Stackabilization

Honghua Li[1,2] Ibraheem Alhashim[1] Hao Zhang[1] Ariel Shamir[3] Daniel Cohen-Or[4]
[1]Simon Fraser University * [2]National University of Defense Technology [3]Interdisciplinary Center [4]Tel Aviv University

Abstract

We introduce the geometric problem of *stackabilization*: how to geometrically modify a 3D object so that it is more amenable to stacking. Given a 3D object and a stacking direction, we define a measure of stackability, which is derived from the gap between the lower and upper envelopes of the object in a stacking configuration along the stacking direction. The main challenge in stackabilization lies in the desire to modify the object's geometry only subtly so that the intended functionality and aesthetic appearance of the original object are not significantly affected. We present an automatic algorithm to deform a 3D object to meet a target stackability score using energy minimization. The optimized energy accounts for both the scales of the deformation parameters as well as the preservation of pre-existing geometric and structural properties in the object, e.g., symmetry, as a means of maintaining its functionality. We also present an intelligent editing tool that assists a modeler when modifying a given 3D object to improve its stackability. Finally, we explore a few fun variations of the stackabilization problem.

Keywords: stackability, stackabilization, shape optimization, structure-preserving deformation

Links: DL PDF WEB

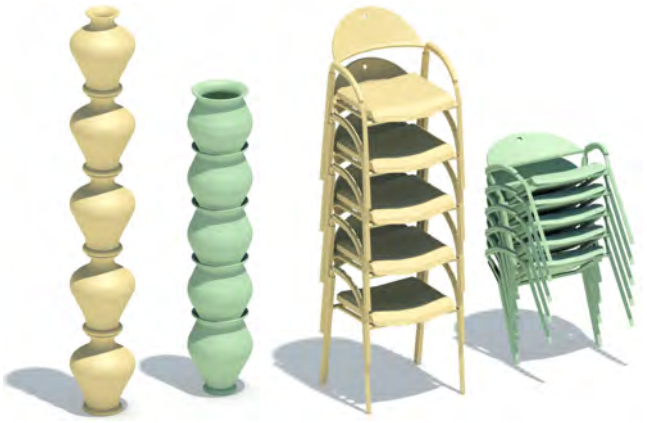

Figure 1: *Stackabilization: improving the stackability of 3D objects (yellow denotes input and green denotes output) by analyzing and optimizing geometry. The object geometry is only altered subtly so as to maintain the intended functionality and aesthetic appearance of the original object.*

1 Introduction

Stacking objects on top of each other is a common task performed by humans. Objects are often stacked to make or save space, for instance, while shipping or storing them. In product design and engineering, efforts have been invested to allow compact stacking without compromising the product's intended functionality. One of the most celebrated examples of stackable objects are chairs, where many space-saving and aesthetic designs of stackable chairs have been realized [Fiell and Fiell 2000].

An intriguing geometric question about stacking is: what makes some 3D objects more amenable to stacking than others. It is clear that concavity plays an important role for simple shapes such as those of bowls or cups. However, to precisely define and improve the *stackability* of a 3D shape turns into an interesting and challenging geometry problem, especially for shapes with more complex structures, such as tables and chairs.

In this paper, we are interested first in a geometric characterization of stackability for 3D shapes and second, in the development of algorithms to make a given 3D object more stackable. We use a new term for such an operation — *stackabilization*. The main challenge in stackabilization lies in the desire to only modify the shape's geometry and structure in subtle ways so as to maximize its stackability, while not adversely affecting its shape, functionality and aesthetic appearance. Understanding stackability and developing tools for stackabilization will assist designers in creating 3D models that can be packed together more compactly. Figure 1 shows the before-and-after of stackabilization of two shapes and illustrates the saving in space achieved.

The need for a specific tool, automatic or semi-automatic, and algorithms to assist a designer or modeler in solving stackabilization is emphasized by the global and unpredictable nature of the problem. As many examples in this paper illustrate, a small modification to the shape can have a great effect on the stackability of objects, while large modifications can have little to no effect at all. The reason is that modifying the shape on one side affects the contact with the opposite side. This problem is amplified when the shapes become more complex with certain parts cluttered, occluded, or inaccessible. Hence, the key problem for designers, even when manually modifying the shape, is to recognize which modifications are better and where to apply them in order to have the largest effect on stackability and the smallest effect on the design. This is exactly the strength of our automatic stackability analysis and the corresponding stackabilization algorithm.

Stackability. Geometrically, stacking an object on top of a copy of itself along a stacking direction can be achieved by translating one object copy along the stacking direction, just until the the two copies have no overlap, i.e., they are just touching each other. We call such a configuration the *stacking configuration* and the surface regions where the two copies maintain contact the *contact regions*. The extent of the minimal stacking translation can be found by looking at the maximum gap between the *upper* and *lower envelopes* of the object, and the contact regions are the regions inside these envelopes where this maximum gap is achieved; see Figure 2.

*e-mail: {honghual, iaa7, haoz}@sfu.ca

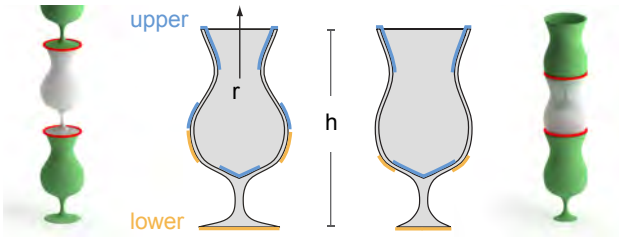

Figure 2: *The components for defining object stackability: the upper envelope (in blue), lower envelope (in orange), the shape extent h along the stacking direction* $\mathbf{r}$*, and contact regions (in red).*

We define the *stackbility* of an object using this maximum gap and its extent along the stacking direction.

When objects are stacked, we call the line strips that connect the centroids of all copies of the objects the *stacking trajectory*. The connected line segments in the stacking trajectory have a constant angle. The stacking trajectory is a straight line only if the translated copies are identical (Figure 1) with a possible rotation about stacking direction as shown in Figure 3(a). In contrast, the stacking trajectory of the cups in Figure 3(b) is not a straight line since the copies are rotated relative to each other and not only about the stacking direction. In some cases, adding an additional rigid transformation for stacking can create an overlap between non-adjacent copies of the object. For example, in Figure 3(a), the stool at the top of the stack intersects the one at the bottom, preventing any more stacking and resulting in a finite stack height.

Optimization. Following the goal of introducing only small modifications to a given object while improving its stackability, our strategy for stackabilization is to make local shape deformations near the contact regions. We perform a best-first search in the space of possible deformations, to locally deform the object in the vicinity of contact regions so as to maximize its stackability, while respecting a multitude of constraints. The constraints imposed serve to define the energy that guides the local search for the best deformations. This energy maintains the shape, structure, and functionality of the object. Specifically, our algorithm strives to maintain the bounding box of the object, with its extent along the stacking direction strictly preserved; it also maintains, as much as possible, the inter- and intra-part symmetries, among other functionality-oriented geometric or structural properties.

We adopt a component-wise controller framework similar to Zheng et al. [2011] as the deformation model for our stackabilization task. The cuboid and generalized cylinder (GC) controllers bound the shape parts and store geometric and structural shape properties among the parts. The local search to improve stackability is carried out in the feasible parameter space of the controllers, while the stored shape properties act, in part, to constrain the search.

Interactive Design. Building on the core module for locally improving stackability, we develop an interactive tool which allows the user to design stackable objects in several manners. In automatic settings, the user only needs to specify a target stacking height of a given number of objects, and the tool computes a handful of different feasible results respecting the user target. In manual settings, the tool visually provides several suggestions for modifications that can increase the stackability of an object. The user can then select one of those suggestions to deform the shape. This procedure can occur iteratively until the desired design is reached. The implementation of our algorithm is available at *http://www.computer-graphics.cn/ hh/projects/stackem/*.

(a) **(b)**

Figure 3: *Different stacking trajectories and object rotations (the stacking results are for illustration purpose only and were not produced using our tool). (a) A straight line trajectory is still possible by only rotating the copies about the stacking direction. Note that rotations can result in intersections between non-adjacent copies as seen in the highlighted region. (b) More general rotations of the stacked copies can lead to a zigzagging stacking trajectory.*

To test the utility and potential of the automatic tool, we conducted a preliminary user study comparing stackabilization using the tool vs. manual shape editing and obtained encouraging initial findings. Finally, we present several extensions to the basic problem of translational stackabilization. First, if the user can partition the shape into a few parts or remove one of its parts, our algorithm can greatly simplify the task of stacking rather complex shapes. We also show that our stackabilization algorithm is adaptable to other deformation methods, such as surface deformation.

2 Related work

Recently, computer graphics researchers have become more interested in leveraging shape analysis techniques to assist in design and engineering tasks such as architectural modeling [Kelly and Wonka 2011], layout design [Merrell et al. 2010; Merrell et al. 2011; Yu et al. 2011], inverse assembly [Lau et al. 2011], functionality visualization [Mitra et al. 2010] and physically valid shape creation [Umetani et al. 2012]. By combing real-time visual feedback with the ease of incorporating engineering and functional constraints resulting from shape analysis, these tools greatly facilitate the otherwise expensive and difficult design process, making rapid prototyping and fabricating of 3D digital objects a reality.

Our work falls into the general problem domain of shape optimization. Early works in computer graphics on the subject have focused on low-level geometric properties such as surface smoothness and triangle quality [Hoppe et al. 1993]. Such properties are not inherently related to the objects themselves but rather are artifacts of the digitization process. More related are works which optimize for higher-level shape properties. Mitra et al. [2007] symmetrize objects with minimal geometry modification. Closer to our work is the 2D escherization problem studied by Kaplan and Salesin [2000], where a given polygon is minimally altered so that it becomes more tilable. In this paper, we introduce and solve a new geometry problem, stackabilization of 3D objects. Our algorithm operates on more complex objects than simple polygons and the modification must be structure-preserving.

In other fields such as mechanical engineering, compact packing of 2D or 3D objects is a well-studied problem. For example, Dong et al. [2011] optimize shape morphing to fit components in a lim-

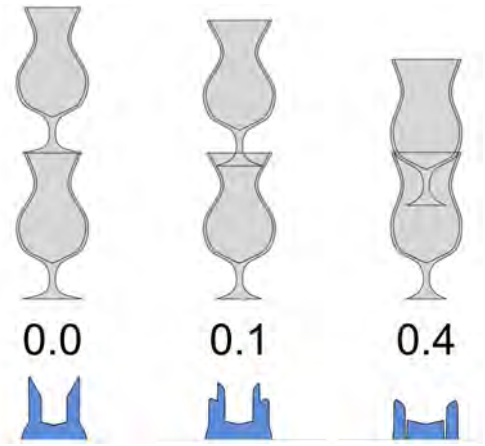

Figure 4: *Slight deformations on the shape may cause significant changes to the stackability (numbers in black) and gap function (in blue), whose maximum determines the minimum translation along the stacking direction.*

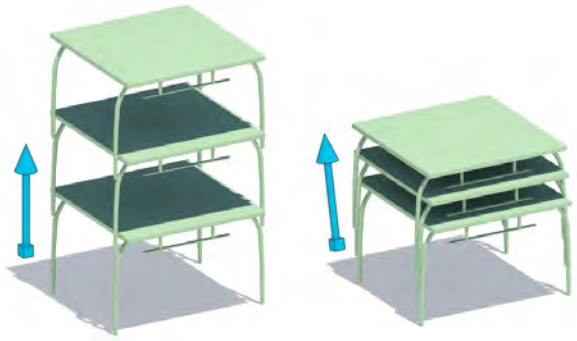

Figure 5: *Changing the stacking direction slightly can change the stackability considerably.*

ited space. They apply a mass-spring physical morphing method to under-hood layout design for vehicles. To the best of our knowledge, the problem of automated stackabilization had not been addressed before. The geometric and structural complexity of the objects and the degrees of freedom allowed in deforming the objects far exceed those addressed in a typical packing problem.

The focus of this work is to characterize object stackability and stackabilization. Our approach requires a deformation model which defines an underlying parameter space in which we formulate the constrained search we need to solve. We utilize either iWires [Gal et al. 2009] or component-wise controllers [Zheng et al. 2011] that are designed for interactive structure-preserving editing.

3 Stackability

Let us first consider the case of stacking that translates identical shape copies along the stacking direction. We define the stackability of an object along a stacking direction $\mathbf{r}$ using the maximum gap between the upper and lower envelopes of the object and its extent along $\mathbf{r}$. Given a 3D shape M and a stacking direction $\mathbf{r}$, we first compute two envelopes of the shape along $\mathbf{r}$. Intuitively, these envelopes represent the two visible surfaces along $\mathbf{r}$ from two opposite directions. For simplicity, we will refer to them as the upper $u(M, \mathbf{r})$ and lower $l(M, \mathbf{r})$ envelopes of the shape (See Figure 2). The difference between the two envelopes along direction $\mathbf{r}$ defines a *gap function*, whose value is the distance between them:

$$f(M, \mathbf{r}) = u(M, \mathbf{r}) - l(M, \mathbf{r}). \tag{1}$$

The maximum of f: $f_{\max}(M, \mathbf{r}) = \max(f(M, \mathbf{r}))$, defines the minimal stacking translation along the stacking direction $\mathbf{r}$.

Changing the shape of the object M would result in a different gap function, as shown in Figure 4, and can produce different stackability measures. The regions on the shape where the maximum of the gap function appears are called the *contact regions* as they are the places where the copies of the shape are in contact while stacking. These regions always occur in pairs, one on the upper and the other on the lower envelope, as illustrated in Figure 2. The contact regions are important for stackability as deformations of the shape near the contact regions tend to have the most impact on the stackability of the shape. Note that the gap function, contact regions, and thus stackability can differ considerably for different stacking directions $\mathbf{r}$; see Figure 5.

To derive our stackability measure, suppose that a shape M has a volume $V(M)$ of its axes orientated bounding box (OBB). Given a stacking direction $\mathbf{r}$, denote $h(M, \mathbf{r})$ as the extent of M along $\mathbf{r}$ (maximum difference in the direction $\mathbf{r}$ between all points on M), $V(M, \mathbf{r})$ as the volume of M's OBB along $\mathbf{r}$, and $f_{\max}(M, \mathbf{r})$ as the maximum of the gap function of M. The total volume of n such shapes without stacking is $V_0 = n \cdot V(M)$, while the total volume of a stack of n such shapes along $\mathbf{r}$ is:

$$V_1 = \left(1 + \frac{(n-1) f_{\max}(M, \mathbf{r})}{h(M, \mathbf{r})}\right) \cdot V(M, \mathbf{r}). \tag{2}$$

Hence, we define the space saving ratio of using stacking as $1 - V_1/V_0$ and at the limit we get the stackability $S(M, \mathbf{r})$ of a shape M along direction $\mathbf{r}$:

$$S(M, \mathbf{r}) = \lim_{n \to \infty} (1 - V_1/V_0) = 1 - \frac{f_{\max}(M, \mathbf{r})}{h(M, \mathbf{r})} \cdot \frac{V(M, \mathbf{r})}{V(M)}. \tag{3}$$

In other words, our definition of stackability reflects the space (volume) saving ratio of stacking multiple copies of an object along a stacking direction $\mathbf{r}$.

Note that in cases when an additional rigid transformation is used to stack objects, one can compute the OBB volume of the stack of n copies V_{stack}, and the shape's original OBB volume $V(M)$, and define the stackability as

$$S(M, n) = 1 - \frac{V_{\text{stack}}}{n \cdot V(M)}. \tag{4}$$

Although this measure depends on n (e.g. $n = 9$ in Fig 3a), many times n is pre-selected, for instance when there are physical restrictions of weight or space. In other cases a relatively large number of objects can be chosen, e.g. $n = 20$.

Stackability Computation. To find the minimal stacking translation in the stacking configuration, there is a need to compute the maximum gap between the upper and lower envelopes of the object. To achieve this, one needs to define the two continuous envelopes and compute their maximum distance. Instead, we utilize graphical processing abilities allowing us to sample the two envelopes by rendering them using OpenGL orthogonal view projection. To compute an "image" sample of the upper envelope, the camera is placed at the centroid of the shape, facing towards a direction opposite to the stacking direction $\mathbf{r}$, and then translated along $\mathbf{r}$ until the entire object can be seen. In this configuration after projection the depth buffer will contain only the closest pixels to the camera which is a

Figure 6: *The original shape (in grey) has two deformed solutions which have almost the same compactness of stacking. The solution produced by our method (in green) keeps the original functionality and aesthetic appearance, while the solution generated by gap-function based method (in orange) loses most of the original functionality. Top left shows the gap functions of these three shapes.*

sampling of the upper envelope of the object. To compute the lower envelope, a similar procedure is applied when the camera is facing towards $\mathbf{r}$ and moved along the opposite direction. Given these two envelope images, the gap function is defined by their difference (see examples at top left of Figure 6). To identify the contact regions we find all positions (pixels) containing the maximum of the gap function. Since the stackability of a shape differs for different stacking directions, we search for the best direction by uniformly sampling the unit sphere and picking the direction $\mathbf{r}$ that achieves the best stackability measure.

4 Improving stackability

In essence, minimizing the maximum of the gap function results in an increase of the stackability of a shape. Figure 4 shows how the maximum of the gap function directly determines the compactness of the stacking. A naïve approach to increase stackability would be to reduce the maximum $f_{\max}(M, \mathbf{r})$ by changing the envelopes and then reconstructing the shape. This approach has several disadvantages. First, given a gap function $f(M, \mathbf{r})$, there are an infinite number of modifications one can apply to the envelopes and it is not clear which one would reduce stackability. Second, the gap function is defined by the envelopes and it hardly reflects the structure of the 3D shape, and reconstructing a new shape from its envelopes is not straightforward. More importantly, simple minimization of the gap can alter the shape of the object and harm its functionality. For example, stackability does not only relate to $f_{\max}(M, \mathbf{r})$, but also to the shape extent $h(M, \mathbf{r})$. In Figure 6, scaling down the gap function while keeping the upper envelope constant results in squashing the stool. This is indeed space saving but destroys the functionality and aesthetic appearance of the original shape.

In contrast, we propose a method that enables improving the stackability of a shape while preserving its pre-existing geometric and structural properties. First, we focus on modifications near the contact regions of the shape instead of the whole envelopes, and second, we use a set of constraints to preserve the shape's structure. However, even when limiting the deformations to these specific regions, there is still a large space of feasible deformations. To this end, we employ a *Best-First Search* in the space of possible shape modifications. To guide the search, we use an energy function that contains both a stackability improvement objective term as well as structure preservation objective terms. Since stackability also depends on the stacking direction chosen (see Figure 5), for each local modification, we also search for the best stacking direction which provides maximal stackability improvements.

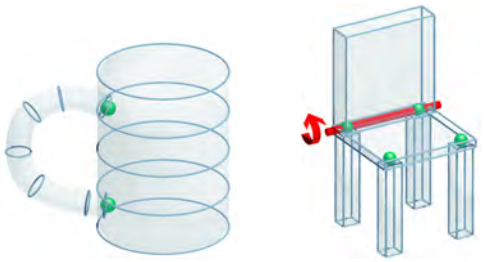

Figure 7: *Two kinds of joints considered in our deformation model: point joints (green spheres) and linear joints (red bar).*

4.1 Deformation model

To perform a structure-preserving editing to a shape, we adopt a deformation model similar to the component-wise controllers proposed in [Zheng et al. 2011]. The construction of component-wise controllers demands a segmentation of the given shape into meaningful parts. Structured objects can be easily decomposed into meaningful parts, e.g., by the method proposed in [Attene et al. 2006]. Connected objects can be segmented with some user assistance similar to [Meng et al. 2011].

Once the shape is segmented, we use two types of primitives to fit the shape parts: cuboid and generalized cylinder (GC). Fitting a cuboid amounts to computing the OBB of a part. To fit a GC, a 1D curve skeleton is first extracted from the component [Au et al. 2008], and then cross section circles are constructed with radii that are defined by the distance from the skeleton to the surface. For each part we select the primitive that fits the component better, but this choice can be specified by the user as well.

Each type of primitive has some predefined degrees of freedom. Both the cuboids and GCs can be rigidly transformed or scaled. In addition, each cross section of a GC can be translated or scaled. The translation or scaling of each cross section is propagated to nearby cross sections using a Gaussian weight (with standard deviation $\sigma = 0.2$) depending on their distance along the axis. To transfer the deformation from the primitive to the underlying geometry, we use mean value coordinates [Ju et al. 2005] for cuboids, and skinning with dual quaternions [Kavan et al. 2007] for GCs.

The structural properties of each component and the inter-relations among the components are key to preserving the structure and function of a given object. In this work, we preserve self symmetry, pairwise symmetry, point joints, and line joints. We adopt the voting method of [Mitra et al. 2006] to detect all the self-symmetries of the entire object. Then we check each component whether it possesses the same self-symmetries or has a symmetric counterpart. Apart from symmetry, we also found that *joints* play an important role in defining the functionality of man-made objects. A *point joint* usually appears for tube-like segments, for example, between a chair leg and seat (green spheres in Figure 7), and between the handle and body of a cup. A *linear joint* occurs between two plate-like segments of the shape, for example between a chair seat and back (red bar in Figure 7). To compute the joints among components, we voxelize the primitives and detect their intersections.

To make use of these relations, we construct a graph, in which each node is a primitive, and each edge is the relation between two nodes. Once a node is modified, this modification will be propagated to its neighbors via the graph edges in a similar way as the controller-based deformation work of Zheng et al. [2011]. During propagation, each edge will modify the other node so as to preserve the relation it defines. In the case where more than one node is mod-

Figure 8: *Progress of stackabilization on a mug and a table model. During each iteration, contact parts (in orange) and active deformation handles (in red) are detected. Contact-driven deformation is applied to the shape to improve its stackability.*

ified, the node that has the most modified neighbors is selected as the next target to be propagated. When a node is constrained by both symmetry and joints, a symmetry relation has precedence.

4.2 Contact-driven deformation

Recall that contact regions are identified by detecting the maximum of the gap function. We refer to the object parts containing contact regions as *contact parts* (colored in orange in Figure 8). We map each contact region to *deformation handles* of the contact parts. A deformation handle is a part of the primitive that is used to drive the deformation of the primitive; it can be a point in, or a face of, a cuboid, or a cross section of a GC (colored in red in Figure 8). Once a deformation is applied to a handle, the result depends on the global primitive properties and relations in the object. For example, inter-part symmetries are preserved during the deformation of one handle (see the legs of the table in Figure 8).

Once the deformation handles of the contact parts are identified, local edits can be applied near the contact regions. The guiding principle for these edits is to separate the contact region pairs by moving the two regions apart. Such a contact-driven method helps reduce the search space for stackability-improving modifications by efficiently locating regions on the object part where modifications are most likely to improve the stackability. Still, an exhaustive search for the best edits is impractical due to the size of the search space for the deformation handles. Hence, we derive a set of rules to prioritize edits based on the different contact region pairings.

There are three types of contact regions: ring-like, linear, and dot-like. Contact regions are all classified as being dot-like unless their shape is considered as a line, by analyzing its bounds, or as a ring by checking whether it has an interior hole. A ring-like contact region always leads to scaling of the deformation handle. A dot-like contact region can either be translated vertically or horizontally. A linear contact region leads to the same local edits as a dot-like contact region, except that moves perpendicular to the stacking direction $\mathbf{r}$ are restricted to be along the direction that is perpendicular to the primary direction of the linear contact region. Moreover, when a single contact region lies on the boundary of the bounding volume, we limit the type of edits to movements perpendicular to $\mathbf{r}$. This way the shape is not squashed as squashing can dramatically destroy the intended functionality of the shape (see Figure 6).

Figure 8 illustrates our contact-driven deformations. The deformation handles are highlighted in red and they are scaled or locally translated depending on the different types of contact regions. The deformation of the contact parts will later be propagated to other parts via the relational edges in the graph that was defined in the shape analysis stage (section 4.1).

4.3 Optimization

To improve stackability, the user sets a target stackability value and our algorithm uses contact-driven deformations (4.2) to attempt to achieve this goal. For some simple shapes, a single deformation may be sufficient to attain the target stackability. However, for shapes with a more complex structure, like tables or chairs, a trivial solution usually does not exist, and multiple iterations of contact-driven deformations are needed. Each iteration can utilize different contact regions, and can generate multiple successor shapes using different modifications. To search for the best solutions which introduces only subtle deformations to the given shape while improving its stackability, we apply an optimization technique using *best-first search* in the space of possible modifications.

To guide the optimization, we define a measure between the deformed shape M and the original shape M_0 that includes both a shape preservation term and a stackability term (for simplicity of notation, we drop the stacking direction $\mathbf{r}$ below):

$$E(M) = \alpha D(M_0, M) - (1-\alpha)\Delta S(M_0, M), \qquad (5)$$

where α is a weighting coefficient between the change of M compared to M_0 and the improvement of stackability. We use $\alpha = 0.2$ in all our experiments to place more weight on stackability improvement rather than shape changes. The second term $\Delta S(M_0, M) = S(M) - S(M_0)$ is the stackability change of the current shape M compared to the original shape M_0. The first term $D(M_0, M)$ measures the shape change of M from M_0, with respect to some structure properties of the shape. In our experiments, we take into account the change of OBB volume, proximity, bounding box and symmetry of the deformed controllers,

$$D(M_0, M) = \frac{1}{4}\Big(D^{vol}(M_0, M) + D^{prox}(M_0, M) + D^{BB}(M_0, M) + D^{symm}(M_0, M)\Big). \qquad (6)$$

Suppose there are p primitives in the object, which are denoted as $\{P_{M_0}\}$ ($\{P_M\}$) before (after) the deformation. The change of volume is the sum over the volume differences of all primitives,

$$D^{vol}(M_0, M) = \frac{\sum_{i=1}^{p} \mid Vol(P_M^i) - Vol(P_{M_0}^i) \mid}{\sum_{i=1}^{p} Vol(P_{M_0}^i)}. \qquad (7)$$

The proximity change is the sum of distance differences between all pairs of primitives,

$$D^{prox}(M_0, M) = \frac{\sum_{1 \le i,j \le p} \mid d(P^i_M, P^j_M) - d(P^i_{M_0}, P^j_{M_0}) \mid}{\sum_{1 \le i,j \le p} d(P^i_{M_0}, P^j_{M_0})}. \tag{8}$$

The bounding box term sums over the extent differences along all three main axes of the entire shape's bounding box,

$$D^{BB}(M_0, M) = \frac{\sum_{i \in \{x,y,z\}} \mid h^i(M) - h^i(M_0) \mid}{\sum_{i \in \{x,y,z\}} h^i(M_0)}. \tag{9}$$

Suppose $T_{M_0} : a_{M_0} \rightarrow b_{M_0}$ is the symmetry map for a symmetric pair $< a_{M_0}, b_{M_0} >$, then the symmetry distortion generated by the deformation is $\delta(T(a_M), b_M)$, where $\delta(\bullet, \bullet)$ sums up the differences between corresponding vertices over all points of the deformation controllers. Assume there are m symmetry pairs in M_0 which need to be preserved, then the symmetry distortion of the deformation is measured by the sum of distortion over all those symmetry pairs,

$$D^{symm}(M_0, M) = \frac{1}{m} \sum_{i}^{m} \delta(T_{M_0}(a^i_M), b^i_M). \tag{10}$$

Algorithm 1 Improve Stackability

Input: Shape M_0, the target stackability $targetS$
Output: Improved shape M

```
currS = 0
U.enqueue(M_0)
while currS < targetS do
    M_c = U.dequeue()
    currS = STACKABILITY(M_c)
    if currS ≥ targetS then
        M = M_c
    else
        E = CONTACTDRIVENDEFORM(M_c)
        for each local modified shape e_i ∈ E do
            U.enqueue(M_c)
        end for
    end if
end while
```

To minimize $E(M)$ in Equation 5, we employ a best-first search (see Algorithm 1). We initialize the *candidate* set U of the search with the given object $U \leftarrow \{M_0\}$. In each iteration, we pick the best candidate shape $M_c \in U$ which has the minimum objective energy (Equation 5). If its stackability reached the desired goal, *targetS*, we stop and return M_c as our result. Otherwise we apply contact-driven deformations to this candidate to create a number of new successors, and we add each of them to U and continue. We use a priority queue for the set U resulting in the best candidate always being picked first in the next iteration.

The proposed algorithm can also generate multiple solutions if it continues the search after the first solution is found. Recall that the computation of stackability involves searching for the best stacking direction, which is computationally expensive. With the observation that the best stacking direction should not change too much during the optimization, the search is performed only inside a cone that is centered at the previous stacking direction on the unit sphere. Furthermore, the searching for stacking direction can also be restricted to lie on the symmetry planes of the shape.

5 User interface and interactions

We present two modes of interaction for stackabilization. In the first, the user specifies a desired stackability value that reflects his/her intended space saving requirement, then *automatic* stackabilization is performed according to Algorithm 1. The results of the algorithm are a handful of solutions that satisfy the stackability requirement, implementing different modifications to the original shape. In the second mode, the user is involved in an interactive editing session, our analysis technique is used to assist user edits, by showing various *suggestions*, while improving the stackability.

Automatic stackabilization. The automatic algorithm starts with the initial configuration and iterates the stackability improvement procedure until achieving the goal set by the user. Since there may exist multiple possible deformations in each iteration, our algorithm explores different paths to the desired value of stackability. The solutions are then sorted based on their energy value. The top-rated results are presented to the user. The final user selection can be based on aesthetics or functionality considerations.

Interactive suggestive stackabilization. In this mode of stackabilization, the stackability improvement process is performed by the user who is assisted by our analysis. At each step, the user is presented with all the possible stackability-improving edits, so the most promising option can be chosen. All the edits are organized in a tree structure, so the user can preview and evaluate the different editing options, and roll back to previous suggested edits as well. After selecting and applying the chosen edit option, new suggested edit options are computed again. This cycle continues until the user is satisfied with both the stackability value and the resulting deformed shape. This mode can better preserve the functionality and aesthetics of the original shape compared to the automatic option, as it utilizes user knowledge throughout the process.

6 Experiments

In this section, we show some visual results of stackabilization and describe a preliminary user study we conducted to test the effectiveness of our stackabilization tool in comparison to manual modeling efforts. Findings from the study are also reported. The key question that motivated the user study is whether our automatic stackability analysis, along with the resulting stackabilization algorithm, has the potential to outperform manual stackabilization.

Results. Container-type objects such as cups, baskets, pots, vases, buckets, are often stored in stacks to save space. Figure 9 shows some results produced by our automatic stackabilization tool. Stackable furniture are often stacked and stored in a compact fashion when not in use. For example, stackable chairs are often used as a means of expanding permanent seating in an auditorium, arena or in a house of worship. Such kinds of furniture come in versions that are stacked horizontally as well as ones that are stacked vertically. Figure 10 shows the stackabilization results of furniture, including chairs, tables and stools.

Note that all these results are automatically generated by the automatic mode of stackabilization. Different choices of controllers may result in different stackabilization outputs, mainly due to the intrinsic degrees of freedom of the controller. Figure 11 shows two different solutions of stackabilization created by fitting either cuboids or cylinders to the legs.

Preliminary user study. We conducted a user study where human participants including both expert modelers and graduate stu-

Figure 9: *Stackabilization results on several container objects: cup, basket, hat, pot and bucket.*

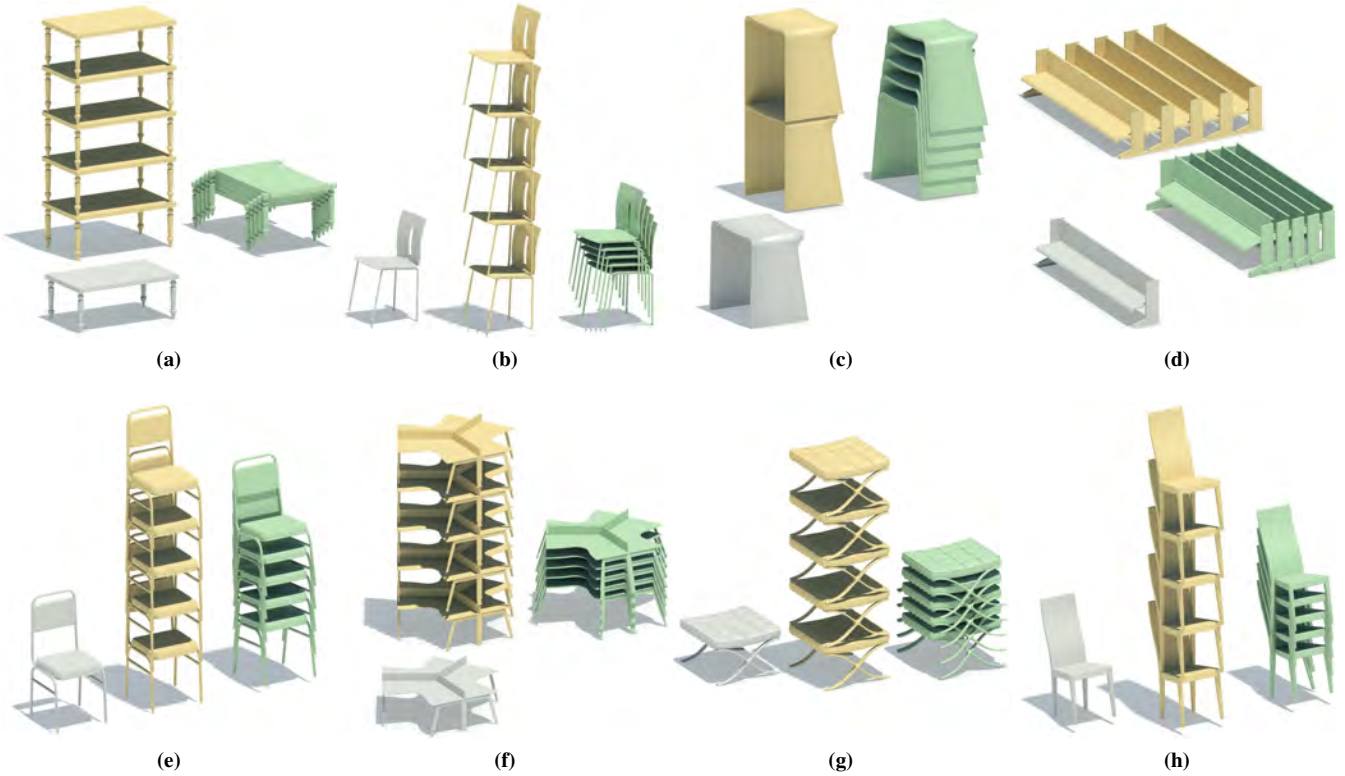

Figure 10: *Stackabilization results on several furniture models.*

Figure 11: *Two different stackabilization solutions for the table are obtained by using cuboids or GCs to fit the legs respectively. These two solutions have the same stackability ($S = 0.8$), but introduce rather different distortions to the shape geometry and functionality.*

dents in visual computing were asked to improve the stackability of 3D objects to target stackability values while trying to maintain the geometry and structure of the objects as much as possible. We provided the users with a minimalistic modeling tool possessing the same degrees of freedom for structure-preserving deformation and editing as our automatic algorithm. The user was allowed to change the stacking direction to achieve the target stackability more quickly. In order not to place too much 3D modeling burden on the participants, we deliberately selected test shapes that are not too complex to analyze or visualize. We expect that more complex 3D shapes will be more difficult to manipulate and stack manually.

The current user study is only informal and not intended to be seen as a formal evaluation of our stackabilization tool. To rigorously compare an automatic tool with manual 3D modeling has many challenges and we leave that for future work. However, the initial findings from the study are encouraging. For one, the study con-

Figure 12: *A failure case: Our analysis algorithm was able to detect the best stacking direction for the swivel chair (left) but failed to improve its stackability within the current search space (middle), while an expert modeler can (right).*

firms that manual editing takes significantly more time to fulfill the stackabilization goals compared to the automatic tool. We should note however, that the modeling tool we developed for manual editing is inevitably limited. Some participants felt that more modeling capabilities should have been offered. In terms of quality of results, we find that the automatic tool generally achieves a better balance between improving stackability and structure preservation. We also asked the expert modelers to provide feedback. Generally, they felt that the shapes are "a bit too easy" in terms of finding the right strategy for improving the stackability. However, they all acknowledged that to more precisely manipulate the 3D objects to achieve the target takes more effort. This is consistent with our belief that stackabilization is a delicate modeling task, raising the potential of automated shape analysis and optimization.

7 Conclusion, discussion and future work

We have presented a method that analyzes and optimizes the geometry of a given 3D shape to increase its stackability. The key to solving the problem of stacking an object onto itself lies in the analysis of the relation between its upper and lower envelopes, which we expressed as the gap function.

In other fields, the problem of fitting two geometries together is referred to as "docking". Typically, docking problems take two separate geometries as input, while here we "dock" the geometry of a shape to itself, or specifically, the geometries of the upper and lower envelopes of the same object. The two docking parties are tightly coupled by the body of the shape itself, constraining the problem. The objective of increasing the stackability is subject to maintaining the overall shape, and thereby its functionality, which are the main challenges one faces when solving this problem.

We used shape analysis to better constrain the possible deformations to a subset where the semantic relations of the shape are preserved. We presented an automatic optimization method to increase the stackability of a given shape, and presented a user interface where guided manual editing of shapes could support better functionality and aesthetics preservation.

Limitations. Our deformation is based on component-wise controllers [Zheng et al. 2011] and supports only cuboids and GCs. This could be extended to other controllers and more sophisticated deformation methods. In addition, our set of feasible deformations

Figure 13: *Folding (and then stacking) is another type of space-saving technique that is not handled by our method. It also poses an interesting, likely more complex, geometry analysis and optimization problem worth exploring.*

that can be applied to the shape is limited. For instance, we do not bend the shape or change its topology, e.g., to create a hole. Such modifications can at times assist stackability without harming the functionality of the object. Figure 12 shows that our tool can search for the best stacking direction, but fail to improve stackability. The reason is two-fold: first the shape is not designed to be stackable, which needs more drastic deformation to stackabilize; second, both the controllers and the allowed deformations are too limited to allow such drastic shape changes. Last but not the least, our current work does not take into account of physical constraints such as stability, which can be a critical requirement in real-world applications. The recent work on physical valid shape creation [Umetani et al. 2012] may be adapted to address this limitation.

For stacking along polynomial trajectories,we suspect that an additional conflict checking mechanism is required for each additional copy in the stack. If we consider a piecewise linear path, we could iteratively add new copies to the stack then optimize the shape such that only non-destructive shape modifications are allowed. However, in the current implementation we do not detect the conflicts between non-adjacent copies.

Future directions. The resemblance of our problem to the docking problem opens opportunities for potentially new problems. For example, extending the problem to deal with articulated shapes, possibly with parts that can be folded like the one demonstrated in Figure 13. Another interesting problem is the docking or stacking of two separate geometries, or a multi-way one, which turns into a packing problem. The uniqueness of our work is that we aim at modifying and optimizing the geometries of the participating objects, rather than to search for rigid (or articulation) transformations that dock or pack them together to save space.

Other extensions to our work include the use of more complex shape modifications such as part removal or object partitioning to improve stackability. For instance, by separating the seat from the chair in Figure 14, both the seat and what remain in the chair are easier to stackabilize. These types of modifications can become rather complex even for a designer, but using our analysis tool greatly simplifies this task. Another direction is using surface deformation to stack up organic shapes, like the bottle in Figure 15. Generally speaking, our method optimizes the geometry of a shape to improve its functionality, or one of its properties. We believe that this will open the way to other application-driven geometric optimizations, beyond stackabilization.

Acknowledgement. We thank the anonymous reviewers for their valuable comments. We thank Yanir Kleinman for initial testing of the stackabilization concept, to Kai Xu for discussion on the con-

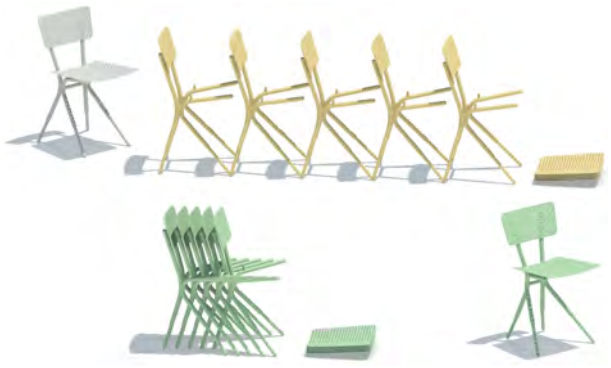

Figure 14: *Object partitioning may make stackabilization easier. The chair shown may be difficult to stackabilize as a whole, but when divided into two parts, each part is easy to stackabilize.*

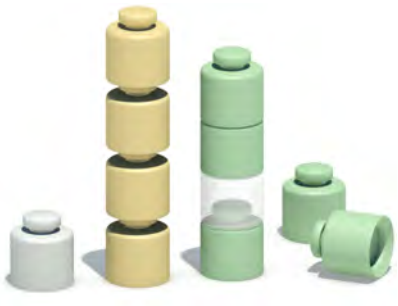

Figure 15: *Our approach can be extended to include other deformation types such as freeform surface deformations needed to improve the stackability of this bottle.*

troller deformation code, and to all the participants of the user study. This work is supported in part by grants from China Scholarship Council, National Science and Engineering Research Council of Canada (No. 611370), the Israel Ministry of Science and Education, and the Israel Science Foundation (No. 324/11).

References

ATTENE, M., FALCIDIENO, B., AND SPAGNUOLO, M. 2006. Hierarchical mesh segmentation based on fitting primitives. *THE VISUAL COMPUTER 22*, 181–193.

AU, O. K.-C., TAI, C.-L., CHU, H.-K., COHEN-OR, D., AND LEE, T.-Y. 2008. Skeleton extraction by mesh contraction. *ACM Trans. Graph. 27*, 3.

DONG, H., GUARNERI, P., AND FADEL, G. 2011. Bi-level approach to vehicle component layout with shape morphing. *Journal of Mechanical Design 133*, 4, 041008.

FIELL, C., AND FIELL, P. 2000. *1000 Chairs*. Taschen.

GAL, R., SORKINE, O., MITRA, N., AND COHEN-OR, D. 2009. iwires: An analyze-and-edit approach to shape manipulation. *ACM Trans. on Graph 28*, 3.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '93, 19–26.

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3 (July), 561–566.

KAPLAN, C. S., AND SALESIN, D. H. 2000. Escherization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 499–510.

KAVAN, L., COLLINS, S., ŽÁRA, J., AND O'SULLIVAN, C. 2007. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '07, 39–46.

KELLY, T., AND WONKA, P. 2011. Interactive architectural modeling with procedural extrusions. *ACM Trans. on Graph 30*, 14:1–14:15.

LAU, M., OHGAWARA, A., MITANI, J., AND IGARASHI, T. 2011. Converting 3d furniture models to fabricatable parts and connectors. *ACM Trans. on Graph 30*, 85:1–85:6.

MENG, M., FAN, L., AND LIU, L. 2011. A comparative evaluation of foreground/background sketch-based mesh segmentation algorithms. *Comput. Graph. 35* (June), 650–660.

MERRELL, P., SCHKUFZA, E., AND KOLTUN, V. 2010. Computer-generated residential building layouts. *ACM Trans. on Graph 29*, 181:1–181:12.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. *ACM Trans. on Graph 30*, 87:1–87:10.

MITRA, N. J., GUIBAS, L. J., AND PAULY, M. 2006. Partial and approximate symmetry detection for 3D geometry. *ACM Trans. on Graph 25*, 3, 560–568.

MITRA, N. J., GUIBAS, L. J., AND PAULY, M. 2007. Symmetrization. *ACM Trans. on Graph 26*, 3, 63:1–8.

MITRA, N. J., YANG, Y.-L., YAN, D.-M., LI, W., AND AGRAWALA, M. 2010. Illustrating how mechanical assemblies work. *ACM Trans. on Graph 29*, 58:1–58:12.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Transactions on Graphics 31*, 4.

YU, L.-F., YEUNG, S.-K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. J. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Trans. on Graph 30*, 86:1–86:12.

ZHENG, Y., FU, H., COHEN-OR, D., AU, O. K.-C., AND TAI, C.-L. 2011. Component-wise controllers for structure-preserving shape manipulation. *Computer Graphics Forum (Special Issue of Eurographics) 30*, 2, 563–572.