Tensor Field Design in Volumes

JONATHAN PALACIOS, LAWRENCE ROY, and PRASHANT KUMAR, Oregon State University
CHEN-YUAN HSU, Bournemouth University
WEIKAI CHEN, University of Hong Kong and USC Institute for Creative Technologies
CHONGYANG MA, Snap Inc.
LI-YI WEI, University of Hong Kong
EUGENE ZHANG, Oregon State University

3D tensor field design is important in several graphics applications such as procedural noise, solid texturing, and geometry synthesis. Different fields can lead to different visual effects. The topology of a tensor field, such as degenerate tensors, can cause artifacts in these applications. Existing 2D tensor field design systems cannot be used to handle the topology of a 3D tensor field. In this paper, we present to our knowledge the first 3D tensor field design system. At the core of our system is the ability to edit the topology of tensor fields. We demonstrate the power of our design system with applications in solid texturing and geometry synthesis.

CCS Concepts: • Computing methodologies → Texturing; Parametric curve and surface models;

Additional Key Words and Phrases: 3D tensor fields, tensor field design, topology, texture synthesis, geometry synthesis, element synthesis

ACM Reference format:
https://doi.org/10.1145/3130800.3130844

Fig. 1. *Different tensor fields can lead to different element synthesis results.* With our tensor field design system, users can create any tensor field and control the topology of the tensor field by deforming, removing, and reconnecting degenerate curves in the field (colored curves in (a)). The modified tensor field topology can lead to improved element synthesis results (before the topological editing (top) and after (bottom) in (b), (c), and (d)). The underlying 3D model is David’s head, which is viewed from the top.
1 INTRODUCTION

Many applications rely on the ability to locally model anisotropy inside a volume, such as texturing [Kopf et al. 2007; Lagae and Drettakis 2011], meshing [Ray et al. 2016], modeling [Ma et al. 2011], and deformation [von Funck et al. 2006]. In these applications, the quality of the results heavily depends on the properties of the guiding anisotropy field, such as continuity and smoothness. Thus, field design has been a key focus in computer graphics and geometry processing [Vaxman et al. 2016].

There are various types of volumetric fields, each with different symmetry properties and consequently suitable for different applications. For example, cross-frame fields (with octahedral symmetry) are ideal for hexahedral remeshing [Nieser et al. 2011] as they are able to properly model fundamental singularities (Figure 3a). On the other hand, the design and processing of cross frame fields is inadequate for the line-type and box-type of objects (e.g. Figure 3c). In contrast, tensor fields can model such objects (Figure 3d and Figure 2). This is because a 3D tensor contains not only directional information (eigenvectors) but also anisotropic sizing information (eigenvalues). Moreover, due to anisotropy, a tensor field can be considered as a collection of three mutually perpendicular line fields. In contrast, such a distinction is not available for cross-frame fields. Consequently, cross-frame field processing does not apply to line field design, while tensor field design naturally leads to line field design. Moreover, a tensor field naturally contains a linear region and a planar region (Section 3), which enables the use of two different elements in the same field (see Figure 2). A cross-frame field does not have this capability. On the other hand, tensor field singularities do not properly model basic irregular vertices and edges in hex meshes (Figure 3). Consequently, tensor field design is inadequate for hexahedral remeshing.

We address the problem of designing volumetric tensor fields. To our knowledge, there has been little work on 3D tensor field design. Existing work on the processing of cross-frame fields in hexahedral remeshing [Huang et al. 2011; Li et al. 2012; Nieser et al. 2011] does not naturally extend to the processing of tensor fields. Furthermore, existing work on cross-frame fields mostly focuses on the automatic generation and modification of a cross-frame field based on the boundary of the domain, while we strive for a user design system for flexibility. The user can generate any tensor field, whether aligned with the boundary or not.

A major challenge facing tensor field design is the existence of singularities, where the tensor field has repeating eigenvalues and the orthonormal frames have discontinuity. Similar to 2D tensor field design, the set of singularities in a 3D tensor field plays an important role in shaping the behavior of the tensor field. Unlike 2D tensor fields, where the singularities (referred to as degenerate points) are isolated points under structurally stable conditions, in

Fig. 2. Tensor field design naturally leads to line field design and frame field design. Given the tensor field designed by our system for a double torus (left: yellow and blue curves are degenerate curves and grey curves are hyperstreamlines following the minor eigenvector field), the tensor field can be used to generate element synthesis (right) with line-type elements (wood sticks) and elements with box-type symmetry (leaves). Moreover, a tensor field has both linear regions and planar regions (Section 3), naturally enabling two different elements to be synthesized with the same field. These properties are not shared by cross-frame fields.

Fig. 3. This figure compares the difference in the types of first-order singularities in a cross-frame field (left) and a tensor field (right). Both fields have a line of singularities (top row), and we show the pattern of the fields projected on a plane perpendicular to the singularity line. The projection of the cross-frame fields on the plane is a 4-RoSy field (a), while the projection of the tensor field is a 2D tensor field (b) (both major and minor eigenvector fields shown). Notice that a cross-frame field singularity has an index of $\frac{k}{4}$ ($k \in \mathbb{Z}$), while a tensor field singularity has an index of $\frac{k}{2}$ ($k \in \mathbb{Z}$). This is because cross-frames have a larger symmetry group than tensors. In quadrangular remeshing (c)-(d) (mesh elements highlighted by the black curves), valence 3 and 5 vertices cannot be modeled by a tensor field singularity, making 2D tensor fields inadequate for quadrangular remeshing. Similarly, 3D tensor fields are inadequate for hexahedral remeshing. On the other hand, for line type of objects (c)-(d) (blue stripes), 4-RoSy fields can lead to visual artifact (c), while 2D tensor fields do not have such artifacts (d).
In this paper, we present to our knowledge the first interactive, 3D tensor field design system. The user can create a tensor field by specifying desired tensor values and local patterns inside the volume, or on the boundary surface, or in both places. The field can be made boundary-conforming, i.e., the surface normal direction is aligned with one of the eigenvectors of the tensor field everywhere on the surface.

At the core of our system is a set of topological editing operations such as degenerate curve removal, degenerate curve deformation, and degenerate curve reconnection, which we have identified and used to control the number, location, and shape of degenerate curves. We also provide robust algorithms to accomplish these degenerate curve editing operations. Figure 1 shows the results of geometry synthesis guided by a tensor field with degenerate points.

We demonstrate the efficacy of our system by applying it to the control of anisotropic Gabor noise, data-driven solid texturing, and geometry element synthesis.

2 PREVIOUS WORK
There has been much work in the design of orientations fields in both 2D and 3D, and it is beyond the scope of this work to thoroughly review all of them. Instead, we will focus on work that is most relevant to this research and refer interested readers to a recently published survey [Vaxman et al. 2016] for a comprehensive review.

Vector field design on surfaces starts with applications in texture and geometry synthesis [Nieser et al. 2012; Praun et al. 2000; Turk 2001; Wei and Levoy 2001], fluid simulation [Stam 2003], and non-photorealistic rendering [Hertzmann 1998; Hertzmann and Zorin 2000]. Zhang et al. [2006] present a vector field design system in which they use Conley index theory to control the number and location of the singularities in the vector field. Fisher et al. [2007] use discrete exterior calculus to design a vector field with control over the singularities in the field. Palacios and Zhang [2007] address the more generic problem of designing 2D orientations under N-way rotational symmetries, which they refer to as N-RoSy’s. Ray et al. [2008] also develop a design system for such orientation fields under rotational symmetries with a focus on the control of location and type of the singularities in the field. They later automate this process to generate a field whose directions conform to the geometry of the surface [Ray et al. 2009]. Bommes et al. [2009] integrate field generation and quadrangular remeshing into the same system which they solve by a mixed-integer solver.

The analysis and design of 3D orientation field is a relatively new topic. Applications of 3D orientation fields include solid texture and geometry synthesis [Bénard et al. 2010; Kopf et al. 2007; Lagae and Drettakis 2011; Lagae et al. 2009; Landes et al. 2013; Ma et al. 2011; Takayama et al. 2008; Wei et al. 2009; Zhang et al. 2011] and hexahedral remeshing [Gregson et al. 2011; Huang et al. 2011; Lévy and Liu 2010; Li et al. 2012; Nieser et al. 2011]. However, it is difficult to specify and control the topology of the orientation fields, such as the number, location, shape, type, and connectivity of degenerate features [Ray et al. 2016; Solomon et al. 2017].

One exception is 3D symmetric tensor fields, for which there has been some work on topological analysis. Delmarcelle and Hesselink extend the notion of degenerate points from 2D tensor fields to 3D tensor fields [Delmarcelle and Hesselink 1994; Hesselink et al.
We have identified a set of editing operations that allows the topology of a 3D tensor field to be modified in an isolated fashion, i.e., without impacting other singularities. To do so, we rely on a number of observations on 3D tensor field topology that have not been made or reported previously. We also develop a unified framework in which these editing operations can be carried out. A preliminary version of this work is described in [Palacios et al. 2016a].

3 BACKGROUND

In this section we review background on 3D symmetric tensor fields mostly relevant to this work. This section is based on published works [Zhang et al. 2007; Zheng and Pang 2004]. As this paper only deals with symmetric tensors, we will omit the word symmetric.

A 3D (symmetric) tensor $T$ has three real-valued eigenvalues: $\lambda_1 \geq \lambda_2 \geq \lambda_3$. They are referred to as the major, medium, and minor eigenvalues, respectively. An eigenvector belonging to the major eigenvalue is referred to as a major eigenvector. Medium and minor eigenvectors can be defined similarly. Eigenvectors belonging to different eigenvalues are mutually perpendicular.

The trace of a tensor $T$ is $\text{trace}(T) = \sum_{i=1}^{3} \lambda_i$. $T$ can be uniquely decomposed as $D + A$ where $D = \frac{\text{trace}(T)}{3}I$ (I is the $K$-dimensional identity matrix) and $A = T - D$. The deviator $A$ is a traceless tensor, i.e., $\text{trace}(A) = 0$. Note that $T$ and $A$ have the same set of eigenvectors. Consequently, the topology of a tensor field can be defined in terms of the topology of its deviator tensor field. Another nice property of the set of traceless tensors is that it is closed under matrix addition and scalar multiplication, making it a linear subspace of the set of tensors. Given these considerations, we will focus on the analysis and design of traceless tensor fields and also omit the term traceless in the remainder of the paper.

When $K = 3$, a tensor can be classified as either linear (L), planar (P), or neutral (N), corresponding to $\lambda_1 - \lambda_2 > \lambda_2 - \lambda_3$, $\lambda_1 - \lambda_2 < \lambda_2 - \lambda_3$, and $\lambda_1 - \lambda_2 = \lambda_2 - \lambda_3$, respectively. For traceless tensors, the above conditions are equivalent to $\lambda_2 < 0$ (linear), $\lambda_2 > 0$ (planar), and $\lambda_2 = 0$ (neutral).

A tensor is said to be degenerate if it has repeating eigenvalues. When $K = 2$, the only degenerate (traceless) tensor is the zero matrix. When $K = 3$, there are three types of degenerate tensors: neutral ($\lambda_1 = \lambda_2 = \lambda_3$), linear ($\lambda_1 > \lambda_2 = \lambda_3$), and planar ($\lambda_1 = \lambda_2 > \lambda_3$). The neutral degeneracy is also referred to as the triple degeneracy, for which all non-zero vectors are an eigenvector. The linear and planar degeneracies are called double degeneracies. The non-repeating eigenvalue is referred to as the non-degenerate eigenvalue, while the repeating eigenvalues are referred to as the degenerate eigenvalues. The non-degenerate eigenvalue is the major eigenvalue for $L$-type degenerate tensors and the minor eigenvalue for the $P$-type degenerate tensors. The eigenvectors corresponding to the non-degenerate eigenvalues are referred to as non-degenerate eigenvectors. The plane perpendicular to a non-degenerate eigenvector is referred to as a degenerate plane. Any vector inside this plane is an eigenvector corresponding to the degenerate eigenvalue.

We now discuss tensor fields and their topology. A tensor field $T(p) (p \in \Omega)$ is a tensor-valued function defined over some domain $\Omega \subset \mathbb{R}^K$. A tensor field can be thought of as a tensor field consisting of its degenerate points.

In 2D, the set of degenerate points of a tensor field are isolated points under numerically stable configurations, i.e., when the topology does not change given sufficiently small perturbation in the tensor field. An isolated degenerate point can be measured by its tensor index [Zhang et al. 2007], defined in terms of the winding number of one of the eigenvector fields on a loop surrounding the degenerate point. The most fundamental types of degenerate points are wedges and trisectors, with a tensor index of $\frac{1}{2}$ and $-\frac{1}{2}$, respectively. The total tensor index of a continuous tensor field over a two-dimensional manifold is equal to the Euler characteristic of the underlying manifold. Consequently, it is not possible to remove one degenerate point. Instead, a pair of degenerate points with opposing tensor indexes (a wedge and trisector pair) must be removed simultaneously [Zhang et al. 2007].

In 3D, the situation is more complicated. Zheng et al. [2004; 2005a] point out that stable topological features consist of degenerate curves, each of which is either linear or planar. The projection of the tensor field onto the degenerate plane exhibits either a wedge or trisector pattern. Consequently, a degenerate curve can be divided into consecutive segments of purely wedge points and trisector points. The boundary points between these segments are transition points, where the non-degenerate eigenvector is perpendicular to the tangent of the degenerate curve. Figure 5 illustrates this with an example. Along degenerate curves, we can observe two 2D degenerate patterns such as wedges (left) and trisectors (right). At the transition point (middle), the projected pattern is neither a wedge nor a trisector.
We now describe our 3D tensor field design system, which consists of two steps. First, the user generates a tensor field by providing constraints inside the volume and/or on the boundary surface. Second, the user performs editing operations to individual degenerate curves of the tensor field. The user interface of our system is shown in Figure 6.

The computational setup of our system is as follows. The input domain is a tetrahedral mesh $M \subset \mathbb{R}^3$. Quite often the input is actually a triangle mesh representing a closed, orientable surface. In this case the volume bound by the triangle mesh is tetrahedralized using existing software such as TetGen [Si 2015]. A 3D tensor field is represented as a set of tensor values, one per vertex of $M$. Piecewise linear interpolation is used to extend the tensor values from a set of points (vertices of $M$) to a continuous tensor field over $M$. When performing tensor field design and editing, we modify the tensor field by modifying its values on the vertices of $M$. After each modification to the tensor field, the degenerate curves are extracted based on the method of Zheng and Pang [2004]. The LP-type as well as wedge/trisector classification along degenerate curves are also performed and color-coded. As shown in Figure 5, an L-type wedge degenerate point is colored in green, while an L-type trisector point is colored in blue. A P-type degenerate point is colored in yellow if a wedge, or red if a trisector. We will describe the first step, i.e., the specification of tensor fields through a set of user constraints, in Section 5. The discussion of topological editing of tensor fields is in Section 6.

5 3D TENSOR FIELD SPECIFICATION

Our specification system is based on a number of requirements. First, it is important to be able to specify the tensor value at a given point in space. Second, if the desired tensor value is degenerate, it is usually important to also specify the degenerate tensor pattern near the point of interest. Third, we would like the interface to be both intuitive and easy-to-use.

Our specification system consists of the following steps. First, the user places design elements using our design system, each of which is a desired tensor value at a given point, usually inside a tetrahedron in the mesh. Second, the design elements are used to generate tensor values at a set of vertices in the mesh, usually the vertices of the tetrahedrons that contain the design elements. Each of such vertices are referred as a fixed vertex. Finally, the tensor values at the fixed vertices are propagated to the remaining mesh vertices as follows.

Propagation. We treat each entry in the tensor field $T_{i,j}$ as a scalar field. A 3D symmetric, traceless tensor field can be considered as five scalar fields defined on the same mesh. The problem of propagating tensor values from the fixed vertices to the remaining vertices is thus converted into computing five scalar fields over the mesh. For each scalar field the values are given at the fixed vertices. To find the values at the remaining vertices, one can use the so-called Laplacian smoothing framework. Basically, for each scalar field, we create a harmonic vertex-based scalar field $\psi$ on $M$ given the boundary conditions (from the values at the fixed vertices). Note that $\psi$ is harmonic if it minimizes the following energy,

$$
\int_{V} \frac{1}{2} \| \nabla \psi \|^2 dx = \sum_{i \in T} \frac{1}{2} V_i \| \nabla \psi \|^2
$$

where $T$ is the set of $M$'s tetrahedrons, $V_i$ is the volume of the tetrahedron $t$, and $\nabla \psi$ is the gradient vector of $\psi$. When $\psi$ minimizes this energy we say that it solves the Laplace equation. This energy is equivalent to using cotangent edge weights [Jacobson 2013].

Once all five scalar fields have been obtained over the whole mesh, they are assembled into a $3 \times 3$ symmetric, traceless tensor field, which satisfies the boundary condition (fixed vertices). Note that this framework has been used successfully in 2D tensor field processing [Alliez et al. 2003; Zhang et al. 2007].

Design. Next, we focus on the details of the second step, i.e., computing the set of fixed vertices and tensor values at these vertices based on user-specified design elements. A design element can be either degenerate or non-degenerate. In the latter, it is in the form of a non-degenerate tensor. A degenerate design element is in the form of

$$
T_0 + (x - x_0)T_x + (y - y_0)T_y + (z - z_0)T_z
$$

Here $(x_0, y_0, z_0)$ is the 3D coordinates of the vertex, $T_0$ is a degenerate tensor, while $T_x$, $T_y$, and $T_z$ are trisectors that may be degenerate or non-degenerate. Together, the coefficients of $T_x$, $T_y$, and $T_z$ form the Jacobian of the tensor field and are responsible for the local tensor degenerate patterns (wedges, trisectors) around the point where the constraint is placed.

We have found that it is easier and often more intuitive to specify not just a single design element, but a set of design elements along a curve. This is especially true when specifying degenerate tensor elements, which naturally form curves. Given a user-specified curve, our system first generates a spline that best captures the sketched curve. The spline curve is then subsampled at a set of evenly spaced points.
The user can also rotate the non-degenerate eigenvector direction. The curve tangent, is assumed to change the eigenvectors by freely rotating the frame in 3D. The user may wish the tensor field to be aligned with the curvature tensor field on the boundary surface. We reuse the idea above, except that we apply the displacement vector 0 to the neighborhood as shown in Figure 7 (right). Then for each vertex \( v \) inside the neighborhood, we locate the closest vertex \( v_0 \) on the curve and applies the displacement vector \( v - v_0 \) into \( v_0 \)'s local linearization (tensor value and Jacobian) to obtain the desired tensor value at \( v \).

Consequently, we compute a tubular neighborhood for the specified curve (cylinder if open curve and torus if a loop). An illustration is shown in Figure 7 (left). We then compute a volumetric parameterization of this neighborhood with respect to the curve, which is equivalent to deforming the neighborhood into a canonical neighborhood as shown in Figure 7 (right). Then for each vertex \( v \) inside the neighborhood, we locate the closest vertex \( v_0 \) on the curve and applies the displacement vector \( v - v_0 \) into \( v_0 \)'s local linearization (tensor value and Jacobian) to obtain the desired tensor value at \( v \).

When two user-specified curves have intersecting neighborhoods, a conflict occurs for vertices inside the intersection. Consequently, we allow the desired tensor values from two curves to be blended, with each tensor value weighted by its distance to the respective curve. This way, a vertex closer to one curve will receive a larger influence from that curve. In our system, the radius of the neighborhood can be adjusted by the user. Generally speaking, the larger the radius, the better control over local tensor patterns around degenerate curves.

**Boundary-conforming tensor fields.** The users often wish the designed tensor field to conform to the boundary surface of the volume, i.e., one of the eigenvector fields is aligned with the surface normal everywhere on the boundary. However, tensor fields generated from the aforementioned Laplacian system are in general not boundary-conforming. To handle this difficulty, we perform one more Laplacian smoothing using some additional fixed vertices. Let \( T \) be the tensor field generated from design elements as described earlier. For each vertex \( v \) on the boundary surface of the domain, we modify the tensor value at \( v \) to be aligned with the boundary normal. Specifically, let \( \delta \) be a local basis at \( v \) such that the normal at \( v, N_v \), is the third vector in \( \delta \). Under this basis

\[
T(v) = \begin{bmatrix} M_B & M_{BN} \\ M_{BN}^T & M_N \end{bmatrix}
\]  

for a wedge constraint. A default Jacobian for a triector constraint simply negates \( T_y \). The user can change the 2D tensor pattern inside the degenerate plane by changing the 2 \( \times \) 2 sub-blocks in \( T_x \) and \( T_y \). The user can also rotate the non-degenerate eigenvector direction freely in 3D, which will lead to the rotation of the 3D frame but the Jacobian coefficients do not change with respect to the new local coordinate system. This can lead to change in the angle between the non-degenerate eigenvector and the curve tangent.

Parameterization. Notice that the subsamples along a user-specified curve are usually not the vertices of the mesh. Recall that in our framework, the fixed vertices are at the vertices of the mesh, not the middle of the tets. While it is possible to find the set of tets containing the user-specified curve and somehow transfer the tensor constraints from the curve to the vertices of these tets, we have found that such an approach leads to poor control over tensor patterns (especially degenerate patterns) near the degenerate curves.

Consequently, if the user wishes a 3D tensor field to be aligned with the curvature tensor field on the boundary surface, we reuse the idea above, except that we first need to convert the curvature tensor, a 2\( \times \)2 tensor, into a 3\( \times \)3 tensor. This is achieved by adding an eigenvalue corresponding to the new eigenvector, the surface normal. The user can also specify the new eigenvalue. Once the curvature tensor has been converted into a 3D tensor field, it is included in the boundary condition.
the boundary of the volume, we also use \( K = R \cap \partial M \) as a boundary condition.

6 TOPOLOGICAL EDITING OPERATIONS

The tensor fields generated in the first step (Section 5) often contain degenerate curves in addition to the ones specified by the user (Figure 8). As mentioned earlier, excessive and misbehaving degenerate curves can lead to visual artifacts in the applications of tensor fields. In this section, we describe three fundamental topological editing operations that we have identified as part of this research:

Degenerate curve deformation which refers to deforming part of a degenerate curve (Section 6.1).

Degenerate curve reconnection which refers to cutting open two degenerate curve segments and stitching together pieces from different segments, thus resulting in two new degenerate curve segments (Section 6.2).

Degenerate curve removal which refers to removing either one degenerate curve or simultaneously two degenerate curves, under respective conditions (Section 6.4).

These operations are designed to impact the least number of degenerate curves, while, together, can provide enough flexibility to modify tensor field topology. Figure 9 shows a sequence of topological editing operations on a tensor field. Next we describe our analysis and algorithms that enable these editing operations.

6.1 Degenerate Curve Deformation

Deforming a degenerate curve \( \gamma \) by itself is straightforward, but we need to resample tensor values at the surrounding mesh \( M \) vertices while keeping the rest intact (boundary condition). To deform part of a degenerate curve \( \gamma \), our system identifies a region \( R \) (Figure 10 (a-b): cylinder), a topological ball that encloses the segment \( \gamma_1 \subset \gamma \) (Figure 10 (b): curve) and its deformed version \( \gamma_2 \) (Figure 10 (a): curve) but no other degenerate points. Next, we strive for a self-homeomorphism \( \phi \) of \( M \) that maps \( \gamma_2 \) to \( \gamma_1 \). Finally, we use the map \( \phi \) to generate a deformed tensor field. Figure 9 shows the result of deforming the longest degenerate curve near Moai’s ear.

Region \( M \). We build the region surrounding the degenerate curves via a sequence of topology-aware morphological dilation and erosion operations. We start with the set of tetrahedrons containing \( \gamma_1 \) and \( \gamma_2 \). Typically the topology of this set, referred to as \( U \), is not a ball. We then compute the distance function \( d_U \) of any tetrahedron in the mesh to \( U \). Based on \( d_U \) we iteratively add one tetrahedron at a time to \( U' \) (initially the same as \( U \)) until \( U' \) is a ball. We then shrink \( U' \) by iteratively removing tetrahedrons from \( U' \) while keeping both \( \gamma_1 \) and \( \gamma_2 \) inside \( U' \) and requiring \( U' \) to remain a topological ball. Once this process terminates, we compute a new distance function, \( d_{U'} \), which measures the distance to the closest tetrahedron in \( U' \), and use it to grow the region \( M \) from \( U' \) out to a user specified distance (30% of the mesh radius works well in our experiments). A tetrahedron with the minimal \( d_{U} \) (or \( d_{U'} \)) will not be added if it contains part of a degenerate curve not intended by this editing operation.

Mapping \( \phi \). We construct the map \( \phi \) by treating it as a volumetric parameterization problem with the conditions that the parameterization is the identity map on the boundary of \( M \) and maps the desired degenerate curve segment \( \gamma_2 \) to the original curve \( \gamma_1 \). Note that degenerate curves are usually inside tetrahedrons. Consequently, we first establish a bijective map of \( \gamma_2 \) to \( \gamma_1 \) by sampling both curves with the same number of sample points. Next, we compute the minimal tetrahedral envelope for \( \gamma_2 \), i.e., any tetrahedron in the envelope...
contains at least part of $\gamma_2$. For each vertex $v_i$ in the envelope, we find the closest sample point $p_1$ on $\gamma_2$ and compute the displacement vector $dv_i = v_i - p_1$. We then set the constraint that $\phi(v_i) = q_i + dv_i$ where $q_i$ is the sample point on $\gamma_1$ corresponding to $p_1$. With these constraints, we solve the parameterization of the remaining vertices in $M$ by solving the Laplacian given by Equation (1) on the coordinates of the points. Note that since our map takes $\gamma_2$ to $\gamma_1$, we are essentially assigning a pre-deformation location to each vertex.

Note that the parameterization may have fold-overs and usually has suboptimal distribution of stretches. This can cause the resulting tensor field to be less smooth. If the fold-over tetrahedrons cover the degenerate curve, the editing operation may fail as there can be extra degenerate curves after the deformation. However, we find that this relatively simple method works well in practice.

**Interpolation.** Once the parameterization is given, we can look up the new tensor value at each vertex $v_i \in M$ by using its pre-image under the parameterization, $\phi(v_i)$. We then set the tensor value at $v_i$ to be that of $\phi(v_i)$. Figure 10 (c-e) shows the process of computing $\phi$ and deforming the tensor field with a 2D illustration.

### 6.2 Degenerate Curve Reconnection

Let $\gamma_1$ and $\gamma_2$ be two segments of degenerate curves that do not share any common degenerate points. Furthermore, the endpoints of $\gamma_1$ are $p_1$ and $q_1$ while the endpoints of $\gamma_2$ are $p_2$ and $q_2$. A degenerate curve reconnection of $\gamma_1$ and $\gamma_2$ will result in a new tensor field whose degenerate curves are the same as the original tensor field except that $\gamma_1$ and $\gamma_2$ are replaced with $\gamma'_1$ and $\gamma'_2$ where $\gamma'_i$ has endpoints of either $p_1$ and $p_2$ or $p_1$ and $q_2$ while $\gamma'_2$ is bounded by the other two points. Figure 11 shows the atomic scenarios.

Note that $(p_1, q_1, p_2, q_2)$ can be on the same degenerate curve or two different degenerate curves. Reconnecting them can change the number of degenerate curves in the field. For example, if these points are on a degenerate loop, after reconnection the loop can be broken up into two. Reconnection allows us to change the connectivity of the degenerate curves to facilitate degenerate curve deformation and removal. Figure 9 shows a reconnection of two mixed degenerate curves (both green/blue) near Moai’s bottom.

**Uniqueness of reconnection.** When two degenerate curves are reconnected, there appear to be two ways of doing so, i.e., (1) $p_1, p_2$ and $q_1, q_2$, and (2) $p_1, q_2$ and $p_2, q_1$. Consequently, it seems that the user would need to specify how the reconnection occurs. The following analysis shows that this is not necessary. Once the user specifies the curves to reconnect, there is only one way to reconnect them. To see this, we need the following result from [Markus 1955], which is also referred to as combing [Vaxman et al. 2016].

Given a line field $L$ defined in $\mathbb{R}^3$, let $B$ be a simply connected, finite region such that $L|_B$ has no singularities/zeros. Then $L$ can be turned into a continuous vector field without singularities.

We now consider a region $B$ where the non-repeating eigenvector field (a line field) has no singularities, then the non-repeating eigenvector field can be turned into a continuous vector field in $B$. Given a chosen orientation of the non-repeating eigenvector field inside $B$, the boundary of $B$ is divided into the incoming region, where the vector field $V$ points into $B$, and the outgoing region, where $V$ points to the outside of $B$ (Figure 11). The following observation is useful.

**Lemma 6.1.** A pure wedge (or trisector) curve intersecting the boundary of the above region $B$ must have its two endpoints in different regions, i.e., one in the incoming region, and the other outgoing region. In contrast, a degenerate curve with one wedge segment and one trisector segment must have its endpoints in the same region, i.e., either both incoming or both outgoing.

**Proof.** Trace the degenerate curve from one end so that the dot product between the forward curve tangent and the oriented non-degenerate eigenvector field is positive at the starting point. Notice that every time a transition point occurs, the dot product changes its sign, i.e., a turn has occurred. To travel from a point...
Theorem 6.1 states that there must be an odd number of transition points between two wedges. In other cases, the reconnection from left to right is unique for other cases. If $T(p_1)$ and $T(q_1)$ are both wedge types, then $p_1$ and $q_1$ cannot be reconnected. This is because if they could be reconnected, then there must be an even number of transition points between the wedges. On the other hand, since both $p_1$ and $p_2$ are in the incoming region, Theorem 6.1 states that there must be an odd number of transition points between $p_1$ and $p_2$, a contradiction. In this case, it is only possible to reconnect $p_1$ with $q_2$. Now, assume that $p_2$ and $q_2$ are trisectors. Using similar arguments (Theorem 6.1) it is straightforward to verify that $p_1$ cannot be reconnected with $q_2$. The only possibility is to reconnect $p_1$ with $p_2$, and $q_1$ with $q_2$. Similar analysis can show that reconnection is unique for other cases. 

Theorem 6.2. Given two degenerate curve segments $\gamma_1$ and $\gamma_2$ to be reconnected inside a simply-connected region $M$, there is only one way to reconnect them.

Proof. Assume that $p_1$ and $p_2$ are in the incoming region, and $q_1$ and $q_2$ are in the outgoing region (e.g. Figures 11a and 11b left). This means that $T(p_1)$ and $T(q_1)$ have the same wedge/trisector type, and $T(p_2)$ and $T(q_2)$ have the same wedge/trisector type. Without the loss of generality, assume that $T(p_1)$ and $T(q_1)$ are both wedge types. If $T(p_2)$ and $T(q_2)$ are both wedges, then $p_1$ and $p_2$ cannot be reconnected. This is because if they could be reconnected, then there must be an even number of transition points between the wedges. On the other hand, since both $p_1$ and $p_2$ are in the incoming region, Theorem 6.1 states that there must be an odd number of transition points between $p_1$ and $p_2$, a contradiction. In this case, it is only possible to reconnect $p_1$ with $q_2$, and $p_2$ with $q_1$. Now, assume that $p_2$ and $q_2$ are trisectors. Using similar arguments (Theorem 6.1) it is straightforward to verify that $p_1$ cannot be reconnected with $q_2$. The only possibility is to reconnect $p_1$ with $p_2$, and $q_1$ with $q_2$. Similar analysis can show that reconnection is unique for other cases. 

There are only five fundamental reconnection scenarios as shown in Figure 11. Note in case (a) there are two scenarios: from left (1 wedge and 1 trisector) to right (two hybrid wedge/trisector), and from right to left. In other cases, the reconnection from left to right and from right to left are equivalent.

Reconnection algorithm. The steps can be summarized as follows:

1. Compute a topological ball $M$ that contains $\gamma_1$ and $\gamma_2$ but no other degenerate points (Figure 12a). The process is similar to computing the envelope $B$ for degenerate curve deformation (Section 6.1).

2. Determine which pair of endpoints should be connected after reconnection.
3. Compute a sheet (Figure 12b: red plate) inside $M$ that separates $M$ into two regions, each of which contains precisely two endpoints that are connected after the reconnection.
4. Reassign tensor values on the interior vertices of the sheet so that there are no degenerate points on the sheet.
5. Recompute the tensor values for the remaining interior vertices of $M$ that are not already on the sheet. This is achieved by simultaneously solving Equation (1) for each entry in the tensor field.

Below we provide more details on steps 2, 3, and 4.

Step 2. To decide how to pair the endpoints after reconnection, we need to convert the non-repeating eigenvector field into a continuous vector field in $M$. This is achieved as follows. We first choose a vertex in the region $M$ and arbitrarily choose a forward direction from two of its unit non-repeating eigenvectors. We then parallel transport this vector to all the other vertices in $M$ and use it to choose the forward direction of each of the non-repeating eigenvectors. This will lead to the conversion of the non-repeating eigenvector field on $M$ to a vector field since it is free of singularities as a line field. Depending on the type of degenerate points the endpoints of the curves are (wedge or trisector) and where they situate on the boundary of $M$ with respect to the vectorized non-repeating eigenvector field (incoming or outgoing), we determine which pair of endpoints to reconnect after reconnection using the idealized illustrations in Figure 11.

Step 3. To generate the sheet, we compute two curves, one for each pair of endpoints after reconnection, by using Fast Marching [Kimmel and Sethian 1998] to find the shortest path between the desired endpoints. We require that the two curves do not intersect, which is generally the case based on our observation. If there are intersections, we perform local adjustment in $M$ to remove the intersections. Next, we compute the distance function from these
two curves to the rest of the vertices in the region $M$ and record which curve is the closest to the vertices. We then extract all the edges in $M$ whose two vertices have different closest curves. The set of tetrahedrons that contains these edges is a thickened version of the sheet.

Step 4. To strive for a degenerate point free tensor field in the sheet, we consider a more generic problem: given a simply-connected, topological ball over which the tensor field’s non-repeating eigenvector field has no singularities, how to modify the tensor field in the region to be free of degenerate points with tensor values fixed at part of or the whole boundary of the region? The algorithm to achieve this is not only useful for degenerate curve reconnection, but also degenerate curve removal (Section 6.4). We provide the details of this in Section 6.3.

6.3 Singularity-Free Tensor Field Generation

Given a simply-connected region $M$ where the non-repeating eigenvector field has no singularities, any degenerate point must be a singularity in the medium eigenvector field. This implies that to compute a degenerate-point-free tensor field inside $M$, we only need to ensure that the medium eigenvector field has no singularities. In addition, as pointed out earlier (Section 6.2), the non-repeating eigenvector field in $M$ can be turned into a continuous vector field. Consequently, this vector field can be considered as a map $\phi$ from $M$ to the unit sphere $S^2$. Given a point $p_0 \in M$, a unit medium eigenvector at $p_0$ can be considered as a tangent vector at $\phi(p_0)$. Figure 14 illustrates this map.

Angle field. We further model the medium eigenvector field as a scalar field, i.e. its angular component, and perform Laplacian smoothing (Equation (1)) on this scalar field with the given boundary conditions. The resulting scalar field (angular component of the medium eigenvector field) will then be combined with the non-repeating eigenvector field (not changed in the process) to generate the new tensor field inside region that is free of degenerate points.

Note that the angular component of a vector depends on the coordinate system. Give two points $q_1$ and $q_2$ on $S^2$, their tangent planes are not the same, nor are their respective coordinate systems correlated. We thus need a continuous, singularity-free vector field defined on $M$ that is also perpendicular to the non-repeating eigenvector field everywhere in the domain. This vector field, denoted by $U$, serves as the references with which we compute the angular component of the medium eigenvector field.

Parallel transport. To compute this reference vector field $U$, one can start with a seed vertex in the region $M$ and assign a reference vector there. This reference vector is then iteratively propagated to the remaining vertices in $M$ through its edges (e.g. breadth-first search). Given an edge $e = (v_0, v_1)$ where the reference vector $U(v_0)$ is available, we obtain $U(v_1)$ by parallel transporting $U(v_0)$ along the geodesic connecting $\phi(v_0)$ and $\phi(v_1)$ (Figure 15).

Jump. Results from classical differential geometry [Lai et al. 2010] state that when parallel transporting a vector along a simple loop on the sphere, the resulting vector may differ from the initial vector by a 2D rotation. The angle of the rotation is proportional to the area of the region enclosed by the loop. This means that when parallel transporting the reference vector $U(v_1)$ from a vertex $v_0$ to $v_1$ and then to $v_2$ where $v_0, v_1, v_2$ form a spherical triangle (Figure 15), the resulting reference vector $U(v_2)$ will be different from directly parallel transporting $U(v_0)$ to $v_2$ along the edge $v_0v_2$. The angular difference $\alpha$ is proportional to the signed area of the spherical triangle $\phi(v_0)\phi(v_1)\phi(v_2)$. We refer to $-\alpha$ as the jump from $v_0$ to $v_2$. Note that in our reference propagation process, the jump is zero for edges used in the propagation and usually non-zero for the ones not used.

Propagation. If jumps are not properly accounted for, singularities can occur in the medium eigenvector field [Lai et al. 2010; Li et al. 2006]. Consequently, our reference propagation algorithm also simultaneously computes jumps for edges not used in the propagation.

As illustrated in Figure 16a, we start with a seed vertex $v_0$ where the reference vector is already defined. We then select one of the tetrahedrons incident to $v_0$ as the seed tetrahedron. Denote this tetrahedron as $t_0$, which has three other vertices $v_1, v_2, v_3$. We first propagate the reference vector from $v_0$ to $v_1$ and set the jump to zero on the edge $v_0v_1$. We handle $v_2$ and $v_3$ in a similar fashion. Next, we set the jump on edge $v_1v_2$ to be the negative of the signed area of the spherical triangle $\phi(v_0)\phi(v_1)\phi(v_2)$. The jumps for the edges of $v_2v_3$ and $v_3v_1$ are computed in the same way.

Once $t_0$ has been processed, we perform region growing from $t_0$ until all the tetrahedrons in $M$ have been covered as demonstrated in Figure 16b. In every iteration, a new tetrahedron $t$ in $M$ that shares a face with the set of visited tetrahedrons is included in the set. Moreover, we require that at any given moment the set of visited tetrahedrons forms a topological ball. Note that there is at most one vertex of $t$ that has not been assigned a reference vector, and at most
We then take into account the jumps. That is, given a vertex $v_0$ has been computed for all the fixed vertices of the region $M$, the jumps computed, we convert the medium eigenvectors at the vertices in the tetrahedron to be the medium eigenvector at $v_0$. To compute the medium eigenvector there, we obtain one of the three vertices where the reference vectors are available, with the corresponding edge given a zero jump (red). The other edges (black) will have their jumps computed based on the areas of triangles with only one black edge.

For vertices in $M$ by solving Equation (1). Similarly, we compute the major eigenvalues in $M$. Finally, since the tensor field is traceless, the medium eigenvalue field can be obtained from the major and minor eigenvalue fields. This leads to a tensor field in $M$ free of degenerate points.

### 6.4 Degenerate Curve Removal

The third topological editing operation is degenerate curve removal, which is responsible for reducing the number of degenerate curves in the field. Figure 9 shows the removal of a degenerate loop inside Moai’s body (yellow/red).

Recall that in 2D tensor fields, at least two degenerate points with opposite tensor indices (wedge and trisector) must be removed simultaneously. However, in 3D tensor fields, there does not exist a known index to measure degenerate points or curves. A degenerate curve may consist of both wedge and trisector segments, and each degenerate curve has an additional linear-planar ($L$–$P$) classification. These additional complexities make it more challenging to decide the necessary and/or sufficient conditions for degenerate curve removal.

Empirically, we have identified the following four scenarios of degenerate curve removal (Figure 17):

- **One-curve removal** One open degenerate curve with opposite wedge/trisector types at its ends is removed (Figure 17a).

- **Two-curve removal** One open wedge degenerate curve is cancelled with one open trisector degenerate curve of the same linear-planar type (Figure 17b).

- **One-loop removal** One degenerate loop with both wedge and trisector sections is removed (Figure 17c).

- **Two-loop removal** A pure wedge degenerate loop is cancelled with a trisector degenerate loop of the same linear/planar type (Figure 17d).

For all but the last case, the enclosing region is a topological ball (simply connected). For each case the removal operation is always theoretically possible if an enclosing topological region can be found that intersects no other degenerate curves. Our degenerate curve removal algorithm for these cases again makes use of the fact that inside these enclosing regions (if they exist), the non-repeating eigenvector field is a vector field. Therefore, we employ a similar framework to that of degenerate curve reconnection (Section 6.2).

Next, we compute $\theta$ for vertices in $M$ that are not part of the boundary condition by using the $\theta$ values at the fixed vertices. To solve for the new angular field, we need to modify Equation (1) to take into account the jumps. That is, given a vertex $v_0$ and the set of vertices $\{v_1, ..., v_k\}$ adjacent to $v_0$, the Laplace of the angular component is

$$\sum_{j=1}^{k} w_j \left( \theta(v_j) - \theta(v_0) - g_{0j} \right)$$

where $g_{0j}$ is the jump from $v_0$ to $v_j$ and $w_j$ is the cotangent weight [Jacobson 2013].

Finally, the $\theta$ field is converted back to a line field by using the reference vector field $U$. This is the singularity-free medium eigenvector field, which, by being combined with non-repeating eigenvector field (not changed during the process), generates the new eigenvector fields in $M$ that is free of degenerate points.

In addition to the eigenvector fields, we also need to compute the eigenvalue fields in order to obtain the tensor field in $M$ free of degenerate points. This is achieved as follows. We consider the major eigenvalue field as a scalar field. Given the major eigenvalues at the fixed vertices of $M$, we find the major eigenvalues elsewhere in $M$ by solving Equation (1). Similarly, we compute the minor eigenvalues in $M$. Finally, since the tensor field is traceless, the medium eigenvalue field can be obtained from the major and minor eigenvalue fields.
for degenerate curve deformation and reconnection. Next, we use the tensor values on the boundary $M$ as constraints and generate the tensor field inside $M$ using the angle-based method in Section 6.3.

For two-loop removal, the enclosing region is a topological solid torus (not simply-connected). Consequently, the angle-based method (in particular, the jump assignment step) would not always result in a singularity-free continuous tensor field inside the region. Therefore, we implement two-loop removal operation as a composite operation. By first reconnecting the two loops, we obtain a single loop with both wedge and trisector segments that can be handled with the one-loop removal.

For all our topological editing operations, the success depends on the shape and size of the region $M$. The larger and rounder the region, the higher the success rate is. We have conducted 47 experiments of applying topological editing operations to fields defined on various 3D models such as the bunny, Moai, buddha, and dragon. The fields were generated either through manual design or, more commonly in our experiments, extrapolation from the curvature tensor on the boundary surfaces. Our algorithms have a success rate of over 90%.

7 PERFORMANCE

Our tool has been tested on a system with Intel(R) Xeon(R) CPU with 3.40 GHz speed with a RAM of 64 GB and an Nvidia Quadro K420 graphics card. We have tested our design system on various models, with a resolution varying from 150,000 tets to 1,000,000 tets. These models can be mathematically defined (e.g., sphere/torus), organic (e.g., Stanford bunny, feline, fertility), or CAD models (e.g., rocker arm and fan disk). The boundary of these tet meshes have a range of 10,000 to 60,000 triangles. The tet meshes are usually generated from the boundary mesh using TetGen, which typically leads to adaptive meshes where the tets are much smaller around sharp features. Our system can handle such meshes since we do not assume a minimal tet mesh resolution anywhere in our algorithm and implementation. Our system can also be used to generate tensor fields on meshes with sharp edges, such as the rocker arm. Solving for the field from boundary or interior constraints takes less than five seconds on all of our test models. All of the editing operations finish within 10 seconds for most models, and within 60 seconds for larger models (> 500,000 tets).

8 APPLICATIONS

In this section we demonstrate applications of our method in solid texture and geometry synthesis.

8.1 Solid Texturing

Example based solid texture. In 2D texture synthesis the goal is to take a small input exemplar texture and generate larger output texture with the same visual characteristics; this problem is well-explored as surveyed in [Wei et al. 2009]. Solid texture synthesis is the logical extension of this problem to the voxel case, and relatively fewer methods have been proposed [Dong et al. 2008; Owada et al. 2004; Pietroni et al. 2007; Takayama et al. 2008; Zhang et al. 2011]. Interestingly, with the exception of [Takayama et al. 2008] most methods still rely on 2D exemplars as the direct input, as they are easier to acquire. When the input texture exhibits anisotropy, an orientation field is needed to guide the orientation and placement of the exemplar texture [Takayama et al. 2008; Zhang et al. 2011]. As generating smooth fields that naturally follow shapes can easily be accomplished by our method (nearly automatically, when using the curvature tensor), our system readily benefits this application. The only difference is that a tensor field is sampled instead of a vector or frame field, and the eigenvectors must be computed and used as the local orientation for each neighborhood. As long as the texture is symmetric (or at least nearly so), the algorithms work without other modification. We adapt the method of [Zhang et al. 2011] to use symmetric tensor fields. See Figure 18 for results.

Anisotropic procedural noise. Procedural noise is a powerful way to model natural phenomena and enrich visual details, with particular advantages such as fast evaluation, low memory consumption, virtually infinite resolution, and appearance control [Ebert et al. 2002; Lagae et al. 2010]. Of particular interest to us is solid anisotropic noise, such as that introduced in [Lagae et al. 2009] and extended in [Bénard et al. 2010; Lagae and Drettakis 2011], which requires an axis at each point in the target domain to guide the orientation of the corresponding anisotropic noise pattern. Vector fields are unable to model all of the features that anisotropic Gabor noise is capable of exhibiting. As far as we know, the results generated in [Bénard et al. 2010; Lagae and Drettakis 2011; Lagae et al. 2009] use only surface tensor fields for 2D noise and constant or simple 3D vector field functions to orient solid noise, likely because of the lack of methods for easily producing 3D tensor fields. Our system addresses this need, and allows the generation of much more complex effects. Also, we find that Gabor noise makes for an apt texture-based method of interactively visualizing 3D tensor fields, because of its fast computation. Examples of our artistic results using Gabor noise can be seen in Figure 18.

8.2 Geometry Synthesis

Placing and orienting 2D image/text and 3D geometry elements has shown to be an important application for many graphical and interactive tasks [Gal et al. 2007; Hausner 2001; Ijiri et al. 2008; Landes et al. 2013; Ma et al. 2011; Maharik et al. 2011; Ostromoukhov and Hersch 1999; Pedersen and Singh 2006; Praun et al. 2001; Roveri et al. 2015; Xu and Kaplan 2007]. These applications require good orientation fields for the corresponding domains for maximal visual benefits. However, as discussed in Section 2, the design of 3D fields has been much less explored than the 2D counterparts. Thus, our method offers benefits for applications that require the placement of 3D elements as exemplified by [Ma et al. 2011].
Fig. 18. Solid texture synthesis based on examples (horse, bunny, and feline) and procedures (Gabor noise rocker arm and streamtubes bunny). The output is rendered through different cross sections (bottom row) to illustrate its volumetric nature.

Fig. 19. 3D geometry element synthesis. Bamboo, stick, and wire patterns are line-type elements while leaf and chain patterns have box-type symmetry.

Unlike prior methods which place elements according to a vector or frame field, our method deals with tensor fields which can generate line fields as well as frame fields satisfying the box symmetry (the Klein group [Conway et al. 2016]). Figure 19 shows various elements applied to a variety of 3D models with tensor fields designed using our tensor field design system. Moreover, due to the co-existence of linear/planar regions in a tensor field, we can have two different elements in the same field (Figure 2). Figure 20 shows the synthesis results using fields that were created completely from scratch and are not aligned with the boundary surface (the sphere in this case). Elements can also be used to visualize the effects of topological editing operations as shown in Figures 1 and 21.

9 LIMITATIONS AND FUTURE WORK

In this paper we introduce the problem of 3D tensor field design and have identified a number of graphics applications. We also provide the first 3D tensor field design system that is interactive, intuitive, and efficient. At the core of our system we provide the capability to design and control degenerate curves, which is tensor field topology. With these we have made a number of theoretical observations of 3D tensor field topology and identified a set of fundamental topological editing operations.

Our system has several limitations. First, our topological editing operations can fail. In the future, we plan to seek more theoretical understandings as well as explore improved algorithms that can lead to topological editing algorithms with guaranteed success. Second, our degenerate curve deformation algorithm relies on the ability to compute a bijective space warping of the region in which the operation is performed. However, we cannot guarantee that the warping is indeed bijective, i.e., foldovers may occur. We plan to explore volumetric parameterization techniques that are guaranteed to produce low-distortion and bijective maps. A series of recent work on the topic has the potential of benefiting our method [Aigerman and Lipman 2013; Campen et al. 2016; Kovalsky et al. 2015, 2016; Rabinovich et al. 2017; Schüller et al. 2013].

In the future, we plan to also improve the UI of our system to make it more accessible to non-expert users. Moreover, we plan
We would like to thank Guoxin Zhang, Shimin Hu, and Kun Xu for their help in proofreading the paper; and the Kong RGC general research fund 17202415.

ACKNOWLEDGMENTS

We would like to thank Guoixin Zhang, Shimin Hu, and Kun Xu for sharing the code of [Zhang et al. 2011]; Yue Zhang, Amy Roy, and Tom Roy for their help inproofreading the paper, and the anonymous reviewers for their valuable suggestions. This work has been partially supported by US National Science Foundation awards (0546881, 0830808, 0917308, 1340112, and 1619383) as well as Hong Kong RGC general research fund 1720415.

REFERENCES


Yufei Li, Yang Liu, Weixei Xu, Wenzhuo Wang, and Baining Guo. 2012. All-hex Meshing Using Singularity-restricted Field. ACM Trans. Graph. 31, 6, Article 177 (2012), 11 pages.


Fig. 21. A degenerate curve removal example. The top and bottom rows show respectively the tensor fields before and after the removal of some degenerate curves as visualized in (a) (degenerate curves only) and (b) (using streamtubes). The differences between the tensor fields are clearly reflected in the element synthesis results (c).