Approximate Pyramidal Shape Decomposition

Ruizhen Hu\textsuperscript{1,2*} Honghua Li\textsuperscript{1} Hao Zhang\textsuperscript{1} Daniel Cohen-Or\textsuperscript{3}
\textsuperscript{1}Simon Fraser University \textsuperscript{2}Zhejiang University \textsuperscript{3}Tel Aviv University

Abstract

A shape is pyramidal if it has a flat base with the remaining boundary forming a height function over the base. Pyramidal shapes are optimal for molding, casting, and layered 3D printing. However, many common objects are not pyramidal. We introduce an algorithm for approximate pyramidal shape decomposition. The general exact pyramidal decomposition problem is NP-hard. We turn this problem into an NP-complete problem which admits a practical solution. Specifically, we link pyramidal decomposition to the Exact Cover Problem (ECP). Given an input shape $S$, we develop clustering schemes to derive a set of building blocks for approximate pyramidal parts of $S$. The building blocks are then combined to yield a set of candidate pyramidal parts. Finally, we employ Knuth's Algorithm X over the candidate parts to obtain solutions to ECP as pyramidal shape decompositions. Our solution is equally applicable to 2D or 3D shapes, and to shapes with polygonal or smooth boundaries, with or without holes. We demonstrate our algorithm on numerous shapes and evaluate its performance.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems;

Keywords: pyramidality, shape decomposition, 3D printing

1 Introduction

Decomposing a complex shape into simpler primitives is one of the most fundamental geometry problems. The main motivation is that most computation and manipulation tasks can be more efficiently executed when the shapes are simple. Perhaps the best known problem instance is convex decomposition [Chazelle 1984] or covering [Cohen-Or et al. 2003]. Other simplicity criteria for surface or solid primitives that have been considered include planarity, compactness, monotonicity, and cylindricality. In practice, approximate versions of the corresponding decomposition problems are often solved, where the resulting components are only approximately convex, planar, etc.: this tends to produce fewer primitives.

We are interested in a shape decomposition problem where the simple primitives sought are pyramidal. A shape $S$ is pyramidal if it has a flat base $B(S)$ (as part of the boundary of $S$) and for any point $p$ inside $S$, the line segment between $p$ and the perpendicular projection $p'$ of $p$ onto $B(S)$ lies entirely inside $S$; see Figure 2(a). Each pyramidal shape $S$ has a designated orientation $O(S)$, which is the up-vector with respect to the base. Obviously, each point on $S$ must be visible from some point on the base, along $O(S)$. The boundary region of $S$ opposite to $B(S)$, along $O(S)$, defines a height function over the flat 2D domain $B(S)$. Compared to other simplicity criteria such as convexity, pyramidality seemingly admits more complex geometry. A desirable consequence is that pyramidal decomposition tends to produce a smaller set of primitives, e.g., as opposed to convex decomposition; see Figure 2(b) vs. (c).

The simplicity of pyramidality mainly owes to its functional nature, since processing of pyramidal shapes is confined to 2.5D. For example, if height values are pre-computed and stored, testing whether a point is inside a pyramidal shape takes constant time. The application which originally motivated our study of pyramidality is 3D printing via Fusion Decomposition Modeling (FDM). FDM is one of the dominant technologies for 3D fabrication. Its layer-based additive printing scheme requires support material to be injected (and later removed) to produce overhangs in a fabricated shape. If the shape is pyramidal, then no support material is needed, leading to savings in both material and print time. Last but not the least, any pyramidal shape is moldable [Rappaport and Rosenbloom 1994; Priyadarshi and Gupta 2004] — one can inject cement or chocolate into a mold and then lift the mold along a direction while leaving the hardened shape completely intact without any breakage.

The goal of exact pyramidal decomposition is to find a decomposition of a 2D or 3D shape with the minimum number of pyramidal parts. This is known to be a hard problem. Fekete and Mitchell [2001] proved that both the 3D version of the problem and the 2D version on polygons with holes are NP-hard. It was suspected that on simple polygons, the problem might be polynomial. However,
we are not aware of any construction algorithm, exact or approximate, for the pyramidal decomposition problem.

In this paper, we address the approximate pyramidal decomposition problem. That is, we are content with the obtained parts being only approximately pyramidal, while still aiming to produce as few parts as possible. This implies that for FDM-based 3D printing, some small amount of material waste may be inevitable. In the case of mold design, the geometry of the produced shape may have to be slightly altered. In practice, exact pyramidal decomposition of many natural objects or man-made shapes with fine geometric details would result in too many parts. Approximate pyramidal decomposition brings the part count to a more manageable level, offering more practical solutions to real-world problems.

Our approach to pyramidal decomposition turns an NP-hard problem into an NP-complete one which admits a practical solution. Specifically, we link pyramidal decomposition to the Exact Cover Problem (ECP) [Vazirani 2001]. ECP takes as input a collection of subsets which together cover a given set \( \mathcal{U} \) and seeks a sub-collection of the subsets which provides an exact (non-overlapping) cover for \( \mathcal{U} \). Given an input shape, any decomposition or partitioning forms an exact cover. As pyramidal decomposition seeks a decomposition with a small number of pyramidal parts, it becomes an ECP problem if we already have a set of pyramidal parts that cover the input shape. As there are infinitely many pyramidal parts in any given shape, e.g., all cuboids contained in the shape are pyramidal, the challenge is to build an appropriate set of candidate pyramidal parts which subsumes optimal decompositions.

We build the set of candidate pyramidal parts bottom-up via clustering. The clustering is based on the likelihood that two primitives interior to the input shape belong to the same, large pyramidal part. Preference is given to larger pyramidal parts since they are more likely to contribute to minimum-sized decompositions. Through clustering, we progressively build larger and larger solid primitives (see Figure 3), which we call cells and then blocks, with each candidate pyramidal part formed by merging blocks. The cells and blocks themselves are not necessarily pyramidal; they resemble superpixels from an over-segmentation [Achanta et al. 2012]. Cells roughly correspond to intersections between large pyramidal parts in the input shape while blocks are formed by merging cells to bring down the count for candidate pyramidal parts.

To the best of our knowledge, we present the first construction algorithm for pyramidal decomposition, exact or approximate. Our focus is on the latter. We demonstrate our algorithm on numerous shapes and evaluate its performance by comparisons to a greedy approach, user judgement, and the ground truth, if attainable. When the input admits a moderately-sized exact pyramidal decomposition, our algorithm tends to find it, or at least a solution close to it. However, since the decomposition depends on clustering, we cannot provide theoretical guarantees or provable bounds.

That said, our clustering-based solution through the connection to set cover is merited by both generality and versatility. In particular, our approach is equally applicable to 2D or 3D shapes, to shapes with polygonal or smooth boundaries, and to shapes with or without holes. Moreover, allowing the pyramidal parts to overlap, i.e., not insisting on a partition, is straightforward as one only needs to change the last step of the algorithm from solving an exact set cover problem to solving an inexact one.

2 Related work and concepts

The majority of methods for shape decomposition or segmentation have been designed for surface decomposition [Shamir 2008], rather than solid decomposition like the problem we tackle in this paper. Segmentation methods can be broadly classified as either top-down or bottom-up. Recursive partitioning is a representative top-down approach. A top-down greedy search would extract the largest pyramidal part and then recurse on the remaining shape. This is clearly suboptimal as it lacks the foresight to prevent fragmentation of the remaining shape. Also, for an approximate decomposition, it is difficult to determine the tolerance allowed when extracting each pyramidal part. Region growing is a typical bottom-up approach. Our work relies on a bottom-up clustering scheme to construct the candidate pyramidal parts.

Convexity. While there are clear differences between convexity and pyramidality, e.g., as one may observe from Figure 2, there are also apparent connections. In particular, one may view pyramidality as a form of “vertical convexity” with the verticality in reference to a flat base. Specifically, while convexity requires the line segment between any two points interior to the shape to lie entirely inside the shape, pyramidality confines the line segment to be perpendicular to the base. Exact convex decomposition is a well-studied problem [Chazelle 1984; Tor and Middleditch 1984], whose complexity status appears to mirror that of pyramidal decomposition. An interesting variation to convex partitioning is to allow the convex parts to overlap [Cohen-Or et al. 2003].

Approximate convex decomposition. Approximate convex decomposition has also received much attention over the years. The approach of Lien and Amato [2007] is perhaps the best known; it operates top-down and recursively divides an input shape by a best cut at concave surface features. While boundary analysis is more efficient than processing shape interiors as in our work, there is no clear boundary characterizations for pyramidality. Recent work of Asafi et al. [2013] takes a bottom-up approach, obtaining clusters.

Figure 3: Overview of our approximate pyramidal decomposition algorithm. Starting with a uniform point sampling of the interior of the input shape (a), a series of clustering steps progressively build cells (b), blocks (c), and finally a set of candidate pyramidal parts (d). Clustering is based on the likelihoods that two primitives, e.g., points, cells, or blocks, belong to the same pyramidal part. An Exact Cover Problem (ECP) is solved over the candidate pyramidal parts to obtain the final pyramidal decomposition.
Monotone polygons and chains. A polygon \( P \) is said to be monotone with respect to a line \( L \) if any line orthogonal to \( L \) intersects \( P \) at most twice. Similarly, a polygonal chain is monotone with respect to \( L \) if any line orthogonal to \( L \) intersects the chain at most once. A monotone decomposition of a polygon is a partition of its boundary into a minimum number of monotone chains [Chandru et al. 1992]. Liu and Ntafos [1988] studied the problem of partitioning a polygon into a minimum number of uniformly monotone polygons, where uniformity requires the resulting polygons to be monotone with respect to a common line.

Clearly, a pyramidal polygon is always monotone with respect to lines parallel to its base and the non-base boundary of a pyramidal polygon is always a monotone chain. However, a monotone polygon is not always pyramidal. Moreover, a monotone (chain) decomposition only partitions the boundary of a shape, not its interior. It is unclear how one may turn such a solution into a solid decomposition consisting of a general set (without the uniformity criterion) of monotone polygons, let alone pyramidal polygons. Figure 4(a) shows that connecting the two ends of a monotone chain may not even produce a simple polygon. When the shape boundary contains multiple surfaces, e.g., for polygons with holes or hollowed 3D models, connecting the surface decompositions to form a proper solid decomposition is far from straightforward; see Figure 4(b). At last, 3D extensions for monotone decomposition appear non-trivial.

In our work, we study pyramidal decomposition, which is necessarily a solid decomposition problem, for both 2D and 3D shapes.

Moldability. A polygon model is \( k \)-moldable if there is some partition of its boundary into \( k \) pieces such that each piece is remov-able in some direction after a material injection process [Priyadarshi and Gupta 2004]. Most relevant research on mold design focuses on 2-moldability [Rappaport and Rosenbloom 1994; McMains and Chen 2005; Li et al. 2009]. As a surface decomposition and constrained by the need to disassemble the molds, \( k \)-moldability may yield no solution while \( k \)-pyramidal decomposition does, e.g., see Figure 4(c). Figure 4(a) shows that there is no direct connection between solutions to \( k \)-moldability and solutions to \( k \)-pyramidal. Instead of designing molds to build a given model as a whole, a pyramidal decomposition yields solid parts each of which is 1-moldable; the parts can then be glued to form the input shape.

Elevation maps. Many methods for segmentation perform surface decomposition [Shamir 2008], where a desirable property of the resulting patches is that each can be parameterized as an elevation map or a terrain over a planar domain. The patch representation would facilitate tasks such as Fourier analysis [Pauly and Gross 2001] and geometry compression [Ochotta and Sauer 2008], among others. Decomposition into elevation maps may be regarded as a 2-manifold version of monotone chain decomposition, where patch boundaries are more complex; they are at least non-planar in general. There is again no straightforward means to convert a surface decomposition into a solid, e.g., pyramidal, decomposition.

Terrain decomposition. During our investigation into pyramidal decomposition, we discovered that a pyramid shape corresponds exactly to a polygon or polyhedron, which was studied by Fekete and Michell [2001]. Additionally, they also identified layered manufacturing as a motivating application for terrain decomposition. However, their work only provides NP-hardness proofs for several instances of the exact terrain decomposition problem and does not offer a construction algorithm. We have found no published follow-up works on terrain decomposition nor any study of the approximate version of the problem.

3D printing. Recent advances in 3D printing technology are powering a revolution in rapid prototyping and manufacturing. New geometry problems aimed at improving the process, results, and reducing the cost of shape fabrication have been emerging. Cali et al. [2012] proposed a method for converting 3D models into printable, functional, non-assembly models with internal friction. Works by Stava et al. [2012] and Prévost et al. [2013] focus on physical properties of printed objects including structural strength and standability. Luo et al. [2012] posed a decomposition problem motivated by 3D printing where the goal is to cut a 3D model into parts each of which can fit, size-wise, into the printing volume. Most recently, Wang et al. [2013] proposed to reduce the material cost for 3D printing by hollowing the inside of a 3D object. For the purpose of 3D printing, pyramidal decomposition also aims to reduce the cost of material and print time, but takes a completely different approach compared to [Wang et al. 2013] and relies on a completely different decomposition criterion compared to [Luo et al. 2012].

3 Overview

The input to our algorithm is a closed 2D polygon or 3D polygonal mesh. The output is an enumeration of pyramidal decompositions ordered by any user-chosen quality criterion. Our algorithm progressively builds larger and larger interior elements of the input shape, from sample points to cells, blocks, and candidate pyramidal parts, and ends with a selection which solves the ECP to obtain the final decompositions; see Figure 3. For simplicity and ease of illustration, the algorithm is mainly described for 2D shapes; extension to 3D is straightforward and discussed towards the end.

Rationale for progressive clustering. Since clustering leads to partitions and the candidate pyramidal parts we seek should overlap each other, they cannot be obtained by directly clustering the point primitives. Our strategy is to find intersections of large pyramidal parts which cover the input shape. The intersections form a partition and they can be merged appropriately to form an overlapping set of candidate pyramidal parts. Obviously, without knowing what these parts are, we must estimate the intersections. We accomplish that with the initial merging step, leading to cells. Blocks are formed by clustering cells to reduce the number of primitives that need to be considered for building candidate pyramidal parts.
Cells. We start by a uniform point sampling of the interior of the input shape. Cells are the next-level shape primitives as fundamental building blocks of pyramidal parts. Our goal is for each cell to contain point samples which are likely to belong to the same large pyramidal parts. In other words, a cell is unlikely to straddle the boundary of a pyramidal decomposition. In this way, the cells can be seen roughly as intersections of these parts. Our approach to cell generation is to let each sample point \( p \) vote for bases of large pyramidal parts which contain \( p \). Each cell is formed by sample points which voted for the same set of bases.

Blocks. We cluster cells into blocks via an affinity measure which integrates cell capacity over all sampled base directions. Interior block boundaries are refined to be more straight (or flat in the 3D case) to better serve as potential bases. This consolidation step considerably reduces the number of atomic units for constructing pyramidal parts and improves their quality.

Candidate pyramidal parts. We merge blocks which share, approximately, at least one common base into a candidate part. By design, each block is a candidate itself, hence it is guaranteed that the union of the set of candidates cover the input shape. We define a pyramidality score and only consider merged candidate parts whose scores pass a prescribed threshold. Each resulting candidate pyramidal part has a designated base and an associated score.

Final decomposition. We apply the Algorithm X [Knuth 2000] to the set of candidate pyramidal parts, which enumerates all the exact covers of the input shape extracted from the candidate set. We can filter as well as sort the solutions produced using any user-specified criterion, such as average pyramidality score.

4 Decomposition algorithm

Given an input shape \( S \), we start by a uniform point sampling of its interior, then progressively build increasingly larger atomic solid primitives towards candidate pyramidal parts. Accordingly, our analysis starts locally and becomes more global as the clustering progresses. We describe each step in this section with Sections 4.1-4.4 covering the 2D case and Section 4.5 on the 3D extensions.

4.1 Cell generation

Denote the sample points in \( S \) by \( p_1, \ldots, p_n \). We let each \( p_i \) vote for a set of bases of pyramidal parts. However, there are too many bases to consider. To reduce the search cost, we sample a set of \( m \) base directions which correspond to \( 2m \) up-vector directions. With respect to each up-vector direction \( u_k \), \( k = 1, \ldots, 2m \), each point \( p_i \) votes for one base \( B_{ik} \) with the highest capacity. The capacity of a base depends mainly on the area of the maximal pyramidal parts induced by the base. Higher-capacity bases receive more votes and support larger pyramidal parts.

We unify all the voting results, from \( n \) sample points and over \( 2m \) up-vector directions, into a base vote matrix \( M = (b_{ik})_{n \times 2m} \), where \( b_{ik} \) is the index of the base that \( p_i \) voted along up-vector \( u_k \). With the premise that points are likely to be in the same pyramidal part if they voted for the same set of bases, we collect points with identical rows to form a cell.

Sampling base directions. We sample a set of base directions \( \mathcal{D} = \{d_1, d_2, \ldots, d_m\} \), with two up-vectors \( u_{j-1} \) and \( u_j \) associated to \( d_j \), and confine our pyramidality analysis to these directions. In particular, the base of each final pyramidal part is along one of the \( d_j \), \( j = 1, \ldots, m \). The base directions can be uniformly sampled, which may be suitable for smooth shapes, or adapt to salient shape features, which is appropriate when certain bases, such as dominant faces of a CAD object, are preferred. In the latter case, preferred directions are selected first with the remaining ones sampled uniformly to cover the 360-degree angle space.

Boundary bias in base sampling. For many man-made shapes, large flat faces are often good bases of pyramidal parts. We thus introduce a boundary bias when sampling base directions. Specifically, given the input 2D polygon \( S \), we collect all of its edge directions and assign weights to them based on edge lengths. Then we recursively select the direction with the highest weight that is at least 5 degrees away from any already selected direction until only directions with weight smaller than a threshold remain. After sampling \( m_0 \) such directions, the remaining \( m - m_0 \) directions are sampled uniformly over the uncovered angle space.

Base capacity. Let \( g \) be a line segment inside or along the boundary of \( S \). We denote by \( u, v \) the pair of opposite up-vectors associated with \( g \). We define the (base) capacity of \( g \) as

\[
\text{capacity}(g) = \frac{\text{pam}(g, S)}{\text{part}(g, S)},
\]

where \( \text{pam}(g, S) \) is the sum of areas of all maximal pyramidal parts based on \( g \) inside \( S \), either with up-vector \( u \) or with up-vector \( v \). The denominator \( \text{part}(g, S) \) is the number of connected regions which would result from decomposing \( S \) using the base and its associated one or more maximal pyramidal parts.

Our definition of base capacity is admittedly heuristic. Considering only the maximal pyramidal parts expresses our preference for larger pyramidal parts. Having \( \text{part}(g, S) \) in the denominator correlates with the desire to induce fewer parts. Lastly, our voting scheme is applied only to bases \( g \) which straddle the shape boundary, i.e., the end points of \( g \) lie on the boundary of \( S \); we do not consider bases which lie entirely inside \( S \).

Figure 5 shows two examples of base capacity measurement with respect to two base directions. Parts shown in light and dark gray are maximal pyramidal parts induced by the base. The value of \( \text{part}(g, S) \) induced by the base shown in (a) is \( 3 \) and \( 4 \) for (d). The color plots show the capacity of bases which straddle the shape boundary.

Base voting. For each base direction, each point votes twice along its two corresponding up-vectors. Given an up-vector direction \( u_k \), we project each point \( p_i \) along direction \( -u_k \) and consider all perpendicular bases in the closure of \( S \) whose induced maximal pyramidal parts contain \( p_i \). Among these bases, \( p_i \) votes for the single base which has the highest capacity. In Figure 6(a), along the up-vector \( u \), points \( p_2 \) and \( p_3 \) both have the bottom line as the unique top-capacity base to vote for. For point \( p_1 \) however, a slab of bases (green strip) all have the same capacity. Let \( \mathcal{A} \) denote the set of all such bases. If \( p_1 \) is above the median base in \( \mathcal{A} \), position-wise along \( u_k \), then we let \( p_i \) vote for that median base. Otherwise,
the set of bases, with respect to up-vectors \( S \) is therefore highly efficient to carry out. While the cells form a par- 

Figure 6: Base voting and filtering. (a) Points vote for their highest-capacity bases along two up-vector directions. The bottom base is voted by both \( p_2 \) and \( p_3 \). The median position base among a set of bases (green strip) having the same capacity is voted by both \( p_1 \) and \( p_2 \). The inset shows a histogram of vote counts. (b) A base direction along which the vote distribution over all bases has low variance. (c) Four bases receiving a low number of votes. All bases covered in (b) and (c) are filtered out. (d) Cells obtained before base filtering are more fragmented than those obtained after; cells generated in the latter case are shown in Figure 8(a).

we let \( p_1 \) vote for the lowest base in \( S \). We index all voted bases and place their indexes into the matrix \( M \).

Vote sharpening. In the last step, prior to cell generation, we sharpen the base vote matrix \( M \) by setting some of its entries to zero. The rationale is that some bases being voted on do not have sufficient support from the sample points and as such, they may add noise to \( M \) and lead to fragmented cells. Sharpening is effectively a merging step, leading to a more coherent set of cells.

We rely on two criteria to filter out low-support bases. First, for each base direction \( d \), we measure the variance of the number of votes cast to each base. If the variance is lower than a threshold, then all bases associated with \( d \) are discounted with their corre- 

Cell formation. After vote sharpening, we simply collect all points with exactly the same rows in \( M \) to form a cell. This first clustering step involves only a one-pass scan of the matrix \( M \) and is therefore highly efficient to carry out. While the cells form a par- 

Figure 7: Base selection and cell generation. (a) Bases perpendicular to up-vectors \( u \) and \( v \) which received sufficient votes. (b) and (c) show regions of points that voted for the same base (same color with the voted base indicated) with respect to \( u \) and \( v \). Gray regions contain points who voted for bases that do not have suf- 

tering. These bases induce regions of points which voted for the same base; they are shown in uniform (non-gray) colors in Figures (b) and (c), respectively for \( u \) and \( v \). Each region defines a cell with respect to an up-vector. The final set of cells can be seen as an in- 

4.2 Block construction

To reduce the number of primitives for candidate pyramidal part selection (Section 4.3), the most time-consuming step of our pipeline, we merge adjacent cells into larger and fewer blocks.

Cell clustering. The merging is carried out by a clustering process based on an affinity matrix \( A \) defined over the set of cells. The matrix \( A \) combines per-direction affinities over all the sampled up- 

Figure 8: Block generation. (a) Cells. (b) Result of cell clustering. (c) Initial block boundaries by tracing edges of triangulation. (d) Refined block boundaries via straight line fitting and final blocks.

vectors: \( A = \sum_{k=1}^{2m} A_k \), where the affinity between two cells \( C_i \) and \( C_j \) along up vector \( u_k \) is defined as

\[
A_k(i, j) = \begin{cases} 
\text{capacity}(B_{jk}), & \text{if } B_{ik} = B_{jk} \\
0, & \text{otherwise}
\end{cases}
\]

where \( B_{jk} \) is the base which cell \( C_j \) voted along the up-vector \( u_k \). After merging cells into a block, the block is also associated with a set of candidate up-vector directions, which is the union of the set of up-vectors from the constituting cells.

We employ Normalized Cuts (NCut) [Shi and Malik 2000] to the affinity matrix \( A \) to cluster the set of cells. However, determining the right number of clusters is non-trivial, and it has an influence on the quality of results. We currently resort to the typical heuristic of enumerating over a varying number of clusters and relying on a quality criterion to determine the right cluster count. Ultimately, the result of our algorithm should be judged by the quality of the final pyramidal decompositions obtained. We define such a quality measure (6) and discuss its pros and cons in Section 4.4. Instead of enumerating over all cluster counts, we first estimate \( k \) using self- 

More clusters do not necessarily lead to better decompositions, as one might have expected. A result with \( k_2 \) clusters typically does not refine results with \( k_1 \) clusters, where \( k_2 < k_1 \), especially when \( k_1 \) and \( k_2 \) are relatively close to each other; see Figure 9 for a simple example. On the other hand, more clusters increase the cost of computing candidate pyramidal parts significantly.

Note that after vote sharpening, one may directly cluster the point samples into blocks using a sample-to-sample affinity measure similar to (2). The significance of having the intermediate step of cell formation is two-fold. Conceptually, cells correspond to intersections between large pyramidal parts that potentially appear in the fi-
nal decomposition; they are atomic building blocks of these pyramidal parts; hence clustering cells retains the integrity of these parts. In practice, the number of cells is typically significantly less than the number of point samples, resulting in performance gains when spectral clustering is executed on cells rather than point samples. Such time savings are typically in the order of hundredsfold when spectral clustering is performed on a few hundred cells rather than on thousands of point samples. Moreover, the cost of cell formation is quite minimal and the number of cells does not grow with the number of point samples.

Whether or not the cell formation step is included, the blocks obtained at the end are typically the same, as long as the same affinity measure based on the sharpened base vote matrix is applied for clustering. However, our cell formation ensures that point samples receiving votes from the same set of bases are clustered, which is desirable for the blocks. Clustering point samples directly, e.g., via spectral clustering, does not provide such a guarantee and may result in unreasonable blocks, as shown in Figure 10.

Block boundary. Initial boundaries for a block are obtained by a dual Delaunay triangulation of the sample points inside the input shape and then taking the triangle edges along the boundaries of the cells which belong to that block. Figure 8(e) shows an example of initial block boundaries obtained. Due to imprecisions of clustering results, noisy boundaries are typical. For practical purpose, we prefer straight cuts. Therefore we refine the boundaries by polygonal line fitting via a simplified version of the Douglas-Peucker algorithm [Douglas and Peucker 1973]. We start with a line segment connecting the two ends of a piece of boundary, run Douglas-Peucker and stop as soon as the result lies inside the shape. Figure 8(d) shows a result with straight block boundaries, yielding the final set of blocks for candidate pyramidal part selection.

4.3 Candidate pyramidal part selection

Candidate pyramidal parts are formed by merging blocks (Section 4.2). We set up a block graph whose nodes are the blocks and whose edges are given by block adjacency. Each block is associated with a set of candidate up-vectors. Any connected subgraph whose blocks share at least one common up-vector is a candidate part. The part needs to be sufficiently pyramidal to be considered for constructing the final decomposition (Section 4.4).

Figure 10: Block construction with and without cell formation. (a) A simple shape with four designated bases (the four boundary edges); dashed lines show the induced cuts after vote sharpening. (b) Five cells are obtained by examining the vote matrix. (c) Results of clustering point samples directly. Samples in the same intersection (region 1) are assigned to different clusters, which would result in unreasonable blocks.

4.4 Final pyramidal decomposition

The NP-complete Exact Cover Problem (ECP) takes as input a set \( \mathcal{U} \) of elements, called the universe, and a collection \( \mathcal{C} \) of subsets of \( \mathcal{U} \). It seeks a sub-collection \( \mathcal{C}^* \subseteq \mathcal{C} \), where each element in \( \mathcal{U} \) appears in exactly one subset in \( \mathcal{C}^* \). We formulate the pyramidal decomposition problem as an ECP as follows.

ECP formulation and solution. The set \( \mathcal{B} \) of blocks we obtain (Section 4.2) form a partition of the input shape \( S \). We take \( \mathcal{B} \) as the universe for ECP. Each candidate pyramidal part is formed by a subset of blocks. Thus the collection of candidate pyramidal parts, \( \mathcal{C} \), is a collection of subsets of \( \mathcal{B} \). For each candidate part \( P \), we compute its pyramidality score (4) along all the candidate up-vectors for computing.

Pyramidality score. For a connected part \( P \), we define its pyramidality score as follows:

\[
\text{pym}^*(P) = \max_u \text{pym}(P, u),
\]

where \( u \) is one of the up-vector directions considered for computing the score. Pyramidality of \( P \) with respect to \( u \) is estimated by

\[
\text{pym}(P, u) = 1 - \frac{\text{pyram_deficit}(P, u)}{\text{proj_area}(P, u)},
\]

where \( \text{proj_area}(P, u) \) is the projected area of \( P \) along \( u \) and \( \text{pyram_deficit}(P, u) \) is the area of the pyramidal deficit region. To define the projected area, we locate the lowest point \( p \) of \( P \) with respect to direction \( u \) and form a base line passing through \( p \) and perpendicular to \( u \). The set of points which lie between the base and any point of \( P \) along \( u \) form the projected region and the area of that region is the projected area. The pyramidal deficit region is formed by the set of points in the projected region which lie outside the shape \( P \). We estimate areas over uniform 2D grids.

Figure 11 shows several examples of pyramidality estimates. A perfect pyramidal part would obtain a maximum score of 1. However, due to discrete sampling of up-vector directions in our work, such a part may not attain the maximum score.

Candidate part traversal. In the block graph, we examine all possible candidate parts incrementally, sorted by their block counts. Initially, each block (a graph node) is a candidate part with size 1. After generating all parts with size \( j \geq 1 \), parts with size \( j + 1 \) are formed by adding one adjacent block \( K \) to each size-\( j \) part \( P \), if \( K \) shares at least one candidate up-vector with \( P \). The set of candidate up-vectors for a part, formed by several blocks, is the intersection of the sets of candidate up-vectors for the blocks.

Candidate pyramidal part. For each candidate part \( P \), we compute its pyramidality score (4) along all the candidate up-vectors for \( P \). The pyramidality score for \( P \) is the maximum obtained. We designate a base for \( P \) as the base which attains that maximum. If \( P \)'s score passes a user-specified threshold \( g^* \), then it is a candidate pyramidal part and considered for pyramid decomposition.
parts provide the collection of subsets of \( \mathcal{R} \) as part of the input to ECP. Now it is clear that a solution to ECP gives an approximate pyramidal decomposition of \( S \). We employ Knuth’s Algorithm X [Knuth 2000] to find all solutions to the ECP.

Note that given a particular threshold \( y^* \), the set of candidate pyramidal parts may not collectively cover the input shape. Figure 12 (bottom) shows a collection of subsets, i.e., the candidate parts, which cover the input shape. Note that we encode the input shape by block indexes; see Figure 12(a). Figure 12(b) shows the optimal decomposition with 2 parts while (c) and (d) show two decomposition solutions having 3 parts. The exact set covers are shown, by matching colors in the bottom table.

**Decomposition quality.** Given a set of solutions to ECP, we need a means to evaluate their quality. Part count would have been the ultimate quality measure for exact pyramidal decompositions, but not for approximate ones. In our current work, we employ an estimate of material saving, aimed at 3D printing applications, to define decomposition quality. Specifically, we define the material saving for printing a decomposition of shape \( S \), i.e., printing the resulting parts \( P_1, \ldots, P_l \) rather than \( S \) as a whole, as

\[
saving(P_1, \ldots, P_l) = 1 - \frac{\sum_{i=1}^l waste(P_i)}{waste(S)},
\]

where \( waste(\cdot) \) denotes the least amount of material wasted when 3D-printing a shape using FDM. When estimating waste, we do consider orientation of the shape/part when layer-printed — we use the best one. That is, we define \( waste(P_i) \) as the smallest area/volume of the pyramidal deficit region for part \( P_i \), over all sampled up-vector directions; see Section 4.3. Compared to (3), material waste is an actual area/volume measure while the pyramidal density score is normalized with respect to area/volume.

One may view (5) as the gain, in terms of material saving when 3D printing, that is afforded by the decomposition. The maximum value for the measure is 1. A value close to 1 implies that the material waste for printing the decomposed parts is negligible relative to that for printing the shape as a whole.

The material saving (5) is defined for a fixed part count \( l \). However, a small number of parts is often desired. Part count can be factored in a measure of average material saving:

\[
pd\_score(P_1, \ldots, P_l) = saving(P_1, \ldots, P_l)/l,
\]

which we currently adopt, and maximize, as the quality score for searching and enumerating pyramidal decompositions.

**Ranking decompositions.** For a given part count, we select the best decomposition which gives the largest amount of material saving (5). However, we do not present a single solution among these best results which has the highest \( pd\_score \), but all of them sorted by the score. This is because the quality score is still somewhat subjective and the desirable properties for a decomposition may be object- and application-dependent. In practice, the user can choose among the sorted decomposition results.

**Overlapping cover.** Aside from 3D printing, molding, and casting, other applications, e.g., collision detection, may find an overlapping pyramidal cover useful, where the obtained parts are allowed to overlap. This is easy to accomplish within our solution pipeline. Only the last step needs a change from solving an ECP to solving an overlapping cover problem. Since the latter encompasses the former, from the same set of candidate pyramidal parts, the best overlapping cover, in either part count or \( pd\_score \) (6), is always at least as good as the best pyramidal decomposition. We adopt the greedy algorithm of Chvatal [1979] to compute overlapping covers. Figure 13 shows two results on star-like shapes, for which allowing for overlaps effectively reduces the part count.

**4.5 Extension to 3D**

Changing our decomposition algorithm from 2D to 3D is fairly straightforward. The following is a list of changes worth noting:

- **Base sampling:** In 3D, there are no longer base directions as each base is part of a plane. Thus we sample base orientations, i.e., up-vectors, instead.
- **Boundary bias:** Instead of biasing on edge directions, we collect face orientations and accumulate face areas as weight.
- **Interior processing:** In 2D, we rely on a uniform 2D grid inside the shape \( S \) to estimate projected areas, to obtain maximal pyramidal parts, and to infer connected regions. In 3D, we naturally work with a uniform voxel grid. Extending the above tasks from 2D grids to 3D grids is straightforward.
- **Base capacity:** Same as the 2D case, we only consider a base which intersects the input shape \( S \) along its entire boundary.
- **Block boundary refinement:** To obtain cell boundaries, we construct a dual Delaunay tetrahedralization of the point samples. Any triangle face that separates two tets containing points from adjacent blocks is on a block boundary. Block boundary refinement is a mesh simplification problem. There are different mesh variants of the Douglas-Peucker algorithm, e.g., see [Heckbert and Garland 1997]. However, it is beyond the scope of our current pursuit to seek a general solution. In our current implementation, we simply find a best-fitting plane to the set of points on a boundary patch. A desirable consequence of this simple solution is that all the interior part boundaries obtained are piecewise planar, allowing the parts to be easily glued together to assemble the full model.
5 Results and evaluation

In this section, we first show results for approximate pyramidal decomposition and then evaluate our algorithm. Experiments are conducted on both synthetic shapes and models obtained from various repositories. Our dataset contains 149 two-dimensional shapes and 80 three-dimensional shapes. Many 2D shapes come from the Brown database [Sharvit et al. 1998] consisting of 99 binary images from 9 different categories. Most 3D models come from AIM@SHAPE, Archive 3D, and other on-line resources. Some models were selected or synthesized to test specific aspects of our algorithm, e.g., the ability to handle holes.

Parameters. Our algorithm has four main parameters: number of sample points \( n \), number of sampled base directions \( m \), number of clusters \( k \) for block generation, and the pyramidality threshold \( y^* \) for candidate part selection. All the results shown in the paper were obtained with \( n = 5,000 \) and \( m = 36 \), and by sampling the base directions with boundary bias. Increasing the number of sampled base directions tends to improve results, in particular with lower values of \( m \). However, beyond \( m = 36 \), we find the change to be quite minimal. Note that there is no guarantee that the increase always improves decomposition. This is due in part to the difficulty and hence sub-optimal results of clustering analysis.

Instead of fixing \( k \), we enumerate over a short sequence (typically 2 to 10) of values for \( k \); see Section 4.2. For the pyramidality threshold, we start with \( y^* = 0.95 \), but if no decompositions are found, we automatically decrease \( y^* \) by 0.05 at a time until a decomposition is found or it reaches the pyramidality of the input shape. There are a few other thresholds all of which hold fixed values. The threshold for voting sharpening is set to be the variance of a hypothesized vote distribution, where we assume \( n \) point samples evenly vote for \( n_b \) bases along that direction. In our experiments, we set \( n_b = \lceil n/100 \rceil \).

Decomposition results. Figures 14 and 15 show a gallery of approximate pyramidal decompositions found by our algorithm for a variety of shapes, 2D or 3D, CAD-like or organic with smooth boundaries, and with or without holes, demonstrating the generality and versatility of our approach. These results all achieve the best average material saving score (6), which appears to provide a strong bias towards low part count.

Table 1: Statistics and timing, measured on an Intel(R) Core(TM) 3.40Hz CPU with 16GB RAM, for pyramidal decomposition. Number of cells, blocks, and candidate pyramidal parts obtained are denoted by \#c, \#b, and \#p, respectively. We report the total processing time for computing the decompositions, as well as estimated (5) and real (in parentheses) material savings incurred.
In Figure 16, we show several different decompositions, for a few shapes in Figure 15, which have the best material savings (5), regardless of part count. Since the part counts are still quite low, one may regard these results as better than their counterparts in Figure 15. As shown in Figure 17, increasing part counts typically results in more material saving, as one would expect.

Figure 18 shows additional results on more complex models leading to higher part counts. Of particular interest is the decomposition of models with interior cavities, e.g., the hollow SIGA model with the SIGGRAPH logo and the cube composed of a Hilbert curve structure. Printing of such a model as whole would result in support material interior to the shape (see Figure 19) which would be hard to remove. This situation provides an added motivation for shape decomposition prior to 3D printing.

Statistics. Table 1 provides timing and other statistics for decomposing models in the top two rows of Figure 15. Primitive counts reported are associated with the result achieving the best quality score (6) over the choices for $k$, the cluster count. It is clearly evident that the progressive construction of cells, blocks, and candidate parts leads to significant reduction of the (ECP) problem size. Reported timing accounts for searching over all cluster counts, which explains the relatively long execution time. The most time-consuming components of the algorithm are cell generation (about 20% time spent) and candidate pyramidal part selection (70%).

Estimated vs. real material saving. The material saving measure (5) employed by our algorithm is only a computational estimate since the $\text{waste}(\cdot)$ measure, which appears in both the denominator and numerator of (5), is an over-estimate. State-of-the-art 3D printers, such as the MakerBot Replicator II used for our experiments, produces less waste than the estimate given by $\text{waste}(\cdot)$. In particular, the layered printing is able to build the layers along a slope that makes a slightly acute angle with the ground; see the CCTV model in Figure 19 in particular. Additionally, regions covered by the support material are printed at a lower density.

In Table 1, we report both estimated and real material savings, where the latter is obtained by replacing $\text{waste}(\cdot)$ in (5) by the actual material consumption as reported by the software that comes with the 3D printer. Note again that a reported saving close to 1 signals significant gains by the corresponding decomposition.

Assembly of printed parts. We physically assemble the printed parts simply by gluing them along their flat, shared boundaries. Figure 19 shows a few assembled results. Perhaps more elegant way of connecting the parts is via specially designed connectors. In this paper, we focus on the pyramidal decomposition problem. Issues related to structural strength or stability of the glued model, as well as those concerned with connector design, are beyond the scope of our current investigation.
2-pyramidality. We test our algorithm on 2-pyramidality, i.e., pyramidal decomposition to two parts (2PD) only. We selected and created 25 two-dimensional shapes and 30 three-dimensional shapes that are non-pyramidal with each admitting an exact 2PD. The 2PDs must be optimal in part count and can serve as the ground truth. The shapes were selected from on-line search of common 2D shapes, logos, and simple 3D objects. We also hand-crafted a number of 2D shapes which we believe to represent difficult cases, e.g., the house shape in Figure 21. We ran our decomposition algorithm and in all 55 cases, a 2PD was found. A solution we obtain may be different from the ground truth since multiple 2PDs may be possible. The last two rows of Figure 20 show a few interesting cases. All other results can be found in the supplementary material.

Comparison to greedy approach. We compare our algorithm to a greedy scheme on the 25 two-dimensional shapes from the last test. Using the same set of sampled base directions as our algorithm, the greedy scheme searches, along any direction, for the pyramidal part with the largest area. The found part is removed and the process repeats until all the connected parts of the remaining shape pass a pyramidality threshold or become too small. Note that neither our method nor the greedy approach tested is designed for 2-pyramidality only. In only 2 of the 25 cases, a two-way decomposition was found by the greedy scheme; in 16 cases, the greedy scheme ended up with a decomposition with more than 4 parts, evidently showing fragmentation. The top row of Figure 20 shows the suboptimal results obtained by the greedy approach, in contrast to the two-way solutions found by our algorithm (second row).

Comparison to human judgement. To further evaluate our algorithm on more complex inputs, for which ground truth is hard to establish, we conduct a user study. We asked 30 users to manually define the optimal, i.e., minimum-sized, pyramidal decomposition in their view, on 20 shapes with varying complexity. Approximate pyramidal parts are allowed and the users can decide how they want to balance between part count and pyramidality of each part. For ease of user interaction, we only consider 2D shapes for this experiment. Each user worked on 10 shapes, yielding 300 decompositions in total, 15 results per shape. The user study material, including all 20 shapes, can be found in the supplementary material.

We run our decomposition algorithm on the 20 shapes and compare our results to user data. However, to obtain a proper quantitative comparison is not straightforward. For one, user judgements on what is the best pyramidal decomposition can vary greatly. More importantly, it is not clear how to combine material saving and part count to arrive at the most appropriate comparative measure. That being said, as the first measure, we stick with the average saving score (6) as the combination. Figure 22 (left) plots color bars to show the percentages of our wins vs. the users’ wins. Averaged over all the 300 user-return results on the 20 shapes in the study, our algorithm wins over the human users 67.7% of the time.

As a second and more stringent measure, given two decompositions, we say that one is absolutely better than the other if it has a smaller part count as well as a higher level of material saving (5). For each shape, we measure the percentage of times our algorithm does absolutely better than all users and percentage of times some user did absolutely better than our algorithm. These percentages are plotted in Figure 22 (right), showing that on 15 out of the 20 shapes, our algorithm has a higher winning percentage.

The user study results appear to suggest that overall, our algorithm even outperforms human users, at least based on the comparative measures employed. It is interesting to see that for some shapes, almost no user was able to obtain the optimal part count, which our algorithm achieves, perhaps with an imperfect decomposition; see Figure 21 for the tree and the house examples. For the tree, only one out of 15 users decomposed the shape into three parts which is close to optimal. For the house example, none of the users decomposed the shape into two parts as we did, which is optimal. On average, users took 138 seconds to decompose a shape. For the house, the average time is 91 seconds, and 123 seconds for the tree. As we...
can see, the users did spare their effort, but were still unable to
discover the optimal solution in difficult cases. This gives evidence
that pyramidal decomposition is a more challenging task to humans
compared to convex or semantic segmentation.

On the other hand, there are cases for which our algorithm is out-
performed; see the third row of Figure 21. This fish example reveals
that our algorithm may not find long straight cuts over a shape. A
possible reason is that as our final cut is formed by boundaries of
connected blocks, as a result of clustering, these boundaries are
hardly collinear. This also explains the imperfect boundaries we
obtained for the tree example in the first row. Overall, it can be
observed by comparing the two sets of results, that while our algo-
thesis tends to provide results with fewer number of parts but per-
haps with less material savings, human users tend to cut the shape
into more pieces with higher pyramidality.

6 Conclusion, limitation, and future work

Pyramidality is a relatively new and unexplored shape property.
Pyramidal shapes are 2.5D and they are highly relevant in applica-
tions such as 3D printing, molding, and casting. At the same time,
pyramidal decomposition is a provably hard problem in general and
to date, there are no known construction algorithms, approximate or
exact. Hence, we believe that a study of pyramidal decomposition
is both of practical and theoretical interests.

In this paper, we make a preliminary attempt and introduce the first
construction algorithm for approximate pyramidal decomposition.
The key insight is that the decomposition problem can be solved
efficiently by turning it into a exact set cover problem. The bottom-
up clustering-based approach is both general and versatile, allowing
the same construction algorithm to process 2D and 3D shapes with
most geometric and topological varieties. The performance of our
algorithm is demonstrated and evaluated on numerous shapes and
through several comparative studies. Naturally, as a first attempt,
our algorithm still leaves much room for improvement.

Shape semantics. Our current work focuses on material sav-
ing and obtaining as few approximate pyramidal parts as possi-
bly. Hence, the decomposition results obtained do not quite fol-
low shape semantics, e.g., finding cuts over concave areas or those
respecting shape symmetry. This is by design. However, our solu-
tion pipeline allows the incorporation of shape semantics quite eas-
ily. We can introduce biases towards symmetric cuts or cuts over
concave areas when sampling base orientations. We can also add
a semantics term in the capacity definition (1) which gives higher
weight to the semantic cuts above. Note however that consideration
of semantics may set conflicting goals as opposed to obtaining a
minimal decomposition. Figure 21 contains examples showing that
symmetric decompositions may lead to suboptimal part counts.

Base selection. Sampling a pre-determined set of base orienta-
tions, rather than deriving the bases or their orientations as part of
an optimization, is obviously undesirable. The final decomposition
is limited by the choice of the set of base orientations, possibly
leading to sub-optimal results. Furthermore, our heuristic defini-
tion of base capacity gives preferences to boundary bases and in-
terior bases which straddle shape boundaries; this may also lead to
sub-optimal bases in the final decomposition. These issues could be
remedied during post processing, i.e., via more advanced boundary
refinement, which must then resort to other heuristics.

Parallelizability. Currently, our algorithm is not implemented in
the most efficient manner; it runs on single-threaded CPUs, requir-
ing minutes to process a shape. However, the most time-consuming
parts of the algorithm, cell generation and candidate pyramidal part
computation, which together take up about 90% of the execution
time, are both highly parallelizable. Cell generation mainly in-
volves base voting, which can be executed independently for each
base direction. Searching for candidate pyramidal parts involves lo-
cal graph traversal and can be parallelized at each node. We expect
a GPU implementation to incur a significant speed-up.

Optimization objectives. For approximate pyramidal decompo-
sition, we still lack a clear optimality criterion, and as such, our
construction algorithm is primarily a forward search+rank scheme
and not an optimization. The average material saving score (6) is
sensible, but seems difficult to optimize directly. It would be ideal
to have a clear pyramidality and part count trade-off in an optimiza-
tion objective as well as in the decomposition results. This may be
possible if the pyramidality score could be factored in earlier in our
progressive clustering, e.g., for cell or block construction. We leave
that for future work. Currently, we have opted to enumerate a set
of solutions, which seems appropriate in the absence of a consen-
sus objective function; it also allows a user to choose application-
specific solutions from a small set of good alternatives.

Further limitations. Our current algorithm does not offer any
theoretical guarantees, such as approximate bounds in terms of
number of parts in a decomposition. If a shape has a finite exact
pyramidal decomposition, our algorithm is not guaranteed to find
it. Figure 23 shows two cases where the best results our algorithm
is able to obtain are quite far from the optimal. Some of the issues
can be attributed to boundary artifacts, lack of proper base direc-
tions, or the limits of clustering. Although the clustering approach
offers versatile options in the affinity measure employed, solutions
to clustering can be sub-optimal. The selection of candidate pyra-
midal parts depends on the generated blocks and block boundary
artifacts can cause the pyramidality score of potentially good can-
didates to fall below the threshold. Refining block boundaries to
maximize the pyramidality of the candidates can be an effective
remedy; we leave that for future work.

Future work. Aside from addressing the limitations mentioned
above, we would like to further analyze the behavior of our algo-
rithm as we increase the number of uniformly sampled base di-
rections towards infinity. It would be desirable to prove a certain
form of convergence. On the technical front, boundary refinement
for decomposition in 3D still leaves room for improvement. As in
practice, parts that are not exactly pyramidal can be printed with-
out waste material, it would be useful to incorporate slope angles
allowed by a 3D printer with no support material into our waste
measure. While our work offers a practical solution to pyramidal
decomposition, it is still of interest to seek algorithms with theoreti-
cal guarantees. For example, it is desirable to settle the complexity
of optimal exact pyramidal decomposition for simple polygons. Fi-
nally, we would like to look into the interesting problem of “pyra-
midalization”, which seeks a minimal way to modify a given shape
to maximally reduce the size of its pyramidal decomposition, so
that the shape becomes more printable, moldable, and castable.
Acknowledgements

We first thank the anonymous reviewers for their valuable comments and suggestions. We are grateful to Joseph Mitchell for his comments on our work and Ibraheem Alhashim for sharing his code and providing suggestions to our work. Thanks also go to Hadar Elor for careful proofreading of early drafts of the paper and Ligang Liu for some initial discussions. This work is supported in part by grants from Natural Sciences and Engineering Research Council of Canada (No. 611370), China Scholarship Council, the Israeli Ministry of Science, and the Israel Science Foundation.

References


