GPU-based Supervoxel Segmentation for 3D Point Clouds

Xiao Dong[a], Yanyang Xiao[b], Zhonggui Chen[a,*], Junfeng Yao[c,*], Xiaohu Guo[d,*]

[a]*School of Informatics, Xiamen University, Xiamen, 361005, Fujian, China*
[b]*School of Information Engineering, Nanchang University, Nanchang, 330031, Jiangxi, China*
[c]*School of Film, Xiamen University, Xiamen, 361005, Fujian, China*
[d]*Department of Computer Science, University of Texas at Dallas, 75083, Dallas, USA*

Abstract

Point cloud processing has received more attention in recent years. Due to the huge amount of data, using supervoxels to pre-segment the points can improve the performance of point cloud processing tasks. There are some supervoxel algorithms generating high-quality results, but their low efficiency hinders the wide application in point cloud processing tasks. In this paper, we try to strike a good balance between the quality and efficiency of point cloud over-segmentation. We propose an algorithm suitable for GPU acceleration, which can generate supervoxel with high efficiency. The algorithm is a seed-based segmentation method, and we carefully design two stages: the clustering stage and optimization stage, each of which can be executed in parallel on the GPU. In the first stage, the algorithm generates an initial segmentation based on well designed energy functions, and the second stage further improves the result by minimizing the segmentation energy. Our method generates good segmentation results and achieves the fastest processing speed compared with the existing methods. We evaluate the supervoxels on three public datasets. Experiments show that our algorithm can generate high-quality segmentation for various point cloud data with high efficiency, which is important for advancing the application of point cloud supervoxels in subsequent processing.

Keywords: Point Clouds, Supervoxel Segmentation, GPU Computation

1. Introduction

Segmentation for point cloud is a core problem in 3D computer vision. Recent years, supervoxel segmentation is becoming increasingly popular in various point cloud processing applications. Due to huge amount of data in 3D point clouds, most tasks require huge memory space and processing time to generate final results. Supervoxel segmentation is an important pre-processing technique, which provides a more natural and compact representation for 3D point clouds, and greatly reduces the number of basic units to process. Supervoxel segmentation for point clouds is widely used in many applications, such as saliency detection Yun and Sim (2016), classification Sun et al. (2018), object detection Wang et al. (2015), recognition Dubé et al. (2017) and semantic segmentation Luo et al. (2015); Huang et al. (2021).

Similar to image and video segmentation Li and Chen (2015); Dong et al. (2021), there are some criteria for evaluating the supervoxel quality of point clouds, such as: boundary adherence, segmentation error and efficiency. The supervoxel segmentation should detect the object boundaries, and each supervoxel should only overlap with one object. Meanwhile, the shape of supervoxel should be compact, especially in the region without features, which produces a simpler adjacency graph for post processing. In addition, supervoxel generation should be efficient, and not reduce the achievable performance of its downstream applications.

Image segmentation algorithms have developed rapidly in recent years, and many fast and accurate methods Ren et al. (2015); Cai and Guo (2016); Liu et al. (2017) have been widely used in subsequent processing Yang et al. (2014a); Wang et al. (2017). It is natural to extend superpixels on images to videos since the primitives are also uniformly distributed. But for point clouds, the generation of supervoxels poses more challenges. The data volume

*Corresponding authors

of a point cloud is much larger than that of an image, and it usually contains noisy points and outliers. The point distribution of a point cloud is non-uniform and disordered, and the neighbor relationships between 3D points require additional calculation.

Given a point cloud, many supervoxel methods voxelize the input points into voxels to reduce the number of basic processing units. For example, the classic VCCS method Papon et al. (2013) is based on voxels, which is efficient, and generates good segmentation on RGB-D point clouds. However, VCCS does not have obvious advantages in the segmentation of outdoor scenes, and it is not sensitive to the boundaries and features of objects. Lately some algorithms Lin et al. (2018); Xiao et al. (2020) aiming for better feature preservation are proposed. They directly act on the 3D points, and need the support of the neighborhood graph of points, resulting in low efficiency on running time. In fact, the longtime consumption hinders the wide application of supervoxels as a pre-processing technology in many point cloud processing tasks.

Performance and efficiency have always been difficult to balance. After studying the previous point cloud supervoxel algorithms, we found that they cannot be easily accelerated by the GPU implementation. For example, the BPSS method Lin et al. (2018) stores all points in a queue, and updates the queue and the energy function as points are processed. The MS method Xiao et al. (2020) maintains a min-heap which stores the order of supervoxel merging pairs, and updates it during processing. The above design of data structures and sequential operations result in algorithms that are not easily parallelizable. In our work, we try to design a highly parallel algorithm and lightweight data structures to generate supervoxels on GPU. Our method highly improves the processing efficiency, and generates segmentation with good quality.

We try to balance the quality and efficiency of supervoxel segmentation. We first define an energy function that is suitable for feature preservation, and then utilize two parallel algorithms to segment the point cloud data on GPU. The first algorithm is a seed-based clustering algorithm, which is converted to a parallel calculation of distances between each voxel and its surrounding seeds. The second algorithm is an optimization algorithm that calculates the cost of swapping a voxel from its current supervoxel to the neighbor supervoxels, then selects voxels that can reduce energy for exchange. In the implementation, clustering and swapping operations can be performed in parallel for all voxels. Compared with existing methods, our algorithm achieves the fastest processing speed and generates segmentation with good quality, which is of great significance for promoting the application of supervoxels in point cloud processing.

2. Related Work

Point clouds are most commonly generated using 3D laser scanners and LiDAR technology. We can also produce a point cloud by projecting an RGB-D image to 3D space in the camera coordinate system. Many superpixel algorithms can be applied to the segmentation of RGB-D images. Depth information can be combined with position and color information to measure the similarity between superpixels. For instance, the classic superpixel method SLIC Achanta et al. (2012) simply adds depth in the distance function and uses local K-means clustering to generate supervoxels. Based on SLIC, Weikersdorfer et al. (2012) proposed a depth-adaptive superpixel method which segments the image in a nine-dimensional feature space composed of coordinates, normals and colors. Dong et al. (2019) tried to cluster the image in a greedy mode by merging small patches with costs in descending order. This method is very sensitive to color changes and can be easily extended to RGB-D image segmentation. Different with the above methods, some algorithms utilize the geometry features of RGB-D images for segmentation. The method proposed by Yang et al. (2013) reconstructs 3D geometry of RGB-D image based on the depth map, and clusters pixels in high dimensional space. Based on it, the authors further proposed a graph-based energy minimization framework Yang et al. (2014b) with label cost for better segmentation. Pan et al. (2016) proposed a method that constructs a triangular mesh and optimizes the segmentation of the mesh based on a geodesic driven metric.

RGB-D image is a simple type of point cloud, often used to represent indoor scenes. The above supervoxel methods utilize the structure of uniformly distributed pixels in RGB-D images, and can not be applied to supervoxel segmentation of unorganized 3D point clouds. For scanned objects and architectures, the point cloud model usually contains non-uniform and noise data. To the best of our knowledge, there are only a few algorithms that directly deal with the segmentation of such irregular point clouds. The most popular algorithm is VCCS Papon et al. (2013) due to its efficiency and simplicity. VCCS is a flow constrained clustering algorithm based on voxels. It first divides the data space into voxels using an octree, and selects some seeds as growing centers that flows the label outward to adjacent voxels. But when the adjacency graph connects different objects, it fails to segment the object at the boundary.

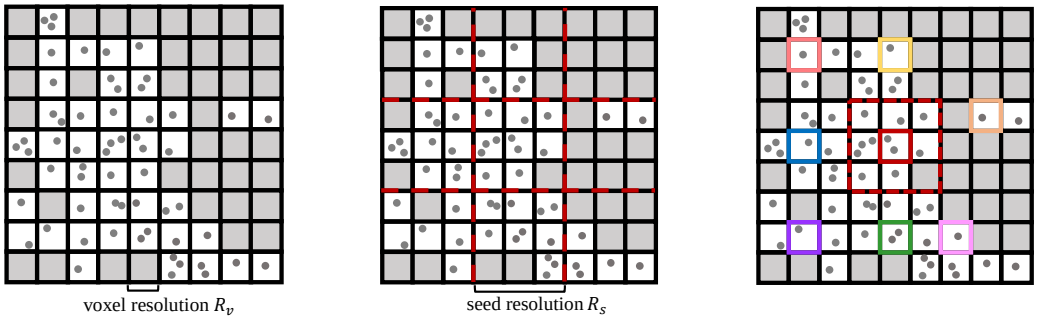


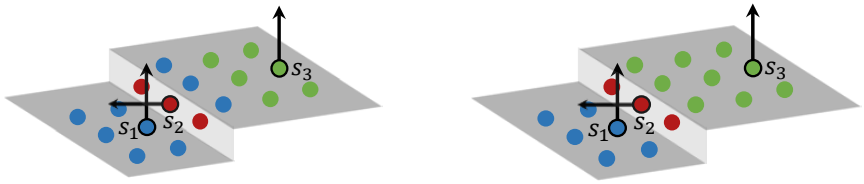
Figure 1: Illustration of points, voxels and seed voxels in 2D case. The original points of the input point cloud are shown as gray dots in (a), and we divide the space into voxels based on the voxel resolution R_v. Note that the small white grids with points inside are voxels. Given seed resolution R_s, we divide the space into grids in (b). We select a voxel closest to the center from each grid as the seed voxel.The seeds are labeled in different colors in (c). For a voxel in red dashed grid, we compute its distances to surrounding seeds and select the nearest one as its label.

Song et al. (2014) proposed a boundary-enhanced supervoxel segmentation (BESS) method for sparse outdoor data. It first detects the boundary points by analysing consecutive points and then expands cluster regions based on the neighborhood graph. This algorithm relies heavily on the detection of boundary points and is very sensitive to noise. To improve it, Kim and Park (2015) introduced a weighted neighborhood graph, and the method operates in a manner that minimizes the cost of the shortest path in the weighted graph. Lately, for better feature preservation, a method called BPSS was proposed by Lin et al. (2018), which formalizes supervoxel segmentation as a subset selection problem, and uses a heuristic method to minimize the objective function. The subset selection problem is NP-hard. The authors presented an approximate solution based on an energy descent method, which consists of fusion and exchange algorithms. Due to the sub-optimal solution of segmentation, the algorithm may fail for the detection of small details. In terms of algorithm efficiency, the BPSS method stores all points in a queue and updates the queue and energy function when all data points are processed, which makes the algorithm very time-consuming when the input data volume is large. Furthermore, sequential operations are difficult to accelerate in parallel on GPU. Xiao et al. (2020) proposed the MS method that is very sensitive to feature changes based on the idea of bottom-up greedy clustering. The MS method builds a global min-heap according to the merging cost of two adjacent supervoxels to store the processing order. After merging two supervoxels into a new supervoxel, it collects new merging pairs and updates the min-heap. The sequential processing makes it difficult to be parallelized. Although the authors tried a variety of acceleration strategies, the algorithm still takes a long time to process large point clouds.

3. Methodology

In this section, we briefly introduce our algorithm implemented on the GPU, which utilizes the parallel computing to generate supervoxels for point clouds. Given a point cloud $\mathbb{P}$ with m points, we first voxelize it into n voxels denoted as $\mathbb{V}$, then we compute the segmentation based on these voxels. As shown in Figure 1, we illustrate the points, voxels and seed voxels in 2D case. In Figure 1a, the points are shown in gray dots. We calculate the bounding box of the input data, and divide the space into voxels based on voxel resolution R_v. The coordinates and normal of a voxel v are the average values of all points inside. Given a lower seed resolution R_s, we divide the space into uniform grids in Figure 1b. Suppose there are k grids with points inside, and we select one voxel as the seed voxel from each grid. In our implementation for seed initialization, we select the voxel nearest to the center of grid as the seed voxel. The seed voxels are shown in different colors in Figure 1c. During segmentation, for each voxel in the red dashed grid, we calculate its distances with eight surrounding seeds and select the nearest one as its label. The labels of points inside a voxel are the same with it. We can easily extend the idea to 3D space for point cloud clustering, where for each voxel we need to compute its distances with surrounding seed voxels. This is the basic idea of our parallel algorithms. All voxels compute their distances with surrounding seeds and choose the nearest one as their labels on the GPU.

As mentioned above, we voxelize the Input Point Cloud $\mathbb{P}$ as voxel set $\mathbb{V}$. The supervoxel segmentation of $\mathbb{V}$ satisfies the following conditions: $\mathbb{V} = \cup S_i, i = 1, ...k$, and $S_i \cap S_j = \emptyset$ for $i \neq j$. A supervoxel S_i is composed of the following voxels: $S_i = \{v \in \mathbb{V} | D(v, S_i) < D(v, S_j), \forall j = 1, ...k, j \neq i\}$. The distance metric $D(v, S_i)$ between a voxel v



(a) (b)

Figure 2: Illustration for plane fitting distance. If we only use position and normal information for seed-based clustering, the supervoxel in blue may overlap two planes as shown in (a). With the plane fitting distance, we can get result in (b).

and a supervoxel S_i will be introduced in the next section. We design parallel algorithms to calculate the distance and generate high-quality supervoxel segmentation.

3.1. Objective Function

Our algorithm includes two stages: first it generates the supervoxels based on Lloyd (1982) iterations; then it optimizes the segmentation to improve the accuracy. Both stages are designed as parallel operations to ensure the efficiency of the algorithm. At the beginning of the algorithm, we voxelize the input data and use voxels as the basic processing unit. Then according to the seed resolution, we partition the data space into k uniform grids, each containing multiple voxels. We use the grid map G to represent the grid indices of all voxels. We choose a voxel from each grid as the seed voxel, note that the index of a grid is the same with the index of the seed in it. For grid i, there is a seed denoted as S_i inside it. Assuming voxel v is located in grid i, we want to calculate the distance between v and its surrounding seeds. We know that a seed has at most 26 neighbor seeds in 3D space. Together with S_i itself, we need to calculate the distance of v with 27 seeds at most. We record the neighbor relations of seeds in the matrix $\mathbf{M}_{nbr}$ with seed indices. The dimension of $\mathbf{M}_{nbr}$ is $k \times 27$. We denote the position and normal of voxel v as $(\mathbf{x}(v), \mathbf{n}(v))$. Denote by $\mathbf{x}(S_i)$ and $\mathbf{n}(S_i)$ the position and normal of supervoxel S_i, which can be from the seed voxel or the average position and normal of the cluster. The spatial distance and normal distance between v and S_i are defined by the following two equations, respectively:

$$D_p(v, S_i) = \|\mathbf{x}(v) - \mathbf{x}(S_i)\|^2,$$
$$D_n(v, S_i) = 1 - (\mathbf{n}(v) \cdot \mathbf{n}(S_i))^2 .$$

Our method is a seed-based clustering method. As shown in Figure 2a, there are three seeds with different colors. If we only utilize the position and normal information of the data, the blue seed may collect the voxels on the upper plane since the distance is small. To suppress this phenomenon, we add a metric called "plane fitting" to measure the distance between a voxel and the plane that the seed represents:

$$D_f(v, S_i) = (\mathbf{n}(S_i) \cdot (\mathbf{x}(v) - \mathbf{x}(S_i)))^2.$$

By combining this metric, we get the segmentation result in Figure 2b, which captures the object boundary better than that in Figure 2a. The final distance function combining above three metrics is as follows:

$$D(v, S_i) = D_f(v, S_i) + \lambda_1 D_p(v, S_i) + \lambda_2 D_n(v, S_i), \tag{1}$$

where λ_1 and λ_2 are the parameters to balance the relative importance of three metrics.

During segmentation, we calculate the distances between a voxel v ant its surrounding seeds. Based on the distance metrics above, we further organize the energies on a supervoxel S_i in following equations:

$$\begin{aligned} E_p(S_i) &= \sum_{v \in S_i} \|\mathbf{x}(v) - \mathbf{x}(S_i)\|^2 \\ &= Tr(\mathbf{M}(S_i)), \end{aligned}$$

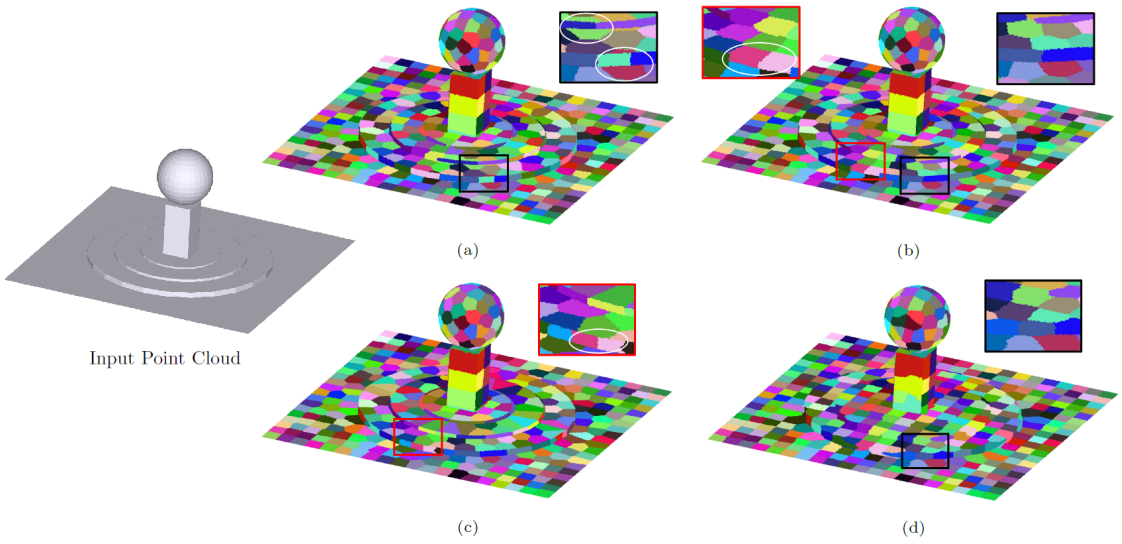


Figure 3: Supervoxel example of different stages. (a) Supervoxels obtained by considering spatial and normal distances at the clustering stage; the green and red supervoxels in the enlarged view overlap with two parallel planes. (b) Supervoxels obtained by combining the additional plane fitting energy at clustering stage; the problem in (a) is solved. However, the pink supervoxel in the magnified figure fails to detect the boundary. (c) Supervoxels after the swapping stage with three distance metrics; the pink supervoxel adheres to the boundary better. (d) Supervoxels by VCCS method, where the sharp features are not detected.

$$
\begin{aligned}
E_n(S_i) &= \sum_{v \in S_i} 1 - (\mathbf{n}(v) \cdot \mathbf{n}(S_i))^2 \\
&= |S_i| - \mathbf{n}(S_i)^T \mathbf{N}(S_i) \mathbf{n}(S_i), \\
E_f(S_i) &= \sum_{v \in S_i} (\mathbf{n}(S_i) \cdot (\mathbf{x}(v) - \mathbf{x}(S_i)))^2 \\
&= \mathbf{n}(S_i)^T \mathbf{M}(S_i) \mathbf{n}(S_i),
\end{aligned}
$$

where $|S_i|$ is the number of voxels in S_i, $\mathbf{M}(S_i)$ is the covariance matrix, and $\mathbf{N}(S_i)$ depends on the normal of voxels inside the supervoxel. They are defined in the following equations:

$$
\begin{aligned}
\mathbf{M}(S_i) &= \sum_{v \in S_i} (\mathbf{x}(v) - \mathbf{x}(S_i))(\mathbf{x}(v) - \mathbf{x}(S_i))^T, \\
\mathbf{N}(S_i) &= \sum_{v \in S_i} \mathbf{n}(v) \mathbf{n}(v)^T.
\end{aligned} \tag{2}
$$

From the above formulas, we can calculate the energies of a supervoxel S_i with its information represented by $\sigma(S_i) = \{\mathbf{x}(S_i), \mathbf{n}(S_i), \mathbf{M}(S_i), \mathbf{N}(S_i), |S_i|\}$. The total energy of the supervoxel S_i is the weighted summation of three energy terms, and we try to minimize the energy function in Equation 4 for the point cloud segmentation.

$$
E(S_i) = E_f(S_i) + \lambda_1 E_n(S_i) + \lambda_2 E_p(S_i), \tag{3}
$$

$$
E(\mathbb{V}) = \sum_i E(S_i). \tag{4}
$$

Algorithm 1: GPU-based Clustering algorithm

Input: A point cloud $\mathbb{P}$ of m points, voxel resolution R_v, supervoxel resolution R_s, and the maximum number of iterations $iter_m$.

Output: Supervoxel segmentation of the point cloud, $\{S_i\}$.

1 Voxelize the point cloud to n voxels based on R_v; `// CPU`
2 Select k voxels as seeds based on R_s;
3 Load the position $\mathbf{x}(v)$ and normal $\mathbf{n}(v)$ of all voxels to the GPU; `// CPU to GPU`
4 Load the grid map G of voxels and neighbor matrix $\mathbf{M}_{nbr}$ of seeds to the GPU; `// CPU to GPU`
5 Set $iter = 0$;
6 **while** $iter < iter_m$ **do**
7 **for** *each voxel* v **do** `// GPU`
8 Get grid index $g(v)$ where v is located, and the neighbor seeds of $g(v)$ from $\mathbf{M}_{nbr}$;
9 Compute the distance between voxel v and its surrounding seeds using Equation 1, and choose the nearest seed as its current label;
10 **end**
11 **for** *each supervoxel* S_i **do** `// GPU`
12 Calculate the average position of S_i;
13 **end**
14 Read the average value of position for all seeds to CPU memory; `// GPU to CPU`
15 **for** *each supervoxel* S_i **do** `// CPU`
16 Find the voxel closest to the average position as the new seed voxel;
17 Update the position $\mathbf{x}(S_i)$ and normal $\mathbf{n}(S_i)$ of the seed.
18 **end**
19 Load $\mathbf{x}(S)$ and $\mathbf{n}(S)$ to the GPU memory for next iteration; `// CPU to GPU`
20 **end**

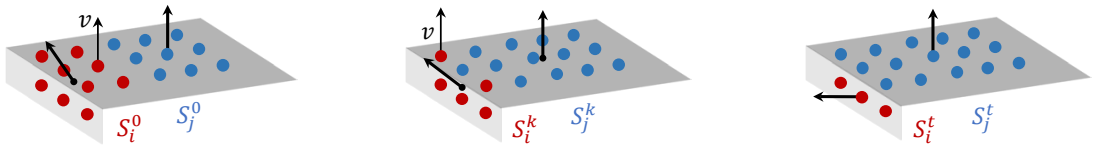
3.2. GPU-based Supervoxel Segmentation Algorithm

In this section, we will briefly introduce the algorithm for parallel computing. Our algorithm contains two stages. The first stage is to generate an initial segmentation using Lloyd iteration, which is a classic optimization method for seed-based clustering. The second stage is an optimization algorithm to further improve the quality by swapping voxels from one supervoxel to another to decrease the total energy.

3.2.1. GPU-based Clustering Algorithm

After voxelization, we compute the grid index of all voxels and the neighbor relationship of all seeds, denoted as grid map G and neighbor matrix $\mathbf{M}_{nbr}$, respectively. For a voxel v, we represent the grid that it belongs to by $g(v)$. According to $\mathbf{M}_{nbr}$, we obtain the surrounding seeds $nbr(g(v))$ of grid $g(v)$. We then calculate the distance between v and $nbr(g(v))$, and select the nearest one as its label. The framework is shown in Algorithm 1. We adopt Lloyd method for the seed-based clustering by iterating the following steps: first the algorithm computes supervoxel segmentation by fixing the seeds; then for each supervoxel, update the seed to be the voxel closest to the centroid of supervoxel for next iteration. To calculate the distance between voxels and supervoxels, we use the position and normal of the seed voxel as the supervoxel information, because that using voxels as cluster centers is more conducive to detecting the boundary of the plane where it is located. In steps #7 to #13, we compute the distances of voxels and surrounding seeds in parallel on the GPU. After step #14, we get the average position of each supervoxel, which may not lie within a valid voxel. We use this position as center to find the nearest voxel from near to far. Since only a few average positions do not locate in the voxel, the sequential operations in steps #15 to #18 perform very quick. In step #19 we update the seed information and reload data to GPU for next iteration. Here we only transmit the seed information between the CPU and GPU, which has a small impact on the efficiency of the algorithm. At the end, we get the supervoxel segmentation using Lloyd iteration.

The Figure 3(a) and 3(b) show the supervoxels at clustering stage. In Figure 3(a), the algorithm only uses spatial and normal distance metrics. We notice that the supervoxels in green and red color overlap two parallel planes. By



(a) (b) (c)

Figure 4: Illustration for swapping operation. We denote the seed position and normal as an arrow with end point. The initial segmentation is shown in (a). After one swapping iteration, the position and normal of seeds are updated. The supervoxels in (c) adhere better to the object boundary.

adding plane fitting constraint, the result in Figure 3(b) will no longer cross parallel planes, and the segmentation is more accurate. However, it still cannot detect the boundary accurately sometimes. For example, the magnified details in red box show that the pink supervoxel crosses two perpendicular planes, thus the upper and lower bounds of the step are not detected. Based on the above observations, we need to further optimize the supervoxel results after clustering to improve the performance on boundary adherence.

3.2.2. GPU-based Optimization Algorithm

In this section, we propose an effective strategy called swapping to further optimize the segmentation. We try to decrease the segmentation energy by swapping voxels from one supervoxel to another. Suppose we try to swap a voxel v from its current supervoxel S_i to other neighbor supervoxel S_j. We denote this operation as $(S_i, S_j) \xrightarrow{v} (S_i', S_j')$, where $S_i' = S_i - v$ and $S_j' = S_j + v$. The energy change is denoted as $\delta E = \delta E(S_i) + \delta E(S_j) = E(S_i') + E(S_j') - E(S_i) - E(S_j)$. We only perform swapping operations that can reduce the energy of the segmentation, i.e., $\delta E < 0$. After clustering stage, we calculate the information of each supervoxel S_i, denoted as $\sigma(S_i) = \{\mathbf{x}(S_i), \mathbf{n}(S_i), \mathbf{M}(S_i), \mathbf{N}(S_i), |S_i|\}$. Here, we need to calculate the energy change of the supervoxel when adding or subtracting a voxel. We set $\mathbf{x}(S_i)$ and $\mathbf{n}(S_i)$ to be the average position and normal of the supervoxel.

Based on the definition of E_p, E_n and E_f, we can easily formulate the energy terms after swapping. Taking S_i for example, the covariance matrix $\mathbf{M}(S_i')$ and normal matrix $\mathbf{N}(S_i')$ after swapping is calculated by the following formulas:

$$\mathbf{x}(S_i') = \frac{|S_i|\mathbf{x}(S_i) - \mathbf{x}(v)}{|S_i| - 1}, \tag{5}$$

$$\begin{aligned}\mathbf{M}(S_i') =& \mathbf{M}(S_i) - (|S_i| - 1)(\mathbf{x}(S_i') - \mathbf{x}(S_i))(\mathbf{x}(S_i') - \mathbf{x}(S_i))^T \\ & - (\mathbf{x}(v) - \mathbf{x}(S_i))(\mathbf{x}(v) - \mathbf{x}(S_i))^T,\end{aligned} \tag{6}$$

$$\mathbf{N}(S_i') = \mathbf{N}(S_i) - \mathbf{n}(v)\mathbf{n}(v)^T. \tag{7}$$

Similarly, we can calculate the new matrices for supervoxel S_j. The energy change can be calculated in $O(1)$ time, which does not depend on the number of voxels in each supervoxel.

Given seed information $\sigma(S_i)$ and $\sigma(S_j)$, the swapping operation is illustrated in Figure 4. We denote the seed as an arrow with end point, the position and normal of seed are average value of the supervoxel. For a voxel v in Figure 4(a), swapping it to the blue supervoxel will obviously decreases the normal and plane fitting energy. In Figure 4 (a) and (b), we try to swap every voxel from its current supervoxel to the adjacent supervoxel to decrease the energy. After several swapping iterations, the supervoxels in Figure 4(c) segment object more accurately. For easier computing, we keep the normal of a supervoxel $n(S_i)$ unchanged for swapping a voxel. During the optimization stage, tuning the parameters λ_1 and λ_2 may speed up the energy convergence, but it does not significantly improve the results. In all experiments, we do not adjust the parameters separately for different point cloud models, and use the same parameter setting for models in the same dataset.

The swapping operations for the voxel are performed in parallel, the pseudo code is shown in Algorithm 2. After clustering, we compute the seed information $\sigma(S)$ on GPU. According to Equation 2, these information can be obtained by the sum operation, which is a common operation in GPU parallel strategies. We represent the energy of the

Algorithm 2: GPU-based Optimization algorithm

Input: The initial segmentation after clustering, and the maximum number of iterations $iter_s$.
Output: Optimized segmentation.

Set $iter = 0$;
while $iter < iter_s$ **do**
  Compute the information of seeds $\sigma(S)$ based on parallel computing on GPU; `// GPU`
  for *each voxel* v **do** `// GPU`
    Get its label $l(v)$ from segmentation result, suppose $l(v) = S_i$;
    Get the grid index $g(v)$ that v is located;
    Initialize swapping cost $sc(v)$ and current closet seed $t(v)$;
    for *each neighbor seed* S_j *of grid* $g(v)$ *in* $\boldsymbol{M}_{nbr}$ **do**
      Calculate the cost of swapping v from the current seed S_i to neighbor seed S_j;
      Record the current minimum swapping cost to $sc(v)$;
      Update the current nearest seed to $t(v)$;
    end
    if $sc(v) < 0$ **then**
      Update the closest seed $l(v) = t(v)$;
    end
  end
end

supervoxel as a lightweight data structure $\sigma(S)$, which can be easily loaded into the GPU memory for the subsequent swapping processing. The steps from #4 to #16 are parallel computing for each voxel on the GPU. We show the effectiveness of swapping optimization in Figure 3(c). The pink supervoxel in the red box of Figure 3(b) fails to detect the upper and lower bounds of the step. After swapping, the supervoxels near boundary are moved to the vertical plane of the step. The swapping optimization significantly improves the quality of supervoxel segmentation. We show an example of supervoxel segmentation in Figure 5. VCCS method can not segment the roof of the pavilion and the fence on the roadside very accurate. Our algorithm generates good segmentation compared with other methods.

4. Experiments

We compare our method with the advanced supervoxel segmentation methods, including VCCS Papon et al. (2013), BPSS Lin et al. (2018) and MS Xiao et al. (2020) methods. We implemented our algorithm in C++, and compared it with other algorithms on a PC with Intel Core I9-9920X and NVIDIA GeForce RTX 2080 SUPER. The parallel computing is implemented by OpenGL Shading Language (GLSL). We name our method as **GPU-SS** (for GPU-based Supervoxel Segmentation) algorithm. We have released our code [1].

In this section, we compare our method with the existing methods on three public datasets: Oakland Munoz et al. (2009), Semantic3D Hackel et al. (2017), and NYUV2 Silberman et al. (2012), varying from outdoor scenes to indoor scenes. Similar to superpixels, the supervoxels for point cloud should preserve the object boundaries and not overlap multiple objects. The boundary recall and segmentation error metrics reflect the quality of supervoxels. As a pre-processing technology, the efficiency is also important. Usually we expect an algorithm to give segmentation results in seconds for post application.

4.1. Implementation Details

In the implementation, we first voxelize the input data based on the voxel resolution R_v. According to previous work Lin et al. (2018); Papon et al. (2013), we can set the value of R_v manually or calculate the value of R_v based on the desired voxel size. Given the number of points m and the size of the bounding volume for a point cloud model,

[1] https://github.com/dongxiao0401/GPUSupervoxelForPointCloud

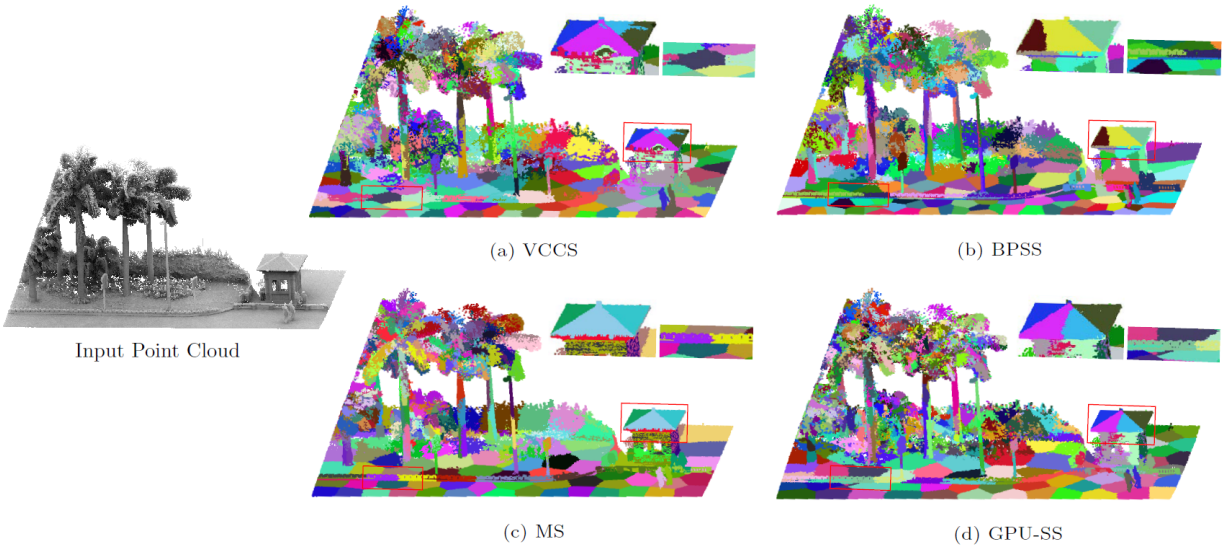


Figure 5: Supervoxel results with different methods. VCCS fails to detect the boundaries of objects. Our method GPU-SS and other two methods generate segmentation with good quality. The boundary recall values of four methods are (0.562, 0.608, 0.616, 0.603), respectively.

based on experience, we can restrict the average number of points inside each voxel to be 20. According to the length, width and height of the bounding volume, we can calculate the value of voxel resolution R_v. The seed resolution R_s determines the number of supervoxels in the point cloud model. In experiments for performance evaluation, the seed resolution R_s is equal to the average size of supervoxels that is specified by the user. In general, the smaller the R_s value, the greater the number of supervoxels, and the higher the performance of the algorithm. In terms of parameters λ_1 and λ_2 in distance function that controls relative importance of different energy terms, we adopt same setting for all point cloud models in one dataset.

4.2. Evaluation Metrics

To quantitatively evaluate the quality of supervoxel segmentation results, we adopt several evaluation metrics Xiao et al. (2020) to measure the supervoxel segmentation of different methods, such as running time, boundary recall (BR), under-segmentation error (UE) and global consistency error (GCE). Before demonstrating the comparison results, we briefly introduce the concept of evaluation metrics.

Efficiency is very important for point cloud segmentation as a pre-processing technique. We test the running time for various algorithms to process point clouds of different sizes. In the statistics, we did not consider the time required to load data and save the results. In addition, we test the running time of different stages in our algorithm to show the efficiency of parallel computing.

Boundary recall measures the percentage of boundary points in ground truth detected by the supervoxel boundaries Martin et al. (2004). A point is called a boundary point if the label of one of its k-nearest neighbor points is different with it. The high boundary recall value means that the segmentation is able to preserve boundary features of the point cloud.

Under-segmentation error is to punish the supervoxel across multiple ground truth segments Levinshtein et al. (2009). For each ground truth segment, we find all supervoxels intersecting with it, and then calculate the number of points outside the ground truth region. The low under-segmentation error means that most supervoxels overlap with one ground truth segment and do not cross the object boundary.

Global consistency error is an object-level metric based on the intersection of the supervoxel and the ground truth segment, which simultaneously evaluates over-segmentation error and under-segmentation error of the supervoxel Martin et al. (2001). It first defines the ground truth to supervoxel error and the supervoxel to ground truth error, and then calculates the total number of intersecting points between the ground truth and supervoxels. GCE value is regularized to range $[0, 1]$, where 0 indicates no error and 1 indicates worst segmentation.

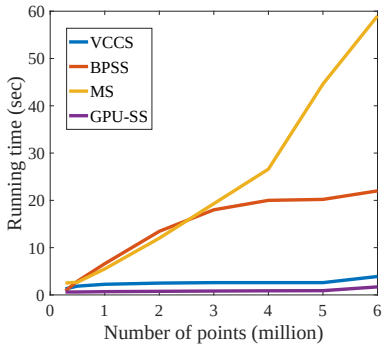


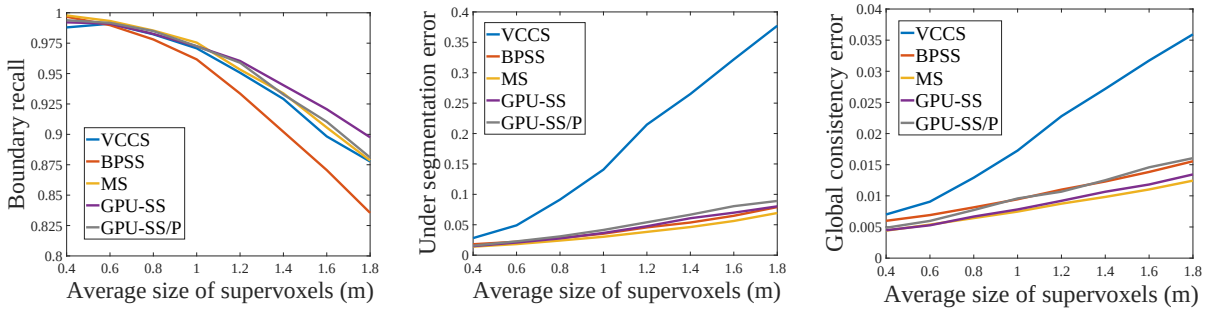
Figure 6: Running time with respect to points number. Our method outperforms the existing methods. For a point cloud with 5 million points, our method only takes 1 second, and VCCS method takes about 2.5 seconds.

Number of points	Stages				Total
	Voxelization	Load(GPU)	Clustering	Optimization	
1 million	0.112	0.56	0.014	0.008	0.694
3 million	0.25	0.56	0.019	0.013	0.842
5 million	0.366	0.57	0.019	0.016	0.971

Table 1: Run time (seconds) of each stage of the algorithm for point cloud with different number of points.

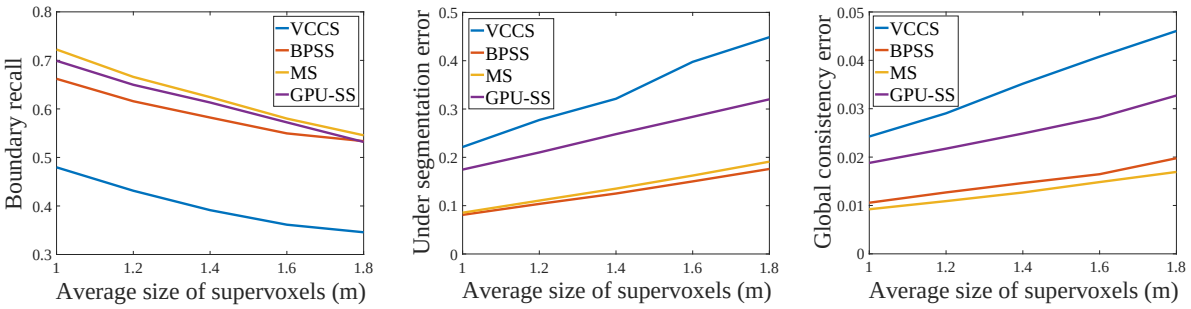
4.3. Performance

Running time is an important indicator of whether supervoxels are used in subsequent applications. Many post-processing tasks Boularias et al. (2015); Zhang et al. (2015); Zhu et al. (2017) use VCCS to segment point clouds mainly because VCCS runs fast. For the point cloud with 5 million points in Figure 3, our method only requires 1 second, VCCS requires 2.5 seconds, BPSS takes about 38 seconds, and MS method takes about one minute. The time cost versus the number of points for algorithms are plotted in Figure 6. We can see that as the number of points increases, the BPSS and MS method require tens of seconds to generate the segmentation. In Table 1, we sub-sample a point cloud to 1, 3 and 5 million points, and show the processing time of our method in different stages. Most of the time of is spent on voxelization and loading data into GPU memory. The parallel operations in clustering and optimization algorithms are very efficient. This also means that we can try other parallel strategies to further improve the segmentation quality in the future. The huge time cost of BPSS and MS methods is unbearable for many applications, especially those with real-time requirement. For many computer vision tasks, our method can replace VCCS method to pre-segment the point cloud, which improves both the speed and quality.



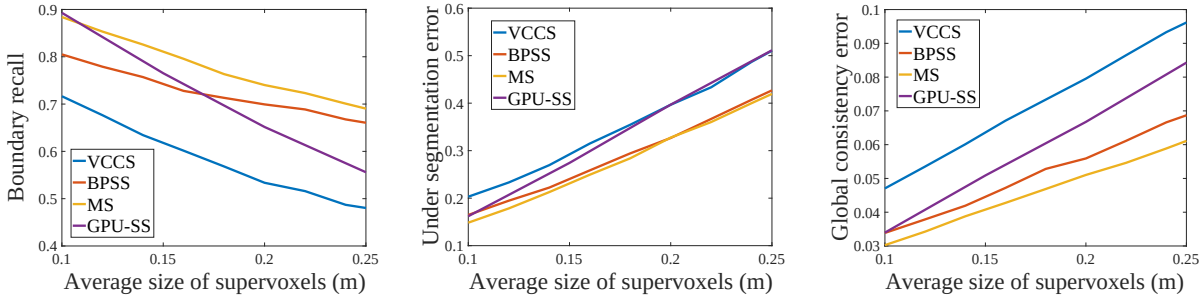
(a) (b) (c)

Figure 7: Experimental results on Oakland dataset. Our method GPU-SS has high performance on three metrics. The method GPU-SS/P shows the performance of our method without plane fitting energy term.



(a) (b) (c)

Figure 8: Experimental results on Semantic3D dataset. Our method is much better than VCCS on three evaluation metrics, especially for boundary recall performance.



(a) (b) (c)

Figure 9: Experimental results on NYUV2 dataset. Our method is much better than VCCS method.

Performance are reflected by boundary recall, under-segmentation error and global consistency error evaluation metrics. In Figure 7, we evaluate the performance of our method GPU-SS, VCSS, BPSS and MS methods on Oakland dataset Munoz et al. (2009). In addition, we show the performance of ablation experiment without the plane fitting energy term on this dataset, which is denoted as GPU-SS/P method. From the evaluation we can see that the plane fitting energy term helps to improve segmentation quality, especially in reducing segmentation errors. In terms of three evaluation metrics, our method GPU-SS has achieved the best performance on this dataset. The Oakland dataset contains outdoor scenes that mainly consists of buildings, cars and trees. We show an example of Oakland dataset in the first column of Figure 10. Compared with other point clouds, the models in this dataset have more planes perpendicular to each other and contain fewer noise. The optimization framework in our algorithm tries to preserve building boundaries with the plan fitting constraint, which is suitable for the feature detection of outdoor scenes.

In addition to Oakland dataset, we also evaluate our method on Semantic3D Hackel et al. (2017) and NYUV2 Silberman et al. (2012) datasets in Figure 8 and Figure 9. Our method has high performance on boundary recall for Semantic3D dataset. The performance of our method on the NYUV2 dataset is comparable to other methods. You can try different settings of parameters λ_1 and λ_2 in application. One problem of NYUV2 dataset is that there are a lot of noise and mislabeled ground truth segments in the point cloud converted from RGB-D image. In the second column of Figure 10, we show an example of indoor scene. We can see that the artificially labeled boundaries are basically accurate, but the ground truth segments are not always correct. In addition, the points on the floor should be on the same plane. But these points are projected from the RGB-D image in the camera coordinate system, resulting in a large difference between their coordinates and normal. These noises are not conducive to the algorithm to accurately segment objects. And because the ground truth is not accurate enough, the evaluation results can only be treated as a reference for the quality of the supervoxels. In the evaluation, our method outperforms VCCS method on NYUV2, especially on boundary recall metric.

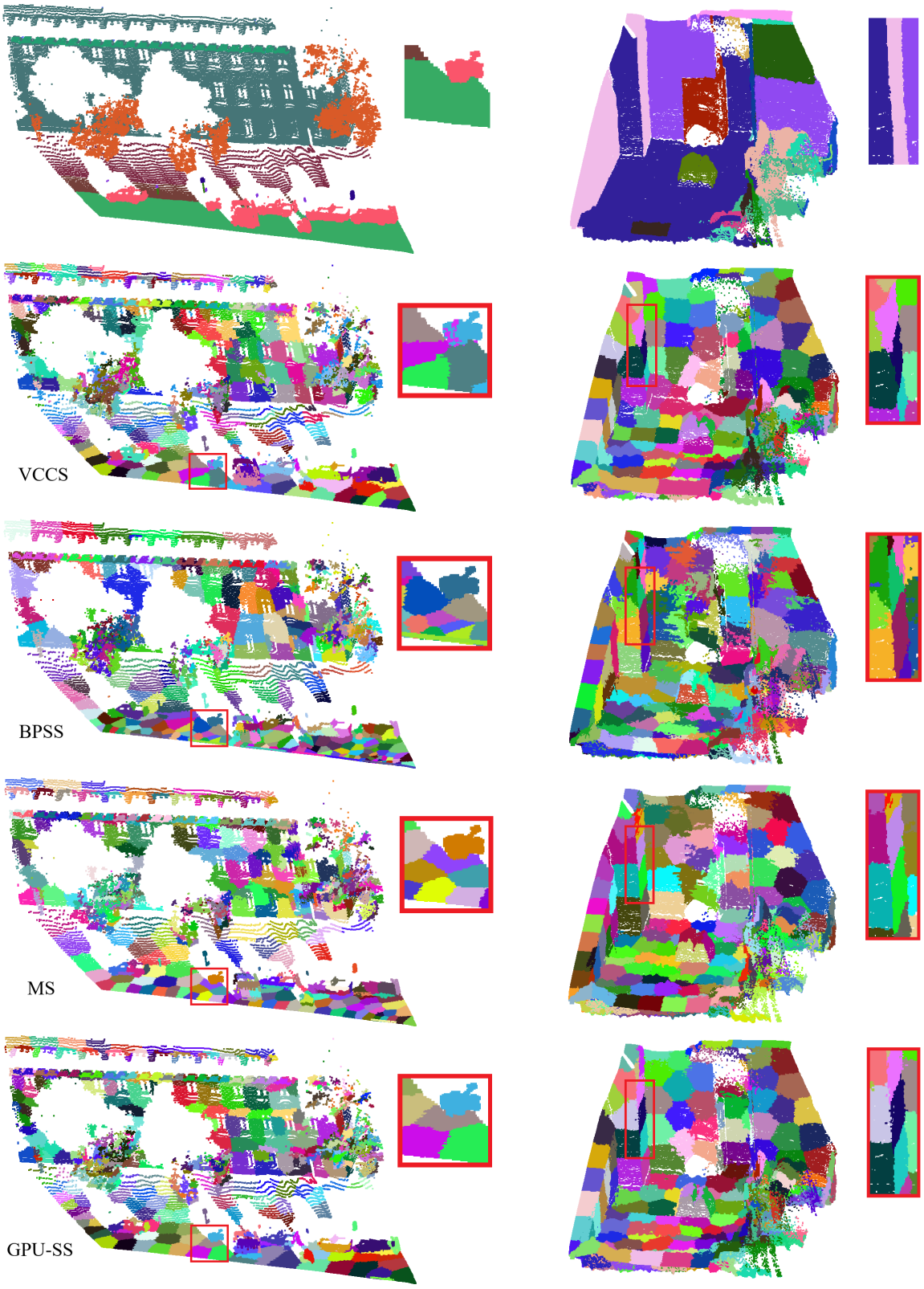


Figure 10: Visual comparison of different methods. The first row is the ground truth of the point clouds from Oakland and NYUV2 datasets. The second to fourth rows are the segmentation obtained by VCCS, BPSS, MS and GPU-SS methods, respectively. The boundary recall values of the methods are (0.822,0.0.819,0.0.808,0.826) and (0.461,0.469,0.528,0.485) for these two point clouds. VCCS fails to preserve object boundaries. Our method generates good segmentation as the state-of-the-arts.

Visual examples of above datasets are shown in Figure 10. We show the ground truth of point clouds in Oakland and NYUV2 datasets in the first row. We also demonstrate the boundary recall values of different methods on the examples. For the point cloud of indoor scene, it is clear that VCCS fails to detect the boundaries of the wall. Our method has high performance on boundary detection, and it can accurately segment objects located in different planes. From the enlarged views we can see that, VCCS can not detect small objects and boundaries accurately. BPSS generates supervoxels with uneven size. MS method generates result with high quality, however, the time consuming is much larger than ours.

5. Conclusions

In this paper, we propose a novel GPU-based supervoxel method for point cloud segmentation. Our algorithm consists of two stages: clustering and optimization. The clustering stage generates initial supervoxel segmentation by a seed-based clustering method, and the optimization stage further improves the result by swapping voxels to neighboring seeds to decrease the segmentation energy. Our algorithms are designed as parallel operations on GPU, while other methods such as VCCS, BPSS and MS contain sequential processing, which are difficult to convert to parallel computing. Our algorithm gives a way to calculate supervoxels on GPU, which guarantees the speed and quality of segmentation at the same time. We provide the performance comparisons on three public datasets and several visual examples. Experiments demonstrate that our method produces supervoxels with the fastest speed while ensuring good segmentation quality.

Acknowledgements

The research of Xiaohu Guo was partially supported by a grant from National Science Foundation (No. 2007661). The research of Yanyang Xiao was supported by the National Natural Science Foundation of China (No. 62102174). The research of Zhonggui Chen was supported by the National Natural Science Foundation of China (Nos. 61972327, 61872308), the Open Project Program of State Key Laboratory of Virtual Reality Technology and Systems, Beihang University (No. VRLAB2021B01), and the Fundamental Research Funds for the Central Universities (No. 20720190011). Junfeng Yao was supported by the National Natural Science Foundation of China (No. 62072388), the Collaborative Project Fund of Fuzhou-Xiamen-Quanzhou Innovation Zone (No. 3502ZCQXT202001), the Industry Guidance Project Foundation of Science Technology Bureau of Fujian Province in 2020 (No. 2020H0047), the Natural Science Foundation of Science Technology Bureau of Fujian Province in 2019 (No. 2019J01601), the Creation Fund Project of Science Technology Bureau of Fujian Province in 2019 (No. 2019C0021) and the Fujian Sunshine Charity Foundation.

References

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Süsstrunk, S., 2012. SLIC superpixels compared to state-of-the-art superpixel methods. IEEE Transactions on Pattern Analysis and Machine Intelligence 34, 2274–2282.

Boularias, A., Bagnell, J.A., Stentz, A., 2015. Learning to manipulate unknown objects in clutter by reinforcement, in: Twenty-Ninth AAAI Conference on Artificial Intelligence.

Cai, Y., Guo, X., 2016. Anisotropic superpixel generation based on mahalanobis distance, in: Computer Graphics Forum, Wiley Online Library. pp. 199–207.

Dong, X., Chen, Z., Liu, Y.J., Yao, J., Guo, X., 2021. GPU-based supervoxel generation with a novel anisotropic metric. IEEE Transactions on Image Processing 30, 8847–8860.

Dong, X., Chen, Z., Yao, J., Guo, X., 2019. Superpixel generation by agglomerative clustering with quadratic error minimization, in: Computer Graphics Forum, Wiley Online Library. pp. 405–416.

Dubé, R., Dugas, D., Stumm, E., Nieto, J., Siegwart, R., Cadena, C., 2017. Segmatch: Segment based place recognition in 3D point clouds, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 5266–5272.

Hackel, T., Savinov, N., Ladicky, L., Wegner, J.D., Schindler, K., Pollefeys, M., 2017. Semantic3D. NET: A new large-scale point cloud classification benchmark. arXiv preprint arXiv:1704.03847 .

Huang, S.S., Ma, Z.Y., Mu, T.J., Fu, H., Hu, S.M., 2021. Supervoxel convolution for online 3D semantic segmentation. ACM Transactions on Graphics (TOG) 40, 1–15.

Kim, J.S., Park, J.H., 2015. Weighted-graph-based supervoxel segmentation of 3D point clouds in complex urban environment. Electronics Letters 51, 1789–1791.

Levinshtein, A., Stere, A., Kutulakos, K.N., Fleet, D.J., Dickinson, S.J., Siddiqi, K., 2009. Turbopixels: Fast superpixels using geometric flows. IEEE Transactions on Pattern Analysis and Machine Intelligence 31, 2290–2297.

Li, Z., Chen, J., 2015. Superpixel segmentation using linear spectral clustering, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1356–1363.

Lin, Y., Wang, C., Zhai, D., Li, W., Li, J., 2018. Toward better boundary preserved supervoxel segmentation for 3D point clouds. ISPRS Journal of Photogrammetry and Remote Sensing 143, 39–47.

Liu, Y.J., Yu, M., Li, B.J., He, Y., 2017. Intrinsic manifold SLIC: A simple and efficient method for computing content-sensitive superpixels. IEEE Transactions on Pattern Analysis and Machine Intelligence 40, 653–666.

Lloyd, S., 1982. Least squares quantization in PCM. IEEE Transactions on Information Theory 28, 129–137.

Luo, H., Wang, C., Wen, C., Cai, Z., Chen, Z., Wang, H., Yu, Y., Li, J., 2015. Patch-based semantic labeling of road scene using colorized mobile LiDAR point clouds. IEEE Transactions on Intelligent Transportation Systems 17, 1286–1297.

Martin, D., Fowlkes, C., Tal, D., Malik, J., 2001. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics, in: Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001, IEEE. pp. 416–423.

Martin, D.R., Fowlkes, C.C., Malik, J., 2004. Learning to detect natural image boundaries using local brightness, color, and texture cues. IEEE Transactions on Pattern Analysis and Machine Intelligence 26, 530–549.

Munoz, D., Bagnell, J.A., Vandapel, N., Hebert, M., 2009. Contextual classification with functional max-margin Markov networks, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE. pp. 975–982.

Pan, X., Zhou, Y., Li, F., Zhang, C., 2016. Superpixels of RGB-D images for indoor scenes based on weighted geodesic driven metric. IEEE Transactions on Visualization and Computer Graphics 23, 2342–2356.

Papon, J., Abramov, A., Schoeler, M., Worgotter, F., 2013. Voxel cloud connectivity segmentation-supervoxels for point clouds, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2027–2034.

Ren, C.Y., Prisacariu, V.A., Reid, I.D., 2015. gSLICr: SLIC superpixels at over 250hz. arXiv preprint arXiv:1509.04232 .

Silberman, N., Hoiem, D., Kohli, P., Fergus, R., 2012. Indoor segmentation and support inference from RGBD images, in: European Conference on Computer Vision, Springer. pp. 746–760.

Song, S., Lee, H., Jo, S., 2014. Boundary-enhanced supervoxel segmentation for sparse outdoor LiDAR data. Electronics Letters 50, 1917–1919.

Sun, Z., Xu, Y., Hoegner, L., Stilla, U., 2018. Classification of mls point clouds in urban scenes using detrended geometric features from supervoxel-based local contexts. ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences 4.

Wang, H., Wang, C., Luo, H., Li, P., Chen, Y., Li, J., 2015. 3-D point cloud object detection based on supervoxel neighborhood with Hough forest framework. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 8, 1570–1581.

Wang, W., Shen, J., Shao, L., 2017. Video salient object detection via fully convolutional networks. IEEE Transactions on Image Processing 27, 38–49.

Weikersdorfer, D., Gossow, D., Beetz, M., 2012. Depth-adaptive superpixels, in: Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), pp. 2087–2090.

Xiao, Y., Chen, Z., Lin, Z., Cao, J., Zhang, Y.J., Lin, Y., Wang, C., 2020. Merge-swap optimization framework for supervoxel generation from three-dimensional point clouds. Remote Sensing 12, 473.

Yang, F., Lu, H., Yang, M.H., 2014a. Robust superpixel tracking. IEEE Transactions on Image Processing 23, 1639–1651.

Yang, J., Gan, Z., Gui, X., Li, K., Hou, C., 2013. 3-D geometry enhanced superpixels for RGB-D data, in: Pacific-Rim Conference on Multimedia, Springer. pp. 35–46.

Yang, J., Gan, Z., Li, K., Hou, C., 2014b. Graph-based segmentation for RGB-D data using 3-D geometry enhanced superpixels. IEEE Transactions on Cybernetics 45, 927–940.

Yun, J.S., Sim, J.Y., 2016. Supervoxel-based saliency detection for large-scale colored 3D point clouds, in: 2016 IEEE International Conference on Image Processing (ICIP), IEEE. pp. 4062–4066.

Zhang, R., Candra, S.A., Vetter, K., Zakhor, A., 2015. Sensor fusion for semantic segmentation of urban scenes, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 1850–1857.

Zhu, Q., Li, Y., Hu, H., Wu, B., 2017. Robust point cloud classification based on multi-level semantic relationships for urban scenes. ISPRS Journal of Photogrammetry and Remote Sensing 129, 86–102.