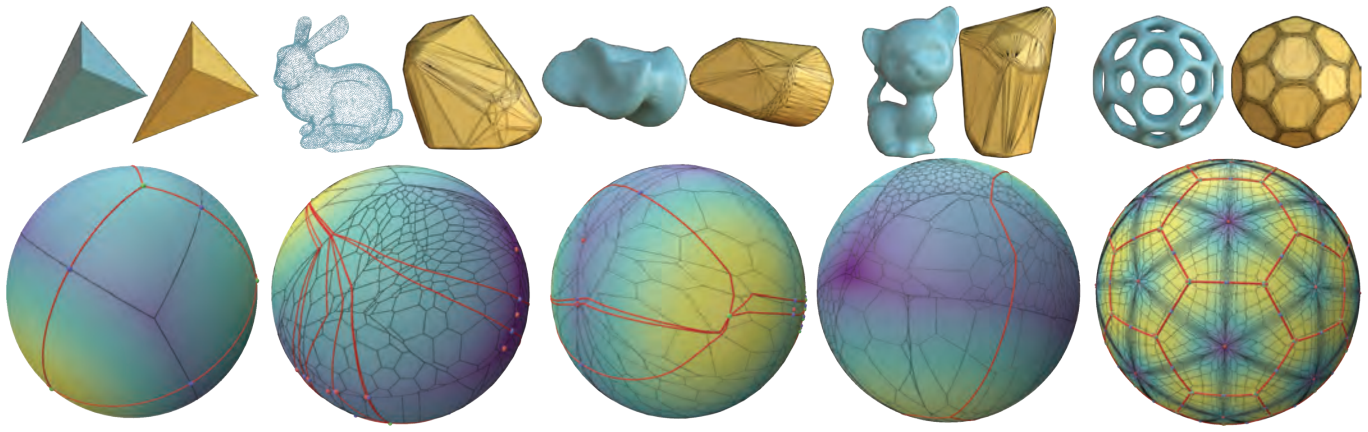


Fig. 8. Morse complex examples (red) drawn on the Gauss map (black) of the convex hull of some models. Stable face normals (red), edge saddle normals (blue), and vertex maximum normals (green) are also shown. For the Bunny example a point cloud is used as input; our method can take any type of input as long as a convex hull of the input can be computed.

3.3 Regularizers for Dice Energy

As mentioned earlier, it is often desirable to have a single stable face at each cluster. One may also prefer to have few unstable faces in the neighborhood of a stable face, leading to a large, flat stable face within each cluster. Moreover, in Section 4.1 we observe that having the stable face normal positioned away from the boundary of its ascending manifold improves agreement with simulation results that incorporate momentum, and possibly even with physical experiments. All of these properties can be incorporated into our optimization framework through regularization terms.

3.3.1 *Cluster Co-planar Energy.*

$$E_{cl}(\mathcal{H}, \mathbf{c}) = \sum_i \sum_{f_k, f_l \in F_i} \|\hat{\mathbf{n}}_{f_k} - \hat{\mathbf{n}}_{f_l}\|^2 \tag{13}$$

This term encourages the stable face normals within a cluster to align closely, potentially merging into fewer stable faces. This is particularly helpful since E_{dice} handles topological changes in the Morse complex of U gracefully; once E_{dice} converges to zero, E_{cl} further promotes merging of stable faces within each cluster.

3.3.2 *Barycenter Energy.*

$$E_{bc}(\mathcal{H}, \mathbf{c}) = \sum_i \sum_{f \in F_i} \|\hat{\mathbf{n}}_f - b(F_i)\|^2 \tag{14}$$

This term encourages the stable face normals within a cluster F_i to approach the barycenter $b(F_i)$ of the cluster region. The cluster region, formed by the union of the ascending manifolds of stable faces in F_i, resembles a closed spherical polygon. Rather than computing the exact barycenter, we approximate it by averaging and normalizing the boundary maximum points of the ascending manifolds.

3.3.3 *Neighbor Attraction Energy.*

$$E_{nb}(\mathcal{H}, \mathbf{c}) = \sum_i \sum_{\hat{\mathbf{n}}_f \in B_\epsilon(\mathbf{u}_i)} \|\mathbf{u}_i - \hat{\mathbf{n}}_f\|^2 \tag{15}$$

This term encourages neighboring faces around a stable face to align with the same normal, promoting coplanarity and producing stable faces with more isolated normals.

Adding up everything, we arrive at the final energy

$$E = E_{\text{dice}} + \lambda_{cl} E_{cl} + \lambda_{nb} E_{nb} + \lambda_{bc} E_{bc} \tag{16}$$

All terms in this energy can be computed alongside the area of ascending manifolds, without adding complexity to the algorithm. We use the Stan Math library [Carpenter et al. 2015] for automatic differentiation to compute gradients of $E(\mathcal{H}, \mathbf{c})$ with respect to the vertex positions of $\mathcal{H}$. In Figure 9, we demonstrate how each regularizer affects the final result.

To minimize E, we use gradient descent with line search. Note that directly updating vertex locations of $\mathcal{H}$ (denoted $\mathbf{x}$) along the gradient, i.e., $\mathbf{x}^{\text{new}} = \mathbf{x} + \nabla_{\mathbf{x}} E$, may result in a surface that is no longer convex. Therefore, for each update (especially during line search), we compute the convex hull of the new vertex set: $\mathcal{H}^{\text{new}} = \text{ConvHull}(\mathbf{x}^{\text{new}})$. This procedure often reduces the number of vertices in $\mathcal{H}^{\text{new}}$. While acceptable in most cases, it presents challenges in Section 5.2, where we work with high vertex-count surfaces. To mitigate this, we observe that the dice energy gradient is sparse, and apply Sobolev preconditioning to diffuse the gradient. As a result, we can find smoother convex shapes; instead of starting with a convex hull of 1000 vertices and ending with only 20, we typically retain around 120 vertices.

3.4 Inverse Convex Hulls

We can also start from non-convex shapes with uniform mass and deform them into objects with the desired resting probabilities, as discussed in Section 5.2.

Starting from an initial concave surface S^0 with center of mass $\mathbf{c}_0$ and convex hull $\mathcal{H}^0$, we apply the dice energy optimization to $\mathcal{H}^0$, assuming a fixed center of mass at $\mathbf{c}_0$. Denoting the resulting

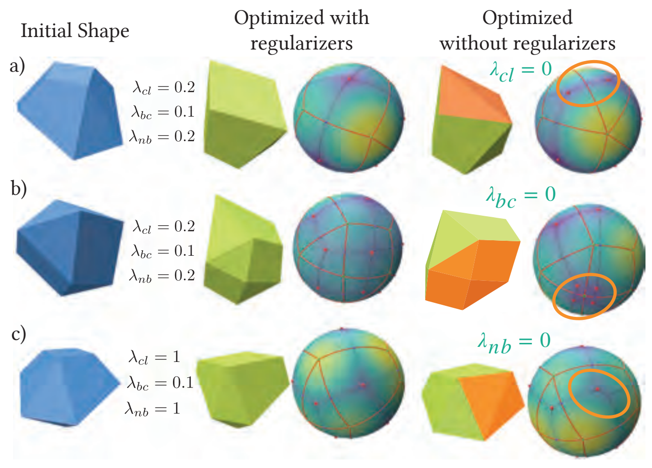


Fig. 9. A dice design example from Section 5.1, where using all regularizers produces desirable shapes (middle column). On the right, we ablate each regularizer to show the adverse effects (highlighted faces). **(a)** Removing the cluster co-planar energy results in two distinct stable faces in the top cluster whose normals are not closely aligned, although their probabilities still sum correctly. **(b)** Removing the barycenter energy allows stable face normals to drift toward the boundaries of their ascending manifolds. Also leading to clumping of normals that should remain distinguishable. **(c)** Without neighbor attraction energy, some polygonal faces split into separate stable and unstable regions. While this does not affect the resting distribution, it may be aesthetically undesirable.

convex hull by $\mathcal{H}^*$, we then reconstruct a surface S^* by solving:

$$S^* = \arg\min_{S} \ d(S, S^0) \tag{17}$$

$$\text{s.t.} \ \operatorname{ConvHull}(S) = \mathcal{H}^* \tag{18}$$

$$\mathbf{c}(S) = \mathbf{c}^0 \tag{19}$$

where $d(S, S^0)$ is a combination of elastic surface energies that measures how much S and S^0 look alike. Although S^0 is a natural reference, it is not mandatory; any S fitting inside $\mathcal{H}^*$ with minor deformation could suffice.

We solve this problem by relaxing constraints 18 and 19 into soft penalties. First, we construct an intermediate surface $\tilde{S}$ satisfying $\operatorname{ConvHull}(\tilde{S}) = \mathcal{H}^*$. Then, we iteratively deform $\tilde{S}$ to obtain S^*, ensuring that $\operatorname{ConvHull}(S^*) = \mathcal{H}^*$ remains true while moving the center of mass to $\mathbf{c}^0$. Details of this procedure are provided in Appendix B.

Finally, note that constraint 19 could be ignored if one were willing to hollow out the interior of $\tilde{S}$ as in [Prévost et al. 2013]. However, maintaining a uniform mass distribution simplifies fabrication, particularly for standard 3D printing processes.

4 Evaluation

4.1 Comparison to Simulation

In order to validate our approach, We compare the resting probabilities predicted by our model with those obtained from a rigid body simulator. Our simulation setup involves repeatedly dropping a test shape onto a flat plane from a fixed height and with uniformly

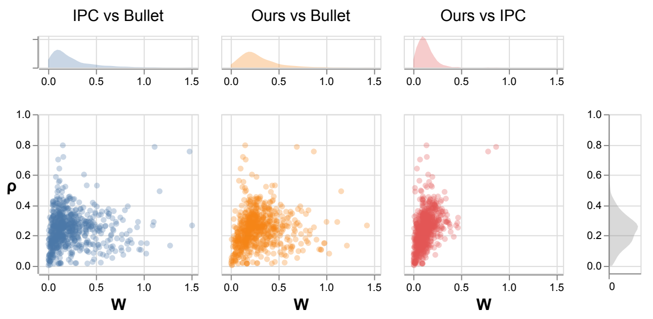


Fig. 10. In the marginal density plot above, we visualize the relationship between of the optimal transport cost (W) and ρ for probabilities obtained from our model, Rigid IPC simulation, and Bullet simulations. The density distribution of W and ρ are on the top and right margin. We see a closer agreement between our model and Rigid IPC, than the other two pairs of comparisons.

sampled initial orientations. The resulting resting orientations are recorded, and the final probabilities are then computed from these statistics.

For the initial orientation, we sample directions $\mathbf{u} \in S^2$ and use them as the default up orientations of the test shape. Specifically, we choose $\mathbf{u}$ from the vertices of a fine icosahedral subdivision of the sphere $(\mathbf{u}_1, \ldots, \mathbf{u}_k)$ and weigh the drop outcome at each $\mathbf{u}_i$ by the dual area of vertex i. We use a subdivision with 1212 vertices.

We use the Wasserstein distance (see [Villani et al. 2009]) to measure the deviation between our predictions and simulation results. Denoting $P = \{\hat{\mathbf{n}}_f, p_f\}$ as the stable face normals and associated probabilities from our model, and $Q = \{\hat{\mathbf{n}}_f, q_f\}$ from simulation, the Wasserstein distance is defined as:

$$W(P, Q) = \min_{\gamma \in \Gamma(P,Q)} \sum_{i=1}^{m} \sum_{j=1}^{m} \gamma_{ij} d(\hat{\mathbf{n}}_i, \hat{\mathbf{n}}_j)$$

where $\Gamma(P, Q)$ is the set of *transport plans* from P to Q, and $d\colon S^2 \times S^2 \longrightarrow \mathbb{R}$ denotes the geodesic distance on S^2. Finding W is straightforward since the supports of P and Q are discrete, consisting of stable orientations (a small number of points on S^2 for most shapes).

We use this measure to gracefully handle large probability discrepancies between faces with very close normals. Due to numerical sensitivity in dynamic simulation, such faces can naturally have large deviations.

We compare probabilities obtained from our model, Rigid IPC, and Bullet across a subset of 700 shapes from the Breaking Bad dataset [Sellán et al. 2022], including zeroth and second-order fracture modes. Results are shown in Figure 10. We observe that our model agrees closely with Rigid IPC, while both differ significantly from Bullet.

Quality of Rigid Body Simulation. Although Rigid IPC is significantly more robust than Bullet, it does not model elastic collisions (bounces) with the ground. During our experiments, we found Bullet to be numerically unreliable, especially for surfaces with convex hulls containing a high number of vertices. Bullet simulations often terminated in configurations not at static equilibrium—e.g., stuck on unstable faces or edges.

These unstable cases usually have normals and potential energy very close to nearby *stable* faces. In such instances, we map the result to the nearest stable face. On average, these "stuck" cases account for about 20% of samples, with higher occurrence for convex hulls exceeding 1000 vertices and containing small dihedral angles. We attribute this to inaccuracies in Bullet's collision handling and integration. In about 1% of cases, the simulation froze on an edge or far from any stable normal. Despite extensive parameter tuning, no single parameter set yielded reliable behavior across all test shapes, leading us to discard such samples.

More importantly, the cost of computing even these approximate probabilities via simulation is orders of magnitude more expensive than our method. Average speed of a single instance of a Bullet simulation is 0.04 seconds. We speed up Rigid IPC experiments by using the convex hull of the input shape for collision with the ground plane, while keeping the original mass moments. After this speed up, a single simulation instance takes 0.1 to 10 seconds on a single core, depending on number of vertices of the convex hull. This leads to hours of simulation for most common shapes.

Based on physical experiments conducted on a set of shapes (see Section 6), we find that for simpler shapes with relatively few vertices, Bullet tends to better match experimental outcomes – possibly due to its handling of elasticity. However, for more complex shapes, Bullet often produces invalid results, while Rigid IPC yields more accurate and reliable predictions.

Difference Between Geometric Model and Rigid Body Simulation. Across our experiments, we observe that differences between our model and simulations tend to occur when two neighboring ascending manifolds have asymmetric distances from their local minima to their shared boundary. Intuitively, when one minimum lies closer to the boundary, momentum may cause "spillover" across the two ascending manifolds.

We quantify this heuristic for a shape $\mathcal{H}$ via:

$$\rho(\mathcal{H}) = \sum_{f_1, f_2 \in C} \min\{p_{f_1}, p_{f_2}\} \left(\phi(f_1, f_2) - 0.5\right) \tag{20}$$

where $\phi(f_1, f_2)$ measures distance asymmetry between two neighboring ascending manifolds (with stable faces f_1, f_2):

$$\phi(f_1, f_2) = \frac{\max\{\angle(\hat{\mathbf{n}}_{f_2}, \hat{\mathbf{n}}_0^\star), \angle(\hat{\mathbf{n}}_{f_1}, \hat{\mathbf{n}}_0^\star)\}}{\angle(\hat{\mathbf{n}}_{f_1}, \hat{\mathbf{n}}_0^\star) + \angle(\hat{\mathbf{n}}_{f_2}, \hat{\mathbf{n}}_0^\star)} \tag{21}$$

Here, $\hat{\mathbf{n}}_0^\star$ is the saddle point normal on the boundary between f_1 and f_2's ascending manifolds, and $\angle(\mathbf{u}, \mathbf{v})$ denotes the angle between unit vectors $\mathbf{u}$ and $\mathbf{v}$.

In Figure 10, we plot ρ versus the Wasserstein distance W for all dataset shapes. We observe a significant correlation between ρ and W; 0.47 Pearson correlation coefficient ([Pearson 1895]) for our model versus Rigid IPC, 0.13 for ours vs Bullet, and 0.07 for Bullet vs Rigid IPC (with respective p-values 1e-20, 5e-4, and 0.06).

4.2 Design Aid

The fast and deterministic nature of our method enables a range of applications from designing realistic scenes, to providing interactive, real time feedback to a user for deforming a shape or moving its center of mass while monitoring resting probabilities. In Figure 11 we show a rendered scene of a living room cluttered with toys resting on their most probable orientation obtained from our algorithm. In a video in the supplement, we show a user interactively moves the center of mass, demonstrates that the MS complex (hence probabilities) can be recomputed in real time. One could also use our method in conjunction with earlier work which restricts the center of mass to a region in space to make a particular face stable [Prévost et al. 2013; Bächer et al. 2014; Wang and Whiting 2016]; one could then use our method to find the center of mass location that maximizes the probability of the corresponding stable orientation. In another example in Figure 12 we find the center of mass for which the faces with top 6 probabilities are close to each other.

5 Inverse Design

In this section we solve the inverse problem of designing shapes with target distributions of resting probabilities. The generality of our framework enables us to handle distributions, and design dice, that cannot be achieved using simpler forms of analysis (e.g., based only on solid angle [Hurst and Tandiman 2024]). For instance, in some cases, to get the target probabilities, unstable faces must exist; *e.g.* a shape with exactly 3 stable faces.

5.1 Convex Dice

Here we find convex polyhedral solid shapes with uniform mass density (similar to a fair die) with unconventional probability distributions. We consider the following cases:

Binomial die. A single die representing outcomes of k fair coin flips, $B(k, 0.5)$. We consider $2 \leq k \leq 6$. The results are shown in Figure 13. In all examples unstable faces exist and are used to achieve the goal distribution.

For all these examples we start with a cylindrical prism with $k+1$ sides, and pointy caps. Initial shapes and final results are shown in Figure 13.

2d6. A single die representing outcomes of sum of outcomes two fair 6-sided die, from $1+1=2$ to $6+6=12$. In other words, the goal is an 11-sided shape with the following distribution:

$$\left\{\frac{1}{36}, \frac{2}{36}, \frac{3}{36}, \frac{4}{36}, \frac{5}{36}, \frac{6}{36}, \frac{5}{36}, \frac{4}{36}, \frac{3}{36}, \frac{2}{36}, \frac{1}{36}\right\} \tag{22}$$

For this example we start with an 11-sided polyhedron and where all the faces are stable and find a shape with the same connectivity that has the mentioned probabilities. We use three different initializations and find the shapes shown in Figure 14.

For each example, the user needs to provide an initial convex shape, that will be deformed into another convex shape $\mathcal{H}^*$, that minimizes the dice energy. Also as mentioned in section Section 3.2, the initial assignment of probabilities q_i to normals $\mathbf{u}_i$, should be made by the user. These initial design choices give the user some leverage on how the final shape looks. However, these initial choices and the non-convex energies that we have in section Section 3.2 lead to not having guaranteed convergence, and some investigation of initial choices and fine-tuning parameters is naturally required. Furthermore, not every initial choice will lead to a solution, or a

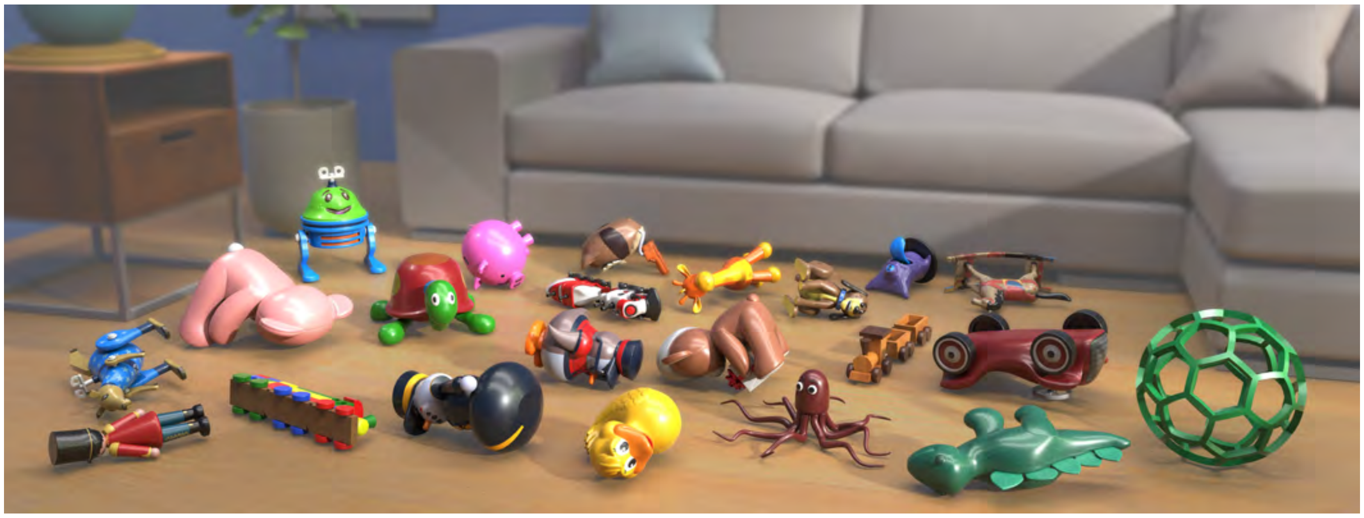


Fig. 11. Our algorithm computes the probability that a given rigid body lands in each possible equilibrium configuration, without performing any dynamical simulation. Here we use it to synthesize a household scene cluttered with toys; for each model we show the most likely configuration.

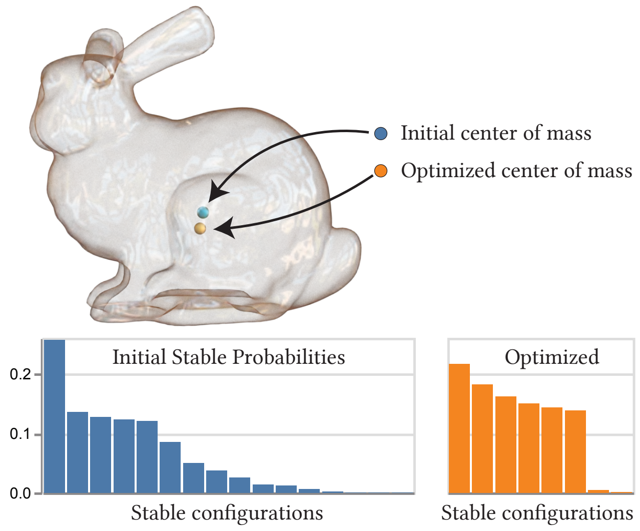


Fig. 12. By optimizing the center of mass, we reduced the number of stable configurations for the classic Stanford bunny, while making the probability of settling in its top six configurations more evenly distributed.

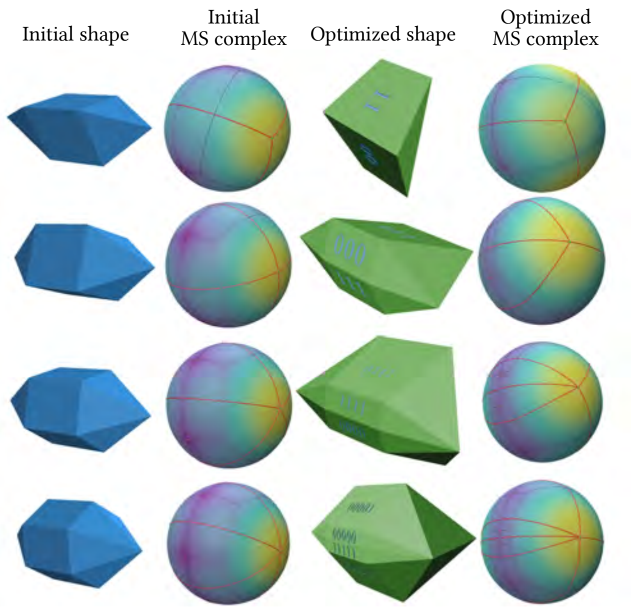


Fig. 13. Binomial die for $k = 2, 3, 4, 5$ (respectively from top to bottom) with different initial cylindrical prism shapes. Each initial and final shape has unstable faces (pointy caps) that help, and sometimes are essential (for $k = 2$) with finding the shape. Outcome of the optimized shapes are engraved on corresponding faces in a post-process. The dice energy parameters used for these shapes ,$(\lambda_{cl}, \lambda_{bc}, \lambda_{nb})$ from top to bottom: $(0.1, 2, 0)$, $(0.1, 2, 0.03)$, $(0.1, 2, 0.1)$, $(0.1, 2, 0.1)$, $(0.2, 2, 0.2)$.

solution might simply not exist (given the polyhedral connectivity), or it might be hard to find a good enough solution. That being said, our method is also far superior to the alternative of trying to design dice with target probabilities by hand: to our knowledge, no previous solutions are known for many of the examples shown above, nor did we find algorithms for designing such dice (e.g., with a mix of stable and unstable faces). Furthermore the evaluation and differentiation steps of our method are both extremely fast and make an easy tool to investigate many design choices.

5.2 Concave Dice

Here we find asymmetric fair die with a given number of stable sides. We start from convex hull of some common models and after finding an optimal convex hull, we follow the procedure from Section 3.4

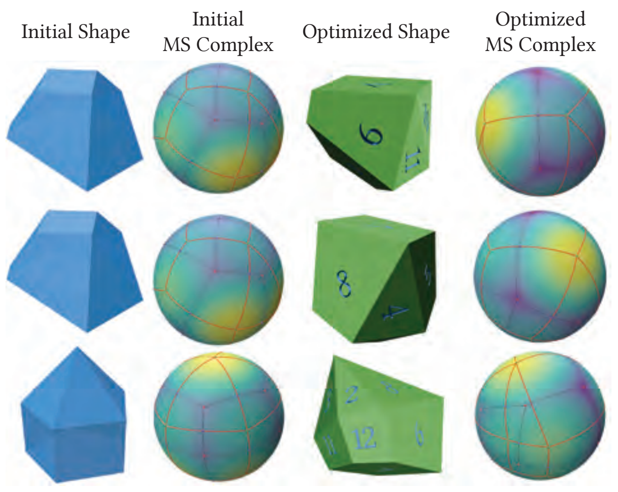


Fig. 14. A variety of dice representing the distribution of the sum of two six-sided dice are shown, all starting from different polygonal shapes with 11 faces. The first and second rows use the same initial shape but different probability assignments to faces. In all these examples, the goal is to keep all faces stable and polygonal faces planar. Outcome of the optimized shapes are engraved on corresponding faces in a post-process.The dice energy parameters used for these shapes, $(\lambda_{cl}, \lambda_{bc}, \lambda_{nb})$, from top to bottom: $(1, 0.1, 1)$, $(0.8, 0.01, 0.8)$, $(0.2, 2, 0.2)$.

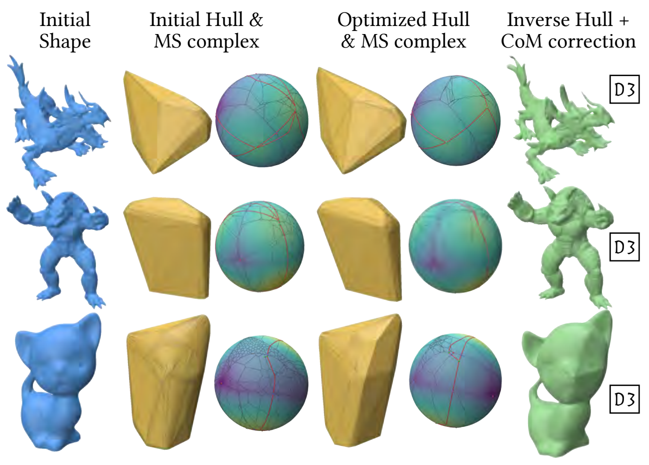

Initial Shape
Initial Hull & MS complex
Optimized Hull & MS complex
Inverse Hull + CoM correction
D3
D3
D3

Fig. 15. Fair concave three-sided die obtained from deforming some popular models. It is often not clear what type of dice is achievable, and a careful search of parameters needs to be carried out to find a dense convex hull with a low dice energy. The amount of deformation increases from top to bottom: minor changes in the dragon model; minor changes to the armadillo's fingers and back, with a low frequency shift in the torso area; the kitten example goes through larger deformations. The dice energy parameters used for these shapes, $(\lambda_{cl}, \lambda_{bc}, \lambda_{nb})$), from top to bottom: $(0.01, 0.01, 0.01)$, $(0.05, 0.3, 0.05)$, $(0.005, 0.3, 0.005)$.

to find a desired concave shape; see Figure 15. The clustered dice energy from Section 3.2 and the follow-up regularizers are key to finding these examples; the starting shapes often have many low probability faces, and because of the high vertex count of the convex hulls, lots of topological changes can happen to the MS complex while deforming the convex hull. In some final results in Figure 15 (like dragon d3 and kitten d3), more ascending manifolds than expected are present. However, because of the regularizer terms in Section 3.2, often these ascending manifolds have a very small area and their local minimum is very close to their boundary; resulting their corresponding configuration to have a negligible probability in presence of momentum (in a simulation or physical experiment). In other cases, multiple neighboring ascending manifolds have their local minima clumped together as a result of being in the same cluster (by Section 3.2 notation), leading to a negligible difference in the shape's orientation when resting on them; so they can be counted as a single resting orientation.

6 Physical Experiments

In this section we experiment with physically fabricated die that were designed in Section 5. Since all the shapes were designed with uniform mass density assumption, an off the shelf printer can be used for manufacturing. Printed dice are shown in the teaser (Figure 1) and in Figure 16. To validate that the expected distributions are achievable, we manually roll each dice on a flat wooden surface M times, where M varies depending on the number of stable faces of the dice; ranging from 300 to 1000 times for each die. A video consisting of recordings of some rolling instances is provided in the supplementary material. The resulting probabilities, obtained by dividing each outcome's frequency by the total number of rolls, is reported in Figure 16. Even though the experiments have a lot of momentum and the die go through lot of bounces in each rolling instance, we observe that our purely geometric algorithm is a good prediction of the resting probabilities.

To ensure fairness in experiments, a die is shuffled and rolled from approximately the same height on the same hard wooden floor, similar to how dice are rolled in a board game. The momentum and bounces off the floor make it impossible to have any control over the final resting orientation. These experiments were also conducted by multiple people to avoid any recurring tossing bias.

The *STL* files of the dice are provided in the supplementary material and can be used to fabricate the dice with any 3D printer that facilitates 100% infill density. In our experience, ensuring uniform 100% infill density is key to accurately fabricating these dice and reproduce the desired probabilities.

We observe that some examples behave as expected and some deviate from our probabilities, and even probabilities found by Bullet/IPC simulations. These observations are mostly in line with our assessment in Section 4.1. All the empirical, expected, and simulated distributions are provided in supplementary material.

7 Limitations and Future Work

We make strong modeling assumptions in order to enable analysis and inverse design. For one, we assume that initial configurations of the body is distributed uniformly over SO(3). It seems plausible that this assumption might be relaxed even within the no-momentum framework—for instance, if we know *a priori* that initial configurations fall only within a subset of SO(3), it may not be too difficult to

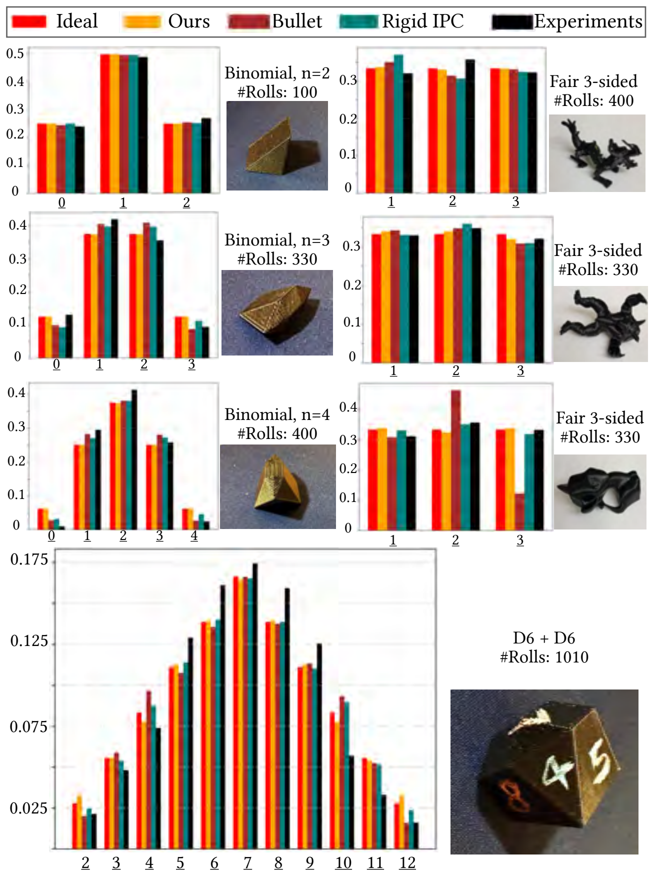


Fig. 16. Result of experiments with 3D printed versions of some of the dice in Section 5. We see better agreement with our model when the shape does not have very low probable faces. Three different Binomial die from Figure 13, three fair 3-sided concave die from Figure 15, and one of the 2D6 dice (from Figure 14) are experimented with.
Important: *since statistics are computed using a finite number of trials, one should not expect probabilities to match exactly—even for perfectly fair dice. E.g., a fair coin flipped just once cannot possibly match the expected 50/50 probability distribution. Instead, there is necessarily variance in the empirical probabilities, which decreases with the number of trials. See Section 4.1 for discussion of how we quantify the difference between probability distributions.*

exclude corresponding regions of the Gauss sphere from our calculations. On the other hand, incorporating some kind of momentum into our geometric analysis is a more challenging question—which we leave to future work.

Our approach might also be adapted to predict the orientation of buoyant objects that float on water. In general, developing a more geometric approach to the statistical analysis of equilibria appears to be a fruitful direction for future work in computational design.

For the inverse design examples in this paper, we always begin with a reference shape and deform it to match a target distribution. Although the process is fully automatic, incorporating semi-automatic steps could lead to more aesthetically pleasing results. For instance, a user could manually adjust the shape to better fit the target convex hull or to achieve the desired center of mass. In some cases, the dice energy fails to fully converge, or the deformations of concave shapes become too large, resulting in self-intersections. Given the efficiency of our forward computation, we believe that a semi-automatic design approach could effectively address most of these issues.

Finally, when designing real physical objects, there are many fabrication issues to consider. For example, very skinny faces, though statistically correct, may be difficult to realize accurately using a given 3D printing process. In this paper we use physical models merely as a mechanism for evaluation, and leave questions of robust manufacturing to future work.

Acknowledgments

This work was generously supported by NSF awards 2212290, 1943123, NSERC Discovery (RGPIN–2022–04680), the Ontario Early Research Award program, the Canada Research Chairs Program, a Sloan Research Fellowship, the DSI Catalyst Grant program and gifts from Adobe Systems. The authors also thank Joseph Sharp for resin printing the dice, Zoë Marschner and Olga Guțan for helping with conducting and recording the dice experiments, and Carnegie Mellon University TechSpark 3D printing facilities for enabling and helping with dice fabrication.

References

Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. 2014. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–10.
Ulrich Bauer, Carsten Lange, and Max Wardetzky. 2012. Optimal topological simplification of discrete functions on surfaces. *Discrete & computational geometry* 47 (2012), 347–377.
G Boothroyd and C Ho. 1977. Natural resting aspects of parts for automatic handling. (1977).
Bob Carpenter, Matthew D. Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt. 2015. The Stan Math Library: Reverse-Mode Automatic Differentiation in C++. *arXiv preprint arXiv:1509.07164* (2015). https://arxiv.org/abs/1509.07164
Erwin Coumans and Yunfei Bai. 2016. Pybullet, a python module for physics simulation for games, robotics and machine learning. (2016).
Shen Dong, Peer-Timo Bremer, Michael Garland, Valerio Pascucci, and John C Hart. 2006. Spectral surface quadrangulation. In *Acm siggraph 2006 papers*. 1057–1066.
Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M. Kaufman, and Daniele Panozzo. 2021. Intersection-free Rigid Body Dynamics. *ACM Transactions on Graphics (SIGGRAPH)* 40, 4, Article 183 (2021).
Hongbo Fu, Daniel Cohen-Or, Gideon Dror, and Alla Sheffer. 2008. Upright orientation of man-made objects. In *ACM SIGGRAPH 2008 Papers* (Los Angeles, California) *(SIGGRAPH '08)*. Association for Computing Machinery, New York, NY, USA, Article 42, 7 pages. doi:10.1145/1399504.1360641
Eitan Grinspun, Anil N Hirani, Mathieu Desbrun, and Peter Schröder. 2003. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Citeseer, 62–67.
Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. https://www.gurobi.com
Attila Gyulassy and Vijay Natarajan. 2005. *Topology-based simplification for feature extraction from 3D scalar fields*. IEEE.
Christian Hafner, Mickaël Ly, and Chris Wojtan. 2024. Spin-It Faster: Quadrics Solve All Topology Optimization Problems That Depend Only On Mass Moments. *ACM Trans. Graph.* 43, 4, Article 171 (sep 2024), 13 pages. doi:10.1145/3658194
Kai Hormann and Günther Greiner. 2000. MIPS: An efficient global parametrization method. *Curve and Surface Design: Saint-Malo 1999* (2000), 153–162.
Paul R Hurst and Vanessa Tandiman. 2024. The Centroid Solid Angle and Probability Models of Square Prism Dice Rolls. In *2024 Joint Mathematics Meetings (JMM 2024)*. AMS.
John C Kern. 2006. Pig data and Bayesian inference on multinomial probabilities. *Journal of Statistics Education* 14, 3 (2006).
Timothy Langlois, Ariel Shamir, Daniel Dror, Wojciech Matusik, and David IW Levin. 2016. Stochastic structural analysis for context-aware design and fabrication. *ACM*

Transactions on Graphics (TOG) 35, 6 (2016), 1–13.

Zishun Liu, Juyong Zhang, and Ligang Liu. 2016. Upright orientation of 3D shapes with convolutional networks. *Graphical Models* 85 (2016), 22–29.

Marco Livesu, Stefano Ellero, Jonàs Martínez, Sylvain Lefebvre, and Marco Attene. 2017. From 3D models to 3D prints: an overview of the processing pipeline. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 537–564.

Lin Lu, Andrei Sharf, Haisen Zhao, Yuan Wei, Qingnan Fan, Xuelin Chen, Yann Savoye, Changhe Tu, Daniel Cohen-Or, and Baoquan Chen. 2014. Build-to-last: Strength to weight 3D printed objects. *ACM Transactions on Graphics (ToG)* 33, 4 (2014), 1–10.

Jacopo De Grossi Mazzorin and Claudia Minniti. 2013. Ancient use of the knuckle-bone for rituals and gaming piece. *Anthropozoologica* 48, 2 (2013), 371–380.

David Moffatt. 1977. Pass the Piggies. https://winning-moves.com/product/pass-the-pigs Dice game.

Marston Morse. 1934. *The Calculus of Variations in the Large.*

Przemyslaw Musialski, Thomas Auzinger, Michael Birsak, Michael Wimmer, and Leif Kobbelt. 2015. Reduced-order shape optimization using offset surfaces. *ACM Trans. Graph.* 34, 4 (2015), 102–1.

Przemyslaw Musialski, Christian Hafner, Florian Rist, Michael Birsak, Michael Wimmer, and Leif Kobbelt. 2016. Non-linear shape optimization using local subspace projections. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–13.

BKA Ngoi and LEN Lim. 1996. Analysing the natural resting aspect of a component for automated assembly using the energy envelope method. *The International Journal of Advanced Manufacturing Technology* 12 (1996), 132–136.

Karl Pearson. 1895. VII. Note on regression and inheritance in the case of two parents. *proceedings of the royal society of London* 58, 347-352 (1895), 240–242.

Jovan Popović, Steven M Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. 2000. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 209–217.

Romain Prévost, Moritz Bächer, Wojciech Jarosz, and Olga Sorkine-Hornung. 2016. Balancing 3D Models with Movable Masses.. In *VMV*.

Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. 2013. Make it stand: balancing shapes for 3D fabrication. *ACM Trans. Graph.* 32, 4, Article 81 (jul 2013), 10 pages. doi:10.1145/2461912.2461957

Jorrit Rouwe. 2024. Jolt Physics. https://github.com/jrouwe/JoltPhysics.

Patrick Schmidt, Janis Born, David Bommes, Marcel Campen, and Leif Kobbelt. 2022. TinyAD: Automatic Differentiation in Geometry Processing Made Simple. *Computer Graphics Forum* 41, 5 (2022).

Silvia Sellán, Yun-Chun Chen, Ziyi Wu, Animesh Garg, and Alec Jacobson. 2022. Breaking bad: A dataset for geometric fracture and reassembly. *Advances in Neural Information Processing Systems* 35 (2022), 38885–38898.

Silvia Sellán, Jack Luong, Leticia Mattos Da Silva, Aravind Ramakrishnan, Yuchuan Yang, and Alec Jacobson. 2022. Breaking Good: Fracture Modes for Realtime Destruction. *ACM Transactions on Graphics* (2022).

Stephen Smale. 1961. On Gradient Dynamical Systems. *Annals of Mathematics* 74, 1 (1961), 199–206.

Yaroslav I Sobolev, Ruoyu Dong, Tsvi Tlusty, Jean-Pierre Eckmann, Steve Granick, and Bartosz A Grzybowski. 2023. Solid-body trajectoids shaped to roll along desired pathways. *Nature* 620, 7973 (2023), 310–315.

Ondrej Stava, Juraj Vanek, Bedrich Benes, Nathan Carr, and Radomír Mech. 2012. Stress relief: improving structural strength of 3D printable objects. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–11.

A. Van Oosterom and J. Strackee. 1983. The Solid Angle of a Plane Triangle. *IEEE Transactions on Biomedical Engineering* BME-30, 2 (1983), 125–126. doi:10.1109/TBME.1983.325207

Péter L Várkonyi and Gábor Domokos. 2006a. Mono-monostatic bodies: the answer to Arnold's question. *Math. Intelligencer* 28, 4 (2006), 34–38.

Péter L Várkonyi and Gábor Domokos. 2006b. Static equilibria of rigid bodies: dice, pebbles, and the Poincaré-Hopf theorem. *Journal of Nonlinear Science* 16 (2006), 255–281.

Cédric Villani et al. 2009. *Optimal transport: old and new*. Vol. 338. Springer.

Lingfeng Wang and Emily Whiting. 2016. Buoyancy optimization for computational fabrication. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 49–58.

Tino Weinkauf, Yotam Gingold, and Olga Sorkine. 2010. Topology-based smoothing of 2D scalar fields with C1-continuity. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1221–1230.

Jun Wu, Lou Kramer, and Rüdiger Westermann. 2016. Shape interior modeling and mass property optimization using ray-reps. *Computers & Graphics* 58 (2016), 66–72.

Juzhan Xu, Minglun Gong, Hao Zhang, Hui Huang, and Ruizhen Hu. 2023. Neural Packing: from Visual Sensing to Reinforcement Learning. *ACM Trans. Graph.* 42, 6, Article 267 (dec 2023), 11 pages. doi:10.1145/3618354

Qingnan Zhou, Julian Panetta, and Denis Zorin. 2013. Worst-case structural analysis. *ACM Trans. Graph.* 32, 4 (2013), 137–1.

A Pseudocode

A.1 Tracing gradients

Here we write out the procedure described for tracing a given initial orientation to a stable orientation. We assume that element classes are identified as explained in the main paper.

Algorithm 1 TraceGradient($\hat{\mathbf{n}}_0, \mathcal{H}, \mathbf{c}$)

```
Input: A unit vector $\hat{\mathbf{n}}_0$ as the initial orientation, A convex shape
       $\mathcal{H}$ uniquely determined by a point set $\{p_1, p_2, \dots, p_n\}$ in $\mathbb{R}^3$,
       A point $\mathbf{c}$ in $\mathbb{R}^3$ inside $\mathcal{H}$ determining the center of mass.
Output: A sequence of unit vectors $\mathtt{N} = \{\hat{\mathbf{n}}_0, \hat{\mathbf{n}}_2, \dots, \hat{\mathbf{n}}_k\}$ where
        $\hat{\mathbf{n}}_i, \hat{\mathbf{n}}_{i+1}$ determines a great arc segment that is along the
        gradient flow of $U$.
 1: $\mathtt{N} \leftarrow \{\hat{\mathbf{n}}_0\}$    ▷Initialize with the first orientation
 2: elem ← ElementWithNormal($\hat{\mathbf{n}}_0$)    ▷Get the unique face, edge,
    or vertex that has $\hat{\mathbf{n}}_0$ as its normal, prioritizing faces, then edges,
    when the normal is shared.
 3: $\hat{\mathbf{n}} \leftarrow n_0$
 4: while elem is not stable face do
 5:     if elem is hinge-type then    ▷Hinge edge or face
 6:         elem ← NextFace(elem)    ▷Face that elem hinge-rolls
    onto
 7:         $\hat{\mathbf{n}}$ ← Normal(elem)    ▷A face normal
 8:     else    ▷ elem is a vertex, or a cartwheel-type edge/face.
 9:         elem ← NextVertex(elem)
10:         for $\mathtt{elem}_{adj}$ ∈ AdjEdges(elem) do    ▷Neighboring edges
11:             $f1, f2$ ← AdjFaces($\mathtt{elem}_{adj}$)
12:             $\hat{\mathbf{n}}_{f1}, \hat{\mathbf{n}}_{f2}$ ← Normal($f1$), Normal($f2$)
13:             $\hat{\mathbf{n}}_{next}$ ← RayArcInt($\hat{\mathbf{n}}^\star_{\mathtt{elem}}, \hat{\mathbf{n}}, \hat{\mathbf{n}}_{f1}, \hat{\mathbf{n}}_{f2}$))    ▷Move
    along the gradient arc until elem's Gauss image boundary
14:             if $\hat{\mathbf{n}}_{next} \neq 0$ then    ▷Intersection; next normal found
15:                 $\hat{\mathbf{n}} \leftarrow \hat{\mathbf{n}}_{next}$
16:                 $\mathtt{N} \leftarrow \mathtt{N} \cup \{\hat{\mathbf{n}}\}$
17:                 elem ← ElementWithNormal($\hat{\mathbf{n}}$)
18:                 break    ▷Don't check other neighbor edges
19: end while
20: return N
```

The subroutine RayArcInt in Algorithm 2 is used for intersecting an arc-ray that starts at $\hat{\mathbf{n}}^\star$ and move towards $\hat{\mathbf{n}}$; $\hat{\mathbf{n}}_{f1}, \hat{\mathbf{n}}_{f2}$ are endpoints of the Gauss image of an edge (the two neighboring face normals).

Algorithm 2 RayArcInt($\hat{\mathbf{n}}^\star, \hat{\mathbf{n}}, \hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2$)

```
Input: Unit vectors $\hat{\mathbf{n}}^\star, \hat{\mathbf{n}}, \hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2 \in \mathbb{R}^3$
Output: A unit vector $\hat{\mathbf{n}}_{int}$ if the great arc segments $(\hat{\mathbf{n}}^\star, \hat{\mathbf{n}})$ and
        $(\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2)$ intersect, $\mathbf{0}$ otherwise.
1: $\mathbf{d} \leftarrow (\hat{\mathbf{n}}^\star \times \widehat{\hat{\mathbf{n}}) \times (}\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2)$    ▷$\mathbf{d}$ is on the intersection of two great
   circles containing the input arcs
2: if $(\mathbf{d} \times \hat{\mathbf{n}}_2) \cdot (\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2) \geq 0$ & $(\mathbf{d} \times \hat{\mathbf{n}}_1) \cdot (\hat{\mathbf{n}}_2 \times \hat{\mathbf{n}}_1) \geq 0$ then
3:     $\hat{\mathbf{n}}_{int} \leftarrow \mathbf{d}$    ▷$\mathbf{d}$ inside the arc
4: if $(-\mathbf{d} \times \hat{\mathbf{n}}_2) \cdot (\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2) \geq 0$ & $(-\mathbf{d} \times \hat{\mathbf{n}}_1) \cdot (\hat{\mathbf{n}}_2 \times \hat{\mathbf{n}}_1) \geq 0$ then
5:     $\hat{\mathbf{n}}_{int} \leftarrow -\mathbf{d}$    ▷$-\mathbf{d}$ inside the arc
6: return $\hat{\mathbf{n}}_{int}$
```

A.2 Building the Saparatrix

Pseudo-code for building the Morse complex and finding the separatrixes that form the boundaries of ascending manifolds:

Algorithm 3 BuildSeparatrix($\mathcal{H}, \mathbf{c}$)

Input: A convex shape $\mathcal{H}$ uniquely determined by a point set $\{x_1, x_2, \ldots, x_n\}$ in $\mathbb{R}^3$, A point $\mathbf{c}$ in $\mathbb{R}^3$ inside $\mathcal{H}$ determining the center of mass.

Output: A set of quadruples BN $= \{(\hat{\mathbf{n}}_1^i, \hat{\mathbf{n}}_2^i, f_1^i, f_2^i)\}_i$ where $\hat{\mathbf{n}}_1^i, \hat{\mathbf{n}}_2^i$ are unit vectors and f_1^i, f_2^i are two stable faces. The great arc connecting $\hat{\mathbf{n}}_1^i, \hat{\mathbf{n}}_2^i$ is on the boundary of the basins of attraction of stable faces f_1^i and f_2^i.

```
BN ← {}
$SE$ ← SaddleEdges($\mathcal{H}$)
for $e_0 \in SE$ do
    $(f_1, f_2)$ ← AdjFaces($e$)                    ▷ Two faces neighboring e
    $f_1^*$ ← DestinedFace($f_1$)                  ▷ Resting face starting from $f_1$
    $f_2^*$ ← DestinedFace($f_2$)
    if $f_1^* = f_2^*$ then                          ▷ Not dividing two different basins
        break
    $\hat{\mathbf{n}} \leftarrow \hat{\mathbf{n}}^\star_{e_0}$
    $\hat{\mathbf{n}}_{next} \leftarrow \mathbf{0}$
    for $p_i \in$ AdjVertices($e_0$) do            ▷ two vertices adjacent to e
        $p \leftarrow p_i$
        while True do
            if $p$ is maximum vertex then
                $\hat{\mathbf{n}}_{next} \leftarrow \hat{\mathbf{n}}^\star_p$
                BN ← BN ∪ $(\hat{\mathbf{n}}, \hat{\mathbf{n}}_{next}, f_1^*, f_2^*)$
                break
            for $e \in$ AdjEdges($p$) do
                $f, f'$ ← AdjFaces($e$)
                $\hat{\mathbf{n}}_f, \hat{\mathbf{n}}_{f'}$ ← Normal($f$), Normal($f'$)
                $\hat{\mathbf{n}}_{next}$ ← ArcsInt($\hat{\mathbf{n}}, \hat{\mathbf{n}}_{vertex}, \hat{\mathbf{n}}_f, \hat{\mathbf{n}}_{f'}$)
                if $\hat{\mathbf{n}}_{next} \neq 0$ then            ▷ Intersection happens
                    BN ← BN ∪ $(\hat{\mathbf{n}}, \hat{\mathbf{n}}_{next}, f_1^*, f_2^*)$
                    $\hat{\mathbf{n}} = \hat{\mathbf{n}}_{next}$
                    $p$ = OtherVertex($e, p$)
                    break
        end while
return BN
```

B Inverse Convex Hulls

Here we detail our method for fitting a nonconvex shape into a convex shape after the dice energy optimization. Given a convex surface $\mathcal{H}^*$, and a reference concave surface S^0 and a given center of mass $\mathbf{c}^0$. We aim find a surface S^* that is the minimizer for the following problem:

$$S^* = \arg\min_S \ \mathrm{d}(S, S^0) \tag{23}$$

$$\text{s.t.} \ \ \mathrm{ConvHull}(S) = \mathcal{H}^* \tag{24}$$

$$\mathbf{c}(S) = \mathbf{c}^0 \tag{25}$$

where $d(S, S^0)$ is a combination of surface energy often used for elastic deformation problems, and we use it to measure how much S and S^0 look alike.

We find S^* is two steps:

(1) **Hull Fill**: Find $\tilde{S}$ where ConvHull($\tilde{S}$) = $\mathcal{H}^*$, and $\tilde{S}$ *looks* like S^0.
(2) **CoM Correction**: Deform $\tilde{S}$ into S^* such that $\mathbf{c}(S^*) = \mathbf{c}^0$, while preserving its convex hull ConvHull(S^*) = $\mathcal{H}^*$.

B.1 Hull Fill

We start with a scaled S^0 that is fully contained in $\mathcal{H}^*$, and iteratively solve the following optimization problem to obtain a sequence of surfaces $\{S^t\}$ that converge to $\tilde{S}$. We deal with a soft version of constraint 24 using a combination of Surface Closest Point Energy and linear inequalities:

$$\tilde{S} = \arg\min_S \lambda_{CP} E_{CP}(S, \mathcal{H}^*) + \lambda_m E_m(S) + \lambda_b E_b(S) \tag{26}$$

$$\text{s.t. } S \subseteq \mathcal{H}^* \tag{27}$$

Here E_{CP} is a Surface Closest Point Energy where the closest point assignment between S and $\mathcal{H}^*$. For every vertex $\mathbf{p} \in \mathcal{H}^*$, denote its closest point on S by $CP(\mathbf{p})$, then E_{CP} is given by:

$$E_{CP}(S, \mathcal{H}^*) = \sum_{\mathbf{p} \in \mathcal{H}^*} \|\mathbf{p} - CP(\mathbf{p})\|^2 \tag{28}$$

Note that $CP(\mathbf{p})$ could lie on a face, edge, or vertex of S, so it can be written as a linear combination of vertex positions of S. The term $E_m(S)$ is a conformal Membrane energy ([Hormann and Greiner 2000]) that uses S^0 as reference, and similarly $E_b(S)$ is a surface Bending energy ([Grinspun et al. 2003]).

The feasible set in this problem is convex, since the constraints are a set of linear inequality constraints corresponding to half-spaces that bound $\mathcal{H}^*$ (a convex surface). The goal is to get $E_{CP}(S, \mathcal{H}^*)$ close to zero, where convex hull of $\tilde{S}$ will be $\mathcal{H}^*$ (satisfying constraint 24). We do this by increasing λ_{CP} iteratively. The other terms ensure that S remains visually close to S^0. We start with S^0 as the initial surface, and denote the deformed surface at every step by S^t. We use a quadratic approximation for the terms E_b and E_m at each step, using S^0 as the reference surface and S^t as the deformed surface. The closest point assignments for E_{CP} (in Equation 28) are also found using S^t as the current surface. This leads to solving a linearly constrained quadratic program to obtain S^{t+1}. For the quadratic approximation we use automatic differentiation (TinyAD library [Schmidt et al. 2022]) to obtain the gradient and projected Hessian of each energy term, and then we use a QP solver (Gurobi library [Gurobi Optimization, LLC 2024]) to solve the resulting optimization problem. Denoting $E_{\text{deform}} = \lambda_{CP} E_{CP}(S, \mathcal{H}^*; S^t) + \lambda_m E_m(S, S_0) + \lambda_b E_b(S, S_0)$, and vertex positions of S^t with $\mathbf{x}^t$:

$$\mathbf{x}^{t+1} = \arg\min_{\mathbf{x}} \ \frac{1}{2}\mathbf{x}^T(\tilde{H}_{\text{deform}})\mathbf{x} + g_{\text{deform}}^T \mathbf{x} \tag{29}$$

$$\text{s.t.} \ \ A_{\mathcal{H}^*}\mathbf{x} \leq b_{\mathcal{H}^*} \tag{30}$$

where $\tilde{H}_{\text{deform}}$ and g_{deform} are projected Hessian and gradient vector of E_{deform}.

B.2 CoM Correction

In this step, we treat with constraint 25 softly and find a solution to the following problem:

$$\tilde{S} = \arg\min_{S} \lambda_{\text{CoM}} E_{\text{CoM}}(S, \mathcal{H}^*) + \lambda_m E_m(S) + \lambda_b E_b(S) \tag{31}$$

$$\text{s.t. } \text{ConvHull}(S) = \mathcal{H}^* \tag{32}$$

where $E_{\text{CoM}}(S, \mathbf{c}^0)$ simply measures the distance between center of mass of S and $\mathbf{c}^0$:

$$E_{\text{CoM}}(S, \mathbf{c}) = \|\mathbf{c}(S) - \mathbf{c}^0\|^2 \tag{33}$$

However, dealing with the convex hull constraint (in 32) is not straightforward and cannot be relaxed like the previous step. So instead, we start with a surface that satisfies this constraint. Specifically we start with $\tilde{S}$ obtained from previous step ($\tilde{S}^0 := \tilde{S}$). Then we iteratively minimize the energy in 31, in a sequence of surfaces $\tilde{S}^t$, while always satisfying constraint 32; i.e. ConvexHull($\tilde{S}^t$) $= \mathcal{H}^*$, $\forall t$.

For this task, we use a first-order method and only use the gradients of terms in 31, and similar to previous step, we increase λ_{CoM} iteratively until E_{CoM} goes to zero. We have the gradients for E_b and E_m from before, and derive the gradient for E_{CoM}.

Denoting vertex positions of S by $\mathbf{x}$, we have:

$$\nabla_{x_i} E_{\text{CoM}} = 2 \frac{\partial \mathbf{c}}{\partial \mathbf{x}_i} \left(\mathbf{c}(\mathbf{x}) - \mathbf{c}^0 \right) \tag{34}$$

where $\frac{\partial \mathbf{c}}{\partial \mathbf{x}_i}$ is a 3-by-3 sub-matrix of the Jacobian of $\mathbf{c}(\mathbf{x})$ corresponding to vertex i.

Since we look for low frequency deformations in this step, we smooth out and diffuse these gradient vectors using Sobolev preconditioning.

To preserve the convex hull of $\tilde{S}^t$ at every step, we simply weigh the gradient vector at every vertex by its distance to $\mathcal{H}^*$; vertices on or close to $\mathcal{H}^*$ barely move, while vertices deep inside $\mathcal{H}^*$ can have a larger displacement.

Denoting $E = \lambda_{\text{CoM}} E_{\text{CoM}}(S, \mathcal{H}^*) + \lambda_m E_m(S) + \lambda_b E_b(S)$, we finally arrive at the following update rule:

$$\tilde{\mathbf{x}}^{t+1} = \tilde{\mathbf{x}}^t - \alpha D_{\mathcal{H}^*} (I + \gamma \mathrm{L})^{-p} \nabla E \tag{35}$$

where $\tilde{\mathbf{x}}^t$ is vertex positions of $\tilde{S}^t$, $D_{\mathcal{H}^*}$ is a diagonal matrix with i'th entry being the distance of vertex i from $\mathcal{H}^*$, and α is the step size found by line search.