DOI: 10.1111/cgf.12050
EUROGRAPHICS 2013 / I. Navazo, P. Poulin
(Guest Editors)
Volume 32 (2013), Number 2

Computing and Fabricating Multiplanar Models

Desai Chen, Pitchaya Sitthi-amorn, Justin T. Lan and Wojciech Matusik

MIT CSAIL

Figure 1: *Our method takes an input mesh of primitives (left) and deforms it to a small number of polygons (center). The reduced mesh can be fabricated using a laser cutter and textured to easily create large physical models (right).*

Abstract

We present a method for converting computer 3D models into physical equivalents. More specifically, we address the problem of approximating a 3D textured mesh using a small number of planar polygonal primitives that form a closed surface. This simplified representation allows us to easily manufacture individual components using computer controlled cutters (e.g., laser cutters or CNC machines). These polygonal pieces can be assembled into the final 3D model using internal planar connectors that are manufactured simultaneously. Our shape approximation algorithm iteratively assigns mesh faces to planar segments and slowly deforms these faces towards corresponding segments. This approach ensures that the output for a given closed mesh is still a closed mesh and avoids introducing self-intersections. After this step we also compute the shape of polygonal connectors that internally hold the whole mesh surface. Both the polygonal surface elements and connectors can be manufactured in a single cutting pass. We validate the use of our method by computing and manufacturing a variety of textured polyhedral models.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Computer graphics has greatly contributed to developing tools for creating 3D models. Currently, a wealth of 3D models exists; there are many 3D modeling tools both for professionals and casual users. However, converting a 3D model to a physical equivalent is still a difficult and expensive process. There is a strong need to develop a suite of different methods to make this process inexpensive, fast, high-fidelity, and thus available to a wide population.

One of the most promising solutions for converting 3D textured models to physical equivalents is 3D printing. Unfortunately, there are still many disadvantages to using this method. First, 3D printing has a high material cost. Printing a human-sized figure would be quite expensive and would take days. Additionally, the size of printed objects is limited by the printing volume of a given 3D printer. Furthermore, printing materials are usually restricted to plastics and only one printer (3D Systems' ZPrinter) supports full-color printing. Finally, manufacturing light empty shells is not possible

using current 3D printing technologies, since a closed surface would be filled with a support material. Overall, we are not aware of a method that allows converting 3D models to physical replicas that is inexpensive, fast, allows for large objects, supports shells, and allows different types of materials.

In this paper we present a process for converting a textured 3D model to a physical copy. We represent a model using planar pieces and connectors, all of which can be manufactured using computer-controlled 2D cutting tools. Constructing models using this method is inexpensive and fast. Furthermore, we can manufacture large and multi-material objects. Our objects are inherently light since they are hollow.

In order to minimize the assembly time, our goal is to approximate the object with a small number of polygonal primitives. Given a target number of primitives and an optional user-specified saliency map over the input mesh, our method iteratively clusters mesh faces, estimates planes for each cluster, and slowly deforms the faces toward each plane. The method minimizes mesh distortion, avoids self-intersections, and retains mesh connectivity. The saliency map allows the user to specify which mesh regions are more important and should be assigned more polygons. Since both the original and the deformed mesh share the same parameterization, supporting textures is straightforward. Textures can be directly printed onto the planar polygons using a flatbed printer, or they can be printed separately and attached afterwards. One of the results of our system is shown in Figure 1.

Once the object is decomposed into planar textured polygons, our process automatically designs planar connectors that internally join neighboring surface polygons. The shape of each connector is adjusted to the angle between the two planes. In cases connectors cannot be placed due to small space or small angle between the planes, we connect the neighboring polygons using a saw-tooth pattern embedded in each polygon.

We believe that there is a wide range of applications for the proposed system. For example, fabricated models can be employed for more effective advertising in contrast to simpler planar cutouts. Our system can also be used to make wooden toys, 3D puzzles, and inexpensive props for theatrical productions. Since the models can be made from rugged materials (e.g., wood, acrylic), it is possible to use them in an outdoor setting as well.

2. Related Work

We review prior work in several relevant areas. First, our work is related to mesh simplification techniques [CMO97, GH97, Hop99, GGH02]. These methods typically work with triangle or quad meshes and simplify models to thousands or hundreds of triangles. However, mesh simplification methods do not work well in the case of extreme model simplification (e.g., when using tens of triangles/quads) because they usually use a greedy approach and they typically only work with triangles and quads. However, our optimization algorithm uses arbitrary polygons and is not restricted to triangles or quads. We show that reasonably complex models can be well represented using only tens of primitives, if arbitrary polygons are allowed. In this respect, our work is more related to variational shape approximation [CSAD04] that approximates a shape with a set of planar polygons. Their method repeatedly clusters the original faces onto a set of planar proxies and then stitches these planar proxies. In contrast, our method gradually deforms the original mesh such that the mesh is always closed. Most simplification algorithms do not check for self-intersections. Cohen et al. [CMO97] show how to avoid self-intersection by computing a 3D Voronoi diagram of triangles. They also check that each newly added triangle does not intersect the current approximation mesh. Our algorithm checks self-intersection at each deformation step in a much simpler fashion, as explained in 3.1.2. Our work is also related to billboard clouds by Decoret et al. [DDSD03], which uses multiple disconnected planes to represent a model. However, that method is focused on efficient rendering rather than on making models that can be manufactured. The work of Holroyd and colleagues [HBLM11] also decomposes a model into a set of planes. In their setting the planes are parallel to each other, allowing the model to be easily manufactured (e.g., by printing on layers of glass).

Another set of methods related to our approach are computational methods for converting 3D models to papercraft figures. Demaine and O'Rourke [DO07] give an overview of geometric algorithms in the area of computational origami and folding. Tachi [Tac09] presents a practical method that computes a single-sheet folding pattern for a given 3D polyhedral surface. Mitani and Suzuki [MS04] approximate a model by a set of continuous triangle strips and show a corresponding physical model made of paper strips. Similarly, Shatz et al. [STL06] and Massarwi et al. [MGE06] present algorithms for segmenting a given mesh into developable parts that can be cut and glued together. These techniques typically require special handling of the boundaries to make sure that the model remains closed. The models can be manufactured only from materials that can bend since the strips are not necessarily planar. We limit our primitives to planar polygons, which makes the model approximation process more difficult. However, this allows us to use a wider range of materials, and the manufactured models can be much larger.

The methods for computationally generating paper pop-ups given a 3D model have also been investigated in the computer graphics community. Glassner [Gla02] reviews the methods for computational design of pop-ups and presents a virtual authoring environment. Li et al. [LSH*10] develop a completely automatic algorithm that outputs a parallel paper architecture that approximates a desired 3D model. This

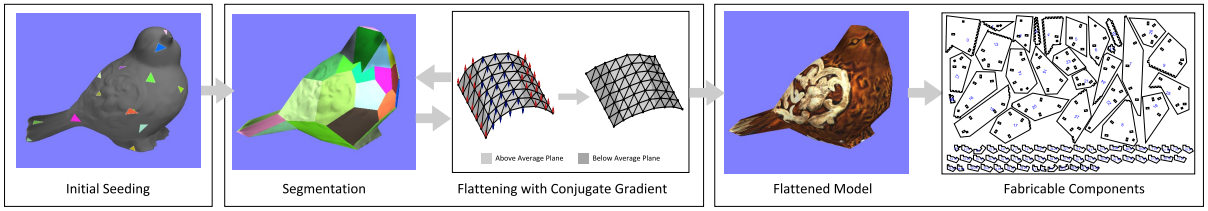


Figure 2: *Overview of the algorithm. Our system first finds some initial cluster seeds, then alternatively iterates between segmentation and flattening (described in 3.1. The result is a flattened model that is further processed into fabricable components.*

work has been extended to more general v-style pop-ups [LJGH11].

The recent approaches that approximate a model using a collection of interlocking planar shapes are also related to our work. SketchChair [SLMI11] is a system that allows designing and fabricating chairs using a 2D sketch-based interface. McCrae et al. [MSM11] introduce an algorithm that greedily selects planes that form a shape-proxy to approximate a given 3D model. The approach of Hildebrand et al. [HBA12] uses a binary space partition tree as a representation for a mutually intersecting planar cut-out. They present an algorithm to generate these cut-outs that guarantees assembly. In contrast, our system attempts to approximate directly the shape and appearance of the input 3D model.

3. Computing Multiplanar Models

Our process for converting a 3D textured model to 2D components that can be manufactured using computer-controlled cutting machines has two main steps. In the first step we present an algorithm that converts a 3D model to a mesh with a small number of polygons (Section 3.1). In the second step, we convert this polygonal mesh to planar components and their corresponding connectors, taking into account constraints imposed by the material (Section 3.2). The overall process is illustrated in Figure 2.

3.1. Deforming 3D Models to Polygonal Meshes

It is challenging to simplify a mesh into a relatively small set of polygons while ensuring that the simplified mesh remains closed and free of self-intersections. We describe a simple algorithm that accomplishes this task. We pose the problem as an iterative, mixed discrete-continuous optimization problem. The *discrete* optimization step involves assigning each mesh triangle to one of k discrete planes. We cast this problem as a Markov-Random-Field and solve it efficiently using a multi-label minimum-cut algorithm [YB01, VK04, YB04]. Based on the computed assignment of faces to planes, we recompute the location of each plane. Then, in the *continuous* step, we deform each face towards its assigned plane by solving a linear system of equations. After this step, we also recompute the k planes.

In the first iteration of the mesh segmentation algorithm, we initialize the planes using a k-means++ algorithm [AV07]. We repeat the two different optimization steps until the system converges. Typically, a small number of iterations is sufficient (approximately 50). It is important to note that our algorithm only moves vertices. The original and deformed mesh have exactly the same number of triangles and connectivity.

3.1.1. Mesh Segmentation

In the discrete step, our task is to segment the mesh assigning one of the k planes to each triangle. We define labels and associate each label f with a plane. This plane is described using the equation $\mathbf{x} \cdot \mathbf{n}_f - d_f = 0$ for all x on the plane, where $\mathbf{n}_f$ is the plane normal and d_f is a constant for the plane. Given plane equations for all labels, we would like to compute an assignment of triangles to labels (planes) such that (1) each triangle lies close to the corresponding plane and (2) neighboring triangles share the same labels. In particular, given a graph, this algorithm seeks a labeling that achieves a local minimum of the cost function defined as follows:

$$E = \sum_p D_p(f_p) + w_V \sum_{p,q} V_{p,q}(f_p, f_q) \qquad (1)$$

where $D_p(f_p)$ is a unary cost function for labeling node p with f_p (often referred as the data term), $V_{p,q}$ is a binary cost function defined on each edge of the graph (often referred as the smoothness term), and w_V is a weight that specifies the relative importance of these two costs. In order to apply a multi-label min-cut algorithm to our problem, we construct a graph where each node of the graph corresponds to a triangle in the 3D model. We then treat two triangles as being adjacent only when they share an edge in the 3D mesh. Multi-label min-cut algorithm can optimize this type of cost function among many other algorithms.

In our case, the unary cost (the data term) is computed as the distance between the triangle and the plane corresponding to the label f. For each triangle p, we compute its center $\mathbf{c}_p$ and normal $\mathbf{n}_p$. The unary cost $D_p(f)$ is a weighted sum of the difference between the normals of triangle p and the plane f_p and the distance from the centroid of the triangle p to the plane f_p. The corresponding expression is defined as

follows:

$$D_p(f) = A_p(|\mathbf{n}_f - \mathbf{n}_p|_{L1} + w_n|d_f - \mathbf{c}_p \cdot \mathbf{n}_f|), \quad (2)$$

where w_n is a user-defined weight, A_p is the area of the triangle. In practice, we set w_n to 10. The binary edge cost function (the smoothness term) is defined as:

$$V_{p,q}(f_q, f_q) = \begin{cases} 0, & f_p = f_q \\ A_{pq}(4 - |\mathbf{n}_p - \mathbf{n}_q|_{L1}), & f_p \neq f_q. \end{cases} \quad (3)$$

A_{pq} is the average area of triangles p and q. Splitting two neighboring triangles into different clusters incurs a cost that decreases linearly with respect to the difference in normals.

User-specified Mesh Saliency: In our system, we would like to employ as few polygons as necessary to represent a 3D model in order to minimize the assembly time. Therefore, it is highly desirable to approximate the original mesh better in visually salient areas. However, automatically detecting these areas is still an unsolved problem. We allow a user to manually specify the salient areas of the mesh (e.g., by painting over the mesh texture map). If the user specifies a weight map w_t for each triangle, it is added to the smoothness term:

$$V_{p,q}(f_q, f_q) = \begin{cases} 0, & f_p = f_q \\ A_{pq}(4 - |\mathbf{n}_p - \mathbf{n}_q|_{L1} + w_t), & f_p \neq f_q. \end{cases} \quad (4)$$

The value of the weight w_t needs to be low in salient areas and high in non-salient areas.

Updating Plane Equations: After computing the label for each triangle, we update the plane equation for each label as well. The normal $\mathbf{n}_f$ for plane/label f is computed as a weighted average of normals for all triangles assigned with label f. The weight corresponds to the area of each triangle. Similarly, we compute the weighed average for all triangle centers assigned to label f. We use it to determine a new value for the constant d_f.

3.1.2. Mesh Deformation

In the continuous step, our goal is to deform the mesh, adjusting the position of each vertex. Unlike previous work on mesh morphing [LDSS99, GSL*99, SP04], the target mesh is unknown. Our main focus is to solve for the new position of each vertex $\mathbf{v}$ according to four objectives. First, the deformed mesh should be similar to the original mesh. Second, each cluster should become more planar. Third, we favor translation and try to preserve the orientation and scale of each triangle. Lastly and optionally, the mesh can be smoothed. We create a linear system using these criteria similarly to deformation gradients [SP04] and solve the system for new vertex positions. We used a standard implementation of conjugate gradient descent as it is sufficiently fast.

After solving for the new positions, we move each vertex to its new position sequentially. When we move a vertex to its new position, we detect faces that self-intersect using the method by Bridson et al. [BFA02]. If the new vertex position introduces intersections, we reverse its position. Next, we provide the implementation details for each objective:

1. **Mesh Similarity:**
   For each vertex $\mathbf{v}$ and its original position $\mathbf{v}_0$, we set the objective to be:
   $$\mathbf{v} = \mathbf{v}_0. \quad (5)$$
   We set the weight of this objective the same for all vertices.
2. **Cluster Flattening:**
   The normal of each cluster $\mathbf{n}$ is computed as an area-weighted average of triangle normals. For each triangle $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ in the cluster, we add the objective that two of its edges should lie in a plane perpendicular to the average normal, that is
   $$(\mathbf{v}_2 - \mathbf{v}_1) \cdot \mathbf{n} = 0, (\mathbf{v}_3 - \mathbf{v}_1) \cdot \mathbf{n} = 0. \quad (6)$$
   We multiply the equations corresponding to this objective by the weight w_p. The value of this weight increases with each iteration of the algorithm. Intuitively, at the beginning we would like to make small changes to the mesh to facilitate changes in label assignments. Later, these assignments become more fixed and we would like to focus on planarity objective.
3. **Transformation Objective:**
   The algorithm avoids texture coordinate distortion by using constraints on scaling and rotation transformations of triangles. In order to represent non-translational transformation of a triangle $(\mathbf{v}'_1, \mathbf{v}'_2, \mathbf{v}'_3)$, we follow the practice of [SP04] and use an auxiliary vertex $\mathbf{v}'_4$ defined as:
   $$\mathbf{v}'_4 = \mathbf{v}'_1 + \frac{\mathbf{n}'}{|\mathbf{n}'|^{1/2}}, \quad (7)$$
   where
   $$\mathbf{n}' = (\mathbf{v}'_2 - \mathbf{v}'_1) \times (\mathbf{v}'_3 - \mathbf{v}'_1).$$
   Here $\mathbf{v}'_i$ is a known vertex position from the previous iteration. Let $\mathbf{T}$ be the transformation on the triangle $(\mathbf{v}'_1, \mathbf{v}'_2, \mathbf{v}'_3)$, $\mathbf{v}_i = \mathbf{T}(\mathbf{v}'_i)$, and
   $$\mathbf{V} = \begin{bmatrix} \mathbf{v}_2 - \mathbf{v}_1 \\ \mathbf{v}_3 - \mathbf{v}_1 \\ \mathbf{v}_4 - \mathbf{v}_1 \end{bmatrix}, \mathbf{V}' = \begin{bmatrix} \mathbf{v}'_2 - \mathbf{v}'_1 \\ \mathbf{v}'_3 - \mathbf{v}'_1 \\ \mathbf{v}'_4 - \mathbf{v}'_1 \end{bmatrix},$$
   Then the non-translational component $\hat{\mathbf{T}}$of the transformation satisfies $\hat{\mathbf{T}}\mathbf{V}' = \mathbf{V}$. Assuming no triangles are degenerate, we can invert $\mathbf{V}'$ to get $\hat{\mathbf{T}} = \mathbf{V}\mathbf{V}'^{-1}$. Therefore, the transformation can be written as a linear combination of new vertex positions. A linear constraint for the transformation therefore would be translated into a linear constraint for new vertex positions. We add the objective:
   $$\hat{\mathbf{T}} = \mathbf{I}, \quad (8)$$
   which states that the non-translational component of the transformation should be similar to the identity matrix.

4. **Smoothness** For each vertex $\mathbf{v}$, the smoothness of the mesh can be enforced by moving the vertex closer to the average position of its neighbors $N(\mathbf{v})$, that is:

$$\mathbf{v} - \frac{1}{|N(\mathbf{v})|} \sum_{\mathbf{u} \in N(\mathbf{v})} \mathbf{u} = 0. \quad (9)$$

3.2. Computing Fabricable Components

The algorithm presented in the previous section deforms the original mesh such that the resulting mesh has at most k polygonal clusters. Next, our goal is to convert this intermediate model to a set of 2D components that can be manufactured and assembled into a physical model. In order to accomplish this, we have to perform the following tasks: (1) extract 2D polygons from triangle clusters; (2) truncate polygon boundaries based on material thickness; (3) create textures for the physical model; and (4) create connectors that join neighboring polygonal components. We will describe all these tasks in sequence.

3.2.1. Converting Triangle Clusters to Polygons

After the deformation step, the intermediate result is still triangular and each face is labeled. For each label, we compute a polygon that is a union of all the triangles with this label. We do this by tracing the boundaries of each polygon. We observe that an edge in a triangle is also a polygon edge only if it is adjacent to two triangles with different labels. We find all these edges and join them to form a polygon. We then remove all vertices that are not at the intersection of three or more clusters. Since the deformation step flattens each cluster up to very small numerical error, this algorithm is sufficient and robust to generate all polygon edges even in the case where the input model contains holes.

3.2.2. Truncating Polygon Boundaries

So far we have assumed that polygons are infinitesimally thin. However, physical materials have a nonzero thickness and therefore, we need to truncate the boundaries of some polygons in order to fit adjacent polygons. Figure 3 shows two cases where truncating is necessary. The first case arises when two planes meet at a convex obtuse angle θ. Then one plane has to be truncated by $t \cdot \sin\theta$, where t is the material thickness. In the second case where two planes meet at a convex acute angle, both planes have to be truncated, one by $t \cdot \cot\theta$ and the second by $t \cdot \cot\frac{\theta}{2}$. Note that this choice preserves the outline of the model. The same truncation can be applied whether two planes share a vertex or an edge. A shared vertex can be treated as an edge with zero length in the direction perpendicular to both plane normals.

3.2.3. Computing Model Textures

Unlike previous approaches (e.g., [CSAD04]) that need to remesh the model, our algorithm preserves all triangle faces and texture coordinates during mesh processing. Moreover, our deformation process attempts to minimize rotation and scaling of the deformed triangles. Therefore, we can generate a good-quality texture for each polygon by reusing the original texture coordinates. There are many algorithms for preserving texture coordinates during mesh simplification [COM98, LSS*98, SSGH01, KLS03]. They offer great inspirations for a more sophisticated algorithm to project texture onto a plane with low distortion.

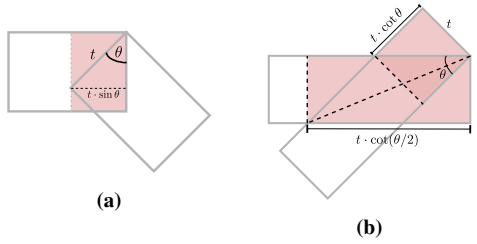


Figure 3: *Side view of adjoined planes.* ***Left:*** *two planes meet at an obtuse angle. One of the plane needs $t \cdot \sin\theta$ of material to be removed along the boundary, where t is the thickness.* ***Right:*** *two planes meet at an acute angle. Both planes need to be truncated. One needs to cut $t \cdot \cot\theta$ and the other needs to cut $t \cdot \cot\frac{\theta}{2}$.*

3.2.4. Connectors

There are several possible methods for connecting two planar pieces. The choice of the method depends on the type of material, its size, strength, connection angle, and aesthetic requirements. We describe two methods that can be employed within the same 2D cutting process and material used to fabricate the polygonal pieces. The first design uses an additional connector element (shown in Figure 4) that attaches to two planar polygons that share an edge. We generate a connector template that is parameterized by the angle between the polygons and the material thickness. In order to place a connector, we first choose a few positions along the edge between two adjacent planes. We then ensure that the connector does not collide with other connectors and other planes. If it does not, then we carve out rectangular slots in both polygons and we add the corresponding connector to the model. We oversize all connectors by a small amount to create an interference fit that holds them in place in the polygons. However, this type of connector cannot be used when the angle between the planes is less than 90° – the two pieces cannot be inserted into the slots. In this case, we use a different connection method – teeth are carved on the edges between two polygonal pieces (shown in Figure 5). These teeth are also oversized to create friction that holds two polygonal pieces together. We also use teeth when a piece is too small to fit a connector.

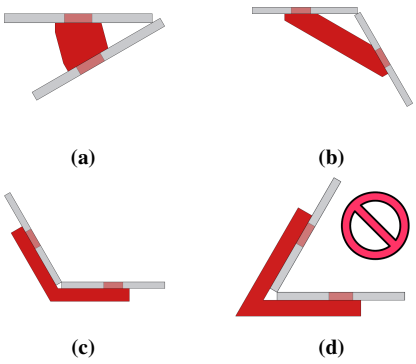


Figure 4: *Types of connectors based on the angle between two planes. Figure (a), (b), and (c) show the connector for acute angles, obtuse angles and concave intersections. Figure (d) shows a concave connector with an acute angle that cannot be assembled.*

4. Results

We have tested our pipeline on a number of different models. In this section, we analyze performance of our system with respect to different parameters. We compare our system to two previous systems, the Variational Shape Approximation [CSAD04] and the Quadratic Error Metric [GH97]. Next, we discuss the physical models we have fabricated. Finally, we discuss limitations of our current system.

4.1. Parameter Settings

The algorithm for computing polygonal deformation takes as input the number of desired polygons k. We experiment with k ranging from 30 to 90. While it is possible to use values larger than 100, it would take a long time to assemble the resulting models. On the other hand, it is difficult to represent models with fewer than 30 planes. We analyze the performance of the algorithm as a function of desired polygons k. We show the results for two different models in Figure 7 (top). Our algorithm iterates between the mesh segmentation step and mesh deformation step. We have used 50 iterations for all our experiments. The mesh segmentation has one parameter, w_V, which controls how smooth the segmentation is. In our experiments we have set its value to around 0.05. In the mesh deformation step, we can set relative importance of four different constraints: the mesh similarity constraint, the

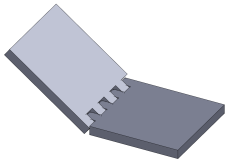


Figure 5: *Toothed connector. In cases the connectors in Figure 4 cannot be used, we generate teeth between two pieces.*

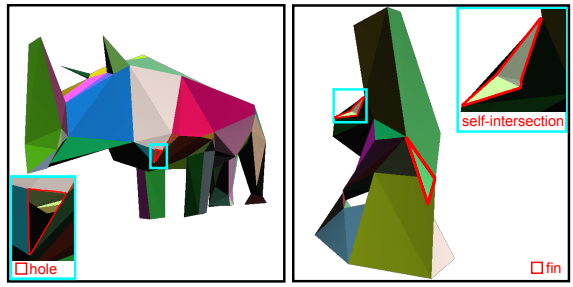


Figure 6: *Examples of a hole, a fin and a self-intersection produced by VSA.*

cluster flattening constraint, the transformation constraint, and the smoothness constraint. We only set the importance of the cluster flattening constraint w_p. We increase this parameter with each iteration of the algorithm, starting with 1 and ending at 2000. We never treat planarity as a hard constraint. However, after the final iterations, each cluster is flat up to negligible numerical errors as shown in Figure 10. We never found this error to be a problem during assembly.

4.2. Comparison to Previous Work

We compared our algorithm against the Quadratic Error Metric [GH97] and the Variational Shape Approximation [CSAD04] using Hausdorff distance [CMS98] as an error metric. Our method produces better models both visually and numerically. VSA does not guarantee that each polygon is planar because enforcing flatness is absent from the algorithm. Therefore, VSA has a slight advantage, as it can keep vertices closer to the original surface. We refer our readers to Cutler et al. [CW07] for difficulties in enforcing planarity of polygons. Table 1 shows errors for the 6 models we used in comparison. Our algorithm performed the best in most cases.

Since none of the algorithms optimize for Hausdorff distance, increasing the number of polygons does not always decrease the Hausdorff distance. When the number of polygons increases, the visual quality of a model is much better. However, Hausdorff distance only measures the distance of the farthest point, which sometimes does not affect the visual quality much. Because of the limitation of Hausdorff distance, we also compare the visual results of two models in Figure 7. Note that we can preserve higher details in the models compared previous works even without user-specified salient areas.

It is worth noting that QEM and VSA do not always produce manifolds. QEM produced fins for some models during our experiments (see Figure 7). Figure 6 shows examples of VSA producing fins, holes and self-intersections. In some cases, VSA produced a fractured triangle soup and then stopped. We were unable to obtain a mesh in those cases and left blanks in Table 1.

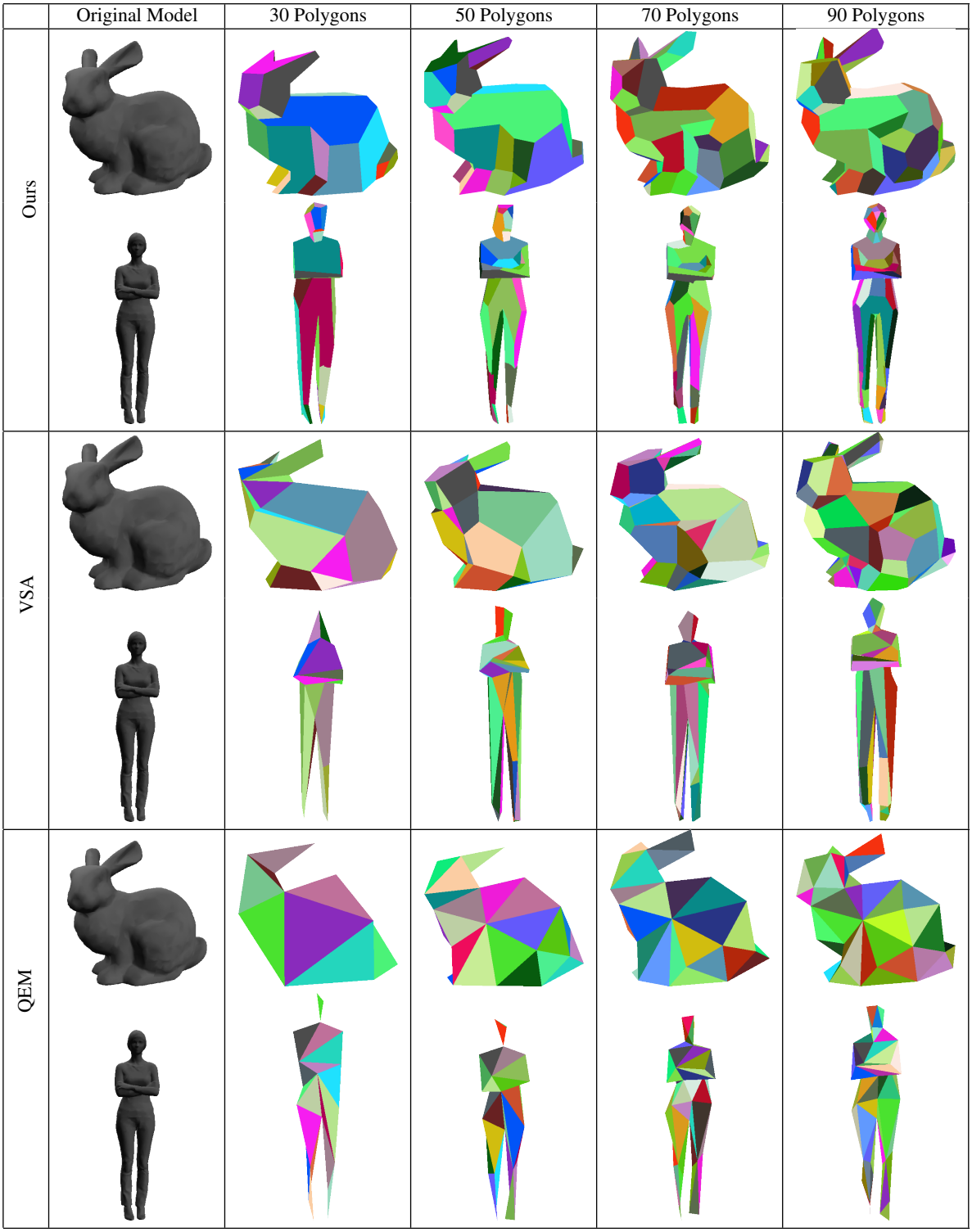


Figure 7: *Comparison of outputs with different number of polygons and methods. The top row shows the results from our method. The middle row compares the results from VSA. The bottom row shows the results from QEM.*

Model	Method	30	50	70	90
Bird	Ours	**4.84**	**3.26**	**1.74**	1.94
	VSA	5.16	3.93	1.94	**1.82**
	QEM	6.97	6.84	3.06	3.06
Bunny	Ours	**4.66**	**4.36**	**2.84**	**1.96**
	VSA	5.90	4.76	3.61	3.37
	QEM	8.25	6.73	5.19	5.19
Gnome	Ours	**6.70**	**4.68**	**2.88**	**2.71**
	VSA	7.02	5.84	4.39	4.94
	QEM	10.23	7.38	5.44	5.42
Human	Ours	**2.31**	**2.38**	**2.29**	**2.26**
	VSA	7.22	4.90	3.89	3.16
	QEM	6.87	7.24	6.52	4.03
Mask	Ours	**2.80**	**1.86**	1.72	**1.08**
	VSA	2.91	2.10	**1.64**	1.33
	QEM	5.45	3.90	3.17	2.22
Rhino	Ours	**7.20**	**8.05**	**4.81**	**3.61**
	VSA	N/A	N/A	N/A	4.35
	QEM	9.59	8.56	7.84	7.84

Table 1: *Comparison of different methods with different number of planar polygons. Our results are computed without user guidance for a fair comparison. In case of QEM, all polygons are triangles. The results are Hausdorff distances measured as a percentage of a diagonal of the bounding boxe for each model. In some cases, we were unable to obtain a result with VSA using code provided by the original authors.*

4.3. Running Times

The running times are quite short even though we are just using standard implementations of multi-label min-cut and conjugate gradient. We typically run the algorithm for models with 10K triangles. In this case, each iteration of the algorithm takes a few seconds with running times roughly equal between the segmentation and deformation steps. This results in running times of a few minutes for each model. Using highly-optimized implementations can further speed up our algorithm. The running times for different models are shown in Figure 10.

4.4. User-specified Saliency

The saliency map proves to be a valuable tool. It adds spatial adaption to the smoothness term in the mesh segmentation step. We show examples of using manually painted saliency maps on two different models in Figure 8. We compare the result of mesh deformation with and without the saliency map term when using the same number of polygons. Observe that when we do not specify the saliency map, the head of the bird and the face of the person are overly simplified. However, we preserve more features in these areas when specifying the appropriate saliency map.

While existing saliency detection algorithms [LVJ05], [PPT*11] can provide saliency maps for our algorithm, we find that painting a coarse saliency map is very simple and offers more control for the user. For example, the human feet contain as much high frequency variation as a human face. The user can specify that feet should use fewer polygons despite the amount of detail they contain.

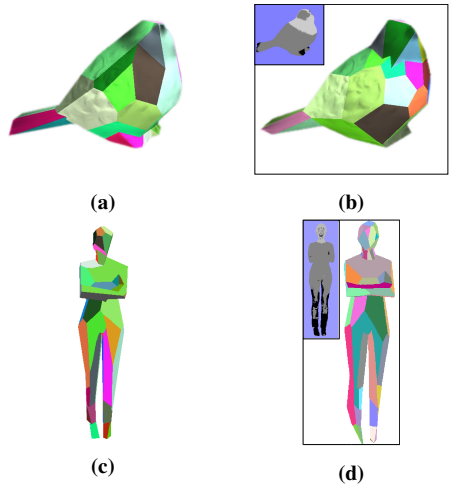


Figure 8: *Comparison of clustering with and without user painted saliency given the same number of clusters.* ***Left:*** *Clustering results without user guidance.* ***Right:*** *User guided clustering. The user painted higher saliency around the head and neck area and lower saliency around the legs and feet. Our algorithm accordingly assigned more planes to the head and neck while sacrificing details around the feet and legs.*

4.5. Manufactured Models

We have manufactured a number of different models using our system (shown in Figure 10). We have used balsa wood, basswood, and acrylic as our substrate materials. We have cut both the models and the connectors using the same computer-controlled laser cutter (we show the complete set of parts that are necessary to assemble one model in Figure 9). We have assembled all parts and connectors by hand, and then used an inkjet printer to print the corresponding textures. Next, the textures have been glued to the models. Alternatively, we can use a flatbed printer to print directly on the manufactured parts. Depending on the complexity of the model and the number of polygons, the fabrication process takes 1-8 hours. Overall, our models look very convincing, especially with the added textures. Moreover, they are inexpensive and relatively easy to make. Our models can be large (e.g., the length of the Rhino model is more than 75cm). In comparison, 3D printing similar models could be difficult and expensive.

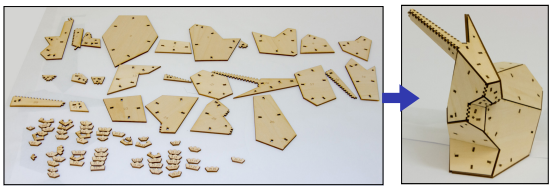


Figure 9: *Manufactured parts and assembled model. Parts were manufactured using a laser cutter with 1/8 inch thick basswood material. The cutting time was about 5 minutes, and the assembling process took roughly an hour and a half.*

4.6. Limitations

Our current system has a number of limitations. The input to the algorithm should be a connected mesh, ideally with nearly equal-area triangles and without self-intersections. Since the algorithm tries to preserve these properties, it does not work well with meshes that do not satisfy these properties. One promising alternative to our self-intersection handling is [HPSZ11]. While our algorithm works well with high-genus meshes, supporting high-genus meshes requires increasing the number of polygons in the output model. The algorithm also has problems with very thin features. Although it can simplify the meshes correctly, the thickness of the material makes it difficult to manufacture the final model. One solution to this problem is to directly incorporate the thickness of the material into the mesh deformation algorithm.

5. Conclusion and Future Work

Our method makes conversion of 3D textured meshes into physical counterparts inexpensive and practical. It decomposes an input mesh into planar polygons and planar connectors that can be easily fabricated using any computer-controlled 2D cutting machine. Then, the manufactured polygonal pieces can be easily assembled into final 3D models using the provided connectors. We believe that our proposed method is a good alternative to 3D printing, especially when the physical models are large. The resulting models can also be made from a variety of different materials. The use of texture improves the realism of the manufactured models, particularly when the deformed models have a low number of polygons.

There are several improvements that can be made to the existing system. Our current models use a low number of polygons, and assembly is not difficult because the polygons and connectors are engraved with appropriate IDs (using the laser cutter). For models with a larger number of primitives it would be necessary to automatically generate assembly instructions (e.g., similarly to the method presented by Agrawala et al. [APH*03]). For larger models it might be also necessary to generate an internal support structure (e.g., similar to Hildebrand et al. [HBA12]). The support structure would allow heavier materials to be used for the polygonal panels. Furthermore, the exterior polygonal panels could potentially be attached in an arbitrary order. Finally, the current method has difficulty handling thin features. One possible solution to solve this problem is to break the assumption that the resulting model is a closed surface, allowing the use of a single sheet of material to represent thin features.

6. Acknowledgements

We would like to thank the anonymous reviewers for their insightful suggestions, Ilya Baran for his helpful comments, and Pierre Alliez for kindly providing code and instructions for the Variational Shape Approximation. This work was partially supported by NSF grant IIS-1116296 and CCF-1138967.

References

[APH*03] AGRAWALA M., PHAN D., HEISER J., HAYMAKER J., KLINGNER J., HANRAHAN P., TVERSKY B.: Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics 22*, 3 (July 2003), 828–837. 9

[AV07] ARTHUR D., VASSILVITSKII S.: k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (2007), p. 1027ÍC1035. 3

[BFA02] BRIDSON R., FEDKIW R. P., ANDERSON J.: Robust treatment of collisions, contact, and friction for cloth animation. *ACM Transactions on Graphics 21*, 3 (July 2002), 594–603. 4

[CMO97] COHEN J., MANOCHA D., OLANO M.: Simplifying polygonal models using successive mappings. In *Proceedings of the 8th conference on Visualization '97* (Los Alamitos, CA, USA, 1997), VIS '97, IEEE Computer Society Press, pp. 395–ff. 2

[CMS98] CIGNONI P., MONTANI C., SCOPIGNO R.: A comparison of mesh simplification algorithms. *Computers Graphics 22*, 1 (1998), 37 – 54. 6

[COM98] COHEN J., OLANO M., MANOCHA D.: Appearance-preserving simplification. In *IN PROC. SIGGRAPHąę98* (1998), pp. 115–122. 5

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 905–914. 2, 5, 6

[CW07] CUTLER B., WHITING E.: Constrained planar remeshing for architecture. In *Proceedings of Graphics Interface 2007* (New York, NY, USA, 2007), GI '07, ACM, pp. 11–18. 6

[DDSD03] DÉCORET X., DURAND F., SILLION F. X., DORSEY J.: Billboard clouds for extreme model simplification. *ACM Transactions on Graphics 22*, 3 (July 2003), 689–696. 2

[DO07] DEMAINE E., O'ROURKE J.: *Geometric Folding Algorithms. Linkages, Origami, Polyhedra.* Cambridge University Press, 2007. 2

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *IN PROC. 29TH SIGGRAPH* (2002), pp. 355–361. 2

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 209–216. 2, 6

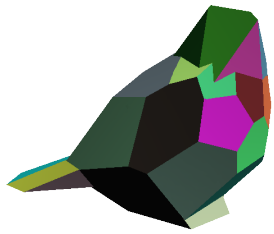
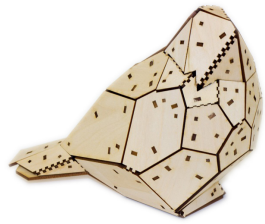
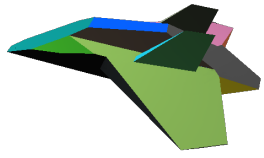
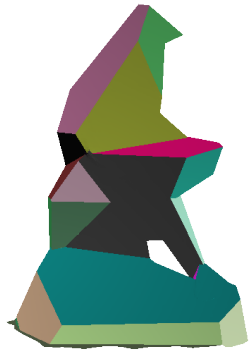
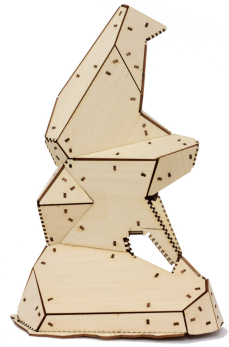
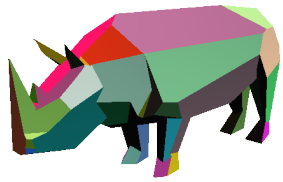
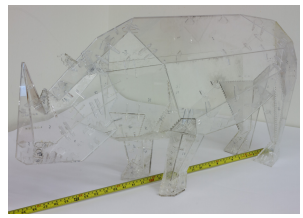
Input Model	Flatten Model	Fabricated	Textured
			
7,027 vertices/ 14,043 faces	45 clusters (17 min)	Basswood	13" tall, 3 Hours Fabrication
			
896 vertices/ 1,788 faces	30 clusters (1 min)	Balsa Wood	6" wide, 1 Hour Fabrication
			
4,989 vertices/ 9,902 faces	40 clusters (10 min)	Basswood	13" long, 3 Hours Fabrication
			
5,064 vertices/ 10,124 faces	85 clusters (12 min)	Acrylic	30" long, 8 Hours Fabrication

Figure 10: *Results of our algorithm on different models. These models have different geometric complexity and topology. We also tested our method on different types of materials and number of polygons.*

[Gla02] GLASSNER A.: Andrew glassner's notebook: Interactive pop-up card design, part i. *IEEE Computer Graphics & Applications 22*, 1 (Jan./Feb. 2002), 79–86. 2

[GSL*99] GREGORY A., STATE A., LIN M. C., MANOCHA D., LIVINGSTON M. A.: Interactive surface decomposition for polyhedral morphing. *The Visual Computer 15*, 9 (Dec. 1999), 453–470. 4

[HBA12] HILDEBRAND K., BICKEL B., ALEXA M.: crdbrd: Shape fabrication by sliding planar slices. In *Computer Graphics Forum (Eurographics 2012)* (2012), vol. 31. 3, 9

[HBLM11] HOLROYD M., BARAN I., LAWRENCE J., MATUSIK W.: Computing and fabricating multilayer models. *ACM Transactions on Graphics 30*, 6 (Dec. 2011), 187:1–187:8. 2

[Hop99] HOPPE H.: New quadric metric for simplifiying meshes with appearance attributes. In *Proceedings of the conference on Visualization '99: celebrating ten years* (Los Alamitos, CA, USA, 1999), VIS '99, IEEE Computer Society Press, pp. 59–66. 2

[HPSZ11] HARMON D., PANOZZO D., SORKINE O., ZORIN D.: Interference-aware geometric modeling. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 137:1–137:10. 9

[KLS03] KHODAKOVSKY A., LITKE N., SCHRÖDER P.: Globally smooth parameterizations with low distortion, 2003. 5

[LDSS99] LEE A. W. F., DOBKIN D., SWELDENS W., SCHRÖDER P.: Multiresolution mesh morphing. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 343–350. 4

[LJGH11] LI X.-Y., JU T., GU Y., HU S.-M.: A geometric study of V-style pop-ups: Theories and algorithms. *ACM Transactions on Graphics (Proc. SIGGRAPH) 30*, 4 (2011), 98:1–10. 3

[LSH*10] LI X.-Y., SHEN C.-H., HUANG S.-S., JU T., HU S.-M.: Popup: Automatic paper architectures from 3d models. *ACM Transactions on Graphics 29*, 4 (July 2010), 111:1–111:9. 2

[LSS*98] LEE A. W. F., SWELDENS W., SCHRÖDER P., COWSAR L., DOBKIN D.: Maps: Multiresolution adaptive parameterization of surfaces. *Computer Graphics Proceedings (SIGGRAPH 98)* (1998), 95–104. 5

[LVJ05] LEE C. H., VARSHNEY A., JACOBS D. W.: Mesh saliency. *ACM Trans. Graph. 24*, 3 (July 2005), 659–666. 8

[MGE06] MASSARWI F., GOTSMAN C., ELBER G.: Papercraft models using generalized cylinders. In *Proc. Pacific Conference on Computer Graphics and Applications* (2006), vol. 15. 2

[MS04] MITANI J., SUZUKI H.: Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Transactions on Graphics 23*, 3 (2004), 259–263. 2

[MSM11] MCCRAE J., SINGH K., MITRA N. J.: Slices: A shape-proxy based on planar sections. *ACM Transactions on Graphics 30*, 6 (2011). 3

[PPT*11] PANOZZO D., PUPPO E., TARINI M., PIETRONI N., CIGNONI P.: Automatic construction of quad-based subdivision surfaces using fitmaps. *Visualization and Computer Graphics, IEEE Transactions on 17*, 10 (oct. 2011), 1510 –1520. 8

[SLMI11] SAUL G., LAU M., MITANI J., IGARASHI T.: SketchChair: An all-in-one chair design system for end users. In *Proc. Fifth International Conference on Tangible, Embedded, and Embodied Interaction* (2011), ACM, pp. 73–80. 3

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 399–405. 4

[SSGH01] SANDER P. V., SNYDER J., GORTLER S. J., HOPPE H.: Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 409–416. 5

[STL06] SHATZ I., TAL A., LEIFMAN G.: Paper craft models from meshes. *The Visual Computer 22*, 9-10 (2006), 825–834. 2

[Tac09] TACHI T.: Origamizing polyhedral surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 2 (2009), 298–311. 2

[VK04] V. KOLMOGOROV R.: What energy functions can be minimized via graph cuts? *IEEE TPAMI 26*, 2 (Feb. 2004), 147–159. 3

[YB01] Y. BOYKOV O. VEKSLER R.: Efficient approximate energy minimization via graph cuts. *IEEE TPAMI 20*, 12 (Nov. 2001), 1222–1239. 3

[YB04] Y. BOYKOV V. K.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE TPAMI 26*, 9 (Sept. 2004), 1124–1137. 3