Feature-Aware Uniform Tessellations on Video Manifold for Content-Sensitive Supervoxels

Ran Yi, Zipeng Ye, Wang Zhao, Minjing Yu, Yu-Kun Lai, Yong-Jin Liu, *Senior Member, IEEE*

Abstract—Over-segmenting a video into supervoxels has strong potential to reduce the complexity of downstream computer vision applications. Content-sensitive supervoxels (CSSs) are typically smaller in content-dense regions (i.e., with high variation of appearance and/or motion) and larger in content-sparse regions. In this paper, we propose to compute feature-aware CSSs (FCSSs) that are regularly shaped 3D primitive volumes well aligned with local object/region/motion boundaries in video. To compute FCSSs, we map a video to a 3-dimensional manifold embedded in a combined color and spatiotemporal space, in which the volume elements of video manifold give a good measure of the video content density. Then any uniform tessellation on video manifold can induce CSS in the video. Our idea is that among all possible uniform tessellations on the video manifold, FCSS finds one whose cell boundaries well align with local video boundaries. To achieve this goal, we propose a novel restricted centroidal Voronoi tessellation method that simultaneously minimizes the tessellation energy (leading to uniform cells in the tessellation) and maximizes the average boundary distance (leading to good local feature alignment). Theoretically our method has an optimal competitive ratio $O(1)$, and its time and space complexities are $O(NK)$ and $O(N+K)$ for computing K supervoxels in an N-voxel video. We also present a simple extension of FCSS to streaming FCSS for processing long videos that cannot be loaded into main memory at once. We evaluate FCSS, streaming FCSS and ten representative supervoxel methods on four video datasets and two novel video applications. The results show that our method simultaneously achieves state-of-the-art performance with respect to various evaluation criteria.

Index Terms—Supervoxels, video over-segmentation, video manifold, low-level video features, centroidal Voronoi tessellation.

1 Introduction

Supervoxels are perceptually meaningful atomic regions obtained by grouping similar voxels (i.e., exhibiting coherence in both appearance and motion) in the spatiotemporal domain. As a special over-segmentation of videos, supervoxels well preserve the structural content while still providing sufficient levels of detail. Therefore, supervoxels can greatly reduce the computational complexity and have been widely used as a preprocessing step in many computer vision applications, such as video segmentation [13], [19], [45], propagation of foreground object segmentation [14], spatiotemporal object detection [27], spatiotemporal closure in videos [17], action segmentation and recognition [15], [25], and many others.

Many methods have been proposed for computing supervoxels, including energy minimization by graph cut [38], non-parametric feature-space analysis [28], graph-based merging [9], [13], [42], contour-evolving optimization [17], [21], [31], optimization of normalized cuts [33], [7], generative probabilistic framework [5] and hybrid clustering [30], [43], etc. These methods can be classified according to different representation formats: (1) temporal superpixels [5], [4], [17], [21], [30], [31], [39]: supervoxels are represented in each frame and their labels are temporally consistent in adjacent frames, and (2) supervoxels [7], [9], [13], [28], [33], [38], [42], [43]: they are 3D primitive volumes whose union forms the video volume. Note that these two representations can be transferred to each other. For example, temporal superpixels can be stacked up frame-by-frame to reconstruct supervoxels. However, individual supevoxels obtained in this way may have disconnected components or have a complex topology type (i.e., having a non-zero genus). On the other hand, a supervoxel can be sliced by related frames to decompose it into temporal superpixels, however, a superpixel sliced in a frame may also consist of disjoint components or have a complex topology type.

Depending on the size of video data, supervoxel methods can also be classified into *off-line* and *streaming* methods. Off-line methods require the video to be short enough such that all video data can be loaded into the memory. On the other hand, streaming methods do not have such a limitation on the video length, i.e., video data is accessed sequentially in blocks and each time only a block is needed to feed into the memory. In a recent survey [41], seven representative supervoxel methods are selected, including five off-line [9], [10], [28], [7], [13] and two streaming [42], [5] methods, to represent the state of the art.

To measure the quality of supervoxels, the following principles have been considered in previous work [13], [22], [38], [41], [24], [44]: (1) *Feature preservation*: supervoxel boundaries align well with object/region/motion boundaries in a video; (2) *Spatiotemporal uniformity*: in non-feature regions, supervoxels are uniform and regular in the spatiotemporal domain; (3) *Performance*: computing supervox-

- *R. Yi, Z, Ye, W. Zhao and Y.-J. Liu are with BNRist, MOE-Key Laboratory of Pervasive Computing, the Department of Computer Science and Technology, Tsinghua University, Beijing, China. R. Yi and Z. Ye are joint first authors. E-mail: {yr16@mails., yezp17@mails., zhao-w19@mails., liuyongjin@}tsinghua.edu.cn*
- *M. Yu is with College of Intelligence and Computing, Tianjin University, Tianjin, China. E-mail: minjingyu@tju.edu.cn*
- *Y.-K. Lai is with School of Computer Science and Informatics, Cardiff University, UK. E-mail: LaiY4@cardiff.ac.uk*
- *M. Yu and Y.-J. Liu are corresponding authors.*

This work was supported by the Natural Science Foundation of China (61725204, 61521002).

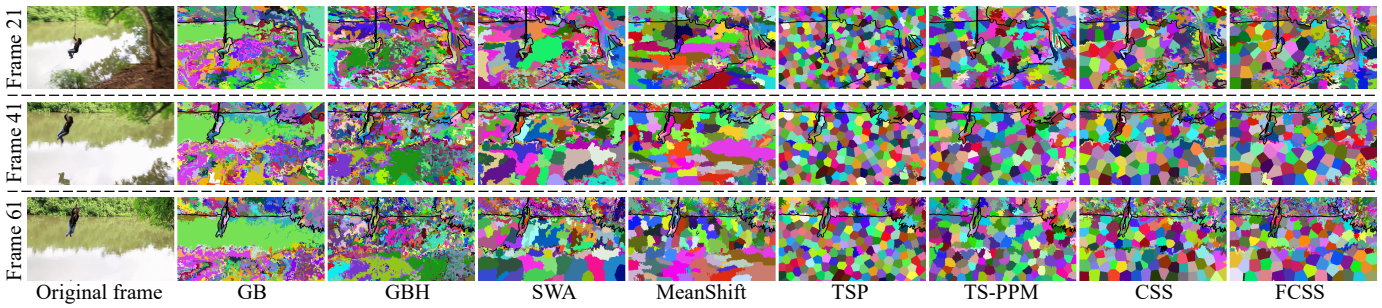


Fig. 1. Superpixels (induced by clipping supervoxels on frames #21, #41 and #61) obtained by GB [9], GBH [13], SWA [32], [33], [7], MeanShift [28], TSP [5], TS-PPM [16], CSS [43] and our FCSS. All methods generate approximately 1,500 supervoxels. TSP, TS-PPM, CSS and FCSS produce regular supervoxels (and accordingly regular clipped superpixels), while other methods produce highly irregular supervoxels. As shown in Section 6, these four methods are insensitive to supervoxel relabeling and achieve a good balance among commonly used quality metrics pertaining to supervoxels, including UE3D, SA3D, BRD and EV, while FCSS runs $5\times$ to $10\times$ faster than TSP, and the peak memory required by FCSS is $22\times$ smaller than TSP and $7\times$ to $15\times$ smaller than TS-PPM. Both FCSS and CSS generate more supervoxels in content-rich areas (e.g., bushes on the lake shore) and fewer supervoxels in content-sparse areas (e.g., lake surface), while FCSS better captures low-level video features (e.g., local object/region/motion boundaries) than CSS, leading to better performance in two video applications (Section 7). See Appendix for more visual comparison and accompanying demo video for more dynamic details.

els is time-and-space efficient and scales well with large video data; (4) *Easy to use*: users simply specify the desired number of supervoxels and should not be bothered to tune other parameters; (5) *Parsimony*: the above principles are achieved with as few supervoxels as possible.

So far, none of existing methods satisfy all above principles. In our previous work [43], we propose a content-sensitive approach to address the parsimony principle. This approach is motivated by an important observation: the scene layouts and motions of different objects in a video usually exhibit large diversity, and thus the density of video content often varies significantly in different parts of the video. Generating spatiotemporally uniform supervoxels indiscriminately in the whole video often leads to under-segmentation in content-dense regions (i.e., with high variation of appearance and/or motion), and over-segmentation in content-sparse regions (i.e., with homogeneous appearance and motion). Therefore, computing supervoxels adaptively with respect to the density of video content can achieve a good balance among different principles.

To compute content-sensitive supervoxels (CSSs), Yi et al. [43] map a video Υ to 3-manifold $\mathcal{M}$ embedded in a feature space $\mathbb{R}^6$. The map Φ is designed in such a way that the volumetric elements in $\mathcal{M}$ give a good measure of content density in Υ, and thus, a uniform tessellation $\mathcal{T}$ on $\mathcal{M}$ efficiently induces CSSs (i.e., $\Phi^{-1}(\mathcal{T})$) in Υ. In this paper, we improve upon our previous work [43] and propose *feature-aware* CSS (FCSS). Our key idea is that among all possible uniform tessellations on $\mathcal{M}$, we find one whose cell boundaries well align with local object/region/motion boundaries in video. To achieve this goal, we improve the restricted centroidal Voronoi tessellation (RCVT) method [23], [43] and make the following contributions.

- To measure the degree of alignment between RCVT's cell boundaries and local video features, we propose an average boundary distance measure d_{bdry} and use it to control the positions of generating points in RCVT.
- We formulate FCSSs by simultaneously minimizing the RCVT energy (leading to uniform cells in RCVT) and maximizing d_{bdry} (leading to good local feature alignment).
- To quickly compute FCSSs, we propose a splitting-merging scheme that can be efficiently incorporated into the well known K-means++ algorithm [2], [40].

Our method has a theoretical constant-factor bi-criteria approximation guarantee, and in practice our method can obtain good supervoxels in very few iterations. By applying the streaming version of K-means++ (a.k.a. K-means# [1]), our method can be easily extended to process long videos that cannot be loaded into main memory at once. We thoroughly evaluate FCSS, streaming FCSS and ten representative supervoxel methods on four video datasets. A visual comparison is shown in Figure 1. The results show that our method achieves a good balance among over-segmentation accuracies (UE3D, SA3D, BRD and EV in Section 6), compactness, and time and space efficiency. As a case study, we also evaluate these methods on two novel applications (foreground propagation in video [14] and optimal video closure [17]) and the results show that FCSS achieves the best propagation and spatiotemporal closure performance.

2 Preliminaries

Our method improves the CSS work [43] that uses RCVT to compute a uniform tessellation of a 3-manifold $\mathcal{M} \subset \mathbb{R}^6$. Theoretically our method is a *bi-criteria approximation* to the K-means problem [2], [1], [40]. We briefly introduce them before presenting our method.

2.1 Video manifold $\mathcal{M}$ and CSS

Simply treating the time dimension in a video equivalently as spatial dimensions results in a regular 3D lattice representation of voxels in $\mathbb{R}^3$, which is not proper due to possibly non-negligible motions and occlusions/disocclusions. To overcome this drawback, some methods (e.g., [13], [5]) use optical flow to re-establish a connection graph of neighboring voxels between adjacent frames. However, even state-of-the-art optical flow estimation methods [36] are still imperfect and may introduce extra errors into supervoxel computation. Recently, Reso et al. [31] propose a novel formulation specifically designed for handling occlusions.

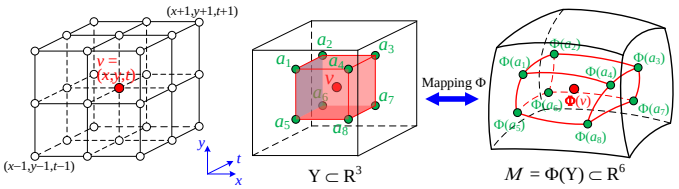


Fig. 2. Left: regular 3D lattices of voxels in $\mathbb{R}^3$. Middle and right: the map $\Phi : \Upsilon \to \mathcal{M} \subset \mathbb{R}^6$ stretches the unit cube $\boxdot_v$ (red box) centered at the voxel $v(x, y, t) \in \Upsilon$ into a 3-manifold $\mathcal{M}$. Each corner a_i of $\boxdot_v$, $i = 1, 2, \cdots, 8$, is the center of its eight neighboring voxels.

On the contrary, without any special treatment for occlusions, the video manifold $\mathcal{M}$ proposed in [43] provides an elegant continuous search space that circumvents the aforementioned drawback.

In [43], $\mathcal{M}$ is constructed as a 3-manifold embedded in a combined color and spatiotemporal space $\mathbb{R}^6$, which stretches a video Υ by a map $\Phi : \Upsilon \to \mathcal{M} \subset \mathbb{R}^6$ (Figure 2):

$$\Phi(v) = (\lambda_1 x, \lambda_1 y, \lambda_2 t, \lambda_3 l(v), \lambda_3 a(v), \lambda_3 b(v)), \quad (1)$$

where a voxel $v \in \Upsilon$ is represented by (x, y, t), (x, y) is the pixel location and t the frame index. $(l(v), a(v), b(v))$ is the color at the voxel v in the CIELAB color space. $\lambda_1 = \lambda_2 = 0.435$ and $\lambda_3 = 1$ are global stretching factors.

The volume of a region $\Phi(\Omega) \subset \mathcal{M}$ depends on both the volume of $\Omega \subset \Upsilon$ and the color variation in Ω. The higher variation of colors in Ω (indicating higher variation of appearance and/or motion), the larger the volume of $\Phi(\Omega)$. Therefore, the volume form on $\mathcal{M}$ gives a good measure of content density in Υ and the inverse mapping Φ^{-1} of any uniform tessellation on $\mathcal{M}$ generates CSSs in Υ.

To measure the volume in $\mathcal{M}$, $\forall v \in \Upsilon$, the volume $V(\Phi(\boxdot_v))$ of $\Phi(\boxdot_v) \subset \mathcal{M}$ is quickly evaluated only once [43], where $\boxdot_v$ is the unit cube centered at the voxel v (Figure 2 middle). Then the volume of $\Phi(\Omega) \subset \mathcal{M}$ is simply the sum $\Sigma_{v_j \in \Omega} V(\Phi(\boxdot_{v_j}))$.

2.2 Restricted Voronoi tessellation and RCVT

RCVT has been used to build uniform tessellations on manifolds [23], [43]. Denote by $S_K = \{s_i\}_{i=1}^K$ a set of generating points and $\mathcal{M}$ a 3-manifold in $\mathbb{R}^6$. The Euclidean Voronoi cell of a generator s_i in $\mathbb{R}^6$, denoted by $C_{\mathbb{R}^6}$, is

$$C_{\mathbb{R}^6}(s_i) \triangleq \{x \in \mathbb{R}^6 : \|x - s_i\|_2 \le \|x - s_j\|_2, \forall j \neq i, s_j \in S_K\}. \quad (2)$$

The restricted Voronoi cell $C_{\mathcal{M}}$ is defined to be the intersection of $C_{\mathbb{R}^6}$ and $\mathcal{M}$

$$C_{\mathcal{M}}(s_i) \triangleq \mathcal{M} \cap C_{\mathbb{R}^6}(s_i), \quad (3)$$

and its mass centroid is

$$m_i \triangleq \frac{\int_{x \in C_{\mathcal{M}}(s_i)} x dx}{\int_{x \in C_{\mathcal{M}}(s_i)} dx}. \quad (4)$$

The restricted Voronoi tessellation $RVT(S_K, \mathcal{M})$ is the collection of restricted Voronoi cells

$$RVT(S_K, \mathcal{M}) \triangleq \{C_{\mathcal{M}}(s_i) \neq \emptyset, \forall s_i \in S_K\}, \quad (5)$$

which is a finite closed covering of $\mathcal{M}$. An $RVT(S_K, \mathcal{M})$ is an RCVT if and only if each generator $s_i \in S_K$ is the mass centroid of $C_{\mathcal{M}}(s_i)$.

Theorem 1. *[23], [43] Let $\mathcal{M}$ be a 3-manifold embedded in $\mathbb{R}^6$ and $K \in \mathbb{Z}_+$ be a positive integer. For an arbitrary set S_K of points $\{s_i\}_{i=1}^K$ in $\mathbb{R}^6$ and an arbitrary tessellation $\{C_i\}_{i=1}^K$ on $\mathcal{M}$, $\bigcup_{i=1}^K C_i = \mathcal{M}$, $C_i \bigcap C_j = \emptyset$, $\forall i \neq j$, define the tessellation energy functional as follows:*

$$\mathcal{E}(\{(s_i, C_i)\}_{i=1}^K) = \sum_{i=1}^K \int_{x \in C_i} \|x - s_i\|_2^2 dx. \quad (6)$$

Then the necessary condition for $\mathcal{E}$ to be minimized is that $\{(s_i, C_i)\}_{i=1}^K$ is an RCVT of $\mathcal{M}$.

Theorem 1 indicates that RCVT is a uniform tessellation on $\mathcal{M}$, which minimizes the energy $\mathcal{E}$.

2.3 Bi-criteria approximation algorithms

The discretized counterpart of RCVT is the solution to the K-means problem on the manifold domain $\mathcal{M}$. Given a fixed K, denote by $S_K^{opt} = \{s_i^{opt}\}_{i=1}^K$ and $\{C_i^{opt}\}_{i=1}^K$ the (unknown) optimal generator set and tessellation on $\mathcal{M}$, respectively, which minimize the energy $\mathcal{E}$. Let S_K and $\{C_i\}_{i=1}^K$ be the generator set and tessellation output from an algorithm $\mathcal{A}$. An algorithm $\mathcal{A}$ is said to be b-approximation if for all instances of the problem, it produces a solution $\{s_i, C_i\}_{i=1}^K$ satisfying $\frac{\mathcal{E}(\{(s_i, C_i)\}_{i=1}^K)}{\mathcal{E}(\{(s_i^{opt}, C_i^{opt})\}_{i=1}^K)} \le b$. An algorithm is called (a, b)-approximation, if it outputs $\{(s_i, C_i)\}_{i=1}^{aK}$ with aK generators and tessellation cells, such that $\frac{\mathcal{E}(\{(s_i, C_i)\}_{i=1}^{aK})}{\mathcal{E}(\{(s_i^{opt}, C_i^{opt})\}_{i=1}^K)} \le b$, where $a > 1$ and $b > 1$.

3 Overview of FCSS

The classic Lloyd method is used in [23], [43] to compute RCVT on manifold $\mathcal{M}$, which iteratively moves each generator s_i to the corresponding mass centroid of $C_{\mathcal{M}}(s_i)$ and updates the RVT. Note that the Lloyd method converges to a local minimum. Among all possible local minimums (each corresponding to a uniform tessellation on $\mathcal{M}$), we propose FCSS which aims at finding one whose cell boundaries well align with local video boundaries. To achieve this goal, our FCSS method is built upon an important observation: when the set of generating points are far away from local object/region/motion boundaries in $\mathcal{M}$, the cell boundaries of their RCVT will well align with these local video boundaries. See Figure 3 for an illustration.

In a video Υ, local boundaries most likely appear in regions with high variation of appearance. Therefore, we can characterize these local regions in $\mathcal{M}$ by the volume $V(\Phi(\boxdot_v))$: the larger the volume $V(\Phi(\boxdot_v))$, the higher the probability that the voxel v lies on a local boundary. Since $V(\Phi(\boxdot_v)) \ge 1$ and it can be extremely large at sharp boundaries, we use the following nonlinear normalization:

$$p_{bdry}(v) = \frac{2}{\pi} \arctan(V(\Phi(\boxdot_v))) \quad (7)$$

to characterize the possibility of a voxel v being on the local boundary. Then our objective can be casted as finding a set of generating points $\{s_i\}_{i=1}^K$, in which each s_i is around the mapped position $\Phi(v')$ of a voxel v' with low boundary

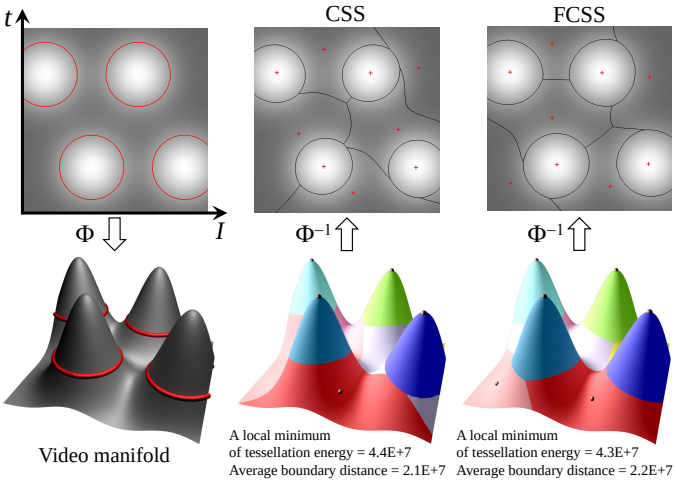


Fig. 3. Comparison of FCSS and CSS generation on a synthetic, degenerate gray video Υ, for easy illustration. In Υ, each image frame at time t is a degenerate 1D gray line image I_t. Supervoxels are generated by a tessellation in Υ. Left: Υ is mapped to a video manifold $\mathcal{M} = \Phi(I,t) \subset \mathbb{R}^3$, whose area elements give a good measure of content density in Υ. Middle and right: two local minimums of the tessellation energy specified in Eq.(6). The generating points are shown in dots on $\mathcal{M}$ and their inverse images by Φ^{-1} are shown in red crosses + in Υ. In this toy example, the local boundaries in Υ can be characterized by the zero crossing of the second derivative of $\Upsilon(I,t)$ (shown in red circles in left). Note that the tessellations in middle and right are generated without this information. The generating points in FCSS are farther away from local boundaries (indicating by a larger *average boundary distance* proposed in Eq.(8)), and the cell boundaries of the corresponding tessellation better capture these local boundaries than that of CSS.

possibility $p_{bdry}(v')$. We formulate this objective by proposing an *average boundary distance* (ABD) for a tessellation $\{s_i, C_i\}_{i=1}^K$:

$$d_{bdry}\left(\{(s_i,C_i)\}_{i=1}^K\right) = \sum_{i=1}^K \int_{x\in C_i} p_{bdry}(x)\|x-s_i\|_2^2 dx, \quad (8)$$

where $p_{bdry}(x) = p_{bdry}(\Phi^{-1}(x))$. The larger the distance d_{bdry} is, the farther $\{s_i\}_{i=1}^K$ are from the local boundaries. Two examples are shown in Figure 3: the average boundary distances of the middle and right tessellations are 2.1×10^7 and 2.2×10^7, respectively.

In the next section, we implement FCSS using a variant of the Lloyd method that finds a uniform tessellation by minimizing the tessellation energy defined in Eq.(6) in such a way that the optimal generating points are determined by minimizing the following ABD-weighted tessellation energy:

$$\begin{aligned}&\mathcal{E}_\alpha\left(\{(s_i,C_i)\}_{i=1}^K\right) = \\ &\mathcal{E}\left(\{(s_i,C_i)\}_{i=1}^K\right) - \alpha d_{bdry}\left(\{(s_i,C_i)\}_{i=1}^K\right) = \\ &\sum_{i=1}^K \int_{x\in C_i}(1-\alpha p_{bdry}(x))\|x-s_i\|_2^2 dx,\end{aligned} \quad (9)$$

where $\alpha > 0$ is a weight to balance the two terms $\mathcal{E}$ and d_{bdry}. In all our experiments, we set $\alpha = 0.8$.

4 Implementation of FCSS

To obtain a feature-aware uniform tessellation $\{(s_i, C_i)\}_{i=1}^K$ in $\mathcal{M}$, our FCSS method consists of the following two steps:

- Initialization (Section 4.1). We apply a variant of the K-means++ algorithm [2], [1], [40] to determine the initial positions of generating points $S_K = \{s_i\}_{i=1}^K$, which ensures an $(O(1), O(1))$-approximation.
- Feature-aware Lloyd refinement (Section 4.2). Observing that the classic Lloyd method converges only to a local minimum and without feature-aware control on the positions of generating points, we optimize the tessellation and the positions of generating points separately by minimizing two energy forms (i.e., Eqs. (6) and (9)), leading to a local minimum of the content-sensitive uniform tessellation on the manifold $\mathcal{M}$ that also optimizes the average boundary distance d_{bdry} to improve feature alignment of cell boundaries. We further propose an efficient splitting-merging scheme that helps move the solution out of local minimums, while preserving the approximation ratio.

Algorithm 1 Initialization

Input: A video Υ of N voxels and the desired number of supervoxels K.
Output: The initial positions of K generating points $S_K = \{s_i\}_{i=1}^K$.

1: Compute $V(\Phi(\boxdot_v))$ for each voxel $v \in \Upsilon$.
2: Choose a point v_1 from all voxels $v \in \Upsilon$ with probability proportional to $V(\Phi(\boxdot_v))$.
3: Set $s_1 = \Phi(v_1)$, $S_1 = \{s_1\}$ and $j = 1$.
4: **while** $j < K$ **do**
5: Choose a point v_{j+1} from all voxels $v \in \Upsilon$ with probability proportional to the cost $c_{S_j}(v)$ (Eq.(11)).
6: Set $s_{j+1} = \Phi(v_{j+1})$, $S_{j+1} = S_j \cup \{s_{j+1}\}$ and $j = j+1$.
7: **end while**

The FCSS method is easy to implement and can obtain high-quality supervoxels in very few iterations. Theoretically FCSS is $(O(1), O(1))$-approximation (Section 4.3). We also present a simple extension of FCSS to streaming FCSS for processing long videos (Section 5).

4.1 Initialization

We apply a variant of K-means++ algorithm to obtain a provable high-quality initialization of generating points $S_K = \{s_i\}_{i=1}^K$. The pseudo-code is summarized in Algorithm 1. In each step, a point in $\mathcal{M}$ is picked up with probability proportional to its current cost (defined as its squared distance to the nearest generator picked so far), and added as a new generator. To compute the required probability in the manifold domain $\mathcal{M}$, we consider the positions of mapped voxels $\Phi(v) \in \mathcal{M}$, $\forall v \in \Upsilon$. With respect to an existing generator $\Phi(v_i)$, the cost of a mapped voxel $\Phi(v_j) \in \mathcal{M}$, $j \neq i$, is

$$\begin{aligned}c_{v_i}(v_j) &= \int_{x\in\Phi(\boxdot_{v_j})}\|x-\Phi(v_i)\|_2^2 dx \\ &\approx V(\Phi(\boxdot_{v_j}))\cdot\|\Phi(v_j)-\Phi(v_i)\|_2^2\end{aligned} \quad (10)$$

Then the cost of picking $\Phi(v_j)$ with respect to an existing generator set S is

$$c_S(v_j) = \min_{v_i\in S} c_{v_i}(v_j) \quad (11)$$

Algorithm 1 runs in $O(NK)$ time. A simple adaption of the proofs in [2], [40] shows the following results.

Lemma 1. *[2], [40] If $RVT(S_K, \mathcal{M})$ is used as the tessellation for the selected K generators, Algorithm 1 is an expected $\Theta(\log K)$-approximation algorithm. If βK ($\beta > 1$) generators are selected, Algorithm 1 is an expected b-approximation algorithm, where*

$$b < 8\left(1 + \frac{1+\sqrt{5}}{2(\beta-1)}\right). \tag{12}$$

4.2 Feature-aware Lloyd refinement

Given the initial generators $S_K = \{s_i\}_{i=1}^K$, $s_i \in \mathcal{M}$, the classic Lloyd method computes $RCVT(S_K, \mathcal{M})$ iteratively by alternating the following two steps:

- Step 1: Fixing the generator set S_K, compute $RVT(S_K, \mathcal{M})$ (ref. Eq.(5));
- Step 2: For each cell $C_{\mathcal{M}}$ in $RVT(S_K, \mathcal{M})$, update its generator to be the mass centroid of $C_{\mathcal{M}}$ (ref. Eq.(4)).

This method converges only to a local minimum with a large number of iterations [8] and without feature-aware control on the cell boundaries of $RVT(S_K, \mathcal{M})$. We introduce the average boundary distance (Eq.(8)) into the tessellation energy (Eq.(9)) and propose the following feature-aware Lloyd refinement and Algorithm 2 summarizes the pseudo-code:

- Step 1 (lines 2&14&24): Fixing the generator set S_K, compute $RVT(S_K, \mathcal{M})$;
- Step 2 (lines 5-13): For each cell $C_{\mathcal{M}}$ in $RVT(S_K, \mathcal{M})$, update its generator to a place $\Phi(v')$ to reduce the weighted tessellation energy (Eq.(9)) which jointly improves content-sensitive uniform tessellation and cell boundary feature alignment, while ensuring the tessellation energy (Eq.(6)) is not increased.
- Step 3 (lines 15-23): Perform the splitting and merging operations to move the solution out of local minimums, while ensuring the tessellation energy (Eq.(6)) is not increased.
- Step 4 (line 4): If $RVT(S_K, \mathcal{M})$ satisfies the stopping condition, then stop; otherwise, return to Step 1.

The implementation of Steps 2 and 3 is presented in Sections 4.2.1 and 4.2.2 respectively. The convergence of Algorithm 2 is proved in Section 4.3.

4.2.1 Feature-aware update of generators

Our objective is to move each generator to a place $\Phi(v')$ whose inverse mapping $v' \in \Upsilon$ has low boundary possibility $p_{bdry}(v')$, and meanwhile the RCVT tessellation energy (Eq.(6)) is decreased. To do so, we minimize the weighted tessellation energy (Eq.(9)) for each generator s_i, i.e., setting

$$\frac{\partial \mathcal{E}_\alpha\left(\{(s_j, C_j)\}_{j=1}^K\right)}{\partial s_i} = 0 \tag{13}$$

which implies that the optimal position $\widetilde{s}_i$ for s_i is

$$\widetilde{s}_i = \frac{\int_{x\in C_i}(1-\alpha p_{bdry}(x))x dx}{\int_{x\in C_i}(1-\alpha p_{bdry}(x))dx}. \tag{14}$$

However, moving s_i to $\widetilde{s}_i$ may increase the tessellation energy (Eq.(6)). We make use of the following proposition.

Algorithm 2 FCSS generation

Input: A video Υ of N voxels, the desired number of supervoxels K, num_{random} the number of repeated random sampling of generators in each iteration, and the maximum number of iterations $iter_{max}$.
Output: K content-sensitive supervoxels.

1: Initialize the generators $S_K = \{s_i\}_{i=1}^K$ (Algorithm 1).
2: Compute $RVT(S_K, \mathcal{M})$.
3: Set $iter = 0$.
4: **while** $iter < iter_{max}$ **do**
5: **for** each cell $C_{\mathcal{M}}(s_i)$ in RVT **do**
6: Compute the candidate position $\widetilde{s}_i$ that minimizes the weighted tessellation energy (Eq.(14)).
7: Compute the mass centroid m_i of $C_{\mathcal{M}}(s_i)$.
8: **if** $\|\widetilde{s}_i - m_i\| \le \|s_i - m_i\|$ **then**
9: Update s_i by $\widetilde{s}_i$.
10: **else**
11: Update s_i by $\widehat{s}_i$, which is the intersection point between the sphere centered at m_i of radius $\|s_i - m_i\|$ and the line segment connecting m_i and $\widetilde{s}_i$.
12: **end if**
13: **end for**
14: Compute $RVT(S_K, \mathcal{M})$.
15: Set $n = 0$.
16: **while** $n < num_{random}$ **do**
17: Randomly pick three generators s_m, s_i, s_j in S_k (Algorithm 4).
18: Check the splitting-merging feasibility of (s_m, s_i, s_j) (Algorithm 3) and put the return values in $(Flag, s'_p, s'_q, s'_k)$.
19: **if** $Flag == TRUE$ **then**
20: Update S_K by splitting s_m into (s'_p, s'_q) and merging (s_i, s_j) into s'_k.
21: **end if**
22: $n = n + 1$;
23: **end while**
24: Locally update $RVT(S_K, \mathcal{M})$.
25: $iter = iter + 1$;
26: **end while**
27: Compute $\Phi^{-1}(RVT(S_K, \mathcal{M}))$ to obtain K supervoxels.

Proposition 1. *Given an $RVT(S_K, \mathcal{M})$, we fix the tessellation $\{C_{\mathcal{M}}(s_j)\}_{j=1}^K$ and move a generator $s_i \in S_K$ to a new position s'_i. Let $C_j = C_{\mathcal{M}}(s_j)$. The tessellation energy (ref. Eq.(6))*

$$\mathcal{E}(\{(s_j, C_j)\}_{j=1}^K) = \sum_{j=1}^K \mathcal{E}(s_j, C_j) = \sum_{j=1}^K \int_{x\in C_j} \|x - s_j\|_2^2 dx \tag{15}$$

satisfies

$$\mathcal{E}(s'_i, C_i) \le \mathcal{E}(s_i, C_i), \tag{16}$$

if and only if

$$\|s'_i - m_i\| \le \|s_i - m_i\| \tag{17}$$

where m_i is the mass centroid of the cell C_i (ref. Eq.(4)).

Proof. Given inequality (17), we have

$$\mathcal{E}(s_i, C_i) = \mathcal{E}(m_i, C_i) + \textstyle\int_{x\in C_i} \|s_i - m_i\|_2^2 dx \ge \mathcal{E}(m_i, C_i) + \int_{x\in C_i} \|s'_i - m_i\|_2^2 dx = \mathcal{E}(s'_i, C_i)$$

Algorithm 3 Check splitting-merging feasibility

Input: Three generators (s_m, s_i, s_j) in S_K and an $RVT(S_K, \mathcal{M})$.
Output: A Boolean variable $Flag$ indicating the feasibility and three new generators (s'_p, s'_q, s'_k).

1: Compute the mass centroids s'_m, s'_i and s'_j of $C_{\mathcal{M}}(s_m)$, $C_{\mathcal{M}}(s_i)$ and $C_{\mathcal{M}}(s_j)$, respectively.
2: Compute the diameter d_m of the cell $C_{\mathcal{M}}(s_m)$ and the points p_{m1} and p_{m2} (see Definition 1).
3: Compute two new cells $C'(p_{m1})$ and $C'(p_{m2})$, which are the Voronoi cells of p_{m1} and p_{m2} in the domain $C_{\mathcal{M}}(s_m)$.
4: Compute the mass centroids s'_k, s'_p and s'_q of $C_{\mathcal{M}}(s_i) \cup C_{\mathcal{M}}(s_j)$, $C'(p_{m1})$ and $C'(p_{m2})$, respectively.
5: Compute $\tau_{m,i,j}$ in Eq. (20).
6: **if** $\|s'_p - s'_m\|_2 > \tau_{m,i,j}$ and $\|s'_q - s'_m\|_2 > \tau_{m,i,j}$ **then**
7: **return** $TRUE$ and (s'_p, s'_q, s'_k).
8: **else**
9: **return** $FALSE$ and $(NULL, NULL, NULL)$.
10: **end if**

On the other hand, if $\mathcal{E}(s_i, C_i) \geq \mathcal{E}(s'_i, C_i)$, then we have $\int_{x \in C_i} \|m_i - s_i\|_2^2 dx \geq \int_{x \in C_i} \|m_i - s'_i\|_2^2 dx$, indicating $\|m_i - s_i\| \geq \|m_i - s'_i\|$. That completes the proof. □

In Algorithm 2 (lines 8-9), we check the condition in Eq.(17) using the optimal position $\widetilde{s}_i$. If it is satisfied, we update s_i by $\widetilde{s}_i$. Otherwise, we set s_i by moving along the direction from m_i to $\widetilde{s}_i$ (the average boundary distance d_{bdry} is expected to be increased along this direction) and locating it at the boundary of the sphere centered at m_i of radius $\|s_i - m_i\|$ (moving s_i to this place does not increase the tessellation energy). In both cases, we try to reduce the weighted tessellation energy, while ensuring the tessellation energy is not increased.

4.2.2 Splitting and merging operations

In Algorithm 2 (lines 16-23), we perform splitting and merging operations for jumping out of a small local search area in $\mathcal{M}$ while the tessellation energy still does not increase. We find that these splitting and merging operations help Algorithm 2 obtain high-quality supervoxels in very few iterations.

A splitting operation $\wedge : s_m \to (s'_p, s'_q)$ splits an RVT cell $C_{\mathcal{M}}(s_m)$ into two new cells $C(s'_p)$ and $C(s'_q)$. Conversely, a merging operation $\vee : (s_i, s_j) \to s'_k$ merges two RVT cells $C_{\mathcal{M}}(s_i)$ and $C_{\mathcal{M}}(s_j)$ into a new cell $C(s'_k)$. Splitting reduces the tessellation energy and merging increases it. The number of generators does not change by applying a pair of splitting and merging operations $(\wedge, \vee) : (s_m, (s_i, s_j)) \to ((s'_p, s'_q), s'_k)$. Our goal is to design a pair $(\wedge, \vee)$ that does not increase the tessellation energy. We make use of the following definition and proposition.

Definition 1. *The diameter d_i of a cell $C_{\mathcal{M}}(s_i)$, $s_i \in S_K$, is the maximum Euclidean distance between pairs of points in the cell, i.e.,*

$$d_i = \max_{\forall x, y \in C_{\mathcal{M}}(s_i)} \|x - y\|_2 \tag{18}$$

Denote by p_{i1} and p_{i2} the two points in $C_{\mathcal{M}}(s_i)$ satisfying $\|p_{i1} - p_{i2}\| = d_i$.

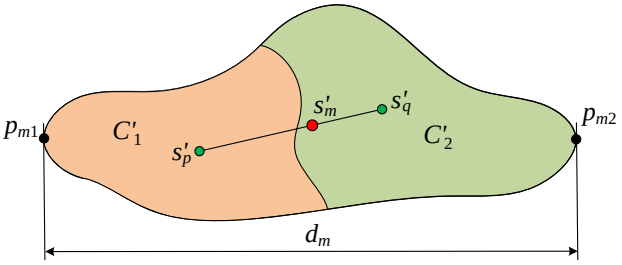


Fig. 4. The diameter $d_m = \|p_{m1} - p_{m2}\|_2$ of an RVT cell $C_{\mathcal{M}}(s_m)$ (shaded area) is the maximum Euclidean distance between pairs of points in this cell. The splitting operation $\wedge : s_m \to (s'_p, s'_q)$ splits an RVT cell $C_{\mathcal{M}}(s_m)$ (shaded area) into two arbitrary new cells C'_1 (orange shaded area) and C'_2 (green shaded area), satisfying $p_{m1} \in C'_1$ and $p_{m2} \in C'_2$, $C'_1 \cap C'_2 = \emptyset$ and $C'_1 \cup C'_2 = C_{\mathcal{M}}(s_m)$. The mass centroids of cells $C_{\mathcal{M}}(s_m)$, C'_1 and C'_2 are s'_m, s'_p, s'_q, respectively. Lemma 2 proves that s'_m lies on the line segment connecting s'_p and s'_q.

Proposition 2. *Let s_m, s_i, s_j be three generators in an $RVT(S_K, \mathcal{M})$. Let (m_m, m_i, m_j) and (s'_m, s'_i, s'_j) be the masses and mass centroids of the cells $C_{\mathcal{M}}(s_m)$, $C_{\mathcal{M}}(s_i)$, $C_{\mathcal{M}}(s_j)$, respectively. Consider a splitting of $C_{\mathcal{M}}(s_m)$ into two arbitrary new cells C'_1 and C'_2, which satisfies $p_{m1} \in C'_1$, $p_{m2} \in C'_2$, $C'_1 \cap C'_2 = \emptyset$ and $C'_1 \cup C'_2 = C_{\mathcal{M}}(s_m)$. Let s'_p, s'_q and s'_k be the mass centroids of C'_1, C'_2 and $C_{\mathcal{M}}(s_i) \cup C_{\mathcal{M}}(s_j)$, respectively. If*

$$\|s'_p - s'_m\|_2 > \tau_{m,i,j} \ \ and \ \ \|s'_q - s'_m\|_2 > \tau_{m,i,j}, \tag{19}$$

where

$$\tau_{m,i,j} = \sqrt{\frac{m_i m_j}{m_m(m_i + m_j)}} \|s'_i - s'_j\|_2 \tag{20}$$

then the pair of operations $(\wedge, \vee) : (s_m, (s_i, s_j)) \to ((s'_p, s'_q), s'_k)$ do not increase the tessellation energy $\mathcal{E}$ in Eq.(6).

Proof. See Appendix. □

To ensure the pair of operations $(\wedge, \vee)$ do not increase the tessellation energy $\mathcal{E}$, in Algorithm 2 (lines 16-21), we check the splitting-merging feasibility condition (Eq.(19)) and Algorithm 3 summarizes the pseudo-code. Note that computing the diameter of an arbitrary region (line 2 of Algorithm 3) is time-consuming. In practice, we compute the axis-aligned bounding box B of $C_{\mathcal{M}}(s_m)$. B is determined by two supporting points p_1 and p_2 in $C_{\mathcal{M}}(s_m)$ and we use them as fast approximations to p_{m1} and p_{m2}.

Lemma 2. *Let s'_m, s'_p and s'_q be the mass centroids as specified in Proposition 2. Then s'_m lies on the line segment connecting s'_p and s'_q.*

Proof. Refer to Figure 4. Let m_m, m_1 and m_2 be the masses of $C_{\mathcal{M}}(s_m)$, C'_1 and C'_2. Since $C'_1 \cap C'_2 = \emptyset$ and $C'_1 \cup C'_2 = C_{\mathcal{M}}(s_m)$, we have

$$s'_m = \frac{\int_{x \in C_M(s_m)} x\, dx}{m_m} = \frac{\int_{x \in C'_1} x\, dx + \int_{x \in C'_2} x\, dx}{m_1 + m_2} = \frac{m_1 s'_p + m_2 s'_q}{m_1 + m_2} = \frac{m_1}{m_1 + m_2} s'_p + \frac{m_2}{m_1 + m_2} s'_q \tag{21}$$

That completes the proof. □

Note that for a region $\Omega \subset \Upsilon$ with a fixed volume, the higher variation of colors in Ω, the larger the volume of $\Phi(\Omega) \subset \mathcal{M}$ and vice versa. Lemma 2 and Proposition 2 imply the following important geometric observation:

- to pass the feasibility checking in inequalities (19), we need to pick up three generators (s_m, s_i, s_j) that have large $\|s'_p - s'_m\|_2$ and $\|s'_q - s'_m\|_2$, and small $\tau_{m,i,j}$;
- a large volume of a cell $C_{\mathcal{M}}(s_m)$ will result in large m_m, $\|s'_p - s'_m\|_2$ and $\|s'_q - s'_m\|_2$, and small $\tau_{m,i,j}$, implying that the larger the volume of a cell $C_{\mathcal{M}}(s_m)$, the more likely it is split, thus producing more generators in content-rich regions.
- small volumes of cells $C_{\mathcal{M}}(s_i)$ and $C_{\mathcal{M}}(s_j)$ will result in small m_i, m_j and $\tau_{m,i,j}$ in Eq.(20), implying that the smaller the volumes of cells $C_{\mathcal{M}}(s_i)$ and $C_{\mathcal{M}}(s_j)$, the more likely they are merged, thus reducing the number of generators in content-sparse regions.

Therefore, to increase the feasibility of the splitting-merging operation at line 18 of Algorithm 2, we estimate content-dense and content-sparse regions in $RVT(S_K, \mathcal{M})$ and collect their corresponding generators into subsets S_{dense} and S_{sparse} in Algorithm 4. If S_{dense} and S_{sparse} contain sufficient generators, we randomly pick two neighboring generators in S_{sparse} to be merged and pick one generator in S_{dense} to be split; otherwise, we randomly pick three generators in S_K. To estimate the content density of cells, we compute the expected cell volume as the average of K cells over the total volume of video manifold $\mathcal{M}$:

$$\mathbb{E}(V(C_{\mathcal{M}})) = \frac{\sum_{v\in\Upsilon} V(\Phi(\boxdot_v))}{K} \tag{22}$$

For each cell $C_{\mathcal{M}}$ in $RVT(S_K, \mathcal{M})$, we compare its volume $V(C_{\mathcal{M}})$ with $\mathbb{E}(V(C_{\mathcal{M}}))$: (1) if $V(C_{\mathcal{M}}) > 4\mathbb{E}(V(C_{\mathcal{M}}))$, we put the generator of this cell into S_{dense}, and (2) if $V(C_{\mathcal{M}}) < \mathbb{E}(V(C_{\mathcal{M}}))/4$, we put the generator of this cell into S_{sparse}. Algorithm 4 summarizes the pseudo-code.

Algorithm 4 Randomly pick three generators

Input: An $RVT(S_K, \mathcal{M})$ and an expected cell volume $\mathbb{E}(V(C_{\mathcal{M}}))$ (Eq.(22)).
Output: Three generators (s_m, s_i, s_j) in S_K.

```
Set S_dense = ∅ and S_sparse = ∅.
for each cell C_M(s_i) in RVT(S_K, M) do
    Compute the volume V(C_M(s_i)).
    if V(C_M(s_i)) > 4E(V(C_M)) then
        S_dense = S_dense ∪ {s_i}.
    else if V(C_M(s_i)) < E(V(C_M))/4 then
        S_sparse = S_sparse ∪ {s_i}.
    end if
end for
if |S_dense| > 2 then
    Randomly pick a generator s_m in S_dense.
else
    Randomly pick a generator s_m in S_K.
end if
Randomly pick a generator s_i in S_sparse.
Collect all neighboring pairs of s_i using 26-connectivity from S_sparse and put them into the set N.
if |N| > 2 then
    Pick a pair (s_j, s_i) in N such that C_M(s_j) has the closest mean color to C_M(s_i).
else
    Randomly pick two generators s_i and s_j in S_K.
end if
return (s_m, s_i, s_j).
```

4.3 Proof of $(O(1), O(1))$-approximation

In all our experiments, we set $iter_{max} = 20$ and $num_{random} = 20$ in Algorithm 2. We show in Section 6 that our algorithm can obtain high-quality supervoxels in 20 iterations. We have the following theoretical results.

Theorem 2. *By selecting $(1+\varepsilon)K$ generators, $0 < \varepsilon < 1$, Algorithm 2 is a bi-criteria $\left(1+\varepsilon, 8\left(1+\frac{1+\sqrt{5}}{2\varepsilon}\right)\right)$-approximation algorithm in expectation.*

Proof. Let $S_K^{opt} = \{s_i^{opt}\}_{i=1}^K$ and $\{C_i^{opt}\}_{i=1}^K$ be the (unknown) optimal generator set and tessellation on $\mathcal{M}$, which minimize the energy $\mathcal{E}$ in Eq.(6). Let $\mathcal{E}_{OPT} = \mathcal{E}(\{(s_i^{opt}, C_i^{opt})\}_{i=1}^K)$. By Lemma 1, for any $K' = (1+\varepsilon)K$ generators selected by Algorithm 1, the expected tessellation energy $\mathcal{E}$ satisfies

$$\frac{\mathbb{E}(\mathcal{E}(\{(s_i, C_i)\}_{i=1}^{K'}))}{\mathcal{E}_{OPT}} \leq 8\left(1+\frac{1+\sqrt{5}}{2\varepsilon}\right) \tag{23}$$

In feature-aware Lloyd refinement, we alternate the two steps — i.e., the feature-aware update of generators and splitting-and-merging operations — until the termination condition is reached. Both steps are designed for not increasing the energy $\mathcal{E}$. Therefore for any tessellation $RCVT(S_{K'}, \mathcal{M})$ output from Algorithm 2, its expected tessellation energy $\mathcal{E}$ satisfies

$$\mathbb{E}(\mathcal{E}(RCVT(S_{K'}, \mathcal{M}))) \leq 8\left(1+\frac{1+\sqrt{5}}{2\varepsilon}\right)\mathcal{E}_{OPT}$$

That completes the proof. □

Theorem 3. *By selecting $(1+\varepsilon)K$ generators, $0 < \varepsilon < 1$, the time and space complexities of Algorithm 2 are $O(NK)$ and $O(N+K)$, respectively.*

Proof. In Algorithm 2 (line 1), the initialization step (by Algorithm 1) takes $O(NK)$ time and $O(N+K)$ space. In the iteration (lines 4-26),

- by using a local search strategy in [23], computing or locally updating RVT takes $O(N)$ time and space;
- feature-aware update of generators takes $O(N)$ time;
- randomly picking three generators by Algorithm 4 takes $O(N)$ time and space;
- both checking the splitting-merging feasibility and applying the splitting-merging operations take $O(1)$ time and space.

As a summary, the time and space complexities of Algorithm 2 are $O(NK + iter_{max}(N + num_{rand}N))$ and $O(N+K)$, respectively. Since we used fixed values $iter_{max} = 20$ and $num_{random} = 20$, the time complexity reduces to $O(NK)$. That completes the proof. □

5 STREAMING FCSS FOR LONG VIDEOS

Using a simple adaption of the streaming K-means algorithm [1], Algorithm 2 is readily extended to a streaming version for processing long videos that cannot be loaded into main memory at once. The streaming FCSS algorithm represents the video manifold $\mathcal{M}$ by an ordered, discretized sequence of weighted points $\widetilde{\mathcal{M}} = \{(x_i, y_i, t_i, w_i)\}_{i=1}^{N}$, where (x_i, y_i, t_i) is the position of voxel v_i in Υ and w_i is the volume $V(\Phi(\boxdot_v))$. Pseudo-code is summarized in Algorithm 5.

Algorithm 5 Streaming FCSS

Input: A video Υ of N voxels and the desired number of supervoxels K.
Output: K content-sensitive supervoxels.

1: Compute the discretized manifold representation $\widetilde{\mathcal{M}} = \{(x_i, y_i, t_i, w_i)\}_{i=1}^{N}$.
2: Initialize $S = \mathcal{M}$.
3: **while** S cannot be loaded into main memory **do**
4: Set $\widetilde{S} = \emptyset$.
5: Divide S into l disjoint batches $\chi_1, \cdots, \chi_l$, such that each batch can be loaded into main memory.
6: **for** each batch χ_i **do**
7: Apply Algorithm 2 to compute $(1+\varepsilon)K$ generators $S_K(\chi_i)$.
8: Compute $RVT(S_K(\chi_i), \chi_i)$.
9: **for** each generator g_j in $S_K(\chi_i)$ **do**
10: Compute the total weight of all points in the cell corresponding to g_j in $RVT(S_K(\chi_i), \chi_i)$ and assign it to g_j as the weight w_j;
11: **end for**
12: $\widetilde{S} = \widetilde{S} \cup (g_j, w_j), \forall g_j \in S_K(\chi_i)$.
13: **end for**
14: $S = \widetilde{S}$.
15: **end while**
16: Apply Algorithm 2 to S for obtaining K supervoxels.

The simple one-pass streaming scheme analyzed in [1] partitions the points $\widetilde{\mathcal{M}}$ sequentially into batches $\{\chi_1, \cdots, \chi_l\}$, such that each batch χ_i of points can be loaded into main memory. For each χ_i, we preform Algorithm 2 to obtain $(1+\varepsilon)K$ generators and the weight for each generator can be determined by the corresponding cell in the RVT applied on χ_i. Finally, we consolidate all weighted generators produced from $\{\chi_1, \cdots, \chi_l\}$ into one single weight point set S. If S is still too large to fit in memory, the above process repeats. When S fits in memory, we apply Algorithm 2 again on S to obtain K supervoxels. Assume that the size of main memory is Ξ (in terms of the point number). Since each batch produces $(1+\varepsilon)K$ generators, to ensure that the process only needs to be performed once, the number of batches l should satisfy both of the following: $\frac{N}{l} \leq \Xi$ (where each batch fits in memory) and $(1+\varepsilon)K \cdot l \leq \Xi$ (where S fits in memory), i.e. $\frac{N}{\Xi} \leq l \leq \frac{\Xi}{(1+\varepsilon)K}$. Here N is the number of weighted points in $\widetilde{\mathcal{M}}$ (i.e., the number of voxels in the video). Ignoring rounding, such l exists, if $\frac{N}{\Xi} \leq \frac{\Xi}{(1+\varepsilon)K}$, i.e., $N \leq \frac{\Xi^2}{(1+\varepsilon)K}$. This shows although our algorithm may repeatedly apply lines 4-14 to reduce the size of S to handle arbitrarily large videos, in practice, doing so *once* already allows processing very large videos, with up to $\frac{\Xi^2}{(1+\varepsilon)K}$ voxels, significantly larger than Ξ voxels that can be handled by non-streaming FCSS. Note that in any case, the whole video only needs to be processed once, and remaining steps involve much smaller set S. In our experiments, we set $\varepsilon = 0.2$.

Theorem 4. *If $(1+\varepsilon)K$ generators, $0 < \varepsilon < 1$, are selected by Algorithm 2, Algorithm 5 is $(O(1), O(1))$-approximation.*

Proof. By Theorem 2, selecting $(1+\varepsilon)K$ generators, $0 < \varepsilon < 1$, makes Algorithm 2 an expected bi-criteria $(O(1), O(1))$-approximation algorithm. Theorem 3.1 in [1] states that if Algorithm 2 is an (a, b)-approximation, the two-level Algorithm 5 is an $(a, 2b+4b(b+1))$-approximation. Accordingly, Algorithm 5 is $(O(1), O(1))$-approximation. That completes the proof. □

6 EXPERIMENTS

We implemented FCSS (Algorithm 2) and streaming FCSS (Algorithm 5) in C++ and source code is publicly available[1]. We compare our method (FCSS and streaming FCSS) with our previous work (CSS and streamCSS) [43] and eight methods: TS-PPM [16] and seven representative methods selected in [41], including NCut [34], [11], [10], SWA [32], [33], [7], MeanShift [28], GB [9], GBH [13], streamGBH [42] and TSP [5]. All the evaluations are tested on a PC with an Intel Core E5-2683V3 CPU and 256GB RAM running Linux. Since FCSS, streaming FCSS, CSS and streamCSS adopt a random initialization, we report the average results of 20 initializations. The performances are evaluated on four video datasets, i.e., SegTrack v2 [20], BuffaloXiph [6], BVDS [37], [12] and CamVid [3], which have ground-truth labels drawn by human annotators.

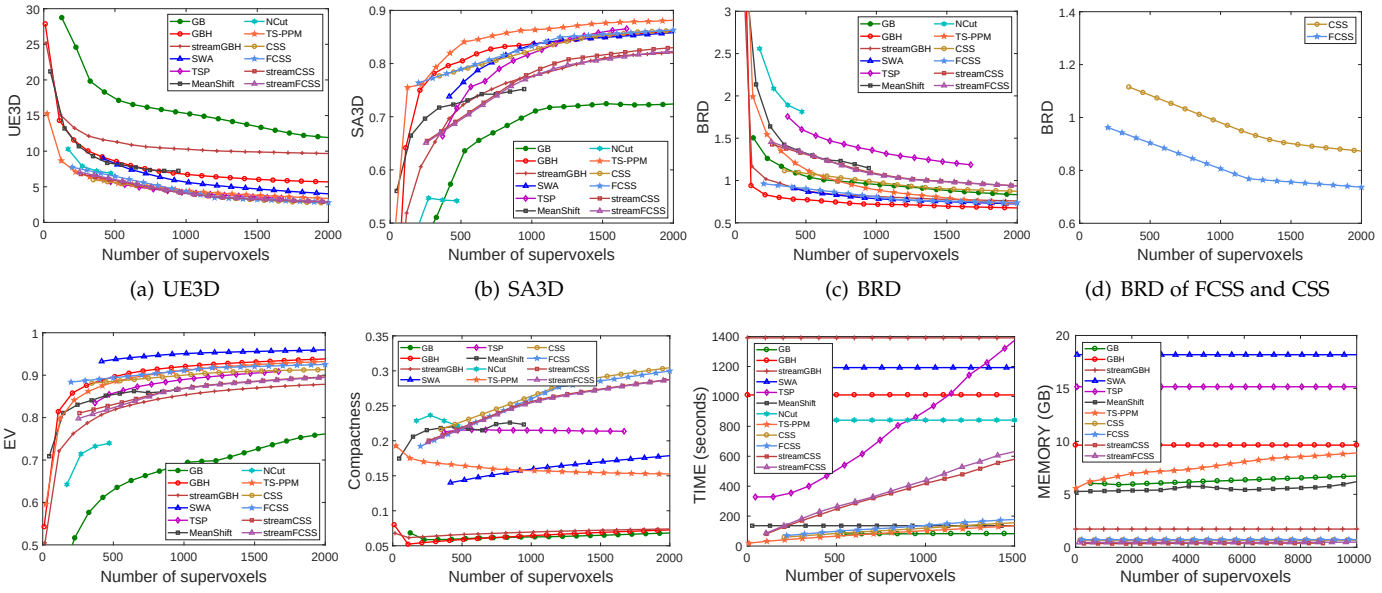
We adopt the following quality metrics that are commonly used for supervoxel evaluation. Some visual comparisons are illustrated in Figure 1, appendix and demo video in supplemental material.

Adherence to object boundaries. As perceptually meaningful atomic regions in videos, supervoxels should well preserve the object boundaries of ground-truth segmentation. 3D under-segmentation error (UE3D), 3D segmentation accuracy (SA3D) and boundary recall distance (BRD) are standard metrics in this aspect [5], [18], [41]. UE3D and SA3D are complementary to each other and both measure the tightness of supervoxels that overlap with ground-truth segmentation. Denote a ground-truth segmentation of a video as $\tilde{G} = \{\tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_{K_G}\}$, and a supervoxel segmentation as $\tilde{S} = \{\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_{K_S}\}$, where K_G and K_S are the numbers of supervoxels for the ground-truth segmentation $\tilde{G}$ and segmentation $\tilde{S}$. The UE3D and SA3D metrics are defined as

$$UE3D = \frac{1}{K_G} \sum_{\tilde{g}_i \in \tilde{G}} \frac{\sum_{\{\tilde{s}_j \in \tilde{S}: V(\tilde{s}_j \cap \tilde{g}_i) > 0\}} V(\tilde{s}_j) - V(\tilde{g}_i)}{V(\tilde{g}_i)} \tag{24}$$

$$SA3D = \frac{1}{K_G} \sum_{\tilde{g}_i \in \tilde{G}} \frac{\sum_{\{\tilde{s}_j \in \tilde{S}: V(\tilde{s}_j \cap \tilde{g}_i) \geq 0.5 V(\tilde{s}_j)\}} V(\tilde{s}_j \cap \tilde{g}_i)}{V(\tilde{g}_i)} \tag{25}$$

1. https://cg.cs.tsinghua.edu.cn/people/~Yongjin/Yongjin.htm



(a) UE3D (b) SA3D (c) BRD (d) BRD of FCSS and CSS

(e) EV (f) Compactness (g) Runtime with respect to K (h) Peak memory without NCut

Fig. 5. Evaluation of ten representative methods and our methods (FCSS and streaming FCSS) on the SegTrack v2 dataset. Superpixels in all methods are unrelabeled. Due to its high computational cost, NCut is run at a fixed frame resolution of 240×160 downsampled from original videos and is not present in (h). Only FCSS achieves good performance on all seven metrics of UE3D, SA3D, BRD, EV, compactness, running time and peak memory. In particular, FCSS is $5\times$ to $10\times$ faster than TSP. The peak memory of FCSS is $22\times$ smaller than TSP and $7\times$ to $15\times$ smaller than TS-PPM. Similar performances are observed on the other three video datasets (BuffaloXiph, BVDS and CamVid), which are reported in Appendix.

where $V(x)$ is the voxel number in a segment x. Both Eqs. (24) and (25) take the average score from all ground-truth segments $\tilde{G}$. A small UE3D value means that very few voxels are leaked from ground-truth segments. The range of SA3D values is $[0, 1]$, where a larger value means a better over-segmentation result. BRD measures how well the ground-truth boundaries are successfully retrieved by the supervoxel boundaries. Denote the t-th frame's ground-truth segmentation as $\tilde{G}^t$, and the t-th frame's supervoxel segmentation as $\tilde{S}^t$. The BRD metric is defined as

$$BRD = \frac{1}{\sum_t \left| B(\tilde{G}^t) \right|} \sum_t \sum_{p \in B(\tilde{G}^t)} \min_{q \in B(\tilde{S}^t)} d(p, q) \qquad (26)$$

where $B(\cdot)$ returns the 2D boundaries in a frame, $d(\cdot, \cdot)$ measures Euclidean distance between two points, and $|\cdot|$ returns the number of pixels in a 2D boundary. As shown in Figures 5a-5c, TS-PPM and FCSS have good performance on UE3D, SA3D and BRD, demonstrating their ability to adhere to object boundaries. GBH and SWA are only good at BRD and SA3D, but not good at UE3D. CSS is only good at UE3D and SA3D, but not good at BRD. TSP and NCut are only good at UE3D, but not good at SA3D and BRD. GB and MeanShift are not good for all three metrics UE3D, SA3D and BRD. All three streaming methods have similar performance on SA3D, while streamFCSS and streamCSS are better in UE3D and streamGBH is better in BRD.

Explained variation (EV). EV is a standard metric that measures the color variations in supervoxels [26], [41], defined as

$$EV = \frac{\sum_{\tilde{s}_i \in \tilde{S}} (\mu(\tilde{s}_i) - \mu) \left| \tilde{s}_i \right|}{\sum_j (x_j - \mu)} \qquad (27)$$

where μ is the average color of all voxels in a video, $\mu(\tilde{s}_i)$ is the average color of the supervoxel $\tilde{s}_i$, and x_j is the color of the voxel j. The score range is in $[0, 1]$, where a larger value means a better representation (i.e., the color in each supervoxel is closer to homogeneity). As shown in Figure 5e, SWA has the largest EV. GBH, FCSS, TS-PPM are better than CSS and TSP, which are in turn better than other methods. For the three streaming methods, streamFCSS and streamCSS have similar performance and are better than streamGBH.

Compactness. It is a measure of shape regularity [44], defined as

$$C(\tilde{S}) = \sum_{\tilde{s}_i \in \tilde{S}} Q(\tilde{s}_i) \frac{\left| \tilde{s}_i \right|}{N}, \text{where } Q(\tilde{s}_i) = \frac{6\pi^{\frac{1}{2}} V(\tilde{s}_i)}{A(\tilde{s}_i)^{\frac{3}{2}}}, \qquad (28)$$

where $\tilde{S}$ is a given supervoxel over-segmentation as used in Eqs.(24) and (25), $A(\tilde{s}_i)$ and $V(\tilde{s}_i)$ are bounding surface area and volume of supervoxel $\tilde{s}_i$, respectively. In many real-world video applications, their solutions rely on minimizing an energy function defined on a spatiotemporal supervoxel graph in a video clip. The shape regularity of supervoxels has a direct influence on the complexity of this spatiotemporal supervoxel graph, and thus, affects the application performance. It was observed that compact supervoxels usually have better segmentation performance than non-compact ones. The larger the compactness value is, the more regular the shape of supervoxels is. As shown in Figure 5f, CSS and FCSS have the largest compactness values. StreamFCSS and streamCSS have similar performance and are better than streamGBH.

Computational cost. We record runtime and peak memory of all twelve methods. All methods are implemented in C or C++ except NCut (Matlab running with 8 threads) and TSP (Matlab with MEX). As shown in Figure 5g, GB, TS-PPM, CSS, FCSS and MeanShift are five fastest methods. As shown in Figures 5h, streamFCSS, streamCSS, FCSS and CSS are four methods that use smallest peak memory.

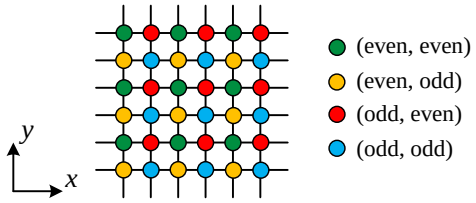


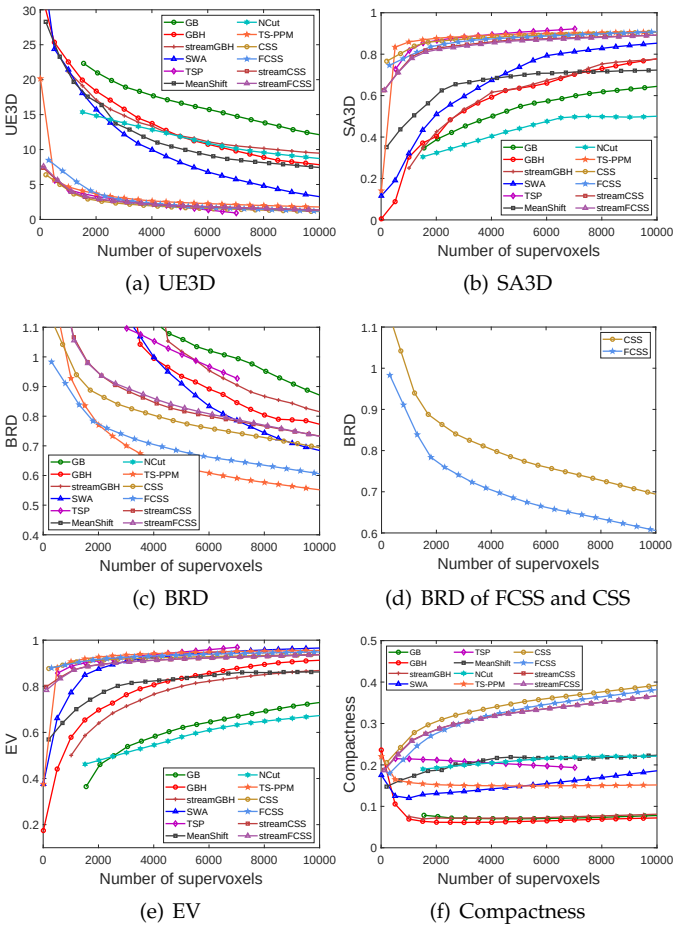
Fig. 6. For easy illustration, we present a superpixel example on a 2D image. Assume 8-connectivity. For an arbitrary image with arbitrary ground-truth segmentation, four *unrelabeled* superpixels are sufficient to achieve a perfect performance on the BRD metric, i.e., BRD $= 0$. These four superpixels are characterized by the parity of the coordinates (x, y) of image pixels; i.e., the green, yellow, red and blue superpixels consist of pixels with coordinates (even, even), (even, odd), (odd, even) and (odd, odd), respectively.

Three more metrics – mean size variation (MSV), temporal extent (TEX) and label consistency (LC) – are used in [41]. MSV and TEX measure the size variation and average temporal extent of all supervoxels in a video. Since our work advocates to adapt the size of supervoxels according to video content density, these two metrics are no longer suitable. LC is evaluated using ground-truth optical flow. As aforementioned, optical flow is only a preprocessing tool to video applications and may introduce extra error into supervoxel evaluation. In Section 7, we directly evaluate these supervoxel methods in two video applications.

Comparison between unrelabeled and relabeled supervoxels. In the original implementation of the seven methods in [41], a supervoxel label may be assigned to multiple disconnected regions. We call such supervoxels *unrelabeled*. Unrelabeled supervoxels may lead to unexpected performance on previous metrics; see Figure 6 for an (extreme) example. Then we further evaluate different supervoxel methods by relabeling supervoxels such that each supervoxel is a simply connected region and each voxel is assigned to exactly one supervoxel: we call such supervoxels *relabeled*. Relabeling supervoxels only affect the number of supervoxels. It does not affect the visual appearance and functionality of supervoxels: if one object/region can be represented by the union of a subset of unrelabeled supervoxels, it can also be represented by a subset of relabeled supervoxels. After supervoxel relabeling, the isolated fragments with less than τ voxels are merged with a randomly chosen neighboring supervoxel. The performance of twelve supervoxel methods after relabeling with $\tau = 5, 10, 50, 100$ are summarized in Figure A5 in Appendix and the results with $\tau = 50$ are shown in Figure 7. The results show that only FCSS, streamFCSS, CSS, streamCSS, TSP and TS-PPM are insensitive to relabeling. Meanwhile, only FCSS achieves good performance on all five metrics of UE3D, SA3D, BRD, EV and compactness.

Comparison with TSP and TS-PPM. FCSS has similar UE3D, SA3D and EV performance with TSP and TS-PPM. FCSS has similar BRD performance with TS-PPM and is better than TSP. FCSS is much better in compactness than both TSP and TS-PPM. FCSS and TS-PPM are $5\times$ to $10\times$ faster than TSP. The peak memory of FCSS is $22\times$ smaller than TSP and $7\times$ to $15\times$ smaller than TS-PPM. Furthermore, in Section 7, we present two video applications and show that FCSS achieves better results than TSP and TS-PPM.

Comparison with CSS. Both FCSS/streamFCSS and CSS/streamCSS use RCVT on video manifold $\mathcal{M}$. Thanks to the feature-aware strategy by forcing cell centroids away from video local boundaries, the FCSS method better fine tunes the cell boundaries to align with the video local boundaries than CSS. As shown in Figure 5d and Figure A1 in Appendix, FCSS outperforms CSS on the metrics of BRD, SA3D and EV, and has similar performance with CSS on UE3D. Meanwhile, FCSS is better than CSS in two novel video applications presented in Section 7.



(a) UE3D (b) SA3D (c) BRD (d) BRD of FCSS and CSS (e) EV (f) Compactness

Fig. 7. Evaluation of *relabeled* supervoxels on the SegTrack v2 dataset. Only FCSS achieves good performance on all five metrics of UE3D, SA3D, BRD, EV and compactness.

7 Applications

We evaluate the performance of various supervoxels in the following two video applications. To faithfully compare different supervoxel methods, we use their original settings, i.e., supervoxels are unrelabeled.

Foreground propagation in video. Given the first frame with manual annotation for the foreground object, a novel approach is proposed in [14] to propagate the foreground region through time, with the aid of supervoxels to guide its estimates towards long-range coherent regions. Youtube-Objects dataset [29] (126 videos with 10 object classes) with foreground ground-truth, is used to perform a quantitative assessment. Seven representative methods (GB, GBH, streamGBH, MeanShift, TSP, TS-PPM and CSS) and our FCSS method are compared. NCut is not compared due to its high computational cost. SWA is not compared since

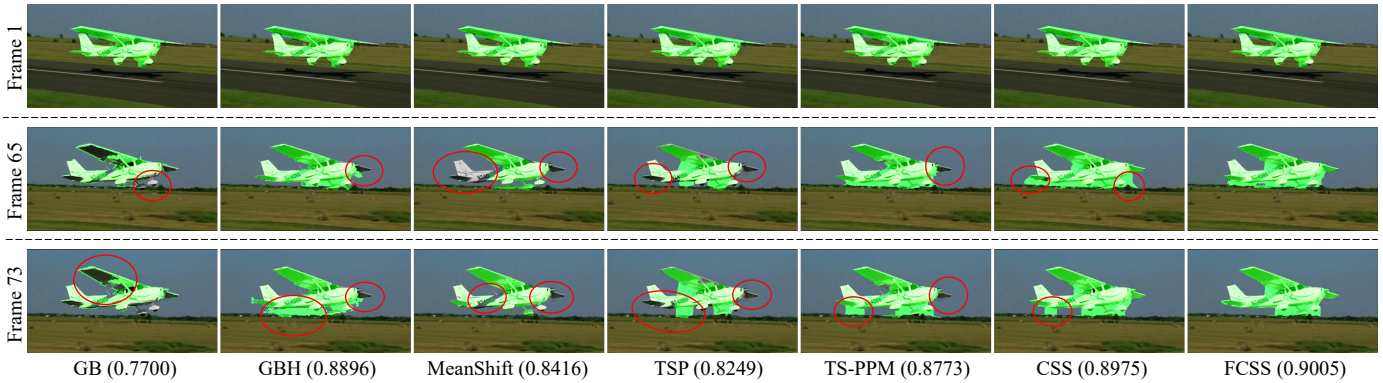


Fig. 9. Foreground propagation results of seven supervoxel methods on one example in Youtube-objects dataset [29]. Three representative frames are selected. The foreground masks are shown in green. The incorrectly labeled areas are circled in red. The average F measure for each example video is shown in the bracket below three frames. The value of the F measure ranges in $[0, 1]$, and larger values mean better results.

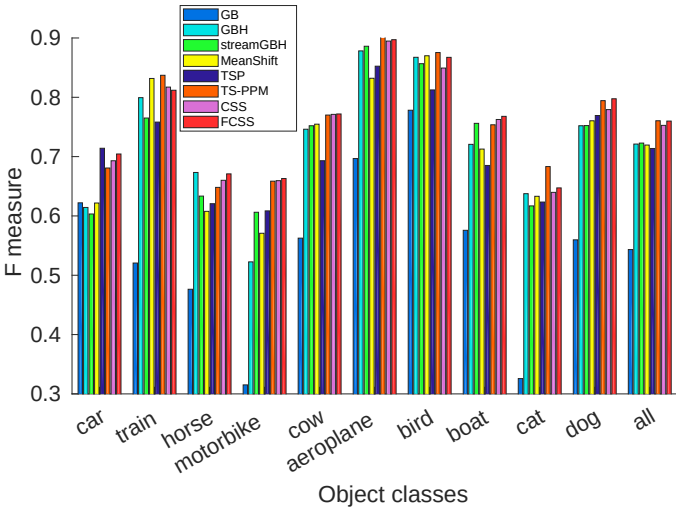


Fig. 8. The average F measures of different supervoxel results on Youtube-Objects Dataset. The results are plotted per object class and each object class contains several video sequences. Larger F measure values mean better foreground propagation results. The results show that FCSS, TS-PPM and CSS are the top three methods overall in the whole dataset. FCSS has better results than TS-PPM in six classes (car, horse, motorbike, cow, boat, dog) and has better results than CSS in nine classes (except for the train class).

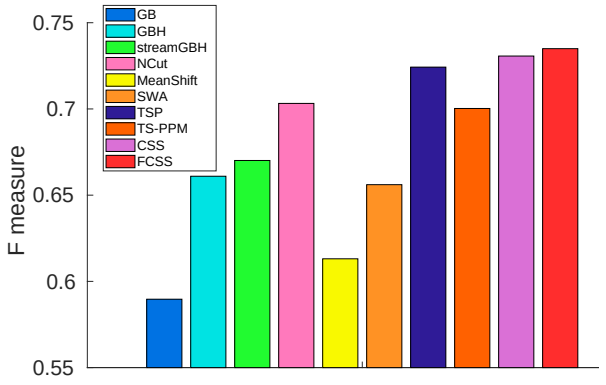


Fig. 10. Average F measures in the spatiotemporal closure application. The results are averaged on Stein et al. [35] dataset. Our FCSS method achieves the best average F measure among all ten methods.

there are many long videos in this dataset and SWA requires huge memory. The average F measures of 10 classes are summarized in Figure 8. F measure values range in $[0, 1]$ and larger values mean better results. These results show that FCSS, TS-PPM and CSS are top three methods in the overall F-measure (i.e., including all object classes). FCSS has better results than TS-PPM in six classes and has the same performance in the overall F-measure. FCSS achieves better results than CSS in nine classes, with an overall better F-measure. Some qualitative results are illustrated in Figure 9.

Optimal video closure by supervoxel grouping. Levinshtein et al. [17] propose a novel foreground object segmentation method which does not need manual annotation on the frame. The idea is to detect spatiotemporal closure for separating an object from background. A novel framework for efficiently searching spatiotemporal closure is proposed by finding subsets of supervoxels such that the contour of union of these supervoxels has strong boundary support in the video. The dataset of Stein et al. [35] in which each sequence has a ground truth segmentation mask, is used to perform a quantitative assessment. Nine representative methods (GB, GBH, streamGBH, NCut, MeanShift, SWA, TSP, TS-PPM and CSS) and our FCSS method are compared. The average F measures across all sequences are summarized in Figure 10. Some qualitative results are illustrated in Figure 11. These results show that FCSS results achieve the best spatiotemporal closure performance.

8 Conclusion

In this paper, we introduce feature-aware content-sensitive supervoxels (FCSS) that have three characteristics: (1) they are regularly-shaped 3D primitive volumes, (2) they are well aligned with local object/region boundaries in video, and (3) they are typically smaller and shorter in content-dense regions (i.e., with high variation of appearance and/or motion), and larger and longer in content-sparse regions. We propose a simple yet efficient algorithm to compute FCSSs by computing a uniform tessellation on the video manifold $\mathcal{M}$ with an elaborate average boundary distance, such that the cell boundaries of obtained uniform tessellation well align with local video boundaries. Our algorithm is easily extended to a stream version for handling long videos. In addition to its easy implementation, our algorithm is theoretically an $(O(1), O(1))$-approximation. Experimental results show that FCSS is the only method that can achieve good performance in all the metrics (i.e., UE3D, SA3D, BRD EV, compactness, running time and peak memory) and is insensitive to supervoxel relabeling. Two video applications are presented, demonstrating that the proposed FCSS

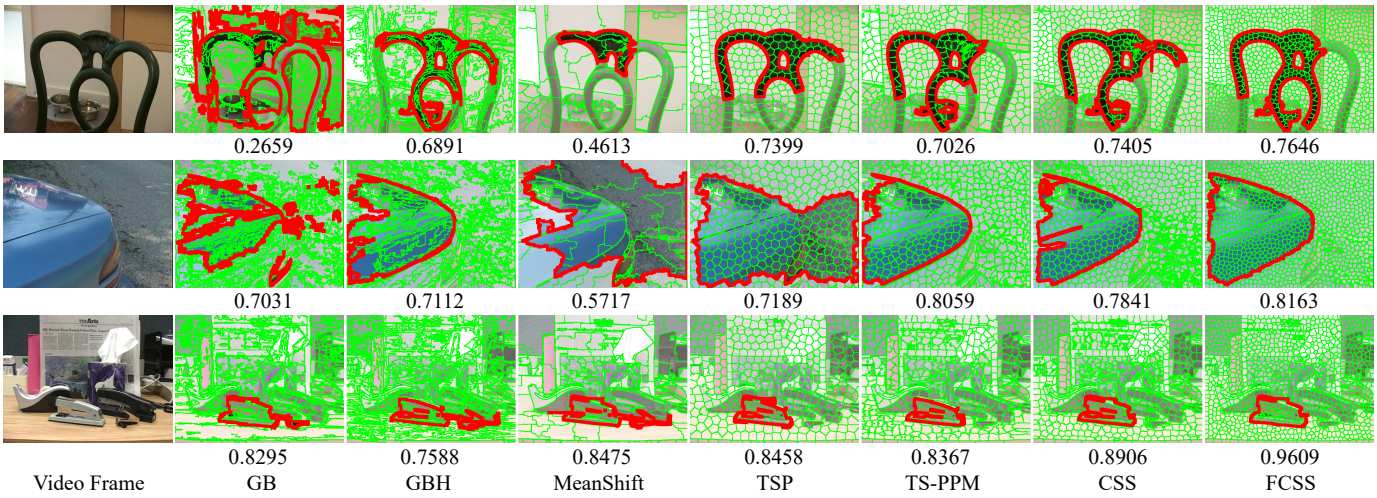


Fig. 11. Spatiotemporal closure results of seven supervoxel methods on three examples in Stein et al. dataset [35]. The optimal closure contours are shown in red, and the boundaries of supervoxels are shown in green. One representative frame is illustrated for each video. The F measure value for each spatiotemporal closure is shown below each frame; the range of the F measure values is $[0, 1]$, and larger values mean better results.

method can simultaneously achieve the best performance with respect to various metrics.

REFERENCES

[1] N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, NIPS'09, pages 10–18, 2009.

[2] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, 2007.

[3] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and recognition using structure from motion point clouds. In *European Conference on Computer Vision*, ECCV'08, pages 44–57. Springer, 2008.

[4] Y. Cai and X. Guo. Anisotropic superpixel generation based on mahalanobis distance. *Computer Graphics Forum (Pacific Graphics 2016)*, 35(7):199–207, 2016.

[5] J. Chang, D. Wei, and J. W. Fisher III. A video representation using temporal superpixels. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 2051–2058, 2013.

[6] A. Y. Chen and J. J. Corso. Propagating multi-class pixel labels throughout video frames. In *Proceedings of Western New York Image Processing Workshop*, 2010.

[7] J. J. Corso, E. Sharon, S. Dube, S. El-Saden, U. Sinha, and A. L. Yuille. Efficient multilevel brain tumor segmentation with integrated Bayesian model classification. *IEEE Trans. Med. Imaging*, 27(5):629–640, 2008.

[8] Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[9] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[10] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the nyström method. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):214–225, 2004.

[11] C. C. Fowlkes, S. J. Belongie, and J. Malik. Efficient spatiotemporal grouping using the nyström method. In *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR '01, pages 231–238, 2001.

[12] F. Galasso, N. S. Nagaraja, T. J. Cárdenas, T. Brox, and B. Schiele. A unified video segmentation benchmark: Annotation, metrics and analysis. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, pages 3527–3534, 2013.

[13] M. Grundmann, V. Kwatra, M. Han, and I. A. Essa. Efficient hierarchical graph-based video segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'10, pages 2141–2148, 2010.

[14] S. D. Jain and K. Grauman. Supervoxel-consistent foreground propagation in video. In *13th European Conference on Computer Vision*, ECCV'14, pages 656–671, 2014.

[15] Y. Ke, R. Sukthankar, and M. Hebert. Spatio-temporal shape and flow correlation for action recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'07, 2007.

[16] S. Lee, W. Jang, and C. Kim. Temporal superpixels based on proximity-weighted patch matching. In *IEEE International Conference on Computer Vision*, ICCV'17, pages 3630–3638, 2017.

[17] A. Levinshtein, C. Sminchisescu, and S. J. Dickinson. Optimal image and video closure by superpixel grouping. *International Journal of Computer Vision*, 100(1):99–119, 2012.

[18] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 31(12):2290–2297, 2009.

[19] C. Li, L. Lin, W. Zuo, S. Yan, and J. Tang. SOLD: sub-optimal low-rank decomposition for efficient video segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'15, pages 5519–5527, 2015.

[20] F. Li, T. Kim, A. Humayun, D. Tsai, and J. M. Rehg. Video segmentation by tracking many figure-ground segments. In *Proceedings of the IEEE International Conference on Computer Vision*, ICCV'13, pages 2192–2199, 2013.

[21] Y. Liang, J. Shen, X. Dong, H. Sun, and X. Li. Video supervoxels using partially absorbing random walks. *IEEE Trans. Circuits Syst. Video Techn.*, 26(5):928–938, 2016.

[22] M.-Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa. Entropy rate superpixel segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '11)*, pages 2097–2104, 2011.

[23] Y.-J. Liu, C. Yu, M. Yu, and Y. He. Manifold SLIC: a fast method to compute content-sensitive superpixels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, pages 651–659, 2016.

[24] Y.-J. Liu, M. Yu, B.-J. Li, and Y. He. Intrinsic manifold SLIC: A simple and efficient method for computing content-sensitive superpixels. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(3):653–666, 2018.

[25] J. Lu, R. Xu, and J. J. Corso. Human action segmentation with hierarchical supervoxel consistency. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'15, pages 3762–3771, 2015.

[26] A. P. Moore, S. Prince, J. Warrell, U. Mohammed, and G. Jones. Superpixel lattices. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR '08, 2008.

[27] D. Oneata, J. Revaud, J. Verbeek, and C. Schmid. Spatio-temporal object detection proposals. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *13th European Conference on Computer Vision*, ECCV'14, pages 737–752, 2014.

[28] S. Paris and F. Durand. A topological approach to hierarchical segmentation using mean shift. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'07, 2007.

[29] A. Prest, C. Leistner, J. Civera, C. Schmid, and V. Ferrari. Learning

object class detectors from weakly annotated video. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'12, pages 3282–3289. IEEE, 2012.

[30] M. Reso, J. Jachalsky, B. Rosenhahn, and J. Ostermann. Temporally consistent superpixels. In *IEEE International Conference on Computer Vision (ICCV '13)*, pages 385–392, 2013.

[31] M. Reso, J. Jachalsky, B. Rosenhahn, and J. Ostermann. Occlusion-aware method for temporally consistent superpixels. *IEEE Trans. Pattern Anal. Mach. Intell.*, DOI:10.1109/TPAMI.2018.2832628, 2019.

[32] E. Sharon, A. Brandt, and R. Basri. Fast multiscale image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '00, pages 1070–1077, 2000.

[33] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt. Hierarchy and adaptivity in segmenting visual scenes. *Nature*, 442(7104):810–813, 2006.

[34] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[35] A. N. Stein, D. Hoiem, and M. Hebert. Learning to find object boundaries using motion cues. In *IEEE 11th International Conference on Computer Vision, ICCV 2007*, pages 1–8, 2007.

[36] D. Sun, S. Roth, and M. J. Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[37] P. Sundberg, T. Brox, M. Maire, P. Arbeláez, and J. Malik. Occlusion boundary detection and figure/ground assignment from optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'11, pages 2233–2240. IEEE, 2011.

[38] O. Veksler, Y. Boykov, and P. Mehrani. Superpixels and supervoxels in an energy optimization framework. In *Europeon Conference on Computer Vision (ECCV 2010)*, pages 211–224, 2010.

[39] P. Wang, G. Zeng, R. Gan, J. Wang, and H. Zha. Structure-sensitive superpixels via geodesic distance. *International Journal of Computer Vision*, 103(1):1–21, 2013.

[40] D. Wei. A constant-factor bi-criteria approximation guarantee for k-means++. In *Annual Conference on Neural Information Processing Systems (NIPS) 2016*, pages 604–612, 2016.

[41] C. Xu and J. J. Corso. Libsvx: A supervoxel library and benchmark for early video processing. *International Journal of Computer Vision*, 119(3):272–290, 2016.

[42] C. Xu, C. Xiong, and J. J. Corso. Streaming hierarchical video segmentation. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part VI*, ECCV'12, pages 626–639, 2012.

[43] R. Yi, Y. Liu, and Y. Lai. Content-sensitive supervoxels via uniform tessellations on video manifolds. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'18, pages 646–655, 2018.

[44] R. Yi, Y. Liu, and Y. Lai. Evaluation on the compactness of supervoxels. In *IEEE International Conference on Image Processing, ICIP'18*, pages 2212–2216, 2018.

[45] C.-P. Yu, H. Le, G. Zelinsky, and D. Samaras. Efficient video segmentation using parametric graph partitioning. In *IEEE International Conference on Computer Vision*, ICCV'15, pages 3155–3163, 2015.