GPU-Based Supervoxel Generation With a Novel Anisotropic Metric

Xiao Dong, Zhonggui Chen, Yong-Jin Liu, *Senior Member, IEEE*, Junfeng Yao, and Xiaohu Guo, *Member, IEEE*

***Abstract*— Video over-segmentation into supervoxels is an important pre-processing technique for many computer vision tasks. Videos are an order of magnitude larger than images. Most existing methods for generating supervovels are either memory- or time-inefficient, which limits their application in subsequent video processing tasks. In this paper, we present an anisotropic supervoxel method, which is memory-efficient and can be executed on the graphics processing unit (GPU). Therefore, our algorithm achieves good balance among segmentation quality, memory usage and processing time. In order to provide accurate segmentation for moving objects in video, we use the optical flow information to design a brand new non-Euclidean metric to calculate the anisotropic distances between seeds and voxels. To efficiently compute the anisotropic metric, we adjust the classic jump flooding algorithm (which is designed for parallel execution on the GPU) to generate anisotropic Voronoi tessellation in the combined color and spatio-temporal space. We evaluate our method and the representative supervoxel algorithms for their capability on segmentation performance, computation speed and memory efficiency. We also apply supervoxel results to the application of foreground propagation in videos to test the performance on solving practical problems. Experiments show that our algorithm is much faster than the existing methods, and achieves good balance on segmentation quality and efficiency.**

***Index Terms*— Supervoxels, video segmentation, anisotropic metric, jump flooding algorithm, GPU.**

I. Introduction

As AN efficient pre-processing technique, superpixel segmentation [1], [11], [12], [24], [26] technology had been widely used in image processing. By stacking the image frames along time axis, many over-segmentation algorithms have been extended from image to video. Video over-segmentation aims to group perceptually similar spatio-temporal (3D) voxels into meaningful atomic patches called supervoxels. Supervoxels well preserve the important structure of video content and can greatly reduce the number of computational units for downstream computer vision applications (e.g., [13], [22], [27]). Furthermore, the design principle of supervoxels can also inspire the hardware design of new camera [30] and the new methodology in video visualization [4].

The quality of supervoxels directly affects the results of subsequent processing task. To well preserve the video content, supervoxels are usually expected to have good performance on multiple aspects [42], such as segmentation accuracy, color homogeneity, shape compactness, processing efficiency and so on. The supervoxel boundaries should adhere well to the object, and every supervoxel should overlap with only one object, which encourages a high-degree of color homogeneity within supervoxels. Meanwhile, the supervoxels are expected to be compact and uniform in the space-time domain, especially on the flat background. The above properties should be maintained with as few supervoxels as possible. In addition, Supervoxel generation should be efficient on running time and memory cost, and should not reduce the achievable performance of its downstream applications.

Many methods have been proposed for computing video over-segmentation in recent years. Except for the temporal superpixel methods, most offline supervoxel methods load all the voxels of video into the memory at once. Considering the limitation of memory, the streaming version [43], [45] are proposed to handle long videos by allowing a small number of frames to be loaded into the memory at any given time.

By meeting the requirement of segmentation quality, recent researches focused on solving the problem of insufficient memory by proposing the streaming version of supervoxel algorithms. However, time consuming is another problem that

Manuscript received April 30, 2021; revised September 17, 2021; accepted October 6, 2021. Date of publication October 22, 2021; date of current version October 28, 2021. The work of Zhonggui Chen was supported in part by the National Natural Science Foundation of China under Grant 61972327 and Grant 61872308, in part by the Fundamental Research Funds for the Central Universities under Grant 20720190011, and in part by the Open Project Program of State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, under Grant VRLAB2021B01. The work of Yong-Jin Liu was supported by the National Natural Science Foundation of China under Grant 61725204. The work of Junfeng Yao was supported in part by the National Natural Science Foundation of China under Grant 62072388, in part by the Collaborative Project Fund of Fuzhou-Xiamen-Quanzhou Innovation Zone under Grant 3502ZCQXT202001, in part by the Industry Guidance Project Foundation of Science Technology Bureau of Fujian Province in 2020 under Grant 2020H0047, in part by the Natural Science Foundation of Science Technology Bureau of Fujian Province in 2019 under Grant 2019J01601, and in part by the Creation Fund Project of Science Technology Bureau of Fujian Province in 2019 under Grant 2019C0021. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Abdesselam S. Bouzerdoum. *(Corresponding authors: Zhonggui Chen; Junfeng Yao; Xiaohu Guo.)*

Xiao Dong and Zhonggui Chen are with the School of Informatics, Xiamen University, Xiamen 361005, China (e-mail: dongxiao0401@gmail.com; chenzhonggui@xmu.edu.cn).

Yong-Jin Liu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: liuyongjin@tsinghua.edu.cn).

Junfeng Yao is with the School of Film, Xiamen University, Xiamen 361005, China (e-mail: yao0010@xmu.edu.cn).

Xiaohu Guo is with the Department of Computer Science, University of Texas at Dallas, Dallas, TX 75080 USA (e-mail: xguo@utdallas.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TIP.2021.3120878, provided by the authors.

Digital Object Identifier 10.1109/TIP.2021.3120878

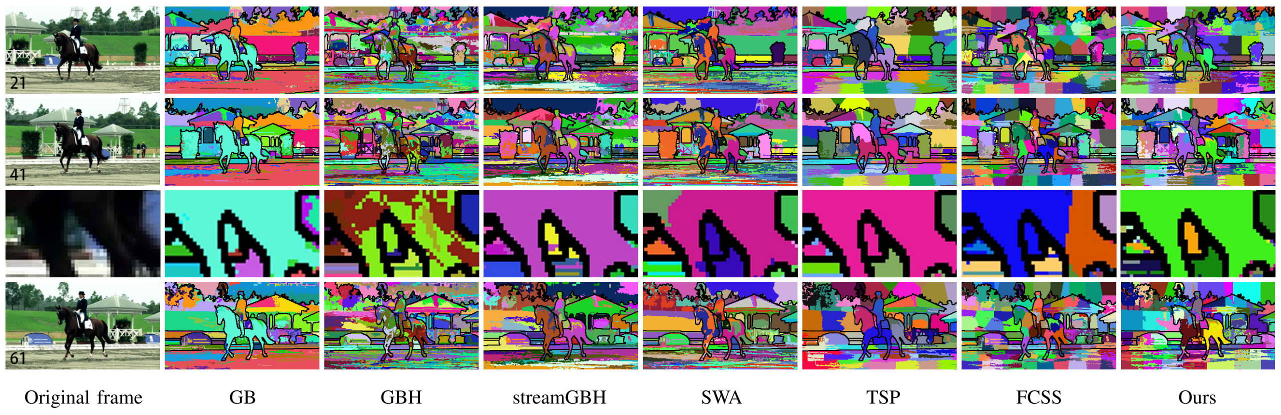


Fig. 1. Segmentation results of representative algorithms, such as GB [16], GBH [20], steamGBH [43], SWA [10], [35], [36], TSP [8], FCSS [46] and ours. The magnified details in the third row show that our method is sensitive to color changes and achieves better boundary adherence. For 500 supervoxels, our processing speed is at least 20 times faster than FCSS and 100 times faster than TSP. See Section V for detailed comparison results.

needs to be solved with great demand. In fact, the longtime consumption hinders the wide application of supervoxels as a pre-processing technology in many video processing fields, especially for some tasks that require real-time performance. Usually, the segmentation quality, processing time and memory usage are conflicting requirements, and in our work we hope to achieve a good compromise between these targets. We propose an anisotropic supervoxel algorithm based on GPU implementation to over-segment the video with good quality, while greatly improving the processing speed. For a video with 51 frames and 7 million pixels, our method produces the result in 2 seconds, while the advanced methods such as TSP [8] and FCSS [46], producing results with similar quality, require more than one minute. Other supervoxel methods such as SLIC [1] have similar processing speed, however, our method generates much better segmentation results as compared to them (see Sec. V for more comparison details).

Our method is a seed-based supervoxel segmentation, and we design an anisotropic distance metric for each seed to classify the voxels. Fig. 1 shows supervoxel results of different methods. For most existing methods [8], [44], [46], supervoxels are costly to obtain on memory and time. We show that it's possible to design a parallel algorithm executed on GPU for supervoxel segmentation, which makes our algorithm far ahead in efficiency. The main contributions of our work are as follows:

- For accurate segmentation of objects in motion, we design a new anisotropic distance metric to calculate the distances between seeds and voxels, which uses the motion information to compensate for the prolonged distance of moving objects' pixels between frames, thus provides more accurate object capturing capability for supervoxel.
- We propose a simple yet effective seed initialization strategy that places more seeds in object area to segment the object more accurately.
- In order to ensure the segmentation quality and processing efficiency at the same time, we design an anisotropic supervoxel algorithm that can be implemented in parallel on the GPU, by extending the jump flooding algorithm [33], [34]. While most existing algorithms are difficult to be implemented on the GPU.

II. Related Work

Many methodologies have been proposed to compute superpixels and supervoxels. For image segmentation, mainstream theories include graph theory [12], clustering based on seeds [24], [37], [38], etc. Most superpixel algorithms can quickly and accurately generate over-segmentation results, and some seed-based superpixel algorithms [1], [38] even achieve real-time processing speed. These methods can be naturally extended to video segmentation, but as the number of pixels increases, the processing speed and memory footprint also increase substantially. We will mainly focus on the current research status of video over-segmentation in this section.

The supervoxels for video over-segmentation can be classified into two categories: supervoxels and temporal superpixels. Supervoxel methods [16], [20], [29], [36], [43], [44] treat the video as a volume and over-segment it in 3D space. Temporal superpixel methods [7], [8], [32] first generate superpixels on one frame and build the temporal correspondences between adjacent frames.

Supervoxel methods can be developed based on multiple principles, such as mode-seeking, graph cut, seed-based clustering and so on. Mean shift is a mode-seeking method. Paris and Durand [29] applied Morse theory to interpret mean shift to decompose the feature space into density modes. Paris *et al.* [28] proposed the hierarchical mean shift by a streaming approach to improve the efficiency of isotropic mean-shift methods. Felzenszwalb and Huttenlocher [16] proposed a graph-based segmentation method (GB), which is suitable for the extension to spatio-temporal video segmentation. Based on GB, Groundmann *et al.* [20] proposed a hierarchical graph-based method (GBH), which constructs a bottom-up hierarchical tree structure of the region graphs. Xu *et al.* [43] extended GBH to a streaming version (streamGBH) to handle arbitrarily long videos, by allowing a small number of frames to be loaded into memory. Nyström Normalized Cut (NCut) was proposed by Fowlkes *et al.* [17], [18], which is based on a technique for the numerical solution of eigen function problems known as the Nyström approximation. SWA [10], [35], [36] is another normalized cut approach that integrates the model-aware affinities into the multilevel segmentation by the weighted aggregation algorithm. SWA produces a hierarchy of segmentations and can be extended to a supervoxel method in 3D. Achanta *et al.* [1] proposed a popular seed-based clustering algorithm, SLIC, which adopts the Lloyd iteration to optimize the Voronoi tessellation in the combined color and spatio-temporal space. Inspired by SLIC, Yi *et al.* [45] proposed the Yi-CSS method that maps the video to a 3-dimensional manifold embedded in the R^6 space. They also proposed qd-CSS [44] which adopts a fast queue-based graph distance to compute the geodesic centroidal Voronoi tessellation, and FCSS [46] that tries to find a uniform tessellation whose cell boundaries well align with objects among all possible centroidal Voronoi tessellations on the video manifold.

Temporal superpixel methods compute a segmentation on the first frame, and propagate it to the subsequent frames. Chang *et al.* proposed the TSP [8] algorithm that adopts a Gaussian process to model the motion for the streaming segmentation. Inspired by SLIC method, Reso *et al.* [32] proposed an energy-minimizing clustering method by utilizing a hybrid clustering strategy working in a global color subspace and local spatial subspaces. Wang *et al.* proposed a superpixel method based on geodesic distance [41], and conducted the superpixel flow on video based on Lucas–Kanade algorithm to find stable correspondent centers between adjacent frames. Cai *et al.* [7] proposed a bottom-up merging superpixel method (GGM) which first generates a coarse image segmentation and then swaps the boundary pixels to improve the compactness. GGM can be extended to generate temporal superpixels. Among the above methods, TSP achieves the state-of-the-art performance and we choose it as the representative temporal superpixel method in this paper.

The temporal superpixel methods are able to process arbitrarily long videos as they load a small number of frames into memory at once. On the contrary, most of the supervoxel methods are off-line methods that need to load all frames at first. To solve this problem, some streaming versions [43], [46] were proposed to deal with long videos. In this paper, we propose a seed-based anisotropic supervoxel method. We make use of the optical flow information to design the anisotropic metric for supervoxels, which are consistent with the motion direction of the object. The anisotropic metric can compensate for the prolonged distance (by making them shorter) for pixels in the motion direction, thus the label of moving object will remain the same in several consecutive frames. We also extend our method to the streaming version to handle long videos that cannot be loaded into memory at once.

III. Preliminaries

Our algorithm is based on a variant of SLIC [1] method and the jump flooding method [33], [34] on the GPU, which we briefly summarize below.

A. SLIC for Videos

SLIC method is a classic image segmentation algorithm that can efficiently generate compact superpixels. Given a video $\mathbb{I}$ with n voxels, SLIC can be easily extended to handle 3D supervoxels by introducing the time dimension to the spatial term. Denote the spatio-temporal term of a voxel v as $\mathbf{p}(v) = [x(v), y(v), t(v)]^\top$, where $[x(v), y(v)]$ are the 2D coordinates in a frame and $t(v)$ is the frame index. Denote the color term of v as $\mathbf{c}(v) = [l(v), a(v), b(v)]^\top$ in the CIELAB space. The basic information of a voxel is a 6-dimensional vector $\mathbf{v}(v) = [\mathbf{p}(v)^\top, \mathbf{c}(v)^\top]^\top$. The desired number of supervoxel is k. Starting from k uniformly sampled seeds $S = \{s_i\}_{i=1}^k$, SLIC uses an adaptation of k-means algorithm to optimize the segmentation. SLIC measures the distance between a voxel v and a seed s_i in the combined space using a normalized Euclidean distance defined as:

$$D(v, s_i) = \sqrt{\left(\frac{d_p}{N_p}\right)^2 + \left(\frac{d_c}{N_c}\right)^2}, \tag{1}$$

where N_p and N_c are two weights to balance the relative importance between the color factor and the spatio-temporal factor, and

$$\begin{aligned} d_p &= \|\mathbf{p}(v) - \mathbf{p}(s_i)\|_2 \\ &= \sqrt{(x(v) - x(s_i))^2 + (y(v) - y(s_i))^2 + (t(v) - t(s_i))^2}, \\ d_c &= \|\mathbf{c}(v) - \mathbf{c}(s_i)\|_2 \\ &= \sqrt{(l(v) - l(s_i))^2 + (a(v) - a(s_i))^2 + (b(v) - b(s_i))^2}. \end{aligned}$$

Different from the conventional k-means method, SLIC computes the distances from the seed s_i to voxels within a local search range $2L \times 2L \times 2L$, where $N_p = L = \sqrt[3]{n/k}$. Once each voxel has been associated to the nearest seed in one iteration, we get the segmentation $\mathbb{I} = \{C_i\}_{i=1}^k$. Then it updates the seed s_i to be the mean vector of the voxels belonging to that supervoxel C_i as shown below:

$$\mathbf{v}(s_i) = \bar{\mathbf{v}}(C_i) = \sum_{v \in C_i} \mathbf{v}(v)/|C_i|. \tag{2}$$

The assignment and update steps can be applied iteratively, and computational results show that the SLIC method converges in several iterations and generates compact supervoxels.

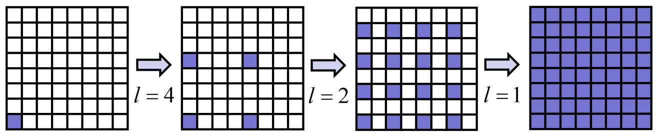


Fig. 2. Jump flooding to propagate the content of a seed at the lowest left corner by halving step length [33].

B. Jump Flooding Algorithm on the GPU

Jump flooding algorithm (JFA) [33] and its variants [34] are an algorithmic paradigm for computing discrete Voronoi diagrams on the GPU. Fig. 2 is an illustration that shows the main idea of JFA. Given an image of size $r \times r$ with a seed located at the lowest left corner, JFA shows an efficient way to flood the content of seed in $\log r$ rounds to all the other grid points, by halving the step length in each round. Here, $r = 8$, and there are $\log r$ rounds with step lengths of $r/2, r/4, \ldots, 1$. At the beginning, each point with a seed s records $< s, f(s) >$ to indicate its closest seed found so far is s, whereas the other points record $< null, null >$. Here $f(s)$ can be the position or other information of the seed s. In a round with step length of l, each point $[x, y]$ queries the information of other points at $[x+i, y+j]$, where $i, j \in \{-l, 0, l\}$. Among these 8 received (plus its current closest seed), the point $[x, y]$ decides which seed, say s^*, is its closest seed found so far, and updates its record as $< s^*, f(s^*) >$ if needed.

JFA can be implemented using OpenGL GLSL with fragment program support. It adopts a so-called ping-pong buffer with two buffers alternating as input and output in consecutive rounds. At the end of $\log r$ rounds, the buffer records the closest seed for each point, which is the approximation of Voronoi diagram. The errors due to the differences between the approximation and the actual Voronoi diagram are hardly noticeable to the naked eye in experiments.

In this paper, we extend the JFA framework for our anisotropic supervoxel segmentation, by introducing a new distance metric in information passing of seeds. The experiments show that our method generates more accurate results with much faster speed than existing methods. Please see Sec. IV-E for the extension of JFA for superpixels and supervoxels.

IV. GPU-Based Anisotropic Supervoxel

In this section, we present the proposed supervoxel method using a motion-based anisotropic distance metric, and its parallel implementation on the GPU.

A. Motivation

Our idea is to introduce the motion information into the seed-based clustering framework of SLIC [1], so that the object segmentation quality can be further improved. The supervoxel generated by SLIC can achieve uniform segmentaion, but it is not accurate enough in object detection. A supervoxel C_i consists of voxels that satisfy the following condition: $C_i = \{v \in \mathbb{I} | D(v, s_i) < D(v, s_j), \forall j = 1, \dots k, j \neq i\}$. Here SLIC only makes use Euclidean distance of the color and position between voxels and seeds, discarding the motion information, which is not optimal due to the non-negligible motions and occlusion/disocclusion. SLIC is unable to segment the objects in motion very well, especially for the cases that the color difference between moving objects and background is not obvious. Using Fig. 3 as an example, the pseudo-color images are superpixels induced by clipping supervoxels on a frame. We generate both the isotropic supervoxels and anisotropic supervoxels using JFA technique on the GPU with the same initialization of seeds, without enforcing the connectivity of supevoxels. Let us focus on the skating man in white clothes, which is magnified in the second column. The color difference between the man's clothes and the background is small, making it difficult to accurately segment the object from the background. The images in the third column show results of isotropic supevoxels using the Euclidean distance defined in Eq. (1). From the magnified details in the fourth column we observe that the isotropic supervoxels fail to detect the neck and shoulders of the man. On the other hand, the optical flow technique is able to detect the motion of objects, which helps the anisotropic supervoxels to distinguish objects from background as shown in the last column.

In order to capture the motion of objects, we make use of the optical flow field to design a non-Euclidean metric to measure the distances between voxels and seeds. Suppose the object is in motion for several consecutive frames, and in the current frame the nearest seed of the voxel v is s. Due to the motion of v, the Euclidean distance $D(v, s)$ in the next frame is getting larger. However, we hope that the seed s could capture the object longer within its search range. That is to find a non-Euclidean metric to keep the distance between the seed s and v rather short, so that the nearest seed for v is still s in subsequent frames. We design a specific anisotropic metric for each seed according to the motion information around it, and the seeds are able to capture the motion of different objects with the aid of anisotropic distance metric.

B. Objective Function

Using the Euclidean metric to calculate the distance between seeds and voxels is equivalent to having a spherical iso-distance surface centered at the seed point, and all voxels intersecting the spherical surface are equidistant from the seed. In terms of the non-Euclidean metric, the iso-distance surface becomes an ellipsoidal surface centered at the seed. If the long axis direction of the ellipsoid could be consistent with the direction of the motion around the seed, the seed could capture objects much easier. Motivated by this idea, we modified the distance based on a metric tensor defined per supervoxel:

$$\begin{aligned} D_M(v, s_i) = \lambda_c \|\mathbf{c}(v) - \mathbf{c}(s_i)\|^2 \\ +\lambda_p((\mathbf{p}(v) - \mathbf{p}(s_i))^\top \mathbf{M}_i (\mathbf{p}(v) - \mathbf{p}(s_i))). \end{aligned} \quad (3)$$

Here, $\mathbf{M}_i$ is a positive definite matrix which stretches or compresses the equidistant sphere according to the motion, so that the main direction of the deformed equidistant surface is consistent with the motion direction. In our formulation, each supervoxel C_i is defined by a center s_i and a metric tensor $\mathbf{M}_i \in \mathbb{R}^{3\times 3}$. The seeds $S = \{s_i\}_{i=1}^k$ and metrics $M = \{\mathbf{M}_i\}_{i=1}^k$ are themselves variables of the functional:

$$E(\{s_i\}_{i=1}^k, \{\mathbf{M}_i\}_{i=1}^k, \{C_i\}_{i=1}^k) = \sum_{i=1}^{k} \sum_{v \in C_i} D_M(v, s_i). \quad (4)$$

We constrain the matrix $\mathbf{M}_i$ to be a positive semidefinite (PSD) matrix, so our objective function is:

$$\begin{aligned} &\min \; E(\{s_i\}_{i=1}^k, \{\mathbf{M}_i\}_{i=1}^k, \{C_i\}_{i=1}^k), \\ &\text{s.t. } \mathbf{M}_i \in PSD. \end{aligned} \quad (5)$$

C. Anisotropic Supervoxel Algorithm

We demonstrate the workflow of the anisotropic supervoxel segmentation in Fig. 4. First, we load video data to GPU, and calculate the forward and backward optical flow fields, which are utilized to calculate the anisotropic metric of the seed. Then we adopt an optimization strategy similar to the classic Lloyd method [25] to obtain the segmentation result. The anisotropic supervoxels will be optimized by iterating following two steps:

1) By fixing the seeds S and metrics M, compute the supervoxel segmentation by jump flooding.
2) For each supervoxel C_i, update its seed s_i to be the centroid and compute the corresponding metric $\mathbf{M}_i$.

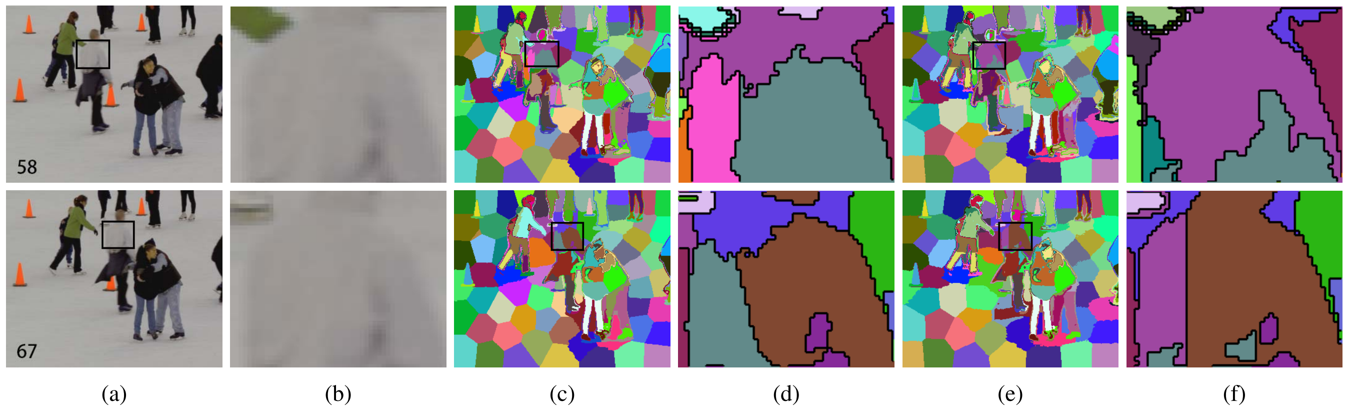


Fig. 3. Comparison of isotropic (SLIC) and anisotropic (our method) supervoxels on moving objects with same initialization. (a): original frames; (b): magnified details of object (the man in white clothes); (c): isotropic supervoxels (SLIC); (d): magnified details of object in (c), the isotropic supervoxels fail to detect the boundaries of the shoulders and neck; (e): anisotropic supervoxels (ours); (f): magnified details of object in (e), the anisotropic supervoxels detect the object more accurately.

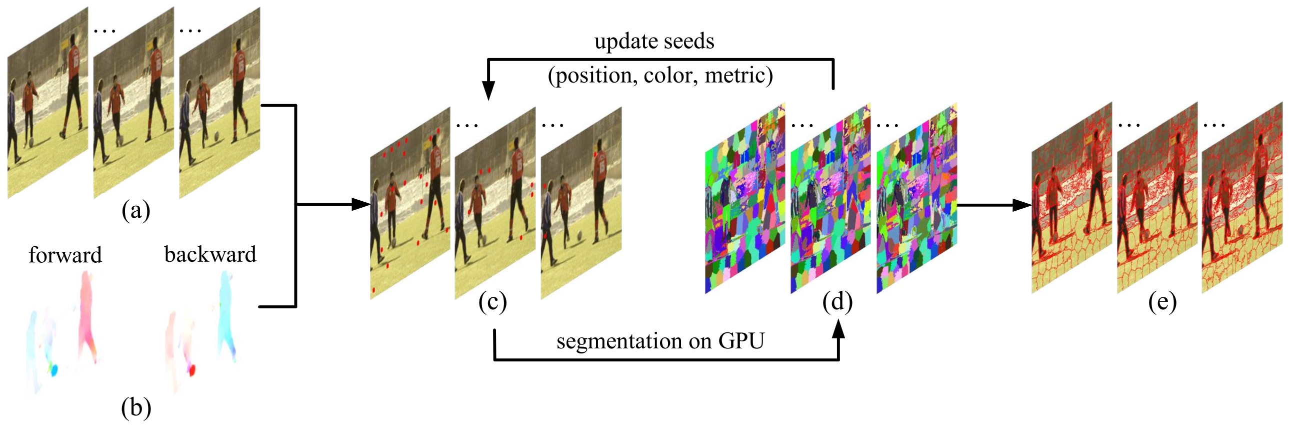


Fig. 4. The workflow of our method. (a) Frame sequences; (b) forward and backward optical flow fields; (c) seed information, including position (red points in the figure), color and anisotropic metric; (d) anisotropic supervoxels on GPU; (e) final result. We optimize the supervoxels by iterating steps (c) and (d) until it converges.

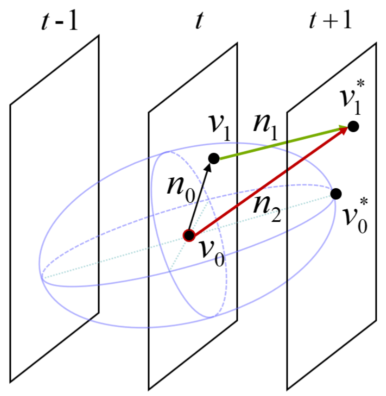


Fig. 5. Metric computation based on optical flow. υ_0 is the position of a seed s_i, υ_1 is its upper right neighbor, and the vector $\mathbf{n}_1$ is the optical flow vector at υ_1. To solve the matrix, we construct the polynomial by constraining $\mathbf{n}_2$ to be of the expected length.

Suppose we have determined the initial position of seeds, we first calculate the metrics M for the seeds, then optimize the segmentation by Lloyd method. In the iteration, the algorithm computes the anisotropic supervoxel segmentation using the jump flooding algorithm (Sec. IV-E). After generating new segmentation, the algorithm updates the seed $\mathbf{v}(s_i)$ to be the centroid of superpixel C_i, and calculates the metric at new position. The algorithm will terminate after several iterations. For the high efficiency, we implement the algorithm on GPU, and we set the maximum number of iterations $iter_m$ to be 20.

1) Computation of Metrics M: During the optimization, one important step is to solve for the non-Euclidean metrics of seeds. We prepare the forward and backward motion fields of all frames using a 2-frame stencil [15] (OpenCV's GPU FarnebackOpticalFlow routines) for the computation of metrics.

For each seed, the anisotropic distance metric redefines the distance between the seed and a voxel by neglecting its motion along the optical flow direction. Thus the iso-distance surface of the matrix is consistent with the moving direction of the object. Fig. 5 is an illustration to show how to construct polynomials to solve the metric based on motion vectors. Suppose the seed s_i is located at υ_0 on frame t, and υ_1 is its upper right neighbor voxel. Assuming that the distance between voxels adjacent in the x, y and t dimensions is of unit length, then vector $\mathbf{n}_0$ is of length $\sqrt{2}$. The green vector $\mathbf{n}_1$ denotes the optical flow of voxel υ_1. If the motion vector is accurate, υ_1^* can be treated the same as υ_1 in the next frame. For a vector $\mathbf{n}$, we denote the squared anisotropic length to be:

$$\|\mathbf{n}\|_{\mathbf{M}_i} = \mathbf{n}^\top \mathbf{M}_i \mathbf{n} = b. \tag{6}$$

Taking vectors $\mathbf{n}_0$, $\mathbf{n}_1$ and $\mathbf{n}_2$ in Fig. 5 for example, the 3D vector and corresponding deformed squared length are as follows:

$$\begin{aligned} \mathbf{n}_0 &= (1,1,0)^\top, \|\mathbf{n}_0\|_{\mathbf{M}_i} = 2, \\ \mathbf{n}_1 &= (i,j,1)^\top, \|\mathbf{n}_1\|_{\mathbf{M}_i} = 1, \\ \mathbf{n}_2 &= \mathbf{n}_0 + \mathbf{n}_1, \|\mathbf{n}_2\|_{\mathbf{M}_i} = (\sqrt{2})^2 + 1^2 = 3. \end{aligned} \tag{7}$$

where (i, j) is the optical flow vector of v_1. $\|\mathbf{n}_1\|_{\mathbf{M}_i}$ is constrained to be 1 under the anisotropic metric $\mathbf{M}_i$ because that the length of $\mathbf{n}_1$ is redefined by neglecting its movement along the optical flow direction.

We take the non-Euclidean squared length $\|\mathbf{n}_2\|_{\mathbf{M}_i}$ as one polynomial equation for computing the metric $\mathbf{M}_i$. There are total 18 vectors like $\mathbf{n_2}$ for seed s_i to its neighbors on the previous and next frames. By constructing 18 equations in the following:

$$\mathbf{n}_p^\top \mathbf{M}_i \mathbf{n}_p = b_p, p = 1, 2, \ldots, 18, \tag{8}$$

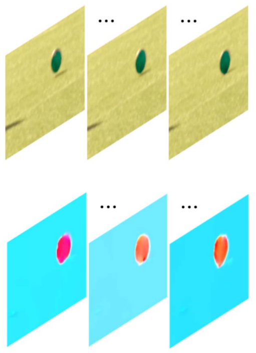
we construct a overdetermined system to solve the Matrix $\mathbf{M}_i$. Here, $\mathbf{n}_p$ is a 3×1 vector, $\mathbf{M}_i$ is a 3×3 positive semidefinite matrix, and b_p is the expected value of the anisotropic length.

In our objective (Eq. (5)), we constrain the matrices M to be positive semidefinite (PSD). PSD is a strong constraint for $\mathbf{M}_i$, which is not always satisfied in the optimization process. In our algorithm, we first guarantee that the matrix is symmetric, denoted as $\tilde{\mathbf{M}}_i$. For those that do not satisfy the PSD constraint, we find a closest positive semidefinite matrix $\bar{\mathbf{M}}_i$ as an alternative. Since the matrix $\tilde{\mathbf{M}}_i$ is symmetric, there will be 6 unknown variables. We denote it as vector $\tilde{\mathbf{m}} = [m_1, m_2, m_3, m_4, m_5, m_6]$. Then transform the overdetermined system (Eq. (8)) to the following equations to solve for the 6 variables of symmetric matrix:

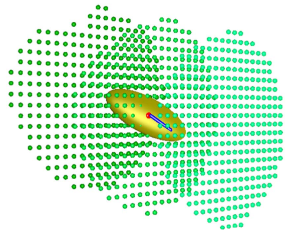
$$\mathbf{A}\tilde{\mathbf{m}} = \mathbf{b}, \tag{9}$$

where $\mathbf{A}$ is a 18×6 matrix and $\mathbf{b}$ is the vector $[b_1, b_2, \ldots, b_{18}]$. The overdetermined system can be solved with pseudo-inverse of $\mathbf{A}$.

Having calculated the symmetric matrix $\tilde{\mathbf{M}}_i$ for each seed s_i, we check if the matrix is positive semidefinite. For the computation of the anisotropic metric, it only needs rough motion information around the seed, and utilizes the optical flow information of 18 neighbors in forward and backward directions, reducing the risk of influences from some problematic flow vector at some pixels. In our implementation, we do not place any seeds at the first or last frame. If the seeds are moved to the first frame during the optimization, our algorithm can construct nine polynomials to solve the variables of the matrix. The matrix $\tilde{\mathbf{M}}_i$ is usually positive semidefinite in most cases. We have tested two optical flow methods in experiments: PyrLKOpticalFlow [5] and FarnebackOpticalFlow [15], and observed that the results are almost independent of the method utilized. We adopt the classic FarnebackOpticalFlow method in our implementation.



(a)



(b)

Fig. 6. Toy model of anisotropic distance matrix. (a): 3 sampled frames in a video and the corresponding optical flow fields. The green masks on football are the superpixels generated by a seed located on it; (b): the ellipsoid is iso-distance surface of the seed matrix, the blue line is the optical flow at the seed, and the green points are voxels of the supervoxels on sampled frames. The main direction of iso-distance surface is consistent with the object motion.

Algorithm 1 Seed Initialization

```
   Input: An input video $\mathbb{I}$ of $n$ voxels, and desired
          number of supervoxels $k$.
   Output: The initial positions of $k$ seeds $S = \{s_i\}_{i=1}^{k}$.
1  Split the video into $k$ uniform grids $\mathbb{I} = \{G_i\}_{i=1}^{k}$,
     taking the centers as the initial position of seeds;
2  Generate an array $F$ with $F_i = 0, i = 1, ...k$;
3  for each seed $s_i$ in grid $G_i$ do
4      Calculate the gradient information in each grid
         $g(G_i) = \sum_{v \in G_i} (g_x(v) + g_y(v) + g_t(v) + \epsilon)$;
5      Compute the anisotropic matrix $\mathbf{M}_i$ of seed $s_i$;
6      Take the difference between $\mathbf{M}_i$ and identity
         matrix $m(G_i) = ||\mathbf{M}_i - \mathbf{I}||_F$ as the measure of
         motion information around the seed;
7      Calculate $A_i = \lambda g(G_i) + m(G_i)$ to represent the
         information richness of objects in the grid;
8      Compute $F_i = \sum_{j=1}^{i} A_j$;
9  end
10 Sample $k$ uniform values between 0 and $F_k$, more
     seeds will be located in the grid with rich information
     of objects.
```

Here we demonstrate a toy example of the computation of anisotropic metrics. Fig. 6(a) shows 3 sampled frames of a rolling football and its corresponding optical flow fields [3]. The background is also in motion due to the movement of the camera. The green masks on the football denote a supervoxel located on it. The red point in figure (b) is the seed, the blue vector denotes the direction of optical flow at the seed point, and the yellow ellipsoid represents the iso-distance surface of anisotropic matrix. It is clear that the main direction of the iso-distance matrix is consistent with the direction of optical flow. With the aid of motion, it is easier for the seed to capture the object in several consecutive frames.

In practice, there may be a few matrices that do not satisfy the PSD constraint. Experiments show that in the scene where the motion of object is relatively fast, optical flow cannot accurately detect the motion of all relative pixels, resulting in a small percentage of the matrices not satisfying the PSD constraint. The failure case usually occurs at a pixel that has similar color and similar motion with its surroundings. Based on the local consistency of the motion, we calculate the PSD matrix of the seed's neighbors that have the same moving direction, and replace the seed matrix with the average matrix of its neighbors. The average of several neighboring PSD matrices is still a PSD matrix. This strategy can solve most of problematic seed matrices. In the worst case, the optical flow method cannot provide an accurate motion field for a local region around the seed, causing the above method failing to find a good alternative to the matrix of the current seed. In this case, we calculate the nearest symmetric PSD matrix of $\tilde{\mathbf{M}}_i$ [21] by the following transformation. First, we take a singular value decomposition of $\tilde{\mathbf{M}}_i$:

$$\tilde{\mathbf{M}}_i = \mathbf{UDU}^\top \tag{10}$$

Then we form the diagonal matrix $\mathbf{D}_+$ by taking the element-wise maximum:

$$\mathbf{D}_+ = max(\mathbf{D}, 0) \tag{11}$$

Finally, we get the closest symmetric PSD matrix $\bar{\mathbf{M}}_i$:

$$\bar{\mathbf{M}}_i = \mathbf{UD}_+\mathbf{U}^\top. \tag{12}$$

Now we can ensure the PSD nature of the seed matrices, and the distance between seeds and voxels can be calculated with Eq. (3). In addition, We provide further analysis in the supplementary document to show the influence of optical flow on the PSD constraint and the effectiveness of the above strategies.

D. Seed Initialization

The optimization of anisotropic supervoxel algorithm will converge to a local optimal solution. Similar to the Lloyd refinement, the optimization is sensitive to initial conditions. We observe that initialization consistent with the video content is able to generate better results than sampling seeds at regular grids. In the following, we introduce an adaptive initialization strategy that adjusts the density of seeds based on video content.

Our initialization strategy is based on the following idea: placing more seed points around the objects is helpful for obtaining more accurate segmentation results. First, suppose we have k uniform grids $\mathbb{I} = \{G_i\}_{i=1}^k$ to represent the video in 3D space, and the centers of grids are treated as candidate initial seeds. Notice that in our implementation, the JFA algorithm generates segmentation by propagating seed information to its neighboring voxels at a fixed step length, thus if there is no seed in a large area, the voxels in that area may not receive any seed information. Considering this, we use the following adaptive scheme that adjusts the candidate seeds according to the information richness in each grid.

The information richness of a grid contains two aspects: the gradient information and the motion information. For the gradient information, usually the gradients around the object

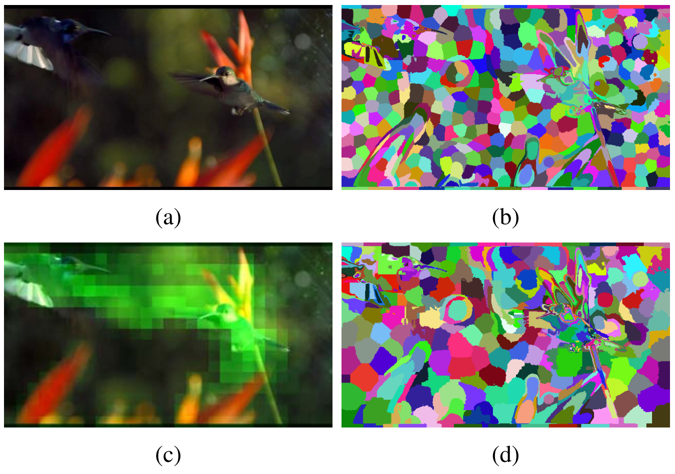


Fig. 7. Seed initialization based on object information. (a): first frame of a video; (b): supervoxels based on uniform seed initialization (in SLIC); (c): first frame with object information masked in green. Normalize the value of A_i to 0-255 as the value of the green channel in corresponding grid. (d): supervoxels based on our seed adjustment strategy, more seeds are located in the area with rich object information.

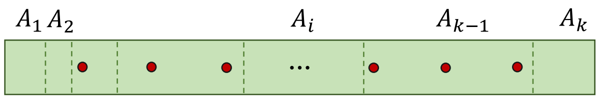


Fig. 8. Sample k seeds based on the information richness A_i of each grid. The value of A_i is proportional to the grid length. More seeds are placed in the grid G_i with higher A_i value.

in x and y directions are more obvious than the background; and the gradient of moving objects in t direction shows the trajectory of the motion. To quickly generate a good initialization, we place more seeds in the grid which contains rich information. As described in Algorithm 1, we first calculate the gradient information $g(G_i)$ in each grid, where the $g_x(v)$ is the gradient of pixel v in x direction, and ϵ is an empirical constant value, preventing the gradient in the grid from being zero. In addition, we measure the motion information at the seed by calculating its anisotropic matrix $\mathbf{M}_i$. We denote the difference between $\mathbf{M}_i$ and the identity matrix as $m(G_i) = ||\mathbf{M}_i - \mathbf{I}||_F$, and it represents the amplitude of movement. We use $A_i = \lambda g(G_i) + m(G_i)$ to represent the information richness of objects in each grid. Then, we generate an array F with k values, with each element storing the grid information, i.e., $F_i = \sum_{j=1}^{i} A_j$. At last, we sample k uniform values between 0 and F_k, as shown in Fig. 8. More seeds are located in the grids with richer object information. In our experiments, we set $\epsilon = 0.1$ and $\lambda = 1$.

We show the illustration of seed initialization in Fig. 7. Given the video of hummingbirds, we split the video into k uniform grids, then calculate the object information in each grid. The object information stored in A_i is normalized to 0-255, and it is taken as the value of the green mask in corresponding grid. Fig. 7(c) shows the first frame covered with the green mask, the area with obvious green is the trajectory of flying hummingbirds. Fig. 7(b) shows supervoxels generated by the SLIC method with uniform initialization, and Fig. 7(d) shows the supervoxels based on our seed initialization strategy. It can be clearly observed that our algorithm places more seeds

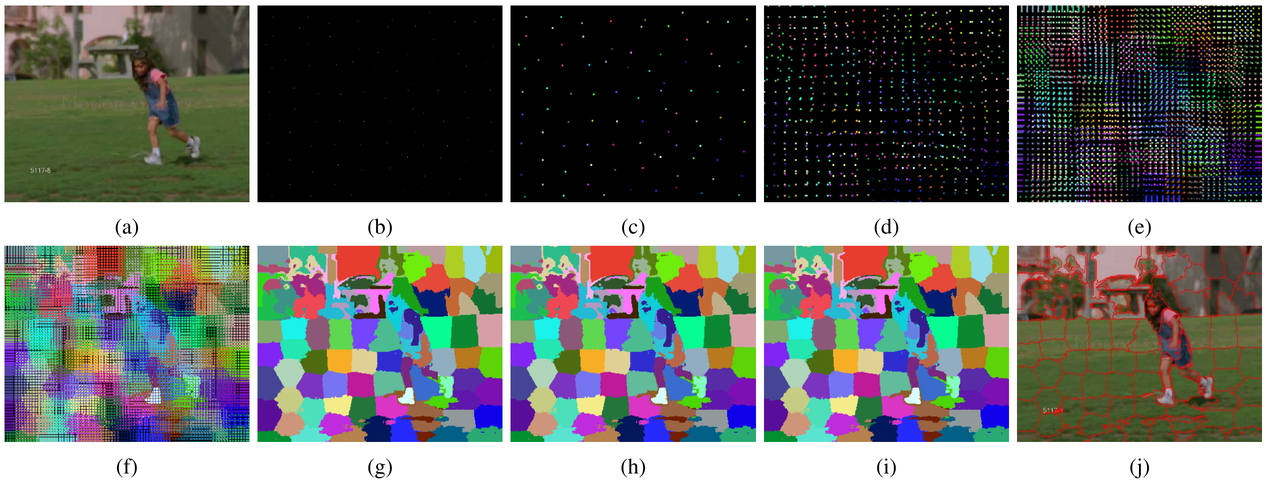


Fig. 9. Procedure of generating Voronoi tessellation on image by 1+JFA with halving step length. (a): original image; (b): initial position of seeds; (c)-(i): flooding results by 7 rounds with step length {1, 18, 9, 5, 3, 2, 1}; (j): superpixels with boundaries.

Algorithm 2 Anisotropic Supervoxel Generation

```
   Input: An input video $\mathbb{I}$ of $n$ voxels, the desired
          number of supervoxels $k$, and the maximum
          number of iterations $iter_m$.
   Output: $k$-partition of the video $\mathbb{I}$.
 1 Load video data to the GPU memory;
 2 Compute the forward and backward optical flow field
     of the video;
 3 Initialize the seeds $S = \{s_i\}_{i=1}^k$ according to the
     strategy in Algorithm 1;
 4 Calculate the metrics $M = \{\mathbf{M}_i\}_{i=1}^k$ of seeds;
 5 Set $iter = 0$;
 6 while $iter < iter_m$ do
 7     Load $S$ and $M$ of seeds to GPU memory;
 8     Compute the supervoxel segmentation using jump
         flooding algorithm. (Sec. IV-E);
 9     for each supervoxel $C_i$ do
10         Update the seed vector $\mathbf{v}(s_i)$ to be the centroid
             $\bar{\mathbf{v}}(C_i)$ of the supervoxel;
11         Calculate the metric $\mathbf{M}_i$ at the new position of
             seed $s_i$. (Sec. IV-C)
12     end
13 end
```

in the motion area, thus is able to achieve more accurate object segmentation results.

E. Supervoxel Generation on GPU

As introduced in the preliminaries, JFA can generate a good approximation of Voronoi tessellation on GPU. Integrating the JFA framework into Lloyd's iteration gives a centroidal Voronoi tessellation (CVT). There are several variants [34] of JFA, among them 1+JFA gives best results with smallest error rate, which performs a pass with step length of 1 before the standard JFA.

Recall that the classic SLIC method [1] clusters pixels in a 5-dimensional Euclidean space combining colors and positions, and iteratively updates the cluster center. First, we implement the SLIC for image superpixels using 1+JFA framework. In SLIC, the local search range of each seed is $2L \times 2L$, we set the maximum step length of JFA to be L, where $L = \sqrt{n/k}$. Here, n is the total number of pixels in image and k is the desired number of superpixels. By performing 1+JFA algorithm once, we get a Voronoi tessellation using Euclidean distance metric. The maximum step length is L, and the first step length is 1. In Fig. 9, given about 100 seeds, the procedure of seeds flooding information is illustrated (in this example, $L = 18$). The second image in the first row shows the initial position of seeds, then followed by 7 rounds of flooding with step lengths $\{1, 18, 9, 5, 3, 2, 1\}$, respectively. The last image shows the boundary adherence of superpixels. These results show that the Voronoi tessellation implemented by the jump flooding framework have very good performance in terms of segment accuracy and boundary adherence.

Next, we extend JFA to video oversegmentation, where every voxel looks for its (maximum) 26 neighbors with a certain step length to choose its nearest seed in 3D space on each round, using $L = \sqrt[3]{n/k}$. We load seeds S and their metrics M into GPU memory in advance, then use the ping-pong buffer alternating as input and output in the rounds of flooding with different step lengths. During the jump flooding procedure, we change the distance measure between seeds and voxels to the metric in Eq. (3) to better catch the motion of objects. The parameter λ_p is related to L, and λ_c is used to adjust the relative importance between color and position. In Fig. 3, the results of our method are anisotropic supervoxels on a video, and the results of SLIC are isotropic supervoxels (which adopts Euclidean distance metric in Eq. (1)). We use the same initialization of seeds in these two methods. The results clearly show that, using the non-Euclidean distance metric consistent with the motion direction is more beneficial to the detection of objects, since the nearest seeds of those voxels in motion tend to remain the same for longer time.

So far we have discussed the computation of anistropic metric, seed initialization and supervoxel generation on GPU. We summarize the detailed steps of this method in Algorithm 2. The algorithm optimizes the segmentation by iterating through steps from #7 to #11. For one iteration, it computes segmentation with given seeds using the JFA

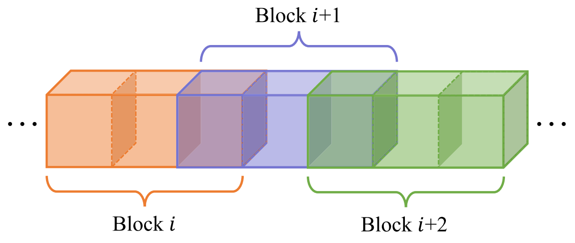


Fig. 10. Streaming GPU-based anisotropic supervoxels for long videos. Our method loads one block into memory at once, and places k seeds for the over-segmentation.

framework on GPU, and calculates the new position and anisotropic matrix of the seed. Then the algorithm uses new seed information for next iteration until the termination condition is reached.

F. Streaming GPU-Based Anisotropic Supervoxel

Our algorithm is able to segment objects with the aid of motion information, while for long videos that cannot be loaded into memory at once, the algorithm should be extended to a streaming version. We use a simple strategy to deal with long videos: the more frames coming, the more seed points should be used. As illustrated in Fig. 10, we set the p frames of video as a block i, place k seeds to segment it, and only load a block in memory at once. With a new block coming in, the corresponding number of new seeds are placed. In order to guarantee the smooth transition between blocks, we divide each block i into three parts, with its first $1/3$ part overlapping with block $i-1$ and its third $1/3$ part overlapping with block $i+1$. For each new block $i+1$, its first $1/3$ part is already in the memory, so we only need to load $2p/3$ new frames into memory. We add new seeds to keep the number of seeds to be k for block $i+1$. For every block, we use Algorithm 2 to generate the anisotropic supervoxels.

V. Experiments

We implemented the algorithm in C++, and compared it with other algorithms on a PC with Intel Core I9-9920X, NVIDIA GeForce RTX 2080 SUPER, and 64 GB RAM running on Linux. We use OpenGL Shading Language (GLSL) to implement the JFA framework on the GPU. We compare our method with several representative methods, including NCut [17], [18], [39], MeanShift [29], GB [16], GBH [20], streamGBH [43], SWA [10], [35], [36], TSP [8], SLIC [1], qd-CSS [44] and FCSS [46]. We name our method as **GPU-AS** (for GPU-based Anisotropic Supervoxels) algorithm, and the streaming version as **streamGPU-AS** algorithm.

A. Evaluation Benchmark

We evaluate the performance based on the LIBSVX4.0 benchmark [42] with multiple performance metrics, including segmentation accuracy, time cost and memory usage.

1) Datasets: The performances of representative algorithms are evaluated on the following datasets: SegTrackv2 [23], BuffaloXiph [9], BVDS [19], [40] and CamVid [6]. SegTrackv2 and BuffaloXiph datasets have frame-by-frame groundtruth annotations, and other datasets provide only sparse annotations. In our experiments, except for NCut (Matlab) and TSP (Matlab with MEX), all other methods are implemented in C or C++. Furthermore, NCut runs with 8 threads in a resized resolution of 240 × 160 down-sampled from the original video. In Fig. 12, we evaluate thirteen supervoxel methods in the range of 0 to 4000 supervoxels per-video on SegTrackv2 dataset. Our algorithm achieves good balance on all metrics, especially on UE3D, BRD, EV, and time metrics. For the performance on other datasets, please refer to the supplementary document.

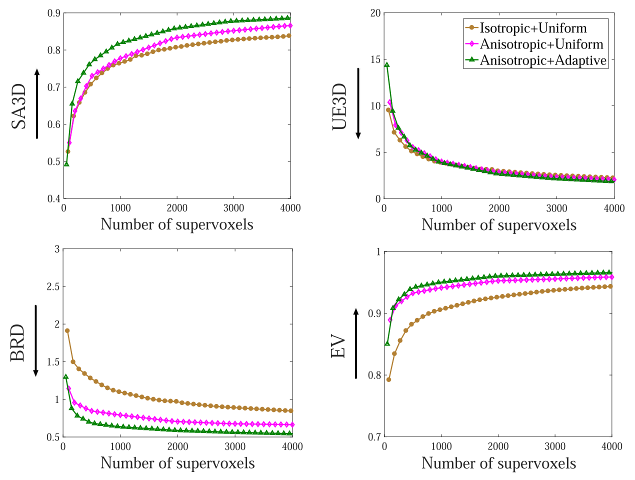


Fig. 11. Ablation experiments on the SegTrackv2 dataset. Isotropic+Uniform is the GPU implementation of the isotropic supervoxel (SLIC) method. Anisotropic+Uniform is anisotropic supervoxel method with same uniform initialization, and Anisotropic+Adaptive is the anisotropic supervoxels with the proposed seed initialization strategy.

2) Evaluation Metrics: The LIBSVX 4.0 benchmark [42] propose multiple standard metrics to evaluate the performance of video over-segmentation based on given human annotation, such as 3D segmentation accuracy (SA3D), 3D undersegmentation error (UE3D), boundary recall distance (BRD), explained variation (EV) and label consistency (LC). SA3D measures the fraction of groundtruth segments that is correctly detected by supervoxels. UE3D measures the fraction of voxels that exceed the boundary of groundtruth when the supervoxels are mapped on it. BRD directly evaluates how well the groundtruth boundaries are successfully retrieved by the supervoxels. EV is proposed to measure the color variation in supervoxels. LC measures the ability of supervoxels tracking objects given groundtruth flows. SA3D and UE3D are two complementary indicators of supervoxel accuracy. For the values of SA3D, EV and LC, the higher the better, while for UE3D and BRD, the lower the better. The video supervoxel is often used as a pre-processing technique, thus the efficiency on space and time is important. We also evaluate the runtime and peak memory cost of these methods.

B. Performance

We summarize the performance of our method and the representative methods on SegTrackv2 dataset in this section. The performance on other three datasets is summarized in supplementary document.

1) Ablation Experiments: Compared with the classic isotropic supervoxel method SLIC, our method improves

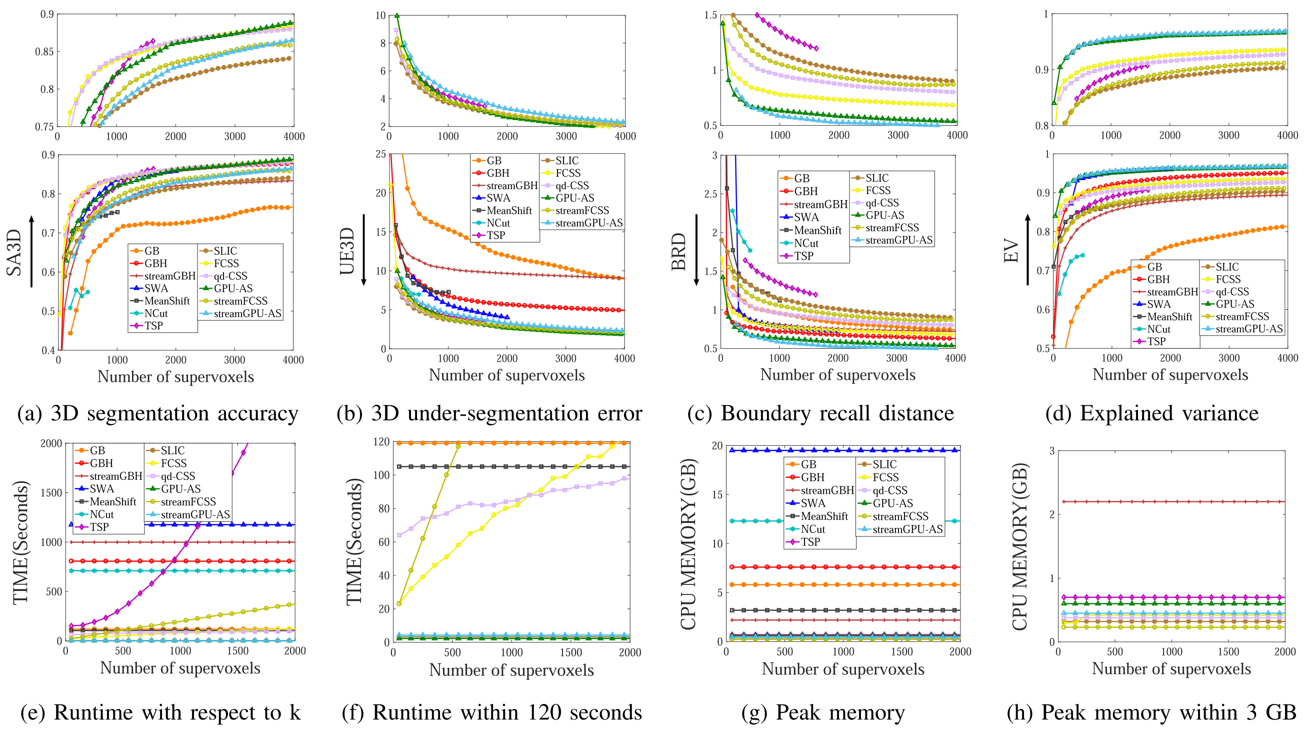
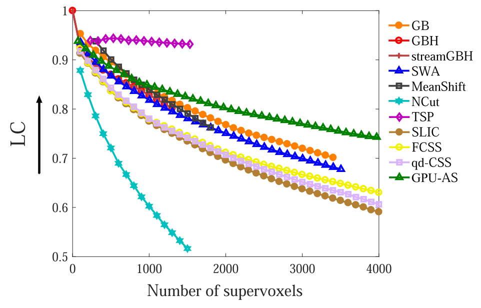
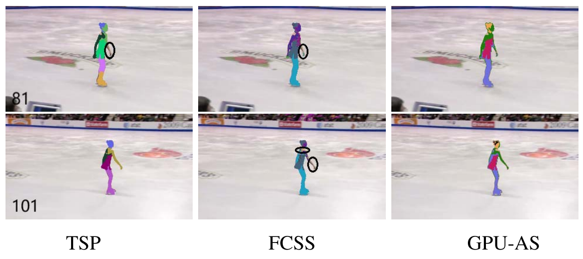


Fig. 12. Performance of representative methods on the SegTrackv2 dataset. Our algorithm has high performance on UE3D, BRD and EV metrics, and achieves a good balance between segmentation quality and efficiency.The top row is the magnified performance curves. We report the runtime and peak memory at a video with $414 \times 352 \times 51$ voxels, our method takes about 2 seconds, requires 600 MB for CPU memory, and 1GB for GPU memory.



(a) Label consistency on the MiddleburyFlow dataset.



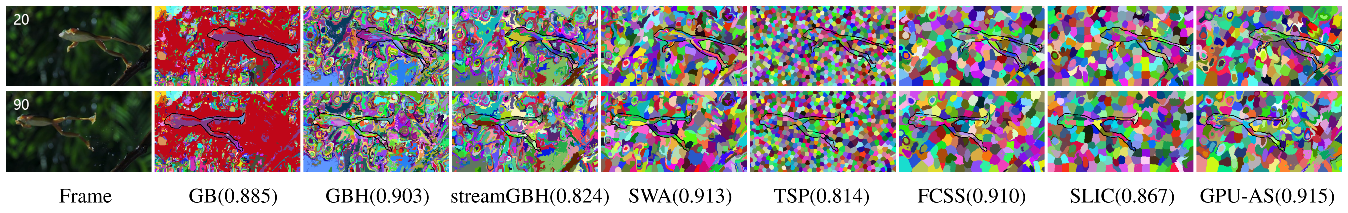
(b) Example of label consistency of three methods.

Fig. 13. Label consistency of supervoxels. Figure (a) shows the LC comparison on MiddleburyFlow dataset. TSP performs best, while our method performs better than other seed-based methods (qd-CSS and FCSS), and achieves a good balance among quality and processing efficiency. In Figure (b), the missed or wrong detections are circled in black.

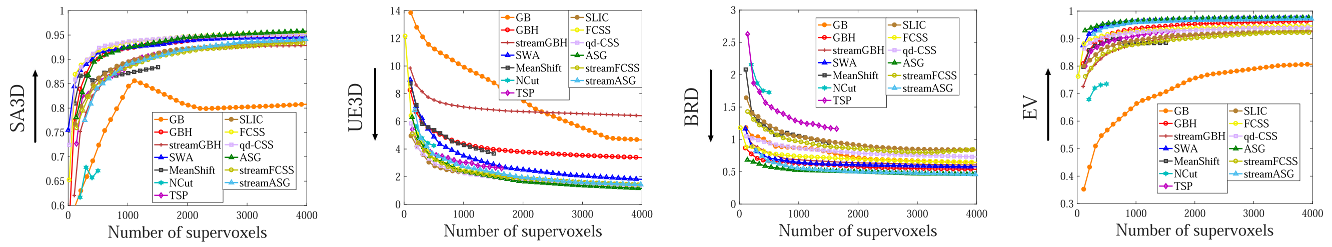
the segmentation quality by proposing the anisotropic distance metric and seed initialization strategy. We clarify the effectiveness of the algorithm in Fig. 11. We first show the performance of Isotropic+Uniform, which is the GPU implementation of SLIC using JFA framework. We also show the anisotropic supervoxels with the same uniform initialization, which is denoted as Anisotropic+Uniform. With the same uniform initialization, anisotropic supervoxels achieve higher segmentation accuracy than isotropic supervoxels. The comparison also shows the improvement by our initialization strategy. Our method with adaptive seed initialization, denoted as Anisotropic+Adaptive, further improves the quality of segmentation.

2) Comparison With Anisotropic Supervoxels: The previous methods such as FCSS, qd-CSS utilize the geodesic centroidal Voronoi tessellation to generate video over-segmentation, which also reflect the anisotropic measurement in the clustering process for the seed. The results in Fig. 12 show that these anisotropic methods can also achieve relatively good performance. However, these previous anisotropic methods do not consider the object motion and cannot be performed on GPU. As a comparison, our algorithm directly uses the motion field to implement the geometric anisotropy of the model, which makes our method particularly suitable for the segmentation of videos containing obvious motions. Furthermore, our anisotropic supervoxel algorithm is the only method that can be implemented in parallel on GPU, which makes our algorithm much more time efficient than existing algorithms.

3) Comparison on SegTrackv2 Dataset: In Fig. 12, we show the performance of thirteen methods on SegTrackv2 dataset. The top row is the magnified details of seven representative methods. (1) For SA3D metric, the performance of GPU-AS gets better as the number of supervoxel increases. GPU-AS sets a fixed search range for every seed, and takes a simple



(a) Supervoxels with SA3D value for a video with continuous object motion



(b) Supervoxel performance on selected videos with continuous motion

Fig. 14. Analysis of anisotropic supervoxels. Our algorithm achieves the best performance on video (a) containing continuous motion of objects. We select some videos with smooth motion from SegTrackv2 dataset. The results summarized in (b) show that our method performs the best for these videos.

strategy to adjust the grid-like seed initialization according to information richness. Other seed-based algorithms FCSS and qd-CSS adopt the K-means++ algorithm [2] to decide initial seed position, which is able to generate better initialization for seeds, however, this strategy is very time consuming. Compared with them, our initialization is relatively even and this is required for the jump flooding procedure on GPU. If there is no seed placed in a large area, some voxels will not receive any seed information during the JFA process, which is not conducive to segmenting objects. (2) UE3D imposes a penalty when supervoxels leak on groundtruth segments. TSP, qd-CSS, FCSS, SLIC and ours have the best performance on UE3D. (3) The performance on EV shows that our algorithm is sensitive to color variation, and the color homogeneity in each supervoxel is high. (4) Our algorithm achieves the best performance on BRD than other methods.

4) Comparison on Efficiency: In Fig. 12, we evaluate the run time and peak memory on a typical video with $414 \times 352 \times 51$ voxels. Our algorithm requires about 600 MB for CPU memory and 1 GB for GPU memory, which can be met by most computers. The GPU memory is mainly used for JFA. For time performance, our algorithm is able to generate the segmentation for the video of 51 frames in about 2 seconds, and the computation time is not affected by the number of supervoxels, which is not satisfied by other seed-based methods FCSS and qd-CSS. For the graph-based methods, such as GBH, streamGBH and SWA, they take more than 10 minutes to process the video. The TSP uses a generative probabilistic model to propagate superpixels in a streaming fashion, which takes much longer time with more supervoxels. In Table I we show run time of each stage on three videos. Our algorithm first loads video data into CPU, then passes it to the GPU and computes the optical flow information. The JFA stage includes Lloyd iteration and matrix solving. This stage takes up half of the total time. In Table II we test the average running time of representative methods on four datasets. Our method has huge advantage on processing efficiency.

TABLE I

Run Time (Seconds) of Each Stage of the Algorithm With $k = 1000$, on Videos in SegTrackv2 Dataset. The Video Resolutions Are $414 \times 352 \times 51$, $400 \times 320 \times 21$, and $640 \times 360 \times 98$, Respectively

Stage / Video	Load (CPU)	Load (GPU)	Optical flow	JFA	Total
Parachute	0.51	0.17	0.26	1.21	2.15
Girl	0.45	0.06	0.12	0.73	1.36
Bird	0.82	0.47	0.71	3.73	5.73

5) Comparison on Label Consistency: Label consistency (LC) measures how well supervoxels track the objects given groundtruth flows. In Fig. 13(a), we show the LC comparisons on the MiddleburyFlow dataset [3]. TSP performs best, while GPU-AS achieves a good balance among quality metrics and running time. GPU-AS outperforms other seed-based algorithms such as SLIC, qd-CSS and FCSS due to its good object tracking capability. Furthermore, we show an example comparison of label consistency between TSP, FCSS and our method in Fig. 13(b). The colored regions are supervoxels that successfully detect the objects according to the definition of the SA3D metric. From the results we can see that, our algorithm shows temporal consistency between frames, and is able to detect more details of objects. The TSP and FCSS methods are not sensitive to the color variation, thus fail to detect the arms of the skater. In addition, we provide more comparison on the LC metric in the supplementary document.

C. Discussion

The anisotropic distance metrics help the seeds detect the object more accurately, as shown in Fig. 14 (a), our method achieves the highest segmentation accuracy for the moving

TABLE II

AVERAGE RUN TIME (SECONDS) OF DIFFERENT METHODS ON DATASETS WITH 1000 SUPERVOXELS

Dataset \ Method	GB [16]	GBH [20]	streamGBH [43]	MeanShift [29]	SWA [10]	TSP [8]	FCSS [46]	qd-CSS [44]	SLIC [1]	GPU-AS
SegTrackv2	140.21	887.52	1103.58	128.63	1309.43	908.86	92.27	90.38	2.04	2.68
BuffaloXiph	125.97	805.74	979.63	116.95	1198.54	885.06	83.03	75.14	1.91	2.02
BVDS	151.84	896.36	1125.57	131.04	1342.93	921.01	105.23	98.07	2.12	3.17
CamVid	70.73	462.95	513.34	58.22	620.89	440.73	51.73	42.66	1.12	2.58

TABLE III

F MEASURE AND ACCURACY MEASURE OF FOREGROUND PROPAGATION ON YOUTUBE-OBJECTS BASED ON DIFFERENT SUPERVOXEL METHODS. FOR EACH CATEGORY, WE HIGHLIGHT THE METHOD WITH THE BEST RESULT IN BOLD. THE LAST ROW SHOWS THE AVERAGE PERFORMANCE OVER TEN CATEGORIES, AND WE HIGHLIGHT THE TOP THREE METHODS

Obj \ Method	GB [16]	GBH [20]	MeanShift [29]	TSP [8]	FCSS [46]	qd-CSS [44]	SLIC [1]	GPU-AS
Aeroplane	0.862, 0.973	0.850, 0.963	0.806, 0.961	0.860, 0.970	0.875, 0.971	0.874, 0.971	0.842, 0.963	**0.881**, **0.974**
Bird	0.802, 0.979	0.602, 0.975	0.742, 0.966	**0.803**, 0.976	0.795, **0.980**	0.743, 0.973	0.745, 0.978	0.774, 0.977
Boat	0.842, 0.962	0.740, 0.956	0.834, 0.965	**0.846**, **0.966**	0.805, 0.953	0.833, 0.963	0.783, 0.943	0.841, 0.958
Car	0.826, 0.917	0.768, 0.896	0.830, 0.911	0.856, 0.919	0.842, 0.915	0.850, 0.916	0.845, 0.916	**0.858**, **0.924**
Cat	0.432, 0.908	0.543, 0.954	0.646, 0.935	0.684, 0.954	0.693, 0.943	**0.715**, **0.956**	0.618, 0.943	0.694, 0.951
Cow	0.820, 0.950	0.801, 0.952	0.785, 0.952	0.826, 0.958	**0.831**, **0.960**	0.830, **0.960**	0.815, 0.953	0.820, 0.956
Dog	0.822, 0.929	0.823, 0.938	0.778, 0.934	0.821, 0.941	0.823, 0.942	0.825, 0.951	0.805, 0.941	**0.827**, **0.946**
Horse	0.757, **0.962**	**0.763**, 0.959	0.726, 0.936	0.758, 0.956	0.760, 0.955	0.752, 0.958	0.739, 0.957	0.760, 0.958
Motorbike	0.682, 0.912	0.630, 0.881	0.665, 0.907	**0.730**, **0.932**	0.707, 0.911	0.693, 0.912	0.683, 0.911	0.703, 0.914
Train	0.858, 0.952	0.613, 0.907	0.856, 0.963	**0.871**, **0.968**	0.840, 0.959	0.865, 0.962	0.805, 0.958	0.853, 0.963
Average	0.770, 0.944	0.713, 0.938	0.766, 0.943	**0.805**, **0.954**	0.797, 0.948	**0.798**, **0.952**	0.768, 0.946	**0.801**, **0.952**

TABLE IV

F MEASURE AND ACCURACY MEASURE OF FOREGROUND PROPAGATION ON YOUTUBE-OBJECTS BASED ON STREAMING SUPERVOXELS. THE LAST ROW SHOWS THAT THE STREAMGPU-AS HAS THE BEST AVERAGE PERFORMANCE OVER TEN CATEGORIES

Obj \ Method	streamGBH [43]	streamFCSS [46]	streamGPU-AS
Aeroplane	0.824, 0.961	0.861, 0.967	0.881, 0.972
Bird	0.768, 0.976	0.730, 0.969	0.724, 0.976
Boat	0.734, 0.951	0.820, 0.961	0.835, 0.959
Car	0.798, 0.907	0.864, 0.922	0.853, 0.920
Cat	0.670, 0.951	0.580, 0.944	0.565, 0.953
Cow	0.790, 0.948	0.785, 0.948	0.751, 0.950
Dog	0.663, 0.920	0.803, 0.937	0.796, 0.940
Horse	0.674, 0.947	0.734, 0.953	0.776, 0.962
Motorbike	0.658, 0.913	0.675, 0.903	0.657, 0.892
Train	0.764, 0.953	0.820, 0.953	0.832, 0.952
Average	0.734, 0.942	**0.767**, 0.946	**0.767**, **0.948**

object. The seeds located on the object and the background have anisotropic distance metrics consistent with different motion directions, which are helpful for segmenting the object more accurately. For these videos without obvious object or background motion, our method performs average. We calculated the average pixel motion (apm) value for the foreground objects in a video. The average apm value over all videos of SegTrackv2 dataset is 2.85. We select some videos with continuous motion from the dataset, which has average apm value 3.24. As shown in Fig. 14(b), our GPU-AS method performs best for these videos according to the performance on comprehensive evaluation.

D. Application

Supervoxels are designed to reduce the computational complexity for video processing tasks. To further show the advantage of our algorithm, we apply the supervoxels to solve practical problems in computer vision. In the following experiment, we take supervoxel results generated by different methods as input for the foreground propagation application, and evaluate the performance.

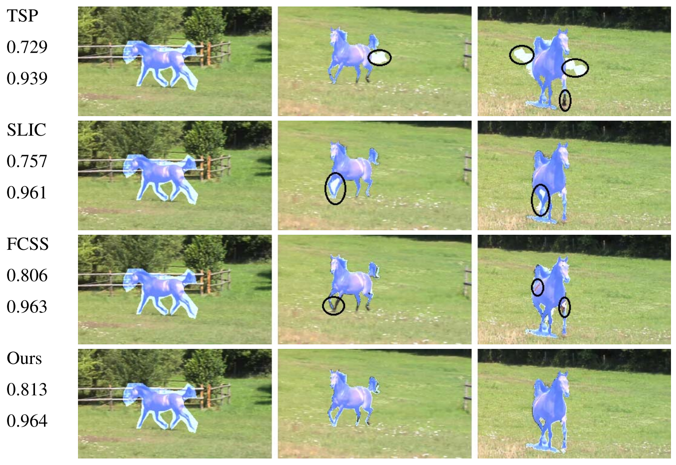


Fig. 15. Foreground propagation example of the representative methods: TSP, SLIC, FCSS and our GPU-AS method, with F measure and Accuracy measure. The first column shows the first frame of the video with manual annotated mask. The mislabeled regions are circled in black. Our method performs best on F and accuracy metrics.

The application [22] propagates foreground object regions in the video. Given an initial frame with manual annotation for the foreground object, it propagates the foreground region to the remainder of frames, by using supervoxels to guide its estimates towards long-range coherent regions. We estimate representative supervoxel methods (GB, GBH, streamGBH, MeanShift, TSP, FCSS, qd-CSS, SLIC) and our method GPU-AS on the Youtube-Objects dataset [31] (126 videos with 10 object categories) with the groundtruth of foreground objects. NCut and SWA are not compared because of their high computational cost and huge memory cost for long videos. We evaluate the results using two measures: F measure and accuracy measure. The F measure $= 2 \cdot PR/(P+R)$, where P is precision and R is recall. The values of F measure and accuracy measure range in [0,1], and the larger values mean better results.

The average F measure and accuracy measure per class on YouTube-Objects dataset are illustrated in Table III and Table IV. We can see that TSP, qd-CSS, FCSS and our method GPU-AS all demonstrate high performance averagely over ten classes. The value of average F measure and accuracy measure using the SLIC supervoxels is lower than using our supervoxels because the isotropic supervoxels have low segmentation accuracy and boundary adherence. Table IV shows the performance of foreground propagation based on three streaming supervoxel methods, our method has high performance averagely over ten categories. Fig. 15 shows foreground propagation results of an example video. The images in the first column are first frames, foreground objects are manual annotated with blue masks. We show foreground propagation results of two frames in the second and third columns. The mislabeled regions are circled in black. Some regions of the object are not successfully detected, and some regions on the background are mislabeled as foreground. We give the values of F measure and accuracy measure below each method. The result based on SLIC has low accuracy in narrow areas, such as the legs of horse in the video. Using the supervoxels of our algorithm, the application [22] achieves the highest quality.

VI. Conclusion

In this paper, we present a GPU-based anisotropic supervoxel generation algorithm which is fast and can obtain video over-segmentation with high quality results. In order for the supervoxels to better capture the motion of objects, we design a new anisotropic metric for each supervoxel to calculate the distance between the seeds and the voxels, and adopt an initialization strategy that is consistent with the video content. The energy minimization in our algorithm is a Lloyd-like optimization with several iterations, where in each iteration, given the position and metric of seeds, our algorithm computes an anisotropic Voronoi tessellation by extending the jump flooding framework on GPU. We evaluate our algorithm on four datasets and apply it in a practical application. Experimental results show that our method has high performances in terms of SA3D, UE3D, BRD, and EV metrics on different datasets, and achieves good balance on segmentation quality and efficiency. Our method generates high quality segmentation especially for videos containing moving objects. The video application shows that our method works well when applying supervoxels to solve practical problems. The parallel implementation with JFA makes our algorithm much more efficient in space and time than the state-of-the-art methods.

The proposed method is very efficient on image and video over-segmentation, which is practical for video applications with large time cost. In the future, we consider applying our method to more applications that require fast segmentation on images, videos or point clouds.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, “SLIC superpixels compared to state-of-the-art superpixel methods,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012.

[2] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” in *Proc. 18th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2007, pp. 1027–1035.

[3] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, “A database and evaluation methodology for optical flow,” *Int. J. Comput. Vis.*, vol. 92, no. 1, pp. 1–31, 2011.

[4] R. Borgo *et al.*, “State of the art report on video-based graphics and video visualization,” in *Computer Graphics Forum*, vol. 31. Hoboken, NJ, USA: Wiley, 2012, pp. 2450–2477.

[5] J.-Y. Bouguet *et al.*, “Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm,” *Intel Corp.*, vol. 5, nos. 1–10, p. 4, 2001.

[6] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, “Segmentation and recognition using structure from motion point clouds,” in *Proc. Eur. Conf. Comput. Vis.* Marseille, France: Springer, 2008, pp. 44–57.

[7] Y. Cai and X. Guo, “Anisotropic superpixel generation based on Mahalanobis distance,” in *Comput. Graph. Forum*, vol. 35. Hoboken, NJ, USA: Wiley, 2016, pp. 199–207.

[8] J. Chang, D. Wei, and J. W. Fisher III, “A video representation using temporal superpixels,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 2051–2058.

[9] A. Y. C. Chen and J. J. Corso, “Propagating multi-class pixel labels throughout video frames,” in *Proc. Western New York Image Process. Workshop*, Nov. 2010, pp. 14–17.

[10] J. J. Corso, E. Sharon, S. Dube, S. El-Saden, U. Sinha, and A. Yuille, “Efficient multilevel brain tumor segmentation with integrated Bayesian model classification,” *IEEE Trans. Med. Imag.*, vol. 27, no. 5, pp. 629–640, May 2008.

[11] X. Dong, Z. Chen, J. Yao, and X. Guo, “Superpixel generation by agglomerative clustering with quadratic error minimization,” in *Computer Graphics Forum*, vol. 38. Hoboken, NJ, USA: Wiley, 2019, pp. 405–416.

[12] X. P. Dong, J. B. Shen, L. Shao, and L. van Gool, “Sub-Markov random walk for image segmentation,” *IEEE Trans. Image Process.*, vol. 25, no. 2, pp. 516–527, Feb. 2015.

[13] X. Dong, J. Shen, L. Shao, and M.-H. Yang, “Interactive cosegmentation using global and local energy optimization,” *IEEE Trans. Image Process.*, vol. 24, no. 11, pp. 3966–3977, Nov. 2015.

[14] Q. Du, V. Faber, and M. Gunzburger, “Centroidal Voronoi tessellations: Applications and algorithms,” *SIAM Rev.*, vol. 41, no. 4, pp. 637–676, Apr. 1999.

[15] G. Farnebäck, “Two-frame motion estimation based on polynomial expansion,” in *Scandinavian Conference on Image Analysis*. Halmstad, Sweden: Springer, 2003, pp. 363–370.

[16] P. F. Felzenszwalb and D. P. Huttenlocher, “Efficient graph-based image segmentation,” *Int. J. Comput. Vis.*, vol. 59, no. 2, pp. 167–181, Sep. 2004.

[17] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, “Spectral grouping using the Nystrom method,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 2, pp. 214–225, Feb. 2004.

[18] C. Fowlkes, S. Belongie, and J. Malik, “Efficient spatiotemporal grouping using the Nystrom method,” in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Dec. 2001, pp. 1–8.

[19] F. Galasso, N. S. Nagaraja, T. J. Cardenas, T. Brox, and B. Schiele, “A unified video segmentation benchmark: Annotation, metrics and analysis,” in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 3527–3534.

[20] M. Grundmann, V. Kwatra, M. Han, and I. Essa, “Efficient hierarchical graph-based video segmentation,” in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2010, pp. 2141–2148.

[21] N. J. Higham, “Computing the nearest correlation matrix—A problem from finance,” *IMA J. Numer. Anal.*, vol. 22, no. 3, pp. 329–343, Jul. 2002.

[22] S. D. Jain and K. Grauman, “Supervoxel-consistent foreground propagation in video,” in *Proc. Eur. Conf. Comput. Vis.* Zürich, Switzerland: Springer, 2014, pp. 656–671.

[23] F. Li, T. Kim, A. Humayun, D. Tsai, and J. M. Rehg, "Video segmentation by tracking many figure-ground segments," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 2192–2199.

[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.*, vol. 40, no. 3, pp. 653–666, Mar. 2018.

[25] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.

[26] D. Ma, Y. Zhou, S. Xin, and W. Wang, "Convex and compact superpixels by edge-constrained centroidal power diagram," *IEEE Trans. Image Process.*, vol. 30, pp. 1825–1839, 2020.

[27] D. Oneata, J. Revaud, J. Verbeek, and C. Schmid, "Spatio-temporal object detection proposals," in *Proc. Eur. Conf. Comput. Vis.* Zürich, Switzerland: Springer, 2014, pp. 737–752.

[28] S. Paris, "Edge-preserving smoothing and mean-shift segmentation of video streams," in *Proc. Eur. Conf. Comput. Vis.* Marseille, France: Springer, 2008, pp. 460–473.

[29] S. Paris and F. Durand, "A topological approach to hierarchical segmentation using mean shift," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2007, pp. 1–8.

[30] S. Pertsel, O. Meitav, E. Pozniansky, and E. Galil, "Digital camera with selectively increased dynamic range by control of parameters during image acquisition," U.S. Patent 8 687 087, Apr. 1, 2014.

[31] A. Prest, C. Leistner, J. Civera, C. Schmid, and V. Ferrari, "Learning object class detectors from weakly annotated video," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3282–3289.

[32] M. Reso, J. Jachalsky, B. Rosenhahn, and J. Ostermann, "Temporally consistent superpixels," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 385–392.

[33] G. Rong and T.-S. Tan, "Jump flooding in GPU with applications to Voronoi diagram and distance transform," in *Proc. Symp. Interact. 3D Graph. Games (SI3D)*, 2006, pp. 109–116.

[34] G. Rong and T.-S. Tan, "Variants of jump flooding algorithm for computing discrete Voronoi diagrams," in *Proc. 4th Int. Symp. Voronoi Diagrams Sci. Eng. (ISVD)*, Jul. 2007, pp. 176–181.

[35] E. Sharon, A. Brandt, and R. Basri, "Fast multiscale image segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2000, pp. 70–77.

[36] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt, "Hierarchy and adaptivity in segmenting visual scenes," *Nature*, vol. 442, no. 7104, p. 810, 2006.

[37] J. Shen, Y. Du, W. Wang, and X. Li, "Lazy random walks for superpixel segmentation," *IEEE Trans. Image Process.*, vol. 23, no. 4, pp. 1451–1462, Apr. 2014.

[38] J. Shen, X. Hao, Z. Liang, Y. Liu, W. Wang, and L. Shao, "Real-time superpixel segmentation by DBSCAN clustering algorithm," *IEEE Trans. Image Process.*, vol. 25, no. 12, pp. 5933–5942, Dec. 2016.

[39] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000.

[40] P. Sundberg, T. Brox, M. Maire, P. Arbelaez, and J. Malik, "Occlusion boundary detection and figure/ground assignment from optical flow," in *Proc. CVPR*, Jun. 2011, pp. 2233–2240.

[41] P. Wang, G. Zeng, R. Gan, J. Wang, and H. Zha, "Structure-sensitive superpixels via geodesic distance," *Int. J. Comput. Vis.*, vol. 103, no. 1, pp. 1–21, 2013.

[42] C. Xu and J. J. Corso, "LIBSVX: A supervoxel library and benchmark for early video processing," *Int. J. Comput. Vis.*, vol. 119, no. 3, pp. 272–290, 2016.

[43] C. Xu, C. Xiong, and J. J. Corso, "Streaming hierarchical video segmentation," in *Proc. Eur. Conf. Comput. Vis.* Florence, Italy: Springer, 2012, pp. 626–639.

[44] Z. Ye, R. Yi, M. Yu, Y.-J. Liu, and Y. He, "Fast computation of content-sensitive superpixels and supervoxels using Q-distances," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3770–3779.

[45] R. Yi, Y.-J. Liu, and Y.-K. Lai, "Content-sensitive supervoxels via uniform tessellations on video manifolds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 646–655.

[46] R. Yi, Z. Ye, W. Zhao, M. Yu, Y.-K. Lai, and Y.-J. Liu, "Feature-aware uniform tessellations on video manifold for content-sensitive supervoxels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 3183–3195, Sep. 2021.

Xiao Dong received the M.Eng. degree from Xiamen University, China, in 2016, where she is currently pursuing the Ph.D. degree with the School of Informatics. She was a Visiting Scholar at The University of Texas at Dallas from 2017 to 2019. Her research interests include computer vision and computer graphics.

Zhonggui Chen received the B.Sc. and Ph.D. degrees in applied mathematics from Zhejiang University in 2004 and 2009, respectively. He is currently a Professor with the Department of Computer Science and Technology, School of Informatics, Xiamen University, China. His research interests include computer graphics and computational geometry http://graphics.xmu.edu.cn/∼zgchen/.

Yong-Jin Liu (Senior Member, IEEE) received the B.Eng. degree from Tianjin University, China, in 1998, and the Ph.D. degree from The Hong Kong University of Science and Technology, Hong Kong, China, in 2004. He is currently a Professor with the Department of Computer Science and Technology, Tsinghua University, China. His research interests include computational geometry, computer vision, and computer graphics. He is a member of ACM http://cg.cs.tsinghua.edu.cn/people/∼Yongjin/Yongjin.htm.

Junfeng Yao received the B.S. and Ph.D. degrees from Central South University in 1995 and 2001, respectively. He is currently a Professor at the School of Film, Xiamen University, China. His research interests include computer graphics, virtual reality, intelligent algorithm research, and industrial process simulation https://cdmc.xmu.edu.cn/info/1010/1062.htm.

Xiaohu Guo (Member, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China and the Ph.D. degree in computer science from Stony Brook University. He is currently a Professor of computer science at The University of Texas at Dallas. His research interests include computer graphics, computer vision, medical imaging, and VR/AR, with an emphasis on geometric modeling and processing. He received the prestigious NSF CAREER Award in 2012 https://personal.utdallas.edu/∼xguo/.