Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Mill and Fold: Shape Simplification for Fabrication

Alessandro Muntoni[a,*], Stefano Nuvoli[b], Andreas Scalas[c,d], Alessandro Tola[b], Luigi Malomo[a], Riccardo Scateni[b]

[a]*Visual Computing Laboratory, ISTI-CNR, Pisa, Italy*
[b]*Department of Mathematics and Computer Science, University of Cagliari, Cagliari, Italy*
[c]*IMATI-CNR, Genova, Italy*
[d]*Department of Informatics, Bioengineering, Robotics and System Engineering, University of Genova, Genova, Italy*

ARTICLE INFO

Article history:
Accepted March 17, 2019

Keywords: 3D Meshes, Geometry Processing, Mesh Simplification, Milling, Fabrication

ABSTRACT

We introduce a pipeline for simplifying digital 3D shapes and fabricate 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 create 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, which consists only in folding adjacent facets at a constrained angle. 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 simplified version of the input model, using the marching cubes algorithm on the original mesh and iteratively performing local changes on the resulting triangle mesh. The user can then perform an additional manual simplification to remove unwanted facets. Finally, an unfolding algorithm, which takes into account the thickness of the material, flattens the polygonal facets onto the 2D plane, so that a CNC milling machine can fabricate it from a sheet of rigid material.

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 typical scenario, the printer deposits a filament, layer by layer, that solidifies and forms the final result following a software-generated path. Apart from specific limitations, which depend on the additive manufacturing technology used, these machines enable the creation of arbitrarily complex geometries.

However, additive manufacturing is not the only technology that allows reproducing physical objects. Subtractive techniques, commonly performed with CNC milling machines, follow a different philosophy: to reproduce a shape, a numerically controlled carving tool moves to remove material from a solid block. These machines are mainly used to produce very regular objects in the mechanical engineering field. CNCs can produce free-form geometries, but the set of feasible shapes is constrained by the characteristics of the machine employed. CNCs can have 3-, 4- or 5-axis, meaning that the tool moves along the three principal axes and has one or two rotational degrees of freedom. The choice of the machine is crucial to determine if the geometry can be fabricated. This aspect, along with the difficulty of producing toolpaths, especially for 4- and 5-axis machines, makes subtractive techniques still "immature" for fabricating free-form geometries.

We propose a novel method to generate a coarse approximation of an input geometry that is suitable for fabrication with 3-axis milling machines. In particular, we simplify a digital shape to a mesh of flat polygonal facets and unfold this geometry onto a set of flat pieces. These pieces can be fabricated

*Corresponding author: E-mail Address: alessandro.muntoni@isti.cnr.it

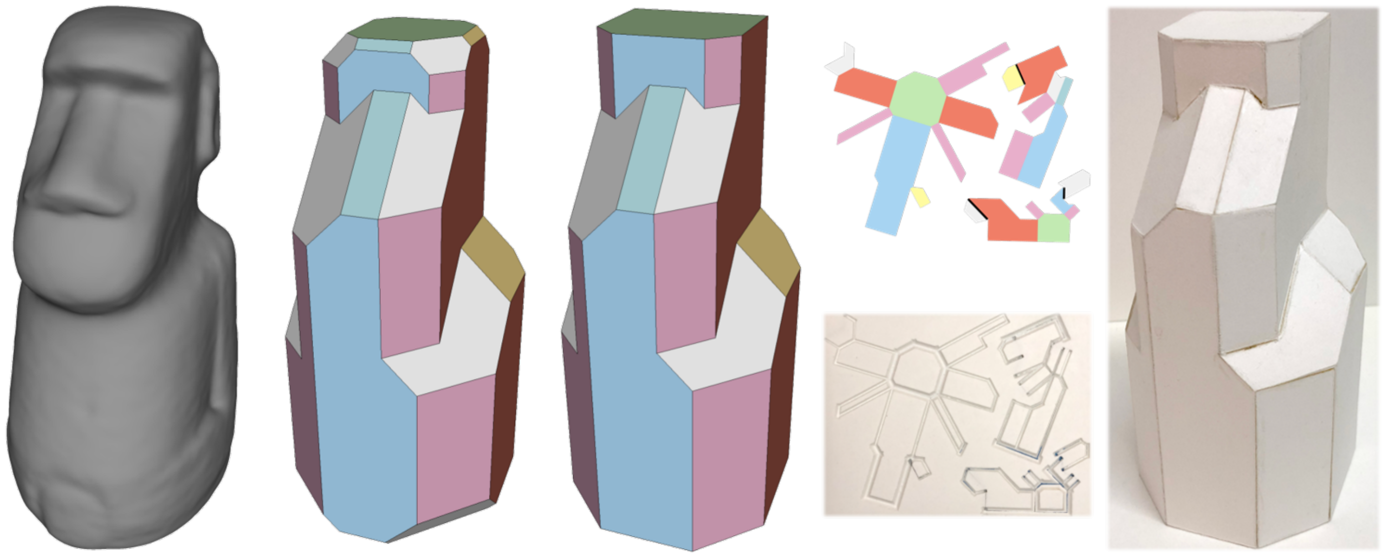


Fig. 1. From left to right: the input digital model; the result of the automatic simplification step (Section 3); the clean model after spurious facets removal (Section 4); on top the unfolding plan for fabrication and on bottom the corresponding carved sheet (Section 5); the final assembled model manufactured using a CNC milling machine and manually folded and glued (Section 6).

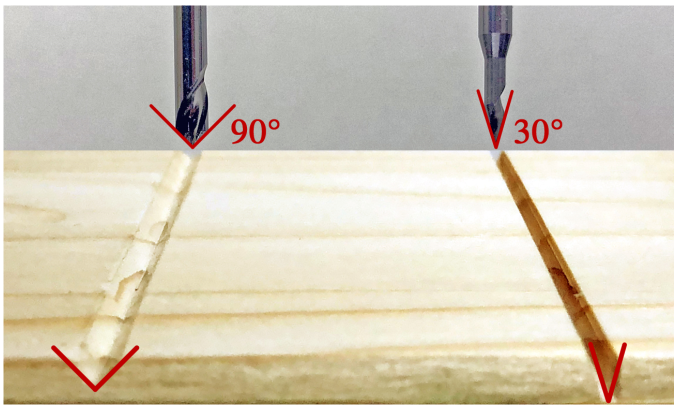
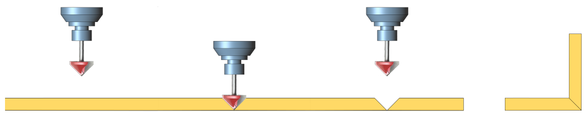


Fig. 2. V-Groove milling cutters can mill furrows with correct angles on rigid materials, and they are available with different milling angles on the market. We show here two cutter element used in our experiments.

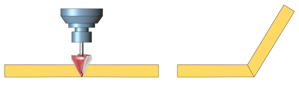
by carving flat sheets of material and, once produced, can be folded and assembled together to form an approximation of the provided input shape (Figure 1).

As discussed in Section 2, many previous works aimed to create a simplified model to obtain a polygon mesh for fabrication purposes. In particular, a recent trend is to process a digital shape to obtain a representation using sheets of material cut by laser-cutter machines. Traditionally, cuts are performed on sheets of flat rigid materials and produce 2D shapes that can be combined to form the desired object. All these works introduced a variety of joint systems to allow an easy assembly process. Joint problems arise when two adjacent primitives should be joined manually along the joint edge at an arbitrary angle. In these cases custom joint systems must be design to enforce the desired angle avoiding errors.

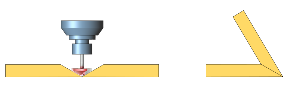
Our idea differentiates from these methods since we want to simplify our models to avoid the problem of designing joints. Therefore we propose an approach that makes the assembly process more manageable and less error-prone. We use CNC machines with V-Groove milling tools to carve our models on a rigid sheet of 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 (Figure 2). In our setup, these V-shaped milling tools enable the production of angled furrows on the edges of our flattened model. This implies that each pair of adjacent flat polygons, once folded, automatically form the desired angle (Figure 3).



(a) A 90° fold needs a 90° tool



(b) A 120° fold needs a 60° tool



(c) A 60° fold needs a 120° tool

Fig. 3. Different folds made using V-Groove milling tools.

To use this carve-and-fold strategy, we need to simplify our model so that all dihedral angles belong to a restricted and well-defined set containing only angles that available V-Groove milling cutters can carve. To enforce this constraint, we require that each normal of the facets of the simplified model must belong to a pre-defined set of values.

To summarize: we designed a simplification strategy whose output is guaranteed to have **normals belonging to a restricted set**. We employ Marching Cubes, working on a binary scalar field defined by the input model 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 in the previous step. We also introduced a novel **unfolding algorithm** suited for our purposes whose output is a 2D representation of the simplified model, which takes also into account the thickness of the flat panel to be milled.
The last step of our pipeline produces the toolpath for the CNC milling machine that fabricates the final model using V-Groove tools.

This paper is an extension and revision of the material contained in "Simplification of Shapes for Fabrication with V-Groove Milling Tools" [1], presented at the 2018 Eurographics Italian Chapter Conference, held in Brescia in October 2018. In this article we included a new unfolding algorithm that better suits for practical fabrication, and a set of fabricated results.

2. State of the Art

Our work fits into the domain of geometry processing for digital fabrication [2, 3, 4]. More precisely, our work belongs to the so called *Stylized Fabrication* domain [5], where the fidelity of the manufactured artifacts is not the primary goal. In our case, we want to produce a physical approximation of the input 3D shape that is easy to manufacture and assemble. To do so, our approach simplifies a 3D model to fabricate it from flat sheets of rigid materials, like plywood. Differently from other works, that rely on laser-cutters, we exploit subtractive technologies, in particular 3-axis milling machines. These machines were recently employed for the fabrication of general free-form shapes using specific decomposition algorithms [6, 7]. In our case, we use special cutting tools called V-Grooves, that enables the creation of easy-to-fold joints with a precise angle.

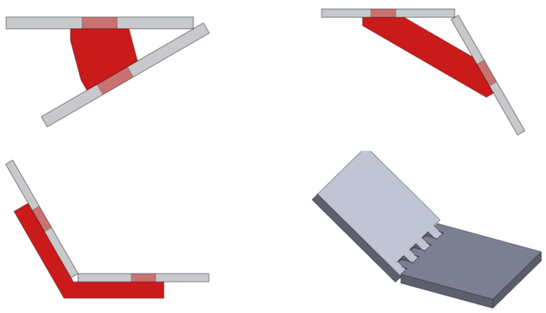
One contribution of our method is the simplification technique that we use to produce a small set of flat polygonal faces that are subsequently flattened and used to produce physical artefacts.

In computer graphics mesh simplification is a well established topic. Previous research in this field was more general purpose, rather than oriented to 3D fabrication, and aimed at reducing mesh complexity while preserving the geometric information of the models [8]. Since our goal is to simplify meshes into a small number of flat primitives, one approach that could fit our purposes is VSA [9]. The authors proposed a method to create a mesh approximation using a variational approach, reducing the input geometry to a set of flat polygonal primitives. However, this method do not satisfy the significant constraint we pose to our system, i.e. the simplified mesh should include only a small number of dihedral angles.
Polycube-based methods [10], instead, fulfil our requirements. These approaches produce simplified models in which every primitive is orthogonal to one of the three principal axes. These methods have been traditionally used in many different contexts of geometry processing and there is a vast literature on the topic (e.g., polycubes generation [11], optimization [12], hexaheralization [13]). Polycube meshes could be physically produced using our manufacturing approach with V-Groove milling tools, since all the dihedral angles between adjacent faces are 90 or 270 degrees. Even if our pipeline could accommodate such simplification strategy, using polycubes would be very restrictive considering the diversity of V-Groove tools available on the market and the poor quality of fabricated models.

Our technique is mostly related to approaches designed for fabrication purposes. There are many methods that produce tangible artifacts using flat, developable primitives that approximate the object surface, such as papercafts [14]. Papercraft techniques produce a set of paper pieces that can be cut, folded and glued together to obtain a papercraft object. However, even if the approximation degree can be quite high, when compared to our approach, resulting papercrafts are hard to assemble. Other methods explicitly investigate novel assembly processes to obtain fabricated models. Some approaches rely, for example, on interlocking mechanisms. Starting from an input geometry, these methods automatically design a set of flat, laser-cut pieces that, once assembled together, creates an approximate representation of a provided geometry [15, 16, 17]. Other methods, also targeted to laser-cut production, offer an interactive tool to create custom interlocking structures [18] or start from an existing interlocking design and optimize it to reduce the wasted material on laser-cut sheets [19]. All these methods produce *stylized* objects inserting flat pieces inside the object volume, but lack in approximating the object surface. In our setup, instead, we want to produce a closed 3D surface made of few polygonal planar facets. Moreover, many of these methods produce a large amount of custom designed pieces; compared to our work, that aims to produce a small set of pieces that can be folded and assembled easily, a large set of all different pieces increases assembly times and complicate the whole manufacturing process.

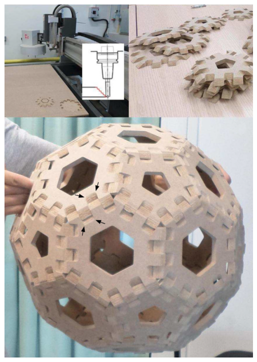
Our approach is closely related to the work of Chen et al. [20], which introduced a technique to approximate an input surface with a small number of planar, polygonal faces that can be fabricated with laser-cutters. However, with this method the assembly process is very complicated: it needs custom made connectors (see red pieces in Figure 4(a)), and it could take several hours for a single model. In our case, having added the constraint on allowed dihedral angles, we enable both fabrication with V-Groove milling tools and also avoid to introduce joints.

Similarly, Chen and Sass [21] proposed a CAD modeler application to generate laser-cut, planar structures with finger joints placed on the facet edges (see Figure 4(b)). This avoids the need to produce additional joints pieces but introduce artifacts on the fabricated model surface (i.e. when the dihedral angle between two adjacent pieces is not 90 nor 270 degrees), Also, this approach is able to produce sufficiently coarse result starting from CAD-like models and cannot be applied to generic free-form shapes.
Su et al. [22] improved the work in [21] by proposing a method that, given a simplified model, generates a set of planar patches having bevel joints that are ready for fabrication. These type of joints control the dihedral angle between adjacent pieces, but have practical issues due to the hardware used for the manufacturing: laser cutters are not able to manufacture them, generating the same issues present in [21], and with 3-axis milling machines are able to produce only approximations of the joints, creating stepped surface on portions that should be flat (Figure 4(c)).



(a) [20]

(b) [21]



(c) [22]

Fig. 4. Joint systems used in related works

In fact, this method achieves a correct reproduction only when combined with additive manufacturing.

A different method [23] was introduced to produce a large object more cheaply and quickly, by combining laser cutting and 3D printing technologies. The approach generates a coarse, internal base object composed of flat, lasercut pieces. Then, a set of thin 3D printed pieces are attached to this structure to reproduce the details of the input object. For the assembly of the lasercut internal structure, similarly to other techniques, custom joints are designed to reproduce the desired angle between pieces. The goal of their work is faithful reproduction and goes beyond the scope of this paper.

3. Surface approximation

The first step of our pipeline relies on the application of the Marching Cubes algorithm to the input shape. Marching Cubes [24] 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, facet normals of the resulting mesh belong to a finite and well-defined set. As a result, all possible dihedral angles between triangles are finite and known. In the next paragraph we describe the set N of all the triangle normals that can be generated.

Normals. A facet normal, which belongs to the set N, is a 3D unit vector v 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** non-zero component:

$$[\pm 1, 0, 0], [0, \pm 1, 0], [0, 0, \pm 1].$$

2. Twelve vectors with **two** non-zero components:

$$[\pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}, 0], [0, \pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}], [\pm\frac{\sqrt{2}}{2}, 0, \pm\frac{\sqrt{2}}{2}].$$

3. Eight vectors with three non-zero components:

$$[\pm\frac{\sqrt{3}}{3}, \pm\frac{\sqrt{3}}{3}, \pm\frac{\sqrt{3}}{3}].$$

Pairs of normals derived from this set form dihedral angles that are mostly of 30°, 45°, and their multiples. Despite some of these angles are not commonly available in the V-Groove tools market, there are commercial services that allow to order personalized V-Groove tools with arbitrary angles.

3.1. Initialization

We generate the lattice by regularly subdividing the bounding box of the input mesh, taking care of having integer lattice coordinates only. The lattice spacing is a function of the average edge length of the mesh multiplied by a user-defined parameter, which determines the resolution of the final simplified mesh. The vertices of the lattice are labeled as 1 if they are inside the surface or 0 if they are outside. We then run the discretized Marching Cubes algorithm [25] onto the lattice with an unambiguous look-up table [26] and a threshold included in the interval $(0-1)$. We thus extract 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 5, the resulting mesh is composed of a significant number of polygonal facets.

3.2. Geometry

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

Mask is a set of adjacent cubes having a specific combination of signs on their vertices which generates, using Marching Cubes, an undesired triangulation in the 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.

Figure 6 shows an example of Mask. The mask in this example is composed of four adjacent cubes with specified signs

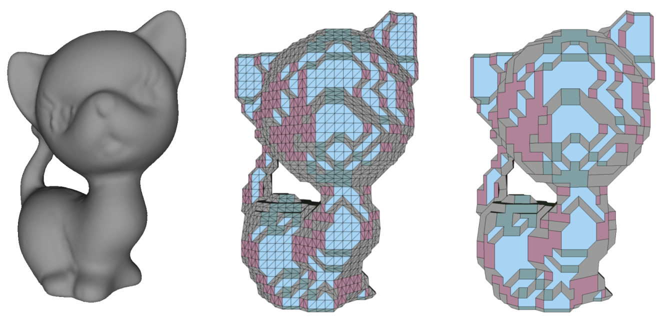


Fig. 5. From left to right: the input model, the triangle mesh obtained running DiscMC, and the polygon mesh resulting from merging the adjacent triangles laying on the same plane. The meshes are, respectively, composed of 3.628 triangles and 408 polygons. Triangles and polygons are color-coded according to their normal.

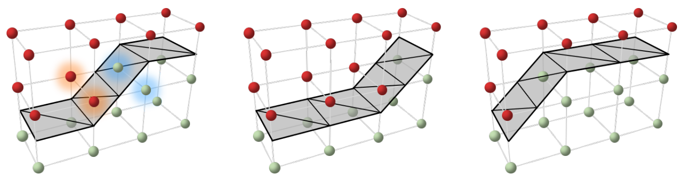


Fig. 6. The configuration on the left is one of our *Masks*. It is a portion of the lattice generating an iso-surface with two facets having normal of type 1 adjacent to a facet with normal of type 2. Switching the signs of the points haloed in blue or orange, respectively, we obtain the configurations at the center and the right. We choose the set of vertices to switch which favors the enlargement of the broader adjacent facet.

on its points, and it has two sets of *Points of Interest* circled, respectively, in orange and blue. 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 rotationally-invariant base mask types that, expanded with rotations and mirroring, generate a total of 340 masks. We describe in detail all the of masks types in the Additional Material.

As shown, 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 keep track of polygonal facet areas using an incremental update strategy and three data structures:

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

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 $O(1)$.

```
MeshSimplification(Masks,Lat);
Data: a set of masks Masks, the lattice Lat
Result: the simplified mesh Mesh
begin
    Mesh ← MarchingCubes(Lat);
    foreach triangle t ∈ Mesh do link t to the cube in Lat
      containing it;
    Seg ← ComputeSegmentation(Mesh);
    foreach facet f ∈ Seg do link f to the triangles in Mesh
      forming it;
    foreach cube c ∈ Lat do Q.push(c);
    while Q is not empty do
        c ← Q.pop();
        if c and the adjacent cubes match with m ∈ Masks then
            SwitchSignes(c,m,Lat,Seg,Mesh);
            Q.push(all modified cubes);
    return Mesh
SwitchSignes(c,m,Lat,Seg,Mesh);
begin
    Poi ← getBestPointsOfInterest (c,m,Lat,Seg);
    foreach vertex v ∈ Poi do Switch (v) in Lat;
    update locally Mesh and Seg;
    return;
```

Algorithm 1: Simplification of the mesh generated by Marching Cubes.

A high-level view of our approximation approach is given in Algorithm 1. We iteratively modify the geometry of the model by putting in a queue all the cubes of the lattice not having equal vertices labels. For each item in the queue, using its neighbors, we first build all the possible local configuration comparable with masks 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 this 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, lead to a configuration already seen in a previous iteration. In our solution, every time we detect 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 7, 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 act as a junction between two orthogonal facets (other examples are in Figure 19 a, e, f, k in Section 7). Using Marching Cubes, these features cannot be avoided. However, one of the goals of our work is to ensure an easy assembly process and these small facets

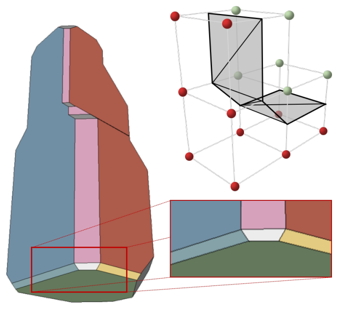


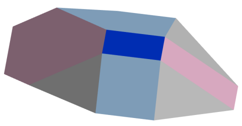
Fig. 7. Dihedral angles of 90° cannot be obtained using the Marching Cubes algorithm. There will always be small facets as junctions between orthogonal facets. In most of cases, these facets are not suitable for the proposed fabrication task and they could not contribute for a better approximation.

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 of input models containing sharp edges.

4. User-driven simplification

We have observed that the automatic approximation step retains unwanted facets in the model. Due to the highly variable nature of such facets, we have set up a Graphical User Interface for allowing the user to select which kinds of facet are *unwanted*. Our interest is twofold: in the immediate, we needed a tool to *manipulate* our mesh and obtain a better fabricable one; in the long term we aim to understand which aspects drive the selection and thus formulate appropriate automatic criteria. Such a tool requires to provide a fast 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 (the blue one in the inset). 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 (see gray facets in inset). Each pair of adjacent facets in this circular buffer generates a half-line (ray), obtained as the intersection of the planes lying on them. Each triplet of facets defines a pair of adjacent rays. We identify all the triplets that generate intersecting rays, and we choose the one which intersection is closest to one of the vertices of the deleted facet. We do this inside the chooseStartingTriplet procedure.

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. We then remove the facet from the circular buffer, and the two external facets of triplet become adjacent, generating a new ray with origin in the new vertex. This procedure is iterated on the triplets until completion. The process behind the deletion of a single facet can be better understood looking at Figure 8.

There are two termination condition:

- the last three (or more) facets in the buffer identify lines intersecting on a single point;
- the last three (or more) facets in the buffer identify lines that do not intersect.

Once the user selects the facet, the system performs the operations described above and, if the first termination condition is satisfied, it outputs the closed surface without the undesired facet. Some examples of facets deleted using this approach are shown on the base of the *Moai* model in Figure 9. Alternatively, if all remaining facets identify lines that do not intersect, an error message warns the user that the selected facet cannot be removed. This is due to the local configuration of the adjacent facets. Some deletion failure cases are shown in Figure 10.

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 is complicated to manage due to the high number of combinations that can occur. It is still an open problem for us, and we plan to tackle it in the future.

```
FacetRemoval(Mesh[],f);
Data: the mesh Mesh[], the facet to be removed f
Result: the simplified mesh Mesh[]
begin
    remove f;
    compute the circular buffer Cb[] of the facets adjacent to f;
    Tr[] ← ChooseStartingTriplet(Cb[]);
    while #Cb[] > 3 and Cb[] still contains consecutive
      triplets that can be closed do
        if Tr [2] can be closed then
            ComputeIntersection(Tr[]) and close Tr [2];
            remove Tr[] [2] from Cb[];
            Tr[] ← (Tr [1], Tr [3], Cb[].next(Tr [3]))
        else
            Tr[] ← (Tr [2], Tr [3], Cb[].next(Tr [3]))
    if #Cb[] = 3 and Tr[] can be closed then
        close last triplet Tr[]
    else
        return error
    return;
ChooseStartingTriplet(Cb[]);
begin
    foreach adjacent triplet Tr[] in Cb[] do
        v ← ComputeIntersection(Tr[]);
        sp ← ClosestStartingPoint(Tr[],v);
        dist ← Distance(v,sp);
        if dist is the shortest distance found then
            Best[] ← Tr[]
    return Best[]
```

Algorithm 2: Facet removal and expansion of the adjacent facets to close the hole.

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

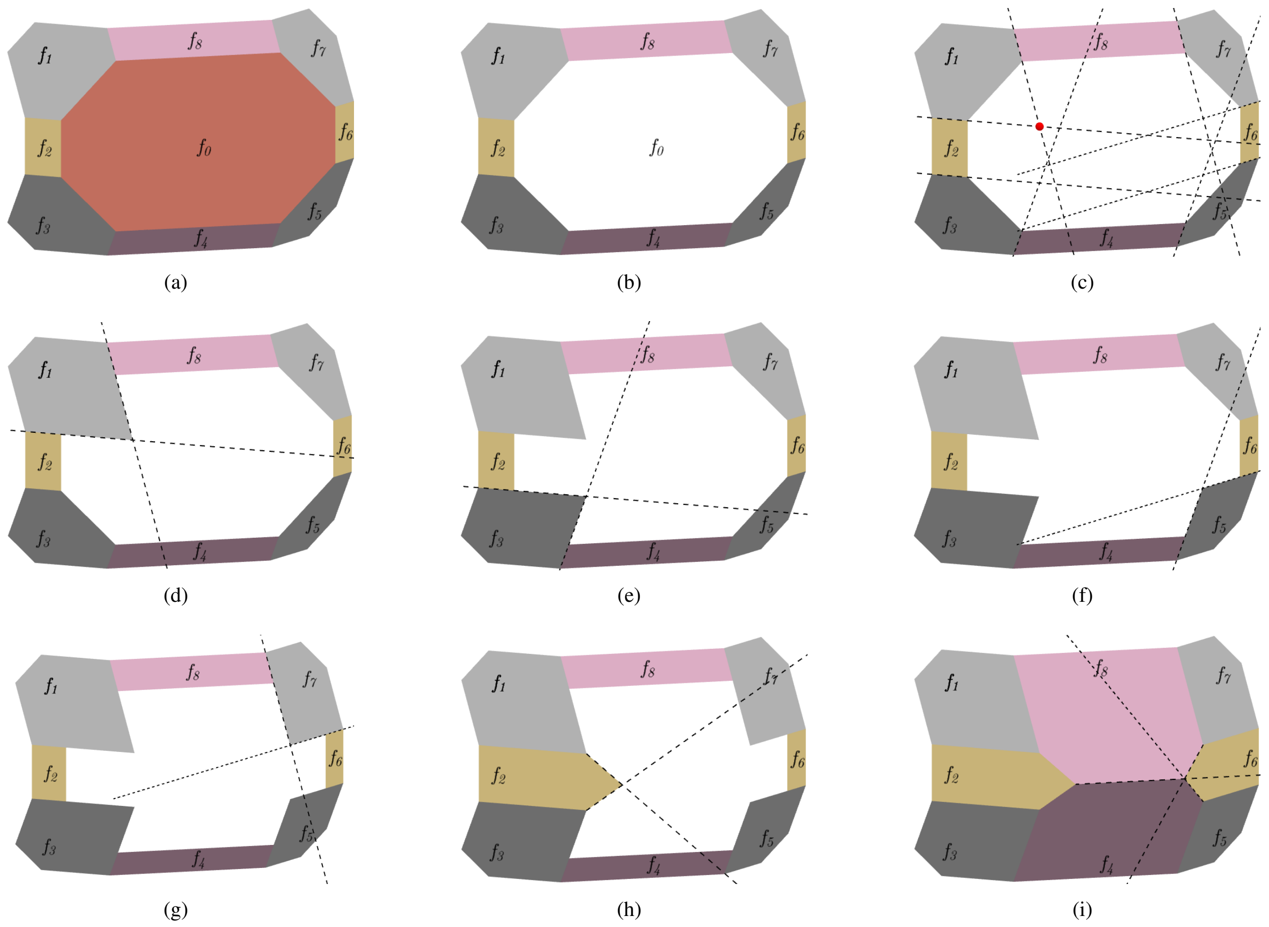


Fig. 8. Deletion of a facet. After removing f_0 (b), we put all its adjacent facets in a circular buffer. We begin consuming facets in the buffer selecting the triplet f_8, f_1, f_2 since the intersection marked in red in (c) is the closest one. We reshape the central facet of the selected triplet (f_1) 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 (e-h) until we assign all the portions of the canceled face and, thus, the surface is closed (i).

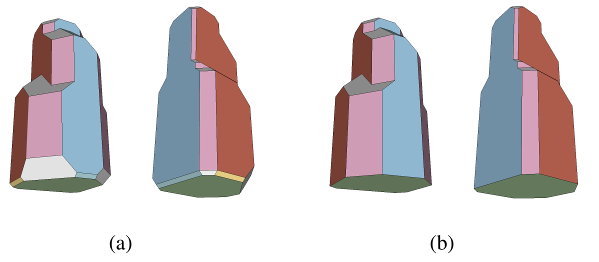


Fig. 9. The *Moai* before (a) and after (b) the removal of some facets from its bottom with our user-driven method. The quality of the approximation does not change while the complexity of the model decreases significantly.

adjacent facets lying on parallel 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 11.

5. Unfolding the mesh

The next step in our pipeline is the unfolding of the mesh to obtain an embedding of its facets on a planar surface. We cut the polyhedron along edges and flatten it onto the plane without introducing distortion. Indeed, due to the rigidity of material employed, the mapping from 3D surface to the 2D domain needs to be isometric. Hence, we are looking for a *net* of the polyhedron, an edge-unfolding of a given 3D shape forming a planar connected piece with no overlapping surfaces [27]. In Figure 12 we show two unfoldings of the *Moai*, before and after the user-driven simplification.

Unfortunately, it is not always possible to obtain an edge-unfolding to one simple non-overlapping polygon. It does not exist an efficient algorithm for determining whether a 3D shape has a net. Even the existence of a net for convex polyhedra has been an open problem since Shephard explicitly posed it in 1975 [28].

For our purposes we do not need a single net edge-unfolding: we can divide the 3D shape into components, each of which unfolds to a simple polygon (Figure 13). After manufacturing each piece separately, we can glue them together to reproduce our simplified shape.

Given a polygon mesh $M = (V, F)$, we are looking for a segmentation into a small number n of connected components C_i that form a partitioning of the mesh and can be individually

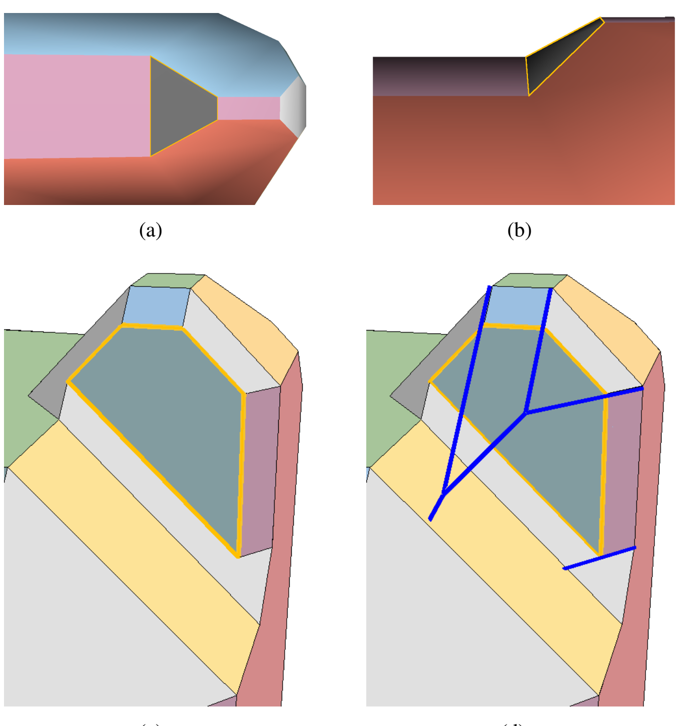


Fig. 10. Non-removable facets. In the first row (a) (b), the planes onto which the pink facets lie are parallel and never intersect, so that removing the facet would leave a hole that can not be closed. In the second row (c) (d), the expansion of the contouring facets would produce a mesh self-intersection.

unfolded in a single piece with no overlaps.

A segmentation for which all the components are unfoldable without overlap always exists, since each polygonal facet is a simple polygon. We aim at obtaining a solution with the lowest number n of components because edges inside a connected component can be just glued and folded to the their final shape while edges on component boundaries need also to be joined before being glued and folded. The more the parts we obtain, the more are the edges on the boundaries, and more difficult it would be to glue them together to reproduce the final shape. Unfortunately, we cannot determine the fewest number of parts that are unfoldable to a net since an upper-bound exists only for convex polyhedra [29].

5.1. *Naive unfolding algorithm*

To obtaining the smallest number of components, at first, we designed a naive heuristic algorithm to unfold the target shape. Given a polygonal mesh, we incrementally flatten onto the plane the highest number of polygonal facets that form a simply connected polygon. The heuristic works as follows:

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.

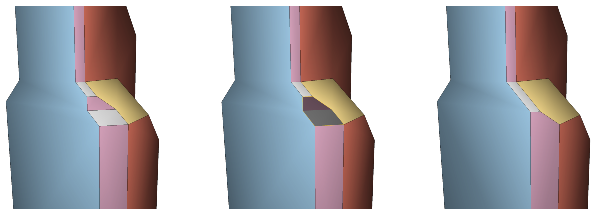


Fig. 11. Selection and deletion of two adjacent facets. The facets cannot be deleted singularly due to the adjacency of two facets lying on parallel planes. If selected together, we can close the surface extending the edges of the adjacent facets on the border of the two deleted facets.

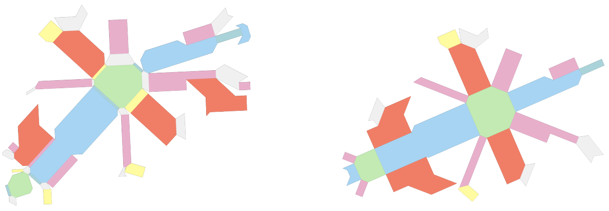
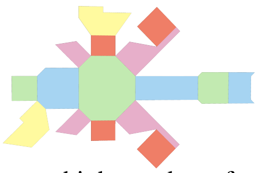


Fig. 12. The two versions of *Moai* shown in Figure 1 before (left) and after (right) the user-driven simplification.

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. The Algorithm 3 lists a pseudo-code of the method.

To pick the neighbor facet, we follow a breadth-first search approach. 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. The seed facet is the largest convex one. To efficiently test intersections we use an auto-balancing axis-aligned bounding box tree, that adapts very well to our dynamic context [30].



This algorithm fulfils our goals when it unfolds the shape in a single component as, for example, the *Duck* mesh in the inset. However, from our experiments we realized that, especially for models having concave angles or a high number of facets, this is typically not the case. When the algorithm gener-

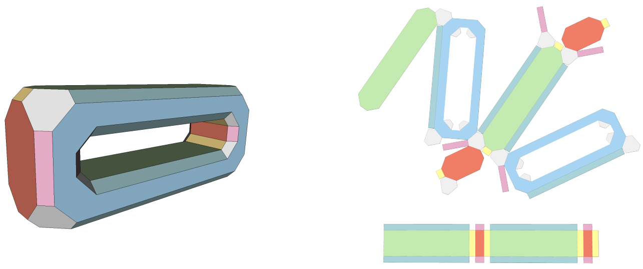
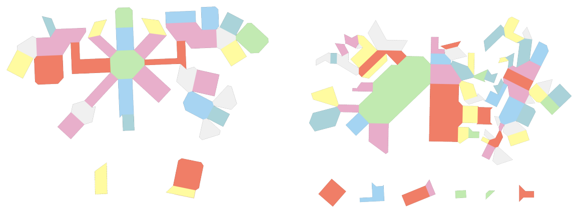


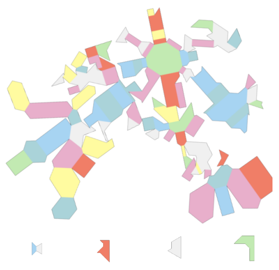
Fig. 13. A polygon mesh for which a net does not exist (left) and its unfolding in two components (right).

```
NaiveUnfolding(Mesh);
Data: the mesh to unfold Mesh
Result: the parts of the unfolding Comp[]
begin
    Facets[] ← the facets of Mesh;
    Comp[] ← $\emptyset$;
    while Facets[] $\neq \emptyset$ do
        initialize cmp[] with the largest seed facet f $\in$
          Facets[];
        Facets[] ← (Facets[] \ f);
        while cmp[] can be expanded do
            ExpansionStep(cmp[],Facets[]);
        add cmp[] to Comp[];
    return Comp[]
ExpansionStep(cmp[],Facets[]);
begin
    Pick the first f $\in$ Facets[] among the neighbor facets on
      the current boundary bound (cmp[]) that causes no
      overlaps when added to cmp[] along an edge;
    Add f to cmp[] along the chosen edge;
    Facets[] ← (Facets[] \ f);
    return;
```

Algorithm 3: Naive method for mesh unfolding.



(a) Max Planck (b) Laurana



(c) Buste

Fig. 14. Unfoldings obtained with the naive approach applied to some complex models. The results illustrate the limitations of the naive algorithm.

ates more than one component, the output is often composed of one large component containing the majority of the facets and several other disjoint components containing a few facets (i.e. from 1 to 3, see Figure 14).

5.2. Smart unfolding

To improve manufacturing, we designed a novel algorithm that reduces the variance of the parts' size. The idea behind the method is to expand an increasing number of components in parallel until all mesh facets are included in one component.

At first, we run the naive algorithm. If the output is a single component the algorithm stops. Otherwise, we select the seed facets of the two largest components obtained by the algorithm, and we expand them in parallel, one facet at a time for each part, until it is not possible to expand them anymore. If the final result includes all mesh facets we have two components with, approximatively, the same number of facets. Instead, if there are remaining facets, we launch the naive algorithm over the subset of remaining facets. Then we reiterate the parallel algorithms adding one more seed obtained from the largest component resulting from the last naive step. We proceed adding seeds in this manner and restarting the parallel expansion until all obtained pieces are expanded in parallel. This procedure is described in detail in Algorithm 4.

The algorithm improves the solution at each iteration. In Figure 15 we show the results obtained with the smart method (input models are the same ones used in Figure 14).

```
SmartUnfolding(Mesh);
Data: the mesh to unfold Mesh
Result: the parts of the unfolding Comp[]
begin
    Comp[] ← NaiveUnfolding(Mesh);
    if #Comp[] = 1 then
        return Comp[]
    numOfComp ← 2;
    repeat
        sort Comp[] by number of facets;
        Comp[] ← the first numOfComp pieces from Comp[];
        Facets[] ← the set of facets not in Comp[];
        foreach cmp[] $\in$ Comp[] do
            cmp[] ← s;    /* s is the seed facet */
            Facets[] ← Facets[] \ s;
        repeat
            foreach cmp[] $\in$ Comp[] do
                if cmp[] can expand then
                    ExpansionStep(cmp[],Facets[]);
        until no cmp[] $\in$ Comp[] can expand;
        if #Facets[] $\neq \emptyset$ then
            Comp[] ← Comp[] +
              NaiveUnfolding(Facets[]);
            numOfComp ← numOfComp +1;
    until #Comp[] = numOfComp;
    return Comp[]
```

Algorithm 4: Smart method unfolding a mesh minimizing the variance of the sizes of the components.

6. Fabrication

The actual manufacturing is a two-stage process: milling and assembly. First, a CNC milling machine works on a rectangular

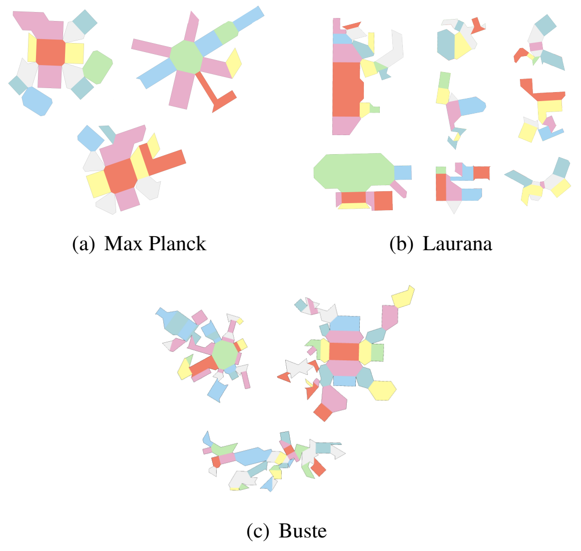

(a) Max Planck (b) Laurana (c) Buste

Fig. 15. Results obtained with the smart unfolding method.

sheet of plywood or other similar material to carve the furrows along the edges and, then, the components are folded and glued together in order to reconstruct the digital shape physically. We aim at obtaining a manufacturing process which minimizes the scrap material and makes the shape simple to assemble. In the following, we detail the fabrication process.

6.1. Packing

During the milling phase, we want to reproduce the components in one or more rectangular sheets with the goal of producing as less scrap material as possible. We pack the components into a rectangular sheets of chosen size using the Rasterized Packer of the VCG Library [31]. This algorithm rotates and translates each component multiple times and, using different heuristics, it tries to fit them in the lowest number of target rectangles. At last it chooses a configuration that minimizes the waste.

On top of the geometrical problem of packing we need to take into account the requirements of the milling machine. The edges of the components represent the centerline of the furrows which have a thickness. Thus, we insert enough padding space to separate the components and avoid intersections. Figure 16 shows how three components are packed into a rectangular sheet for manufacturing a 14cm high object.

During the unfolding stage, we have not taken into account the size and the shape of the sheets used for fabrication. Indeed, a single component can exceed the maximum size of a sheet of material. We avoid this by adapting the unfolding algorithm to produce components that can be packed into rectangular sheets of a given size. Each time a candidate facet is tested for component expansion, we check if the oriented bounding box of the current unfolded polygon fits into the target rectangular sheet. In Figure 17 we show how the same mesh of Figure 16 is packed in three rectangular sheets to manufacture a bigger object (25 cm height). For this case we use the Rotating Calipers method [32] implemented in CGAL [33].

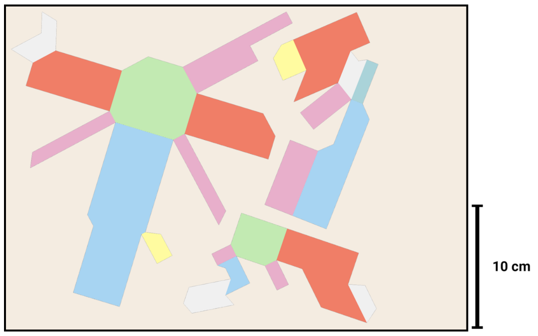


Fig. 16. Unfolding and packing the Moai model (14cm height) in a rectangular sheet of 37×26 cm.

6.2. Manufacturing

For actual manufacturing of the models, we performed post-processing of the unfoldings to generate the toolpaths. First, it is necessary to separate concave from convex angles: the machine carves the former on one side of the sheet, and the latter on the other. We then partition the set of edges using their dihedral angle, to sequentially carve each subset with different V-Groove tools. The remaining parameters fed to the toolpath generation procedure are: the thickness of the sheet and the height and diameter of the tools.

To reproduce the outer surface with the correct size we need to take into account that the edges have a thickness. The concave angles, thus, *push away* on both sides their incident faces of a quantity equal to half of the upper width of the furrow. An additional dummy facet appears on the unfolding to accommodate this. We further modified the unfolding stage to add these dummy facets. The facets' sizes depend on the thickness of the target sheet to be carved. The unfolding of Figure 1 is an example of result where dummy facets are shown in black.

For the toolpath generation, we can use any software that allows engraving manufacturing. When the target sheet height is greater than the available V-Groove flute length, as in our experiments, we transform each edge into a dense aligned set of parallel paths and use a standard toolpath generating tool. We used Autodesk Fusion 360 for toolpath generation.

6.3. Assembly process

Once we have the components manufactured we can assemble the final model. In Figure 18 we show an example of the manipulation needed to glue together the various fabricated parts. A more extensive visual explanation is available in the video provided as supplementary material.

7. Results

The approximation proposed in Section 3 is based on Marching Cubes and therefore, our outputs have the same properties guaranteed by this algorithm: all our results are water-tight 2-manifold meshes made of polygonal flat facets with normals belonging to a restricted, known set. The method is very quick: every approximate model presented in this paper are generated within ten seconds.

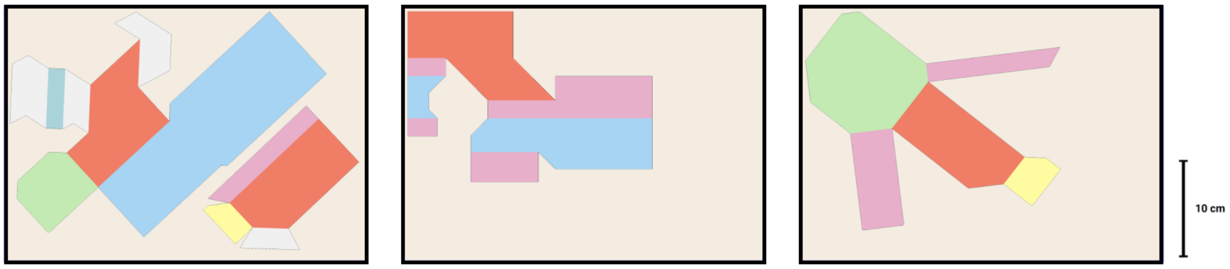


Fig. 17. Unfolding and packing the Moai model (25cm height) in rectangular sheets of size 37×26 cm.

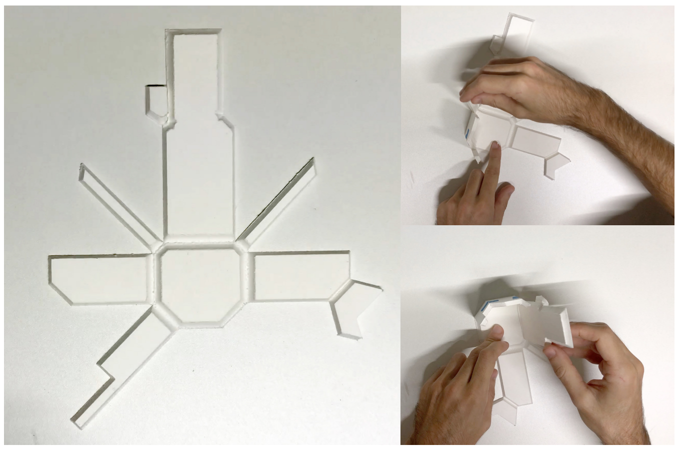


Fig. 18. An example of the assembly process for the Moai model.

Model	# FAS	# FUS	Red. (%)	*n*
Abstract Sculpture	108	67	38.0%	4
Bimba	115	72	37.4%	6
Buste	128	91	28.9%	10
Duck	41	16	61.0%	1
Egyptian Statue	293	99	66.2%	16
Laurana	159	68	57.2%	8
Kitten	242	125	48.3%	11
Max Plank	67	36	46.3%	3
Moai	43	23	46.5%	1
Pensatore	113	87	23.0%	7
Sphynx	118	73	38.1%	9

Table 1. Number of polygonal flat facets in the approximate models (# FAS) and after the user-driven intervention (# FUS). The third column shows the reduction percentage after the user interaction. The fourth column lists the number of components n in the edge-unfolding of the simplified shape.

The user-driven simplification tool described in Section 4 is simple to use after some training, and required around 4-5 minutes to produce each presented result. The simplification time is only due to the user navigation: the most significant part of the time is due to searching for the undesired facets, while the elimination and the expansion of the neighbors is instantaneous.

Several results obtained with the approximation and simplification methods are shown in Figure 19. For each model, we show the result obtained by the surface approximation in the center column, and after the user-driven simplification in the right column. For the same results, the number of polygonal facets and the number of components are reported in Table 1.

The unfolding method described in Section 5 is always able to produce a low number of non-overlapping pieces similar in size. Moreover, once the size of the tangible object is chosen, the packing process enables us to perform a manufacturing that minimizes the scrap material on sheets of a given size. The entire process to unfold and pack of shape takes well within a second in all tested cases. We show two results in Figure 20.

For each simplified model, we were able to obtain a result that can be manufactured and reconstructed into the target physical object. We produced five different objects: three in plywood (a cube: Figure 21; a modified sphere: Figure 22; a 3-genus model: Figure 23) and two in sturdy cardboard (the Moai model: Figure 1; the Laurana model: Figure 24). The fabrication process produced, for each tested input model, an object which can be folded into the target shape reducing the waste. The milling time relates to the complexity of the input model and the size of the target object. It took approximately four hours to manufacture the Moai model while we were able to fabricate the sphere, much simpler, in just one hour.

The assembly proved to be quite straightforward. We paid particular attention to the folding order for the 3-genus model since the internal portion of the shape next to the holes has to be glued before closing the shape. We assembled a relatively complex model as the Moai in approximately fourty minutes, excluding the time needed for the glue to dry.

8. 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.

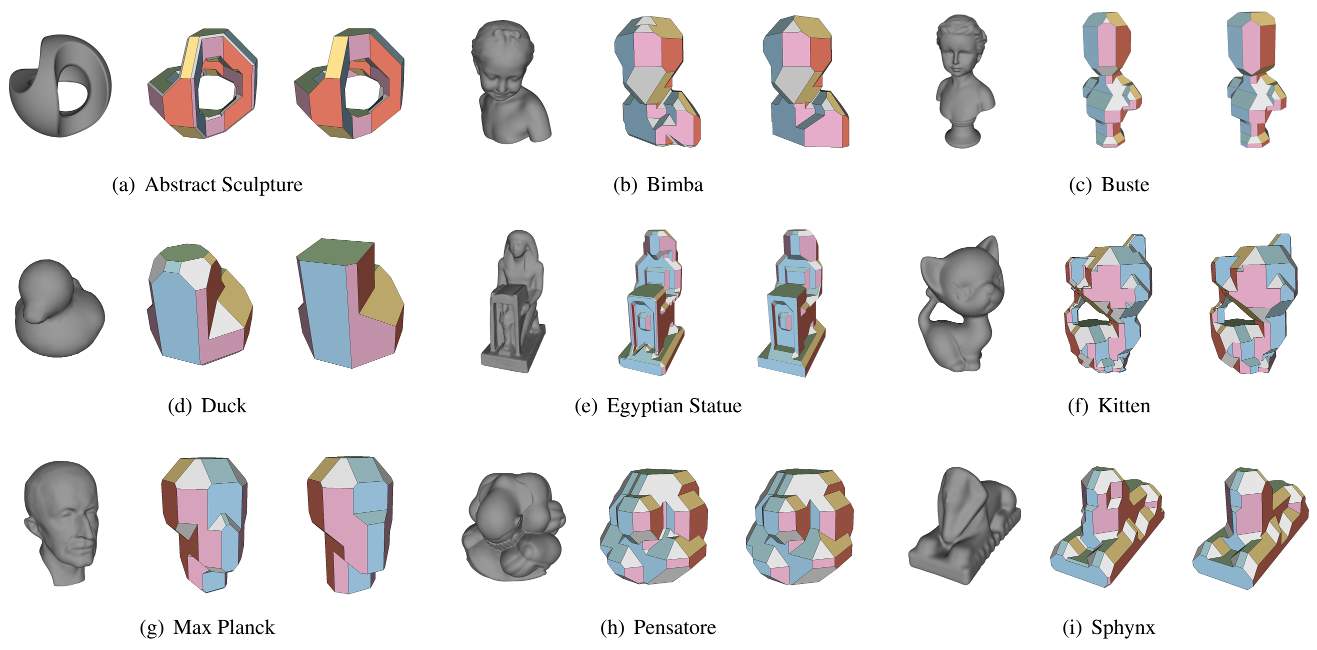


Fig. 19. Models processed with our pipeline not yet fabricated. We first process the input model (left) with our automatic approximation algorithm (center), and then the user operates to remove the undesired facets with our interactive simplification tool (right).

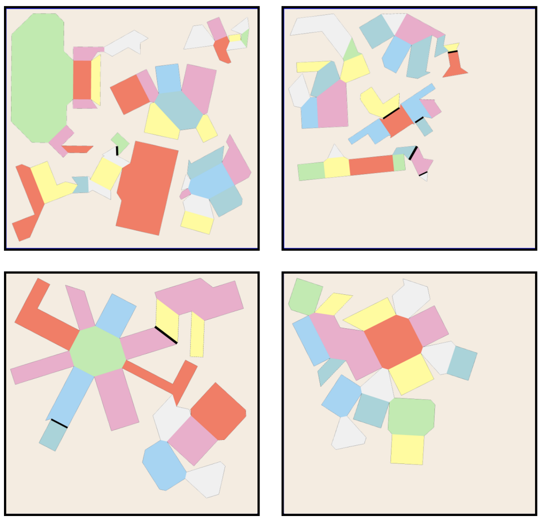


Fig. 20. Results of the unfolding and packing process for the Laurana (top) and Max Planck (bottom).

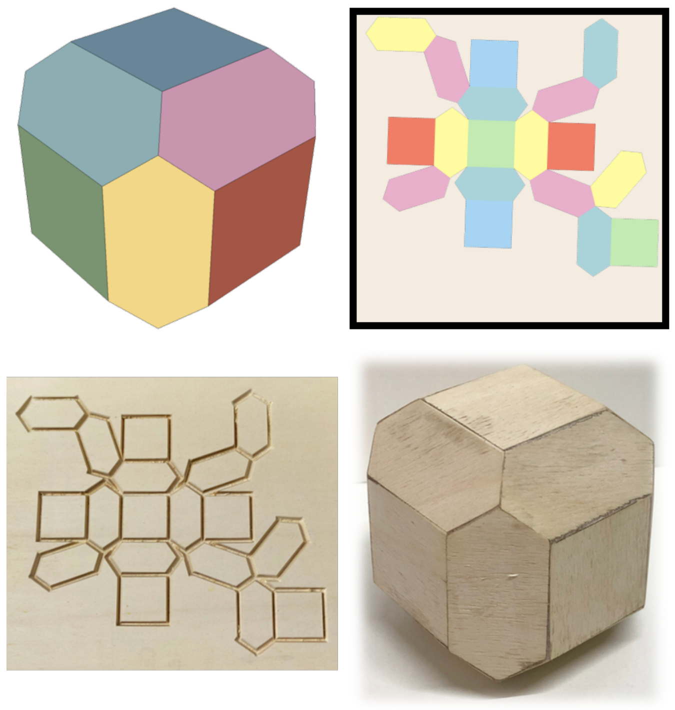


Fig. 22. Manufacturing of a modified sphere. From top left to bottom right: the digital simplified model, its unfolding, the carved sheet of plywood, and a picture of the fabricated model.

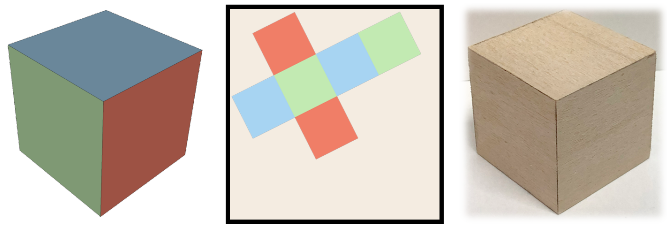


Fig. 21. Manufacturing of a cube. From left to right: digital model, unfolding, and a picture of the fabricated model.

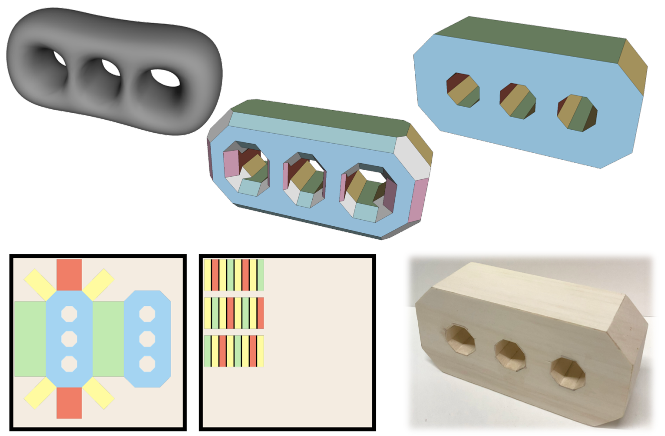


Fig. 23. Manufacturing of a 3-genus mesh. From top left to bottom right: the input digital model, the simplified model, the model after the user interaction, its unfolding in two sheets, and a picture of the fabricated model.

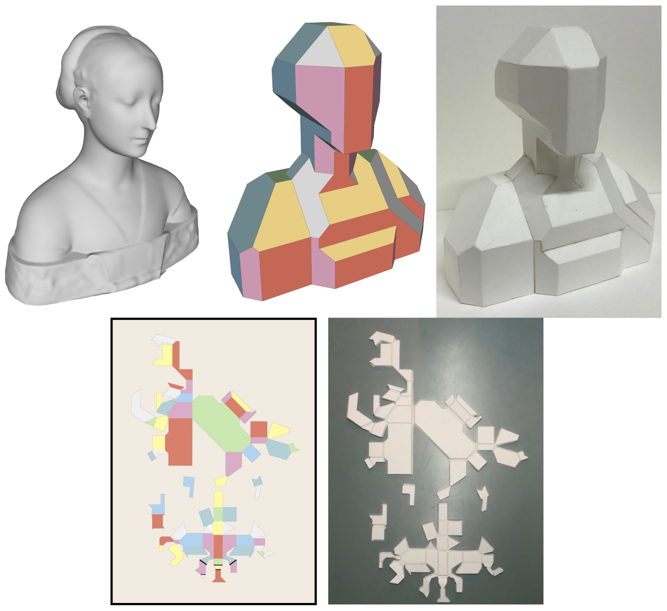


Fig. 24. Manufacturing of the Laurana Model, 22cm tall. From top left to bottom right: the input digital model, the simplified model after the user interaction, the assembled model, its unfolding (rendered and milled).

Surface approximation. We plan to introduce explicit and implicit symmetry control. In the current setup, we cannot guarantee that the final manufactured object is symmetric starting from a symmetric input. One possibile solution could automatically detect the shape symmetry with an algorithm like the one presented in [34], and then apply the algorithm to half of the model, obtaining the other half in post-processing by mirroring.

User-driven simplification. We plan to add automatic criteria to suggest facets for selection to the user. 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.

Cutting and unfolding. We intend to study the vast literature on cutting, unfolding and packing related to texture mapping to understand if some techniques used for generating texture atlases could be applied to our pipeline. The main difference is that we do not allow deformation in the projection of the mesh onto the plane for the unfolding, but the final assembly could benefit from choosing different cutting and unfolding not obtained with our current growing approach.

Acknowledgements

The implementation relies over the library CG3Lib [35] 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 Ph.D. 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

[1] Muntoni, A, Scalas, A, Nuvoli, S, Scateni, R. Simplification of Shapes for Fabrication with V-Groove Milling Tools. In: Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference. The Eurographics Association; 2018, p. 1–12.
[2] Liu, L, Shamir, A, Wang, CC, Whiting, E. 3D printing oriented design: geometry and optimization. In: SIGGRAPH ASIA Courses. 2014, p. 1–1.
[3] Umetani, N, Bickel, B, Matusik, W. Computational tools for 3D printing. In: SIGGRAPH Courses. 2015, p. 9–1.
[4] 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 2017;36(2):537–564.
[5] Bickel, B, Cignoni, P, Malomo, L, Pietroni, N. State of the art on stylized fabrication. Computer Graphics Forum 2018;37(6):325–342.
[6] Muntoni, A, Livesu, M, Scateni, R, Sheffer, A, Panozzo, D. Axis-Aligned Height-Field Block Decomposition of 3D Shapes. ACM Trans Graph 2018;37(5):169:1–169:15.
[7] Zhao, H, Zhang, H, Xin, S, Deng, Y, Tu, C, Wang, W, et al. DSCarver: Decompose-and-spiral-carve for Subtractive Manufacturing. ACM Trans Graph 2018;37(4):137:1–137:14.
[8] Cignoni, P, Montani, C, Scopigno, R. A comparison of mesh simplification algorithms. Computers & Graphics 1998;22(1):37 – 54.
[9] Cohen-Steiner, D, Alliez, P, Desbrun, M. Variational shape approximation. ACM Transactions on Graphics (TOG) 2004;23(3):905–914.
[10] Tarini, M, Hormann, K, Cignoni, P, Montani, C. Polycube-maps. ACM transactions on graphics (TOG) 2004;23(3):853–860.

[11] Livesu, M, Vining, N, Sheffer, A, Gregson, J, Scateni, R. Polycut: monotone graph-cuts for polycube base-complex construction. ACM Transactions on Graphics (TOG) 2013;32(6):171.

[12] Cherchi, G, Livesu, M, Scateni, R. Polycube simplification for coarse layouts of surfaces and volumes. Computer Graphics Forum 2016;35(5):11–20.

[13] Livesu, M, Muntoni, A, Puppo, E, Scateni, R. Skeleton-driven Adaptive Hexahedral Meshing of Tubular Shapes. Computer Graphics Forum 2016;35(7):237–246.

[14] Mitani, J, Suzuki, H. Making papercraft toys from meshes using strip-based approximate unfolding. ACM transactions on graphics (TOG) 2004;23(3):259–263.

[15] Schwartzburg, Y, Pauly, M. Fabrication-aware design with intersecting planar pieces. Computer Graphics Forum 2013;32(2pt3):317–326.

[16] Cignoni, P, Pietroni, N, Malomo, L, Scopigno, R. Field-aligned mesh joinery. ACM Transactions on Graphics (TOG) 2014;33(1):11.

[17] Richter, R, Alexa, M. Beam meshes. Computers & Graphics 2015;53:28–36.

[18] McCrae, J, Umetani, N, Singh, K. Flatfitfab: interactive modeling with planar sections. In: Proceedings of the 27th annual ACM symposium on User interface software and technology. ACM; 2014, p. 13–22.

[19] Koo, B, Hergel, J, Lefebvre, S, Mitra, NJ. Towards zero-waste furniture design. IEEE transactions on visualization and computer graphics 2017;23(12):2627–2640.

[20] Chen, D, Sitthi-amorn, P, Lan, JT, Matusik, W. Computing and fabricating multiplanar models. Computer graphics forum 2013;32(2pt3):305–315.

[21] Chen, L, Sass, L. Fresh press modeler: A generative system for physically based low fidelity prototyping. Computers & Graphics 2016;54:157–165.

[22] Su, Z, Chen, L, He, X, Yang, F, Sass, L. Planar structures with automatically generated bevel joints. Computers & Graphics 2018;72:98–105.

[23] Song, P, Deng, B, Wang, Z, Dong, Z, Li, W, Fu, CW, et al. Cofifab: coarse-to-fine fabrication of large 3D objects. ACM Transactions on Graphics (TOG) 2016;35(4):45.

[24] Lorensen, WE, Cline, HE. Marching cubes: A high resolution 3D surface construction algorithm. ACM siggraph computer graphics 1987;21(4):163–169.

[25] Montani, C, Scateni, R, Scopigno, R. Discretized marching cubes. In: Proceedings of the conference on Visualization '94. VIS '94; Los Alamitos, CA, USA: IEEE Computer Society Press; 1994, p. 281–287.

[26] Montani, C, Scateni, R, Scopigno, R. A modified look-up table for implicit disambiguation of Marching Cubes. The Visual Computer 1994;10(6):353–355.

[27] Damian, M, Flatland, R, O'Rourke, J. Grid vertex-unfolding orthogonal polyhedra. Discrete & Computational Geometry 2008;39(1):213–238.

[28] Shephard, GC. Convex polytopes with convex nets. Mathematical Proceedings of the Cambridge Philosophical Society 1975;78(3):389403.

[29] Pinciu, V. On the fewest nets problem for convex polyhedra. In: CCCG - Canadian Conference on Computational Geometry. 2007, p. 21–24.

[30] Jiménez, P, Thomas, F, Torras, C. 3D collision detection: a survey. Computers & Graphics 2001;25(2):269–285.

[31] Cignoni, P, Ganovelli, F, et al. VCG Library: The Visualization and Computer Graphics Library. 2018. Http://github.com/cnr-isti-vclab/vcglib/.

[32] Toussaint, G. Solving geometric problems with the rotating calipers. 1983.

[33] CGAL, Computational Geometry Algorithms Library. 2018. Https://www.cgal.org.

[34] Panozzo, D, Lipman, Y, Puppo, E, Zorin, D. Fields on symmetric surfaces. ACM Transactions on Graphics (TOG) 2012;31(4):111.

[35] Muntoni, A, Nuvoli, S, et al. CG3Lib: A structured C++ geometry processing library. 2018. Https://github.com/cg3hci/cg3lib.