Disjoint Convex Shell and its Applications in Mesh Unfolding

Yun-hyeong Kim[a], Zhonghua Xi[b], Jyh-Ming Lien[b]

[a]*Korea Institute of Science and Technology, Seoul, Republic of Korea*
[b]*George Mason University, Fairfax VA, USA*

Abstract

In this work, we study a geometric structure called *disjoint convex shell* or simply DC-shell. A DC-shell of a polyhedron is a set of pairwise interior disjoint convex objects that collectively approximate the given polyhedron. Preventing convex objects from overlapping enables faster and robust collision response and more realistic fracturing simulation. Without the disjointness constraint, a physical realization of the approximation becomes impossible. This paper investigates multiple approaches that construct DC-shells from shapes that are either composed of overlapping components or segmented into parts. We show theoretically that, even under this rather simplified setting, constructing DC-shell is difficult.

To demonstrate the power of DC-shell, we studied how DC-shell can be used in mesh unfolding, an important computational method in manufacturing 3D shape from the 2D material. Approximating a given polyhedron model by DC-shells provides two major benefits. First, they are much easier to unfold using the existing unfolding methods. Second, they can be folded easily by both human folder or self-folding machines. Consequently, DC-shell makes paper craft creation and design more accessible to younger children and provides chances to enrich their education experiences.

Keywords: Convex approximation, Decision boundary, Optimization, Folding, Unfolding, Convexification

1. Introduction

Approximating a 3D mesh by disjoint convex objects enables more applications than approximating it using overlapping convex objects. Examples include faster and more robust collision response, better local penetration depth estimation, faster volumetric meshing and more realistic fracturing simulation. Without this disjointness constraint, physical realization of the approximation is not even possible. We call this geometric structure, disjoint convex shell or simply **DC-shell**. The word *shell* is used to avoid confusion with convex hull.

There exist several ways to produce DC-shell, including solid convex segmentation [1]. To the best of our knowledge, our work is the first attempt that produces practical non-overlapping convex approximation whose number of disjoint convex components is significantly smaller than those created by existing methods. The most relevant work that we found is in computational geometry. There, researchers studied methods to cover disjoint convex sets with non-overlapping convex polygons [2]. Our problem, on the other hand, is to find non-overlapping convex shapes approximating non-convex overlapping sets.

In this work, we propose an optimization method that constructs DC-shells from the convex hulls enclosing the parts of a composite shape, i.e., shape that is composed of multiple parts or is segmented either manually or algorithmically from a single mesh. The convex hulls of these parts often overlap. Many meshes come in this form, in particular those available in public model-sharing sites, such as *Thingiverse* or Google 3D warehouse, or those created for video games and animations. Later in this paper, we will show that, even under this somehow simplified setting, the problem of converting overlapping convex objects to disjoint convex objects is difficult and computationally expensive. Moreover, straightforward heuristic methods often generate undesirable convex shells. Figure 1 illustrates some failed examples obtained from a method that trims the overlapping convex shapes using the cut planes that least-squares fit to the boundary intersections. This heuristic method (detailed in Section 3.1) in many cases generates results with significant volume loss.

The proposed method can produce consistent results with little volume loss. The key contribution of this method is an easy-to-implement and efficient approach that first obtains an initial guess of the cut planes via support vector machine (SVM) [3] of points sampled from the interior of the input convex hulls, and then minimizes the volume loss via the derivative of exact volume computation between a moving cutting plane and a pair of convex shapes. Because the convex hulls of the parts in a composite shape often contain long and skinny triangles and, because the cutting process often introduces new sliver triangles near the cuts, it is critical that these slivers are removed. To remove slivers of DC-shells while guarantees the convexity and disjointness, we contribute a new remeshing method in Section 4. As illustrated in Fig. 1 and later in Section 6, our results show that the DC-shells created by the

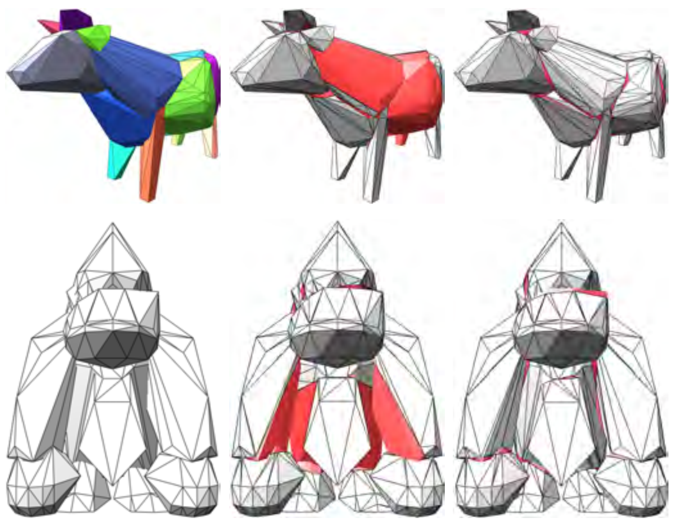


Figure 1: **left**: Overlapping convex shapes depicting a cow and Donkey Kong. **middle**: Cutting overlapping convex objects via boundary intersections (detailed in Sec. 3.1) results in significant volume loss (shown in red), which is defined as the volume inside the input shape but outside the disjoint convex shell. **right**: Disjoint convex shells created by our optimization method described in Sec. 3.2 has significantly smaller volume loss.

proposed method have significantly lower volume loss than those created by least-squares-fit and SVM-only methods.

Throughout the paper, we will use mesh unfolding to demonstrate the power of DC-shell. Mesh unfolding is an important computational problem in manufacturing a 3D shape from a 2D material [4, 5, 6]. Designing a foldable 3D shape usually involves two main foldability analysis steps: instantaneous unfolding and continuous folding. Instantaneous unfolding transforms a polyhedron to a 2D representation instantaneously. The continuous folding problem aims to find a foldable path that transforms the net to its folded shape continuously without self-intersection. Both steps are significantly easier for convex shapes than non-convex shapes [7, 8].

Through its application in mesh unfolding, we show that DC-shells can be rigidly unfolded into high-quality nets with minimal cut length and coverage. We verify the foldability of these nets created from DC-shells algorithmically and through a user study with 102 school-age children. We show that children can work together and create rather complex 3D paper crafts around an hour in a hands-on story-telling class. Because these 3D paper crafts are made of blocks of convex objects, students can freely reconfigure their poses and even enlarge certain parts before assembly. Fig. 2 shows an example of a composite shape (Yoshi) obtained from thingiverse.com that contains overlapping parts and its DC-shells and fabricated paper model.

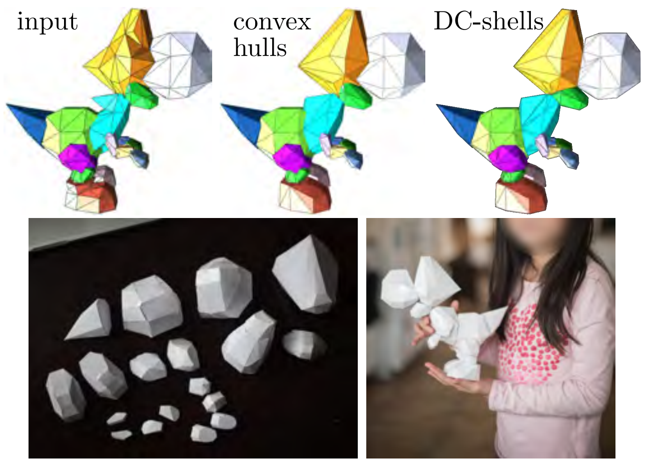


Figure 2: The top three figures, from left to right, show the composite shape of Yoshi model obtained from thingiverse.com, its convex hulls and its DC-shells. The bottom two photos show the folded disjoint convex shells and assembled Yoshi model.

2. Related Works

2.1. Convex Decomposition and Approximation

Convexity provides significant algorithmic benefits in many problems and motivates researchers to approximate shapes with convex objects. Covering a shape with non-overlapping convex shapes has been extensively studied. For example, exact convex decomposition [1, 9, 10] segments mesh into disjoint convex components. Another example is tetrahedral mesh generation. Both approaches can produce a large number of convex components. Exact convex decomposition is known to produce $O(r^2)$ convex components for a polyhedron with r reflex edges.

Approximations that allow overlapping convex shapes are more prevailing. For instance, approximating a shape with a set of primitive convex shapes, such as spheres, boxes, ellipsoids, and capsules, usually, forms a hierarchy of overlapping volumes. Another example is approximate convex decomposition (ACD) [11, 12, 13, 14] that approximates the input mesh by a set of nearly convex shapes. Applications of ACD usually work solely with the convex hulls of the segmented parts and ignore the input mesh. These convex hulls usually overlap.

2.2. Polyhedra Unfolding

In the problem of rigidly unfolding the polyhedron to a 2D structure called the *net*, convexity admits simple edge unfolding of every convex polyhedron [8]. Finding a valid net of a given polyhedron is known to be nontrivial because a polyhedron with $||F||$ faces can have approximately $O(2^{\sqrt{||F||}})$ different unfoldings and most of them contain overlaps especially for non-convex polyhedra. However, if input shapes are convex polyhedra, heuristic methods

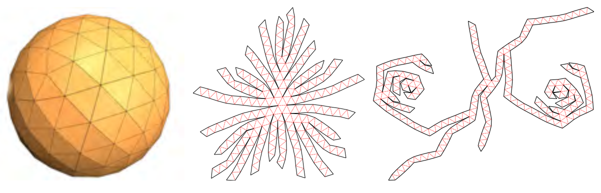

(a) Mesh (b) Net1 (c) Net2

Figure 3: A convex mesh its nets found by two heuristic methods: Steepest Edge for net1 and Flat Tree for net2 [7]. Net1 has a straight-line folding path in the configuration space.

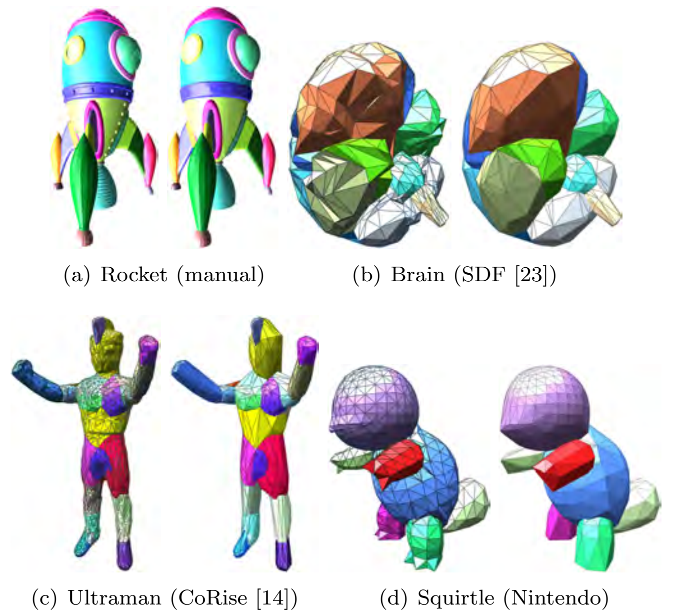
work well in practice. Most, if not all, nets of convex polyhedra with $||F||$ facets can be obtained in $O(||F||log||F||)$ time. Mitani and Suzuki [4] decompose the mesh into a few patches and approximate each patch with a strip, a generalized cylinder or a developable surface. Schlickenrieder [7] proposed heuristics methods for unfolding a polyhedron to a net. Straub and Prautzsch [15], Takahashi et al. [16] extend [7] to unfold non-convex polyhedra. Recently, Xi et al. [17] proposed an approach that segments a model into a smell set of semantic and easily unfoldable parts by learning from failed unfoldings. It is worth noting that (rigid) mesh unfolding faces computational challenges and issues different from a line of mesh flattening works that focuses on approximating surface patches with developable surfaces [18, 19]. All aforementioned works generate non-overlapping unfoldings, however, it is not guaranteed that there exists a continuous folding motion that transforms the net back to its original shape.

In order to make a physical copy of the foldable shape that can be continuously folded to its target shape from the net, we need to find a feasible folding path that can bring the net to its target shape without self-intersection. Again, if the input polyhedron is convex, the start state, i.e., the flat net and the goal state often can be linearly connected in the configuration space. This property significantly reduces the path planning time to find a continuous folding motion. Fig. 3 shows a convex polyhedron and its nets, one of whose folding path is a straight line in the configuration space. However, when dealing with non-convex shapes, previous works [20, 16, 21, 22] show that each step remains extremely challenging.

3. Building Disjoint Convex Shell (DC-shell)

Let us consider a composite shape P composed of components $\{P_i\}$, which may either overlap or overlap at boundaries with disjoint interiors. Fig. 4 shows examples of composite shapes obtained from various sources. Given a pair of adjacent components P_i and P_j, the intersection of their convex hulls $C_i \cap C_j$ is usually not empty.

A first important observation is that the separator of two overlapping convex shapes C_i and C_j must be linear. If the separator is curved, one of the separated shapes must be non-convex. Given that the separator must be a plane, it is desirable to constrain the cutting plane $\mathcal{H}$ so that the disjoint shapes are separated without a gap in between (i.e. with overlapping boundary on $\mathcal{H}$). Fig. 5 illustrates this idea. To ease our future discussion, we will use the term 'trim' to describe the operation of removing a small volume of an object by interesting the object and the closed half space $\mathcal{H}^+$ or $\mathcal{H}^-$ bounded by $\mathcal{H}$. We will also use C'_* to denote the trimmed shape, i.e., $C'_i = C_i \cap \mathcal{H}^+$ and $C'_j = C_j \cap \mathcal{H}^-$.


(a) Rocket (manual) (b) Brain (SDF [23])

(c) Ultraman (CoRise [14]) (d) Squirtle (Nintendo)

Figure 4: Examples of composite shape used in this paper including models created by manual segmentation (a), part-aware decomposition (b)&(c), and models composed of multiple overlapping parts (d). In all these examples, the convex hulls of their parts overlap.

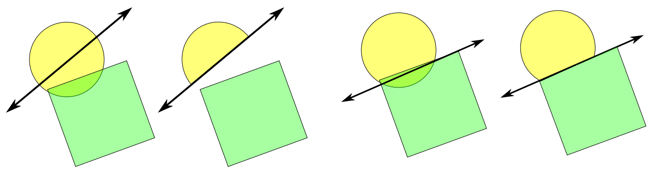


Figure 5: **left**: The cut results in a gap after trimming. **right**: Overlapping boundary with disjoint interior is more desirable.

In the rest of this section, we will start with a heuristic that finds the cutting plane via least squares fitting and discuss its limitations. To address these limitations, we then formulate an optimization problem and show that solving this optimization problem efficiently is challenging even in the discrete domain.

3.1. A Heuristic using Least Squares Fitting

Naturally, our first attempt computes the intersection of convex hull boundaries ∂C_i and ∂C_j and the cut plane $\mathcal{H}$

least-squares fits the intersection. Because the cut plane $\mathcal{H}$ fits the surface intersection, it is guaranteed that the trimmed shapes C_i' and C_j' is interior disjoint but exterior connected without any gap. In many cases, this simple method creates decent results as shown in Fig. 6.

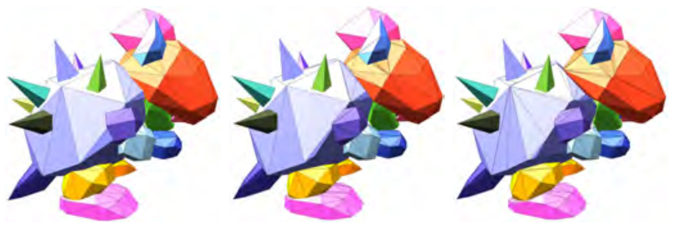


Figure 6: **left**: overlapping convex hulls of a Bowser model. **middle**: DC-shells created using least-squares fitting heuristic. **right**: Results of proposed method.

Although this least-squares fitting approach is extremely fast and easy to implement, its quality is unbounded. In particular, the volume trimmed by the cut plane is not considered as shown in the examples in Fig. 1. In the cases that ∂C_i and ∂C_j intersect at coplanar or nearly coplanar facets, have multiple connected-component intersection due to extrusion, or have highly non-planar intersection, least-squares fitting is too local and greedy and is clearly not a good approach. Fig. 7 shows two examples of such complications.

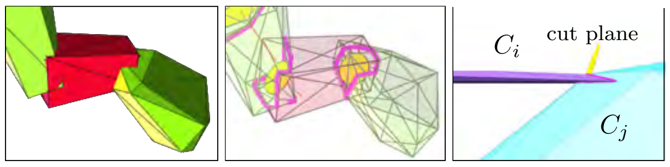


Figure 7: Complex near-coplanar overlap and extrusion between two convex objects. Cut planes are shown as yellow discs.

3.2. Disjoint Convex Shells

To provide higher quality disjoint convex shells, it is desirable to minimize the amount of volume removed by plane $\mathcal{H}$ in $\mathbb{R}^3$, therefore $\mathcal{H}$ should be derived from the following optimization function:

$$\arg\min_{\mathcal{H}} \left(\mathrm{vol}(P_i \cap \mathcal{H}^+) + \mathrm{vol}(P_j \cap \mathcal{H}^-)\right) \ , \qquad (1)$$

where $\mathcal{H}^+$ and $\mathcal{H}^-$ are the half-spaces on the positive and negative sides of $\mathcal{H}$. Because P_i and P_j not only are non-convex but also often contain surface holes and are non-manifold, the optimization problem in Eq. 1 becomes ill-defined for many composite models. Instead, we will focus on the following problem:

$$\arg\min_{\mathcal{H}} \left(\mathrm{vol}(C_i \cap \mathcal{H}^+) + \mathrm{vol}(C_j \cap \mathcal{H}^-)\right) \ . \qquad (2)$$

In addition, to ensure that no gap is introduced by $\mathcal{H}$, $\mathcal{H}$ must pass through a point in $C_i \cap C_j$. Let $o_{\mathcal{H}}$ be a 3D point on $\mathcal{H}$ and $n_{\mathcal{H}}$ be the normal direction of $\mathcal{H}^+$. The constrains ensure that $o_{\mathcal{H}} \in C_i \cap C_j$ and the objective function determines the exact position $o_{\mathcal{H}}$ and orientation $n_{\mathcal{H}}$ so that the trimmed volume is minimized.

The key to solving the aforementioned optimization problem is a formulation expressing the volume of the intersection between a known convex object and unknown 3D plane. In the literature, it is known that the function of the volume of a convex object intersected by a moving half-space is highly non-linear even in $\mathbb{R}^3$. In fact, computing the exact volume of a convex shape intersected by a translating plane in arbitrary dimension is known to be #P-space hard [24, 25, 26]. Several approximation methods were introduced [25, 27]. Almost all of these methods adopted the idea of uniform sampling of points inside the convex object [28]. The idea is that, if points are sampled uniformly inside the convex shape, the number of points approximates the volume. Naturally, we will start off our discussion by assuming that the volumes are approximated by samples in C_i and C_j.

3.2.1. Approximating the Volume with Samples

In this section, we will present two methods that use uniform samples to determine the separating plane. It is important to note that both methods can also handle points that are created from the segmented components P_i even though our discussion assumes that points are sampled from C_i.

Let S_i and S_j be two point sets in $\mathbb{R}^3$ whose convex hulls overlap. We assume that S_i and S_j are uniformly distributed in a subspace of $\mathbb{R}^3$, such as the space enclosed by their convex hulls. The objective is to find a plane $\mathcal{H}$ that minimizes the number of points from the same set to be separated by $\mathcal{H}$.

$$\arg\min_{\mathcal{H}} \left(||S_i \cap \mathcal{H}^+|| + ||S_j \cap \mathcal{H}^-||\right) \ , \qquad (3)$$

where $||X||$ is the cardinality of a finite set X.

We first show that finding all optimal solutions of $\mathcal{H}$ takes $O(||S_i + S_j||^{3/2})$ by reduction from the classic ham sandwich problem.

Arrangement of Dual Points

Duality has been used in problems such as linear programming. In this method, we will use an idea similar to that solves the *ham sandwich problem* [29, 30, 31]. That is, we will find the plane $\mathcal{H}$ in the dual space in which a 3D point $P^* = (a, b, c)$ represents a 3D plane P in primal space with normal direction $(a, b, \sqrt{1 - a^2 - b^2})$ and offset c from the origin. Similarly, a point $Q = (a, b, c)$ in the primal space becomes a 3D plane Q^* in the dual space. The problem of finding $\mathcal{H}$ is then equivalent to finding a point $\mathcal{H}^*$ such that the sum of the number of planes S_i^* below $\mathcal{H}^*$ and the number of planes S_j^* above $\mathcal{H}^*$ is minimized. Fig. 8

illustrates this idea in 2D. Next we show that finding all optimal cuts takes $O(||S_i + S_j||^{3/2})$ time by reducing the problem of 3D ham-sandwich cuts.

Theorem 3.1. *Given two finite point sets S_i and S_j whose convex hulls overlap, finding all optimal cuts of Eq. 3 takes $O(||S_i + S_j||^{3/2})$ time.*

Proof. The proof essentially follows the same construction from the work by Lo et al. [32] for finding the Ham-Sandwich cuts that takes $O(n^{3/2})$ time for n 3D points. To solve this dual version of our optimization problem, one will have to compute two arrangements $\mathcal{A}(S_i^*)$ and $\mathcal{A}(S_j^*)$ of the planes S_i^* and S_j^*, respectively. In each arrangement, every facet will be annotated with the number of planes below the facet, called *level* [29]. It is easy to show that every point on the same facet has the same level. Furthermore, in each arrangement, we can collect all facets of the same level and form a 2-manifold patch. There can be $O(||S_i|| + ||S_j||)$ patches. The computation of optimal cuts becomes the search of a pair of intersecting patches $(p_i \in \mathcal{A}(S_i^*), p_j \in \mathcal{A}(S_j^*))$ such that the level of p_i minus the level of p_j is minimized. In the ham-sandwich problem, this pair of patches is known to have levels $||S_i||/2$ and $||S_j||/2$ but in our case the levels with minimal difference are unknown to us. Fortunately, it is easy to see that these levels can be obtained by "jumping" from one patch to an intersecting patch as long as the level difference can be further reduced. The jumping operation requires $O(1)$ time computation using the computed arrangements and there can be at most $O(||S_i|| + ||S_j||)$ jumps.

Finally, the solution is reported as an edge form by the pair of intersecting patches with minimal level difference. This concludes the reduction of *3D ham-sandwich problem* to the problem of finding optimal cuts. □

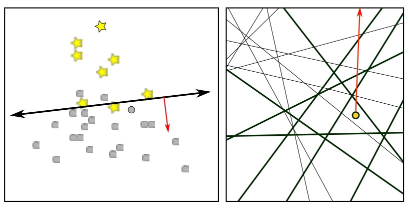


Figure 8: (a) The problem of finding a cutting line that minimizes the sum of the number of stars below the line and the number of circle above the line. (b) The problem of (a) is equivalent to finding a point so that the sum of the number of thick lines (dual of the circles) vertically below and the number of thin lines (dual of the stars) vertically above the point is minimized.

Unfortunately, a practical implementation of [32] requires an efficient algorithm for level construction that is nontrivial to implement, and a naïve implementation, which finds all possible solutions of $\mathcal{H}$, takes $O(||S_i + S_j||^3)$ is impractical since we only need one of such $\mathcal{H}$. In the next section, we investigate a more efficient approach using Support Vector Machine.

Using Support Vector Machine

Support Vector Machine (SVM) [3] is a widely used classifier due to its solid mathematic background, its fast computation speed and high classification accuracy. SVM finds a linear hyperplane to separate positive and negative data in feature space. The hyperplane is parameterized by $\mathbf{c}$ and d, where $\mathbf{c}$ is the normal vector and d is the offset of the hyperplane. The distance from the closest point to the hyperplane is called *margin*. SVM minimizes $||\mathbf{c}||^2 + \mu \sum_i \zeta_i$ where μ is user parameter and ζ_i is classification error, subject to the constraint:

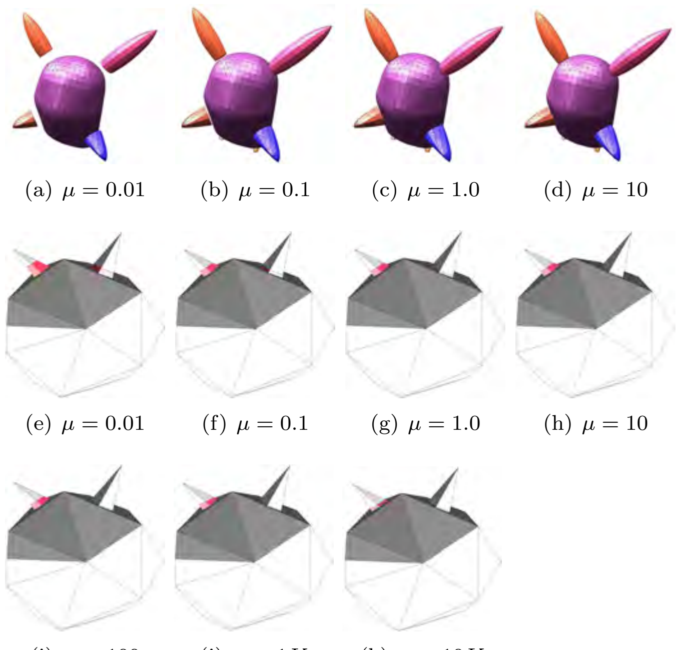
$$y_i(\mathbf{c}^T \mathbf{x}_i + d) >= 1 - \zeta_i \ ,$$

where $y_i \in \{-1, 1\}$ indicates the label of $\mathbf{x}_i$. This is known to be a quadratic programming problem and can be solved efficiently in most cases.

Given point sets S_i and S_j, finding the cut plane using SVM to minimize the Eq. 3 is straightforward as we do not consider nonlinear kernel functions and the dimensionality of our problem is low comparing to the dimensionality of typical classification problems solved by SVM. However, it is worth noting that, in our problem of separating two convex shapes, the penalty parameter μ plays a critical term as in majority of cases, the point sets S_i and S_j are not linearly separable and therefore the main task is to minimize the loss function $\sum_i \zeta_i$ that is proportional to the distance of misclassified points to the margin.

This becomes problematic because the penalty parameter μ might need to be different for different models and even between different parts within the same model. For models whose convex hulls are linearly separable, small μ is sufficient. For models with significant overlap between convex models, in particular when two overlapping convex shapes have significant difference in size (i.e., one convex shape is much larger than the other one), μ needs to be very large to ensure correct separation. Fig. 9 shows examples of different μ creating different results. Sometimes, the gaps cannot be eliminated even when μ is as large as ten thousand. In addition, as μ increases, the optimal solution becomes harder to find and SVM takes much longer time to coverage and often fails to converge. Solutions to handle such sampling discrepancy for classification problems usually involve assigning weights to samples or increasing samples. Both approaches are not viable to us because the size of samples represents volume.

It might also be tempting to separate only the points in $C_i \cap C_j$, instead of all points sampled, to avoid the aforementioned pitfalls. However, again, this local and greed approach will create undesirable results in which the cutoff volume becomes unbounded.


(a) $\mu = 0.01$ (b) $\mu = 0.1$ (c) $\mu = 1.0$ (d) $\mu = 10$

(e) $\mu = 0.01$ (f) $\mu = 0.1$ (g) $\mu = 1.0$ (h) $\mu = 10$

(i) $\mu = 100$ (j) $\mu = 1K$ (k) $\mu = 10K$

Figure 9: **top**: Changing penalty parameter μ from 0.01 to 10 affects SVM cuts. **bottom**: Gaps shown in red below the spikes narrow at different rates as μ increases from 0.1 to 10,000 because the spike on the left is buried deeper than the one on the right.

3.2.2. Exact Volume Computation

The issue of the penalty parameter μ can be addressed if we can be less dependent on the sampling discrepancy. The volume of the intersection between a convex shape $C = \{x \in \mathbb{R}^n : b_i - \mathbf{a_i}^T x \geq 0\}$ and a translating plane $\mathcal{H}(x) = \mathbf{c}^T x + d$ is known to be the sum of a linear function between the vertices of C and $\mathcal{H}(x)$ [26]:

$$V_C = \sum_{v \in C} \frac{\max(0, H(v)^n)}{n! \delta_v \gamma_1 \cdots \gamma_n} \ , \tag{4}$$

where γ_i satisfies $\mathbf{c} = \sum_i \mathbf{a_i} \gamma_i$ and δ_v is $|det([\mathbf{a_1} \cdots \mathbf{a_n}])|$, and $\mathbf{a_i}$ is the normal of a hyperspace of C containing v.

Computing Eq. 4 in general space is #P-space hard mainly due to the curse of dimensionality and the vertices v are unknown. Fortunately, our problem is in 3 space and all the vertices of C are known. In 3D, the volume can be simplified to

$$V_C = \sum_{v \in C} \frac{\max(0, \mathcal{H}(v)^3)}{6 \delta_v \gamma_1 \gamma_2 \gamma_3} = \sum_{v \in (C \cap \mathcal{H}^+)} \frac{(d + \mathbf{c}^T v)^3}{\Delta_v} \ . \tag{5}$$

Now, if we let $\mathcal{H}$ translate, then V_C is a function of d. Given two overlapping convex shapes C_i and C_j, our goal is to find the offset d such that $D = V_{C_j} - V_{C_i}$ is minimized[1]. To determine d that minimizes D between a pair of consecutive vertices of C_i and C_j, we simply find the roots of the following quadratic equation $\frac{\partial D}{\partial d} = 0$, i.e.,

$$\sum_{v \in (C_j \cap \mathcal{H}^+)} \frac{3(d + \mathbf{c}^T v)^2}{\Delta_v} - \sum_{u \in (C_i \cap \mathcal{H}^+)} \frac{3(d + \mathbf{c}^T u)^2}{\Delta_u} = 0 \ . \tag{6}$$

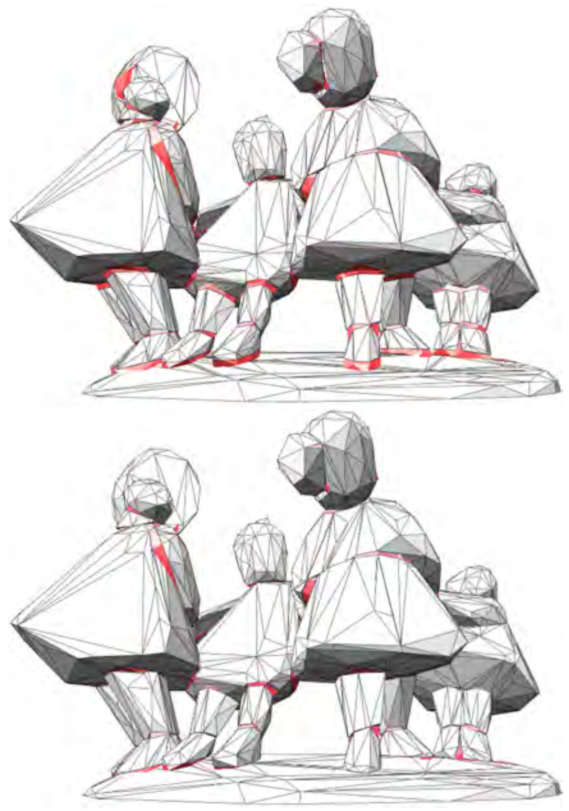


Figure 10: **top**: DC-shells created using SVM. The red regions are volumes trimmed from the original input to create disjoint convex objects. The volume loss 1.62% of the volume of the union of the input convex hulls. **bottom**: DC-shells created using exact volume computation. The volume loss is at 0.86%.

Fig. 10 compares the DC-shells created using SVM with and without minimizing D. The volume loss is decreased by half using exact volume computation.

Note that, it may be tempting to discard SVM entirely and find the cut plane $\mathcal{H}$ by minimizing Eq. 6 with the full degrees of freedom. This means, $\mathbf{c}$ is also unknown, and we will need to solve three first-order partial derivatives $\frac{\partial D}{\partial \mathbf{c}_x} = 0$, $\frac{\partial D}{\partial \mathbf{c}_y} = 0$ and $\frac{\partial D}{\partial d} = 0$. However, solving $\frac{\partial D}{\partial \mathbf{c}} = 0$ is very challenging as $\mathbf{c}$ appears in both nominator and denominator of D, thus requiring us to find roots of three degree-4 polynomials with multiple degree-6 polynomial constraints. This will require further investigation in our future research.

[1]We actually wanted to minimize $V_{C_j} + \text{vol}(C_i) - V_{C_i}$, but $\text{vol}(C_i)$ is a constant so it is equivalent to minimizing $V_{C_j} - V_{C_i}$.

3.2.3. Handling Interference Between Cuts

Our discussion so far has centered around the case of pair-wise overlapping. When more than two convex objects overlap sharing a common region, these cut planes can have mutual interference. To describe this phenomenon more clearly, we first observe that:

Observation 3.2. *A cut plane $\mathcal{H}_{ij}$ between C_i and C_j can only affect the other overlapping pairs C_i and C_k if $\mathcal{H}_{ij}^-$ encloses $C_i \cap C_k$. When this happens, we say that $\mathcal{H}_{ij}$ and $\mathcal{H}_{ik}$ interfere mutually.*

Notice that, this observation implies the scenarios of three or more overlapping objects sharing a common region. Moreover, this observation also implies that interference can only be realized through a shared object, i.e., C_i in our example.

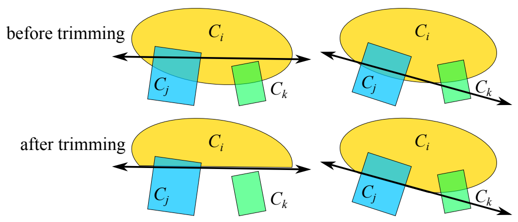


Figure 11: The cut between C_i and C_j on the left interferes with C_i and C_j. The cut on the right does not.

It is important to note that under these interferences, our objective remains the same: minimizing volume loss. Therefore, the cut plane $\mathcal{H}_{ij}$ interfering $\mathcal{H}_{ik}$ should still trim C_i and C_j as little as possible and, at the same time, it should also minimize the amount of interference defined as the volume of $\mathcal{H}_{ij}^- \cap (C_i \cap C_k)$. This additional requirement can be added to Eq. 6 to find a cut plane $\mathcal{H}$ between C_i and C_j that minimizes it inference to all affected C_k:

$$\frac{\partial D}{\partial d} + \lambda \frac{\partial(\sum_k \text{vol}(C_k \cap C_{i\vee j}, \mathcal{H}^-))}{\partial d} = 0 \ , \qquad (7)$$

where λ is a user defined weight that penalizes interference between $\mathcal{H}$ and the pair C_k and C_i or C_k and C_j. In all experiments reported in this paper, we chose a small $\lambda = 0.01$. Eq. 7 can be solved using the same strategy discussed in the previous section as $C_k \cap C_{i\vee j}$ is also convex. Fig. 12 shows before and after inference is penalized.

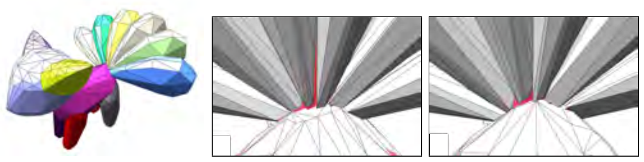


Figure 12: Cuts between the tails and the torso of a Vulpix model interference and make some tails completed separated from the torso.

4. Convex Shell Simplification and Regularization

Convex shapes considered in this paper are created by taking the convex hulls of composite shape. Consequently, their facets tend to be skinny and often densely packed near certain spots. Before these convex objects are trimmed, it is often desirable to simplify and remesh them to decrease the complexity and increase their regularity. The trimming process discussed earlier may often produce many small and skinny sliver triangles. It is critical that these slivers can be removed. Remeshing also makes unfolding and folding easier, thus ease the fabrication process.

There have been extensive study of convex and non-convex polyhedra simplification and remeshing, e.g., see reviews in [33]. In our case, we require that the simplified and remeshed shape maintains its convexity and interior disjointness. We further require that the simplified and remeshed convex shape must enclose the original convex shapes to avoid introducing new gaps. Finally, under these critical constrains, we would like to minimize the gained volume after remeshing.

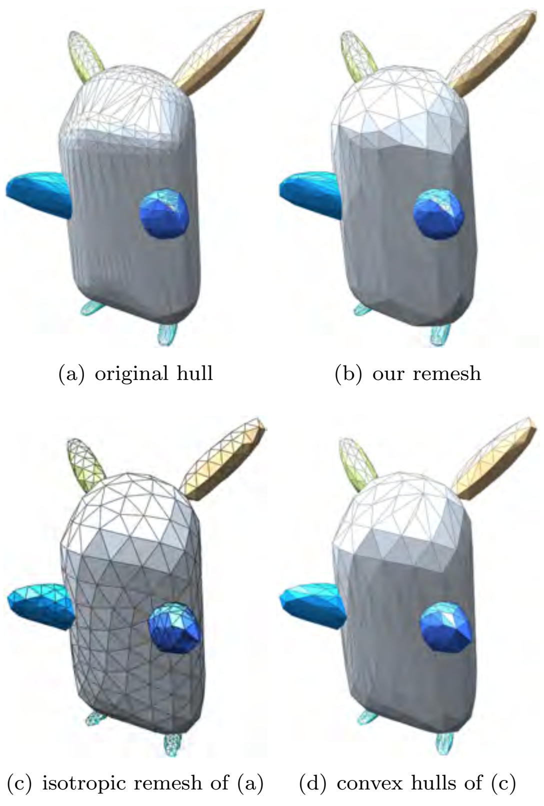
Simplification. The progressive hull introduced by Sander et al. [34] can be easily adopted for our requirements. The approach is based on a sequence of edge contractions, in which an edge is contracted to a vertex. The vertex is placed according to a constrained optimization problem. The constraints are that the vertex must be placed inside the cut planes and outside the plane of every face incident to the edge. Equivalently, every tetrahedron formed by connecting the new vertex to a face incident to the edge must add volume to the shape. The progressive hull objective function minimizes the total added volume. The next edge to contract is the one whose contraction adds the least volume. Edges with no solution satisfying the constraints are skipped. The constraints and objective function are linear, since the signed volume of a tetrahedron with three fixed vertices is linear:

$$\frac{A}{3} n \cdot (v - v_0) \ ,$$

where v is the free vertex, v_0 is any one of the fixed vertices, and n and A are the outward unit normal and area of the triangle formed by the three fixed vertices. As a result, the constrained optimization for each edge contraction is a small linear programming problem that can be solved efficiently.

Regularization. Although many remeshing methods exist including isotropic remeshing [35] and as-equilateral-as-possible remeshing [36]. To the best of our knowledge, no remeshing method can ensure convexity. In fact, our study also finds that, while some vertices are outside the original convex shape, most vertices are inside. This introduces gaps between parts after remeshing. An example of such problems is shown in Fig. 13(c). To regain convexity, Fig. 13(d) shows the convex hulls of Fig. 13(c) but regularity is destroyed.

We propose a new method to address this problem using Laplacian smoothing [37] that moves every vertex to the center of its neighbors. We formulate the remeshing problem as a sequence of quadratic programming problems. Each quadratic programming problem minimizes the distance between a vertex v and the center of its neighbors with linear constrains, similar to the aforementioned constraints for progress hulls construction, ensuring that the mesh remains convex and disjoint after moving v. A result obtained from the proposed method can be found in Fig. 13(b). An extensive study of this new method can be found in Section 6.3.


(a) original hull (b) our remesh

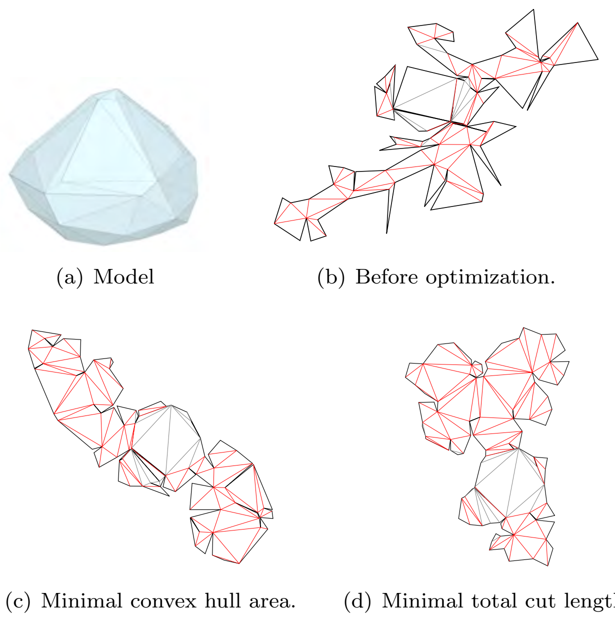
(c) isotropic remesh of (a) (d) convex hulls of (c)

Figure 13: Remeshing results using the proposed method (b) and using isotropic remesh (c) and then reenforce convexity using convex hulls (d). Notice the skinny triangles on Pikachu's belly in (d).

5. Unfolding and Folding DC-shells

While DC-shell can be used in many other domains (such as fracturing shown in the supplementary video), our work on creating DC-shell is motivated by mesh folding and unfolding. Although efficient algorithms exist for unfolding convex shapes [7], they did not consider the quality of nets (e.g. ease of fabrication and foldability). In this work, we extend a genetic-based algorithm [17] to optimize the nets of DC-shells using the fitness function $f = \sum_i w_i x_i$, where x_i is the feature value and w_i is its corresponding weight.

Convex hull area and total cut length are two features considered here. Intuitively, a smaller convex hull area means the net has a tighter representation which reduces the fabrication time; it also indicates the net has fewer long branches that are harder to fold. Since all cut edges need to be glue together when folded, shorter total cut length also makes folding more easier and makes the foldable object mechanically more stable. In Fig. 14, we show an example of net optimization, from which we can see that, initially, the net (Fig. 14(b)) contains a lot of skinny triangles like spines. After optimization, we get more desirable nets (Fig. 14(c) and Fig. 14(d)) whose convex hull area or total cut length reduced by more than 40%.


(a) Model (b) Before optimization.

(c) Minimal convex hull area. (d) Minimal total cut length.

Figure 14: Optimized nets of a Bulbasaur's seed model. Convex hull area is reduced by 41.42% and total cut length is reduced by 50.41%.

6. Experimental Results

We implemented the proposed DC-shell methods in C++, which will be released to the public domain after this work is published. All data reported in this paper were collected on a 2015 MacBook Pro with a 2.5 GHz Intel Core i7 CPU and 16 GB Memory.

6.1. Running Time

Fig. 15 shows the running times of our implementation of all three methods discussed in Section 3, namely LSF (least-squares fit) heuristic method, SVM and exact volume optimization methods. In our experiments, we simplify and regularize the meshes 10 iterations with maximum volume increase ratio less than 5% of the original volume using the method discussed in Section 4. According to Fig. 15, LSF is fastest (including simplification and regularization time) but the proposed optimization methods using either SVM or exact volume computation are only 2 to 10 seconds slower, except for the "Hover bike" model. SVM and exact methods are much slower in this example because Hover bike contains 49 components and many of them are small and deeply embedded in some larger convex parts. Images of Hover bike and Mother models can be found in Supplementary Material.

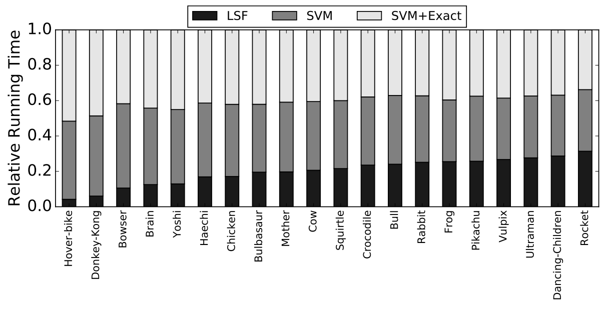


Figure 15: Ratios in running time. The value 1.0 indicates the total running time obtained from the sum of all three methods. The running time measured in seconds can be found in the supplementary material.

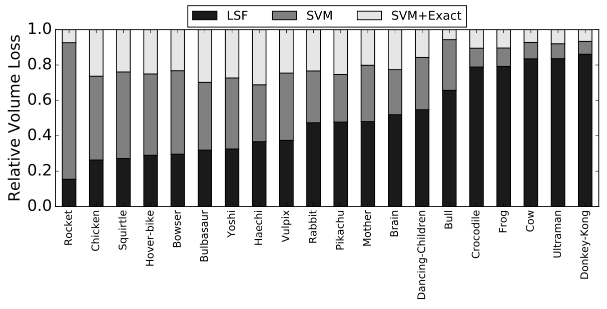


Figure 16: Ratios of volume loss. The value 1.0 indicates the total volume loss obtained from the sum of the loss from all three methods. Quality of DC-shells produced using least squares fit (LSF), SVM and exact methods is measured by volume loss defined in Eq. 8. The absolute volume loss can be found in the supplementary material.

6.2. Quality Comparison

The quality of DC-shell is measured by volume loss from trimming the input convex shapes. That is to say, a DC-shell has higher quality if it removes less material from input to provide disjointness. More specifically, volume loss is defined as

$$\frac{\text{vol}_{orig} - \text{vol}_{shell}}{\text{vol}_{orig}}, \tag{8}$$

where vol_{orig} is the volume of the union of the original convex shapes and vol_{shell} is the sum of the volume of all disjoint convex shells.

Case study. We start by measuring the volume loss of the proposed methods with models obtained from various sources. Fig. 16 reports the volume loss in percentage of all DC-shells generated from least-squares fit, SVM, and the exact methods. As the original convex objects are not disjoint, to obtain the original volume, we need to perform boolean union. CGAL is used to perform this operation. In the case that CGAL failed to create a 2-manifold mesh, a volumetric method is used to provide a tight upper bound of the union. The least-squares fit (LSF) method performs quite well on some models such as Squirtle but extremely poor on the others such as Bull model with over 24% volume loss. The SVM method has smaller volume loss than LSF in most cases but performed worse than LSF on Bowser, Spaceship, and Squirtle models and still has over 10% volume loss on the Bull and Hover bike models. Finally, the exact method consistently provides the highest quality DC-shell in all models experimented. Even fro the cases that LSF and SVM methods perform well, the exact method can always reduce volume loss even further.

Controlled study. In this experiment, two intersecting convex shapes have various degree of overlapping. Fig. 17 shows the DC-shells generated. LSF method consistently generates low quality results with large volume loss (11.43% on average) due to none-planar local intersection. The optimization methods using SVM and exact volume computation, on the overhand have only 2.1% and 0.9% volume loss, respectively. In general, SVM and exact methods produce similar results, except the last example where two bars overlapping vertically. The exact method produces more natural DC-shells with smaller volume loss.

6.3. Results from Convex Remeshing

The quality of regularization is measured by the weighted average fatness of the mesh facets. Fatness of a given triangle f is defined as $\cos\angle a + \cos\angle b + \cos\angle c - 1$, where $\angle a$, $\angle b$, $\angle c$ are internal angles of f. The optimal fatness is 0.5. Then weighted average fatness of a given polyhedron P is $\frac{\sum_{f\in P} A_f \text{fatness}(f)}{\sum_{f\in P} A_f}$, where A_f is the area of f.

Fig. 18 shows weighted average fatness from 1 to 100 iterations of remeshing at four different levels of maximum volume increase percentages: 1%, 10%, 100%, and 1000%. Larger maximum volume increase percentages allows more room for optimization, thus should provide better regularization. From the plots, we can see that, with 1% maximum volume increase, the facet fatness increases gradually. At 10% to 1000% maximum volume increase, the fatness increases drastically and barely change after it reaches a certain fatness. The weighted average fatness increase by approximately 25% at least after regularization. The plots in Fig. 18 also shows the fatness of the convex hulls of remeshed convex objects using isotropic remeshing implemented in CGAL, i.e., Fig. 13(c) and (d). The fatness of

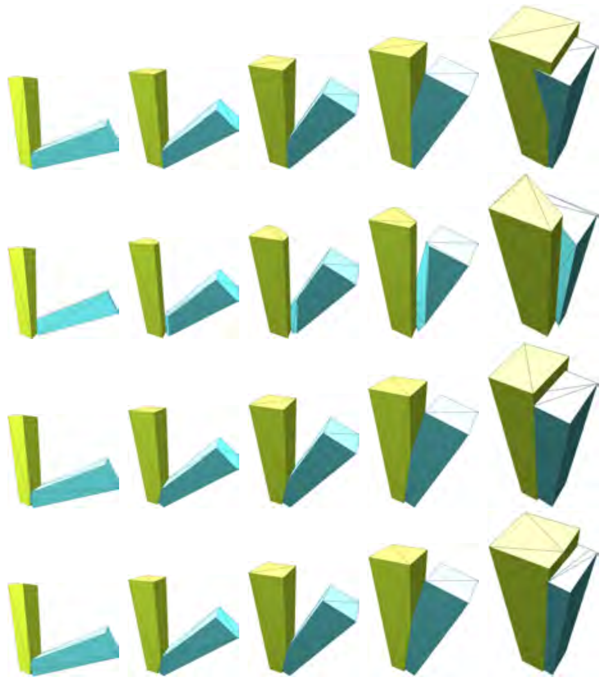
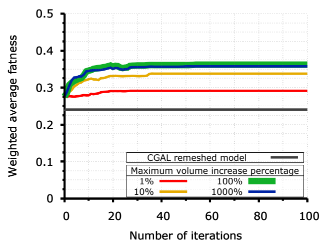


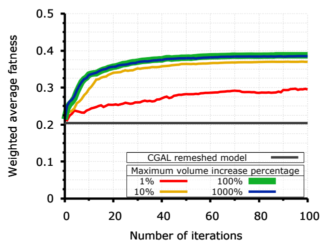
Figure 17: A controlled study with two intersecting bars. From top to bottom: input overlapping bars, and DC-shells created by least-squares fit heuristic, SVM and exact methods.

convex shapes obtained through this approach are consistently lower than ours.

Due to space limit, a complete report on convex remeshing can be found in supplementary material.



(a) Mother

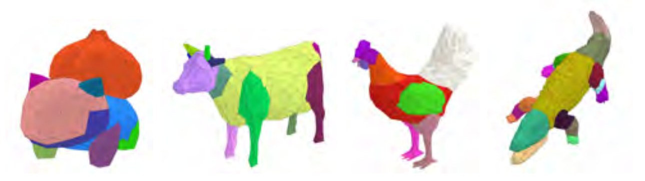


(b) Rocket

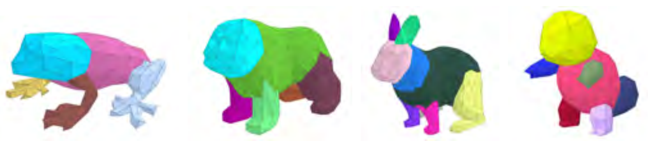
Figure 18: Weighted average fatness with respect to number of iterations at four different percentages of maximum volume increase: 1%, 10%. 100%. and 1000%. More plots can be found in Supplementary Material.

7. Fabricating Physical Models

This section details the application of DC-shell in making paper craft. Composite shapes used in the experiment are shown in Fig. 19. Although not demonstrated here, we believe DC-shell can also facilitate other manufacturing process, including traditional subtractive manufacturing (e.g., wood carving) and modern additive manufacturing methods (e.g., 3d printing).



(a) Bulbasaur (b) Bull (c) Chicken (d) Crocodile



(e) Frog (f) Haechi (g) Rabbit (h) Squirtle

Figure 19: Models used in our fabrication experiments in Section 7.

Generating Foldable Nets. We first unfold the original composite shape and its DC-shells into nets. For fair comparison, we simplify the original model such that the number of triangles equals to total number of triangles of all DC-shells. Once we obtained the optimized nets, motion planning technique [38] is adopted to find a folding motion that continuously transform the net from the flat state to the folded state. Continuous motion is critical which ensures the model is physically realizable, besides, it can also act as a folding guide for human folders. The computation time of unfolding and folding all components is reported in Table 1, from which we can see that DC-shells significantly reduce the unfolding and path planning time by several orders of magnitude. It is important to note that some parts of the composite shapes cannot be unfolded.

Fabrication and Comparison. We compare the folding times required to fold the original shape and DC-shells using three models from Fig. 19 whose original mesh can be unfolded successfully. Recall that the the original shape and DC-shell have the same number of facets. Details on the tools and fabrication steps can be found in Supplementary materials. Fabricated results are shown in Table 2 and Fig. 20. The subjects both commented that nets of DC-shell are easier to fold comparing to that of original mesh patches, and DC-shells are easier to assemble since they share cutting planes while nearly convex patches only share cutting edges.

User study. We conducted a user study with 102 elementary school students during class hours. The students are between 9 and 12 years old, and they worked together in groups of three or four. Depending on their lessons, students fold a subset of models from Fig. 19. Upon completion, students then decorated their finished craft for a puppet show.

To ensure that the students can complete their folding on time, we enlarged some small parts (e.g., horn, ear, beak). We also added square marks and short strokes on to the nets that represent which vertex is a starting point to fold

Table 1: Folding and unfolding time in seconds of segmented parts and their DC-shells. Running times marked with $*$ indicate certain parts of the original models have no valid nets.

	Bulbasaur	Chicken	Bull	Crocodile	Haechi	Rabbit	Frog	Squirtle
Number of triangles	1138	1028	1038	1570	608	936	610	542
Number of parts	12	12	12	17	6	10	6	7
Patch Unfolding Time (s)	112.384	469.288*	462.868*	684.161*	174.831*	95.717*	36.122	15.179
Patch Folding Time (s)	72.468	541.273*	201.821*	89.394*	4.442*	63.931*	294.184	9.584
DC-shell Unfolding Time (s)	0.065	0.068	0.074	0.124	0.032	0.059	0.045	0.034
DC-shell Folding Time (s)	1.255	1.382	1.478	2.031	0.843	0.985	0.732	0.426

Table 2: Fabrication time by two adult subjects

	Bulbasaur	Frog	Squirtle
Original mesh	118 mins	104 mins	90 mins
DC-shells	80 mins	59 mins	60 mins

nets and which edges are assembled, respectively. These two types of marks help students fold without detailed instructions. We taught students the meaning of these marks and ask them to fold each net into a convex shape.

The folded models are shown in Fig. 21. The study didn't show significant differences in fabricating time between 9-year-old and 12-year-old students in Table 3. The younger students did the paper craft as well as the older students did, although Bulbasaur, chicken and bull models have more than 1000 triangles. This indicates that DC-shell eases the fabrication process. In addition, the crocodile model has 2.9 times and 1.7 times more triangles that those of Squirtle and Rabbit, nonetheless, the difference in fabricating time between the crocodile model and the others is only 9%.

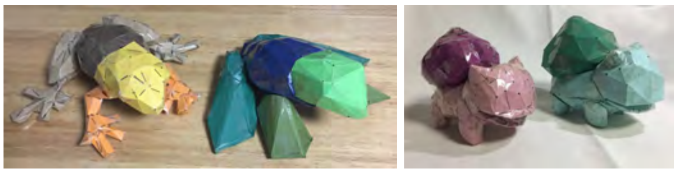


Figure 20: Models built from the composite shape and DC shells.

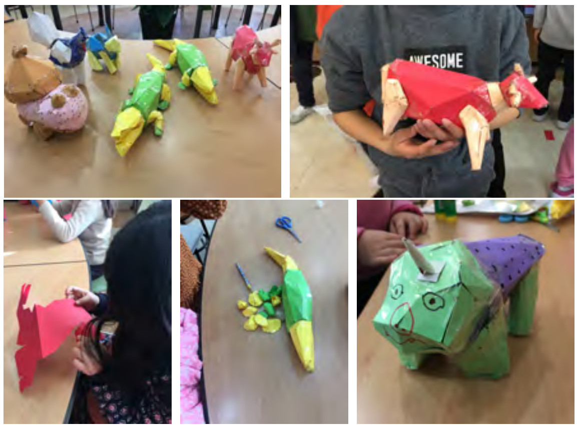


Figure 21: Paper crafts created by 9-year-old school children.

8. Conclusion, Limitations and Future Work

This paper presented an optimization method that creates pairwise disjoint convex sets from a composite shape. We showed that this seemingly simple problem is in fact computational expensive and simple solutions do not work. Our methods combined SVM and exact volume computation to find cut planes that minimize volume loss from trimming the overlap. The quality of the proposed methods was shown experimentally better than heuristic methods using composite shapes from various sources. Mesh unfolding was used to further demonstrate the benefits provided by DC-shells via virtual (algorithm) folders and human folders. Our user studies showed that DC-shells made paper craft creation and design more accessible to school-age children and provided chances to enrich their education experiences.

One major limitation of our approach is its dependency on composite shapes. Future work will investigate generation of DC-shells without segmentation. It is worth noting that Huang and Wang [39] proposed a method to simplify BSP-represented volume by iteratively merge nodes of a BSP tree. The nodes of BSP tree represent disjoint convex volume. In addition, as indicated at the end of Section 3.2.2, volume loss may be further reduced without SVM and minimized with theoretical guarantees.

9. Acknowledgement

We thank Eun Sook Kim, a Master teacher at Seoul Hancheon elementary school, for including our paper crafts as part of the teaching materials in her classes and helping our experiments. This work was supported in part by NSF EFRI-1240459 and CNS-1205260. Many models used in this paper are obtained from `www.thingiverse.com` and all pokemon models are from `roestudios.co.uk/project/3d-pokemon-models/`.

References

[1] B. Chazelle, Convex decompositions of polyhedra, in: Proc. 13th Annu. ACM Sympos. Theory Comput., 1981, pp. 70–79.

[2] H. Edelsbrunner, A. D. Robison, X. Shen, Covering convex sets with non-overlapping polygons, Discrete Math. 81 (1990) 153–164.

Table 3: Fabricating time in minutes. Data collected from groups of three or four 9 and 12-year-old students at an elementary school.

Model	Bulbasaur	Chicken	Bull	Crocodile	Haechi	Rabbit	Frog	Squirtle
Number of triangles	1138	1028	1038	1570	608	936	610	542
Fabricating time (number of students)	47 (3)	35 (3)	50 (3)	65 (4)	63 (3)	48 (3)	50 (3)	49 (3)
	65 (3)	50 (3)	43 (3)	60 (3)	41 (3)	62 (3)	20 (3)	60 (3)
	65 (3)	55 (3)	40 (3)	55 (3)	50 (3)	57 (3)		60 (3)
	75 (3)	45 (3)	48 (3)		33 (3*)	74 (3)		50 (3)
	40 (3*)	45 (3*)	60 (3*)			35 (3*)		55 (3*)
Average time	58.4	46	48.2	60	46.75	55.2	35	54.8

* test conducted by 12-year-old students.

[3] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (1995) 273–297.

[4] J. Mitani, H. Suzuki, Making papercraft toys from meshes using strip-based approximate unfolding, in: ACM Transactions on Graphics (TOG), volume 23, ACM, 2004, pp. 259–263.

[5] I. Shatz, A. Tal, G. Leifman, Paper craft models from meshes, The Visual Computer 22 (2006) 825–834.

[6] F. Massarwi, C. Gotsman, G. Elber, Papercraft models using generalized cylinders, in: Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on, IEEE, 2007, pp. 148–157.

[7] W. Schlickenrieder, Nets of polyhedra, Master's thesis, Technische Universität Berlin, 1997.

[8] M. Ghomi, Affine unfoldings of convex polyhedra, Geometry & Topology 18 (2014) 3055–3090.

[9] C. Bajaj, T. K. Dey, Convex decomposition of polyhedra and robustness, SIAM J. Comput. 21 (1992) 339–364.

[10] B. Chazelle, D. Dobkin, N. Shouraboura, A. Tal, Strategies for polyhedral surface decomposition: An experimental study, Comput. Geom. Theory Appl. 7 (1997) 327–342.

[11] K. Mamou, F. Ghorbel, A simple and efficient approach for 3d mesh approximate convex decomposition, in: Image Processing (ICIP), 2009 16th IEEE International Conference on, IEEE, 2009, pp. 3501–3504.

[12] M. Ghosh, N. M. Amato, Y. Lu, J.-M. Lien, Fast approximate convex decomposition using relative concavity, Computer-Aided Design 45 (2013) 494–504.

[13] O. V. Kaick, N. Fish, Y. Kleiman, S. Asafi, D. Cohen-Or, Shape segmentation by approximate convexity analysis, ACM Transactions on Graphics (TOG) 34 (2014) 4.

[14] G. Liu, Z. Xi, J.-M. Lien, Nearly convex segmentation of polyhedra through convex ridge separation, in: Symposium on Solid & Physical Modeling (SPM); also appears in Journal of Computer-Aided Design, Berlin, Germany, 2016.

[15] R. Straub, H. Prautzsch, Creating optimized cut-out sheets for paper models from meshes, Technical Report, Karlsruhe Institute of Technology, 2011.

[16] S. Takahashi, H.-Y. Wu, S. H. Saw, C.-C. Lin, H.-C. Yen, Optimized topological surgery for unfolding 3d meshes, in: Computer Graphics Forum, volume 30, Wiley Online Library, 2011, pp. 2077–2086.

[17] Z. Xi, Y.-H. Kim, Y. J. Kim, J.-M. Lien, Learning to segment and unfold polyhedral mesh from failures, in: Shape Modeling International (SMI); also appears in Journal of Computers & Graphics, Berlin, Germany, 2016.

[18] D. Julius, V. Kraevoy, A. Sheffer, D-charts: Quasi-developable mesh segmentation, in: Computer Graphics Forum, volume 24, Wiley Online Library, 2005, pp. 581–590.

[19] C. Wang, Computing length-preserved free boundary for quasi-developable mesh segmentation, IEEE Transactions on Visualization and Computer Graphics 14 (2008) 25–36.

[20] G. Song, N. M. Amato, A motion-planning approach to folding: From paper craft to protein folding, Robotics and Automation, IEEE Transactions on 20 (2004) 60–71.

[21] Z. Xi, J.-M. Lien, Folding rigid origami with closure constraints, in: International Design and Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE), ASME, Buffalo, NY, 2014.

[22] Z. Xi, J.-M. Lien, Plan folding motion for rigid origami via discrete domain sampling, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, 2015.

[23] L. Shapira, A. Shamir, D. Cohen-Or, Consistent mesh partitioning and skeletonisation using the shape diameter function, The Visual Computer 24 (2008) 249–259.

[24] J. Lawrence, Polytope volume computation, Math. Comput. 57 (1991) 259–271.

[25] B. Büeler, A. Enge, K. Fukuda, H.-J. Lüthi, Exact volume computations for polytopes: a practical study, in: Abstracts 12th European Workshop Comput. Geom., Universität Münster, 1996, pp. 57–64.

[26] L. Khachiyan, Complexity of polytope volume computation, in: J. Pach (Ed.), New Trends in Discrete and Computational Geometry, volume 10 of *Algorithms and Combinatorics*, Springer-Verlag, 1993, pp. 91–101.

[27] I. Z. Emiris, V. Fisikopoulos, Efficient random-walk methods for approximating polytope volume, in: Proceedings of the thirtieth annual symposium on Computational geometry, ACM, 2014, p. 318.

[28] R. L. Smith, Efficient monte carlo procedures for generating points uniformly distributed over bounded regions, Operations Research 32 (1984) 1296–1308.

[29] H. Edelsbrunner, R. Waupotitsch, Computing a ham-sandwich cut in two dimensions, J. Symbolic Comput. 2 (1986) 171–178.

[30] C.-Y. Lo, W. Steiger, An optimal-time algorithm for ham-sandwich cuts in the plane, in: Proc. 2nd Canad. Conf. Comput. Geom., 1990, pp. 5–9.

[31] S. Bespamyatnikh, D. Kirkpatrick, J. Snoeyink, Generalizing ham sandwich cuts to equitable subdivisions, Discrete Comput. Geom. 24 (2000) 605–622.

[32] C.-Y. Lo, J. Matoušek, W. Steiger, Algorithms for ham-sandwich cuts, Discrete & Computational Geometry 11 (1994) 433–452.

[33] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, B. Lévy, Polygon mesh processing, CRC press, 2010.

[34] P. V. Sander, X. Gu, S. J. Gortler, H. Hoppe, J. Snyder, Silhouette clipping, in: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 327–334.

[35] P. Alliez, E. De Verdire, O. Devillers, M. Isenburg, Isotropic surface remeshing, in: Shape Modeling International, 2003, IEEE, 2003, pp. 49–58.

[36] S. Fuhrmann, J. Ackermann, T. Kalbe, M. Goesele, Direct resampling for isotropic surface remeshing., in: VMV, Citeseer, 2010, pp. 9–16.

[37] D. A. Field, Laplacian smoothing and delaunay triangulations, International Journal for Numerical Methods in Biomedical Engineering 4 (1988) 709–712.

[38] Z. Xi, J.-M. Lien, Continuous unfolding of polyhedra - a motion planning approach, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015, pp. 3249 – 3254.

[39] P. Huang, C. C. Wang, Volume and complexity bounded simplification of solid model represented by binary space partition, in: Proceedings of the 14th ACM Symposium on Solid and Physical Modeling, ACM, 2010, pp. 177–182.