Abstract

Volume compaction is a geometric problem that aims to reduce the volume of a polyhedron via shape transform. Compactable structures are easier to transport and in some cases easier to manufacture, therefore, they are commonly found in our daily life (e.g., collapsible containers) and advanced technology industries (e.g., the recent launch of 60 Starlink satellites compacted in a single rocket by SpaceX). It is known in the literature that finding a universal solution to compact an arbitrary 3D shape is computationally challenging. Previous approaches showed that stripifying mesh surface can lead to optimal compaction, but the resulting structures were often impractical. In this paper, we propose an algorithm that cuts the 3D orthogonal polyhedron, tessellated by thick square panels, into a tree structure that can be transformed into compact piles by folding and stacking. We call this process tree stacking. Our research found that it is possible to decompose the problem into a pipeline of several solvable local optimizations. We also provide an efficient algorithm to check if the solution exists by avoiding the computational bottleneck of the pipeline. Our results show that tree stacking can efficiently generate stackable structures that have better folding accuracy and similar compactness comparing to the most compact stacking using strips.

CCS Concepts

• General and reference → Design; • Computing methodologies → Mesh geometry models; Volumetric models;

1. Introduction

Physical shape compaction, to be distinguished with compression that reduces the storage space of digital geometry data, is the problem of reducing the volume of a (rigid) geometry by rearrange its spacial configuration [LZ16]. Such compaction has often been used in applications, such as furniture designs and commercial product packaging. In the future, compaction will become even more important as human activities are growingly extended to microscopic worlds and deep spaces, from the designing of nano-scale robots to encapsulating a large space construction into a rocket. The idea of compaction also sparks the wildest imagination in fictional worlds such as the Hoi-Poi capsules in Dragonball and superheroes suites (e.g., Iron-man and Ant-man) in recent Marvel movies.

Shape compaction also has applications in 3D printing oversized shape. Several recent works approached this via computational methods called segment and pack [CZL+15, VGB+14, LBRM12]. Given a 3D shape, these methods dissect the solid volume or its thick surface into pieces, which are then tightly packed for printing. Printed components can be assembled to produce the original shape. However, segmented objects are not suitable for applications that require integrated functionality such as electronic circuit.

In this paper, we explore an alternative universal method in creating compact structures using a special type of folding motion called stacking. Such structure has a stacked state that fits in a user defined space for fabrication and transportation. An advantage over the segmentation-based compaction is that this new structure consists of a single connected component thus avoids the aforementioned drawbacks for segmented pieces. An example is shown in Figure 1 where a fish model of 854 faces is compacted into a volume that is 250 times smaller using the proposed method in this paper.

Our approach to shape compaction is inspired by the polyhedral unfolding problem in computational origami. Due to recent advances in unfolding 3D shapes [SWK08, ALC+13, FTD+14, LKP+14], it is now possible to design a complex foldable structure, with one degree
We introduce a set of stacking rules, following the concept of the unfolding to generate the stacking. Cutting the mesh edges of the dual-graph unfolded mesh edge. The mesh is then a dual edge v

...to the 3D shape than tree-structured stacking, given small noises in the motion control of the folding hinges. This paper follows the same design principle but is aimed at creating a tree structure from a given 3D shape that can be compactly stacked. Our results show that tree stacking can efficiently generate stackable structures for many complex voxelized models (orthogonal polyhedra), and these structures have better folding accuracy and similar compactness compared to the most compact stacking using strips.

2. Preliminaries

In this section, we discuss the definition of the stacking problem. We introduce a set of stacking rules, following the concept of the polyhedral unfolding problem. We also discuss the criteria for the most compact and optimal stacking considering the folding accuracy of the structure besides the volume. We show that finding the optimal stacking is computationally unfeasible, and thus heuristics and approximations are needed to solve this problem efficiently.

2.1. Stackable Structure

In order to stack a polyhedral surface with non-negligible thickness into aligned columns, we assume that the input is an orthogonal polyhedron that is tessellated with identical square panels of thickness τ (see Figure 2). It is worth noting that, the proposed concept is not limited to orthogonal polyhedra but applicable to all mesh surfaces covered by identical equilateral polygons. Let us abstract the stacking structure only using the topology information of the polyhedron. We define a dual-graph \( G = \langle E, V \rangle \) of the mesh, where each square face becomes a vertex \( v \in V \), vertices are connected by a dual edge \( e \in E \) if their corresponding faces in the mesh shares a mesh edge. The mesh is then unfolded by generating a spanning tree \( T \) of the dual-graph \( G \) with an arbitrary reference \( v_i \in V \) as root and cutting the mesh edges \( h \) corresponds to \( e_i \in \{ e \mid \forall e \in G \land e \notin T \} \). We call such structure an unfolding. We use the tree-structured unfolding to generate the stacking.

2.2. Stacking Rules

The tree structure can be viewed as a multi-arm robot: the square faces are rigid linkages rotating around mesh edge \( h \) as a hinge. We define a configuration \( Q = \{ q_i \} \) of such robot as the angle \( q_i \) of mesh edge \( h_i \) between every child node \( v_i \) and parent node \( v_{i-1} \), depending on their topological distance in \( T \) from a reference node \( v_i \). In the original 3D quad mesh \( q \in \{ -\pi, \pi, 0 \} \) as the faces are either perpendicular to or coplanar with the neighboring faces. We define a stacking as the following tuple \( < G, T, Q > \) for a given 3D model. Following the notation in Section 2.1, we call \( T \) the tree unfolding, and call \( Q \) the stacked state which is a particular configuration that satisfies the following rules:

1. For all \( q_i \in Q \), we have \( q_i \in \{ -\pi, \pi, 0 \} \).
2. If \( q_1 = -\pi \), we call it a valley fold, and the front side (facing the exterior of 3D model) of \( v_1 \) and \( v_{i-1} \) must touch.
3. If \( q_i = \pi \), we call it a mountain fold, and the back side (facing the interior of 3D model) of \( v_i \) and \( v_{i-1} \) must touch.
4. If \( q_i = 0 \), we call it a side fold, and, by the nature of hinge rotation, the front or back side of \( v_i \) and \( v_{i-1} \) must face the same direction in the stacked state.

Note that, we can ignore the thickness of the faces in these rules, because the desired thicknesses can be added in a post-processing procedure using the design of a uniform hinge.

Given these rules, we aim to compute a compact stacking of a given 3D model, which involves tuning \( T \) and \( Q \). The problem of tuning \( T \), in a sense, is similar to the polyhedral unfolding problem [Sch97], of which the solution space is exponential and the most effective solvers often resort to genetic algorithms [TWS11, XKKL16]. Apparently, the problem space of tuning \( Q \) is also exponentially large, as searching \( Q = \{ q_1, q_2, ..., q_{n-1} \} \) in a 3\(^n\) search space, given the number of square faces \( n \) in the mesh.

To ensure a tree structure being stackable, we must also ensure that the motion between the 3D state \( Q_{3D} \) to stacked state \( Q \) is collision free. Since this motion planning problem is not the main focus of this paper, we will give a brief discussion in Section 5.3.

2.3. Kinematics of Stacking

Let \( \vec{p} = [x, y, z, o, f]^T \) be the position, orientation and facing of \( v \) in the stacked state, \( [x, y, z]^T \in \mathbb{R}^3 \) as we can lay all the faces in a 3D grid space. Apparently, each face can only face up or down, which can be denoted as \( f \in \{ -1, 1 \} \). We also have the hinge information \( l = \{ 0, \frac{\pi}{2}, \frac{3\pi}{2} \} \) in the 3D model, denoting which of the four sides that the child link connects to the parent link.
Following the rules and definitions, given a parent state \( p_{i-1} \), we can calculate the child state \( p_{i} \) using the configuration \( q \) and linkage \( l \) using the following kinematics equations:

\[
\begin{align*}
x_i &= x_{i-1} + \left(1 - \frac{q_i}{\pi}\right) \cdot \cos(o_{i-1} + l) \\
y_i &= y_{i-1} + \left(1 - \frac{q_i}{\pi}\right) \cdot \sin(o_{i-1} + l) \\
z_i &= z_{i-1} + \frac{q_i}{\pi} f_{i-1} \\
o_i &= o_{i-1} + f_{i-1} \cdot l + \pi(1 - \left(\frac{q_i}{\pi}\right)^2) \\
f_i &= (1 - 2\left(\frac{q_i}{\pi}\right)^2) \cdot f_{i-1}
\end{align*}
\]

subject to the constraint that no any pair of face panels can be in the same place in the stacked state to avoid self-collision.

\[ <x_i,y_i,z_i> \neq <x_j,y_j,z_j> \quad \text{for all } i \in 1..n, j \in 1..n \wedge i \neq j \]

### 2.4. The Most Compact and Optimal Stacking

There are many ways to define compactness, thus the optimal stacking may vary depending on the application. One of the goals is to reduce the dimension of the structure so it is as compact as possible. Additionally, we expect the stacked model to fit in a bounding box \( BBox \) of size \( W \times D \times H \), which represents the width, depth and height respectively.

One of compactness measurement is the volume ratio (VR),

\[
VR = \frac{V_{BBox}}{V_{BBox}}
\]

which is simply the volume of the bounding box of the stacking divided by the volume of the bounding box of the original mesh. However, this measurement does not reflect the overall dimension of compactness. Under this measurement, a one pile stacking may have the exact volume ratio as the original shape if the shape is a rectangular panel like shown in Figure 4.

Another widely used compactness measure is the sum of three dimensions, the compactness ratio (CR), defined as:

\[
CR = \frac{|W| + |D| + |H|}{|W| + |D| + |H|}
\]

for the original dimension \( W, D, H \) and the compacted dimension \( W, D, H \).

As both measurements have flaws in quantifying compactness, we consider the following desired properties for an optimal stacking.

1. The optimal stacking should have the number of user-defined piles and the pile placement can fit in a minimum \( W \times D \) rectangle.
2. The optimal stacking should be a tree structure that minimizes the chain components (to be defined below).
3. All piles should have similar heights in the optimal stacking.

Considering each quad face as a rigid unit linked to other units with a rotational axis, the stackable structure can be treated as a self-folding machine transforming from the stacked state to the 3D shape. This raises the issues of accuracy [VW86] and repeatability [SCY02] in such robotic systems, i.e. with noises in the motion controls, whether it can be accurately folded to the desired 3D shape, or can be folded multiple times while still achieve the same shape. Both issues are highly related to the structure of the stacking. Figure 3 shows such inaccuracy by adding small Gaussian noises to the control of folding links. The chain-structured stacking suffers large gaps and self-intersections in the folded shape due to the imperfect control. This is because as the quad face is located further down in the link chain from the base link, it gains cumulative errors from the previous links in the forward kinematics [VW86]. In order to have structures that are suitable for self-folding with imperfect motion control, we can minimize the chain components in the structure.

![Figure 3: The folding inaccuracy of (a) chain-structured stacking and (b) tree-structured stacking of the Table-154 model with small Gaussian noises (zero means and 0.01 rad standard deviation) added to the target hinges angles.](image)

For tree-structured stacking, a subset of the units linking to each other all with two neighboring nodes forms a chain component. In order to shorten the length of the chain components, we want the stackable structure to have more high-degree nodes (HDN) with three or four neighbors. In other words, more HDNs are desired to achieve better folding accuracy. And, ensuring similar heights for all piles reduces the maximum height and limit the number of piles in the stacking, and therefore improve the overall compaction.

These properties are hard to formulate mathematically and often times they contradict with each other. In this paper, we propose an algorithm to generate a stacking that simultaneously maximizes the folding accuracy and ensures the compactness of the stacking.

### 2.5. Two Special Cases

In this section, we discuss two special cases of the stacking: Hamiltonian stacking and polyhedral net compaction/stacking. Both cases satisfy all the stacking rules, however, we will show that they are far from optimal in terms compactness or folding accuracy.

#### 2.5.1. Hamiltonian Stacking

Shown in Figure 4, if it is possible to find a Hamiltonian path in the dual-graph of \( P \) (i.e. strip unfolding [MS04]), we can easily stack the path into a pile. Because an orthogonal polyhedron is tessellated by quads, its dual is a 4-connected planar graph that must permit Hamiltonian paths and cycles [Tut56]. This gives us the theoretical guarantee that all orthogonal polyhedra, such as our voxelized meshes, can always be stacked into one or even multiple piles in time linear to the number of panels [XWH*17]. However,
because the Hamiltonian stacking is built on strip or chain structure and lacks HDNs, it is not a suitable structure for accurate folding. In addition, transforming this structure from a stacked state to the target state, the torque needed to rotate a hinge near the start of the strip is much larger than rotating a hinge near the end of a strip.

Figure 4: (left) Quad faces in a 4-by-4 panel model linked as a Hamiltonian path. (right) A Hamiltonian stacking of the panel model.

2.5.2. Polyhedral Net Compaction

Another way to find a stacking of a given polyhedron is by cutting along a subset of edges of the polyhedron and flatten the polyhedron into a planar shape. If this planar shape has no self-overlapping, we call it a polyhedral net [Sch97, TWS’11, XKKL16]. An example of the polyhedral net of the Mountain model with 34 faces is shown in Figure 5(b). Apparently, this net is a stacking, as it satisfies all the rules defined in Section 2.2. In addition, most nets are tree-like structures that provide better folding accuracy than strip structures. However, it is not very compact as it occupies a large area on the 2D plane.

Given a polyhedral net, it is possible to further compact the dimension of the net by alternating the hinges folding angles in one direction as shown in Figure 5(c). We call such operation net compaction. Unfortunately, net compaction only compacts the unfolding in one dimension, and the resulted stacking is long and narrow. We cannot further stack the model in the direction perpendicular to the initial compaction as it violates the rules of stacking introduced in Section 2.2.

Figure 5: Net compaction: compacting the polyhedral net of the Mountain-34 model.

In addition to the issues with compactness, computing a net by cutting edges (i.e. edge unfolding), even for orthogonal polyhedra [AD11], requires time exponential to the number of panels. It is also worth mentioning that the stackable structure does not need to be flat as a polyhedral net. This implies the stacking problem, while can be regarded as a subset of the unfolding problem, may have a larger solution space than edge unfolding. However, this does not mean stacking is an easier problem as our goal it not just finding valid stackings, but the optimal ones.

3. Tree-structured Stacking

While both Hamiltonian stacking and net compaction provide promising properties, they lack folding accuracy or compactness. In this section, we discuss the challenges and solutions in designing a compact tree-structured stacking.

The design goal of shorter chain components often contradicts with the need for compactness. Let \( T \) be a spanning tree of the dual-graph of a 3D model. We observe that increasing the number of HDN in \( T \) will increase the number of piles thus decrease compactness. Take the simple case in Figure 6 for example. Given a surface with five faces, there exist three different topologies of \( T \):

(a) the unfolding is topologically equivalent to a chain, and the most compact stacking has only one pile.
(b) the unfolding has one face with three neighboring faces, denoted as a \( D_3 \) face, in which case the most compact stacking has two piles.
(c) the unfolding has one face with four neighboring faces, denoted as a \( D_4 \) face, in which case the most compact stacking has three piles.

Figure 6: Different topologies of unfolded tree structure and their most compact stacked state.

Apparently, the number of piles in the stacked state is related to the number of \( D_3 \) and \( D_4 \) faces, as each panel only has two sides to accommodate its neighboring faces.

3.1. Compacting a Given Tree

Let us first assume that a spanning tree \( T \) of the dual-graph of a 3D model is given and our objective is to find a \( Q \) that can fold \( T \) into the most compact stacking. We briefly discuss a branch-and-bound algorithm that can be used to find the most compact stacking for \( T \) with the minimum number of piles in the stacking. We will remove this constraint later in Section 4 to discuss an algorithm to determine a tree that admits the most compact stacking among all possible trees of a given polyhedron.

Our goal is to grow a search tree so that a path of length \( k \) from the root to a leaf in this search tree encodes the folding motion that
stacks $k$ connected panels of $T_s$ into one or multiple piles. The root of the search tree contains only the root node of $T_s$, and each edge in the search tree describes how the child node is stacked on, below or beside the current panel. When the search tree expands from one of its leaves, a face $v_i$ of $T_s$ that is not yet stacked (along this particular branch) will be folded from its parent $v_{i-1}$ with either valley, mountain or side fold, neither of these situations will result in less number of piles. Therefore, every time we expand the search tree, the number of piles for stacked faces along the expanded branch cannot decrease.

In addition, based on the observation regarding $D_3$ and $D_4$ faces above, we can draw the following conclusion: $T_s$ can be stacked to less than $P$ piles

$$P \leq 1 + n_{D3} + 2n_{D4}, \quad (9)$$

where $n_{D3}$ and $n_{D4}$ are the number of $D3$ and $D4$ faces in $T_s$, respectively.

Because Equation 9 gives us a conservative estimation on the number of piles in the optimal state, we can bound the search tree by limiting the current number of piles the conservative estimation in Equation 9 or the maximum number of piles in all sibling branches of the same depth, as a stacking exceeds the upper bound is guaranteed sub-optimal, thus the search does not need to continue in this branch.

![Figure 7: The stacking of polyhedral net in Figure 5 (b) by minimizing the number of piles. Using a branch-and-bound search, only 2525 out of 333 search nodes are visited.](image)

This approach can find an optimal solution for small models as shown in Figure 7. However, it remains computationally expensive for larger models with merely 100 or more faces. Moreover, the stacking is not optimally compact and supporting structures must be used for 3D printing such structure. In short, it is unlikely to find the most compact stackable structure of target shapes without considering the design of $T_s$.

### 4. Finding Optimal Stackable Tree

From what we learned in the previous section, we need to optimize $Q$ (stack configurations) and $T$ (stackable structure) together in order to obtain the optimal stacking for a given 3D model. Therefore, this problem space is $4^8 \times 3^9$. From the existing literature, optimizing $T$ for the polyhedral net unfolding problem is already pushing the problem space is $4^8$. Finding Optimal Stackable Tree

To obtain the optimal stacking for a given 3D model. Therefore, this problem space is $4^8 \times 3^9$. From the existing literature, optimizing $T$ for the polyhedral net unfolding problem is already pushing the problem space is $4^8$. Finding Optimal Stackable Tree

From the existing literature, optimizing $T$ for the polyhedral net unfolding problem is already pushing the problem space is $4^8$. Finding Optimal Stackable Tree

We segment the original model into multiple components by computing a partition of its dual-graph $G$ into $k$ subgraphs $G_i$. For the purpose of illustration, we use the 9-by-9 planar quad mesh shown in Figure 8 as an example. If we ensure that at least one Hamiltonian path $P$ exist in $G_i$, we can stack the corresponding mesh patch into a single pile Figure 8(c). The height of a pile, $H = |V(P)| = |V(G_i)|$ where $V$ is the set of vertices in the topology. However, if we want the pile to be reconfigurable, we would prefer that there exist multiple $P$s in $G_i$. This means we want to maximize the inner connectivity, i.e. the number of edges $|E(G_i)|$, within $G_i$ to increase the possibility of finding multiple $P$s. Also, as we have mentioned in Section 2.4, we also prefer each subgraph to have a similar number of vertices, so that we all the piles have similar heights. This partitioning problem can be treated as a balanced graph partitioning problem [AR04] where a graph is partitioned into $k$ components of roughly equal size while minimizing the capacity of the edges between different components of the cut. In our case, all the edges have weight 1, thus we will minimize the number of edges in the cut. There are many existing algorithms to solve this problem in a relatively efficient way. We use generalized Fiduccia-Mattheyses algorithm [San89] in our implementation which is efficient enough to handle the scale of the stacking problem, with $|V(G)|$ being few hundreds to thousands, and can be directly used when $k > 2$.

### 4.2. Pile Placement

Apparently, if $k = 1$, the algorithm degenerates to Hamiltonian stacking. If $k = 2$, the placement is trivial, as we simply put he
we seek to find the optimal $G_v$ where each subgraph $G_v$ becomes a node $v_{hyper}$ in $G_{hyper}$. If the number of edges in $G$ between two subgraphs $|E(G_i, G_j)| > 0$, then $v_{hyper}$ and $v_{hyper}$ is linked by an edge $e_{hyper}$, and the weight of the edge $W(e_{hyper}) = |E(G_i, G_j)|$. We then place $G_{hyper}$ onto the 2D grid using the following way: each node $v_{hyper}$ occupies a single grid cell $g$, and we construct $G_{hyper}$ that $V(G_{hyper}) = V(G_{hyper})$ and $e_{hyper} \in G_{hyper}$ iff $v_{hyper}$ is adjacent to $v_{hyper}$ on the grid. Finally, we seek to find the optimal $G_{hyper}$ by

$$G_{hyper}' = \arg \max_{v_{hyper} \in G} \sum W(e_{hyper})$$

As shown in the example for a planar quad mesh in Figure 10, the placement in (b) is optimal. As we do not expect $k$ to be large, a dynamic-programming solver designed for subset sum [SE94] can efficiently solve this optimization.

![Figure 10: (a) The hypergraph of partitioned panel model shown in Figure 8. (b)-(d), possible pile placement as to place the hypergraph into 2-by-2 square grid.](image)

4.3. Hamiltonian Path Optimization

So far, we have segmented the mesh into $k$ components and each component finds an arbitrary Hamiltonian path $P_v$ so that each component sticks into a pile. Each pile also corresponds a node $v_{hyper}$ in $G_{hyper}$. If we set $T_v = \cup P_v$ and assign $Q_v$ in the way similar to how we generate Hamiltonian stacking, we have a stacking of high compaction. Apparently, this unfolding is not a single component nor in a tree structure. We need to optimize $P_v$ locally in each pile $G_v$ and find as many connections between the piles as possible.

Select an arbitrary $v_{hyper}$ as a reference node in $G_{hyper}$, for any child node $v_{hyper}$ that is connected to its parent node $v_{hyper}$, depending on their topological distance from $v_{hyper}$ in $G_{hyper}$. We recursively update the $P_v$ (corresponds to $v_{hyper}$) based on $P_v$. Recall that in $G_{hyper}$, we have $W(e_{hyper}) > 0$, so that there must be some $v_i' \in P_v$ may potentially be connected to a node $v_j'$ in $P_v$. Also, as we mentioned the Graph Partitioning step that $G_v$ is generated in a way that maximizes the $E(G_i)$ to increase the number of $P_v$. Let $v_i'$ be the $i$th node in $P_v$, we place the stacked pile of $P_v$ next $P_v'$ with $v_i'$ and $v_i'$ being the bottom. In the workspace, $p_i'$ is the position of the panel of $v_i'$. We also ensure that the facing direction $f_i = f_j$, i.e., both the bottom panels have front side facing the same direction. According to the kinematics (Section 2.3), we have $z_i' = z_1'$ for all $z$ as well as $f_j = f_j'$, which means the $j$th element in $P_v$ has the same height $z$ and side facing $f$ in the workspace for both $P_v$ and $P_v'$. We define the following function:

$$f_{match}(x) = \begin{cases} 1 & \text{if the edge } v_i', v_i' \text{ exists in } G \\ 0 & \text{otherwise} \end{cases}$$

$f_{match}(x)$ is called the matching function. The idea is to check every pair of nodes $v_i'$ and $v_i'$ in $P_v'$ and $P_v'$ that are on the same height $x$, if they were originally neighbors in the dual-graph $G_v$, then they are matching nodes. Then we can update $P_v'$ using

$$P_v' = \arg \max_{P_v \in G_v} \sum_{x=1}^{\min(H(P_v'), H(P_v'))} f_{match}(x) \cdot f_{geo \_match}(x)$$

Where $f_{geo \_match}(x)$ is another matching function we will discuss in the next subsection. We update all $P_v$ recursively using Equation 12, except for $P_v$ in $v_{hyper}$, which we use a randomly generated Hamiltonian path. We also record all the $x$ that satisfy $f_{match}(x) = 1$ and put in the sorted vector $\vec{x}$ and ensure $x_i < x_i+1$ (to be used in the next step). We are using brute force search to solve Equation 12 with branch-and-bound if $|V(P_v')|$ is small (a worst case exponential complexity).
to handle this situation, we have

\[ f_{\text{match}}(x) = \begin{cases} 1 & \text{if condition 2 is satisfied,} \\ 0 & \text{otherwise} \end{cases} \]

However, even considering \( f_{\text{geo,match}}(x) \) in our equation, we still cannot ensure the \( v_i' \) and \( v_j' \) are geometrically matching. We have Figure 9 to illustrate the situation. To handle this situation, we need to rotate \( v_i' \) in the pile, i.e., change the orientation \( \alpha_i' \) (defined in Section 2.3) to accommodate geometrical mismatches. If \( |x| = 1 \), meaning there is only one matching component, we can simply rotate the entire pile along with \( v_i' \). However, if \( |x| > 1 \), we have to break \( P' \), and \( v_i' \) has to be detached from \( v_{i+1}' \) or \( v_{i-1}' \). We denote such broken edge as \( e_{\text{break}}'. \) \( v_i' \) can then be attached to \( v_j' \) by an edge \( e_{\text{match}}' \). Figure 9(d) illustrates such \( e_{\text{match}}' \) as a “bridge” connecting the neighboring piles. Finally, we have

\[ T_s = \bigcup_{i} P_i + \bigcup E_{\text{match}}' - \bigcup E_{\text{break}}' \quad \text{for all } i \in \{1..k\} \]

where \( E_{\text{match}}' \) and \( E_{\text{break}}' \) are the sets of all matching and broken edges in pile \( P_i \). Finally, \( T_s \) can be trivially derived in linear time.

### 4.5. Guaranteed Single Component Conditions

Recall that in the Hamiltonian Path Optimization step, we found the set of matching nodes \( x \) between adjacent piles. However, \( |x| > 0 \) is not guaranteed because there might be no \( P' \) that satisfies \( f_{\text{match}}(x) > 0 \). In this case, \( T_s \) will be a forest rather than a single connected component. In many applications, if the large 3D shape is broken into multiple parts for manufacture in compact space, assembling a large number of separate parts back to the 3D shape is often difficult due to the connectivity information being lost. Therefore, \( T_s \) being a single connected component is desirable.

Meanwhile, the Hamiltonian Optimization is non-trivial as it requires to enumerate through all possible Hamiltonian paths to find the one with the optimal matching. In practice, while we only need to find Hamiltonian paths within each pile, the number of nodes in one pile can go over 30 for complex models, which makes the optimization difficult. However, there are heuristic based algorithms to find Hamiltonian paths or cycles such as the public available yet state-of-art TSP solver Concorde [ABCC06]. We propose the following theorem that ensures the single connected component requirement and accelerate the optimization:

**Theorem:** \( T_s \) is guaranteed a single connected component if

1. There exists a Graph partitioning \( \{G_i^s\} \) that satisfies \( \forall G_i^s \), there exists at least one Hamiltonian cycle in \( G_i^s \)
2. There exists a pile placement that satisfies \( f_{\text{geo,match}}(x) = 1 \) for all piles.

**Proof:** If condition 2 is satisfied, we only need to ensure that there is at least one \( v_i' \) in \( P' \) that satisfies in \( f_{\text{match}}(x) > 0 \). Using condition 1, because there exists a Hamiltonian cycle in \( G_i^s \), we can always break the Hamiltonian cycle into \( P' \) in a location so that \( v_i' \) becomes the same height in \( P' \) as the matching \( v_i' \) in \( P \) regardless the value of height. Therefore, there is always a least one “bridge” found between \( P \) and \( P' \), and \( T_s \) is a single connected component.

We can efficiently generate partitioning and pile placement ensure we can get \( T_s \) a single connected component, without actually going into Hamiltonian Optimization which is the most computationally expensive part of the pipeline. As previously discussed in Section 2.5.1, Hamiltonian paths and cycles always exit in 4-connected planar graph [Tut56]. For tree-stacking, although it requires a Hamilto-
nian path found in each pile, the guarantee is still satisfied, as in the worst case it can degenerate to Hamiltonian stacking.

5. Experimental Results

We implemented the proposed method in single-threaded C++. We show select models and stackings of our results in Figure 11. The 3D polyhedral surfaces are shown alongside the compact stacking and the hinges are shown in grey. The number in the model name indicates the number of thick panels in the mesh.

5.1. Optimal Stacking

Assuming the panels considered in this section have size $1 \times 1 \times \tau$, where $\tau$ ranges from 0.01 to 0.2. We can calculate the bounding boxes, $BBox_m$ and $BBox_s$, which are the dimension of the bounding box of the 3D shape ($W_m \times D_m \times H_m$) and stacked shape ($W_s \times D_s \times H_s$) as defined in Section 2.4, respectively. Our experiments find that the proposed method can generate stackable structures with significantly better folding accuracy and a near-optimal compact ratio comparing to Hamiltonian stacking. We summarize the results in Figures 12 to 17 and analyze them in the aspects of compactness, folding accuracy and computational cost.

5.1.1. Folding Accuracy

We are interested in the topology of the stackable structure which is highly related to the folding accuracy, as a long chain structure suffers cumulative control errors in the forward kinematics. With more faces that are linked to three or four other faces, i.e. the number high-degree nodes (HND as defined in Section 2.4), our algorithm maximizes the number of HDNs which exponentially reduce the length of chain components in the structure. The globally optimal solution (i.e. compact stacking with maximum possible HND) still requires exhausting the search space (i.e. checking all the Hamiltonian paths in each pile). However, the dual-graph is considered locally optimal every time a search reaches the end of the search tree. In Figure 11, the HDNs are shown as “bridges” between different partitions in the 3D shape or piles in the stacked shape. Figure 14 shows the average length of the chain components in the stackable structure, normalized by the number of faces in the model. A Hamiltonian stacking with a single long chain component has the normalized length 1.0, while our tree stackings have consistently shorter chain components regardless of the geometry of the model. The average length of the chain components is less than 40 for the Fish-854 model in the compact stacking as small as 5 piles. Figure 15 shows the folding position errors based on the simulation illustrated in Figure 3.
5.1.2. Compactness

In this section, we show that, by gaining significant folding accuracy, the proposed method only sacrifices a small amount of compactness measured via the volume ratio (VR) and compactness ratio (CR) defined as Equation (7) and (8). Compare to the most compact structures generated by the Hamiltonian stacking, the structures created by tree stacking is near optimal in both VR and CR. For VR, the single-pile stacking always has the best ratio because the panels are tightly compressed, the VR is the limit of the model can be compacted. As we have mentioned in Section 2.4, it is desired to have piles with similar heights as fewer spaces are "wasted". However, the stacking and pile placements both contribute to the compactness measurement. Our algorithm can generate different stacking with a placement that satisfies user requirement. As shown in Figure 11(d) and 9(d), the former 3 by 1 placement may have better VR, but the latter can be fit into a 2 by 2 platform.

![Figure 15: The folding position errors simulated by adding small Gaussian noise (zero means and 0.01 rad standard deviation) to the hinges angles, averaged over the number of quad faces in the model.](image)

5.2. Fabrication

The running time generally increases as the complexity of the model. However, an exception is reported for Fish-854 that consumes much more time than Table-1964 even though the number of faces is much smaller. The reason is that Fish-854 has a more complex geometry profile, i.e. more non-flat components, which makes the Hamiltonian matches much difficult to find. Geometry profile may lead to the local optimal or failure of the algorithm. If the shape is open and there are quad faces on the boundary with the only edge connected to the shape, in which case the Humiliation path has to start or end from this face which leads to limited solution space. Examples can be found for simple shapes as in Figure 6. Fortunately, such geometries are uncommon in practice for large models. In general, stacking with more piles is more expensive to compute as all the piles must have at least one matching face to the parent pile or the entire algorithm must restart.

![Figure 17: Running time (in seconds) for computing various stackings of the models tested.](image)

5.1.3. Computational Cost

Our method can generate stackable structures within minutes on a regular laptop computer (e.g., we use Macbook Pro with a 2.9 GHz Intel Core i7 CPU with 16GB Memory). As shown in Figure 17, generating single-pile stacking is significantly faster as no matching process is done.

![Figure 18: Mountain-34 model in 2-pile tree stacking fabricated using 3D printing, comparing with the size of US letter paper (215.9 by 279.4 mm). Four additional hinges (shown in mid-bottom picture) are added in the final step to reinforce the structure.](image)
In this section, we report our evaluation of the stackable structures created by the proposed algorithm via 3D printing. Shown in Figure 18, we fabricated using 3D printer the components for the Mountain-34 model in 2-pile tree stacking and assembled manually via folding. The design consists of 34 square panels of size 50 × 50 × 6.25 mm including the hooks for holding the 33 hinges. Only the frames of the square panel were 3D printed for speeding up the fabrication and better illustrating the 3D structure. The structure can be easily transformed between the original 15 × 15 × 10 cm³ stiff 3D structure and 10 × 12.5 × 5 cm³ stacked state.

5.3. Finding Continuous Folding Motions

While planning the motion for articulated structure with many degrees of freedom is non-trivial, many algorithms in this area are still under development and beyond the scope of this paper. We use a discrete domain sampling based planner [XL15] to find such motion to show it is physically realizable. We show the folding motions of the Donut-32 (shown in Figure 19) and Panel-81 model (shown in Figure 20) from their stacked state to the 3D shapes respectively. The running time for the planner to find a continuous folding path is 30 seconds for the Donut-32 model and 90 seconds for the Panel-81 model.

6. Conclusion and Discussion

In this paper, we propose a novel approach to fold a polyhedral surface into a much more compact form called stacking which enables us to fabricate (e.g. 3D-printing) and transport a large 3D model from a much smaller workspace. Our algorithm is based on the local stackability of Hamiltonian stacking that overcomes the drawback of the chain-structured Hamiltonian stacking, while achieving high compactness. Instead of a global optimal search strategy, our approaches divide the problem into a pipeline and each step is heuristically optimized to ensure the solution space of the next steps, and therefore our algorithm beats many traditional polyhedral

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.
unfolding algorithms while doing the essentially same task in terms of efficiency. Because our algorithm is based on partitioning, it can be adapted to applications where each partition uses a component of the geometry that has semantic meanings. As each partition is stacked into a single pile, it can potentially ease the process of assembling the semantic parts.

Limitations and Future Works The stacking relies on the surface being voxelized or remeshed into other regular shapes. It is difficult to approximate complex smooth shapes with a small number of identical faces. In the future development, it is possible to algorithmically fit the smooth surface into the thick panels or use deformable panels to better approximate the curvature of the smooth shape.

Additionally, we found that in some cases, a graph-structured net can also be stacked without collisions. Consider the example in Figure 21, the cube can be stacked through the process without breaking the loop consisting of the four side faces. This would bring more folding accuracy as well as structural stiffness in practice.

Figure 21: An unfolding of the Cube model that can stack into two piles while remaining its graph structure during the folding process.

References