An extension of the Felzenszwalb-Huttenlocher segmentation to 3D point clouds

Mihai-Cotizo Sima and Andreas Nüchter

Automation Group, School of Engineering and Science,
Jacobs University Bremen gGmbH, 28759 Bremen, Germany

ABSTRACT

This paper investigates the segmentation algorithm proposed by Felzenszwalb and Huttenlocher[1] and its compatibility to 3D point clouds acquired with state-of-the-art 3D laser scanners. To use the algorithm, we adapt the range and intensity data to the smoothed graph structure used by the algorithm. We investigate the influence of the algorithm's parameters to its performance and result that are meaningful to both the machines and the humans.

Keywords: graph-based segmentation, 3D point clouds, 3D laser scanner

1. INTRODUCTION

The three core components of human perception are grouping, constancy, and contrast effects.[2] Segmentation in machine vision approaches this natural way of observing the world by splitting images in components with constant attributes, and grouping them together. This paper focuses on segmentation of 3D point clouds. The resulting approach for segmentation is reliable and efficient. It combines recent results in computer vision with the emerging technology of 3D laser scanners.

The problem of segmentation is one of the main research areas of computer vision; many existent algorithms produce satisfying results based on different local characteristics of the images. Felzenszwalb and Huttenlocher propose a graph-based algorithm which, despite its greedy-approach, satisfies global properties by identifying not only constant intensity regions, but also regions of high variations or gradients.[1] We push forward this implementation, by switching from the 2D images to 3D point clouds, and investigating the performance of the algorithm in this situation. We expect that, since this segmentation approach is based on graphs which are inherently dimensionless, no impact on the performance will be observed. However, it is important to investigate the influence of the algorithm parameters, and whether there are other hidden constraints still to be added.

2. RELATED WORK AND PRELIMINARIES

Image segmentation is the process through which a set of contours or segments are extracted from the features of an image. An important requirement is that the pixels in the interior of a region are similar with respect to a characteristic or property, but adjacent regions are kept significantly different. Much work has been conducted in the area of image segmentation and even of segmenting range images. For the latter there exists an excellent overview and comparison.[3] Segmenting 3D point clouds is an emerging topics, as the recent introduction of kinect-like devices and time-of-flight cameras give a boost to applications requiring 3D point cloud processing.[4–6] Many of these approaches are similar to the basic technologies described below and/or rely purely on planar structures. We aim at a one-to-one transfer of the Felzenszwalb-Huttenlocher segmentation[1] to 3D point clouds. This section concludes with a description of this method.

There are several basic segmentation methods existent at the moment. One of the most basic ones involve thresholding a gray scale image, transforming it to a binary image based on the "clipping level". One efficient method of thresholding was designed by Otsu.[7] His algorithm splits the image into two classes based on a minimum intra-class variance. Otsu's method is regarded as a histogram-based method. Generally, this technique requires computing the histogram out of all the pixels in the image and then using its shape, i.e., local maxima

Further author information: Andreas Nüchter, E-mail: andreas@nuechti.de, Telephone: +49 177 7951270

and minima, to produce clusters. Moreover, to refine the clusters, one will recursively apply the method to each cluster. An important further basic clustering algorithm is the k-means clustering. This approach is not limited only to 2D or 3D imagery, but rather to general feature-spaces. It divides n entities into k clusters such that each cell is a cluster around the nearest mean. The standard k-means algorithm has no guarantee that it will converge. The choice of the initial seeds severely influences the termination of the algorithm and the quality of the output.

Segmentations are also achieved by following intensity transitions that are not smooth, as they should be in natural regions. These transitions are detected as edges; however, an important problem is that edges usually do not form closed shapes, and therefore additional algorithms connecting the edges are needed to produce segmentations.

A further classic approach is growing regions from a set of seed points, by adding neighboring pixels to regions for which they are closest to the mean. Initially, the seed points were input by the user, to indicate the objects of interest in the image. An alternative method starts from a single region and, while picking pixels neighboring the frontier, decide if a new region is to be created based on whether their deviation is larger than a threshold value or not.

Graph-based segmentation methods generate a graph based on the pixels of the image and their properties, with the weights of edges connecting neighboring pixels proportional to the level of similarity of the two pixels. These nodes are then clustered based on a similarity criterion. One method was discussed by Shi and Malik in.[8] Their segmentation method searches for minimal cut by splitting the graph in k sub-graphs such that the maximum of the cuts is minimum. This approach has the deficit of favoring small isolated sets, i.e., the more edges are cut, the higher the cost of the cut. To fix this, the authors propose a normalized cut. This cut-based approach is effective in preserving the non-local properties of the image; however, it focuses only on the characteristics of a cut rather than of the whole segmentation.[1]

Felzenszwalb and Huttenlocher proposed an approach that can be extended to general feature spaces and not only treats the local properties, but also the global characteristics of an image.[1] They use the weight of an edge $w(v_i, v_j)$ between two vertices as the level of dissimilarity of two points. (difference in intensity, range, color, location etc) A segmentation splits nodes such that edges inside a component have relatively low weights and edge between components have relatively high weights. For every component, the internal difference is defined as the maximum edge weight in the minimum-spanning tree (MST) of the component:

$$\mathrm{Int}(C) = \max_{e \in MST(C)} w(e) \tag{1}$$

The external difference is, analogously, the minimum weight of an edge connecting a vertex from one component and a vertex from another component:

$$\mathrm{Dif}(C_1, C_2) = \min_{v_1 \in C_1, v_2 \in C_2, (v_1, v_2) \in E} w((v_1, v_2)) \tag{2}$$

One can replace the condition $(v_1, v_2) \in E$ in Eq. (2) with a constraint on the weights of nonexistent edges, such that $w((v_1, v_2)) = \infty$ if $(v_1, v_2) \notin E$. An alternative definition for Eq. (2) uses the median weight and not the smallest one, in order to make it robust to outliers. A thresholding function is used to make two components distinct based on their sizes: more exactly, for small components, a stronger evidence for a boundary is needed for the components to be distinct. The function is defined as:

$$\tau(C) = \frac{k}{|C|} \tag{3}$$

The parameter k sets preference for large or small components, depending on whether its value is large or small. However, it does not represent a minimum component size. Based on these quantities, the minimum internal difference of two components is defined as:

$$\mathrm{MInt}(C_1, C_2) = \min\{\mathrm{Int}(C_1) + \tau(C_1), \mathrm{Int}(C_2) + \tau(C_2)\} \tag{4}$$

Furthermore, the central predicate, which indicates whether there is evidence of a boundary between two components, is defined as:

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } \mathrm{Dif}(C_1, C_2) > \mathrm{MInt}(C_1, C_2) \\ \text{false} & \text{otherwise.} \end{cases} \tag{5}$$

To decide on the effectiveness of the segmentation algorithm, we must know whether it is too coarse or too fine:

1. A segmentation is too fine if there exists a pair of regions for which there is no evidence of a boundary between them.

2. A segmentation is too coarse if there exists a refinement of the segmentation which is not too fine.

It can be proven that, for any graph, a segmentation exists such that it is neither too fine, nor too coarse. The proposed algorithm that constructs the segmentation is presented in Algorithm 1.

Algorithm 1: FH Segmentation

```
Sort all the edges according to their weight;
Assign each vertex to its own component for the initial segmentation S^0;
forall edges e_i = (v_1, v_2) ∈ E do
    Consider v_1 ∈ C_1 and v_2 ∈ C_2 (the two distinct components containing the vertices of the edge, as
    they exist in the previous step S^{i-1});
    if w(e_i) ≤ MInt(C_1, C_2) then
        Merge C_1 and C_2 in S^i;
    else
        S^i = S^{i-1};
```

The above algorithm is easily extended to nearest neighbor graphs by mapping each pixel in the image to a point in the feature space (e.g. (x, y, r, g, b) or (x, y, z, i)) and by using the Euclidean distances L_2 as edge weights. Each point should be connected to a limited number of neighbors for keeping the run-time low, e.g. using nearest k neighbors or the neighbors in a radius of r. Also, to further refine the quality of the segmentation, one takes into consideration both the reflectance of the point and the distance from the origin. Following the method, the segmentation algorithm is run twice (once for each image), and then the two resulting images are combined into one third segmentation, where two points are in the same region only if they are in the same regions for both the original segmentations.[1] Another approach creates and uses a (possibly weighted) metric involving both the reflectance and the range – for example a weighted Euclidean distance.

3. EXTENSION TO 3D POINT CLOUDS

A 3D laser scanner records the coordinates (x, y, z) and, optionally, the intensity (reflectance) of every point in the cloud. The points are converted to a feature space that uses spherical coordinates, producing intensity images (φ, θ, i) or range images (φ, θ, r).

To fully use the 3D information stored in point clouds, we create the graphs processed by the segmentation algorithm: each point represents a graph vertex, and for the edges we connect each pixel to its k nearest neighbors. Each edge is weighted by the distance between points and by difference in reflectance.

$$w((a, b)) = \|\vec{a} - \vec{b}\| + |i(a) - i(b)| \tag{6}$$

We use the approximate k nearest neighbors (aKNN) library.[9] It provides methods for computing the k nearest neighbors of a vertex, searching all the points in a given radius from a vertex, but also the approximate variants of these algorithms. Both the exact and the approximate searches are as appropriate for this case, in which each point must be connected to some other points with which it shares characteristics or not. Edges in the former

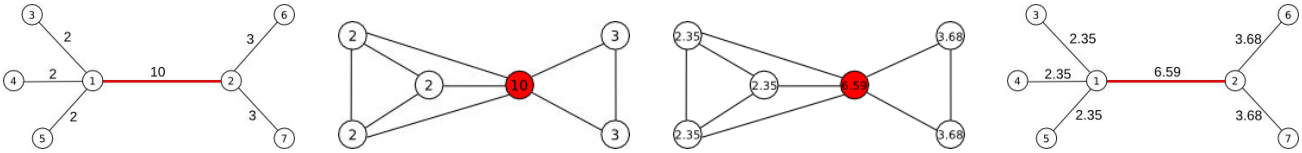


Figure 1. The four steps of applying Gaussian smoothing. (1) a possible initial graph; the edge of interest is in red. (2) the "dual" graph, obtained by converting each edge to a vertex, and by connecting each edge to all the other edges with which it shares a vertex; here, the vertices are assigned the weights of the original edges, with the vertex corresponding to the edge of interest in red. (3) the result of applying Gaussian smoothing with $\sigma = 4$ to the values of the vertices; only the direct neighbors of each vertex have an influence. (4) the reconstructed graph; in red, the original edge of interest has a "smoothed" value.

category reinforce segment cohesion, while edges in the latter category augment the inter-component distinction. The user provides the number of neighbors k and, optionally, the radius r as parameters: in the former case, the k nearest neighbors, and in the latter, at most k random neighbors which are at a distance smaller than r are used. We must emphasize that the former method is efficient for dense images, while the latter is appropriate for sparser 3D scans.

The original implementation of the Felzenszwalb-Huttenlocher uses Gaussian smoothing of the input image to smooth the weights of the graph edges. In our case, the most suitable approach is to directly apply Gaussian smoothing to the edges resulting after the aKNN searches. Each edge is smoothed using the Algorithm 2. The following notations are used: $G(x; \mu, \sigma)$ is the Gaussian function of mean μ and standard deviation σ applied to value x, η is the normalization factor and w'_e is the resulting weight. Furthermore, the algorithm is graphically explained in Figure 1.

Algorithm 2: Gaussian smoothing of an edge

Input: edge $e = (a, b)$ of weight w_e, σ for Gaussian smoothing
Output: smoothed weight w'_e

$\mathcal{L} = \{\}$;
forall *edges* $e' = (a', b')$ *with weight* $w_{e'}$ **do**
  if $a' \in \{a, b\}$ *or* $b' \in \{a, b\}$ **then**
    $\mathcal{L} = \mathcal{L} \cup \{(w_{e'}, G(w_{e'}; \mu = w_e, \sigma))\}$;
$\eta = \sum_{i \in \mathcal{L}, i=(w,\alpha)} \alpha$;
$w'_e = \frac{1}{\eta} \sum_{i \in \mathcal{L}, i=(w,\alpha)} w \cdot \alpha$;

After the Gaussian smoothing of the graph weights, the segmentation algorithm can proceed without any modification. However, the necessary final step is to iterate through all the resulting segments and merge those which are smaller than the minimum component size. After this post-processing step, the point clouds is grouped according to the result and then further used, for example for object identification.

4. RESULTS

The first analyzed data set is the medium-density scan, i.e., 3600 scan lines each with 1000 3D points, acquired by a Riegl VZ-400 scanner in an indoor lab environment, cf. Fig. 2: The walls are clearly distinct from the floor, the squares on the checkerboard are observable, and the human silhouettes are nearly always separated from the floor – results we did not obtain by applying the Felzenszwalb-Huttenlocher segmentation to range/intensity image pairs from the scanner. Fig. 2 presents the scan as images just for visualization purposes. Table 1 presents performance evaluation with respect to run times for a typical segmentation. Computing nearest neighbors is the most expensive step in the procedure. Recently, a lot of research has been performed to obtain speed-ups for this crucial step.[10, 11]

Figure 2. Spherical reflectance image (left) and its segmentation (right), using $\sigma = 0.2$, $K = 50$, $I = 30$, and 8 nearest neighbors for every point.

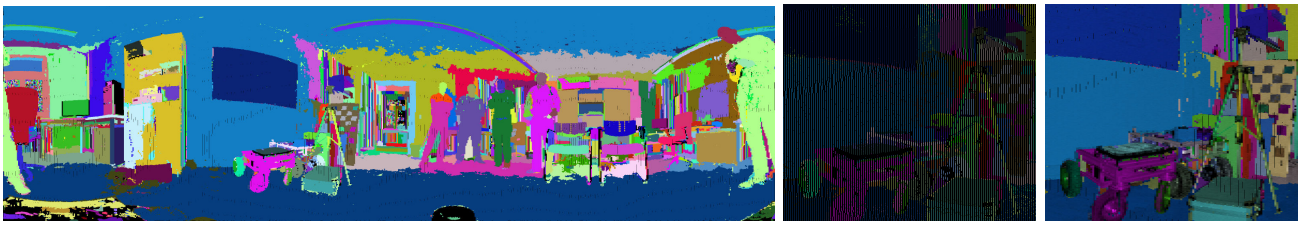


Figure 3. Segmented spherical image of the coarse data set (left), with a coarse (middle) and diluted detail (right). Parameters used: $\sigma = 1$, $K = 10$, $I = 40$.

A further lab data set has been acquired using a coarse resolution that requires the usage of radius search, instead of the k-nearest neighbors search. We randomly pick 4 neighbors within the radius. The result of this strategy is a homogeneous segmentation, with correctly segmented walls and silhouettes. The sparsity of the data set and the effect of dilated points can be observed respectively in Figure 3.

Several outdoor scans were made on the campus of Jacobs University. These data sets contain various challenges, such as occlusions, i.e., trees, or the high number of points. Figure 4 contains a scan made in front of a campus building: It can be observed that the cars are segmented, in a case even the number plate is in its own segment. Also, the background trees are segmented in their components, while the foreground trees, i.e., the closest tree has each branch segmented in a different component; this behavior can be assigned to the lower variance (in both distance and intensity) of the points corresponding to background trees.

Figure 5 shows two further results obtained on outdoor data sets. The left subfigure shows a correctly segmented car. In the right part of the figure, a tree is partly occluding a building in a scan taken from a point placed to the lower-left relatively to the viewpoint. It can be observed that some occlusions are not big enough to separate segments. For example the dark green segment in the upper-center part. A radius search was used to obtain such a result, which suggests that a higher radius could make the entire roof of the building belong to be not so over-segmented.

5. CONCLUSIONS AND FUTURE WORK

This work proved that the Felzenszwalb-Huttenlocher segmentation algorithm is suitable for 3D point clouds. To obtain good results, an extension of the Gaussian smoothing to graph edges was designed. Possible applications of this work include object recognition and image registration.

Our future research will focus on the edge weighting function. Currently, the function is defined as in Eq. (6). An improved version of this function uses normalized distance and intensity, i.e., the distance is divided by the average distance, and the intensity interval is converted to $[0, 1]$.

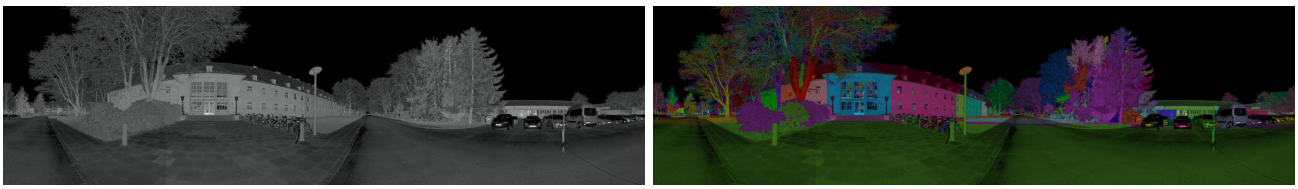


Figure 4. Spherical intensity image of an outdoor data set (left) and its segmentation (right). Parameters used: $\sigma = 2$, $K = 300$, $I = 100$.

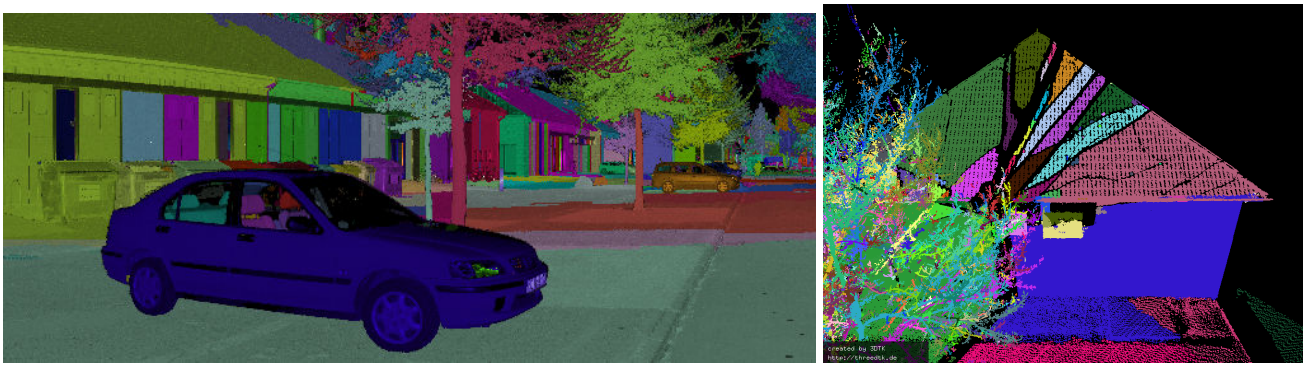
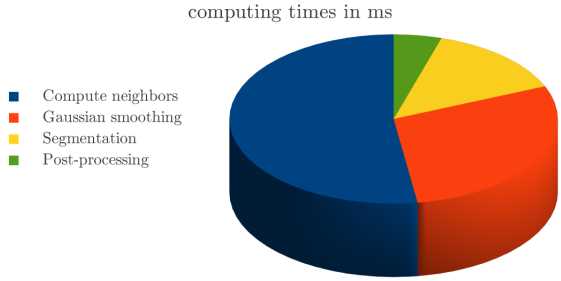


Figure 5. Results of the segmentation algorithm. Segmented car (left) and building (right). The right image is a perspective projection of a part of the 3D point cloud. Parameters used: $\sigma = 2$, $K = 300$, $I = 100$.

Table 1. Run times in milliseconds for segmenting an indoor environment.

sub routine	run time
Compute neighbors	42725
Gaussian smoothing	23672
Segmentation	11389
Post-processing	3849



REFERENCES

[1] Felzenszwalb, P. and Huttenlocher, D., "Efficient graph–based image segmentation," *International Journal of Computer Vision* **59** (September 2004).

[2] Peterson, M. A. and Rhodes, G., "Perception of faces, objects, and scenes : analytic and holistic processes," (2003).

[3] Hoover, A., Jean-Baptiste, G., Jiang, X., Flynn, P. J., Bunke, H., Goldgof, D., Bowyer, K., Eggert, D., Fitzgibbon, A., and Fisher, R., "An experimental comparison of range image segmentation algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* , 1–17 (July 1996).

[4] Pauling, F., Bosse, M., and Zlot, R., "Automatic Segmentation of 3D Laser Point Clouds by Ellipsoidal Region Growing," in [*Proceedings of the Australasian Conference on Robotics and Automation (ACRA), December 2-4, 2009, Sydney, Australia*], (December 2009).

[5] Oehler, B., Stueckler, J., Welle, J., Schulz, D., and Behnke, S., "Efficient Multi-Resolution Plane Segmentation of 3D Point Clouds," in [*Proceedings of the 4th International Conference on Intelligent Robotics and Applications (ICIRA)*], (2011).

[6] Douillard, B., Underwood, J., Kuntz, N., Vlaskine, V., Quadros, A., Morton, P., and Frenkel, A., "On the Segmentation of 3D LIDAR Point Clouds," in [*Proceedings of the IEEE International Conference on Robotics and Automation ('11)*], 2798–2805 (May 2011).

[7] Otsu, N., "A threshold selection method for gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics* (1979).

[8] Shi, J. and Malik, J., "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2000).

[9] "Approximate k-nearest neighbors." `http://www.cs.umd.edu/~mount/ANN/` (May 2012).

[10] Elseberg, J., Borrmann, D., and Nüchter, A., "Efficient Processing of Large 3D Point Clouds," in [*Proceedings of the XXIII International Symposium on Information, Communication and Automation Technologies (ICAT '11)*], IEEE Xplore, Sarajevo, Bosnia (October 2011).

[11] Garcia, V., Debreuve, E., and Barlaud, M., "Fast k nearest neighbor search using gpu," in [*IEEE CVPR Workshop on Computer Vision on GPU*], (June 2008).