Extraction of the Quad Layout of a Triangle Mesh Guided by its Curve-Skeleton

FRANCESCO USAI
Università di Cagliari
MARCO LIVESU
Università di Cagliari
ENRICO PUPPO
Università di Genova
MARCO TARINI
Università dell'Insubria and ISTI-CNR
and
RICCARDO SCATENI
Università di Cagliari

Starting from the triangle mesh of a digital shape, mainly an articulated object, we produce a coarse quad layout that can be used in character modeling and animation. Our quad layout follows the intrinsic object structure described by its curve skeleton; it contains few irregular vertices of low degree; it can be immediately refined into a semi-regular quad mesh; it provides a structured domain for UV-mapping and parametrization. Our method is fast, one-click and it does not require any parameter setting. The user can steer and refine the process through simple interactive tools during the construction of the quad layout.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Quad mesh, curve skeleton, free-form modeling, animation

ACM Reference Format:

Usai, F., Livesu, M., Puppo, E., Tarini, M., and Scateni, R., YYYY. Coarse Quad Layouts from Curve-Skeletons. ACM Trans. Graph. VV, N, Article XXX (Month YYYY), nn pages.

Authors' addresses: F. Usai, M. Livesu, and R. Scateni (corresponding author), Università di Cagliari, Cagliari, Italy; email: riccardo@unica.it; E. Puppo, Università di Genova, Genova, Italy; M. Tarini, Università dell'Insubria, Varese, Italy, and ISTI-CNR, Pisa, Italy.

1. INTRODUCTION

In recent years, modeling packages based on *virtual sculpting* are becoming more and more popular, especially in the entertainment industry: ZBrush [Pixologic 2007] was the first program to adopt such a paradigm, soon followed by other systems such as MudBox [Autodesk 2007] and 3D Coat [Pilgway 2009]. For free-form shapes, virtual sculpting is more flexible, easier and more intuitive than classical modeling with NURBS or direct polygon editing. However, virtual sculpting systems produce irregular meshes, just like scanning systems, while target models for the modeling and animation pipeline are most often surfaces defined upon a quad-based control mesh (e.g., NURBS, subdivision surfaces). The connectivity of the control mesh, the alignment of mesh elements, as well as the placement of its singularities are deciding factors whether a mesh is usable for subsequent processing.

Converting an unstructured mesh into a more regular and structured model is a challenging problem, which is known under the name of *retopology* among practitioners, and *remeshing* in the geometry processing literature. In spite of many attempts to automate this task [Bommes et al. 2013], there still exists no satisfactory method for practical applications: some methods can convert an unstructured triangle meshes into a quad or quad-dominant mesh, but the result does not capture the high-level features required for editing, deformation, or animation. On the other hand, modeling systems provide rudimental tools, requiring substantial manual intervention from an experienced user, to convert the resulting unstructured meshes into more regular and structured models: in this case, an acquired or sculpted object most often is used merely as a guide for the artist to create a new, structured shape from scratch, often consuming more time than the actual modeling task.

In contrast, more traditional CAD approaches allow an artist to design an object through shape refinement, usually starting at a block and proceeding through subdivision, extrusion and stitching operations, which are complemented with local mesh editing during the finer stages of design. This process is certainly more difficult and tedious than virtual sculpting, but it directly produces a quad control mesh reflecting the intrinsic object structure.

We propose an automatic method that mimics the refinement-and-extrusion approach to recover a coarse quad layout following

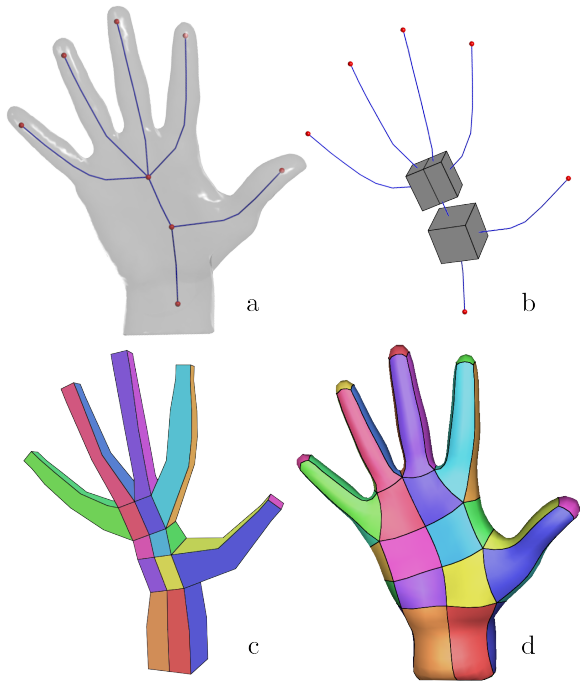


Fig. 1. We start at an input object and we extract its line skeleton (a); we place a subdivided box at each branching node of the skeleton (b); we produce a "fattened skeleton" by extruding tubes with quadrangular section along branches (c); we project this mesh onto the input surface to obtain a coarse quad layout (d).

the overall structure of an object. We address shapes whose main morphological structure can be captured by a curve-skeleton. This class is quite wide and of high practical utility: it includes all the shapes featuring body and limbs, like those used for typical virtual characters (e.g., all anthropomorphic creatures, and most animals, real or imaginary alike), as well as several mechanical objects, plants, some anatomical structures, etc.

The pipeline of our method is depicted in Figure 1. Starting at a skeleton extracted from an irregular surface mesh with an off-the-shelf method, we automatically extract a representation of the object made of branching boxes and "tubes" extruded from their faces along skeleton branches. This assembly of boxes and tubes provides the topology of a coarse quad layout with edges aligned along limbs. A parametrization of the input shape on such quad layout provides a base mesh to model the object at hand. The quad layout we produce is ready for further refinement without the need for extensive retopology.

The skeleton is usually not able to capture fine detail like, for instance, the facial features of a head, or the scales of a dragon. However, the modeling of fine detail is beyond the scope of this work, and complementary to it: our coarse quad layout in fact provides a basis for several possible tasks in the advanced stages of modeling.

1.1 Contribution

Our main contribution is the derivation of a *pure quad layout* from a triangle mesh and its curve skeleton. This is performed with an automatic method that is fast, one-click and does not require parameter setting. It can be applied to objects of any genus, provided that the curve-skeleton is accurate enough to represent its structure. We also support interactive techniques that allow the user to change the structure of the quad layout on-the-fly, by correcting the choices of the automatic system. We present applications to semi-regular quad meshing and UV-mapping.

2. RELATED WORK

Several automatic methods for quad meshing have been proposed during the last decade. See [Bommes et al. 2013] for a recent survey. Such methods are successful in producing dense quad meshes aligned to curvature or shape features, but they fail to address other quality criteria that are relevant to the production pipeline. In fact, they do not provide enough control on the placement of singularities and the meshes they produce seldom embed the object structure. Some automatic methods have been proposed to produce a semi-regular coarse quad layout, either as a post-processing [Bommes et al. 2011; Tarini et al. 2011], or directly [Bommes et al. 2013; Campen et al. 2012]. All these methods are computationally expensive, and they need a heavy manual tweaking of the input parameters that most often have no intuitive relation to user desiderata.

Our work is most related to proposals in [Bærentzen et al. 2012; Bærentzen et al. 2014; Ji et al. 2010; Wu and Liu 2012; Zhang et al. 2007]. In such works, the surface of an object is seen as composed of: *tubular structures*, corresponding to branches of the curve skeleton; *caps*, corresponding to leaf nodes; and *connecting structures*, corresponding to branching nodes. Meshing tubular structures and caps is relatively straightforward, while connecting structures is the real challenge. In [Zhang et al. 2007], connecting structures are taken from a library of templates, depending on the underlying connectivity of the skeleton, while tubes are generated independently about branches. In [Ji et al. 2010], tubes are extruded first, while connecting structures are obtained from the convex hull of tube boundaries. In [Wu and Liu 2012], tubes are also built first, then a smooth cross field is imposed on the remaining portion of the surface and a quad mesh is laid over it by using an optimization method similar to [Ray et al. 2008]. In [Bærentzen et al. 2012], polyhedra are built first at branching nodes, then tubular structures are extruded from loops of edges of such polyhedra. In [Bærentzen et al. 2014] tubes are grown following a harmonic field over the surface, which is computed starting at user-defined poles: polar caps are built around poles and connections arise at meeting boundaries of tubes themselves. In this latter work, the skeleton is not taken as input, but it is rather generated upon the tubular structure resulting from user-specified poles. Meshes obtained from [Bærentzen et al. 2012; Bærentzen et al. 2014; Ji et al. 2010] are quad-dominant and tend to introduce irregular vertices of high valence at connecting structures; undesirable cluttering of many small faces also occurs around polar patches. The method of [Zhang et al. 2007] as well introduces irregular vertices of high valence. The method of [Wu and Liu 2012] provides no control on the placement of singularities and is just suitable for generating fine meshes. None of these works is explicitly addressing the problem of producing a coarse quad layout.

Mapping a 3D shape to a polycube [Tarini et al. 2004] is another way of generating a coarse quad layout. In fact, a pure quad subdivision can be trivially computed in the polycube domain, resulting in a mesh that naturally embeds a layout reflecting its topological structure. Extremely coarse layouts can be computed by carefully rounding polycube corners to integer locations. However, this operation is non trivial, as rounding corners may introduce both topological issues and additional distortion in the mapping. State of the art algorithms for polycube computation [Huang et al. 2014; Livesu et al. 2013] do not specifically address this problem and therefore

they tend to produce overly complex quad layouts (see Fig. 12, third and fourth row).

A closely connected and extensively studied problem is the construction of a surface parameterization defined over a quad-based domain. In [Madaras and Ďurikovič 2012] this task has been approached by decomposing the initial surface into tubular shapes, following a skeleton structure. The resulting parameterization, however, is designed for texture mapping and could not be easily used for remeshing.

Another recent trend concerns user-assisted techniques. In [Peng et al. 2011], local operators are provided to edit a quad mesh by moving around paired configurations of singularities. In [Peng et al. 2014; Takayama et al. 2013; Takayama et al. 2014] polygons sketched on the surface can be filled with patterns generated automatically on the basis of simple constraints provided by the user. In [Campen and Kobbelt 2014a] strips of edges are interactively laid on the surface along directions chosen by the system, allowing the user to quickly sketch quad layouts. These interactive methods can be seen as orthogonal and complementary with respect to our contribution, as they can be used to refine the model once a coarse quad layout has been automatically generated.

A review of the literature on curve-skeletons is beyond the scope of this paper, a description of the problem and of the various methods at the state of the art can be found in [Livesu and Scateni 2013; Tagliasacchi et al. 2012] and references therein.

3. METHOD

As discussed in Section 2, our objects consist of "tubes" meeting at branching elements. We aim at a pure quad layout that is aligned with tubes, so we can expect the meshing endowed by such layout to be fully regular along tubes, while singularities will appear at branching elements. In order to set the structure of such branching elements, we rely on the following observations from the common practice in interactive modeling of quad meshes:

—Most irregular vertices should have valence either 3 or 5; vertices with valence 6 can be useful along symmetry axes (as the collapse of two coincident vertices of valence 5), while higher valences are seldom used;

—Tubes with a quadrangular section are the most natural choice, since they can be extruded by cut-opening quad faces; such an extrusion introduces just four irregular vertices of valence 5;

—The basic modeling primitive is a hexahedral box, it provides a mesh covering a genus zero volume with a minimal number of irregular vertices (eight with valence 3), and it can be arbitrarily subdivided by "gridding" its faces without introducing further irregular vertices; note that the possibility to subdivide a box implies that an arbitrarily high number of tubes can be extruded from its faces.

We take as input a triangle mesh together with its curve-skeleton. The curve-skeleton can be computed with standard methods or it can be modeled interactively, as in the definition of an animation rig, and it must endow the overall structure of the object, which will be modeled by our coarse quad layout (Subsection 3.1). As typical in virtual sculpting, our initial primitives for modeling are subdivided hexahedral blocks, which are assigned to *branching nodes* of the curve-skeleton. The correct orientation of such blocks with respect to their incoming branches, which is crucial in defining the topology of the final quad layout, is set by resolving a local optimization problem (Subsection 3.2). We extrude other blocks along the branches, thus obtaining a "fattened skeleton" with quadrilateral faces (Subsection 3.3). This initial mesh contains T-junctions, which

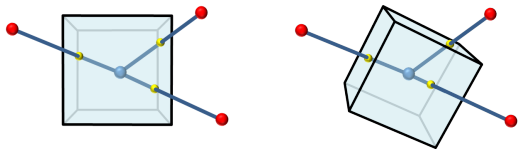


Fig. 2. An axis-aligned cube has all the three branches of the skeleton piercing the front face (left). Such a configuration would require splitting the front face in three facets. After reorienting it, the three branches pierce three different faces forming a "tripod" configuration, with no further splitting necessary (right).

are removed with a minimal subdivision by resolving an Integer Linear Programming (ILP) problem (Subsection 3.4). Next, we parameterize the input surface onto the surface of our base mesh to extract a quad layout (Subsection 3.5). All the previous parts are fully automatic, one-click, fast and do not require any parameter setting. Optionally, some choices can be modified or integrated by user interaction (Subsection 3.6).

3.1 Input

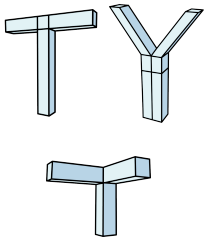
The input of our method is an object that can be roughly approximated by an assembly of *generalized cones*, that is surfaces that are swept out by moving a cross-section of smoothly varying size along an axis [Binford 1971; Marr 1980]. The high-level morphology of such an object can be successfully described by a *curve-skeleton*. Knowing also the diameters of *medial balls* tangent to the surface, or a complete mapping of the surface to its curve-skeleton, may allow for a more or less accurate reconstruction of the original surface, but this is not a stringent input requirement. In our experiments, we have employed different algorithms for extracting a skeleton, such as [Dey and Sun 2006; Livesu et al. 2012; Tagliasacchi et al. 2012], as well as manually designed skeletons (see Section 5).

For the rest of this paper, we will assume our input to be a triangle mesh $\mathcal{M}$ together with its curve-skeleton, represented as a graph $K_{\mathcal{M}}$, where each of its nodes has a position in 3D space. We classify the nodes of $K_{\mathcal{M}}$ as: *Joint nodes* (**Jn**) that have two incident arcs; *End nodes* (**En**) that have only one incident arc; and *Branching nodes* (**Bn**) that have more than two incident arcs.

3.2 Setting branching elements

We mimic the interactive modeling approach by subdivision-and-extrusion, by placing a skeleton-aware hexahedral box at each **Bn**, and allowing faces of such boxes to be possibly subdivided into *facets* in order to support the extrusion of an arbitrary number of tubes.

3.2.1 *Box alignment.* Branching nodes encode the way different tubular structures are connected. Typical, but not exhaustive, examples for three branches meeting together are T- and Y-branchings, and "tripods" (formed by branches that are nearly mutually orthogonal). Such configurations are easy to realize with cubic joints (see inset). Note that in Y-branching the upper branches pierce the same face of the **Bn** box, thus requiring this face to be split into two facets; such a split shall propagate along the complex to avoid T-junctions.



For branching nodes of higher degree, the combinatorics of configurations may become rather complex, so

we face the problem of selecting the most suitable configuration in a general way. We orient boxes at **Bn**'s trying to keep their faces as orthogonal as possible with respect to the directions of incoming branches (see Fig. 2).

In order to do this, we resolve the following optimization problem independently at each **Bn**. Given the branching node n_i and the set of k_i directions of its incident arcs $\{\mathbf{d}_1^i, \ldots, \mathbf{d}_{k_i}^i\}$, we search for the orthonormal basis $U_iV_iW_i$ that minimizes the function:

$$f(U_i, V_i, W_i) = \sum_{j=1}^{k_i} \left|\mathbf{d}_j^i \cdot U_i\right| + \left|\mathbf{d}_j^i \cdot V_i\right| + \left|\mathbf{d}_j^i \cdot W_i\right|$$

The **Bn** will then be a box axis-aligned to basis $U_iV_iW_i$. Note that each term of the summation achieves its minima when one of the three coordinate axes is aligned with the branch, and its maxima when the branch stabs any corner of the box. The summation acts as a trade-off for the (mis-)alignment of the different branches to the basis.

As can be noticed f contains absolute values, hence is not C^1 continuous. To improve the stability of the solution, we follow an approach similar to [Huang et al. 2014], minimizing the following function for $\varepsilon \to 0$ [El-Attar et al. 1979]:

$$f_\varepsilon(U_i, V_i, W_i) = \tag{1}$$
$$= \sum_{j=1}^{k_i} \sqrt{\left(\mathbf{d}_j^i \cdot U_i\right)^2 + \epsilon} + \sqrt{\left(\mathbf{d}_j^i \cdot V_i\right)^2 + \epsilon} + \sqrt{\left(\mathbf{d}_j^i \cdot W_i\right)^2 + \epsilon}$$

We encode this minimization as a non-linear constrained problem, by imposing that U_i, V_i and W_i are unit-length and mutually orthogonal, and we resolve it with Ipopt [Wächter and Biegler 2006]. We initialize the basis $U_iV_iW_i$ by computing the PCA of the set of directions $\{\mathbf{d}_1^i, \ldots, \mathbf{d}_{k_i}^i\}$. We start resolving the problem with $\varepsilon = 0.5$, and we take each solution as warm start for the next iteration halving the value of ε; four iterations are always sufficient for our purposes.

3.2.2 *Initial box subdivision.* We now have a set of oriented boxes associated to each **Bn**. Let B_i be the box associated to **Bn** n_i with k incident arcs. Each arc a_{i_k} incident at n_i is associated to the face of B_i pierced by the branch containing a_{i_k}. When more than one arc are associated to the same face of B_i, we split that face into *facets* following patterns allowing for tubular structures corresponding to different arcs to be extruded independently. To do this, we compute all the intersection points between a face f and its incident branches of skeleton: if f has b_f incident branches, we split f into b_f parallel rectangular facets. In order to decide the direction in which f will be split, we project its intersection points to the two cardinal directions parallel to the sides of f (in the $U_iV_iW_i$ basis), and we select the direction X_i on which such projection spans a larger interval. Each facet of f is then assigned to its corresponding incident branch. An example of split box is depicted in Fig. 3. Note that the resulting tessellation of the cube can have T-junctions (i.e., the two half-edges of the same edge are not split in the same number of portions); this issue is fixed later for the whole quad layout with a global process (see Subsection 3.4).

3.3 Extruding tubes

The next step is to place, along the paths defined by the branches of the skeleton, "tubes" with a quadrilateral section that connect either **Bn**-**Bn** pairs, or **Bn**-**En** pairs, with a minimal torsion. To this aim, we associate a square ring to each **Jn** and we concatenate such squares with longitudinal edges nearly parallel to the skeleton. The square ring associated to each **Jn** lies in a transversal plane, which is orthogonal to the tangent direction of the skeleton (trivially estimated as the average direction of its two incident arcs) while its rotation about the axis is set according to a Bishop frame [Bergou et al. 2008].

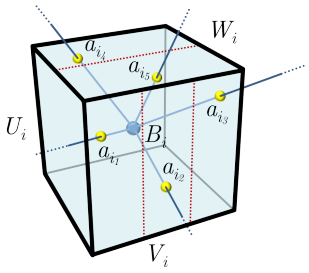


Fig. 3. A box with three branches (a_{i_1}, a_{i_2}, and a_{i_3}) piercing its front face, and two branches (a_{i_4} and a_{i_5}) piercing its top face. The front face is split into three facets along the V_i direction and the top face is split into two facets along the W_i direction. The resulting mesh has T-junctions.

3.3.1 *Branching node to End node.* For a **Bn**-**En** pair, we start at the **Bn** end of the branch, removing the facet pierced by the corresponding branch of skeleton, and extruding the tube along it. We set a 2D reference frame aligned with the sides of the removed facet; the Bishop frame is set at each point along the branch by parallel transport of this initial frame along the branch, through all **Jn**'s to the **En** (see [Bergou et al. 2008] for details). An open box is placed centered at the **En** and oriented according to the direction of the incident arc of skeleton and the Bishop frame. The tube is assembled by joining corresponding corners of the rings in chains, starting at the open facet of the **Bn** up to the open face of the **En**. The orientation of the 2D frames at all **Jn**'s along the path, as well as at the **En**, set the orientation of the rings about the branch, so that no torsion occurs along it (see, e.g., the nose and the ears in Figure 4).

3.3.2 *Branching node to Branching node.* For a **Bn**-**Bn** pair, we set the Bishop frame as before, by fixing the reference frame at one of the two ends. In general, there will be a mismatch between the orientation of the Bishop frame and the orientation of the branching box at the other end, for an angle θ on the plane of the frame. Such a mismatch induces a torsion on the tube that we distribute by interpolating θ along the branch and rotating the rings at intermediate **Jn**'s accordingly. This is easily done by arc-length

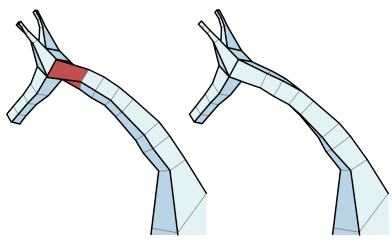


Fig. 4. If tubes are extruded along branches by using just the Bishop frame, all torsion occurs in a single red block (left image); we interpolate torsion along the neck, to minimize box-to-box torsion.

parameterization along the branch (see the neck in Fig. 4 for an example).

3.4 Conforming tessellation

The mesh built in the previous step is connected and watertight, but in general it can have T-junctions, because some edges of the **Bn**'s may have been subdivided differently at the two sides (Figure 5).

Previous approaches for T-junctions removal are able to generate a conforming mesh at the cost of two linear equations per face, imposing that opposite edges of the same face will be subdivided in the same number of intervals [Bommes et al. 2010; Tong et al. 2006]. However, these formulations are not general enough as they do not handle the case in which multiple faces are incident at both sides of the same edge (Figure 5). We propose here an alternative formulation that can handle such case and requires a remarkably smaller number of unknowns.

Both **Bn**s and **En**s are essentially cubes arbitrarily oriented in the space, therefore they naturally embed a 3D frame represented by the three orientations of its edges. For each such node we set three unknowns, representing the number of splits required along each direction in order to achieve conformity.

Tubes are cylinders with a quadrilateral section. For each tube we set two unknowns, representing the number of splits along the two edge directions of its cross-section.

The total number of unknowns required by our system is therefore $3n + 2m$, with n being the number of branch nodes and end nodes, and m the number of tubes. Note that this number relates to the *structure* of the model not to the *resolution* of the mesh, as any tube can be tessellated with an arbitrary number of quads along its length without affecting the size of the problem we are going to solve.

Tubes are connected to nodes through a face, or a portion of it. When a cube and a tube meet at a face they must agree on the number of subdivisions of their edges along two directions. Let us consider the example in Figure 5: the tubes $t^0, t^1 t,^2$ pierce the uv face of the cube C. In order to achieve conformity C and $t^0, t^1 t,^2$ must agree on the number of subdivisions along the directions u and v, generating four linear equations

$$\begin{cases} C_u = t^0_u = t^1_u = t^2_u \\ C_v = t^0_v + t^1_v + t^2_v \end{cases} \tag{2}$$

The face vw of C is pierced by the tubes t^3, t^4. In a similar manner, to achieve conformity C and t^3, t^4 must agree on the number of subdivisions along the directions v and w, generating three more equations

$$\begin{cases} C_v = t^3_v + t^4_v \\ C_w = t^3_w = t^4_w \end{cases} \tag{3}$$

To produce a conforming mesh we therefore solve for the number of subdivisions of each cube and each tube, generating a homogeneous linear system $AX = 0$. As we are only interested in non-trivial solutions that prescribe a positive number of subdivisions for each element we add a further constraint $X \geq \mathbf{1}$, where $\mathbf{1}$ is a vector with all entries at 1 and the inequality is intended component-wise, that is, each edge consists of at least one segment. We want to split each edge into a minimal number of sub-edges while satisfying the constraints above. This gives us the following ILP problem:

$$\begin{aligned} &\operatorname{argmin} \mathbf{1}^T X \\ &AX = 0 \\ &X \geq \mathbf{1} \end{aligned} \tag{4}$$

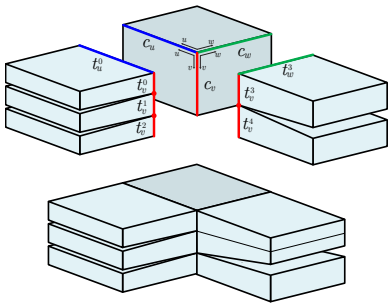


Fig. 5. We weld tubes on cubes in a conforming way (bottom) by imposing that the number of splits of a cube along each direction matches with the sum of the number of splits of each incident tube along the same direction.

We minimize the energy above by using [Gurobi Optimization 2013].

Note that our approach is inherently *volumetric*, in the sense that the number of subdivisions we prescribe generates surface meshes that can be trivially turned into full hexahedral meshes just by *gridding* the interior of each tube and each cube.

3.4.1 *Barriers.* The problem as described above admits a feasible solution for any object of genus zero, i.e., whenever the curve-skeleton is a tree. However, objects of higher genus may contain configurations that do not admit a solution, as depicted in Figure 6. These situations, called *barriers*, arise because of loops of constraints (corresponding to non-trivial cycles in the curve-skeleton) that cannot be satisfied simultaneously.

In short, we locate barriers and we remove each of them by splitting the volumetric complex at some point along its corresponding cycle, until the system becomes feasible. In the worst case, we split all cycles and the complex becomes of genus zero, hence a feasible solution is guaranteed. After a conforming complex is obtained, we re-join each split by glueing corresponding ends at a common grid; the mismatch between abutting ends generates *lids* on the surface of the complex, introducing new irregular vertices (see Figure 6).

In more detail, we proceed as follows. We first detect cycles in the graph having as nodes the feature points of the skeleton (leaves and branching nodes) and as arcs the skeleton paths connecting them. This is done at the cost of a depth-first traversal. Note that there is a one-to-one match between arcs and tubes. Next, for each such cycle c, we collect all the equations containing the variables associated to its tubes. Let A_c be the sub-matrix of A composed by these equations only. Then c is a barrier if and only if the ILP restricted to constraints in A_c also does not admit a solution.

Barriers can be broken by splitting the variables corresponding to cubes that appear along a cycle. Consider for instance the example in Figure 6. It contains a single loop consisting of cube C and tube t^1, and it corresponds to the equations shown in the figure, which also involve tube t^2. We can always break the barrier by splitting variables C_u and C_w into C'_u, C''_u and C'_w, C''_w, respectively, generating the new set of constraints as shown in the figure. After resolving the ILP problem for the new variables, the cube is split according to the larger value between each pair of solutions C'_*, C''_*. The end of the tubes corresponding to the equation that involves the other variable are attached to the cube to cover a sub-grid of facets of the subdivided cube. The other portion of the face of the cube is now on the surface, tagged *lid* in Figure 6.

The cube at which a given barrier is broken can be selected either automatically, or by the user; in our implementation, we arbitrarily selected the cube with the least number of tubes incident to it. Note

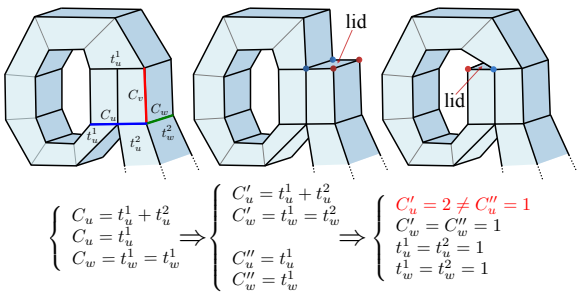


Fig. 6. The mesh on the left has barriers: constraints coming from the top and the bottom faces of the cube cannot be satisfied at the same time; the barrier is broken by splitting variables C_u and C_w corresponding to the blue and green axes; two possible solutions ca be found, as depicted on the center and right, which add a transversal facet (a lid) that introduces four more irregular vertices: two of valence 3 (in red) and two of valence 5 (in blue). At the bottom, we show the system of constraints that characterize the barrier, the way it is modified by splitting variables and the resulting solution: the equations in red show the mismatch between values assigned to a split variable.

also that there are degrees of freedom in matching the split variables, which can bring to different solutions, as shown in the figure. In our implementation, we just take one such solution in a systematic way (by setting a local reference system for each face of the cube and selecting the sub-grid at its lower-left corner. Additional geometric criteria could be used to select an optimal matching among all possible solutions.

Note that barriers seldom occur in practice. In fact, most generalized cylinders are of genus zero, and also higher genus objects usually do not induce barriers.

3.5 Coarse quad layout and mapping

The conforming quad-mesh we obtained can be further coarsened by removing all the edge loops that are transversal to the tubes. The resulting quad-mesh $\mathcal{Q}$ bears the topology of the final pure quad layout we want for the original mesh $\mathcal{M}$. To conclude our pipeline we need to compute a parameterization of $\mathcal{M}$ over $\mathcal{Q}$, that is, to define a bijective mapping between the two. To do this we assign to each vertex v of $\mathcal{M}$ a position on $\mathcal{Q}$, which is specified by an index i of a quad q_i of $\mathcal{Q}$ and a pair of parametric coordinates (normalized in $[0,1] \times [0,1]$) inside q_i. This per-vertex assignment implicitly partitions $\mathcal{M}$ into quadrilateral *regions*, each corresponding to a quad of $\mathcal{Q}$. Edges of $\mathcal{Q}$ are implicitly mapped into *arcs* traced over $\mathcal{M}$, which separate the regions but are not, in general, composed of edges of $\mathcal{M}$. Similarly, vertices of $\mathcal{Q}$ are implicitly mapped to general positions over $\mathcal{M}$, not necessarily vertices.

The final mapping is constructed in two phases: first we compute a raw assignment by projecting $\mathcal{Q}$ onto $\mathcal{M}$, then we improve the mapping by an interleaved iterative process of refinement and resizing.

3.5.1 *Initial mapping.* The raw assignment does not need be accurate: it can produce fold-overs, distortions, and suboptimal positioning of region boundaries on $\mathcal{M}$. We proceed as follows:

Skeleton "fattening": an embedding of $\mathcal{Q}$ is obtained placing its boxes centered at the nodes of the curve-skeleton, orienting them as described in the previous sections, and sizing them according to the radii of medial balls; in case we don't have the radii of medial balls available, their sizes are roughly estimated as the distance between each node of the skeleton and its closest point on $\mathcal{M}$; this creates a sort of polymerized "fattened skeleton" around the curve-skeleton.

Fig. 7. Left-most: original mesh $\mathcal{M}$ color-coded according to the quads in $\mathcal{Q}$(left). Then: examples of different assemblages of macro-regions. Triangles of $\mathcal{M}$ which span multiple macro-regions (which are frozen during relaxation) are shown darkened.

Initial projection: each vertex of $\mathcal{Q}$ is projected to $\mathcal{M}$ along the surface normal estimated on the fattened skeleton averaging neighbouring faces; if projection fails, the closest point on $\mathcal{M}$ is selected.

Subdivision: each face q_i of $\mathcal{Q}$ is iteratively subdivided using the Catmull-Clark subdivision scheme, obtaining a corresponding gridded sub-domain q_i; at each iteration of the subdivision process, newly created vertices are projected to $\mathcal{M}$ as in the previous step.

Back-projection: $\mathcal{M}$ is projected over each q_i, and, thus, over $\mathcal{Q}$, using the same strategy as in the previous steps.

Finally we remove the edge loops along the tubular structure. From now on the geometry of $\mathcal{Q}$ is no longer relevant: in any further processing, $\mathcal{Q}$ is considered only as a collection of 2D rectangles, complete with adjacency information. In this sense our domain $\mathcal{Q}$ is "abstract", as defined in [Pietroni et al. 2010].

3.5.2 *Refinement.* The refinement phase optimizes the parametrization in the interior of each region, as well as the locations and shape of the arcs and the positioning of vertices of the layout in $\mathcal{M}$. It also solves the fold-overs potentially created in the initial mapping. This phase closely follows the technique proposed in the second part of [Tarini et al. 2011], that we recap here just for completeness. Alternatives could be employed equally well, as [Campen and Kobbelt 2014b].

The procedure consists of an iterative relaxation of the positions in $\mathcal{Q}$ assigned to vertices of $\mathcal{M}$. At each iteration, $\mathcal{Q}$ is partitioned in a number of "macro-regions", each consisting of a small groups of adjacent quads of $\mathcal{Q}$, having a disk-like topology (see Fig. 7): a macro-region can consist of either the 1-ring of a vertex of $\mathcal{Q}$("vertex" macro-region), or two quads of $\mathcal{Q}$ sharing an edge ("edge" macro-region), or a single quad ("face" macro-region). Parametrization inside each macro-region is optimized by means of Mean Value parametrization [Floater 2003], while keeping boundary vertices fixed. Specifically, we freeze in place all vertices of any triangle not entirely falling inside a single macro-region. At the end of the iteration, the macro-regions are disassembled back into the original quads of $\mathcal{Q}$. Note that relaxation may move a vertex of $\mathcal{M}$ to a different quad of $\mathcal{Q}$.

Different assemblies of macro-regions are used at each iteration (see Fig. 7), so that each vertex v of $\mathcal{M}$ eventually undergoes optimization (i.e., appears in the interior its macro-region). At every iteration, the partition is constructed with a simple greedy algorithm: we incrementally assemble quads into macro-regions until no quad is left; for each vertex and each edge of $\mathcal{Q}$, we keep track of the last iteration at which that element underwent relaxation (i.e. was not on the border of its macro-region); first we assemble "vertex" macro-region, starting with the ones constructed around vertices of $\mathcal{Q}$ which has been frozen on the border for the most iterations;

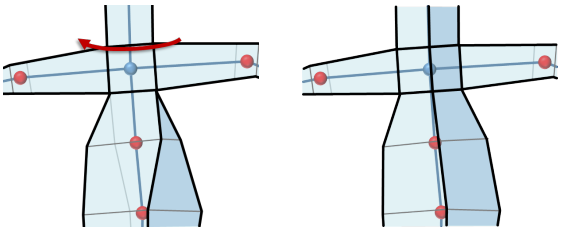


Fig. 8. The automatic cube orientation is generating an unnatural torsion (left). The user can pick the cube, select a rotation axis, and adjust it (right).

remaining quads are assembled into "edge" macro-regions, prioritizing with the same criterion; lastly, singled-out quads form "face" macro-regions.

The relaxation performed in the interior of the macro-region also serves the purpose of unfolding fold-overs which are potentially present in the initial mapping: if necessary, this can be enforced in the optimization system by adding linear constraints, after [Bommes et al. 2013]. Nothing guarantees that every folded configuration will fall entirely inside one macro-region in at least one partition (so that it can be fixed), but in our experiments we never encountered any failure case. If need arises, ad-hoc countermeasures could be easily designed to forcefully assign the entire folded configuration (and its neighboring vertices) to a single macro-region.

After [Khodakovsky et al. 2003], in "vertex" macro-region constructed around an irregular vertex of $\mathcal{Q}$, quads are transformed by means of an exponential mapping, in order to flatten them on a plane. Although this mapping is perfectly conformal in the continuous case, it can occasionally increase the conformal energy, or even introduce new fold-overs, due to the discrete nature of the mesh; this is a minor (and rare) technical difficulty which could always be solved with a local and temporary refinement of $\mathcal{M}$; more practical countermeasures, which we employed, consist in freezing or rolling back newly created folded triangles, and rolling back any relaxation process which results in a global increase of conformal energy.

We stop the process as soon as the increase of the global conformal energy does not exceed a predefined threshold. Each iteration cannot decrease the total conformal energy, which is preserved by all the mappings to and from macro-regions, thus guaranteeing convergence.

3.5.3 *Resizing.* We employ an additional optimization, interleaved with the above, to determine the relative sizes of the abstract quads in $\mathcal{Q}$. This will be useful later on to extract a semi-regular quad mesh (see Section 4.1).

First, the ideal aspect ratio r_i is found separately for each quad q_i, striving to minimize the total conformal energy of [Lévy et al. 2002]. This sub-problem is solved in closed form for each interior triangle and area-averaged for each quad.

Next, we find a globally consistent assignments of the widths and heights of all quads, which better satisfies the aspect ratios, up to a global scale. Recall that, in our abstract domain, an adjacency defined between two sides of a pair of quads implies the equality of their lengths. We construct a system where each variable v_1, v_n corresponds to either the width or the height of a quad, and we reduce the set of variables with the above equalities. If a given quad q_i v_w width and v_h height, we would like $v_w/v_h = r_i$, which we rewrite as $\log v_w - \log v_h = \log r_i$. We solve the corresponding overdetermined linear system in the least squares sense (with one variable arbitrarily set to the unit) to recover the logs of all the variables, and thus the widths and heights of each quad.

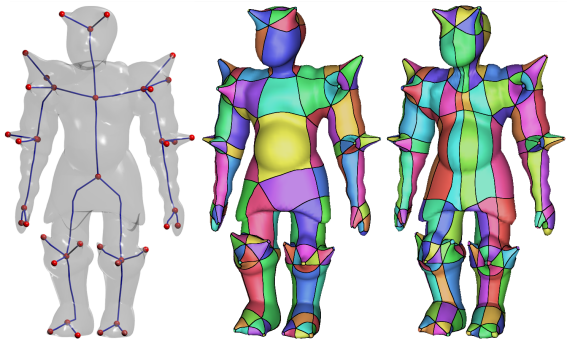


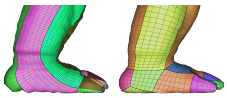
Fig. 9. Left: curve-skeleton for the Warrior dataset. Middle: the quad-layout produced by our framework with automatic box orientation (Section 3.2.1). Right: an alternative quad-layout produced by manually reorienting branching boxes, leveraging the user interaction so as to emphasize the symmetry of the layout (Section 3.6.1).

3.6 User interaction

The whole pipeline described so far is fast enough to be executed at interactive frame rate on meshes of moderately large size. This allows an end-user to integrate our pipeline, which is otherwise fully automatic, with optional interactions aimed to control the choices taken by the system. Specifically, we designed three forms of interaction.

3.6.1 *Reorienting branching nodes.* The user can select and reorient a single **Bn** to change the layout of boxes, hence the structure of the coarse quad layout (see Fig. 8). In this way the output can be made to comply to additional desiderata, like for example the preservation of symmetry in the quad layout (see Fig. 9).

3.6.2 *Adding branching nodes.* By construction we consider only **Bn** having at least three incident arcs. It can be useful, to account for sharp variations of the shape to insert **Bn** with only two exiting branches piercing two adjacent faces of the box. Recall that the skeleton captures the topology of the tubular parts without taking into account bending. Therefore, a branch is always modeled, by default, with a regular sequence of boxes, whose geometry is properly adjusted to follow the skeleton.



In some cases, is it convenient to break the regularity by inserting a new **Bn**, as depicted in the inset where the heel is remeshed. This choice depends most often on semantics of the modeled object, and, therefore, we leave to the user the possibility to impose it. For instance, the trunk of an elephant is probably always best modeled with a straight sequence of boxes, no matter how sharply it bends, while an ankle of a human always requires an L-joint, even if the foot is extended.

3.6.3 *Retouching parameterization.* The end user can retouch the mapping during its optimization (Subsection 3.5), by constraining a vertex v of $\mathcal{Q}$ to be mapped over a specific position p on the mesh, assuming that p is currently mapped on a quad of $\mathcal{Q}$ which is adjacent to v. We express p as a triangle index and barycentric coordinates inside it. To fulfill the constraint, we force the image of p to be mapped to the center of the macro-region built around v, in any subsequent optimization iteration that is performed on that macro-region. This translates to a simple linear constraint which can be included in the optimization.

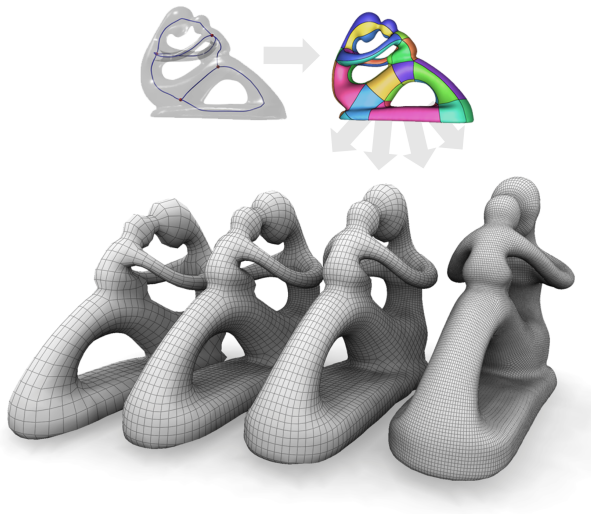
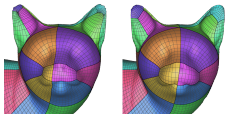


Fig. 10. Examples of semi-regular quad meshes at arbitrary resolutions that can be trivially extracted from our coarse quad layout (top-right); the curve skeleton which served as input (top left).



Note that, in all other iterations, the triangle containing v is already fixed on the boundary. This interaction can be useful for both the applications we devise for our technique: to pin-point the exact location of irregular vertices of a remeshing (Subsection 4.1), or to get some control over the locations of seams of a UV-mapping (Subsection 4.2).

4. APPLICATIONS

We present applications of our technique to semi-regular quad meshing and UV-mapping. Each of these applications requires further processing, exploiting the structure of the pure quad layout that we have built so far.

4.1 Semi-regular quad meshing

We generate a quad remeshing of the original mesh, having the same set of irregular vertices of the corresponding quad layout, by performing a regular sampling over the parametric domain $\mathcal{Q}$. The resolution is arbitrary and it is determined by a user-defined scaling factor s: the widths and heights of $\mathcal{Q}$, as computed in Section 3.5.3, are scaled by s and then rounded to the nearest integer value.

The vertices of the remeshing mesh are placed at integer locations of the parameterization. This is simply done traversing each face of $\mathcal{M}$ once, and producing all the vertices inside it with a rasterization approach. Such vertices are connected with a natural grid connectivity.

By construction, produced remeshings are pure-quad, conforming, and semi-regular (in the sense of [Bommes et al. 2013]). See Fig. 10 for an example.

4.2 UV-mapping

For texture mapping applications, the coarse quad layout directly provides a parameterization domain for UV-mapping. The UV-mapping can be computed equally well either for a quad remeshing (as seen above), or for the original triangle mesh $\mathcal{M}$. In the latter case, we need to split few triangles of $\mathcal{M}$ in order to accommodate the necessary seams.

The rectangles of $\mathcal{Q}$ must be packed inside the global texture rectangle, by means of 2D translations of an integer number of texels, plus 2D rotations by a multiple of $\pi/4$.

One can use a simple rectangle-packing heuristic, as at the top left of Fig. 11. It is also possible to attach a few rectangles side by side, when the corresponding quads in $\mathcal{Q}$ are adjacent, in order to reduce the number of texture seams, as in the bottom left and right part of Fig. 11. These tasks, which have a direct equivalent in typical UV-mapping construction (either by artists or by automatic approaches), are made easy by the extremely coarse nature of $\mathcal{Q}$.

To choose which rectangle to attach, we adopt a simple heuristic. First, we assign a measure of "importance" to each edge of $\mathcal{Q}$, where the "important" edges are the ones which should not become seams. In our experiments we have used the length, but any semantic-based criterion can be easily embedded. Next, we incrementally assemble rectangles into sets of "pieces", joining them side by side. We analyze each edge e once, in descending order of importance: if the two rectangles incident at e belong to different pieces, we try to join them, but we reject the operation if it causes any overlap in UV space (note that a single join operation can merge several edges). Finally, we resort to a simple packing of bounding rectangles of the final pieces.

By construction, our UV-mappings are "griddable": they have texel grids aligned across chart boundaries, conferring to the texture mapping the ability to hide seams even in presence of bilinear interpolation and MIP-mapping, as shown in [Ray et al. 2010].

5. RESULTS

We tested the method described on more than twenty different models, including different poses of the same shape. Some results are summarized in Table I and Figures 1, 10, 11, 13, 14, and 15. Further results are available in the supplemental material.

For all these models we were able to produce a coarse quad layout during interactive sessions. The sessions took from less than one minute for simplest models (e.g., the cactus), to few minutes for the more complex ones, which needed a finer reorientation of the **Bn**'s. In all cases, the automatic part of the processing and the feedback after each user operation are almost immediate, up to the creation of the fattened skeleton and coarse quad layout topology. This is also demonstrated in the video available in the supplemental material. The parametrization part can take a few seconds, depending on the size of the input mesh, but this is needed just once, after the user is satisfied with the topology of the quad layout. Parametrization retouching is almost immediate and fully compatible with user interaction, too.

Our coarse quad layouts, as well as any quad mesh obtained out of them, have most irregular vertices with valence 3 or 5, seldom with valence 6. In our experiments, we have also processed a few objects of genus higher than zero and we never experienced the presence of barriers.

It is worth to notice that the user does not need to set up any parameter and that interaction is purely based on the appearance of the base complex. In the last two columns of Table I we report some summary information on the number of user actions needed to compute the final results. A visual comparison between a layout automatically computed by our system and a layout edited by the user is given in Figure 9. Note that most models required either one or no box reorientation and that almost all L-joints inserted by box addition occurred at heels. These figures just give a hint on the simplicity of the overall procedure we set up.

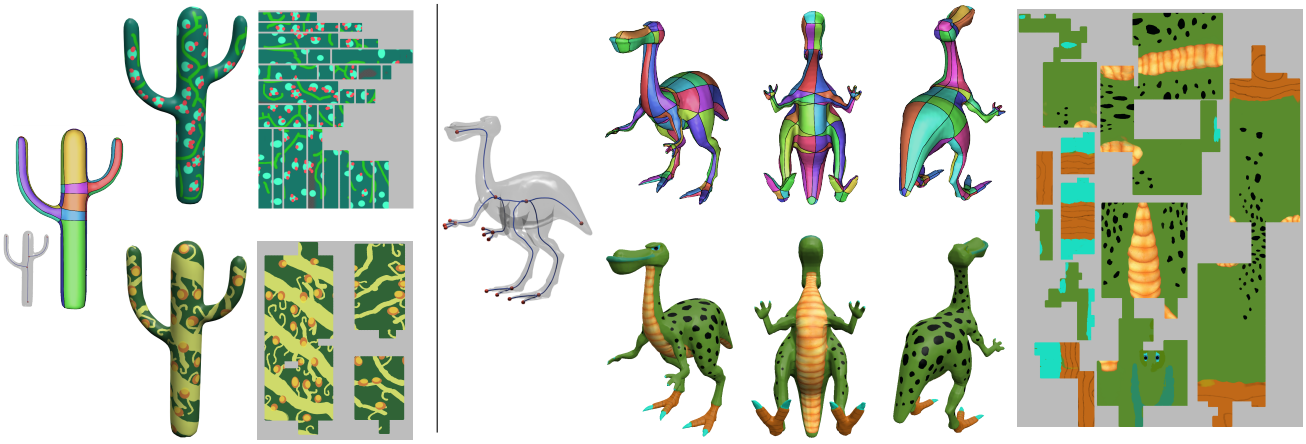
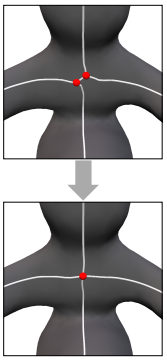


Fig. 11. Examples of UV-mappings obtained from our coarse domains. Left: two alternative UV-mappings for the Cactus dataset, resulting from a trivial packing of the abstract domain (top) and from with the simple packing heuristic described in Section 4.2 (bottom). Right: a more complex example, for the Dinopet dataset. All demonstrative textures were manually painted by artists using standard software.



We used four different algorithms for the computation of the curve-skeletons. Armadillo, Cat, Cactus, Dinopet, Rocker Arm, Dinosaur, Octopus, Horse and Guy were skeletonized using [Livesu et al. 2012]; Hand using [Livesu and Scateni 2013]; Fertility and Elk using [Tagliasacchi et al. 2012]; Vessel, Dragon, Centaur, Big Buddy, Joker and Warrior using [Dey and Sun 2006]. For [Tagliasacchi et al. 2012] and [Dey and Sun 2006] whenever necessary we manually merged nearby branching nodes so as to achieve a one-to-one relation between skeleton arcs and the logical components of the 3D shape (see inset aside) whereas [Livesu et al. 2012] and [Livesu and Scateni 2013] already embed a heuristic for the automatic simplification of the skeleton.

5.1 Quad layout comparisons

We made extensive comparisons of our results with other methods aimed at producing a pure quad layout from a mesh of triangles, using the number of domains and the number of singularities as a quality metric. Comparisons with two models used in all the other works are shown in Table II and Fig. 12. Note that resuts from [Livesu et al. 2013] and [Huang et al. 2014] have been produced with the default parameters; parameter tuning might reduce the number of domains, but any method restricted to polycube topologies is likely to result in more domains than our method, on models with a good skeleton. We provide further comparisons in the additional material. Note that, even if our method is designed as a support for animation, and thus mostly usable on humanoid or animal models, it outperforms the state-of-the-art even on a mechanical model like Rockerarm and a model with high genus like Fertility.

We report a separate comparison with quad layouts extracted from the PAM models of [Bærentzen et al. 2014]. Such models are just quad-dominant and have most irregular vertices of valence 6 or 8 at branching structures and vertices of high valence at poles. For the sake of comparison, we obtain a quad layout from a PAM as follows: we first remove the stars of poles and we build one domain per pole; then we extract domains from the remaining pure quad mesh by tracing separatrices incident at irregular vertices. A comparison on the Armadillo dataset is depicted in Fig. 13. The PAM layout contains 191 domains, 18 of which are polar patches; it contains 22 non-polar irregular vertices, all of which have valence at least 6, and 18 poles with valences between 4 and 36. Our layout contains 216 domains, 72 vertices of valence 3, 48 vertices of valence 5 and 8 vertices of valence 6. In spite of a slightly higher number of domains and singularities, our method produces a much more uniform layout and contains no vertex with high valence. Transforming a PAM into

Table I. Summary of experiments. For each model we show: the number of faces in the triangle mesh (rounded to thousands); the number and type of skeleton's nodes; the number of domains and irregular vertices (valence 3, valence 5, valence 6 or more); the interactive operations of the user for reorienting and adding **Bn**'s, if any. The name of each model comes together with a reference to the figure showing it.

Model	kTri's	Skel. nodes (**Bn**,**En**,**Jn**)	Domains	Valence (3, 5, 6+)	User inter. Reor.	Add
Armadillo[13]	11	(7,18,50)	216	(72,48,8)	3	2
Big Buddy[15]	27	(4,11,66)	132	(44,24,6)	–	2
Cactus[11]	10	(2,4,22)	30	(16,8,0)	–	–
Cat[15]	56	(3,7,29)	64	(30,10,6)	2	–
Centaur[15]	31	(5,16,72)	166	(64,32,12)	1	–
Dinopet[11]	9	(6,12,79)	136	(48,24,8)	3	3
Dinosaur[14]	4	(2,6,191)	38	(24,8,4)	–	–
Dragon[15]	100	(25,47,121)	604	(188,132,24)	8	–
Elk[14]	48	(5,7,33)	72	(28,20,4)	–	–
Fertility[12]	37	(4,0,45)	46	(2,26,0)	–	1
Guy[15]	27	(4,5,44)	90	(34,22,2)	1	2
Hand[1]	21	(2,6,15)	48	(24,12,2)	–	–
Horse[15]	17	(3,8,36)	70	(32,12,6)	1	–
Joker[15]	27	(7,13,83)	154	(56,40,4)	1	2
Octopus[15]	33	(1,9,112)	56	(36,4,12)	–	–
Rockerarm[12]	20	(3,3,27)	28	(12,12,0)	1	–
Vessel[14]	99	(14,19,141)	304	(81,63,6)	1	–
Warrior[9]	27	(13,28,82)	234	(112,48,28)	4	2

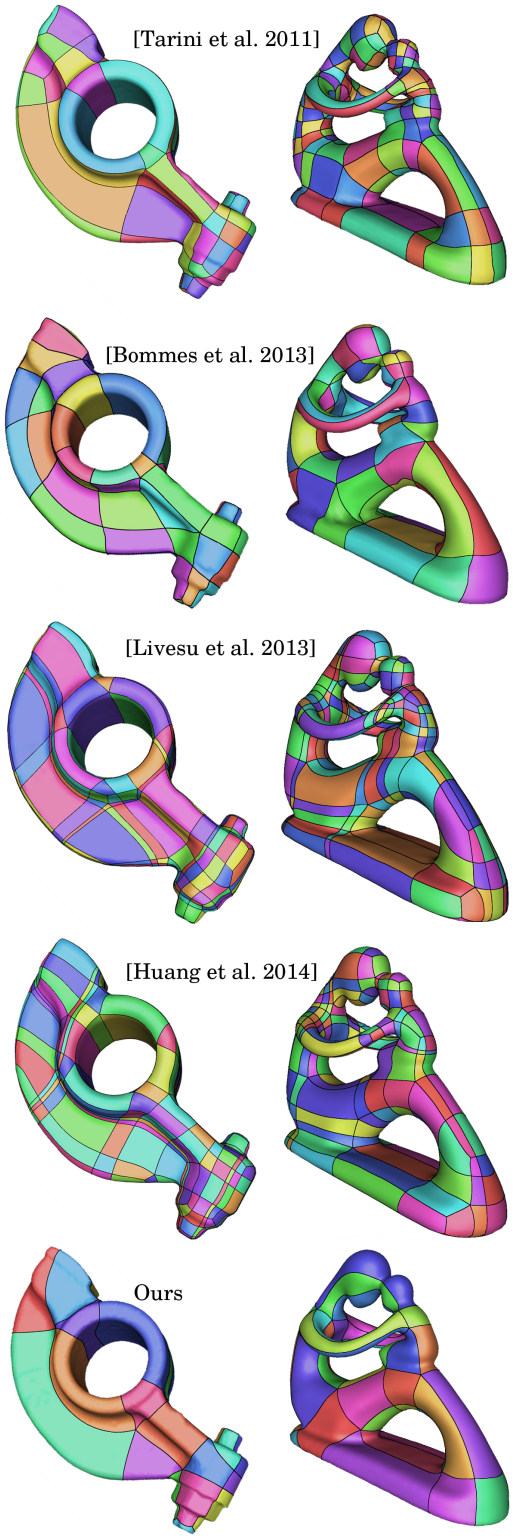


Fig. 12. Visual comparison of the quad layouts listed in Table II.

Table II. Comparison on quad layouts obtained on the reference models most used in literature; we report the number of domains of the quad layout, and the number of irregular vertices with their valence (†one valence 6 vertex).

Model	Domains	Val 3	Val 5
Fertility			
[Tarini et al. 2011]	253	12	36
[Bommes et al. 2013]	85	12	36
[Livesu et al. 2013]	420	35	59
[Huang et al. 2014]	288	36	†59
Ours	46	2	26
Rockerarm			
[Tarini et al. 2011]	98	18	18
[Bommes et al. 2013]	66	15	15
[Livesu et al. 2013]	352	31	31
[Huang et al. 2014]	426	32	32
Ours	28	12	12

a pure quad mesh may require one step of subdivision, which would introduce further singularities of valence 3 and fragment the quad layout into many more domains.

5.2 Remeshing comparisons

In Fig. 14 we compare our results with some state-of-the-art methods designed to produce a good quad remeshing. We also report histograms showing closeness of angles in the resulting quad mesh to a right angle. Further numerical results on the Rockerarm and Fertility models are reported in Table III. Note that while all methods tend to produce faces with nearly right angles on average, our method usually achieves a smaller relative standard deviation (RSD) in the distribution. In the Dinosaur dataset (middle of Fig. 14), although the histogram bell of [Zhu et al. 2015] is slightly narrower, their mesh has a number of quite sharp angles, resulting in a higher RSD (15.83% against our 7.78%). Only in the Vessel dataset shown in the bottom part of Fig. 14, the method of [Zhu et al. 2015] achieves a slightly better distribution of angles, but our layout contains about half a number of domains and they appear to be much better distributed on the surface of the object.

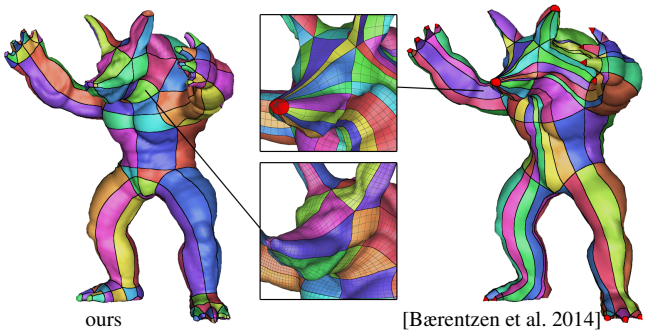


Fig. 13. Comparison between a quad layout obtained with our method (left), and a layout extracted from a PAM model (right). Polar patches of the PAM layout are depicted in red.

Table III. Comparisons on remeshing of two reference models most used in the literature; we report the average value of angles of faces and the relative standard deviation (RSD).

Model	Avg	RSD
Fertility		
[Bommes et al. 2009]	89.86	9.29 %
[Tarini et al. 2011]	89.92	12.21 %
[Bommes et al. 2013]	89.91	18.22 %
[Livesu et al. 2013]	89.98	19.22 %
[Huang et al. 2014]	89.95	10.91 %
Ours	89.97	6.96 %
Rocker arm		
[Bommes et al. 2009]	89.95	8.79 %
[Tarini et al. 2011]	89.97	7.50 %
[Bommes et al. 2013]	89.92	18.57 %
[Ebke et al. 2013]	89.74	9.21 %
[Livesu et al. 2013]	89.98	16.24 %
[Huang et al. 2014]	89.96	10.82 %
Ours	89.96	8.45 %

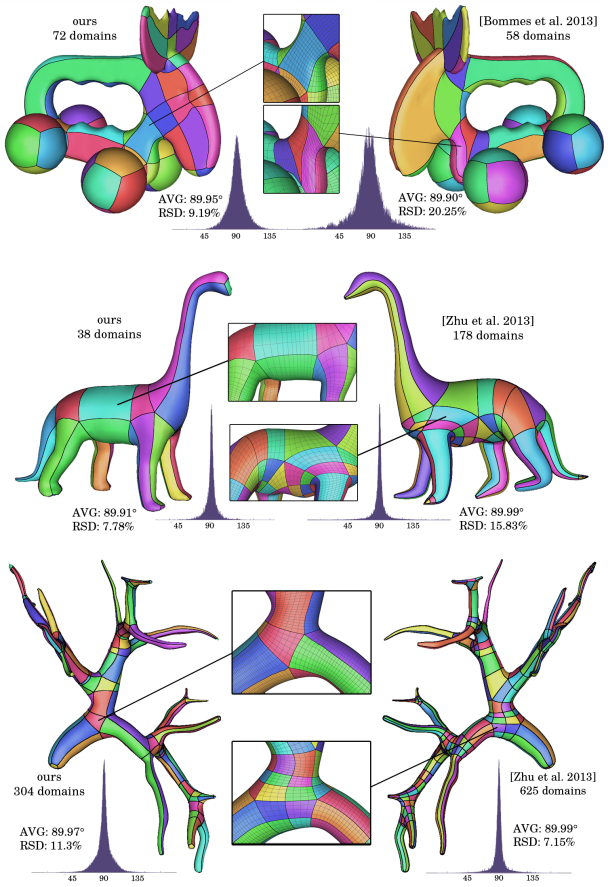


Fig. 14. Visual and numerical comparisons with [Bommes et al. 2013] and [Zhu et al. 2015]. Images on the right side are mirrored to better show correspondences.

6. CONCLUDING REMARKS

We have presented a novel method for computing a coarse quad layout that fits an input surface. Our method is simple to use, supports friendly user interaction and is designed to complete the transformation pipeline in minutes. Relying on the curve-skeleton, it is able to capture the structure of the original shape and to cover its surface with quad domains aligned to its elongated parts. Our quad layout supports easy extraction of semi-regular pure quad meshes at user-defined resolution, and domains for UV-mapping. Meshes generated with our method have few singularities of low valence. Interactive techniques to refine the layout and sculpt finer details of the mesh can be easily integrated.

6.1 Limitations

Our method is intended for shapes defined as assembly of generalized cones, a class of objects suitable for models used in animation. Generally speaking, all shapes that cannot be described well with a curve skeleton are out of the scope of our method. For the same reasons we are not able to process shapes with cavities, like a cup. We, for instance, tried to apply our method to the Botijo mesh, but have been unable to perform the initial mapping step due to the intrinsic characteristics of the shape. However, as shown in Section 5, reasonable results can be obtained even on some objects not directly intended for animation.

We did not address alignment of the quad layout to sharp creases (e.g., the border of ears in cat, horse and armadillo, the edge of the sunshade in guy). This could be added by snapping creases to borders of domains during parametrization, as proposed in [Tarini et al. 2011]. Multi-scale detection would be necessary to capture just creases that are relevant at scales coarse as the desired quad layout, while disregarding fine details.

6.2 Future work

There are several interesting extensions that can be added to our interactive system. Following [Bærentzen et al. 2014], it can be extended to a modeling system, by giving the user the possibility to edit the curve-skeleton associated to the original mesh, extruding branches, cloning parts, or specifying the properties of each branch (e.g., the radii of medial balls). The interactive session can use symmetry planes or points to modify the skeleton and, thus, the base complex and the quad layout. The base complex can be used to model a cage around the shape for animation purposes. Or it can be taken as the control mesh of a subdivision surface: optimal positions of nodes of the control grid can be found easily, such that the limit surface fits the original mesh; fine detail can be easily added via displacement mapping. Consistent quad layouts of different poses of the same object can be easily obtained (see results in the supplemental material): this provides the basis for a cross-parametrization, which can be used for the purpose of animation. The quad layout $\mathcal{Q}$ resulting from our method can be also seen as the outer surface of a solid mesh of hexahedra: in order to obtain a geometric hexahedral mesh, one should assign 3D positions to all the internal vertices (e.g., by computing harmonic functions that use the coordinates of the boundary vertices as Dirichlet boundary conditions).

Finally, we observe that the problem of generating high quality curve-skeletons is still open. While state-of-the-art methods are capable of extracting meaningful skeletons from complex shapes they often fail at understanding the shape at a higher level, generating a robust one-to-one correspondence with its logical components. We therefore plan to further investigate this problem, in order to benefit

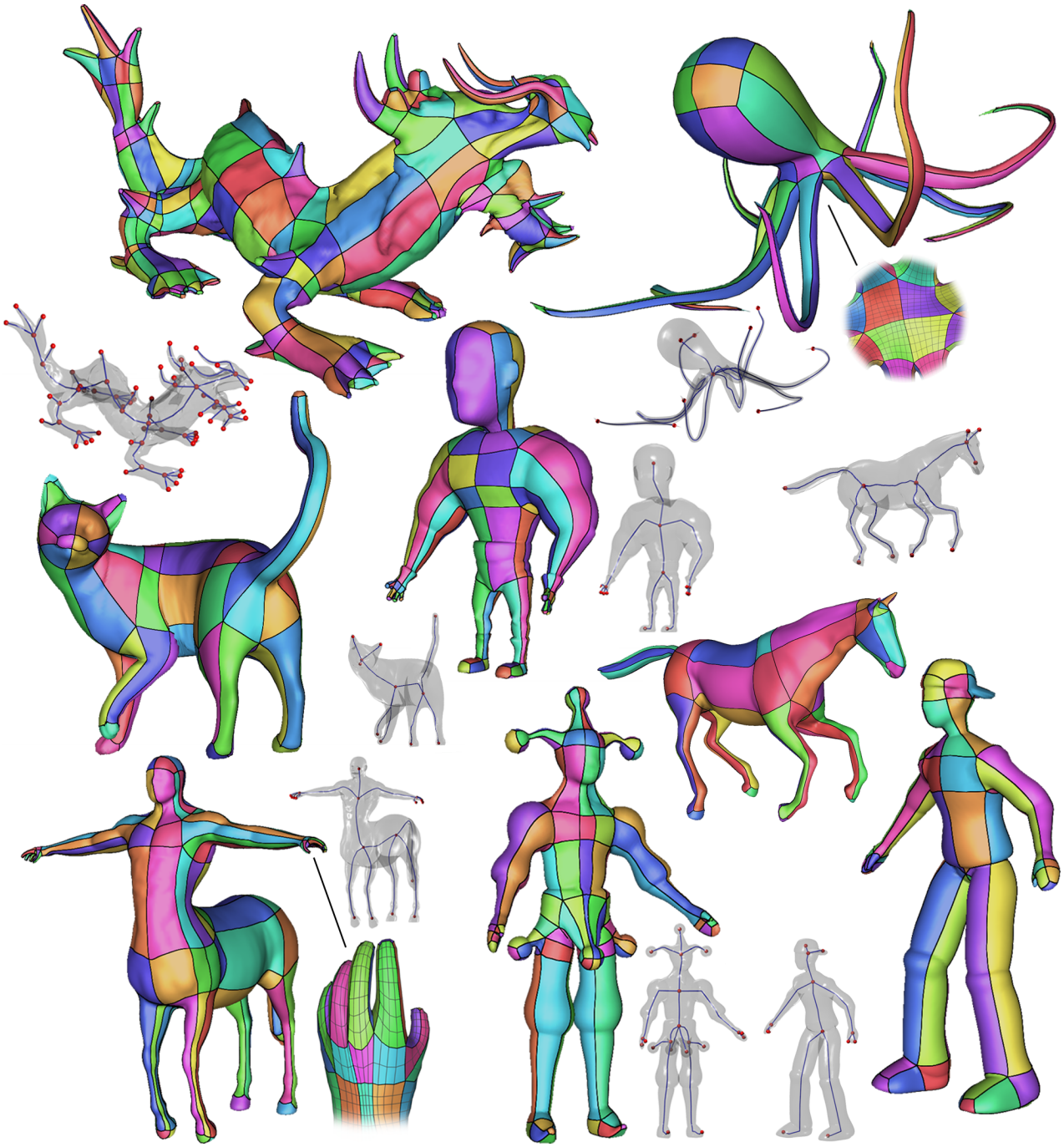


Fig. 15. A gallery of coarse quad layouts computed with our method. For each model we also show the curve-skeleton used as input (see also additional examples in figures 1, 13, 10, 11, 9 and 14).

our algorithm as well as any other method that requires such a high level of abstraction of a 3D shape.

ACKNOWLEDGMENTS

This work has been partly developed while Marco Livesu was at University of British Columbia, Vancouver, BC, Canada.

REFERENCES

AUTODESK. 2007. Mudbox. `http://www.autodesk.com`.

BÆRENTZEN, J., MISZTAL, M., AND WEŁNICKA, K. 2012. Converting skeletal structures to quad dominant meshes . *Computers & Graphics 36,* 5, 555 – 561. Shape Modeling International (SMI) Conference 2012.

BÆRENTZEN, J. A., ABDRASHITOV, R., AND SINGH, K. 2014. Interactive

Shape Modeling Using a Skeleton-mesh Co-representation. *ACM Trans. Graph. 33,* 4 (July), 132:1–132:10.

BERGOU, M., WARDETZKY, M., ROBINSON, S., AUDOLY, B., AND GRINSPUN, E. 2008. Discrete Elastic Rods. *ACM Trans. Graph. 27,* 3 (Aug.), 63:1–63:12.

BINFORD, T. O. 1971. Visual Perception by Computer. In *Proceedings of the IEEE Conference on Systems and Control (Miami, FL)*.

BOMMES, D., CAMPEN, M., EBKE, H.-C., ALLIEZ, P., AND KOBBELT, L. 2013. Integer-grid Maps for Reliable Quad Meshing. *ACM Trans. Graph. 32,* 4, 98:1–98:12.

BOMMES, D., LEMPFER, T., AND KOBBELT, L. 2011. Global Structure Optimization of Quadrilateral Meshes. *Comput. Graph. Forum 30,* 2, 375–384.

BOMMES, D., LÉVY, B., PIETRONI, N., PUPPO, E., SILVA, C., TARINI, M., AND ZORIN, D. 2013. Quad-Mesh Generation and Processing: A Survey. *Comput. Graph. Forum 32*, 51–76.

BOMMES, D., VOSSEMER, T., AND KOBBELT, L. 2010. Quadrangular Parameterization for Reverse Engineering. In *Proceedings of the 7th International Conference on Mathematical Methods for Curves and Surfaces*. MMCS'08. Springer-Verlag, 55–69.

BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-integer Quadrangulation. *ACM Trans. Graph. 28,* 3, 77:1–77:10.

CAMPEN, M., BOMMES, D., AND KOBBELT, L. 2012. Dual loops meshing: quality quad layouts on manifolds. *ACM Trans. Graph. 31,* 4, 110:1–110:11.

CAMPEN, M. AND KOBBELT, L. 2014a. Dual Strip Weaving: Interactive Design of Quad Layouts Using Elastica Strips. *ACM Trans. Graph. 33,* 6 (Nov.), 183:1–183:10.

CAMPEN, M. AND KOBBELT, L. 2014b. Quad Layout Embedding via Aligned Parameterization. *Computer Graphics Forum 33,* 8, 69–81.

DEY, T. K. AND SUN, J. 2006. Defining and Computing Curve-skeletons with Medial Geodesic Function. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP '06. 143–152.

EBKE, H.-C., BOMMES, D., CAMPEN, M., AND KOBBELT, L. 2013. QEx: Robust Quad Mesh Extraction. *ACM Trans. Graph. 32,* 6, 168:1–168:10.

EL-ATTAR, R., VIDYASAGAR, M., AND DUTTA, S. 1979. An Algorithm for l_1-Norm Minimization with Application to Nonlinear l_1-Approximation. *SIAM Journal on Numerical Analysis 16,* 1, 70–86.

FLOATER, M. S. 2003. Mean value coordinates. *Computer aided geometric design 20,* 1, 19–27.

GUROBI OPTIMIZATION. 2013. `http://www.gurobi.com`.

HUANG, J., JIANG, T., SHI, Z., TONG, Y., BAO, H., AND DESBRUN, M. 2014. ℓ1-based Construction of Polycube Maps from Complex Shapes. *ACM Trans. Graph. 33,* 3 (June), 25:1–25:11.

JI, Z., LIU, L., AND WANG, Y. 2010. B-Mesh: A Modeling System for Base Meshes of 3D Articulated Shapes. *Comp. Graph. Forum 29,* 7, 2169–2177.

KHODAKOVSKY, A., LITKE, N., AND SCHRÖDER, P. 2003. Globally Smooth Parameterizations with Low Distortion. *ACM Trans. Graph. 22,* 3 (July), 350–357.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. Graph. 21,* 3 (July), 362–371.

LIVESU, M., GUGGERI, F., AND SCATENI, R. 2012. Reconstructing the Curve-Skeletons of 3D Shapes Using the Visual Hull. *IEEE Transactions on Visualization and Computer Graphics 18,* 11, 1891–1901.

LIVESU, M. AND SCATENI, R. 2013. Extracting Curve-skeletons from Digital Shapes Using Occluding Contours. *The Visual Computer 29,* 9, 907–916.

LIVESU, M., VINING, N., SHEFFER, A., GREGSON, J., AND SCATENI, R. 2013. PolyCut: Monotone Graph-Cuts for PolyCube Base-Complex Construction. *ACM Trans. Graph. 32,* 6, 171:1–171:12.

MADARAS, M. AND ĎURIKOVIČ, R. 2012. Skeleton-based 3D Surface Parameterization Applied on Texture Mapping. *Journal of Applied Mathematics, Statistics and Informatics 8,* 2, 5–19.

MARR, D. 1980. Visual Information Processing: The Structure and Creation of Visual Representations. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences 290,* 1038, 199–218.

PENG, C.-H., BARTON, M., JIANG, C., AND WONKA, P. 2014. Exploring Quadrangulations. *ACM Trans. Graph. 33,* 1, 12:1–12:13.

PENG, C.-H., ZHANG, E., KOBAYASHI, Y., AND WONKA, P. 2011. Connectivity Editing for Quadrilateral Meshes. *ACM Trans. Graph. 30,* 6, 141:1–141:12.

PIETRONI, N., TARINI, M., AND CIGNONI, P. 2010. Almost isometric mesh parameterization through abstract domains. *IEEE Transactions on Visualization and Computer Graphics 16,* 4, 621–635.

PILGWAY. 2009. 3D-Coat. `http://3d-coat.com`.

PIXOLOGIC. 2007. ZBrush. `http://pixologic.com`.

RAY, N., NIVOLIERS, V., LEFEBVRE, S., AND LÉVY, B. 2010. Invisible Seams. In *EUROGRAPHICS Symposium on Rendering Conference Proceedings*, J. Lawrence and M. Stamminger, Eds. Eurographics, Eurographics Association.

RAY, N., VALLET, B., LI, W. C., AND LÉVY, B. 2008. N-symmetry Direction Field Design. *ACM Trans. Graph. 27,* 2, 10:1–10:13.

TAGLIASACCHI, A., ALHASHIM, I., OLSON, M., AND ZHANG, H. 2012. Mean Curvature Skeletons. *Comp. Graph. Forum 31,* 5, 1735–1744.

TAKAYAMA, K., PANOZZO, D., SORKINE-HORNUNG, A., AND SORKINE-HORNUNG, O. 2013. Sketch-based Generation and Editing of Quad Meshes. *ACM Trans. Graph. 32,* 4, 97:1–97:8.

TAKAYAMA, K., PANOZZO, D., AND SORKINE-HORNUNG, O. 2014. Pattern-Based Quadrangulation for N-sided patches. *Comp. Graph. Forum (Proceedings of SGP) 33,* 5, 177–184.

TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. PolyCube-Maps. *ACM Trans. Graph. 23,* 3 (Aug.), 853–860.

TARINI, M., PUPPO, E., PANOZZO, D., PIETRONI, N., AND CIGNONI, P. 2011. Simple quad domains for field aligned mesh parametrization. *ACM Trans. Graph. 30,* 6, 142:1–142:12.

TONG, Y., ALLIEZ, P., COHEN-STEINER, D., AND DESBRUN, M. 2006. Designing Quadrangulations with Discrete Harmonic Forms. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP '06. Eurographics Association, 201–210.

WÄCHTER, A. AND BIEGLER, L. T. 2006. On the Implementation of an Interior-point Filter Line-search Algorithm for Large-scale Nonlinear Programming. *Math. Program. 106,* 1, 25–57.

WU, J. AND LIU, L. 2012. Generating Quad Mesh of 3D Articulated Shape for Sculpting Modeling. *Journal of Advanced Mechanical Design, Systems, and Manufacturing 6*, 354–365.

ZHANG, Y., BAZILEVS, Y., GOSWAMI, S., BAJAJ, C. L., AND HUGHES, T. J. 2007. Patient-specific vascular NURBS modeling for isogeometric analysis of blood flow. *Computer methods in applied mechanics and engineering 196,* 29, 2943–2959.

ZHU, X., JIN, X., AND YOU, L. 2015. Analytical solutions for tree-like structure modelling using subdivision surfaces. *Computer Animation and Virtual Worlds 26,* 1, 29–42.

Received Month YYYY; accepted Month YYYY