Detail-Replicating Shape Stretching

Ibraheem Alhashim · Hao Zhang · Ligang Liu

Abstract Mesh deformation has become a powerful tool for creating shape variations. Existing deformation techniques work on preserving surface details under bending and twisting operations. Stretching different parts of a shape is also a useful operation for generating shape variations. However, under stretching, texture-like geometric details should not be preserved but rather replicated. We propose a simple and efficient method that helps create model variations by applying non-uniform stretching on 3D models with organic geometric details. The method replicates the geometric details and synthesizes extensions by adopting texture synthesis techniques on surface details. We work on analyzing and separating the stretching of surface details from the stretching of the base mesh resulting in the appearance of preserved details. The efficiency of our method is attributed to a local parametrization of the surface with the help of curve skeletons. We show a variety of experimental results that demonstrate the usefulness of this intuitive stretching tool in creating shape variations.

Keywords Detail-replication · Stretching · Geometry synthesis · Mesh editing

1 Introduction

Surface details are essential in depicting realistic 3D models. The simplest form of surface detail representation is textures. Extending the representation to parallax mapping adds an extra level of realism and is often used in real-time rendering for its efficiency in simulating surface details. However, these texture representations lack the capability to represent more complex organic surface details such as thorns, scales or realistic bark that require geometric primitives.

Editing meshes in the presence of complex surface details is a challenging task. State-of-the-art mesh deformation methods work on preserving surface details under simple rotations and translation of some deformation controllers [8]. Non-uniform scaling or stretching operations cause extreme distortion to these details limiting the amount of deformation possible for modelers when creating variation on existing models (e.g. elongating a snake or resizing a patterned vase). Applying current shape manipulating scaling on organic models and meshes obtained with laser scanners often results in loss of surface properties and other visual artifacts (see Fig. 1). Another difficulty in dealing with organic models arises when the details are composed of several accumulated patterns that are challenging to identify or separate (see Fig. 2). The process of applying stretching to detailed meshes is then a time consuming process that involves the modeler introducing geometry to the model and synthesizing similar details. This is especially the case when dealing with shapes containing organic patterns. Being able to efficiently produce shape variations from existing shapes by stretching could allow a modeler to populate a scene more quickly.

In this article, we present an interactive and efficient algorithm for applying 1D stretching of detailed mesh parts with minimal user interaction and natural looking results. We encode the surface details of the input as a 2D texture, therefore reducing the problem space to synthesis on a plane. We also constrain the synthesis process such that the results blend well with the original mesh while minimizing distortions to the

I. Alhashim, H. Zhang
Simon Fraser University, Canada E-mail: iaa7@sfu.ca, haoz@cs.sfu.ca

L. Liu
Zhejiang University, China E-mail: liangliu@zju.edu.cn
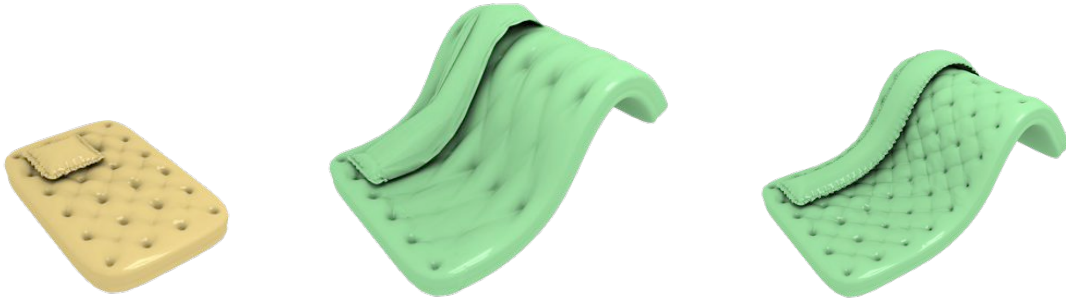
Fig. 1 An example of a stretching operation: (left) input pillow and mattress shapes, (middle) applying a stretching operation by interpolation results in large distortions to the surface details, (right) stretching with detail-replication using our method preserves the natural look of the surface details.
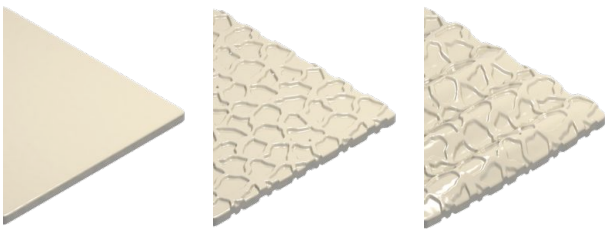
Fig. 2 An example of an accumulated surface texture. This poses a challenge when determining the base shape.

original surface details. The generated details on the extended area matches the frequency, scale, and topology of the source. An overview of our framework can be seen in Fig. 3. We also show that separating surface details into different levels can help when dealing with complex or inhomogeneous details. The efficiency of our method allows the user to interactively test and modify different synthesis parameters on large and highly detailed meshes.

We achieve this by first computing a *base mesh* that serves as an approximation of the general shape of the input. A curvilinear *grid* is then constructed on the desired stretch region from the *base mesh* with the help of a curve skeleton. The region is then approximated by a 2D texture computed by projecting this region onto the grid. A stretching operation is specified by the user with a simple 3D curve. The stretching factor is estimated and the representing 2D texture is synthesized accordingly. A corresponding new geometry is then reconstructed and stitched with the source with as minimum distortion as possible.

2 Related Work

2.1 Detail preservation by replication in 2D

In the image domain, the work by **(author?)** [14] proposed re-synthesizing texture patches from the source image to preserve details around modified feature curves. This allows for free form deformations of a user defined feature curve on the image while preserving the frequency of the details. By replicating different source patches in accordance to the feature curve the appearance of stretched patterns caused by moving and/or bending the curve is reduced. Another work [5] introduced the PatchMatch method which allows the user to apply stretching and widening operations on an image while preserving the details in a similar way to [14]. The method produces high quality outputs due to randomly finding patches that preserve image coherence.

The method in [35] can resize 2D images containing symmetric patterns by separating the process into two phases. First, it detects regions with translational symmetry and segment the image into symmetric and non-symmetric regions. Next, it resizes the different regions using different techniques and then merges the results. When resizing the image, symmetric regions are resized using a summarization algorithm that removes or replicates cells from the extracted lattices in symmetric regions. The non-symmetric regions are resized using optimized warping and the two results are seamlessly merged, using graph cuts, to produce the final image. This method only works on symmetric patterns that can be identified using their symmetry detection algorithm. Organic patterns, however, tend not to exhibit regular patterns that can be easily detected.

The reliance on existing background details and the ability to blend pixel colors makes such methods non-trivial in the context of 3D shape deformations. In our 3D replication, the details are constrained by the boundaries of the mesh's part, therefore, the idea of a background is analogous to a single colored background in an image. Perhaps more challenging to represent in a flat domain are inner details containing geometry of non-zero genus.
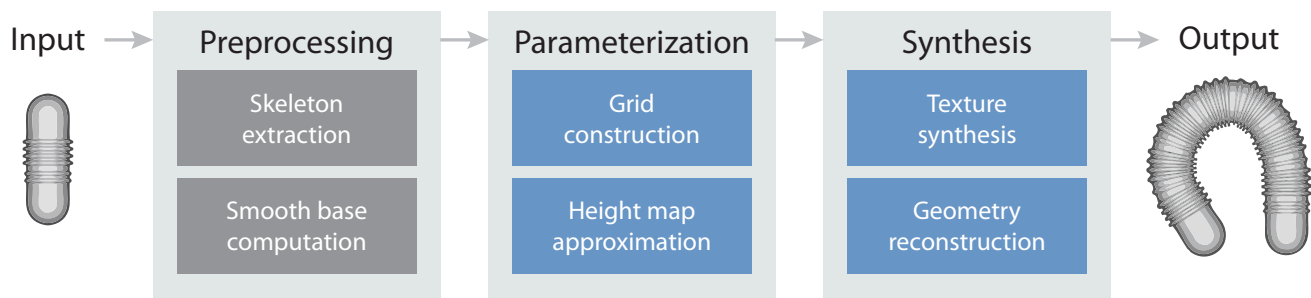
Fig. 3 Pipeline of our detail-replicating stretching algorithm.

2.2 Procedural modeling

Designing objects using procedural modeling or CAD software allows us to incorporate surface details easily by adding rules that correspond to the geometric details as shown by [28, 27]. An inverse procedural modeling framework in [7] takes a piece of exemplar 3D geometry and extracts shape generation rules. The method relies on finding symmetries by aligning salient feature lines on the input. This approach works well on shapes with highly symmetrical and distinct features. However, when considering organic shapes this approach might not be suitable since we may deal with small or continuously varying surface details and patterns. Furthermore, the extracted rules do not allow the freedom to merge different pieces having largely different orientations. Hence, procedural modeling is not general enough for describing organic shapes or applying free form edits. Our proposed method tries to deal with high frequency geometric details which is still a hard problem in the context of inverse procedural modeling.

2.3 Shape deformation

State-of-the-art 3D deformation methods focus on protecting large scale features of the shape. A survey by **(author?)** [8] discussed different linear deformation methods that preserve details under bending and twisting deformations. In [32] the method applies shape deformation operations on an intrinsic surface representation that encodes each vertex by its relative neighborhood based on the Laplacian of the mesh. A user can define a region of interest for editing and then manipulate the shape. The surface of the area affected by that editing operation is then reconstructed in such a way that the original details of the shape are preserved as much as possible. However, such methods would ignore any form of stretching or resizing of parts but they can incorporate such operations by allowing interpolation similar to the techniques used in [19, 12]. Unfortunately, with in-

terpolation, details are distorted regardless of the technique being used.

Another approach extracts a descriptive set of wires and their relations allowing for intuitive resizing while preserving major shape characteristics [17]. The wires allow for more intuitive editing operations analogous to real life armatures used in sculpture. The method is suited for man-made shapes and works well on shapes with smooth surfaces but interpolates complex surface details during a stretch.

A non-homogeneous resizing method presented in [20] protects some model features, particularly the distinct ones, during resizing based on a vulnerability map. The vulnerability computation is based on a per-face metric that combines slippage (a measure of surface persistence under a transformation [18]) with normal curvature. The model is then embedded into a protective volumetric grid and grid-based space-deformations are applied during resizing. The contribution of each cell in the grid to the scaling transformation is then computed and the final transformations are carried back to the model by interpolation. The method allows shape-aware scaling of the entire model while partial scaling operations are also possible by adding hard constrains to a group of surrounding cells. This method is analogous to image re-targeting methods [34] that work on preserving salient features rather than replicating them. Again, surface details will either be distorted by interpolation or can be preserved in their original form on the expense of stretching other regions in the model. Such restriction is not well suited when stretching organic shapes.

2.4 Anisotropic resizing

The method by **(author?)** [9] performs anisotropic resizing on meshes with surface details. They extend the grid idea from [20] to incorporate geometric textures by separating them from the underlying surfaces and reproducing them on the scaled surfaces using texture

synthesis. Their method extracts the texture after a segmentation process which would not be suitable in shapes with non-homogeneous surface details as different patterns belong to different segments. Furthermore, the output of their geometric texture is sensitive to the mesh density, thus limiting the reproduction of the details when models have low triangle count.

Our method is similar in that we replicate the surface details to compensate for the extended area introduced by the stretching or scaling operation. Advantages of our approach over [9] can be seen in: dealing with low density triangular models as well as large scale meshes, working with non-homogeneous surface details, dealing with multi-layered surface details (i.e. large and small scale details of the surface), and the intuitive user interface that allows for free form stretching operations. Our experimental results show that our method is significantly faster and it is able to deal with more types of details.

2.5 Cut-and-paste

Cut-and-paste methods are used to combine different parts of different models to generate new shapes. The work in [6] described a number of algorithms based on multi-resolution subdivision surfaces that achieve cut-and-paste edits at interactive rates. One limitation of their method is that it only works with regions that are homeomorphic to a disk. In [16] a proposed topology-free cut-and-paste editing method deals with regions of non-zero genus while minimizing any possible distortions caused by incompatible geometry between source and target. The described surface parametrization is used to transfer the details onto the target surface by identifying a shared simple planar base surface. [31] introduced another cut-and-paste tool that allows the user to drag one mesh part onto another with some overlap and the system snaps them together using their proposed Soft-ICP algorithm. Also, recent work done by [30] explored shape reuse and composition in 3D mesh modeling by combining ideas similar to cut-and-paste along with other techniques.

Our proposed method is related to cut-and-paste in the sense that we automatically cut patches of the surface from the same model and paste them coherently in order to replicate the surface detail. The pasted elements cover the introduced surface area during the stretching of the mesh's part. An advantage of our approach over cut-and-paste is that the stretched part is merged more naturally with the source mesh.

3 Detail-replicating stretching

Inspired by deformation techniques in the image domain [14], we apply the concept of detail replication when performing a stretching operation. We replicate the surface details of the region on the detailed mesh we intend to stretch. We adopt 2D texture synthesis to help guide the geometry synthesis process.

First, we represent the surface details as a height-field like image that approximates the actual details. This is done by computing a base mesh and generating a surface approximating grid with the help of curve skeletons. The second step is to synthesize more of the same pattern along the direction of the skeleton edges in the selected region. The amount of stretching is specified by a user drawn 3D curve. The synthesized image is then used to construct geometrical surface patches along the stretching direction. The final step is to combine all the patches into one closed surface that starts and ends with the exact geometry of the source region. In the following we describe each stage of our method in more detail.

3.1 Base mesh

We start by computing a base mesh $\mathcal{B}$ that defines the underlying geometry of the detailed input mesh $\mathcal{D}$. The typical approach is to apply a fairing process (mesh smoothing) that gradually smooths the high frequencies of the surface texture. As suggested by [1], applying few steps of mean curvature flow results in a good vertex-to-vertex correspondence between $\mathcal{D}$ and $\mathcal{B}$ and minimizes vertex sliding that typically occurs when fairing using the umbrella operator.

Curvature flow smooths the mesh by iteratively moving vertices along the surface normal $\boldsymbol{n}$ with the speed of the mean curvature $\bar{\kappa}$ [13]. We start the fairing process by computing the non-zero coefficients of the matrix K representing the matrix of the curvature normals. We compute the entries of K with the following discrete expression for the curvature normal at each vertex x_i:

$$-\bar{\kappa}\boldsymbol{n} = \frac{1}{4A} \sum_{j \epsilon N(i)} (\cot \alpha_j + \cot \beta_j)(x_j - x_i), \qquad (1)$$

where α_j and β_j are as described in Fig. 4b, and A is the sum of the areas of the adjacent triangles of x_i as in 4a.

We then iteratively solve the following linear system representing the implicit integration of the diffu-
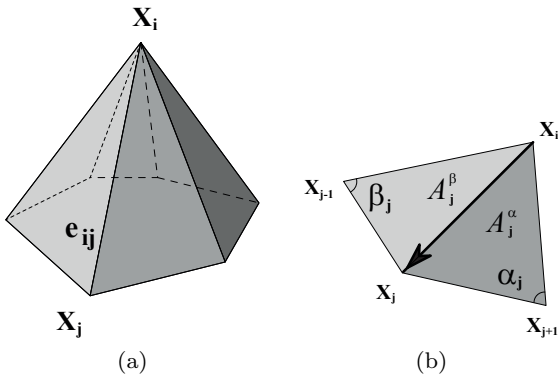
Fig. 4 A vertex x_i and its adjacent faces (a), and one term of its curvature normal formula (b)

sion equation using the backward Euler method as described in [13]:

$$(I - \lambda dt K)X^{n+1} = X^n, \qquad (2)$$

where λdt is the smoothing time step and as shown in [13] increasing the value results in smoother meshes. This linear system is sparse, thus, we are able to solve it efficiently using a preconditioned conjugate gradient (PCG) solver. For preconditioning we use the typical diagonal preconditioner $\tilde{A}$ where $\tilde{A}_{ii} = 1/A_{ii}$.

This process of smoothing generally introduces shrinkage of $\mathcal{B}$. One possible way to compensate for this shrinkage is to scale $\mathcal{B}$ back to its original volume. The correction factor given by [13] is $\beta = (V^0/V^n)^{\frac{1}{3}}$, where V^0 is the volume of $\mathcal{D}$ and V^n is the volume of $\mathcal{B}$ at iteration n. Let x_k^1, x_k^2, x_k^3 be the three vertices of the k-th triangle of the mesh, then the volume computation for a discrete mesh is given by the following expression [23]:

$$V = \frac{1}{6} \sum_{k=1}^{nbFaces} g_k \cdot N_k, \qquad (3)$$

where $g_k = \frac{x_k^1 + x_k^2 + x_k^3}{3}$ and $N_k = \overrightarrow{x_k^1 x_k^2} \wedge \overrightarrow{x_k^1 x_k^3}$. Each vertex of $\mathcal{B}$ is then multiplied by the computed volume correction factor β. Depending on the size of the surface detail relative to the entire shape, it might not be ideal to apply the volume correction step as it may introduce large vertex sliding. An adaptive volume correction method would be more suitable, however, to the best of our knowledge no such method exists.

The amount of smoothing is specified by the time step λdt. The larger the parameter value the smoother the mesh gets. The optimal value depends on the surface details of the input. To automatically assign such a

value would require a method that can accurately separate the pattern from the underlying surface. In general, the problem of detecting pattern symmetries, regularity, and repetitive structure is a fundamental problem in computer graphics. Several attempts have been made that work on extracting structure from repeated patterns in 2D images [25, 26, 10]. These attempts, however, assume regular structure in the provided input which is not always the case.

In our framework we relay on a user specified value of λdt. The efficiency of the fairing method and our parametrization technique allows for an interactive process in which the user can experiment with different λdt values and check the results immediately even with large complex models.

3.2 Curve skeletons for shape abstraction

Curve-skeletons are 1D structures that approximate the topology and shape of 3D objects. These shape descriptors have many applications such as shape registration, retrieval, and deformation. By representing a complex shape in these abstract curves we are able to reduce the complexity in such applications. A survey by **(author?)** [11] classifies different methods and analyzes the advantages and limitations of each class.

In our implementation we utilize extracted curve skeletons using the method in [3]. In the extraction process we specify parameters such that the resulting skeleton is smooth and nicely embedded inside the shape. We use these skeletons when constructing our approximate surface parametrization. The skeleton also simplifies the process of selecting mesh regions in which the user intends to apply stretching using the produced segmentation.

3.3 Surface Parameterization

A parameterization is a bijective mapping from one surface onto another provided they have the same topology. If the topology differs between the two surfaces, cuts or other topology changing operations are required on the source mesh. In our context, we are interested in parameterizations that can handle any mesh regardless of the topology. With this relaxed condition we are able to construct an approximating parameterization mesh that is invariant to the topology of the input.

We will refer to our parameterization domain as the cylindrical *grid* as it is constructed by sweeping along the curve skeleton and sampling the cross-section of $\mathcal{B}$. This cylindrical grid $\mathcal{G}$ is used in computing the approx-

imate geometric texture as an image and its geometry reflects the geometry of the base $\mathcal{B}$.

3.4 Cylindrical grid

Given a segment on the curve skeleton selected by the user, we construct a cylinder like surface by sweeping. The quality of the extracted curve-skeleton affects the construction of $\mathcal{G}$ and we may need to refine the edges on that segment. We do so by subdividing its edges and then performing Laplacian smoothing of the entire segment. The refined segment vertices are then used as control points for a Bézier spline C to ensure a desirable continuity. We then parametrize C with $t \in [1, 0]$ and uniformly sample it based on the specified grid resolution h.

The ideal grid resolution is dependent on the surface texture. More detailed textures require a larger h to ensure a more faithful representation in the resulting texture image. In practice we set h in the range of 70 to 140 pixels, Fig. 6 shows the result of different resolution values. Given the grid resolution h, we compute the size used for the grid cells y with the following expression:

$$y = \frac{2\pi r}{h},\qquad(4)$$

where r is the closest distance from the base mesh $\mathcal{B}$ to the curve C computed by projecting a sample of vertices onto the skeleton segment. We can now compute the grid width:

$$w = \frac{L_C}{y},\qquad(5)$$

where L_C is the arc length of the spline C . The values h and w would respectively represent the height and width of the approximating texture image.

Next, we sample w cross-sections by intersecting a plane p_i with the base mesh $\mathcal{B}$. The plane p_i passes through C at $t_i = \frac{i}{w}$ with a normal $\boldsymbol{n}_i$ equal to the tangent of C at t_i. The resulting cross-section S_i forms a closed polygon. In the case of missing geometry or multiple curves we simply select the largest connected part and then close that polygon.

An arbitrary vector $\boldsymbol{a}_0$ on the plane p_0 is computed and transported along C. We compute at each t_i the closest point on the polygon S_i to the curve C along the direction of $\boldsymbol{a}_i$ and denote this point as the *start* of S_i. These start vectors can be rotated around the point t_i on the curve if the original pattern exhibits

(a)   (b)

(c)
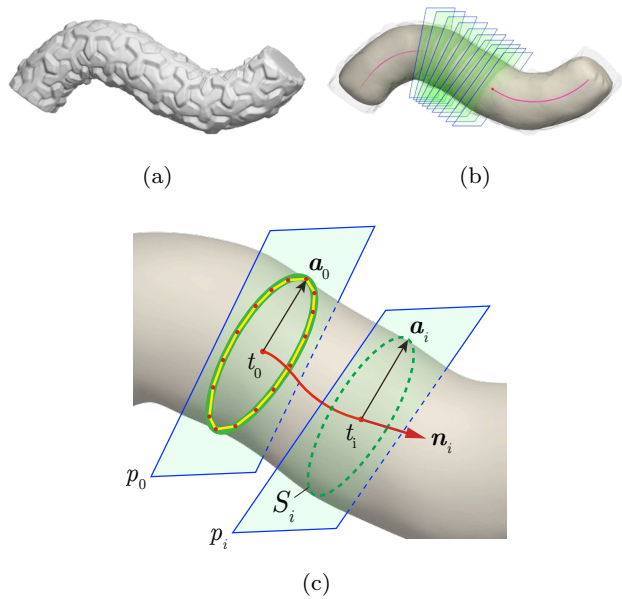
Fig. 5 Grid construction by sweeping: (a) the source mesh. (b) the computed base mesh with the medial curve representing the provided skeleton. We sample along the curve using w planes shown in green. (c) detailed view of the base mesh sampling process during grid construction. The green curves on each plane represent the cross-section computed from the base. The yellow polygon is an equidistant polygon sampled from the closest intersecting point to $\boldsymbol{a}_i$ and it contains w edges.

some twisting. We then resample S_i by walking along the polygon in h segments of length:

$$\Delta e = \frac{L_{S_i}}{h}.\qquad(6)$$

Lastly we orient S_i such that its normal is in the same direction as $\boldsymbol{n}_i$. The re-sampled S_i can now be encoded as a vector of scalar values representing the distances between $C(t_i)$ and the samples. This vector Y_i helps to indicate the shape of the underlying structure at t_i. Fig. 5c shows an overview of this grid construction via our sweeping process.

The geometric construction of the grid is done by connecting the corresponding sample points on S_i with the points on S_{i+1}. The height value from equation (5) results in grid cells with an aspect ratio of a square-like cell minimizing distortions. The resulting shape of the grid represents an abstraction of $\mathcal{B}$ with a resolution dependent on h. Fig. 7 shows an example of our grid construction.
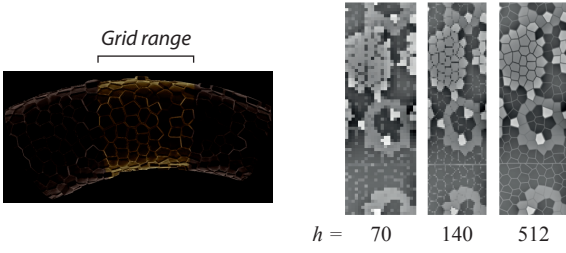
Fig. 6 Grid resolution: the grid defined on the blue region of the input shape (left) and the resulting texture image at different resolutions (right).

Source mesh   Generated grid

Fig. 7 Cylindrical grid: a mesh with geometric surface details (left), the grid constructed from the base shown in red and blue (right)

3.5 Geometric texture representation

We use the constructed grid $\mathcal{G}$ to generate a 2D image that approximates the surface details of $\mathcal{D}$ and to parametrize their geometry by projecting the vertices to the nearest cell in $\mathcal{G}$.

Computing the texture image I is a straightforward operation. Let $\boldsymbol{n}^c$ be the normal of the grid cell c_{xy} and $\boldsymbol{p}_c$ is the centroid, where $x \in [1, w]$ and $y \in [1, h]$ then the pixel value d

$$d = \|\boldsymbol{q} - \boldsymbol{p}_c\|, \tag{7}$$

at the pixel I_{xy} is:

$$I_{xy} = \begin{cases} d & \text{if } \boldsymbol{p}_c \text{ is below } \mathcal{D} \\ -d & \text{otherwise} \end{cases}, \tag{8}$$

where $\boldsymbol{q}$ is the intersection point on $\mathcal{D}$ computed by shooting a ray with origin $\boldsymbol{p}_c$ and direction $\boldsymbol{n}^c$. The sign of d is determined by which side of the plane defined by the intersecting face on $\mathcal{D}$ does $\boldsymbol{p}_c$ falls into. We perform these intersection tests efficiently using a space partitioning octree [33].

We then parametrize the geometry of $\mathcal{D}$ onto the domain of $\mathcal{G}$ using the following steps: for each vertex $\boldsymbol{v}$ of the selected region on $\mathcal{D}$, we search for the closest grid cell using ray-triangle intersections. The ray $\boldsymbol{r}$ has the origin $\boldsymbol{v}$ and normalized direction $-\boldsymbol{d}$, where $\boldsymbol{d}$ is the normal of the corresponding vertex on the base $\mathcal{B}$. For each hit, we encode the location of $\boldsymbol{v}$ with two quantities: a scalar *height* value; and a *shift* quantity with

respect to a local frame on the intersected grid cell. The main idea of this process is to project the source onto the grid in a flattening manner.

In order to compute the height value, we first compute the weights of the mean value coordinate (MVC) [15] of the intersection point inside the cell. Let α_i be the angle at the intersection point v_0 in the virtual triangle $[v_0, v_i, v_{i+1}]$ (see Fig. 8) where $v_{i>0}$ are the corners of the cell, the mean value coordinates are computed using the weights:

$$\lambda_i = \frac{w_i}{\sum_{j=1}^{k} w_j}, \qquad w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|v_i - v_0\|}$$

and in our cells k is always equal to 4 corners. It is now possible to reconstruct the intersection point v_0 using:

$$v_0 = \sum_{j=1}^{4} \lambda_i v_i.$$

When the intersection point falls on the boundary of the cell we only need to linearly interpolate on that edge. The height value q at v_0 is then defined as:

$$q = \boldsymbol{n}^c \cdot (\boldsymbol{v} - v_0).$$

The shift factor depends on the local frame of the grid cell and it is defined by the two scalar quantities m and θ using the following equations:

$$p' = v_0 + (q\boldsymbol{n}^c),$$

$$\boldsymbol{s} = \boldsymbol{v} - p', \qquad \boldsymbol{k} = v_4 - v_1,$$

$$m = \|\boldsymbol{s}\|,$$

$$\theta = \arccos(\frac{\boldsymbol{s} \cdot \boldsymbol{k}}{\|\boldsymbol{s}\| \|\boldsymbol{k}\|}). \tag{9}$$

The shift factor helps compensate for any vertex sliding that occurs during the smoothing operation and allows us to encode points that do not constitute simple displacements from $\mathcal{B}$.

With these three quantities we can reconstruct the vertex $\boldsymbol{v}$ given a new cell by the following expression:

$$\boldsymbol{v} = v_0' + q\boldsymbol{n}^c + m\mathbf{R}, \tag{10}$$

where v_0' is the intersection point computed in the new cell and $\mathbf{R}$ is the rotated vector $\boldsymbol{k}'$ along the axis $\boldsymbol{n}^{c'}$ with angle θ using a the shortest quaternion rotation. The entire vertex displacement encoding is detailed in Fig. 8.
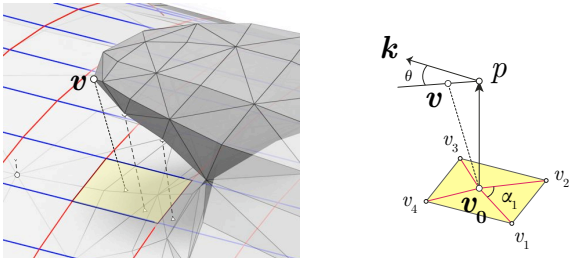
Fig. 8 Mesh projection onto the grid cells.

3.6 Synthesizing stretched parts

The synthesis process starts after the user specifies the stretching operation by simply drawing a curve E' that extends the original length of the selected segment of $\mathcal{D}$. The most practical approach of texture synthesis in our context is the one that produces the largest connected patches. We experimented with various 2D texture synthesis methods [21, 22, 29] and the most seamless results were produced by the one that uses the entire input as the patch.

3.6.1 Texture synthesis of details

Given the input texture I with dimensions w and h defined by our grid, we are interested in synthesizing I' such that its width is equal to:

$$w' = w + e, \qquad e = \frac{L(E') - L(C)}{y},$$

where the function $L(x)$ is the arc length of the curve x, and y is the segment length as in equation (4). We perform the synthesis by copying the entire patch I overlapped by b pixels n times where $n = \lceil \frac{w'}{w} \rceil$. The amount of overlap b is highly dependent on the source texture. Complex patterns may require larger overlap to allow for more room to circumvent some distinct features of the pattern.

In the overlap region we compute the minimum error boundary cut by computing a weighted L^2-norm of differences between the two overlapping patches A and B at each pixel p:

$$D_p(A, B) = \|A_p - B_p\|_2.$$

We then find the minimum cut on a graph M in which each pair of adjacent pixels s,t in the overlap have an arc with the following cost:

$$M_{s,t}(A, B) = D_s(A, B) + D_t(A, B). \tag{11}$$

We then connect the top row with a start node and the bottom row with an end node with arcs of zero cost and find the shortest path c given by Dijkstra's algorithm. The computed path c defines a binary mask that we use to copy the new patch B to I'. In order to preserve the initial surface details in I we start the synthesis with a simple copy of I as the first step. To add some variety on the output we can slightly randomize the band size b or the source of subsequent patches. The output of the entire synthesis process is the matrix T that represents the corresponding pixel indices of I' with respect to the source I. Fig. 9 shows an overview of the tiling process. Using this method, the complexity of our texture synthesis algorithm is bounded by the complexity of solving Dijkstra's algorithm for each patch. A drawback of this approach is the apparent repetition when synthesizing highly random patterns.
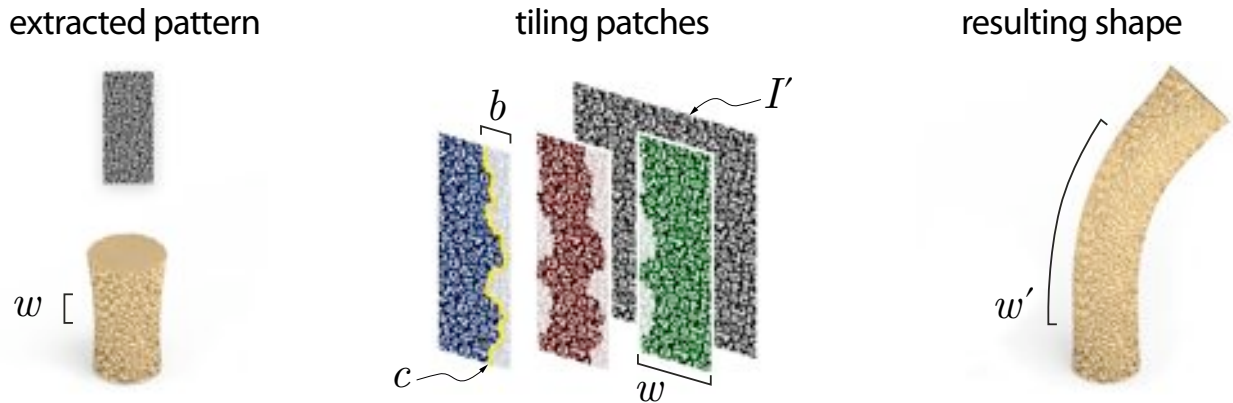
3.6.2 Geometry reconstruction

We use the indices matrix T to construct the actual geometry of the extended area. First, we combine the two curves C and E' into one curve E. To extend the grid $\mathcal{G}$ to the new width of E we start by splitting $\mathcal{G}$ in half and copy the exact structure of its first half to the target grid $\mathcal{G}'$. We then go over the middle section of E and generate interpolated S_i cross-sections until we reach $i = w' - \frac{w}{2}$ where we switch back to simply copying from the end half of $\mathcal{G}$. The idea behind this split is to preserve as much original geometry as possible to help minimize any distortions resulting from our approximations.
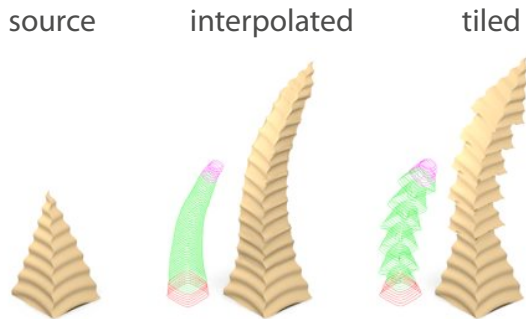
Interpolating the cross-sections of the middle region might not always be the best option. If the input pattern corresponds to both local and global changes on the surface then a better approach is to correspond these cross-sections with the patches given by T. Fig. 10 shows an example where cross-section interpolation is preferable.

We now have the extended grid $\mathcal{G}'$ with the new width specified by the user's stretching operation. The next step is to reconstruct the geometry of each patch in T by finding complete faces $f \in F$ belonging to that patch. A face f is complete if all of its vertices are located inside the patch. We do this by constructing a hash table of all the indices of the vertices collected from the patch's cells and then simply test against the set of faces in the selected region of $\mathcal{D}$. We can then reconstruct the geometry of the vertices in the set F using equation (10). The result of this process is a set of separated patches of geometry representing the synthesized image I'.

extracted pattern   tiling patches   resulting shape

Fig. 9 Texture synthesis by tiling: the pattern is extracted as a grayscale image (left), we synthesize an extension I' by tiling the patches with an overlap b and compute the best cut c (middle), after the geometry reconstruction process we obtain the extended shape (right).

source   interpolated   tiled

Fig. 10 Extending cross-sections by (middle) interpolation or by (right) copying the source.

3.7 Patch stitching

The final step in our framework is the stitching of the synthesized patches. We formulate the problem as that of hole-filling [2, 24]. Between the patches A and B there exists a small empty region that separates the geometry of the two. Our objective is to fill this region such that B blends well with A resulting in the least amount of visual artifacts. Aesthetically pleasing stitches are the ones that triangulate the area while avoiding triangles that stand out by having long edges or sharp curvature.

The general approach to hole-filling is to compute the minimum area triangulation of the hole, refine the new triangles to match the surrounding ones, and finally apply a surface fairing operation [24]. We apply a similar three steps method where we: treat the boundary of the two consecutive patches; zip the two boundaries $\boldsymbol{B}_A$ and $\boldsymbol{B}_B$ by simply advancing to the best adjacent vertices; then apply a local fairing operation on bad triangles.

Our objective in the boundary treatment step is to produce a more flat and smooth boundary. For every adjacent edges e_1 and e_2 we add a new boundary trian-

gle between them if their angle $\phi < \frac{5\pi}{9}$ and the dihedral angle between their respective two boundary faces is $\omega < \frac{\pi}{4}$. These angle conditions can be seen in Fig. 11. The first condition ϕ ensures a flatter boundary and the second condition ω helps preserve sharp edges of the original shape. We iteratively apply this refinement stage until no new faces are added to the boundary. The resulting edges on the final boundary are more well adjusted for the second zipping stage.

We start the zipping stage by first aligning the boundaries $\boldsymbol{B}_A$ and $\boldsymbol{B}_B$ and assigning the start vertices on each boundary to the closet pair v_i^A and v_j^B. We traverse the two boundaries in a greedy manner and construct connecting faces along the way [4]. Fig. 12 shows an example result of our boundary stitching approach. Local quality measures can be further applied [2] since our stitching process does not guarantee having regions free of self-intersecting triangles.

3.8 Adding variation

A common feature of organic patterns is randomness in both structure and frequency. In such patterns, our synthesized parts may exhibit an artificial look of repetition, for instance, tree trunks do not have the exact cross-section as they grow. Our method allows the user to produce some random appearance on the resulting cross-sections. These fluctuations on the surface can follow a pattern themselves (e.g. a wave) or can be randomly generated from gradient noise as shown in Fig. 13. A nice feature of this randomness effect is that high frequency details are not equally affected as the base surface due to the separation in our synthesis process.
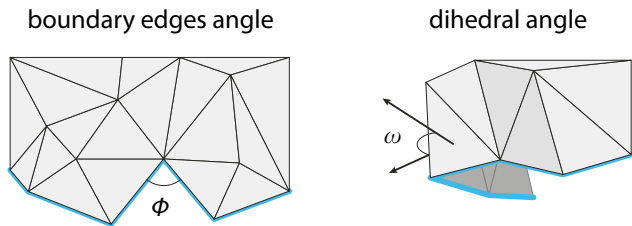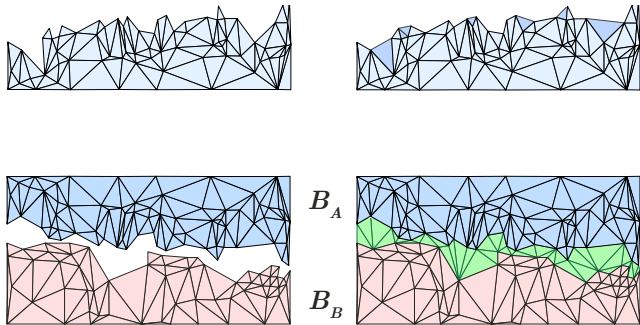
boundary edges angle   dihedral angle

Fig. 11 The boundary treatment tests applied.

Fig. 12 Boundary treatment step (top) and an example of our patch stitching method (bottom).

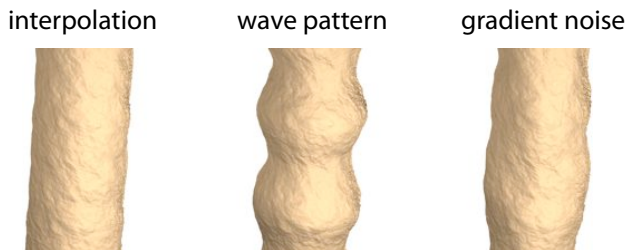interpolation   wave pattern   gradient noise

Fig. 13 Variation on the cross-sections

4 Implementation Details

Our detail-replicating stretching tool is implemented in C++ on a 3.0 GHz quad core PC equipped with 4GB memory. The graphical user interface is implemented in Windows using the Qt framework. Graphics functionality is implemented in OpenGL and most stretching operations are done at interactive rates.
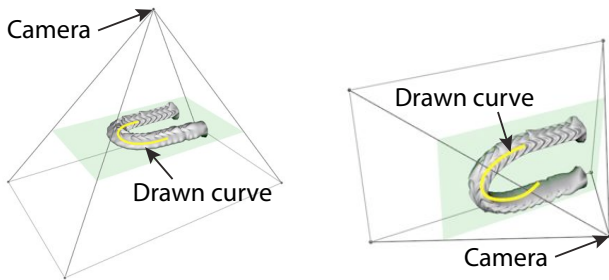
In an editing session, the user starts by loading an arbitrary triangular mesh. Our method supports editing of meshes containing boundaries and can be of non-zero genus. We assume the availability of a previously extracted curve skeleton that contains both the skeleton graph and a basic segmentation of the mesh. The user then simply selects a point on the surface and the corresponding node on the skeleton is set as the start. Next the user clicks on a different part of the the surface that specifies the end. We assign to each edge on the skeleton a weight of one and run Dijkstra's algorithm to obtain the shortest path from the start to the

end. We then collect all the branches and loops between the two selected nodes in this path. This step is necessary for surface details that are complex enough to produce branching on the skeleton. The result of this user interaction is a set of faces and their major medial curve.

There are three parameters that control the texture extraction and synthesis: the smoothing step size, grid resolution, and the overlap size. All three parameters are highly dependent on the input pattern. The smoothing parameter λdt specifies the amount of smoothing needed to obtain a base mesh. In most of our examples we set $\lambda dt = 0.001$ by default and for objects with more complex surface details we increase it to around 0.01. The second parameter is the grid resolution h which we set to 70 pixels by default. For more regular structured patterns a lower grid resolution produces good results with low computational cost. The last parameter is the overlap b that is used by the texture synthesis process. The best overlap size is one that covers the largest feature of the pattern. We set the default value as 20% of the width of the input image.

The next step is drawing the curve defining the stretching operation. We allow the user to draw a curve on the screen that specifies the stretching direction and path during the synthesis. We project the cursor's position in screen space onto the focal plane defined by the camera. For each different stretching operation the user needs to rotate the camera such that the viewing direction is almost orthogonal to their desired direction of stretching (Fig. 14). After the curve is drawn, we embed this curve in 3D in a way that ensures better continuity with the original shape. Let v_a be a unit vector representing the starting tip of the user drawn curve and let v_b be the tangent vector on the curve skeleton at the middle of the selected input region (see Fig. 15b). We compute the rotation that aligns v_a with v_b and apply it to the drawn curve. We then translate the drawn curve so that its starting point at a is on the mid-point b of the selected part of the curve skeleton (see Fig. 15).

The final step is tuning the parameters previously mentioned to achieve aesthetically pleasing stretching. The efficiency of our method allows for an interactive editing experience. The synthesis process is fast and usually takes two seconds to finish on medium sized objects (about 50K vertices). Perhaps the most challenging parameter to automate is the overlap width which is still an open problem in 2D texture synthesis as it relates to texture and pattern recognition. To better understand how the process works in practice, the reader is invited to watch the video accompanying this paper.

Mesh	Points count	Stretch Factor	Time (s)
Capsule	3360	9.2	0.640
Stanford Dragon	1845	19.9	1.738
Armadillo	1549	3.2	0.219
XYZ Dragon	3023	7.7	1.032

Table 1 Timing statistics for our stretching examples in Fig. 16. The number of points are the number of vertices considered in the selected mesh region to be stretched.

Fig. 14 Applying two different stretching operations by sketching different curves (yellow) from different viewing directions

We realize that the nature of our synthesis process allows for a large number of opportunities of parallel optimizations that can lead to near real-time editing. Such parallel opportunities are during grid construction, the tile-based synthesis, and the patch stitching. We exploited some of these opportunities of parallelism using OpenMP, for example when smoothing, and we have witnessed a considerable performance increase.

5 Results and Examples

We present examples of stretching operations on various shapes using our method. Fig. 16 shows a number of different meshes with stretched parts. The timing for each of these stretching operations are listed in Table 1. These times, however, do not include the preprocessing stage required once per loaded mesh. Depending on the size of the input mesh, skeleton extraction would requires a couple of seconds to several minutes using the mesh contraction method [3]. Base computation via implicit fairing typically takes a couple of seconds for our largest mesh (about 170K vertices).

Fig. 17 shows more examples using our method. In Fig. 17b the mattress and the pillow were stretched separately using the same user drawn stretching curve. Fig. 17c shows a mesh containing multiple parts with different surface details and two variations resulting from different stretching operations. Fig. 18 is a good example showing the usefulness of our stretching tool as opposed to any interpolating method. The details on both the dragon's torso and the base are replicated thus appearing to be preserved while the model's coverage is expanded.

Fig. 19 shows two examples where our method fails to produce natural looking results. Several factors affect the outcome of a stretching operation. Mesh quality can affect the stitching process greatly since we relay on the existing triangulation of the source shape. Another factor is the topological complexity of surface details. For details with genus higher than zero, the grid con-

structed is not sufficient to encode them resulting in visible distortions.

6 Limitations

Currently we can only handle well surface details of low topological complexity. Surface details of non-zero genus are not encoded accurately resulting in distorted reconstructions. A possible solution to this problem is to apply a preprocessing process that might simplify such regions.

Another limitation of our implementation is the ability to stretch along a single direction. If we consider a box with varying surface details on all faces we are only able to replicate details along one of its three main axis. For a successive expansion of the box on a different axis, we need to compute a new grid along that axis. One solution is to construct a box grid that truthfully represents the underlaying shape of the input. The reliance on curve skeletons also limits our ability to deal with shapes for which it is not easy to compute such skeletons for (e.g. thin sheets).

As shown in Fig. 19, not all surface patterns we experimented with replicated without distortion. Due to our surface approximation, we might fail to accurately encode the original surface geometry resulting in visible artifacts when reconstructing the stretched surface.

7 Conclusion and Future Work

We have presented an interactive and intuitive mesh editing tool that preserves organic like surface details during 1D stretching. Instead of interpolating details during an extension operation, we replicate parts of the surface by synthesizing and stitching appropriate patches from the original mesh. Separating the synthesis of large scale details from small ones allows us to apply such global operations on the shape with low distortions and minimum artifacts along the seams. We demonstrated our efficient technique with a variety of synthesized examples that are generated in a matter of seconds.

(a)

(b)
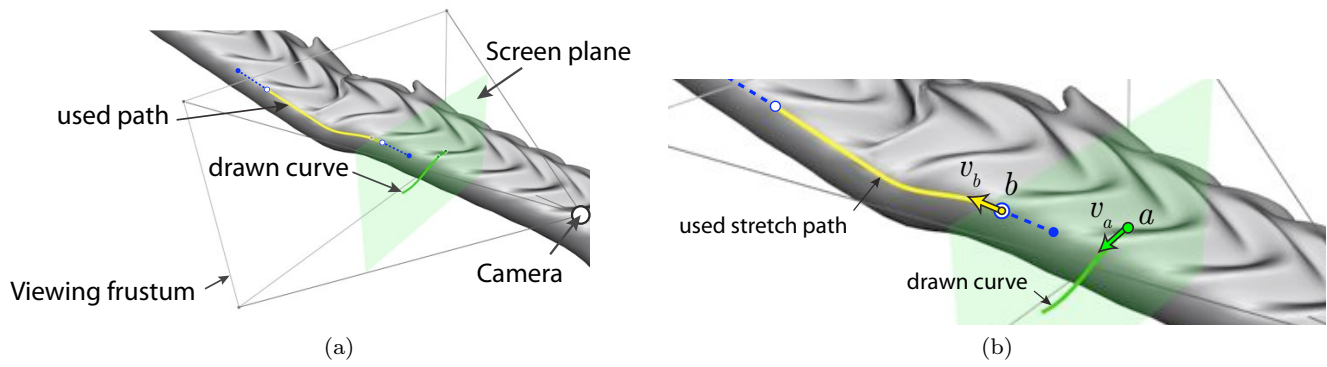
Fig. 15 Details of the embedding of user drawn stretching curves. In order to ensure continuity, user drawn curves are transformed to align with the existing curve skeleton of the source shape. This is done by transforming the start of the drawn curve (green curve) at a to match the curve skeleton at the middle, point b, of the selected input region (dashed blue line).


(a)

(b)

(c)

(d)
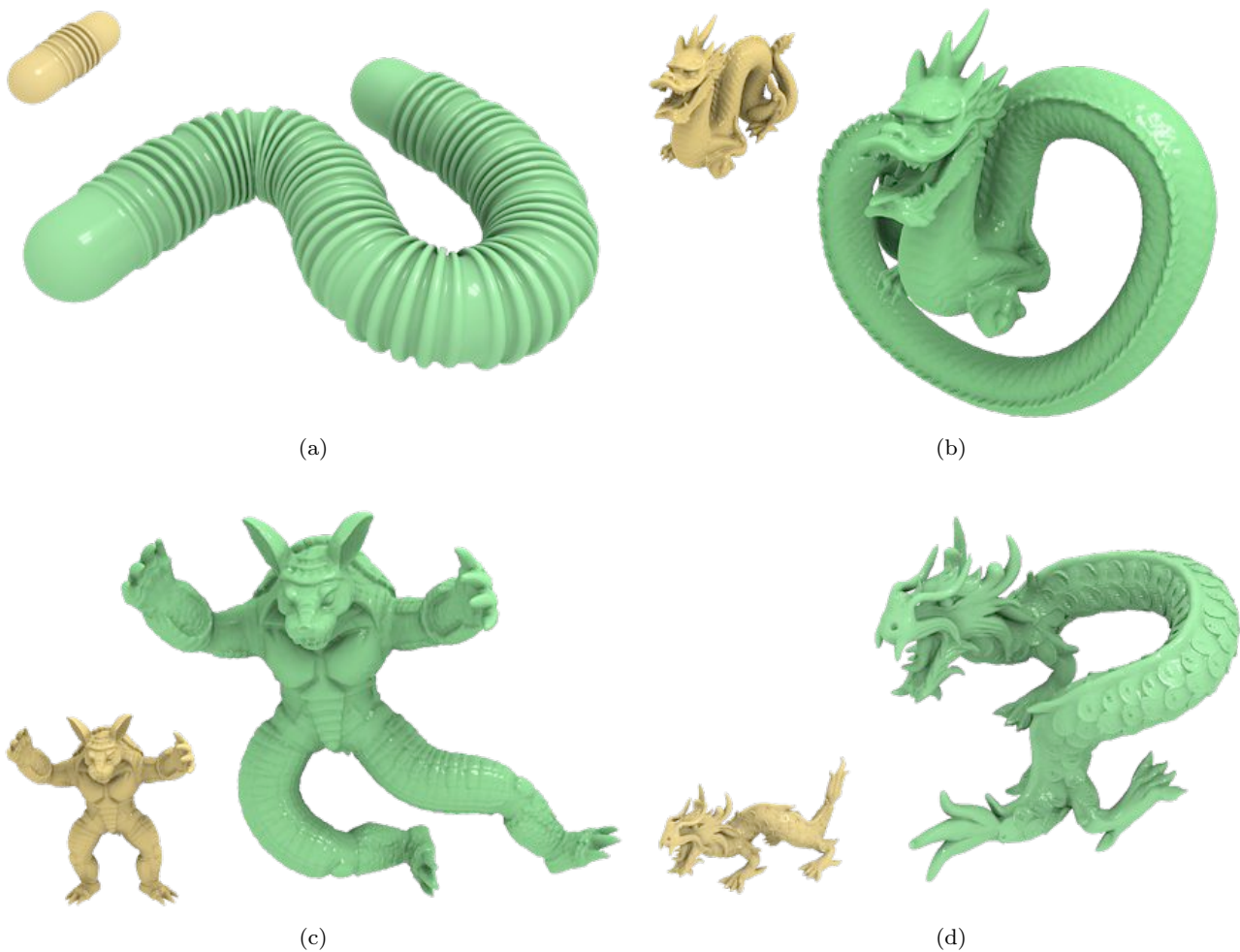
Fig. 16 Results of stretching operations on various meshes. Input meshes are shown in yellow, and in green is the output of a user defined stretching operation using our method.

(a)

(b)

(c)
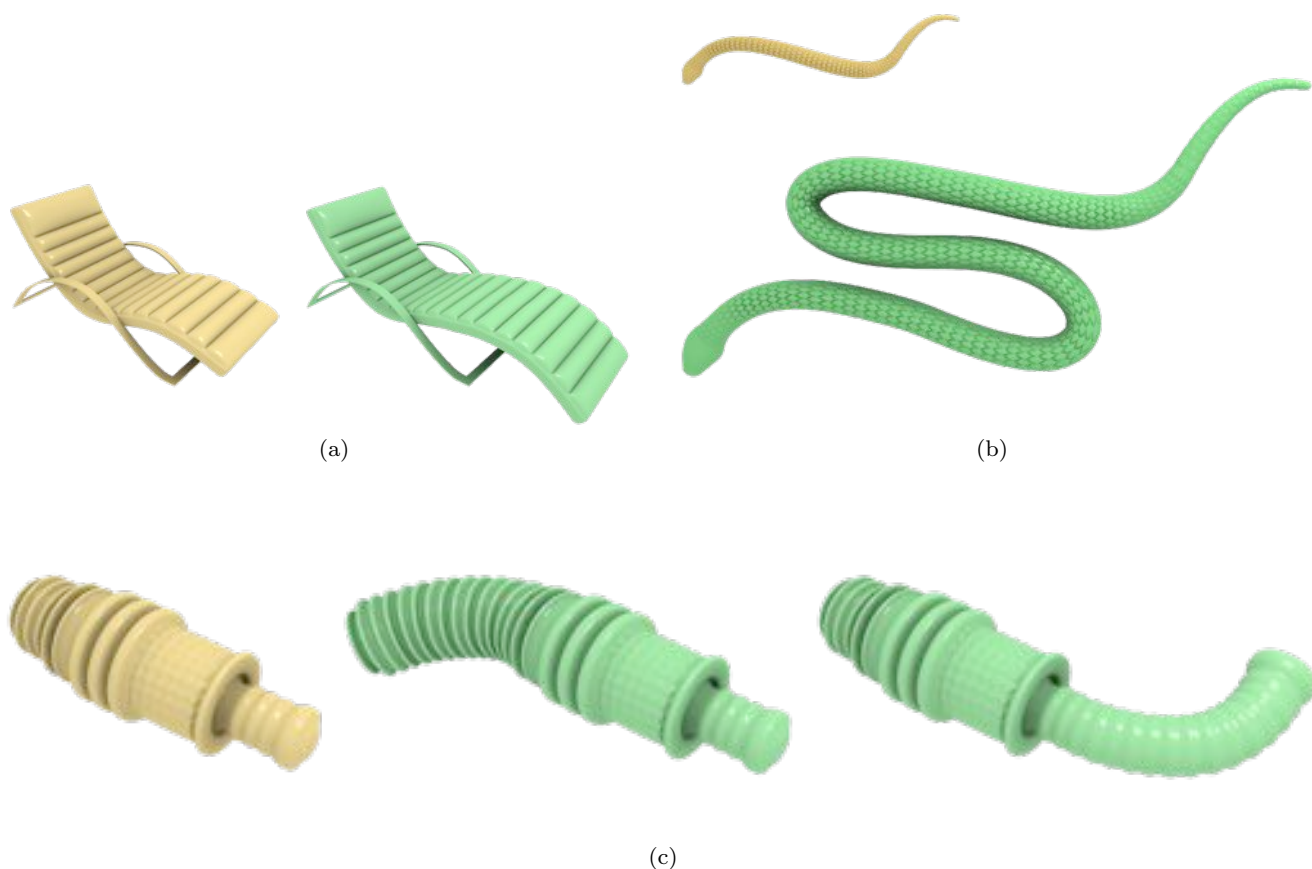
Fig. 17 More stretching results. (a) the seat is stretched independently from the base. (b) a snake model stretched using our method. The scales are made of dense geometry. (c) a mesh with several possible parts to stretch.
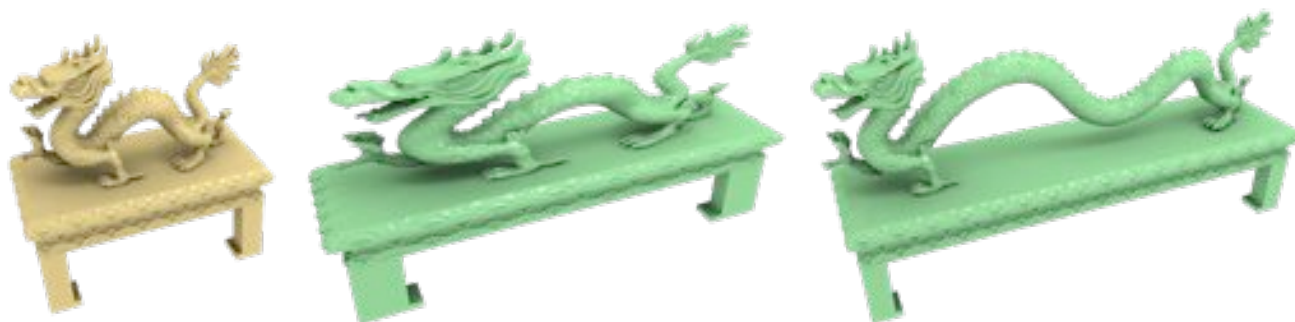

Fig. 18 Preserving surface details via replication: (left) the source model. (middle) details are distorted when stretching by a free-form deformation using a 2x2 lattice. (right) using our method, the surface details appear to be preserved. Note that the dragon and the base were stretched separately.

There are several avenues of future work to explore. A more general grid needs to be constructed to accommodate stretching of arbitrary surfaces. Such a grid would also enable stretching operations to be performed in 2D. One idea is to employ a polycube parameterization or any quad re-meshing of a base mesh when constructing such a flexible grid. Also, problems in texture synthesis and arbitrary boundary stitching still apply to our method and it can greatly benefit from advances in these areas.

References

1. V. Andersen, M. Desbrun, J. A. Baerentzen, and H. Aanaes. Height and tilt geometric texture. In *ISVC '09: Proceedings of the 5th Inter-*
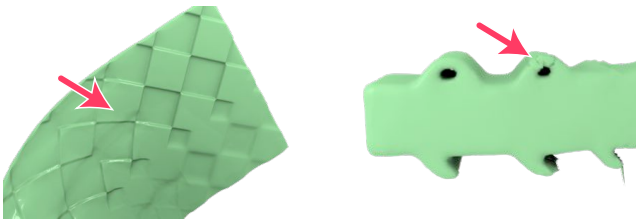
Fig. 19 Some failure cases due to coarse approximations (left) or topologically complex surface details (right).

national Symposium on Advancesin Visual Computing, pages 656–667, Berlin, Heidelberg, 2009. Springer-Verlag.

2. M. Attene. A lightweight approach to repairing digitized polygon meshes. *The Visual Computer*, 26:1393–1406, 2010. 10.1007/s00371-010-0416-3.

3. O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, 27:44:1–44:10, August 2008.

4. G. Barequet and M. Sharir. Filling gaps in the boundary of a polyhedron. *Comput. Aided Geom. Des.*, 12(2):207–229, 1995.

5. C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. Patchmatch: a randomized correspondence algorithm for structuralimage editing. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, pages 1–11, New York, NY, USA, 2009. ACM.

6. H. Biermann, I. Martin, F. Bernardini, and D. Zorin. Cut-and-paste editing of multiresolution surfaces. *ACM Trans. Graph.*, 21(3):312–321, 2002.

7. M. Bokeloh, M. Wand, and H.-P. Seidel. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph.*, 29:104:1–104:10, July 2010.

8. M. Botsch and O. Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):213–230, 2008.

9. L. Chen and X. Meng. Anisotropic resizing of model with geometric textures. In *SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 289–294, New York, NY, USA, 2009. ACM.

10. M.-M. Cheng, F.-L. Zhang, N. J. Mitra, X. Huang, and S.-M. Hu. Repfinder: finding approximately repeated scene elements for image editing. *ACM Trans. Graph.*, 29:83:1–83:8, July 2010.

11. N. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *Visualization and Computer Graphics, IEEE Transactions on*, 13(3):530 –548, may. 2007.

12. T. Derose and M. Meyer. Harmonic coordinates. In *Pixar Technical Memo 06-02, Pixar Animation Studios*, 2006.

13. M. Desbrun, M. Meyer, P. Schroder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

14. H. Fang and J. C. Hart. Detail preserving shape deformation in image editing. *ACM Trans. Graph.*, 26(3):12, 2007.

15. M. S. Floater. Mean value coordinates. *Comput. Aided Geom. Des.*, 20(1):19–27, 2003.

16. H. Fu, C.-L. Tai, and H. Zhang. Topology-free cut-and-paste editing over meshes. In *GMP '04: Proceedings of the Geometric Modeling and Processing 2004*, page 173, Washington, DC, USA, 2004. IEEE Computer Society.

17. R. Gal, O. Sorkine, N. J. Mitra, and D. Cohen-Or. iwires: an analyze-and-edit approach to shape manipulation. *ACM Trans. Graph.*, 28(3):1–10, 2009.

18. N. Gelfand and L. J. Guibas. Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '04, pages 214–223, New York, NY, USA, 2004. ACM.

19. K. Hormann and M. S. Floater. Mean value coordinates for arbitrary planar polygons. *ACM Transactions on Graphics*, 25:1424–1441, 2006.

20. V. Kraevoy, A. Sheffer, A. Shamir, and D. Cohen-Or. Non-homogeneous resizing of complex models. *ACM Trans. Graph.*, 27(5):1–9, 2008.

21. V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286, 2003.

22. S. Lefebvre and H. Hoppe. Parallel controllable texture synthesis. *ACM Trans. Graph.*, 24(3):777–786, 2005.

23. S. Lien and J. Kajiya. A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *CGA*, 4(10):35–41, 1984.

24. P. Liepa. Filling holes in meshes. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 200–205, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

25. Y. Liu, R. T. Collins, and Y. Tsin. A computational model for periodic pattern perception based on frieze and wallpaper groups. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26:354–371, March 2004.

26. Y. Liu, W.-C. Lin, and J. Hays. Near-regular texture analysis and manipulation. *ACM Trans. Graph.*, 23(3):368–376, 2004.

27. P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, 2006.

28. Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, New York, NY, USA, 2001. ACM.

29. M. Sabha and P. Dutré. Image welding for texture synthesis. In *Vision, Modeling, and Visualization 2006*, pages 97–104. IEEE Computer Society, 2006.

30. R. Schmidt and K. Singh. meshmixer: an interface for rapid mesh composition. In *SIGGRAPH '10: ACM SIGGRAPH 2010 Talks*, pages 1–1, New York, NY, USA, 2010. ACM.

31. A. Sharf, M. Blumenkrants, A. Shamir, and D. Cohen-Or. Snappaste: an interactive technique for easy mesh composition. *Vis. Comput.*, 22(9):835–844, 2006.

32. O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 179–188. Eurographics Association, 2004.

33. L. Szirmay-Kalos, V. Havran, B. Balázs, and L. Szécsi. On the efficiency of ray-shooting acceleration schemes. In *Proceedings of the 18th spring conference on Computer graphics*, SCCG '02, pages 97–106, New York, NY, USA, 2002. ACM.

34. D. Vaquero, M. Turk, K. Pulli, M. Tico, and N. Gelfand. A survey of image retargeting techniques. In *Proceedings of the SPIE 7798, 779814*, San Diego, California, August 2010.

35. H. Wu, Y.-S. Wang, K.-C. Feng, T.-T. Wong, T.-Y. Lee, and P.-A. Heng. Resizing by symmetry-summarization. *ACM Transactions on Graphics (SIGGRAPH Asia 2010 issue)*, 29(6):159:1–159:9, December 2010.

I. Alhashim is a Ph.D. candidate in the Graphics, Usability, and Visualization lab at Simon Fraser University in Burnaby, Canada. He also holds a B.Sc. from Portland State University, Oregon. His research interests include geometry synthesis and shape generation.
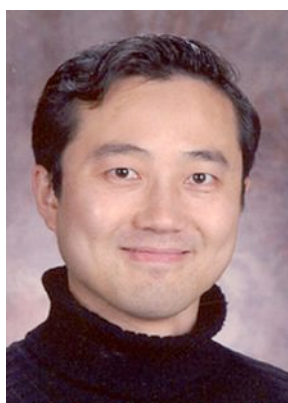
H. Zhang co-directs the Graphics, Usability, and Visualization Lab at Simon Fraser University, Canada, where he is an associate professor in the School of Computing Science. He received his Ph.D. from the Dynamic Graphics Project (DGP), Department of Computer Science, University of Toronto in 2003 and M.Math. and B.Math. degrees from the University of Waterloo. His research interests include geometry processing and computer graphics. Recently, he has served on the program committees of Eurographics, SGP, Pacific Graphics, among others. He was a winner of the Best Paper Award from SGP 2008.

L. Liu is a professor at Department of Mathematics, Zhejiang University. He received his PhD degree in computer graphics from Zhejiang University in 2001. After he had worked at Microsoft Research Asia during 2001 and 2004, he moved to Zhejiang University. He paid an academic visit at Harvard University during 2009 and 2011. His research interests include computer graphics, geometric modeling and processing, and image processing.