STAG: Smart Tools and Applications in Graphics (2018)
M. Livesu, G. Pintore and A. Signoroni (Editors)

Simplification of Shapes for Fabrication with V-Groove Milling Tools

Alessandro Muntoni[1], Andreas Scalas[2], Stefano Nuvoli[3], Riccardo Scateni[3]
[1]VCLab, ISTI-CNR, Pisa, [2]CNR-IMATI, Genova, [3]University of Cagliari

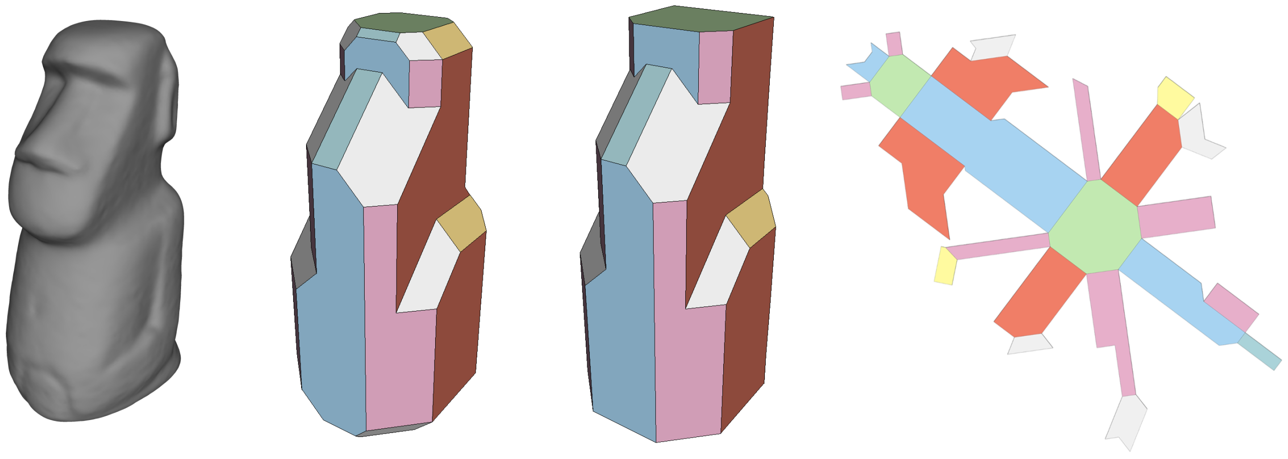


Figure 1: *From left to right: the original model; the result of the automatic simplification (Section 3); the clean model after spurious facets removal (Section 4); its unfolding for fabrication using CNC milling machines.*

Abstract
We introduce here a pipeline for simplifying digital 3D shapes with the aim of fabricating them using 2D polygonal flat parts. Our method generates shapes that, once unfolded, can be fabricated with CNC milling machines using special tools called V-Grooves. These tools make V-shaped furrows at given angles depending on the shape of the used tool. Milling the edges of each flat facet simplifies the manual assembly that consists only in folding the facets at the desired angle between the adjacent facets. Our method generates simplified shapes where every dihedral angle between adjacent facets belongs to a restricted set, thus making the assembly process quicker and more straightforward. Firstly, our method automatically computes a simplification of the model, iterating local changes on a triangle mesh generated by applying the Marching Cubes algorithm on the original mesh. The user performs a second manual simplification using a tool that removes spurious facets. Finally, we use a simple unfolding algorithm which flattens the polygonal facets onto the 2D plane, so that a CNC milling machine can fabricate it with a sheet of rigid material.

CCS Concepts
•***Computing methodologies*** $\rightarrow$ ***Mesh models; Shape analysis;*** *Shape modeling;*

1. Introduction

Fabrication of digital objects has found a considerable interest by researchers in computer graphics and geometry processing. 3D printers are the most commonly used machines to produce physical representations of digital objects. In the most typical scenario, the printer deposits a filament, layer by layer, that solidifies and forms the final result following a software-generated path. These machines permit printing arbitrarily complex geometries assuming to process the input mesh in order to compute supports and internal filling.

However, 3D printers are not the only technology that allows reproducing physical objects. Subtractive techniques, usable with CNC milling machines, follow a different philosophy compared to 3D printers: a numerically controlled tool moves in order to remove the material from a solid block, until reproducing the desired shape. These machines are mainly used to produce very regular objects in the mechanical engineering field. They can produce free-form geometries, but the characteristics of the used machine constrain them: they can have 3-, 4- or 5-axis meaning that the tool moves along the three principal axes plus one or two rotations axes. The choice of the machine is crucial to determine if the geometry can be fabricable. This aspect, along with difficulty on producing toolpaths especially for 4- and 5-axis machines, makes subtractive techniques still "immature" to use for fabrication of free-form geometries.

Another possible approach for obtaining physical representations of digital shapes is to apply some operations, such as deformations or simplifications, in such a way that the result becomes suitable for special fabrication techniques. There is much previous work focused on processing a digital shape in order to obtain a representation using blocks of material cut by a laser-cutter machine. Laser-cutters can perform straight or curved cuts on a block with limited thickness, using a high-power laser. These machines can be used to perform precise cuts on sheets of flat rigid materials, such as plywood or rigid paper, and produce 2D shapes, that can be combined to form the desired object.

As discussed in Section 2, many previous works aimed to create a simplified model to obtain a polygon mesh for fabrication purposes. All these works introduced a variety of joint systems used to allow an easy assembly process. Joint problems arise when two adjacent primitives (cut with machines like laser-cutters) should be joined manually along the joint edge at an arbitrary angle between them. A proper joint system allows the user to quickly obtain the desired arbitrary angle avoiding errors which can be propagated and that could cause an inadequate representation of the desired shape.

Our idea differentiates from this approach since we want to simplify our models so that there is no need to address the problem of designing joints necessary for the manual assembly process after the manufacturing. At the same time, we propose an approach that makes the manual assembly process more manageable and less error-prone. We plan to use CNC machines with V-Groove milling tools to carve our models on a sheet of rigid material (e.g., plywood, stiff paper, plexiglass). V-Groove (or V-Router) cutters are accessories for milling which allows engraving furrows on blocks of millable materials. These V-shaped milling tools enable to mill exact angles corresponding to the solid angle of the tool. There are plenty of available tools with different cutting angles on the market, with the most common being multiples of 30 and 45 degrees (Figure 2).

Once finished the carving process, it is possible to quickly fold the sheets, until reaching the desired dihedral angle along the edge (Figure 3).

To use this carve-and-fold strategy, we need to simplify our model so that all its internal dihedral angles belong to a restricted and well-defined set containing only angles that a V-Groove milling cutter can carve. As a direct consequence of this constraint, we have that each normal of the facets of the simplified model belong to a pre-defined set of values.

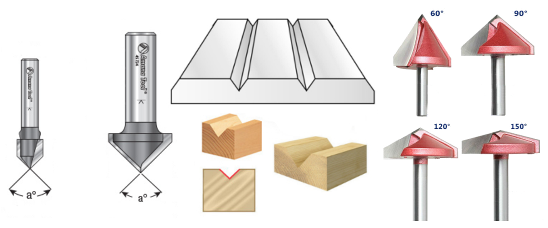
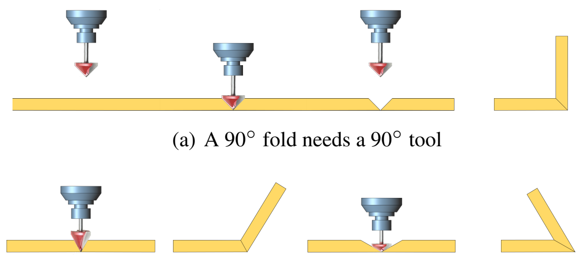


Figure 2: *V-Groove milling cutters can mill furrows with correct angles on rigid materials, and they are available with different milling angles on the market (courtesy of `www.toolstoday.com` and `www.aliexpress.com`).*



(a) A 90° fold needs a 90° tool

(b) A 120° fold needs a 60° tool (c) A 60° fold needs a 120° tool

Figure 3: *Different folds made using V-Groove milling tools.*

We propose a method whose output has **normals belonging to a restricted set**. We use Marching Cubes, working on a binary scalar field defined by the input model in order to obtain a final model with a reduced number of flat facets. We designed and developed a **GUI** that allows the user to select and remove all the unneeded facets obtained at the previous step. We introduce a novel **unfolding algorithm** suited for our purposes whose output is a 2D representation of the simplified model that is fabricable using a CNC milling machine with V-Groove tools.

2. State of the Art

Our work fits into the domain of geometry processing for digital fabrication [LSWW14, UBM15, LEM*17]. More precisely, we aim to simplify a model in order to fabricate it with polygonal portions of rigid flat materials, like plywood. This category of works relies massively on laser cut machines. We address a new challenge, using subtractive methodologies, specifically 3-axis CNC milling machines, with special cutting tools called V-Groove tools, that allows creating easy-to-fold joints. Below we review related simplification technologies.

Mesh simplification and approximation have been trend-topics in recent years.

In [CSAD04] the authors propose a method which produces an approximation of the surface using a variational approach. The result is a general-purpose and sound technique, applicable to a digital object of genus greater than zero. Unfortunately, the results do not satisfy the significant constraint we pose to our system: a small number of angles of fixed size.

In [ZLAK14] the authors try to solve the problem of representing an input surface using Zometool, a mathematical modeling system used in various areas. Their approach introduces an angle constraint: every Zometool node has a small set of possible directions, and therefore the angle formed along the edges belongs to a well-known restricted set. However, they do not need to simplify the input mesh with the goal of obtaining a small number of faces on their output mesh.

We compare ourselves more directly to methods of fabrication of 3D digital shapes using flat sheets of rigid materials. Richter and Alexa in [RA15] represent an input geometry using beams with a rectangular cross-section. They present some beams fabricable in wood, manufactured with laser cutter machines and assembled with particular joints (Figure 4). Schwartzburg and Pauly in [SP13] and Cignoni and colleagues in [CPMS14] present methods for the fabrication of 3D models interlocking planar pieces and strips. All these three works propose a solution to problems which are substantially different from ours: they do not aim to create a surface but to generate a set of strips or planar pieces that are joined together (with interlocks or another type of joints). We, instead, want to produce a foldable sheet which, once folded, is a 3D surface composed of few polygonal planar faces.

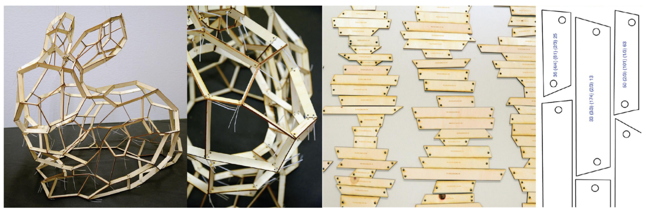


Figure 4: *Joints used in [RA15].*

Mitani and Suzuki solve a problem more strictly comparable to our in [MS04]. They simplify an input shape, producing a set of triangle strips that can be cut, folded and glued together to obtain a papercraft object. However, the primitives of the output are limited to triangles, and the assembly process is quite tricky, mostly suitable for papercraft lovers only.

Chen and colleagues in [CSaLM13] introduce a method which solves a problem very similar to ours. The authors approximate the input surface to a 3D mesh with a small number of planar polygonal faces for fabrication with CNC cutter machines. However, their assembly process is very complicated: it needs proper connectors (some examples are shown in figure 5), and it could take several hours for a single model. In our method, we introduce an additional constraint on the possible dihedral angles between adjacent faces, to simplify the assembly process.

Chen and Sass [CS16] try to solve the assembling problem creating a novel interlocking system with unique joining features (Figure 6). Unfortunately, their work focuses only on models produced with CAD tools and cannot apply to free-form shapes.

Song and colleagues in [SDW*16] combine laser cutting and 3D printing in order to produce a large object more cheaply and quickly. They first produce an internal base object composed of flat

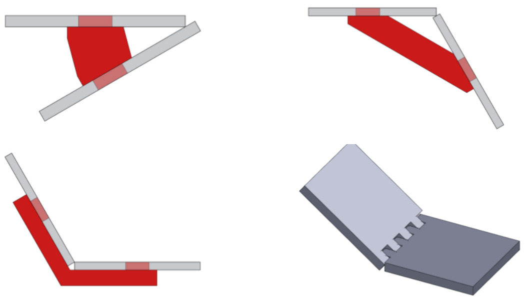


Figure 5: *Joints used in [CSaLM13].*

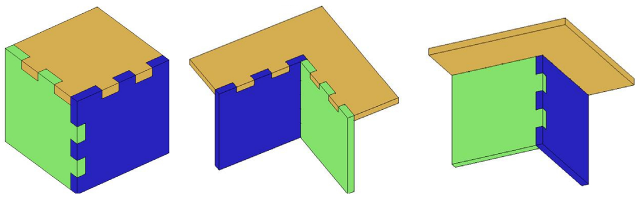


Figure 6: *Joints used in [CS16].*

pieces cut with a laser cutter machines, and then they attach thin 3D printed pieces in order to introduce the details of the input object. To assemble the internal structure, they propose joints designed explicitly for the reproduction of the desired angle between pieces (Figure 7). The goal of their work goes far beyond our purposes and, moreover, they do not assemble a single sheet with a carve-and-fold approach but use joints.

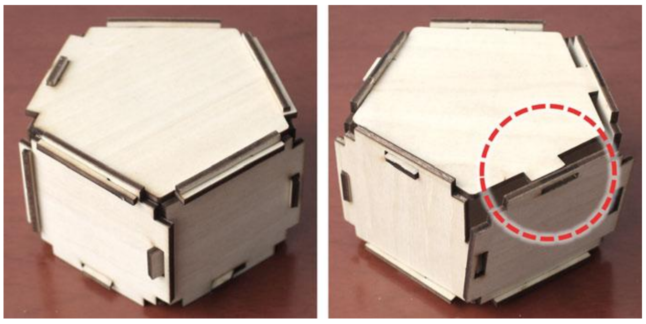


Figure 7: *Joints used in [SDW*16]*

Polycubes are simplified models in which every primitive is orthogonal to one of the three principal axes ([THCM04]). They resemble somehow to the models we aim to produce. There is a vast literature on automatic generation of polycubes (e.g., [LVS*13]), optimization of existing polycubes (e.g., [CLS16]) and use of polycubes for the generation of hexahedral meshes (e.g., [LMPS16]). However, polycubes are very simple 3D meshes, and they cannot resemble the original model when it has slanted faces with respect to the axes. Moreover, having only 90-degree dihedral angles on the final results would be very restrictive compared to the V-Groove tools for milling available on the market.

Finally, in [BCMP18] the authors analyze many related works which reflect the current state-of-the-art in the stylized fabrication of 3D shapes.

3. Surface approximation

The first pass of our pipeline relies upon the application of the Marching Cubes algorithm to the input shape. The Marching Cubes algorithm [LC87] generates a triangle mesh of an iso-surface starting from a scalar field. We generate the input scalar field of boolean values immersing the input shape in a regular lattice of cubes. By construction, each possible triangle normal of the resulting computed mesh belongs to a finite and well-defined set. As a direct consequence, all the possible dihedral angles between triangles are finite and known. We describe the set N of all the triangle normals which we can generate in the next paragraph.

The normals. A normalized face normal which belongs to the set N is a 3D vector where:

$$v_x, v_y, v_z \in \{+s, 0, -s\} \quad \text{with :} \quad s \in \{1, \frac{\sqrt{2}}{2}, \frac{\sqrt{3}}{3}\}.$$

Normals in the set N are 3D vectors divided into three categories as listed below.

1. Six vectors with **one** component different from 0:
$$[\pm1,0,0],[0,\pm1,0],[0,0,\pm1].$$
2. Twelve vectors with **two** components different from 0:
$$[\pm\frac{\sqrt{2}}{2},\pm\frac{\sqrt{2}}{2},0],[0,\pm\frac{\sqrt{2}}{2},\pm\frac{\sqrt{2}}{2}],[\pm\frac{\sqrt{2}}{2},0,\pm\frac{\sqrt{2}}{2}].$$
3. Eight vectors with three components different from 0:
$$[\pm\frac{\sqrt{3}}{3},\pm\frac{\sqrt{3}}{3},\pm\frac{\sqrt{3}}{3}].$$

Pairs of normals from this set form dihedral angles that are mostly of 30°, 45°, and their multiples. Since these angles correspond to available V-Groove tools, Marching Cubes perfectly suits to our purpose.

3.1. Initialization

We generate the lattice regularly subdividing the bounding box of the input mesh, taking care to scale it to have only integer lattice coordinates. The vertices of the lattice are labeled 1 if they are inside the surface and 0 if they are outside. We then run the discretized Marching Cubes algorithm [MSS94b] onto the lattice with an unambiguous look-up table [MSS94a] and a threshold included in the interval $(0-1)$, extracting an iso-surface mesh with a restricted set of facet normals. Even if we can merge all the adjacent triangles laying on the same plane, as shown in Figure 8, the resulting mesh is composed of a significant number of polygonal facets. The lattice spacing is a function of the average edge length of the mesh multiplied by a user-defined parameter, which determines the “granularity” of the final simplified mesh.

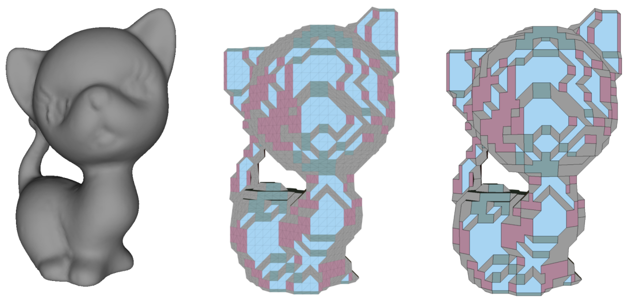


Figure 8: *From left to right: the input model, the triangle mesh obtained running DiscMC, and the polygon mesh resulting from merging the adjacent triangles laying on the same plane. The meshes are, respectively, composed of 3.628 triangles and 408 polygons. Triangles and polygons are color-coded according to their normals.*

3.2. Geometry

The main idea behind our method is to change signs in the regular lattice in order to obtain a smaller number of polygonal facets when re-running Marching Cubes on it. To this purpose, we introduce the concept of *Mask*.

Mask It is a set of adjacent cubes having a specified combination of signs on their vertices which generates a combination of adjacent triangles by Marching Cubes that we do not want to have in our output.

Every mask includes at least one set of *Points of Interest* representing the vertices which signs, once switched, simplify the resulting mesh. Applying the masks, we enlarge broad facets and make small facets disappear.

In Figure 9 there is an example of Mask. Four adjacent cubes with specified signs on its points compose it, and it has two sets of *Points of Interest* circled, respectively, in orange and cyan. Our algorithm selects one of these two sets and switches it in order to change the triangulation in two possible ways. We designed a basic set of 18 different types of masks that, when applying rotations to them, generate a total of 340 masks. We describe in detail all the types of masks in the Additional Material.

As we have seen, for some masks there are different ways to modify the local geometry using different sets of Points of Interest, but we can choose only one set. We give priority to larger polygonal facets, and to do this we need to keep track of the areas of each polygon. We avoid to repeat this computation any time we change the lattice, and consequently the mesh, introducing three data structures, linked each other, containing:

- the **lattice**;
- the **triangle mesh** obtained running Marching Cubes;
- the **polygonal set** obtained merging adjacent triangles.

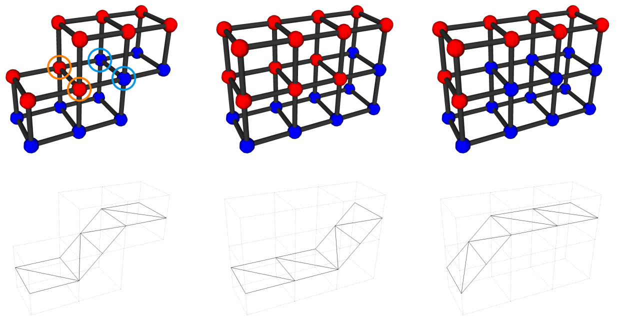


Figure 9: *In this example, we have a* Mask *(left) including a portion of the lattice which generates a surface including two facets with normal of type 1 adjacent to a facet with normal of type 2. Switching the signs of the points circled in cyan or orange, respectively, we obtain the configurations at the center or the right. We choose the set of vertices to switch which favors the enlargement of the broader adjacent facet.*

Each cube of the lattice, once traversed by Marching Cubes, generates triangles and we keep cross-links between them. Every triangle links to the polygonal facet containing it. Using the information in the data structures, we can efficiently choose the Points of Interest to switch. We choose the ones that link to the larger polygon, and we modify only the involved polygons whenever a sign switches. This approach guarantees that every switch of the sign is a local operation on the mesh, with time complexity $O(1)$.

Algorithm 1 Simplification Algorithm

```
    Input: set of masks $SM$, inital lattice $L$
    Output: simplified mesh $M$
 1: procedure SIMPLIFICATION($SM, L$)
 2:     $M \leftarrow$ MarchingCubes($L$)
 3:     Link all $M$'s triangles to their $L$'s cubes
 4:     $S \leftarrow$ ComputeSegmentation($M$)
 5:     Link all $S$'s polygonal facets to their $M$'s triangles
 6:     Push all $L$'s cubes in the queue $Q$
 7:     while $Q$ is not empty do
 8:         $c \leftarrow Q$.pop()
 9:         if $c$ and its neighborhood matches with $m \in SM$ then
10:             SWITCHSIGNES($c, m, L, S, M$)
11:             Push all modified cubes into $Q$
12:         end if
13:     end while
14:     return $M$
15: end procedure
16:
17: procedure SWITCHSIGNES($c, m, L, S, M$)
18:     $P \leftarrow$ getPointsOfInterest($L, S, c, m$)
19:     $\forall p \in P$, switch($p$) in $L$
20:     Update locally $M$ and $S$
21: end procedure
```

A high-level view of our approximation approach is given in Algorithm 1. We iteratively modify the geometry of the model by putting all the cubes of the lattice not having all the eight vertices equal in a queue. For each item in the queue, we first build all the possible local configuration comparable with masks, using its neighbors, and, then we perform a pattern matching against the set of *Masks*.

When we find a match, we switch the best set of points of interest, we update all the data structures, and we push back all the modified cubes in the queue. This choice allows us to identify a local set of triangles that was *shifted away* by the chosen mask. The process ends when the queue is empty, or when we detect a loop. In the latter case, the queue contains only the cubes generating loops. In our experiments, loops always involve local configurations of signs (and triangles) that with a sequence of sign switches leads to a configuration already seen in a previous iteration. Our solution is that any time we find a loop we pick the configuration with the less number of flat polygonal facets. It is worth to remind that the result of this approximation is a manifold and watertight mesh.

As shown in Figure 10, the output models of our approximation method have small facets that connect large orthogonal facets. Even when an input mesh presents some sharp 90° edges, the output of our approach presents small facets which acts as a junction between two orthogonal facets (see as an example Figure 16 a, e, f, k in Section 6). Using the Marching Cubes algorithm, these features cannot be avoided. However, one of the goals of our work is to ensure an easy assembly process and these small facets contribute to make it more challenging and error-prone. This problem led us to introduce the interactive manipulation step described in Section 4 which aims to remove these unwanted features. The goal is to find a result more suitable for our fabrication process and that is a better approximation when the input model has sharp edges.

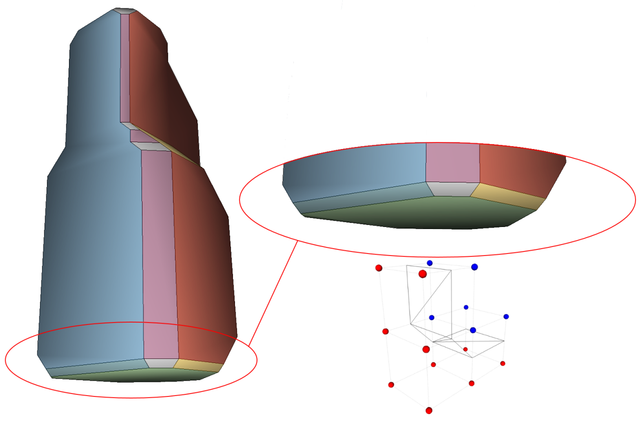


Figure 10: *Dihedral angles of 90° cannot be obtained using the Marching Cubes algorithm. There will always be small facets as junctions between orthogonal facets. In most of cases, these facets are not suitable for the proposed fabrication task and they could not contribute for a better approximation.*

Algorithm 2 Removes a facet and expands its adjacent facets to close the hole left by the deletion

```
    Input: the facet to be removed $f_0$
    Output: the simplified mesh $M$
 1: procedure FACETREMOVAL($f_0$)
 2:     Remove $f_0$
 3:     $b \leftarrow$ all the facets adjacent to $f_0$, in counterclockwise order.
 4:     $t \leftarrow$ CHOSESTARTINGTRIPLET($b$)
 5:     while $b$ contains more than two elements do
 6:         $r^1 \leftarrow$ getRay($t[1]$, $t[2]$)
 7:         $r^2 \leftarrow$ getRay($t[2]$, $t[3]$)
 8:         if $r^1$ and $r^2$ intersects then
 9:             if $b$ has exactly three elements then
10:                 $r^3 \leftarrow$ getRay($t[1]$, $t[3]$)
11:                 if $r^1, r^2, r^3$ intersect in the same point then
12:                     closeTriplet($t$)
13:                 end if
14:                 Empty $b$
15:             else
16:                 closeFacet($t[2], r^1, r^2$)
17:                 $t \leftarrow (t[1], t[3], b.next(t[3]))$
18:                 Remove $t[2]$ from $b$.
19:             end if
20:         else if |successive triplets with no intersection| < 3 then
21:             $t \leftarrow (t[2], t[3], b.next(t[3]))$
22:         else
23:             End procedure with error
24:         end if
25:     end while
26:     if $|b| = 2$ then
27:         Close the last couple of facets
28:     end if
29:     if |triplets with intersection| < 2 then
30:         End procedure with error
31:     end if
32: end procedure
33:
34: procedure CHOSESTARTINGTRIPLET($b$)
35:     for each $f_i$ in $b$ do
36:         $r_i^1 \leftarrow$ getRay($f_i, b.prev(f_i)$)
37:         $r_i^2 \leftarrow$ getRay($f_i, b.next(f_i)$)
38:         $iv_i \leftarrow$ intersection between $r_i^1$ and $r_i^2$
39:         $d \leftarrow$ ||$iv_i$ - the closest between $r_i^1$ and $r_i^2$ vertices||
40:         if $d$ is the shortest found distance then
41:             best $\leftarrow i$
42:         end if
43:     end for
44:     $t \leftarrow (b.prev(f_{best}), f_{best}, b.next(f_{best}))$
45:     return $t$
46: end procedure
```

4. User-driven simplification

We have seen that the automatic approximation step keeps unwanted facets in the model. Due to the highly variable nature of such facets, we set up a Graphical User Interface for allowing the user to decide which kinds of facet are *unwanted*. Our interest is twofold: in the immediate, we needed a tool to massage our mesh and obtain a better fabricable one; on the long run we have the aim of understanding which aspects lead such a selection and thus formulate automatic criteria. Such a tool requires to provide a speedy facet deletion and mesh restructuring to keep the resulting mesh manifold and watertight.

The facet deletion procedure is described in Algorithm 2. The first step removes the selected facet. All the adjacent facets are now unbounded. We take these facets and insert them into a counterclockwise-ordered circular buffer. Note that, if a facet shares only a vertex (and not an edge) with the selected facet, the facet will not be considered adjacent. Each pair of adjacent facets in this circular buffer generates a half-line (ray), obtained as the intersection of the planes lying on them, and its origin depends on the local configuration of the involved facets. If the two facets were adjacent also in the initial configuration, the origin of the half-line would be the vertex not incident to the deleted facet. If the two facets were not adjacent (as the green and red facets in Figure 11(a), they shared only a vertex, and that vertex will be the origin of the half-line. This is what is done inside the getRay function called in the algorithm.

Each triplet of facets defines a pair of adjacent half-lines. We identify all the triplets that generate half-lines intersecting, and we choose the one which intersection is closest to one of the vertices of the deleted facet. We do this inside the CHOSESTARTINGTRIPLET procedure. If less than two triplets generate intersections, this means we cannot close the surface and, thus, we cannot remove the chosen facet. The system reports this condition to the user.

If we can remove the facet, we add the selected intersection to the new mesh, and we close the facet at the center of the triplet prolonging its two edges which meet at the intersection point (expandFacet). We then remove the facet from the circular buffer, and the two external facets of triplet become adjacent, generating a new half-line with origin in the new vertex. The procedure iterates on the triplets until completion. The process behind the deletion of a single facet can be better understood looking at Figure 11.

There are three termination condition:

- there are only two facets left in the buffer;
- the last three (or more) facets in the buffer identifies lines intersecting on a single point;
- there are three successive triplets having no intersection.

Once the user select the facet, the system immediately outputs one of two possible results:

- the closed surface without the undesired facet, or
- an error message which communicates to the user that it is not possible to close the surface due to the local configuration of the adjacent faces.

Our algorithm works correctly on facets with an entirely convex or entirely concave neighborhood, and when the intersections do not involve facets that are not adjacent to the selected one. This last case

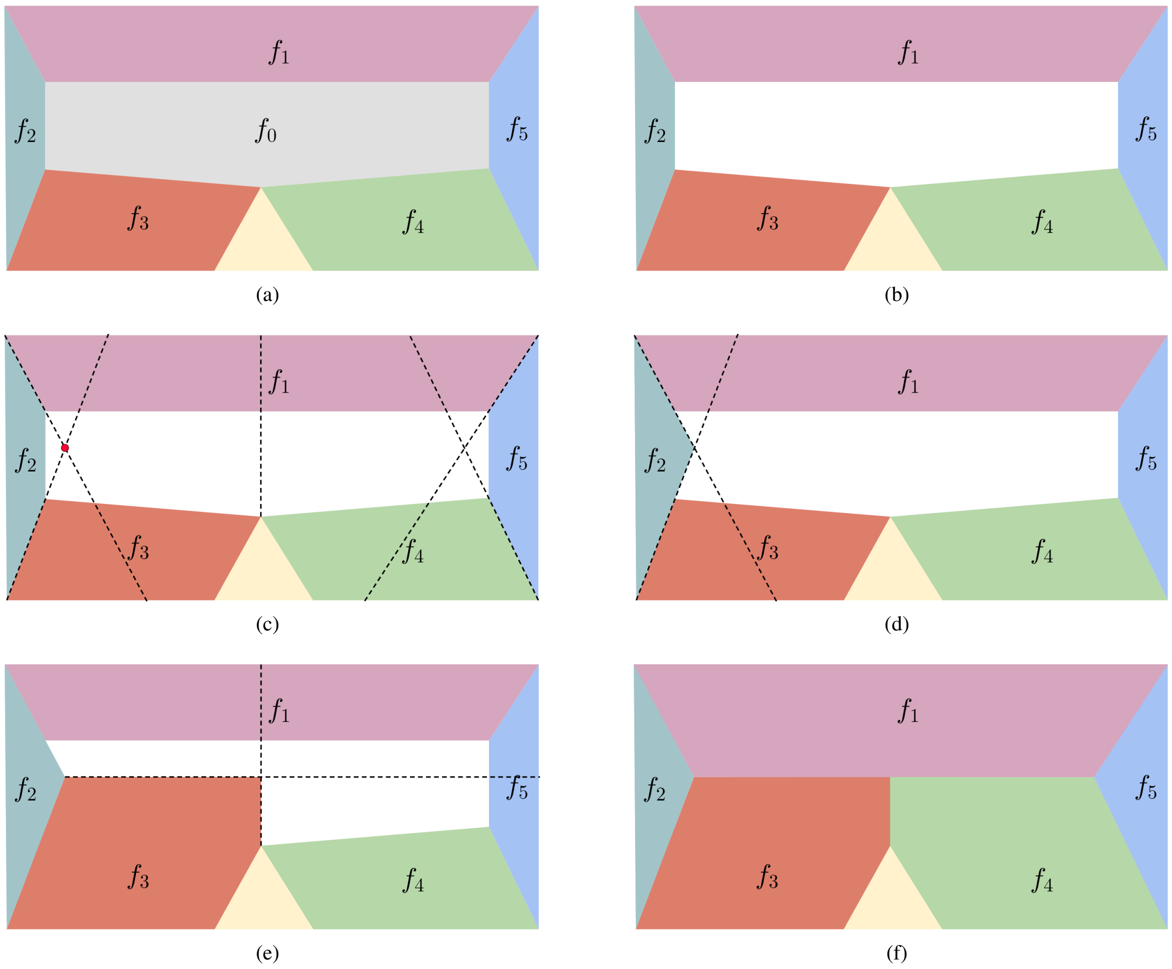


Figure 11: *Deletion of a facet. After removing f_0, we put all its adjacent facets in a circular buffer. We begin consuming facets in the buffer selecting the triplet f_1, f_2, f_3 since the intersection marked in red in (c) is the closest one. We reshape the central facet of the triplet (f_2) adding the adjacent portion of the canceled facet, and we remove it from the buffer (d). The process is iterated for the next nearest intersection until we assign all the portions of the canceled face and, thus, the surface is closed (f).*

is complicated to manage due to the high variety of possibilities that can happen. It is still an open problem for us, and we plan to tackle it in the future. In the current application, we show an error message. We show an example of facets deleted on the bottom of the *Moai* using this approach in Figure 12.

The user can also select multiple adjacent facets and delete them in a single step. This feature allows the simplification of shapes having local configurations in which a facet has two adjacent facets with the same normal but lying on different planes. In this case, deleting only one facet would be impossible. The deletion algorithm used is the same, and we show an example in Figure 13.

5. Unfolding the shape

The last step in our pipeline is the unfolding of the shape to be able to carve the furrows on a planar surface. There are three types of unfoldings: edge-unfolding, vertex-unfolding, and general unfolding [DO08]. We perform the edge-unfolding of a polyhedron which we cut along its edges and flatten into the plane in such a way that each facet preserves the distances among its components. We are looking for the *net* of a polyhedron, an edge-unfolding of a given 3D shape which forms a simply connected polygon that has no overlapping edges. In figure 14 we show the nets obtained for the two versions of *Moai* before and after the user-driven simplification.

A 3D shape can be flattened onto the plane if and only if every vertex has at least one incident cut edge, that is an edge where the polyhedron is cut along when unfolded [DO08]. These edges are a spanning tree in the 1-skeleton of the unfolded shape, that is the graph formed by its vertices and edges. Thus each net of a polyhedron has a corresponding distinct spanning tree of the graph. There is a one-to-one correspondence between the spanning trees in the 1-skeleton and the spanning trees in the dual graph (formed by facets and edges of the shape). The edges in a spanning tree of the dual graph represent the edges connecting the flattened polygonal

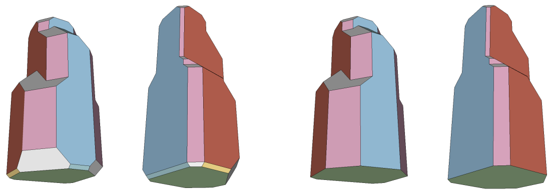


Figure 12: *The* Moai *before (a) and after (b) the removal of some facets from its bottom with our user-driven method. The quality of the approximation does not change while the complexity of the model decreases significantly.*

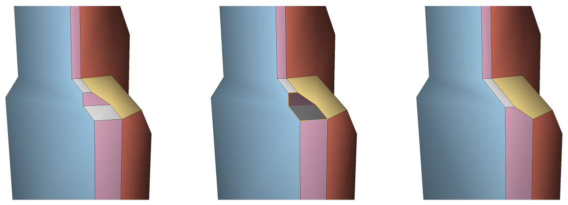


Figure 13: *Selection and deletion of two adjacent facets. The facets cannot be deleted singularly due to the adjacency of two facets with the same normal but lying on different planes. If selected together, we can close the surface extending the edges of the adjacent facets on the border of the two deleted facets.*

facets in the net. In other words, they are are the edges that the edge-unfolding process does not cut.

Unfortunately, it is not always possible to obtain an edge-unfolding to a simple non-overlapping polygon. There does not exist an efficient algorithm for determining whether or not a 3D shape has a net since it is a combinatorial problem. Even the existence of a net for one of the most straightforward categories of polyhedra, the convex ones, has been an open problem since Shephard explicitly posed in 1975 [She75].

For our purposes we do not necessarily need an edge-unfolding to a single net; hence we can divide the 3D shape into components, each of which unfolds to simple polygons (figure 15). Once manufacturing each piece separately, it is possible to glue them together to reproduce the original shape. Given a polygon mesh $M = (V, F)$, we are looking for segmentation into the fewest number n of disjoint components C_i which we can unfold in a single piece with no overlaps. More formally, $\forall i, j \in \{1, ..., n\}$ such that $i \neq j$ it holds that:

$$C_i \subseteq F, \quad C_i \neq \emptyset, \quad \bigcup_{i=1}^{n} C_i = F, \quad C_i \cap C_j = \emptyset$$

Additionally, if $|C_i| > 1$ it holds that for each facet $f \in C_i$ there exists at least a distinct facet $g \in C_i$ such that f and g are adjacent in the polygon mesh M. In other words, there exists a spanning tree of the subgraph of the dual graph composed by the facets of C_i that represents an unfolding to a simple connected polygon.

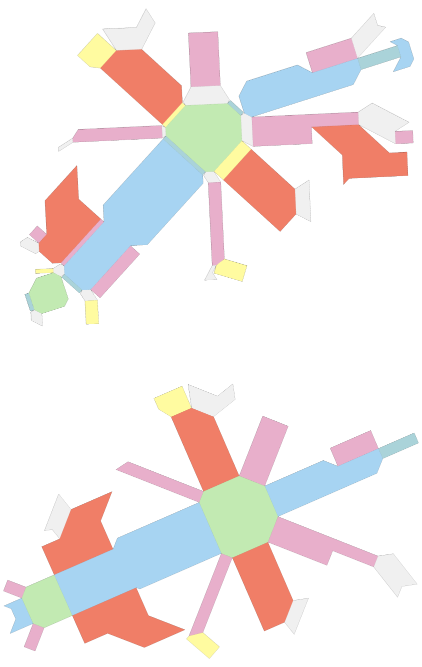


Figure 14: *The two versions of* Moai *shown in Figure 1 (before and after the user-driven simplification) unfolded. Top, before simplification; bottom, after user editing.*

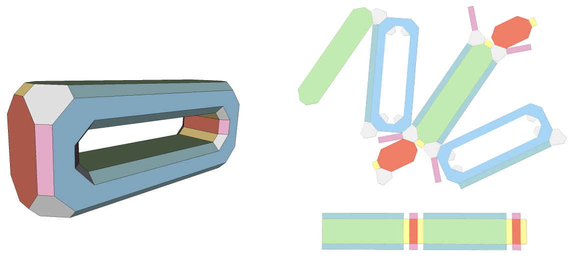


Figure 15: *A polygon mesh for which a net does not exist (a) and its unfolding in two components (b).*

A segmentation for which all the components are unfoldable without overlap always exists: the polygonal facets of the shape are simple polygons. The extreme case is each disjoint piece containing a single polygon which can unfold without any overlap. We look for a solution with the lowest number n of components. Unfortunately, we cannot determine the fewest number of parts that are unfoldable to a net. Only for convex polyhedra, an upper-bound exists [Pin07].

We use a heuristic to obtain a low number of components. Our idea is simple: given a polygon mesh, we incrementally flatten onto the plane the highest number of polygonal facets that form a simply connected polygon. The heuristic works as follows:

Model	# FAS	# FUS	Red. (%)
Abstract Sculpture	108	73	32.4%
Bimba	115	72	37.4%
BU	128	91	28.9%
Duck	41	16	61.0%
Egyptian Statue	293	99	66.2%
Fandisk	189	45	76.2%
Gentildonna	188	78	58.5%
Kitten	242	127	47.5%
Max Plank	67	36	46.3%
Moai	43	23	46.5%
Pensatore	113	88	22.1%
Sphynx	118	75	36.4%

Table 1: *Number of polygonal flat facets in the approximate models (# FAS) and after the user-driven intervention (# FUS). The third column lists the percentage of reduction, in number of facets, after the user interaction.*

1. Pick a seed facet, its perimeter is the first boundary $\mathcal{B}$ of the current unfolding $\mathcal{U}$;
2. Pick one of neighbour facets on $\mathcal{B}$, say f;
3. If f causes no overlap expand $\mathcal{U}$ to include f and update $\mathcal{B}$;
4. Go to step 2.

We stop when it is no longer possible to expand $\mathcal{U}$, and we obtain a first component $C_1 \subseteq F$. If $C_1 = F$ we have finished. Otherwise, we pick another seed facet among the remaining ones, and we iterate the process. To pick the neighbor facet, we follow a breadth-first search approach. The seed facet is always the largest convex one (if it exists). This method has the advantage of spreading in different directions the growing polygon, causing relatively low overlaps and producing unfoldings which are usually fitting in a rectangular sheet with less possible scraps.

At each incremental step of the algorithm, we need to efficiently test whether or not the edges of the current candidate facet are intersecting with an edge in the current unfolding. We use a data structure which enables us to perform fast queries for 2D segment intersections and to add new elements efficiently. We use an auto-balancing axis-aligned bounding box tree (AABB tree) that adapts very well to dynamic contexts in which the data structure is continually changing, and the objects are not often colliding [JTT01].

6. Results

We report several results obtained with our method in Figure 16. For every model we shown the result obtained by the surface approximation in the center column, and after the user-driven simplification in the right column. The number of polygonal facets of every result is reported in Table 1. In Figure 17 we show some unfolded model obtained with our method.

The approximation proposed in Section 3, as we have explained, is strongly based on Marching Cubes and therefore, our method guarantees all the properties that Marching Cubes guarantees on its outputs: every presented result is a water-tight 2-manifold mesh, with polygonal flat facets with normals that belongs to a restricted and well-known set of normals. The method is very quick, we do not report the times required for every approximate model because they are always around ten seconds.

The user-driven simplification tool described in Section 4 is quite simple to use for an expert user, and it required about 5 minutes to produce every presented result. However, the simplification time is only due to the user navigation: the most significant part of the time has been spent searching for the undesired facets, while the elimination and the expansion of the neighbors (or printing the error message) is instantaneous.

The unfolding method described in Section 5 is always able to produce a relatively low number of non-overlapping pieces that can be reconstructed into the target physical object as we show in Figure 17.

Our results are preliminary, and we still have to define how actually to fabricate them. We, unfortunately, do not have available yet the machinery that we plan to use for carving the furrows in plywood and cardboard.

7. Conclusion and Future Works

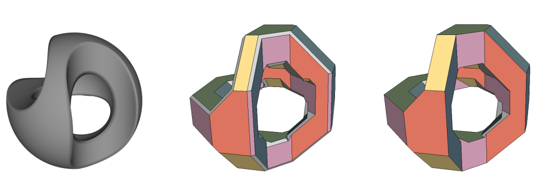
We proposed a novel method that enables the simplification of digital shapes for an easy and quick assembly process after the fabrication using CNC Milling and V-Groove tools. Our results are preliminary and we plan to improve them in multiple ways.

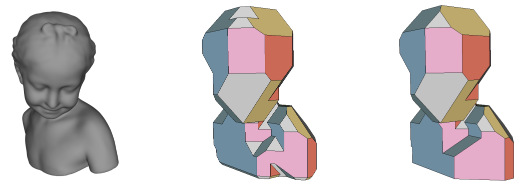
Surface approximation. Interesting future work would be the definition of a set of rules that automatically define the set of masks necessary to do some operations. We also plan to exploit symmetry, since now we do not guarantee to produce a symmetric output starting from a symmetric input. A possible solution to the problem would first automatically detect if the shape is symmetric with a state of the art approach like the one presented in [PLPZ12], and then applying the algorithm to half of the model, reflecting the other in post-processing.

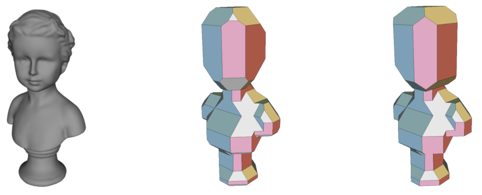
User-driven simplification. We want to add automatic criteria to point to the user the faces he or she wants to select. We also plan to improve the identification of a solution involving non-adjacent facets. To reach this goal we need to take in account not only the 1-ring of the selected facet but an n-ring, where n is a parameter to be carefully studied.

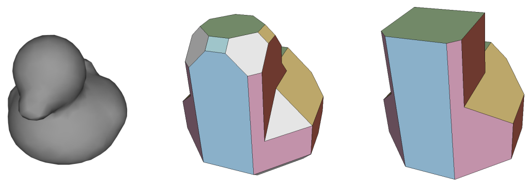
Unfolding. It could be interesting to find new algorithms to better distribute the facets in disjoint components. Indeed, even if the furrows made with V-Groove tools are very precise, the fabrication process can be error-prone. Hence, the parts composed of several polygonal facets are more sensitive to error propagation. A study about the trade-off between the number of elements and error-proneness would be needed.

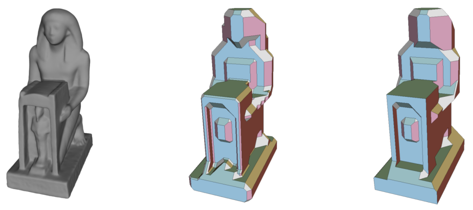
So far, in our method, we have not taken into account the size and the shape of the sheets used for fabrication. However, having this information and the desired target object size, it would be interesting to adapt our algorithm to pack the components into sheets of any form, not only rectangular. We can take advantage of the dynamism of our method: as we dynamically avoided the overlapping edges in

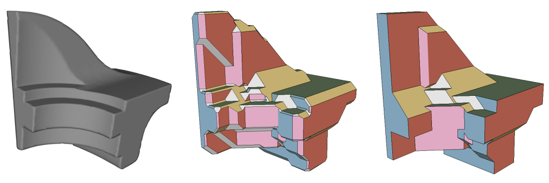

(a) Abstract Sculpture

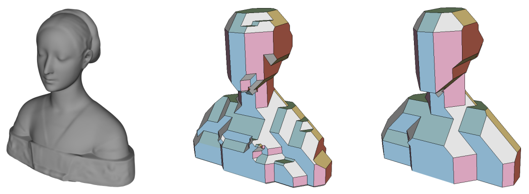

(b) Bimba

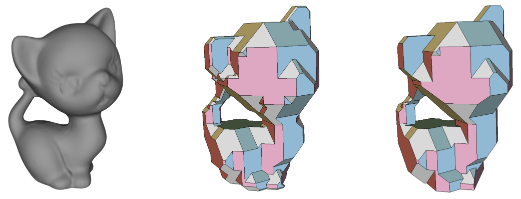

(c) BU

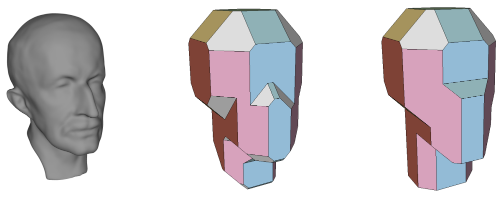

(d) Duck

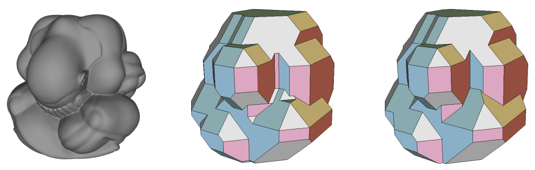

(e) Egyptian Statue


(f) Fandisk


(g) Gentildonna


(h) Kitten


(i) Max Plank


(j) Pensatore

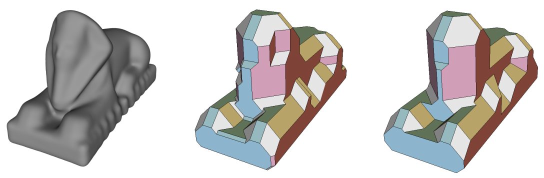
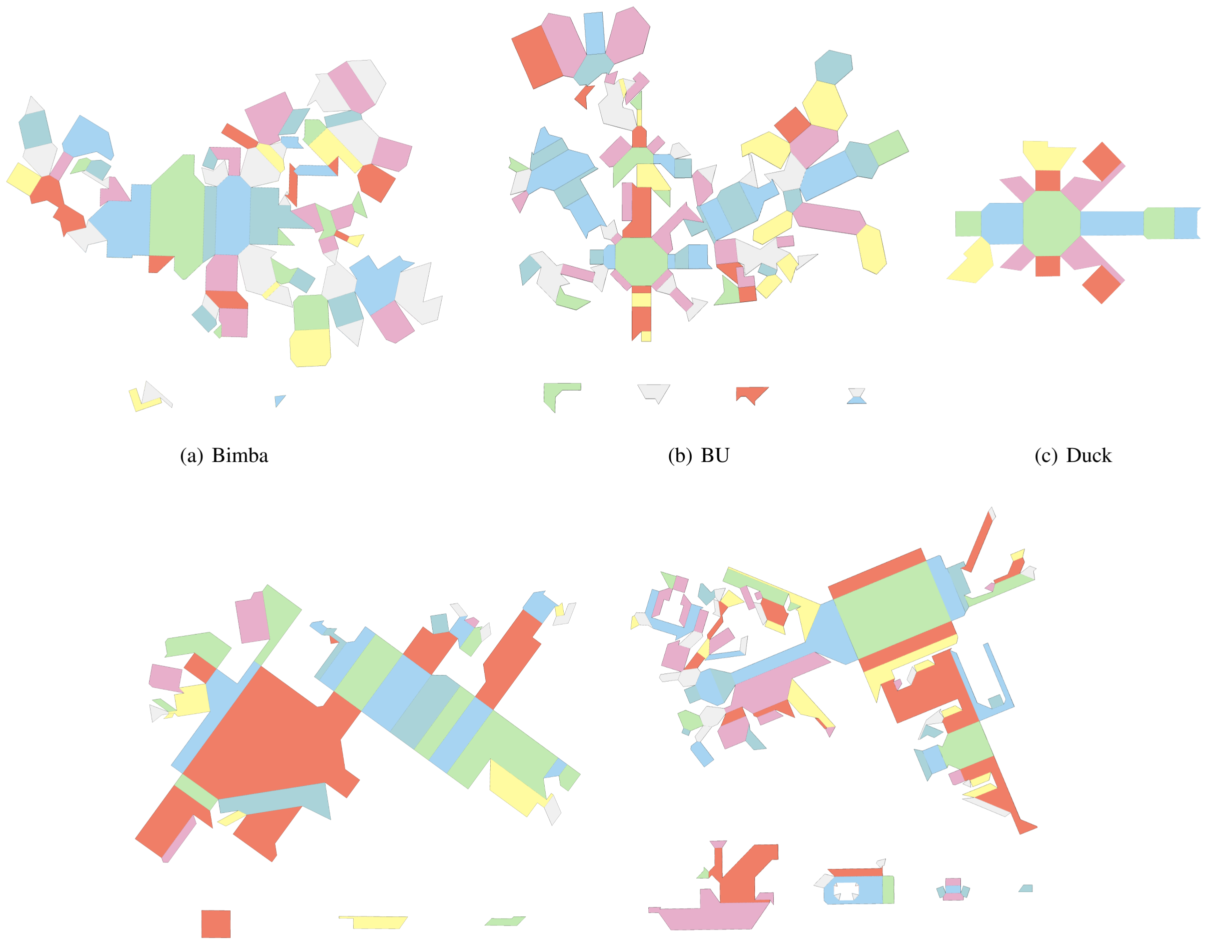

(k) Sphynx

Figure 16: *Results obtained with our method. We first process the input model (left) with our automatic approximation algorithm (center), and then the user operates to remove the undesired facets with our interactive simplification tool (right).*



(a) Bimba (b) BU (c) Duck

(d) Fandisk (e) Egyptian Statue

Figure 17: *Results obtained with our unfolding method applied on some simplified models in figure 16.*

a net, we can analogously design a technique to build each unfolding in such a way that it fits into a given 2D shape.

Toolpaths and fabrication. For manufacturing the results, it is required an analysis of the processed unfoldings to generate the right toolpaths. First of all, it is necessary to separate concave from convex angles: the machine carves the former onto the top of the sheet, and the latter onto the bottom. The second step is the grouping of the edges by their dihedral angle, to sequentially carve them with the right V-Groove tools. The toolpath also depends upon the thickness of the sheet and the cutting height and diameter of the tool.

Acknowledgements

The tool has been implemented with the library CG3Lib [MN*18] for all the steps of our algorithm. This work was partly financed by the DSURF PRIN 2015 (2015B8TRFM) project. Stefano Nuvoli gratefully acknowledges Sardinia Regional Government for the financial support of his PhD scholarship (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.)

References

[BCMP18] BICKEL B., CIGNONI P., MALOMO L., PIETRONI N.: State of the art on stylized fabrication. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 325–342. 4

[CLS16] CHERCHI G., LIVESU M., SCATENI R.: Polycube simplification for coarse layouts of surfaces and volumes. *Computer Graphics Forum 35*, 5 (2016), 11–20. 3

[CPMS14] CIGNONI P., PIETRONI N., MALOMO L., SCOPIGNO R.: Field-aligned mesh joinery. *ACM Transactions on Graphics (TOG) 33*, 1 (2014), 11. 3

[CS16] CHEN L., SASS L.: Fresh press modeler: A generative system for physically based low fidelity prototyping. *Computers & Graphics 54* (2016), 157–165. 3

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

[CSaLM13] CHEN D., SITTHI-AMORN P., LAN J. T., MATUSIK W.: Computing and fabricating multiplanar models. *Computer graphics forum 32*, 2pt3 (2013), 305–315. 3

[DO08] DEMAINE E. D., O'ROURKE J.: *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*, reprint ed. Cambridge University Press, New York, NY, USA, 2008. 7

[JTT01] JIMÉNEZ P., THOMAS F., TORRAS C.: 3d collision detection: a survey. *Computers & Graphics 25*, 2 (2001), 269–285. 9

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics 21*, 4 (1987), 163–169. 4

[LEM*17] LIVESU M., ELLERO S., MARTÍNEZ J., LEFEBVRE S., ATTENE M.: From 3d models to 3d prints: an overview of the processing pipeline. *Computer Graphics Forum 36*, 2 (2017), 537–564. 2

[LMPS16] LIVESU M., MUNTONI A., PUPPO E., SCATENI R.: Skeleton-driven Adaptive Hexahedral Meshing of Tubular Shapes. *Computer Graphics Forum 35*, 7 (Oct. 2016), 237–246. 3

[LSWW14] LIU L., SHAMIR A., WANG C. C., WHITING E.: 3d printing oriented design: geometry and optimization. In *SIGGRAPH ASIA Courses* (2014), pp. 1–1. 2

[LVS*13] LIVESU M., VINING N., SHEFFER A., GREGSON J., SCATENI R.: Polycut: monotone graph-cuts for polycube base-complex construction. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 171. 3

[MN*18] MUNTONI A., NUVOLI S., ET AL.: CG3Lib: A structured C++ geometry processing library., 2018. https://github.com/cg3hci/cg3lib. 11

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

[MSS94a] MONTANI C., SCATENI R., SCOPIGNO R.: A modified look-up table for implicit disambiguation of Marching Cubes. *The Visual Computer 10*, 6 (1994), 353–355. 4

[MSS94b] MONTANI C., SCATENI R., SCOPIGNO R.: Discretized marching cubes. In *Proceedings of the conference on Visualization '94* (Los Alamitos, CA, USA, 1994), VIS '94, IEEE Computer Society Press, pp. 281–287. 4

[Pin07] PINCIU V.: On the fewest nets problem for convex polyhedra. In *CCCG* (2007). 8

[PLPZ12] PANOZZO D., LIPMAN Y., PUPPO E., ZORIN D.: Fields on symmetric surfaces. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 111. 9

[RA15] RICHTER R., ALEXA M.: Beam meshes. *Computers & Graphics 53* (2015), 28–36. 3

[SDW*16] SONG P., DENG B., WANG Z., DONG Z., LI W., FU C.-W., LIU L.: Cofifab: coarse-to-fine fabrication of large 3d objects. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 45. 3

[She75] SHEPHARD G. C.: Convex polytopes with convex nets. *Mathematical Proceedings of the Cambridge Philosophical Society 78*, 3 (1975), 389âĂŞ403. 8

[SP13] SCHWARTZBURG Y., PAULY M.: Fabrication-aware design with intersecting planar pieces. *Computer Graphics Forum 32*, 2pt3 (2013), 317–326. 3

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: Polycube-maps. *ACM transactions on graphics (TOG) 23*, 3 (2004), 853–860. 3

[UBM15] UMETANI N., BICKEL B., MATUSIK W.: Computational tools for 3d printing. In *SIGGRAPH Courses* (2015), pp. 9–1. 2

[ZLAK14] ZIMMER H., LAFARGE F., ALLIEZ P., KOBBELT L.: Zometool shape approximation. *Graphical Models 76*, 5 (2014), 390–401. 3