Video Vectorization via Tetrahedral Remeshing

Chuan Wang†§, Jie Zhu‡, Yanwen Guo‡, Wenping Wang†, *Fellow, IEEE*

***Abstract*—We present a video vectorization method that generates a video in vector representation from an input video in raster representation. A vector-based video representation offers the benefits of vector graphics, such as compactness and scalability. The vector video we generate is represented by a simplified tetrahedral control mesh over the spatial-temporal video volume, with color attributes defined at the mesh vertices. We present novel techniques for simplification and subdivision of a tetrahedral mesh to achieve high simplification ratio while preserving features and ensuring color fidelity. From an input raster video, our method is capable of generating a compact video in vector representation that allows a faithful reconstruction with low reconstruction errors.**

***Index Terms*—video, vectorization, representation**

I. Introduction

Image vectorization is becoming increasingly important as vector-based graphical contents are being gradually used in computers, by cellphones and on the Internet. The problem becomes particularly needed as devices with different resolutions and DPIs from cellphones, tablet PCs to HDTVs dominate the market. One may have the experience that when switching from a low DPI display to a higher one, the physical size of a raster image will become smaller, causing users' eyes feel unconformable. In this case, if we simply magnify the original image using interpolation, there will introduce obvious zig-zag artifacts along the feature boundaries. The essential reason behind is that raster images are of a discrete representation, which has a fixed resolution. In contrast, vector graphics use a continuous representation which provides an intrinsic advantage, *pixel resolution independent*, which enables images displayed with significantly varying resolutions without zig-zag artifacts. Moreover, resolution independence also enables the storage of a vector graphics constant to the varying resolutions of displays, making it a *compact* representation, especially when the display's resolution is huge. Driven by this demand, there has recently been a resurgence of interest in the research of image vectorization techniques [1–5] which aim to convert a raster image into a vector graphics. Similarly, with videos becoming a more important media form, converting legacy raster videos into vectorized representation is also becoming an imperative issue, which can also bring the two advantages of vector graphics. However, to the best of our knowledge, there is little work in the literature on video vectorization.

Generating a vectorized video from a raster video is a challenging task. Simply applying existing image vectorization techniques to individual frames would not work because the lack of consideration of temporal coherence between video frames would lead to unacceptable flickering. It is thus natural to treat the input video as a spatial-temporal volume and consider what is the most suitable geometric representation for the purpose of video vectorization. Inspired by existing image vectorization techniques which are mostly based on a 2D mesh representation, as the gradient quadrilateral mesh [1, 2] and triangular patches with curved or straight boundaries [4, 5], we propose to use 3D tetrahedral meshes for video vectorization. Note that, however, this extension from 2D meshes to 3D meshes for video vectorization is far from straightforward and a number of technical challenges need to be addressed to develop an effective video vectorization method.

Our approach

We present an effective method for generating a resolution-independent vector-based video from an input raster video. Our vector representation is based on a sparse tetrahedral mesh with colors defined at its vertices. The core of our method consists of new techniques for simplification and subdivision of tetrahedral meshes defined over the spatial-temporal volume of the input video.

A basic requirement for vectorized video representation is feature preservation. To meet the requirement, we first over-segment the input video into a set of *temporal superpixels* (TSPs) whose boundaries conform with important visual feature and region boundaries, thereby preserving the features. An initial regular tetrahedral mesh is built over the piecewise defined, possibly discontinuous color field defined on the TSPs. Then we simplify the initial mesh to obtain a feature-preserving sparse tetrahedral mesh while preserving features, which constitutes the vector representation of the video. Our mesh simplification algorithm uses iterative edge contractions based on an extended quadric error metrics (QEM) as approximation error. Finally, for video displaying we propose a tetrahedral mesh subdivision method for reconstructing details from the sparse control mesh to produce a raster video. Fig. 9 shows an example of a vectorized video by our method. We validate the effectiveness of our approach with extensive experiments.

In summary, we present an effective video vectorization algorithm based on simplification and subdivision of tetrahedral meshes. This is the first vectorization method for generic

†: Department of Computer Science, The University of Hong Kong, Hong Kong, China. E-mail: cwang@cs.hku.hk, wenping@cs.hku.hk

‡: National Key Lab for Novel Software Technology, Nanjing University, Nanjing, China. E-mail: magickuang@126.com, ywguo@nju.edu.cn. Yanwen Guo is the corresponding author.

§: Big Data and Machine Learning Lab, Lenovo, Hong Kong.

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org., provided by the author. This is a supplementary video that demonstrates the approach for converting raster video into its vectorized version. Contact cwang@cs.hku.hk for further questions about this work.

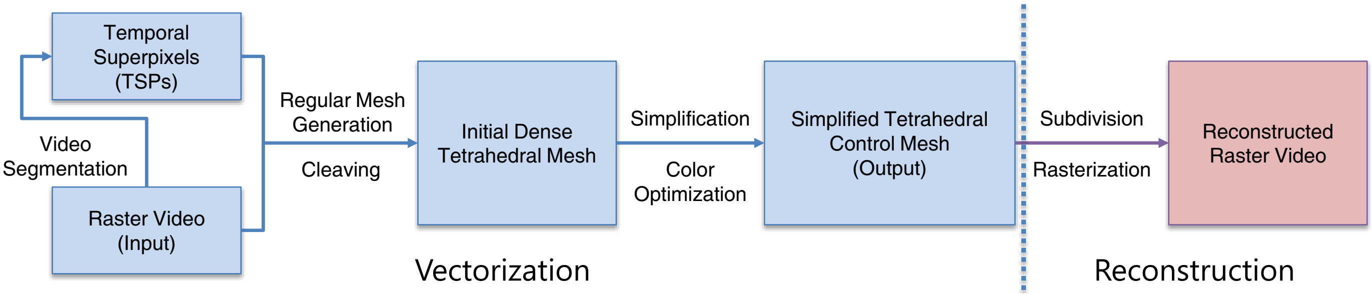


Fig. 1. Pipeline of our video vectorization (left of dashed line) and reconstruction (right of dashed line) framework.

videos. Our method achieves very high simplification ratio (typically 1%) and well preserves visual features and color fidelity.

II. Related Work

A. Image vectorization

There has been much work on vectorization of non-photographic images, such as fonts, cartoon [6] and line drawings [7, 8]. Recently, vectorization of photographic images has aroused a lot of interest [1–5, 9, 10]. Some methods produce explicitly a parametric surface mesh associated with color as the vector-based image representation. The geometric primitives include the gradient mesh consisting of topologically planar Ferguson patches with mesh-lines [1, 2], triangular patches with curved boundaries [4], and the traditional triangular meshes with straight edges [5]. Other methods adopt a mesh-free representation. For instance, *diffusion curves* [3] creates curves with color and blur attributes, and models the color variation as a diffusion from these curves by solving a Poisson equation. [11] automatically generates sparse diffusion curve vectorizations of raster images by fitting curves in the Laplacian domain.

We note that it is infeasible to simply extend the above image vectorization techniques to each video frame independently because spatial coherence of the video cannot be guaranteed in this way. In contrast, our method aims to produce a *three-dimensional*, vector-based video representation by working directly on the spatial-temporal video volume.

B. Cartoon vectorization

Some methods work on cartoon vectorization [12, 13]. Zhang et al. [14] proposed a layered representation of cartoon animations in which the pre-segmented regions and decorative lines are vectorized separately. Our approach differs from this method in several aspects. First, our approach is suitable for generic video vectorization and can be applied to cartoon animation as well, while this method is only applicable to vectorization of cartoon animations by taking advantage of the particular nature of cartoons, such as easy image decomposition into background and foreground, and distinctive decorative lines, which are not possessed by generic videos. Therefore, this method is not general enough to be applicable to full-color videos. Second, our method yields a unified vector-based video representation, while the output of this method is the vectorization of segmented regions, decorative lines, together with their motions across frames. Third, our method works automatically, while user-assistance is required in [14] to complete the static background by this method.

C. Vectorization for other applications

It is noted that vector representation has also been used to model solid textures [15] and volumetric objects [16], and to achieve resolution independent texture mapping [17].

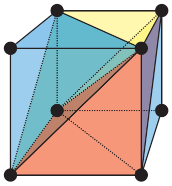
III. Overview

Our video vectorization framework takes a raster video as input and produces a tetrahedral mesh as the final vector representation. Specifically, the pipeline consists of the following steps: for an input raster video, we first segment the video into multiple temporal superpixels. Then we create an initial dense tetrahedral mesh by regular mesh generation and cleaving. Further, simplification and color optimization are performed to generate the final simplified tetrahedral control mesh, that is our proposed vector-based representation of the video. Finally, subdivision and rasterization are used to reconstruct raster video for display. (See Fig. 1).

A. Feature detection via video segmentation

The essence of video vectorization is to find functions to fit the color of the raster video. The edge features, or the boundaries of color instant change in the input video should be represented by discontinuous functions to avoid blur at the boundaries. For single image vectorization, Canny edge detector [18] is often used for this purpose. However, to our best knowledge the similar idea of detecting "3D edges" cannot be easily generalized to video sequences. Consequently, to extract the boundary features in 3D video volume, we perform video segmentation to partition a video clip into *temporal superpixels* (TSPs). One TSP is a set of pixels across multiple frames with similar colors, and in this paper we apply [19] because it can generate higher quality segmentations compared with other existing methods like [20–22], which are temporally coherent with stable region boundaries. Meanwhile, each TSP generated by [19] has smooth internal color changes, which makes it easy to be fitted by a low-order continuous function as the tetrahedral mesh represents. Otherwise, the color will be difficult to fit, incurring obvious artifacts when the video is zoomed in. Also, after this step each pixel is tagged with the ID of the TSP containing it.

B. Regular tetrahedral mesh generation and cleaving



We generate an initial tetrahedral mesh from the original video with each pixel as a vertex in the mesh. Specifically, we view the video as a 3D volume and create a regular 3D lattice with all the vertices located at video pixels. Then for each neighboring $2 \times 2 \times 2$ vertices forming a cell, we split it into six lattice tetrahedra with equal volume as the right figure shows. Compared with more advanced meshing algorithms like 3D Delaunay Triangulation [23], this method has two advantages. First, based on the pixel locations of a video, creating such a regular 3D lattice is intuitive and straightforward; Second, since each vertex denotes a pixel in the video, such a dense mesh actually keeps all the color information. The mesh is thus a perfect, geometric fit for the video, enabling us to leverage some powerful tools in geometry processing for video vectorization. To transfer the feature boundary information in the segmented video to the initial mesh, we need to separate the initial mesh into multiple sub-meshes with each corresponding to one TSP in the video. For this purpose, we apply a multi-material tetrahedral meshing algorithm Cleaver [24] to the lattice tetrahedral mesh. Here each TSP is viewed as one material. The Cleaver algorithm splits each tetrahedron whose four vertices have inconsistent TSP IDs into several smaller tetrahedra, to ensure that the feature boundaries detected by video segmentation do not pass through any tetrahedron. Specifically, there are five unique interface patterns that Cleaver handles, corresponding to different patterns of TSP IDs tagged at the vertices (Fig. 2 (a)∼(e)). For each pattern, Cleaver re-meshes the tetrahedron into smaller tetrahedra considering topology consistency of adjacent tetrahedra (see Fig. 2). After being processed by Cleaver, the lattice tetrahedral mesh becomes an even denser mesh, and it is ready for subsequent simplification. Meanwhile, this initial mesh has the following properties:

- The tetrahedral mesh is cut into several parts. Each part corresponds to a TSP of the original video. We call each part "*sub-tet-mesh*" in the rest of this paper.
- Each tetrahedron has a unique TSP ID, meaning that it belongs to only one TSP.
- Each original vertex is located at the internal region of a tetrahedron, meaning that it has a unique TSP ID, and we call it an "*internal vertex*".
- The newly-added vertices on the cutpoints are at sub-pixel locations so they are shared by tetrahedra with different TSP IDs. Such a vertex is called a "K-class *boundary vertex*", where K is the number of the TSPs sharing the vertex, so it has been naturally determined by the previous video segmentation process. Typically K ranges from 2 to 5 in our experiments.

C. Tetrahedral mesh simplification and color optimization

With colors defined at mesh vertices, the initial tetrahedral mesh is a dense geometric representation of the input raster video. To remove redundancy in the mesh, we shall simplify this mesh to obtain a compact representation while maintaining an accurate approximation of the color field. To this end, we apply an edge contraction based simplification method where the edge contraction operation is guided by an error metric similar to that used in the Quadric Error Metrics (QEM) mesh framework in [25]. However, unlike QEM, we introduce an additional term to encourage mesh uniformity. Please refer to Section IV for detailed explanations about our simplification scheme. After mesh simplification, we perform boundary surface fitting to improve the boundary accuracy of the sub-tet-mesh. In addition, the colors on vertices are optimized so that the reconstructed video produces the minimal reconstruction error.

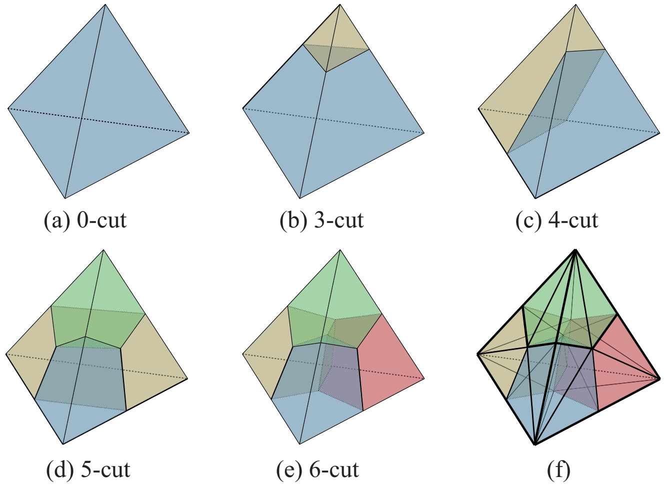

(a) 0-cut
(b) 3-cut
(c) 4-cut
(d) 5-cut
(e) 6-cut
(f)

Fig. 2. (a)∼(e) The five patterns of TSP IDs tagged at a tetrahedron's vertices. (a) Single TSP ID. (b) and (c) 2 TSP IDs. (d) 3 TSP IDs. (e) 4 TSP IDs. (f): the remeshing result of a 6-cut case. Figures courtesy of Bronson et al. [24].

Given the tetrahedral control mesh associated with color information, we reconstruct the video in two steps: mesh subdivision and rasterization. Subdivision aims at generating a dense mesh with smoother color change and more curved geometric shape, so as to enhance the quality of the images produced by rasterization. In this paper, we apply a subdivision method that extends the subdivision scheme [26, 27] from surface to the 3D volume. This subdivision scheme ensures topology consistency of the boundary surfaces of sub-tet-meshes, while preserving color discontinuities across different sub-tet-meshes. To rasterize the subdivided mesh with some specified resolutions, we first find the tetrahedron that a pixel belongs to and compute its color by linearly interpolating the colors at the vertices of the tetrahedron.

IV. Simplification of Tetrahedral Mesh

We first segment the input video into N_T TSPs so that the initial tetrahedral mesh consists of N_T sub-tet-meshes. Each vertex v_i of the video mesh is associated with 6 values, including its three-dimensional coordinate $\mathbf{x} = (x, y, t)$ in the video volume and its RGB color $\mathbf{c} = (r, g, b)$. Then we simplify this initial dense mesh by viewing a video as a 3D manifold in the 6D space, since each vertex v is a 6D vector $[x, y, t, r, g, b]^\top = [\mathbf{x}, \mathbf{c}]^\top$. As introduced in the overview, we perform an edge contraction based mesh simplification method which uses Quadric Error Metrics (QEM) [25, 28] to guide the order of edge contraction. To encourage mesh uniformity, we further extend the original QEM by introducing an additional term. To make our paper self-contained, we introduce the original QEM first and further explain our additional term in the first sub-section.

A. QEM-based Tetrahedral Mesh Simplification

1) QEM: In [28], edge contraction, also called edge collapse, is proposed as the atomic mesh decimation operation. With a contraction $v_j \to v_i$, v_j and all tetrahedra incident to e_{ij} are removed, and v_i will move to a new position v_i^* minimizing a quadratic error function. The quadric error of a vertex is the sum of squared distances from the vertex to the 3D *flats* (i.e affine subspaces) spanned by the tetrahedra that are associated with the vertex. Specifically, suppose that a vertex v is associated with P tetrahedra. Then its quadric error is

$$\Delta(v^*) = \sum_{s\in S(v)} D_s^2(v^*) = \sum_{s\in S(v)} \left(\sum_j n_j^{s\top}(v^* - v)\right)^2 \quad (1)$$

where $S(v)$ is the set of subspaces associated with v, each of which is spanned by one of the P tetrahedra. $D_s(v)$ is the distance from v to subspace s, and the n_j^s are the unit norms of the subspace s. Using homogenous coordinate $\tilde{v}^* = (v^*, 1)$, Eq. 1 can be written as a quadratic form

$$\begin{aligned}\Delta(v^*) &= \sum_{s\in S(v)} \left(\sum_j n_j^{s\top}(v^* - v)\right)^2 \quad (2)\\ &= \sum_{s\in S(v)} \left(\sum_j (n_j^{s\top}, -n_j^{s\top} v)\tilde{v}^*\right)^2 \\ &= \tilde{v}^{*\top}\left(\sum_{s\in S(v)}\sum_j (n_j^{s\top}, -n_j^{s\top} v)^\top (n_j^{s\top}, -n_j^{s\top} v)\right)\tilde{v}^* \\ &= \tilde{v}^{*\top}\mathbf{Q}\tilde{v}^* = \Delta(\tilde{v}^*)\end{aligned}$$

where the quadric error is encoded in the matrix $\mathbf{Q}$. The size of $\mathbf{Q}$ is 7×7 because $\tilde{v}^*$ is a 7D vector.

The optimal contraction position v^* for an edge e_{ij} connecting v_i and v_j is found by minimizing the following objective function, that is

$$\min_v \quad v^\top(\mathbf{Q}_i + \mathbf{Q}_j)v \quad (3)$$

Let $\tilde{v}^* = [x^*, y^*, t^*, r^*, g^*, b^*, 1]^\top$ denote the solution to the above minimization problem. Then the cost of edge contraction of e_{ij} is

$$\text{Cost}(e_{ij}) = \tilde{v}^{*\top}(\mathbf{Q}_i + \mathbf{Q}_j)\tilde{v}^* \quad (4)$$

After contraction, the accumulated quadratic error $\mathbf{Q} = \mathbf{Q}_i + \mathbf{Q}_j$ is inherited by the merged vertex v_i.

2) Extended QEM: To produce high quality approximations, Garland and Zhou [28] use a greedy strategy and maintain a minimal heap keyed on the error cost of edges. In each iteration, the algorithm selects an edge with the minimal cost from the heap and performs edge contraction if this contraction will not cause topology error or tetrahedron flipping, otherwise the algorithm simply ignores it and tries for another one. In practice, when applying the conventional QEM to a region with constant or linear color change, all the QEM error terms become zero so they provide no guidance on how to select edges to contract. As a result, such regions will be over-simplified, causing the entire topology of tetrahedral mesh to fall in a locked state so that edges cannot be contracted any more without flipping tetrahedra. To address this problem with QEM, we introduce a new quadric error terms in our extended QEM framework. Specially, the quadric error defined for a vertex not only includes the original quadric error defined in Eq. 1, but also embodies the squared distance from it to the average position $\bar{\mathbf{x}}$ of the vertices it inherits from iterative edge collapse. That is, if a vertex v moves to v^*, the quadric error is defined as

$$\Delta'(v^*) = (1-\eta)\Delta([\mathbf{x}^*, \mathbf{c}^*]^\top) + \eta\,\|\mathbf{x}^* - \bar{\mathbf{x}}\|_2^2 \quad (5)$$

The introduction of the second term is based on the intuition that when contracting an edge located at a region where color field is near constant or linear, we expect the simplified mesh to be as uniform as possible. The parameter η serves to balance the two terms. In our experiments η is set to 0.1. Clearly, Eq. 5 can be written in a quadratic form. We initialize the error matrix for each vertex before edge contraction as follows,

$$\mathbf{Q}' = \mathbf{Q} + \eta \begin{pmatrix} \mathbf{I}_{3\times3} & \mathbf{O}_{3\times3} & -\mathbf{x} \\ \mathbf{O}_{3\times3} & \mathbf{O}_{3\times3} & \mathbf{O}_{3\times1} \\ -\mathbf{x}^\top & \mathbf{O}_{1\times3} & \mathbf{x}^\top\mathbf{x} \end{pmatrix} \quad (6)$$

To measure the degree of simplification, we define *simplification ratio* as the value of the number of vertices in the simplified control mesh N_c over that in the initial dense mesh N_v. Our approach can simplify a tetrahedral mesh to up to 1% of its original size (see Tab. I). Fig. 3 shows two simplified results for the video CHAMPAGNE, at simplification ratios 1.0% and 0.5% respectively. For the constant colored background regions, our method generates large and uniform tetrahedra and for the feature edges, more tiny tetrahedra exist to characterize the details. With simplification going on, large tetrahedra in the background are processed before those near the features being generated so that the details are faithfully preserved after simplification.

3) Preserving boundaries: There are two kinds of boundaries to be preserved during simplification. One is "*internal boundaries*", i.e. the boundaries shared by multiple sub-tet-meshes, and the other is "*external boundaries*", corresponding to 6 outer faces of the video cuboid in 3D space. The vertices on the *external boundaries* are further categorized into "*face vertex*", "*edge vertex*" and "*corner vertex*". During edge contraction, if two vertices on the edge belong to different categories, care must be taken to where to select the optimal contraction position. We assign a priority to each vertex according to its category to reflect the vertex's feature

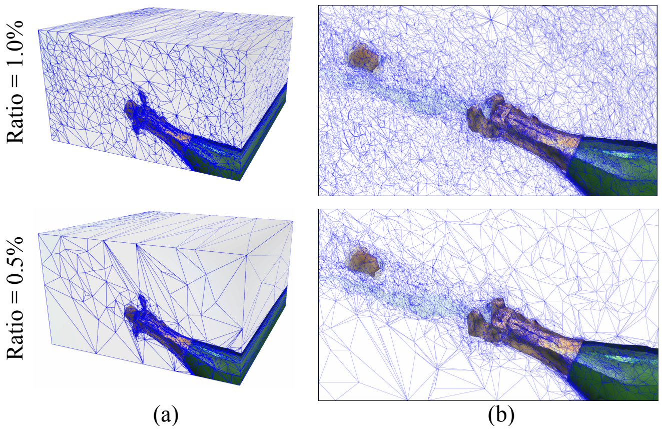


Fig. 3. The simplified results for video CHAMPAGNE, at simplification ratios 1.0% (Row 1) and 0.5% (Row 2). (a) The simplified control meshes for the video. (b) The cross sections at frame 22.

importance, and the optimal position will be determined by the priorities of the two vertices. Specifically, vertex on the *external boundaries* is prior to *non-external*, while in *non-external boundary*, *internal boundary* is prior to *non-internal boundary*. The contraction position tends to favor those vertices with higher priority. We illustrate the vertex categories in Fig. 5 and please refer to the table in our supplementary material for the details of contraction position involving boundary vertices.

B. Boundary Surface Fitting

A sparse control mesh after simplification produced by the previous simplification step might have its internal boundaries deviate significantly from their original positions in the input video. To improve the accuracy of the internal boundaries, we formulate and solved a surface fitting problem by minimizing the following objective function optimize the geometric positions of the vertices on the control mesh as follows,

$$\min_{\mathbf{v}_c} \quad E(\mathbf{v}_c) = E_F(\mathbf{v}_c) + \lambda E_L(\mathbf{v}_c) \tag{7}$$

where $\mathbf{v}_c$ are the vertices of the control mesh, λ a parameter balancing the energy term on fitting error E_F and mesh deformation error term E_L. The fitting error E_F is the sum of squared distances from vertices on the internal boundaries in the densely subdivided mesh and their target positions on the original surfaces,

$$E_F(\mathbf{v}_c) = \sum_{k=1}^{N_s} \left\| v_k^t - v_k \right\|_2^2 = \sum_{k=1}^{N_s} \left\| v_k^t - \alpha_k \mathbf{v}_c \right\|_2^2 \tag{8}$$

where v_k and v_k^t are respectively the position in the subdivided mesh and the target position of a boundary vertex. N_s is the number of such vertices. Every v_k can be substituted as a linear combination α_k of vertices on the control mesh as defined by the subdivision masks. The target position v_k^t is its projection onto the original boundary surface. Minimizing E_F only may cause flipped tetrahedra because the internal vertices remain unchanged relative to those boundary vertices. To solve this issue, we generalize the volumetric graph Laplacian [29] to the tetrahedral mesh and control the mesh deformation by constraining the changes of Laplacian, i.e. the mesh deformation error E_L. The Laplacian coordinate of a vertex encodes the difference between the vertex and a linear combination of its neighboring vertex. E_L is therefore expressed as

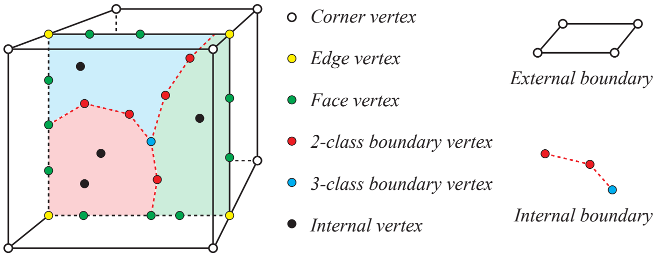


Fig. 5. Vertex categories, marked with separate colors. The tetrahedral mesh for a video is a cuboid, its 6 faces are "external boundaries". The red dashed line segments represent "internal boundaries" shared by multiple sub-tet-meshes (marked as 3 colored regions in the figure).

$$E_L(\mathbf{v}_c) = \sum_{k=1}^{N_c} \left\| \delta_k - \tilde{\delta}_k \right\| = \sum_{k=1}^{N_c} \| l_k \mathbf{v}_c - l_k \tilde{\mathbf{v}}_c \|_2^2 \tag{9}$$

where $\tilde{\delta}_k$ and δ_k are the Laplacian coordinates of v_k before and after optimization separately. N_c denotes vertex number of the control mesh. $\tilde{\mathbf{v}}_c$ represents positions of the vertices before optimization. l_k is a $1 \times N_c$ vector encoding the coefficients to compute δ_k. We set δ_k as

$$\delta_k = v_k - \frac{1}{|\mathcal{N}_k|} \sum_{v_j \in \mathcal{N}_k} v_j \tag{10}$$

where $\mathcal{N}_k$ is the set of neighbors of v_k and $|\mathcal{N}_k|$ is its cardinality.

The solution to Eq. 7 is

$$\mathbf{v}_c = (\mathbf{A}^\top \mathbf{A} + \lambda \mathbf{L}^\top \mathbf{L})^{-1} (\mathbf{A}^\top \mathbf{v}^t + \lambda \mathbf{L}^\top \mathbf{L} \tilde{\mathbf{v}}_c) \tag{11}$$

where $\mathbf{A}$ is a $N_s \times N_c$ matrix with each row being α_k. $\mathbf{L}$ is a $N_c \times N_c$ matrix with each row being l_k. $\mathbf{v}^t$ is a $N_s \times 3$ matrix composed of all v_k^t with $k = 1, 2, ..., N_c$. λ is set to 1.0 in implementation. The energy function 7 can be optimized by iterations to get an optimal and stable solution. We use 5 iterations in our implementation.

C. Color Optimization

The above steps simplify and optimize only the positions of all mesh vertices. After that, we globally optimize the colors of these vertices. We optimize an energy term E_C formulated similar to E_F in Eq. 8, but we use $\mathbf{v}_c$ to encode colors instead of geometric positions, while N_s becomes the total number of vertices in the subdivided mesh, and v_k^t represents the color sampled at the 3D position of the nearest vertex to v_k within the sub-tet-mesh it belongs to. We use the Conjugate Gradient (CG) algorithm to solve a sparse linear system for minimizing this energy.

V. SUBDIVISION OF TETRAHEDRAL MESH

We shall introduce in this section a tetrahedral mesh subdivision technique for reconstructing a raster video from a vectorized video represented as a tetrahedral control mesh. The

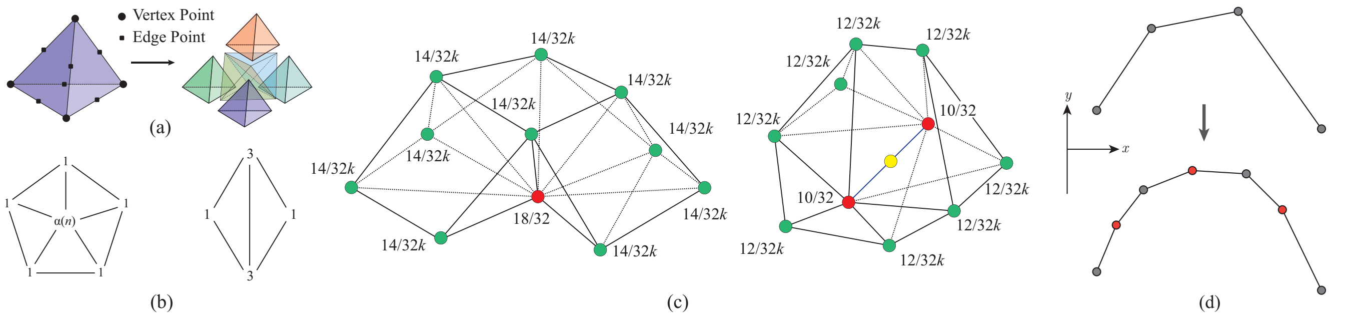
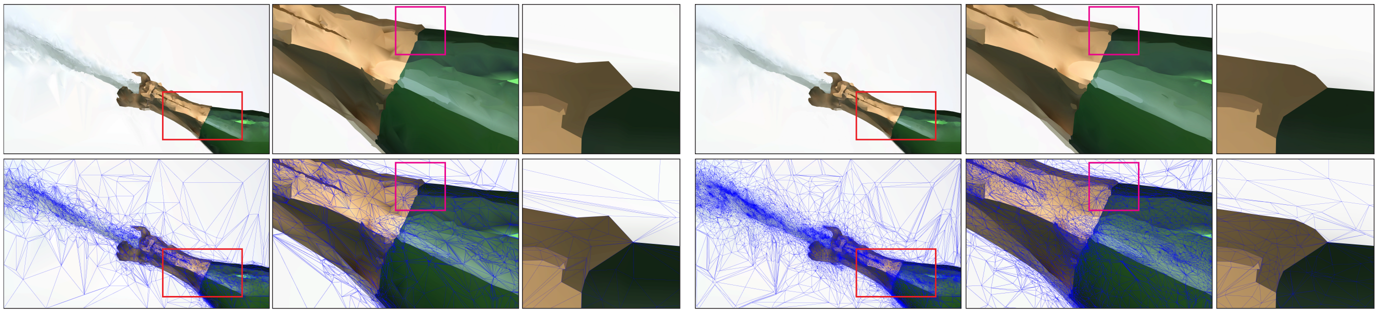


Fig. 4. Illustration of tetrahedral mesh subdivision. (a) A tetrahedron is replaced by 8 tetrahedra. (b) Subdivision masks defined on boundary vertices with left for vertex points and right for edge points. (c) Subdivision masks defined on internal vertices with left for vertex points and right for edge points. (d) A 2D illustration of subdivision. By inserting new vertices (in red) on the edges and updating the positions of the original vertices, the line segments become curved after a subdivision step. If we view y axis as the color, after the subdivision, the color also becomes smooth.



(a) Control mesh (b) Zoom-in view of red boxes in (a) (c) Zoom-in view of pink boxes in (b) (d) Subdivided mesh (e) Zoom-in view of red boxes in (d) (f) Zoom-in view of pink boxes in (e)

Fig. 6. Control mesh vs. subdivided mesh. Cross sections of the two meshes at frame 28 of video CHAMPAGNE. Two rows are visualization of the non-wired and wired results, respectively. By subdividing the mesh, the tetrahedrons become denser in (d) compared with in the control mesh (a). (e) Compared with (b), after subdivision, the color of the rendered frame becomes smooth. (f) Subdivision makes the boundaries more curved compared with those in the control mesh (c), which contains obvious zig-zag artifacts.

simplified control mesh occupies the whole 3D video volume and consists of N_T sub-tet-meshes yielded by the Cleaver algorithm where N_T also denotes the number of TSPs. The 3 color channels of RGB are viewed as 3 scalar fields separately defined on the 3D domain. Since the simplified control mesh is sparse, directly rasterizing such a sparse control mesh will bring two kinds of visual artifacts:

1) Zigzag non-curved boundaries caused by geometrically non-smooth surface between sub-tet-meshes.
2) Non-smooth color transition within each sub-tet-mesh caused by linear interpolation of rasterization.

Fig. 6(c) and (b) illustrate the two types of artifacts respectively, and we apply mesh subdivision to addressing these two issues.

A. Subdivision Method

Subdivision is a powerful and easily implemented algorithm used to smooth meshes. The smoothing procedure is accomplished by inserting new vertices between edges and updating the positions of the original ones. Taking 2D line segments as a toy example (Fig. 4(d)), by inserting new vertices on the edges, the original line segments become curved, where the positions of the newly-inserted vertices are the linear combinations of the original ones. This idea is widely used in generating smooth triangular meshes such as Loop's subdivision algorithm [26], which is recommended to read for a better understanding of this problem. If color is also defined on the vertices, the linear combination will be also applied to it, making the color smoothed as the shape of the mesh.

Our problem involves tetrahedral mesh instead of triangular mesh. So we extend the method in [26] as follows. Let $M = M^0$ represent the simplified control mesh. Our subdivision procedure $M^s \rightarrow M^{s+1}$ works by first computing vertex positions of M^{s+1} as affine combinations of their adjacent vertices in M^s, according to a set of subdivision *masks* defined on vertices and edges so that each vertex in M^{s+1} is either updated by a vertex point or derived from an edge from M^s. Then we replace each tetrahedron by 8 tetrahedra. We use the masks defined in [30] in implementation (see Figs. 4(a) and 4(c))

$$\text{Vertex point:} \qquad \hat{v}_i = \frac{18}{32}v_i + \frac{14}{32k}\sum_{v_j\in\mathcal{N}_i} v_j,$$

$$\text{Edge point:} \qquad \hat{v}_{ij} = \frac{10}{32}(v_i + v_j) + \frac{12}{32k}\sum_{v_l\in\mathcal{N}_{ij}} v_l$$

where the $\hat{v}_i$ and $\hat{v}_{ij}$ are vertex and edge point respectively, $\mathcal{N}_i$ and $\mathcal{N}_{ij}$ the set of neighboring vertices of vertex v_i and the set of edge e_{ij}, respectively, and k is the number of elements in the corresponding set. The above scheme smoothes both the mesh vertices as well as their associated colors. However, the boundary vertices tend to be pulled inward

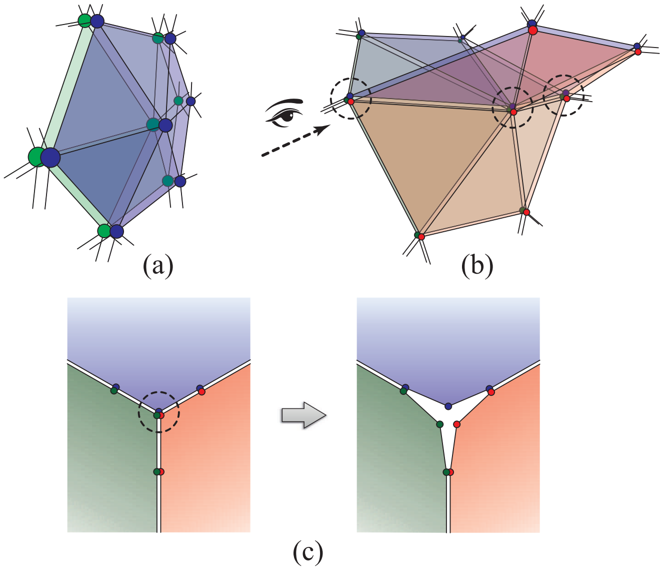


Fig. 7. Subdivision at the boundary surfaces of sub-tet-meshes. Each color represents a sub-tet-mesh and points close to each other denote duplicates of a boundary vertex in its incident sub-tet-meshes. (a) A 2-class boundary vertex (the middle one), the adjacent relations of the two sub-tet-meshes (the navy and green) are the same. (b) 3 sub-tet-meshes meet and result in 3-class boundary vertices (marked by dashed circles). (c) The interfaces seen from the view angle as in (b). For a 3-class vertex (the middle one), its 3 duplicates (navy, green, red) will update the positions with the affine combination of the neighbors in their own sub-tet-meshes, yielding an inconsistent position and a "gap" among the sub-tet-meshes (right).

by the internal ones if we apply the masks in [30] to both kinds of vertices indiscriminately. This will cause each sub-tet-mesh to shrink and finally lead to "gaps" between neighboring sub-tet-meshes. To overcome this problem, we apply Loop's subdivision masks [26] to the boundary vertices instead so that positions of the boundary vertices are updated independent of the internal ones during subdivision.

B. Discontinuity Preservation at Boundaries

With the above subdivision method, at the boundaries of sub-tet-meshes the color will be non-smooth, though still continuous, since the boundary vertices are often shared by multiple sub-tet-meshes. To model *color discontinuities* across sub-tet-meshes, we further explicitly split the tetrahedral mesh at boundary surfaces of all the sub-tet-meshes. Specifically, we create K_i duplicates (including the original one) for each boundary vertex v_i^B that is shared by K_i sub-tet-meshes, and assign duplicate to each of its incident sub-tet-meshes. To make all the duplicates of v_i^B always meet at a common position, we impose the constraint for K-class boundary vertices ($K > 2$) that their positions should keep unchanged during subdivision. On the contrary, for a K-class boundary vertex with $K \leq 2$, the positions can be updated independently, since the adjacent relations are the same at each side of the sub-tet-meshes (Fig. 7(a)).

Fig. 6 shows the cross sections of a control mesh and its subdivision mesh at frame 28 of video CHAMPAGNE. By comparing the control mesh with its subdivided mesh, we can see the boundaries and color within all the regions become smooth (Fig. 6(e)(f)) while the color discontinuity at the boundaries is still faithfully preserved.

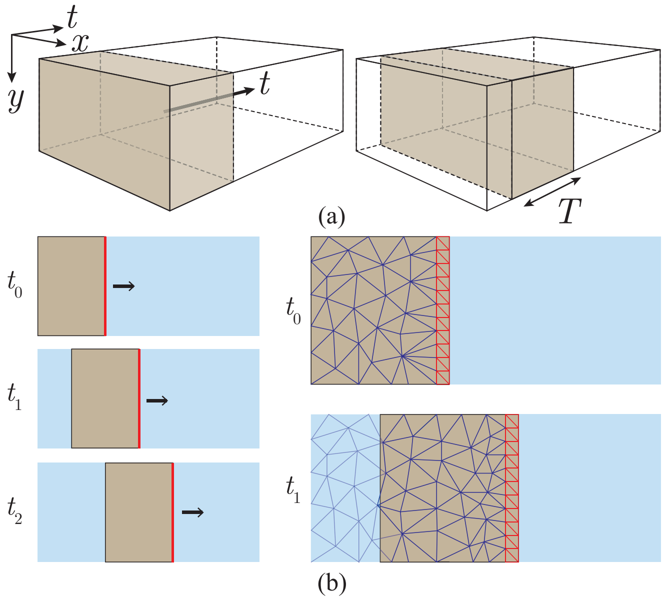


Fig. 8. (a) The 3D view of the sliding window in a video, moving along time axis. Red line represents the sliding boundary. (b) left: the 2D side view of the sliding window in 3 time steps. right: the illustration of the simplification in each sliding window. At t_0, the tetrahedrons at the sliding boundary (red) is fixed. With the sliding window moving half of the window size, in t_1 they are simplified while the tetrahedrons at t_1's sliding boundary is still fixed.

VI. STREAMING PROCESSING OF TETRAHEDRAL MESH

The method introduced above is valid but requires the whole video data to be entirely loaded into memory before simplification and subdivision. Obviously with limited memory resources, this strategy can handle several frames only, which is far from what we expect in practise. For this sake, we develop a strategy of streaming processing to solve this issue.

The strategy works as follows. First, the video data can be viewed as a volumetric shape in 3D, as illustrated in Figs. 3(a) and 8(a). In the box-shaped video data, we introduce a sliding window where our algorithm locally generates a small part of the mesh every time. The sliding window is designed to move along the time axis, and its length is specified as T (Fig. 8(a)). With the window sliding through the entire data, the final mesh is generated and the memory cost can be limited to be proportional to the window size, which makes it more controllable and flexible for video processing.

However, directly simplifying the mesh within each sliding window independently cannot preserve the topology consistency across the windows, so the *sliding boundary*, the boundary surface in the time direction (red lines in Fig. 8(b)) has to be carefully handled to keep consistency. Our strategy is that at current position t_{cur}, the tetrahedrons on the sliding boundary is fixed, considering the lack of neighbor information in next sliding window. We then move the sliding window forward for half of the window size. After the sliding window moves forward, the tetrahedrons on the last sliding boundary change to internal ones so that they can be simplified properly. The procedure runs iteratively until the sliding window moves to the end of the sequence. Fig. 8(b) illustrates the procedure. In our experiments, we found setting T to 10 brings us the vectorized videos with smooth frame transition and peak memory cost kept lower than 8GB. A group of the detailed memory cost data is shown in Table I.

VII. Experimental Results

We have implemented our video vectorization algorithm and tested it on a variety of raster videos. Our results can be found in Figs. 9, 10, 11, 12 and 15, as well as the accompanying video[1].

We first choose two synthesized videos to test our algorithm. Fig. 10 shows two video clips from TV ads, which contain sharp visual features, considerable object motions, as well as details. For example in the video iPhone, the dark spot near the flash is so tiny making it easily lost when the video is represented as a sparse mesh. While in the video Chocolate, frequent motion of salient edges makes it difficult to generate a compact, yet still accurate vectorized video. However, in our reconstructed video these features are faithfully preserved and frame transition is smooth. In terms of the visual quality of individual frame, our method also produces reasonably good results, which can be compared with those generated by image vectorization method [5].

In Figs. 9 and 11, we show the results for two videos recording the process of flower blooming. Unlike the two synthesized videos above, both videos contain many irregular motions of the foreground objects as well as occlusion and visibility change. The two factors indeed make it more challenging for us to simplify the tetrahedral mesh and to keep the temporal coherence. Although the simplification ratio is set to 1.0%, our approach still produces faithfully reconstructed videos without apparent visual artifacts. Since it is resolution-independent, a vectorized video can be magnified at arbitrary resolution. To illustrate the sparsity of the control mesh, we visualize the cross sections of the mesh at the corresponding frame positions. Large tetrahedra are generated by our approach to represent those flat background, while more detailed tetrahedra are produced near the region containing substantial color variation which are used to characterize local details, e.g. the flower cores, the boundaries of petals and stems, etc. Some zoom-in views of the cross sections of the subdivided meshes are also illustrated for better visualization.

A. Comparison

1) Temporal Coherence: To show the importance of frame coherence, we first compare our method with a state-of-the-art image vectorization method [5], which is applied to each video frame individually, frame by frame. Such an image vectorization method is expected to produce higher quality result, but when applying it to each of the video frames, we have found that the reconstructed video is unstable and annoying visual artifacts like flicker present. This is because applying the image vectorization method to each video frame directly cannot preserve the spatio-temporal coherence of the video, even after a carefully parameter is tuned. For example, for the video iPhone in Fig. 10, frame 37 in Liao et al.'s results is rather good as the edges around the camera is sharp and smooth; while in frame 86, some edges and a small black spot are lost. Such visual features, however, are well preserved in our results.

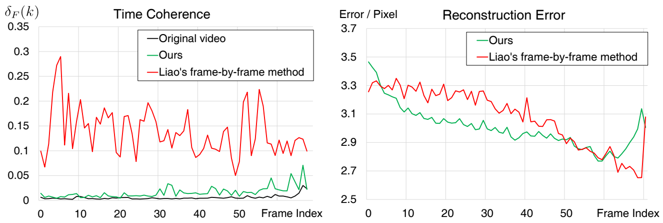


Fig. 14. $\delta_F(k)$ and reconstruction error, for video Mountain.

In order to weaken the influence of complex motion to the results, in Fig. 14 we show the results for a video sequence with smooth motion, Mountain. In the video the cloud moves slowly, while the mountain is almost static in the distance. Even with such relatively simple settings, the frame-by-frame method [5] produces unstable results with flickering, as the accompanying video shows. In contrast, our method yields very stable results and sharp features are well preserved, with a 2.0% simplification ratio. To quantitatively characterize the temporal coherence, we compute the change of optical flow throughout the entire video sequence. Specifically, we first accumulate the summed square distance $\delta_F(k)$ between the optical flow maps of every two successive frames of the video, f_k and f_{k+1}. The sum further regularized by the total number of video pixels is taken as the measure of temporal smoothness for the entire video frames, noted as Δ_F.

Intuitively, severe flickering will lead to a high Δ_F since optical flow will change frequently. To visualize the results, Fig. 14 plots $\delta_F(k)$ for every two successive optical flows on the results of Mountain by our and Liao et al.'s method, comparing with the original video. We can see our result is as stable as the input video.

2) Reconstruction Error: We further evaluated the reconstruction quality in terms of per-pixel reconstruction error, which is also used by previous image vectorization methods. The results show that our method achieves lower reconstruction error compared to [5] with the same simplification ratio. In Fig. 14 right, we plot the reconstruction error for each frame of the video Mountain. The simplification ratio in this example is set to 2.0% for both methods. The curve illustrates that for most frames our method yields lower reconstruction error than Liao et al.'s approach, which also means more accurate reconstruction. Fig. 12 also reveals some details in the results of both methods. Although the highly textured regions like the lawn become blurry that are beyond the capability of vector representation, our method still preserves those sharp features. For example, our rasterized video faithfully reconstructs the mountain ridge. The overall reconstruction errors are 3.00/pixel and 3.07/pixel for our and Liao et al.'s results respectively.

We also analyzed how the simplification ratio affects the

[1] You may also see it from the following link: https://goo.gl/mg3hxN

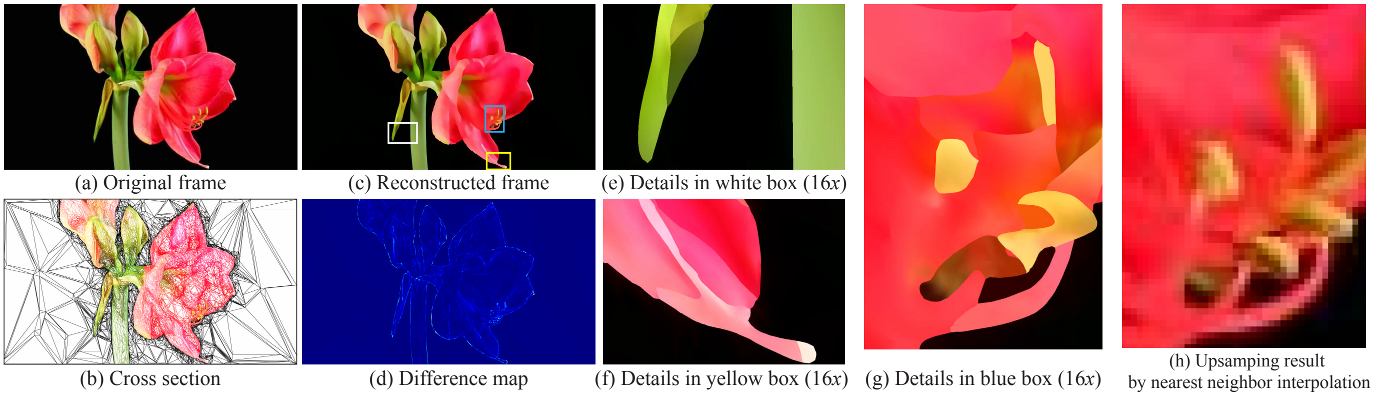
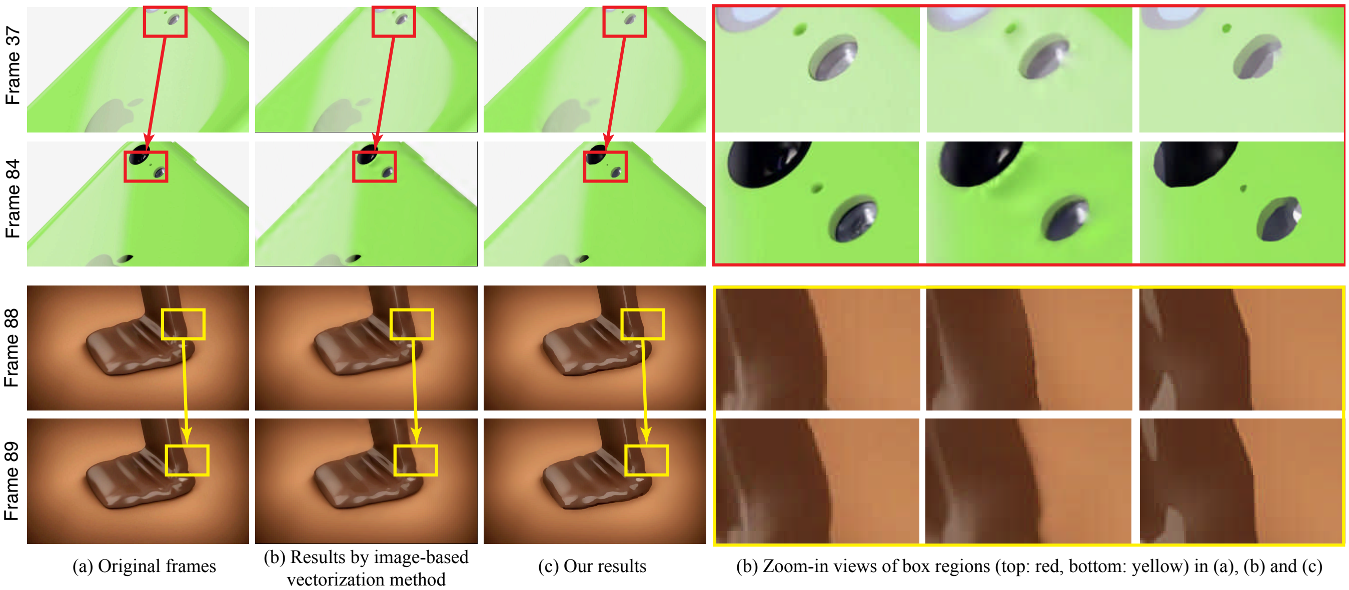

(a) Original frame
(c) Reconstructed frame
(e) Details in white box (16x)
(b) Cross section
(d) Difference map
(f) Details in yellow box (16x)
(g) Details in blue box (16x)
(h) Upsamping result by nearest neighbor interpolation

Fig. 9. Vectorization of the video Blooming I, at frame 67. (a) The original frame. (b) The cross section of the simplified control meshes at the same position of frame 67. (c) The reconstructed frame from the vectorized video representation. (d) Difference map between the reconstructed frame and original frame. Figures (e), (f), and (g) are the zoomed-in views of the box regions in (c), magnified at 16× resolutions. (h) The upsampling result of the region in the original frame, corresponding to the blue box in (c), by nearest neighbor interpolation (16×). Please refer to the accompanying video for this and the other results in this paper.



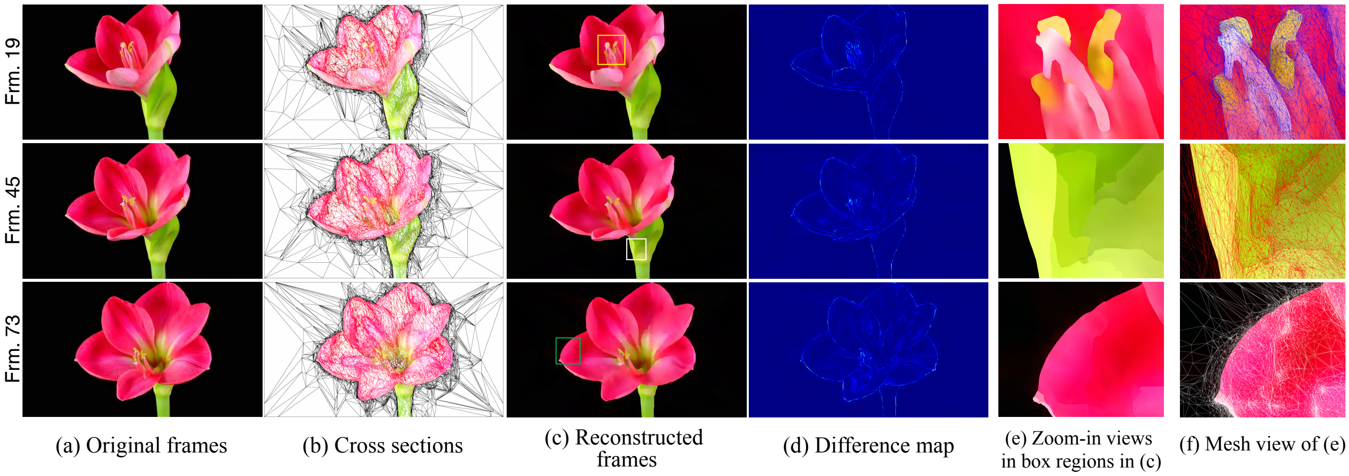
(a) Original frames
(b) Results by image-based vectorization method
(c) Our results
(b) Zoom-in views of box regions (top: red, bottom: yellow) in (a), (b) and (c)

Fig. 10. Reconstructed frames in video iPhone and Chocolate, grouped in top and bottom 2 rows. In each group, each row shows results of one frame. Columns from left to right: (a) The original frames. (b) Image-based vectorization results by [5]. (c) Our results. (d) is zoom-in views of the regions circled by the same color boxes in (a), (b) and (c). Results show that our method faithfully reconstructs the details and more importantly ensures the temporal coherence. Taking the iPhone video as an example, from Frame 37 to 84, the dark spot near the flash is lost by the image vectorization method although the same parameter is set for both frames. But it remains in our results (red boxes). Likewise, our method also produces consistent vectorized details for the main edges in the video Chocolate, while blurry edge is produced in Frame 89 by Liao et al.'s method (yellow boxes).

reconstructed image quality with the video Mountain shown in Fig. 13, with ratio settings 2%, 5%, 10% and 20%. We found that with higher ratio values, more texture details became visible, e.g. the edges in the cloud. We also computed the reconstruction errors for these settings, i.e. 3.00/pixel to 2.23/pixel for 2% to 20%, demonstrating the image quality is better preserved with more tetrahedrons.

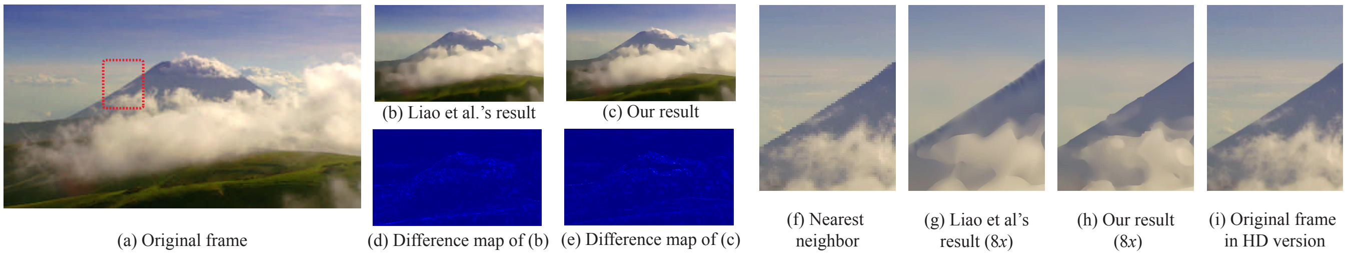
Fig. 15 shows results for three additional real videos and one animation, Dog. The reconstruction error of our results is typically around 2.0/pixel, and simplification ratios are set no larger than 2.0% except for the animation containing large motions in the foreground (the dog) and textured background, which is 5.0% instead. All our results look smooth without flicking artifacts. We list the statistics data in Table I for detailed references.

3) Computational Cost and Storage: Our algorithm runs in a streaming mechanism so that the memory required for the initial mesh can be controlled. The length of the sliding window T was set to 10 in our experiments. With this setting, our algorithm typically costs less than 8GB RAM and takes ≈ 60 minutes for a video of size $640 \times 360 \times 70$ on an Intel Core i7 @3.40 GHz processor, in comparison with ≈ 1GB RAM and ≈ 40 seconds for a single image of the same size by image vectorization [5]. In terms of computational cost for vectorizing a video, while our method does not reveal its outperformance compared with image vectorization which



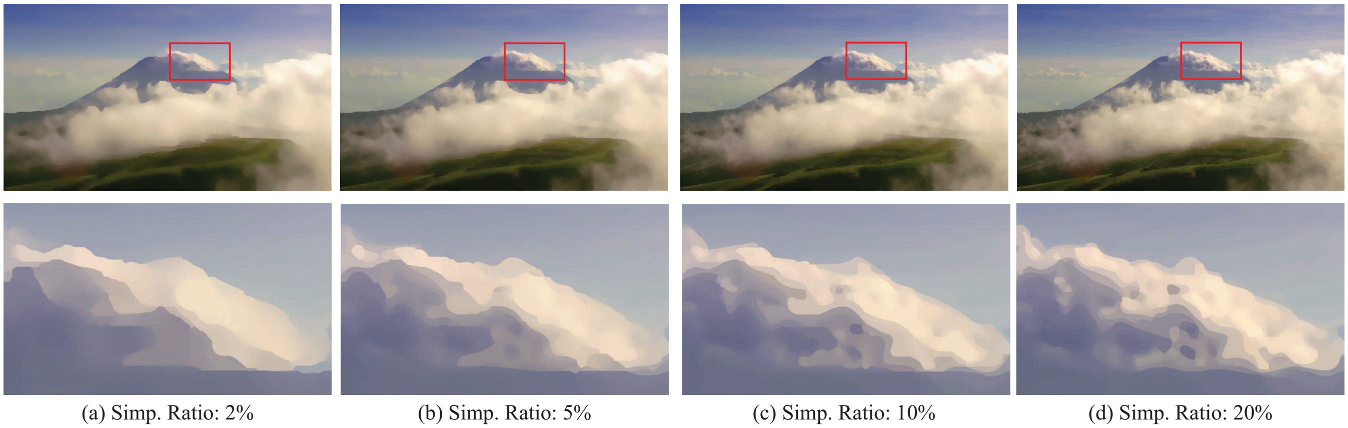
(a) Original frames (b) Cross sections (c) Reconstructed frames (d) Difference map (e) Zoom-in views in box regions in (c) (f) Mesh view of (e)

Fig. 11. Results of video BLOOMING II at frames 19, 45 and 73 (Rows 1∼3). The original video is of size 640 × 360 × 95. (a) Original frames. (b) The cross sections of the control tetrahedral meshes at corresponding frames. (c) Our reconstructed frames (d) The difference maps. (e) and (f) are the magnified details (16×) of the box regions shown in (c) and their corresponding cross sections of the subdivided meshes.



(a) Original frame (b) Liao et al.'s result (c) Our result (d) Difference map of (b) (e) Difference map of (c) (f) Nearest neighbor (g) Liao et al's result (8*x*) (h) Our result (8*x*) (i) Original frame in HD version

Fig. 12. Results of video MOUNTAIN. (a) Original frame. (b)(c) Reconstructed frames by Liao et al's and our methods. (d)(e) Difference maps of (b)(c) respectively. (f)(g) and (h) are zoom-in views of the red box regions in (a), by nearest neighbor interpolation, Liao et al's and our methods respectively, rastered at 8× resolution. (i) is the red box region in the HD-version original frame. Compared with (i), our result (h) preserves the sharp feature at the mountain ridge.



(a) Simp. Ratio: 2% (b) Simp. Ratio: 5% (c) Simp. Ratio: 10% (d) Simp. Ratio: 20%

Fig. 13. Reconstructed image quality in various simplification ratios. (a) ∼ (d): reconstructed frame 32 of video MOUNTAIN by simplification ratios 2% to 20%, with reconstructed errors being 3.005/pixel, 2.827/pixel, 2.615/pixel and 2.231/pixel. In the zoom-in views (the second row), more texture details such as the edges in the cloud become visible with higher ratio settings.

does not need to concern continuity of vectorized results across frames. Considering our algorithm handles not only frames but also inter-frame connections, we believe our method is still comparable with theirs because on average each frame costs 0.8GB RAM and ≈ 50 seconds.

For the amount of storage required, we applied the streaming compression algorithm specifically designed for tetrahedral volume meshes [31] to our generated tetrahedral mesh. We also achieved comparable results to those produced by applying zip compression to the triangular meshes of the video frames generated by Liao et al's method. We show the time, memory cost and storage in Table I.

B. Limitations

We have shown that our method can deal with a wide variety of videos. One limitation for vectorizing a video is that it

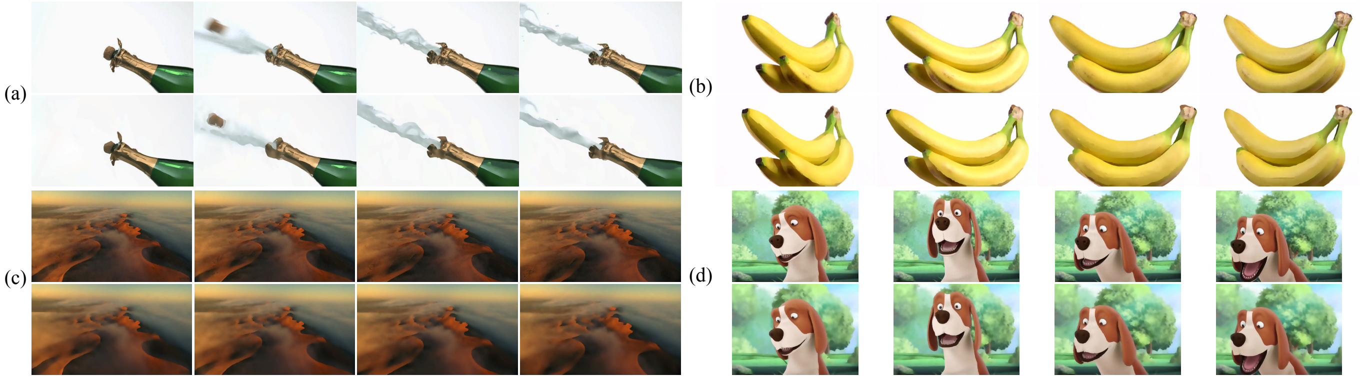


Fig. 15. Reconstructed frames from our vector video representation for four videos. (a) ∼ (d) CHAMPAGNE, BANANA, DESERT and DOG. The odd rows show the original frames and even rows show the corresponding frames from our reconstructed videos. See our accompanying video for more details.

	Resolution	Ours						Liao et al.'s				
		S.R.	R.E.	Time	RAM	RAM/frame	Storage	S.R.	R.E.	Time	RAM	Storage
BLOOMING I	640×360×70	1.0%	2.163	3,442s	7.25GB	0.73GB	2.51MB	1.0%	2.936	2,741s	1.12GB	2.36MB
BLOOMING II	640×360×95	1.1%	2.862	4,548s	7.18GB	0.72GB	3.73MB	1.5%	3.042	3,691s	1.11GB	3.25MB
DOG	373×270×57	5.0%	4.293	1,408s	3.57GB	0.36GB	3.70MB	5.1%	5.014	1,345s	0.43GB	4.12MB
IPHONE	480×270×100	0.5%	1.325	2,698s	4.34GB	0.43GB	813KB	1.1%	1.379	1,987s	0.52GB	876KB
CHAMPAGNE	480×270×100	0.5%	2.914	2,454s	4.52GB	0.45GB	933KB	1.0%	3.899	2,198s	0.57GB	1.02MB
CHOCOLATE	448×252×100	1.0%	1.523	2,897s	3.48GB	0.35GB	1.48MB	1.0%	1.571	1,996s	0.43GB	1.62MB
MOUNTAIN	406×231×70	2.0%	3.005	1,775s	2.87GB	0.29GB	1.88MB	2.0%	3.079	1,057s	0.46GB	1.92MB
DESERT	508×288×80	1.0%	2.637	2,435s	5.10GB	0.51GB	1.62MB	1.1%	3.047	1,724s	0.67GB	1.87MB
BANANA	450×256×160	2.0%	2.866	3,492s	2.56GB	0.26GB	5.20MB	2.0%	2.928	3,061s	0.57GB	5.04MB

TABLE I
PERFORMANCE COMPARISON: OURS VS. LIAO ET AL.'S METHOD. S.R. AND R.E. ARE SHORT FOR SIMPLIFICATION RATIO AND PER PIXEL RECONSTRUCTION ERROR, RESPECTIVELY. NOTE THAT RAM FOR OUR METHOD WAS TESTED WITH THE SLIDING WINDOW $T = 10$, SO WE ALSO SHOW THE PER FRAME COST IN COLUMN RAM/FRAME. FOR THE AMOUNT OF STORAGE OF LIAO ET AL.'S METHOD, IT IS EVALUATED UNDER THE SAME SIMPLIFICATION RATIOS SPECIFIED IN OUR METHOD.

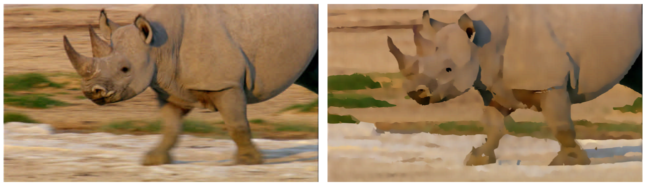


Fig. 16. Failure case caused by rapid motion in the video RHINO. Frame No. 14 of the original frame (left) and our constructed result (right). In this example, the control mesh is simplified to a ratio of 5.0%. From AFRICA (2013) (©BBC).

may not be suitable to represent video objects with fine and rapidly changing details, as image vectorization methods are vulnerable to highly detailed image regions such as textures. For such cases, the reconstruction error will be relatively large. For example for a video of a running rhino shown in Fig. 16, the segmented TSPs contain sharp color change because of the rapid motion. This leads to color mix-up across sharp features.

VIII. CONCLUSIONS

We have presented a vector-based video representation and its associated video vectorization and reconstruction methods. The core of our method is the simplification and subdivision of tetrahedral meshes defined over the spatial-temporal video volume. Our vector-based video representation offers the benefits usually found in vector graphics, such as compactness and scalability, as indicated by our experiments and comparisons. Since editability is another advantage of vector graphics compared with raster graphics, as future work we plan to explore the applications such as shape editing and color editing that will benefit from the editability of our video vector representation.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their constructive comments which helped improve this paper greatly. This work was supported in part by the National Natural Science Foundation of China under Grants 61373059, 61672279, and 61321491 and the Natural Science Foundation of Jiangsu Province under Grants BK20150016.

REFERENCES

[1] J. Sun, L. Liang, F. Wen, and H.-Y. Shum, "Image vectorization using optimized gradient meshes," in *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3. ACM, 2007, p. 11.

[2] Y.-K. Lai, S.-M. Hu, and R. R. Martin, "Automatic and topology-preserving gradient mesh generation for image vectorization," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3. ACM, 2009, p. 85.

[3] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin, "Diffusion curves: A vector representation for smooth-shaded images," in *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, vol. 27, 2008. [Online]. Available: http://maverick.inria.fr/Publications/2008/OBWBTS08

[4] T. Xia, B. Liao, and Y. Yu, "Patch-based image vectorization with automatic curvilinear feature alignment," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5. ACM, 2009, p. 115.

[5] Z. Liao, H. Hoppe, D. Forsyth, and Y. Yu, "A subdivision-based representation for vector image editing," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 11, pp. 1858–1867, 2012.

[6] J. J. Zou and H. Yan, "Cartoon image vectorization based on shape subdivision," in *Computer Graphics International 2001. Proceedings*. IEEE, 2001, pp. 225–231.

[7] R. D. Janssen and A. M. Vossepoel, "Adaptive vectorization of line drawing images," *Computer vision and image understanding*, vol. 65, no. 1, pp. 38–56, 1997.

[8] X. Hilaire and K. Tombre, "Robust and accurate vectorization of line drawings," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 6, pp. 890–904, 2006.

[9] G. Lecot and B. Lévy, "Ardeco: Automatic region detection and conversion," in *Proceedings of the 17th Eurographics conference on Rendering Techniques*. Eurographics Association, 2006, pp. 349–360.

[10] B. Price and W. Barrett, "Object-based vectorization for interactive image editing," *The Visual Computer*, vol. 22, no. 9-11, pp. 661–670, 2006.

[11] G. Xie, X. Sun, X. Tong, and D. Nowrouzezahrai, "Hierarchical diffusion curves for accurate automatic image vectorization," *ACM Trans. Graph.*, vol. 33, no. 6, pp. 230:1–230:11, 2014.

[12] D. Sỳkora, J. Buriánek, and J. Žára, "Sketching cartoons by example," in *Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2005, pp. 27–34.

[13] D. Sykora, J. Burianek, and J. Zara, "Video codec for classical cartoon animations with hardware accelerated playback," in *Advances in Visual Computing*. Springer, 2005, pp. 43–50.

[14] S.-H. Zhang, T. Chen, Y.-F. Zhang, S.-M. Hu, and R. R. Martin, "Vectorizing cartoon animations," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 4, pp. 618–629, 2009.

[15] L. Wang, K. Zhou, Y. Yu, and B. Guo, "Vector solid textures," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, p. 86, 2010.

[16] L. Wang, Y. Yu, K. Zhou, and B. Guo, "Multiscale vector volumes," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6, p. 167, 2011.

[17] X. Sun, G. Xie, Y. Dong, S. Lin, W. Xu, W. Wang, X. Tong, and B. Guo, "Diffusion curve textures for resolution independent texture mapping." *ACM Trans. Graph.*, vol. 31, no. 4, p. 74, 2012.

[18] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[19] M. Grundmann, V. Kwatra, M. Han, and I. Essa, "Efficient hierarchical graph-based video segmentation," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2141–2148.

[20] J. Chang, D. Wei, and J. W. Fisher, III, "A video representation using temporal superpixels," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[21] C. Wang, Y. Guo, J. Zhu, L. Wang, and W. Wang, "Video object co-segmentation via subspace clustering and quadratic pseudo-boolean optimization in an mrf framework," *IEEE Transactions on Multimedia*, vol. 16, no. 4, pp. 903–916, 2014.

[22] S. Paris, "Edge-preserving smoothing and mean-shift segmentation of video streams," in *European Conference on Computer Vision*. Springer, 2008, pp. 460–473.

[23] J. Clement, P. Sylvain, and M. Teillaud, "3D Triangulations, C-GAL, Computational Geometry Algorithms Library," https://doc.cgal.org/latest/Triangulation_3/.

[24] J. R. Bronson, J. A. Levine, and R. T. Whitaker, "Lattice cleaving: Conforming tetrahedral meshes of multimaterial domains with bounded quality," in *Proceedings of the 21st International Meshing Roundtable*. Springer, 2013, pp. 191–209.

[25] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216.

[26] C. Loop, "Smooth subdivision surfaces based on triangles," 1987.

[27] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle, "Piecewise smooth surface reconstruction," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, 1994, pp. 295–302.

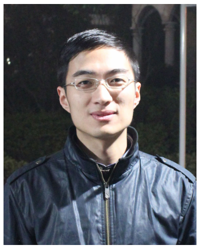
[28] M. Garland and Y. Zhou, "Quadric-based simplification in any dimension," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 2, pp. 209–239, 2005.

[29] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum, "Large mesh deformation using the volumetric graph laplacian," in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 496–503.

[30] Y.-S. Chang, K. T. McDonnell, and H. Qin, "A new solid subdivision scheme based on box splines," in *Proceedings of the seventh ACM symposium on Solid modeling and applications*. ACM, 2002, pp. 226–233.

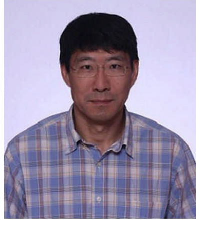
[31] S. Gumhold, S. Guthe, and W. Straßer, "Tetrahedral mesh compression with the cut-border machine," in *Proceedings of the conference on Visualization'99: celebrating ten years*. IEEE Computer Society Press, 1999, pp. 51–58.

Chuan Wang received his Ph.D degree from The University of Hong Kong in 2015, and B.Eng degree from University of Science and Technology of China in 2010. He is currently a computer vision staff researcher in Lenovo Group Limited, Hong Kong. He worked as a visiting scholar in National Laboratory of Pattern Recognition, Chinese Academy of Sciences, Beijing, China in 2009 and State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou, China in 2010. His research interests include video analysis and computer vision.



Jie Zhu received the B.Eng degree from Jiangnan University, Wuxi, China in 2012. He is now a master candidate in the Department of Computer Science and Technology at Nanjing University. He worked as a visiting scholar in The University of Hong Kong in 2013. His research interests include computer vision and computer graphics.

Yanwen Guo received the Ph.D degree in applied mathematics from the State Key Lab of CAD & CG, Zhejiang University, China, in 2006. He is currently an associate professor at the National Key Lab for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Jiangsu, China. He was a visiting scholar in the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign in 2013. His research interests include image and video processing, vision, and computer graphics.



Wenping Wang received the Ph.D degree from the University of Alberta, Edmonton, Canada. He is a professor and the department head of the Department of Computer Science, The University of Hong Kong. His research interests include computer graphics, visualization, and geometric computing. He is the program cochair of several international conferences, including Conference on Shape Modeling (SMI'09), and the conference chair of Pacific Graphics 2012 and SIGGRAPH Asia 2013 etc. He is an IEEE Fellow.