Sketch-Based Generation and Editing of Quad Meshes

Kenshi Takayama
ETH Zurich

Daniele Panozzo
ETH Zurich

Alexander Sorkine-Hornung
Disney Research Zurich

Olga Sorkine-Hornung
ETH Zurich

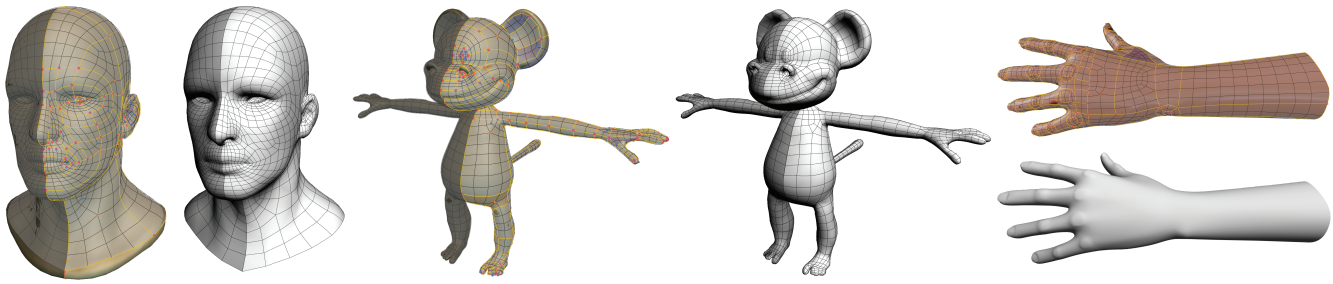

Figure 1: *Results created by two professional artists using our novel sketch-based quad remeshing tool. The smooth subdivision surfaces defined by the coarse quad meshes demonstrate the suitability of our approach for practical production pipelines.*

Abstract

Coarse quad meshes are the preferred representation for animating characters in movies and video games. In these scenarios, artists want explicit control over the edge flows and the singularities of the quad mesh. Despite the significant advances in recent years, existing automatic quad remeshing algorithms are not yet able to achieve the quality of manually created remeshings. We present an interactive system for manual quad remeshing that provides the user with a high degree of control while avoiding the tediousness involved in existing manual tools. With our sketch-based interface the user constructs a quad mesh by defining patches consisting of individual quads. The desired edge flow is intuitively specified by the sketched patch boundaries, and the mesh topology can be adjusted by varying the number of edge subdivisions at patch boundaries. Our system automatically inserts singularities inside patches if necessary, while providing the user with direct control of their topological and geometrical locations. We developed a set of novel user interfaces that assist the user in constructing a curve network representing such patch boundaries. The effectiveness of our system is demonstrated through a user evaluation with professional artists. Our system is also useful for editing automatically generated quad meshes.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling packages;

Keywords: quad meshing, edge flow, sketch-based interfaces

Links: DL PDF WEB VIDEO DATA

1 Introduction

The generation of pure quadrilateral meshes is an important step in the production pipeline of movies and video games, where the Catmull-Clark subdivision is ubiquitously used to generate smooth surfaces. The automatic generation of such meshes is a very active research topic, and geometric modeling packages [3D-Coat 2013; ZBrush 2013] now include automatic quad remeshing algorithms.

The quality of a quad mesh and its suitability to a given application heavily depends on the placement of the singularities (i.e., vertices where more or less than four quadrilaterals meet), and on the alignment of the mesh with semantic features. The latter often does not directly correspond to geometric notions such as principal curvature directions; for example, artists may prefer a certain anisotropy in flat or spherical parts of the model because they anticipate deformations due to articulation of the shape. Important concepts in this respect are the so-called "edge flow" and "edge loops" – chains of consecutive edges in the quad mesh that can locally be thought of as the grid lines. Artists and designers often wish to explicitly control the edge flow and be able to prescribe precise positioning of edge loops.

Optimizing the alignment of the quad mesh and the amount and positions of singularities is a challenging task due to the global effect of every change in the quad mesh connectivity. It is generally impossible to refine, coarsen or otherwise edit a quad mesh only locally without introducing additional singularities. Thus, automatic quad meshing methods [Kälberer et al. 2007; Bommes et al. 2009] cast the problem as a single, mixed-integer, global energy minimization and solve it using customized greedy solvers. Due to the inherent complexity of the problem, it is not feasible to expose to the user the control over every single quadrilateral or edge loop while still producing high-quality meshes with low metric distortion. In other words, while influencing the overall alignment of the mesh to a given field of directions is already achieved by the mentioned automatic methods, hard constraints on edge loops, placement of singularities and other editing operations are not fully supported. This is a major limitation that restricts the practical usability of existing methods to the generation of dense quad meshes, where a fine level of control is not required. Even in this setting, it is common to manually remesh parts of the surface to improve their quality.

Surprisingly, manual generation of quad meshes has not received much attention, neither in the research community nor in the industry. Most of major modeling packages provide tools to manually retopologize surfaces (i.e., convert a triangle mesh into a quad mesh), that

ultimately reduce to manual placement of the majority of the vertices. These tools clearly allow full control over the mesh, but modeling quad meshes with them is slow and requires a lot of redundant user input. Further, it is often quite challenging even for professional artists to manually design a perfect quad mesh on the first try. Since the quality of a quad mesh is a global property, the correction of a single mistake might require regeneration of the entire mesh.

We propose a novel interactive approach to quad remeshing that allows the user to sketch a coarse curve network; every segment of the curve network will become part of an edge of the quad mesh, and every intersection between curves will become a vertex. The interior of every bounded polygon is automatically meshed to match a user-provided number of edges on every side, while singularities are automatically inserted when needed. Such free-form, flexible and at the same time explicitly controlled approach to "sketching" the edge flow is enabled by our particular representation of the quad mesh: instead of focusing on patches with regular connectivity inside, we allow a controlled number of singularities in each patch, and explicitly control the number of subdivisions of each patch side. Singularities can also be moved geometrically and topologically inside each patch. This paradigm provides complete control when desired, while removing most of the tediousness of previous approaches and enabling fast interactive experimentation until the desired connectivity is found. The curve network is stored in a special data structure that supports efficient insertion and removal of curves and links the curve network with the tessellated patches.

Our system is equipped with two novel user interfaces that assist the user in efficiently sketching the curve network: spine sketching and autocompletion. The spine sketching tool allows the user to draw a single stroke and create a new patch whose edge flow is aligned with the stroke. The autocompletion tool analyzes the region around the mouse cursor and suggests a closed region that connects existing parts of the curve network in the vicinity of the cursor. This tool is often used to connect regions generated with the spine sketching tool. The system also provides some specialized tools that allow the user to, for example, quickly sketch strokes on cylindrical parts and to change the topology of the curve network.

We demonstrate that our system allows professional artists to quickly remesh complex 3D shapes into quads while precisely controlling the density and edge flow of the quads (Figs. 1-2). Our approach is also useful for editing quad meshes consisting of patches of regular grids automatically generated using recent quad remeshing algorithms.

2 Related work

Automatic quad mesh generation has been extensively studied [Bommes et al. 2012], due to the broad range of applications in computer graphics and engineering. While existing methods are able to produce high quality dense quad meshes (e.g., [Kälberer et al. 2007; Huang et al. 2008; Bommes et al. 2009; Zhang et al. 2010]), the automatic generation of coarse meshes is still an open challenge, in particular because the number, type, and placement of singularities is of critical importance and very hard to optimize for.

Many existing approaches are based on the generation of a N-rotational symmetry field over a triangle mesh, starting from curvature analysis and manually placed directional constraints or singularities [Palacios and Zhang 2007; Ray et al. 2008; Crane et al. 2010; Lai et al. 2010]. The field is then used to generate a dense quad mesh, that is then simplified [Bommes et al. 2011; Tarini et al. 2011] or used directly to generate a coarse patch layout [Campen et al. 2012]. Daniels II et al. [2011] used a few feature curves to define a scalar field on the surface that imposes a partitioning of the surface, which is then transformed into a quad mesh by applying

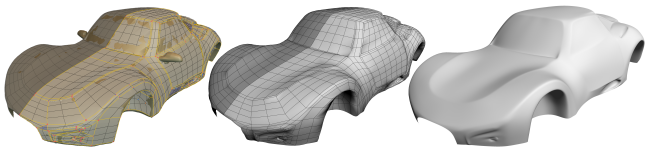

Figure 2: *Retopology of a car model, performed by a professional artist. From left to right: patches created in our sketch-based tool; final coarse quad mesh; Catmull-Clark subdivision surface.*

templates. Tierny et al. [2012] proposed to extract a Reeb atlas from user sketches and use it with connectivity textures to generate a pure quad mesh. Tong et al. [2006] proposed to use a singularity graph to compute a global surface parameterization with discontinuities across singularity edges, from which a pure quad mesh is extracted.

These methods can be used to automatically generate quad meshes while incorporating sparse user constraints. However, the link between the provided constraints and the final quad mesh is not straightforward, and it is thus impossible for the user to control the output of these methods efficiently and at a sufficiently fine scale.

Connectivity editing for quadrilateral meshes was introduced by Peng et al. [2011] and allows the user to move and cancel irregular vertices under specific rules. Their technique is useful for manually fixing topological problems often seen in automatically generated quad meshes. In contrast, our main goal is to enable fast manual creation of quad meshes from scratch using sketch-based user interface.

3D freeform curve sketching techniques have been explored with a main focus on either user interface design [Bae et al. 2009] or algorithms to infer 3D geometry from 2D sketches [Schmidt et al. 2009]. Bessmeltsev et al. [2012] proposed a method to generate quad mesh patches that represent 3D surfaces defined by such 3D curves. In contrast to these scenarios where 3D curves are drawn into an ambient 3D space, we assume a 3D surface mesh as input and focus on how to assist the user in drawing a curve network on the 3D surface that is suitable as a layout of quad mesh patches.

Mesh segmentation is related to our problem of designing a curve network on a 3D surface in the sense that each segmentation boundary can be treated as a curve of the network. While there have been various effective techniques proposed in the literature for both automatic and interactive approaches [Chen et al. 2009; Fan et al. 2012], we believe that retopologizing a model is not equivalent to segmenting a model because artists often create specific edge flows that cannot be captured by existing geometric descriptors. We therefore chose to allow the user to sketch arbitrary curves on the surface to define the curve network.

Suggestive modeling techniques have been explored in different contexts using various approaches such as geometric pattern matching [Igarashi and Hughes 2001], data-driven suggestions [Chaudhuri and Koltun 2010; Tierny et al. 2011], probabilistic reasoning [Chaudhuri et al. 2011], and guidance by physical validity [Umetani et al. 2012]. Our autocompletion technique is conceptually similar to the one proposed by Igarashi and Hughes [2001], but we deal with the problem of suggesting curves on surfaces instead of lines on planes.

3 Overview

Figure 3 shows an overview of our approach. After importing a 3D surface mesh model as input, the user can sketch curves on it freely (Fig. 3a). Curve intersections are detected and marked as corners of the resulting quad mesh patches (Fig. 3b). When a loop of curve segments with four corners is detected, a rectangular patch

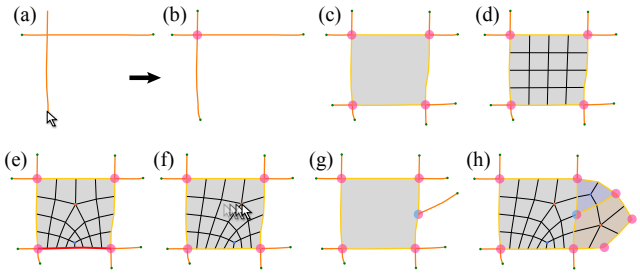

Figure 3: *Brief overview of some of the basic features of our system. (a-d) Sketch-based patch generation, (e-f) topology control inside patches, and (g-h) T-junctions and non-quadliteral patches.*

of the quad mesh is generated automatically (Fig. 3c-d). The user can modify the topology of the quad mesh by either changing the number of edge subdivisions at one side of the patch (Fig. 3e), or moving a pair of irregular vertices by dragging the mouse (Fig. 3f). The curve network is topologically flexible, allowing T-junctions (Fig. 3g) and triangular and pentagonal patches (Fig. 3h, shown in blue and brown, respectively).

The interior of each detected patch is quadrangulated using existing algorithms [Nasri et al. 2009; Takayama et al. 2013]. Specifically, we employ Nasri et al.'s algorithm for pentagonal patches, and Takayama et al.'s generalization of that algorithm for triangular and quadrilateral patches, as it supports much coarser quadrangulations. In the following, we present a set of novel user interfaces for efficiently sketching a curve network representing patch boundaries on the model, as well as the important technical details necessary to implement the system.

4 User interface

The simple combination of patch quadrangulation algorithms [Nasri et al. 2009; Takayama et al. 2013] with basic curve sketching tools (e.g., curve creation, deletion, deformation) would already achieve a faster workflow compared to existing manual tools (Fig. 3). However, it would still be very tedious for the user to draw and adjust every single stroke by hand, especially when working on complex models consisting of many individual parts and semantic features. We thus investigated a number of improvements to the interface, consulting and iterating with an artist. The main result is two novel user interfaces, *spine sketching* and *autocompletion*, and a number of smaller additional tools that are useful in practice as well.

A central design choice for all developed interfaces was to enable an efficient and as intuitive as possible interaction for the user, consistenly based on the same sketching interface and without requiring complex interactions, as in other existing tools. All these features are best demonstrated in the accompanying video.

Spine sketching. With this tool, the user only needs to draw a single stroke, and the system generates a rectangular patch whose center line closely follows the user's stroke (Fig. 4a). According to artists' feedback, this behavior is intuitive because a rectangular patch naturally implies an edge flow represented by its center line. The patch width is set according to a user-controlled brush size. A sketch near existing patch boundaries results in the patches being connected. Two directions are considered for snapping to existing patches: *orthogonal* snapping occurs when the stroke's endpoints are close to an existing patch boundary (Fig. 4b), while *parallel* snapping is used when the stroke is parallel to a nearby patch boundary (Fig. 4c). The tool also checks if corners are on the boundary, in which case the resulting patch will snap to them (Fig. 4d-e). Parallel snapping can stop and resume at some point along the stroke, depending on the configuration of existing surrounding patches; snapping stops either when the snapped patch boundary deviates too much from the stroke in orientation (Fig. 4f), or when the brush does not overlap with the neighboring patch anymore during sketching (Fig. 4g).

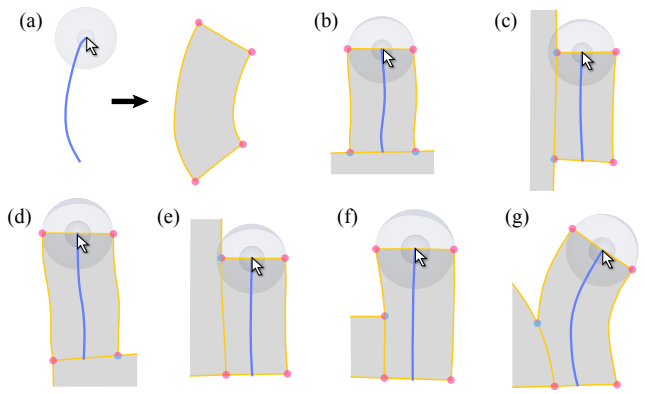

Figure 4: *The spine sketching interface for rapid patch creation.*

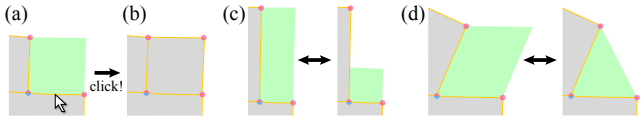

Figure 5: *Examples for the autocompletion feature.*

Autocompletion. This tool provides real-time suggestions of likely patch candidates in the proximity of existing patch boundaries (Fig. 5a). The user can accept the current suggestion by clicking (Fig. 5b). This is useful, for instance, for filling small remaining gaps after spine sketching of more globally relevant patches. The suggested patch snaps to any nearby existing patch boundaries, with priority to snapping to corners. By default the system suggests the largest possible quadrangular patch, but the user can also choose smaller (Fig. 5c) or triangular alternatives (Fig. 5d). Implementation details about this tool are given in Section 5.

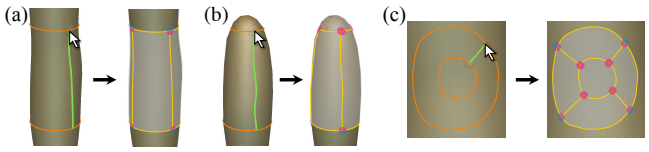

Figure 6: *Sketching for cylindrical features.*

Cylinder sketching. This tool is tailored for sketching on cylindrical parts such as arms and legs. When the user sketches a stroke connecting a pair of loops, typically created using a standard planar cutting tool, the system identifies a few corresponding pairs of points on the loops based on arc length parameterization and connects them by tracing geodesic paths, producing multiple patches simultaneously (Fig. 6a). If the loop encloses a disc-like region, the system also generates a patch filling that region. This is useful for completing the end cap of a cylindrical part, e.g. at fingertips (Fig. 6b). The tool can also be used for connecting a pair of nested loops in a planar region (Fig. 6c).

Alignment to geometric features. It is generally desireable that a quad mesh aligns with salient geometric features such as ridges and

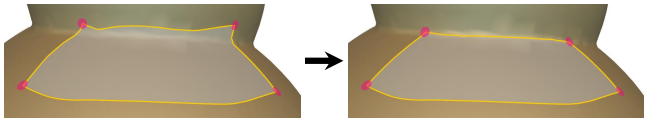

Figure 7: *Automatic alignment of strokes to geometric features.*

valleys of the original model. To facilitate the creation of curves that accurately align with such features, the system provides a simple curve modification tool based on the geometric snakes algorithm [Lee and Lee 2002] that iteratively moves a curve towards more geometrically salient locations (Fig. 7).

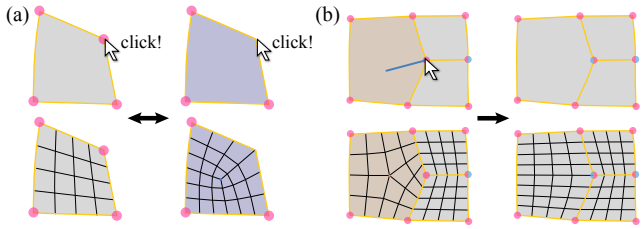

Figure 8: *Efficient editing of corner types.*

Corner type editing. The placement of corners has a fundamental influence on the resulting mesh topology; hence our system provides a simple tool for efficient editing of corner configurations. The user can delete a patch corner with two adjacent curve segments or create a new corner on a curve segment by simple clicking (Fig. 8a). The user can also switch between a T-junction and an ordinary corner by dragging the mouse starting near the respective vertex, and releasing the mouse button at the vertex position (Fig. 8b).

5 Implementation details

In the following we briefly describe how we implemented two key components of our system: the curve network representation and the curve analysis algorithm.

5.1 Curve network representation

Required properties of our curve network representation are:

- topological flexibility, supporting open endpoints and T-junctions;
- curve-by-curve (instead of patch-by-patch) construction;
- instant detection of loops and their orientation;
- handling of elements in different levels (e.g., individual curve vertices and coarse quad mesh vertices) in a unified way.

This is in contrast to previous work on generating surface patches from curve networks [Schaefer et al. 2004; Bessmeltsev et al. 2012], where a complete curve network is assumed as input, with all loops already correctly identified. Li et al. [Li et al. 2005] also proposed a data structure for representing exact embeddings of planar maps on manifold surfaces, which would meet all but the last requirement listed above. Here we describe a simple curve network representation, as shown in Fig. 9, which meets all of these requirements and is well-suited for our purposes.

While the user sketches a curve on the model surface, the curve is densely sampled and the resulting points and segments are added as curve network vertices and halfedges, respectively. Each vertex stores a surface normal in addition to its 3D position. When the

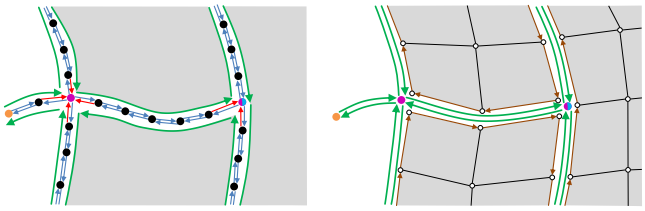

Figure 9: *Curve network representation. (Left) Black points represent curve network vertices, while purple and orange points represent corners and open endpoints, respectively. Red and blue arrows depict curve network halfedges with and without the corner flag, respectively. Green arrows represent halfchains and gray areas represent the corresponding quadrangulated patches. Notice that a T-junction is formed on the right by a non-corner halfedge incident to a corner vertex, visualized with a blue semicircle. (Right) White points represent quad mesh vertices. Brown arrows depict boundary halfedges of patches.*

sketched curve intersects with an existing curve segment, the system computes their intersection and assigns a *corner* flag to halfedges incident to the intersection (red arrows in Fig. 9 left). A corner vertex is defined as the one being pointed to by at least one corner halfedge. A T-junction is represented as a halfedge that is incident on a corner vertex but is not assigned the corner flag. When new halfedges are added, the system traces over halfedges to find consecutive halfedges connecting vertices that are either corners or open endpoints. From these halfedges a new entity called *halfchain* is created that refers to the beginning and ending of the sequence. When consecutive halfchains form a loop with three, four, or five corners, a patch is generated. Each halfchain in the loop is assigned a reference to the generated patch, and the patch is assigned a reference to one of these halfchains (similar to the standard halfedge data structure). A pair of opposing halfchains shares a number of edge subdivisions used for quadrangulating the patch.

Geometric information such as 3D position and normal is obtained for vertices at patch boundaries by simple sampling along their corresponding curve network segment. For vertices inside a patch, geometric information is first interpolated from the boundaries using uniform Laplacian smoothing, followed by projection to the model surface along the interpolated normal. Each patch is a separate quad mesh with its own set of vertices, faces, and halfedges (Fig. 9 right). To establish adjacency information between neighboring patches, each halfedge on the boundary of each patch stores a reference to the halfchain it belongs to and an index representing its position on the halfchain. This allows us, for example, to easily trace a global edge flow across neighboring patches.

Loop orientation detection. When a loop of halfchains is detected, the system generates a patch only if the loop is oriented counterclockwise when viewed from the exterior of the surface (otherwise, the system would generate two overlapping patches from a single loop). To detect a loop's orientation, we compute

$$A = \sum_i \frac{1}{2} \left((\mathbf{p}_i - \mathbf{c}) \times (\mathbf{p}_{i+1} - \mathbf{c}) \right) \cdot \mathbf{n}_i \qquad (1)$$

where $\mathbf{p}_i$ and $\mathbf{n}_i$ denote position and the normal of the i-th vertex on the loop, respectively, and $\mathbf{c}$ denotes the loop's center of mass. We regard A as a rough estimate of the signed area of a region on the surface enclosed by the loop, and the loop orientation is detected as counterclockwise if $A > \lambda l^2/4\pi$ where l is the loop's length (we use a threshold $\lambda = 0.1$).

5.2 Curve analysis algorithm

Here we briefly describe how curves representing existing patch boundaries are analyzed when using spine sketching and autocompletion. Our basic idea is to locally parameterize the model surface near the user's mouse input by employing the stroke parameterization technique of Schmidt [2013] (Fig. 10), and then analyze the configuration of nearby curves in this 2D parameter space, making subsequent processes much simpler.

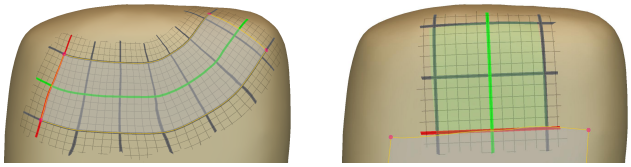

Figure 10: *Local parameterization used in spine sketching (left) and autocompletion (right).*

For spine sketching, the analysis is done by following the description in Section 4. For autocompletion, the analysis is based on the number of corners and their respective angles formed by the boundary curves. Fig. 11 enumerates all the cases considered. We initially set the region to be analyzed as the locally parameterized region and run the analysis algorithm. To produce suggestions with smaller sizes, we repeatedly shrink the region to be analyzed and run the algorithm while keeping the local parameterization.

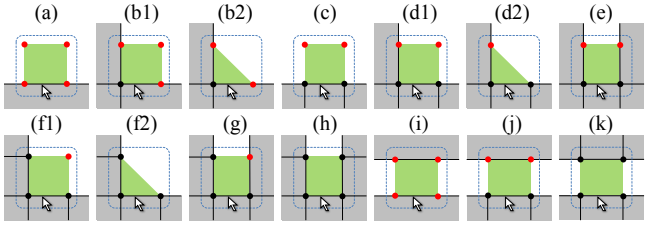

Figure 11: *Configurations of the curves and corners considered in our autocompletion algorithm. Gray regions represent existing patches. The blue dotted line represents the region to be analyzed. Black points represent existing corners, while red points represent corners created by the algorithm. The green region represents the suggested patch. The algorithm considers cases where there are zero (a), one (b1-b2), two (c-e), three (f1-g), and four (h) corners on a sequence of boundary curves. The algorithm also considers cases where there are two separate sequences of the boundary curves (i-k).*

6 Results

Evaluation. We hired two professional 3D artists to test our prototype system. Both had approximately six years of experience in retopology tasks, in particular in the generation of low-poly characters for interactive games. We presented the tool and explained all the features in a twenty-minute training session and then asked them to retopologize a model from the Stanford repository and some additional models of their choice.

Figures 1, 2 and 12 (first two columns) show results created by the artists (see Table 1 for the statistics). The artists generated high quality quad meshes using only a few patches.

The feedback from the artists was very positive, confirming that our new tool is significantly more efficient than currently available tools [3D-Coat 2013; ZBrush 2013] for manual retopology tasks. The most appreciated feature was the possibility to edit the topology by changing the number of edge subdivisions at patch boundaries, followed by the option to topologically move irregular vertices within patches. The spine sketching tool was mostly used in the beginning of the process, to generate large patches and experiment freely with different edge flows. However, in the latter stages when the curve network was partially created, it did not provide sufficient precision; the autocompletion tool and manual drawing were used more frequently, especially to close small gaps or to connect existing patches. The cylinder sketching and corner type editing tools were also used frequently.

Overall, both artists agreed that the new interface, combined with the patch generation algorithm, is superior to all the existing tools they know and use. They asked to further adjust the interfaces and hotkeys to make them more usable with pen input devices.

model	user	# tri	# quad	#△	# □	#⬠	time
head	artist I	6406	808	10	62	1	1.2
monkey	artist II	39996	1104	17	132	1	1.5
car	artist II	100312	1170	27	140	0	1.3
bunny	artist II	70374	1075	13	57	0	0.9
hand	artist I	34390	1548	27	195	1	2.0
horse	author	50000	2398	8	137	4	0.8
armadillo	artist I	358150	2439	15	154	5	0.9
spider	author	194996	6287	21	208	2	2.5

Table 1: *Statistics of the created results. Legends: # tri: number of triangles in the input 3D model; # quad: number of faces in the resulting quad mesh (half the actual size in the case of symmetric models); number of patches constituting the quad mesh: triangular (#△), rectangular (#□), pentagonal (#⬠); time: modeling time in hours.*

Additional results. Figure 12 (two rightmost columns) show a few additional results created by one of the authors who had no prior experience in retopology tasks. These results demonstrate that our tool is also accessible for nonprofessionals.

Editing of automatically generated quad meshes. Recently, a lot of attention has been devoted to methods for automatic generation of quad meshes consisting of a few regular-grid patches defined by separatrices connecting irregular vertices [Bommes et al. 2011; Tarini et al. 2011; Campen et al. 2012]. Figure 13 demonstrates that our tool can also be used to quickly edit such automatically generated quad meshes.

7 Limitations and future work

While we focused on letting the user freely draw arbitrary curves on surfaces to define curve networks, we expect that incorporating mesh segmentation techniques [Chen et al. 2009; Fan et al. 2012] and feature line extraction techniques (e.g., [Hildebrandt et al. 2005]) could further reduce the necessary user input; these techniques could be used to automatically generate sketches that can be converted into patches using the proposed system. The autocompletion tool would also become more powerful by employing, in addition to the geometric pattern matching, data-driven techniques [Chaudhuri and Koltun 2010] and probabilistic reasoning [Chaudhuri et al. 2011].

Takayama et al.'s robust patch quadrangulation algorithm [Takayama et al. 2013] only supports rectangular and triangular patches. For pentagonal patches we use Nasri et al.'s algorithm [2009], but it fails in some extreme cases (e.g., one side having many edge subdivisions

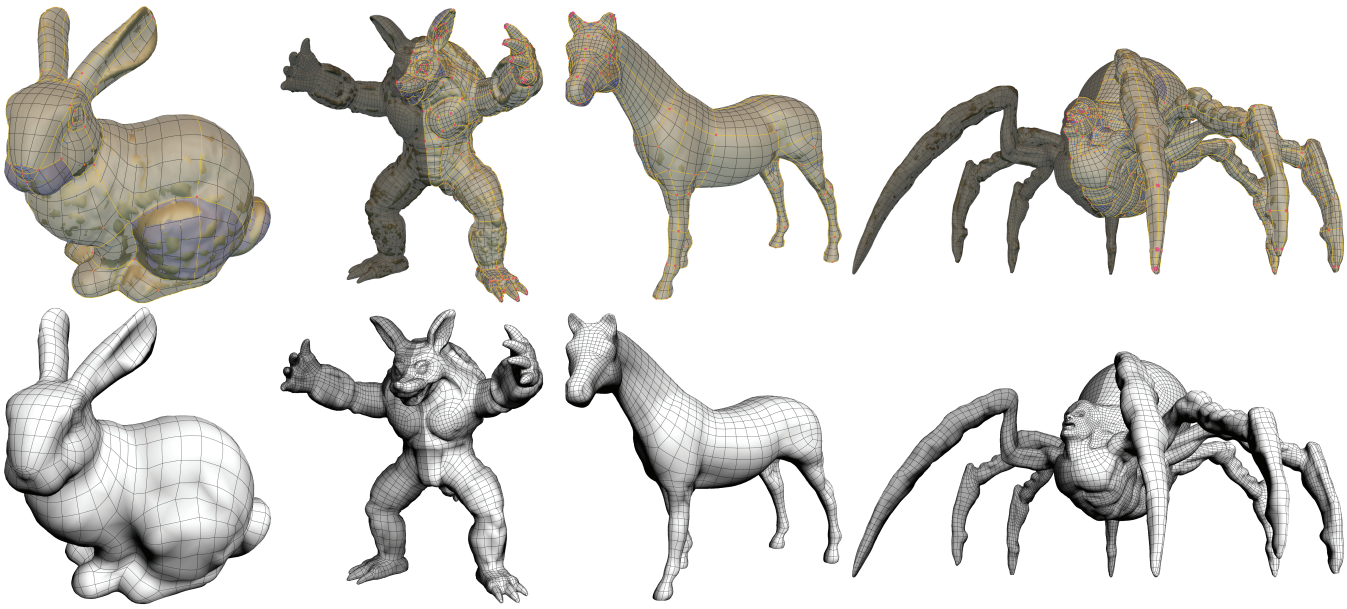

Figure 12: *Retopology results created by the two professional artists (Bunny, Armadillo) and one of the authors (Horse, Spider).*

while the other sides having only one edge). Extending Takayama et al.'s algorithm to five and more sided regions is thus desirable, but it is not straightforward since the number of cases to be considered in their enumerative approach would grow quickly.

Acknowledgements

We are deeply grateful to Maurizio Nitti for the illuminating discussions and concept sketches that were tremendously helpful for the development of our system. We also thank Maurizio Nitti and Alessia Marra for their participation in the user evaluation and feedback. We thank Jun Saito and his colleagues for a useful discussion. We are grateful to Felix Hornung for helping us with the technical illustrations and to Emily Whiting for narrating the video. The Bunny and Armadillo models are courtesy of the Stanford 3D Scanning Repository. The Horse and Bimba models are courtesy of the AIM@SHAPE Shape Repository. The Head and Hand models are from ZBrush©Pixologic Inc. and TurboSquid, respectively. Other models are kindly provided by Maurizio Nitti. This work was supported in part by the ERC grant iModel (StG-2012-306877), by an SNF award 200021_137879 and by a gift from Adobe Research. Kenshi Takayama's stay at ETH Zurich is funded by JSPS Postdoctoral Fellowships for Research Abroad.

References

3D-COAT, 2013. Pilgway. Version V3, `http://3d-coat.com/`.

BAE, S.-H., BALAKRISHNAN, R., AND SINGH, K. 2009. EverybodyLovesSketch: 3D sketching for a broader audience. In *Proc. UIST*, 59–68.

BESSMELTSEV, M., WANG, C., SHEFFER, A., AND SINGH, K. 2012. Design-driven quadrangulation of closed 3D curves. *ACM Trans. Graph. 31*, 6, 178.

BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3, 77.

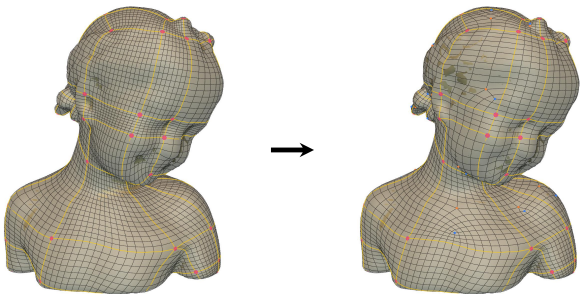

Figure 13: *Editing an automatically generated quad mesh consisting of patches of regular grids [Tarini et al. 2011] (left) by changing the number of edge subdivisions at patch boundaries and moving irregular vertices (right).*

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. 2012. Quad meshing. In *Eurographics 2012 State of the Art Reports*, 159–182.

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.

CHAUDHURI, S., AND KOLTUN, V. 2010. Data-driven suggestions for creativity support in 3D modeling. *ACM Trans. Graph. 29*, 6, 183.

CHAUDHURI, S., KALOGERAKIS, E., GUIBAS, L. J., AND KOLTUN, V. 2011. Probabilistic reasoning for assembly-based 3D modeling. *ACM Trans. Graph. 30*, 4, 35.

CHEN, X., GOLOVINSKIY, A., AND FUNKHOUSER, T. A. 2009. A benchmark for 3D mesh segmentation. *ACM Trans. Graph. 28*, 3.

CRANE, K., DESBRUN, M., AND SCHRÖDER, P. 2010. Trivial connections on discrete surfaces. *Comput. Graph. Forum 29*, 5, 1525–1533.

DANIELS II, J., LIZIER, M. A. S., SIQUEIRA, M. F., SILVA, C. T., AND NONATO, L. G. 2011. Template-based quadrilateral meshing. *Computers & Graphics 35*, 3, 471–482.

FAN, L., MENG, M., AND LIU, L. 2012. Sketch-based mesh cutting: A comparative study. *Graphical Models 74*, 6, 292–301.

HILDEBRANDT, K., POLTHIER, K., AND WARDETZKY, M. 2005. Smooth feature lines on surface meshes. In *Proc. SGP*.

HUANG, J., ZHANG, M., MA, J., LIU, X., KOBBELT, L., AND BAO, H. 2008. Spectral quadrangulation with orientation and alignment control. *ACM Trans. Graph. 27*, 5, 147.

IGARASHI, T., AND HUGHES, J. F. 2001. A suggestive interface for 3D drawing. In *Proc. UIST*, 173–181.

KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2007. QuadCover: Surface parameterization using branched coverings. *Comput. Graph. Forum 26*, 3, 375–384.

LAI, Y., JIN, M., XIE, X., HE, Y., PALACIOS, J., ZHANG, E., HU, S., AND GU, X. 2010. Metric-driven RoSy field design and remeshing. *IEEE TVCG 16*, 1, 95–108.

LEE, Y., AND LEE, S. 2002. Geometric snakes for triangular meshes. *Comput. Graph. Forum 21*, 3, 229–238.

LI, W.-C., LEVY, B., AND PAUL, J.-C. 2005. Mesh editing with an embedded network of curves. In *Proc. SMI*, 62–71.

NASRI, A., SABIN, M., AND YASSEEN, Z. 2009. Filling N-sided regions by quad meshes for subdivision surfaces. *Comput. Graph. Forum 28*, 6, 1644–1658.

PALACIOS, J., AND ZHANG, E. 2007. Rotational symmetry field design on surfaces. *ACM Trans. Graph. 26*, 3, 55.

PENG, C.-H., ZHANG, E., KOBAYASHI, Y., AND WONKA, P. 2011. Connectivity editing for quadrilateral meshes. *ACM Trans. Graph. 30*, 141:1–141:12.

RAY, N., VALLET, B., LI, W., AND LÉVY, B. 2008. N-symmetry direction field design. *ACM Trans. Graph. 27*, 2.

SCHAEFER, S., WARREN, J., AND ZORIN, D. 2004. Lofting curve networks using subdivision surfaces. In *Proc. SGP*, 103–114.

SCHMIDT, R., KHAN, A., SINGH, K., AND KURTENBACH, G. 2009. Analytic drawing of 3D scaffolds. *ACM Trans. Graph. 28*, 5.

SCHMIDT, R. 2013. Stroke parameterization. *Comput. Graph. Forum 32*, 2.

TAKAYAMA, K., PANOZZO, D., SORKINE-HORNUNG, A., AND SORKINE-HORNUNG, O. 2013. Robust and controllable quadrangulation of triangular and rectangular regions. Tech. rep., ETH Zurich.

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

TIERNY, J., DANIELS, II, J., NONATO, L. G., PASCUCCI, V., AND SILVA, C. T. 2011. Inspired quadrangulation. *Computer Aided Design 43*, 11, 1516–1526.

TIERNY, J., DANIELS II, J., NONATO, L. G., PASCUCCI, V., AND SILVA, C. T. 2012. Interactive quadrangulation with Reeb atlases and connectivity textures. *IEEE TVCG 18*, 10, 1650–1663.

TONG, Y., ALLIEZ, P., COHEN-STEINER, D., AND DESBRUN, M. 2006. Designing quadrangulations with discrete harmonic forms. In *Proc. SGP*, 201–210.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph. 31*, 4, 86.

ZBRUSH, 2013. Pixologic, Inc. Version 4.4, `http://www.pixologic.com/zbrush/`.

ZHANG, M., HUANG, J., LIU, X., AND BAO, H. 2010. A wave-based anisotropic quadrangulation method. *ACM Trans. Graph. 29*, 118:1–118:8.