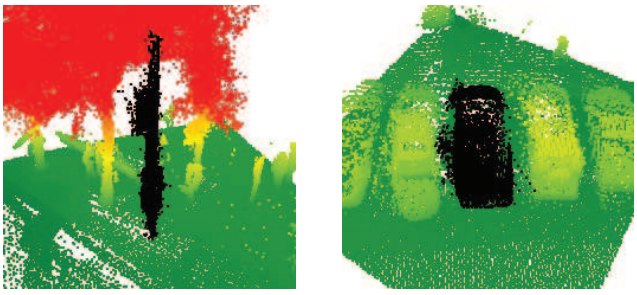
Min-Cut Based Segmentation of Point Clouds

Aleksey Golovinskiy
Princeton University

Thomas Funkhouser
Princeton University



Traffic Light Car

Figure 1. Example segmentations. Our method is able to extract foreground points from background clutter. (For easier visualization, points are drawn with colors representing their heights)

Abstract

We present a min-cut based method of segmenting objects in point clouds. Given an object location, our method builds a k-nearest neighbors graph, assumes a background prior, adds hard foreground (and optionally background) constraints, and finds the min-cut to compute a foreground-background segmentation. Our method can be run fully automatically, or interactively with a user interface. We test our system on an outdoor urban scan, quantitatively evaluate our algorithm on a test set of about 1000 objects, and compare to several alternative approaches.

1. Introduction

As 3D scanning technologies advance, the promise of ubiquitous 3D data is fast becoming reality. In particular, 3D point clouds of entire cities are becoming available. This explosion of data fuels a need for algorithms that process point clouds. The segmentation of point clouds into foreground and background is a fundamental problem in processing point clouds. Specifically, given an estimate for the location of an object, the objective is to identify those points that belong to the object, and separate them from the background points. Besides the essential task of separating foreground from background, segmentation can be helpful for localization, classification, and feature extraction. In this paper, we describe and evaluate a min-cut based segmentation algorithm that was summarized in [6] as a part of a system to detect objects in outdoor urban scans.

The problem of segmenting objects in 3D point clouds is challenging. The foreground is often highly entangled with the background. The real-world data is noisy. Sampling is uneven: ground-based scans have point densities that dominate from the direction the scan is taken, and airborne scans have poor sampling for nearly vertical surfaces. In addition, data sets such as the one studied in this paper consist of point clouds aggregated from both land and airborne scans, leading to considerable discrepancies in sampling rates between different objects and often different surfaces of the same objects. Finally, non-reflective surfaces such as windows are missing. Examples of results of our method overcoming some of these difficulties are shown in Figure 1

Since large-scale outdoor point cloud scans are an emerging source of data, there is not much work describing segmentations of such scans. What work exists mostly focuses on the extraction of geometric primitives or parts([13, 18]) rather than entire objects. We adapt the techniques of computer vision ([1]) and computer graphics (e.g. [9]), where graph-cut based methods have been used to, respectively, separate foreground and background in images, and decompose 3D surfaces into parts. We extend such methods to 3D point clouds. Unlike images, we cannot use colors or textures as cues, and unlike most computer graphics (and CAD) segmentation problems, the input is a noisy point cloud representing a scene, rather than a clean surface model of an individual object.

We propose a min-cut based segmentation method. Our method works by creating a nearest neighbors graph on the point cloud, defining a penalty function that encourages a smooth segmentation where the foreground is weakly connected to the background, and minimizing that function with a min-cut. The method was summarized as part of a system of object detection for urban outdoor scenes in [6]; in this paper, we expand on that summary with a more detailed description of the algorithm and discussion of the design choices, examples, and an in-depth evaluation.

2. Previous Work

We summarize previous work in three related areas: segmentation of point clouds, part decomposition of 3D objects, and segmentation of images.

Point Cloud Segmentation. Some work has been done on segmenting point clouds. In some scenarios, such as [3], the input is a point cloud representing a single object, and the goal is to decompose the object into patches. The algorithms proceed by either reconstructing a mesh and then segmenting it, or by segmenting the point cloud directly. While some work has been done on segmentation of point clouds in scenes, the emphasis is usually on extracting geometric primitives (such as in [13] and [18]) using cues

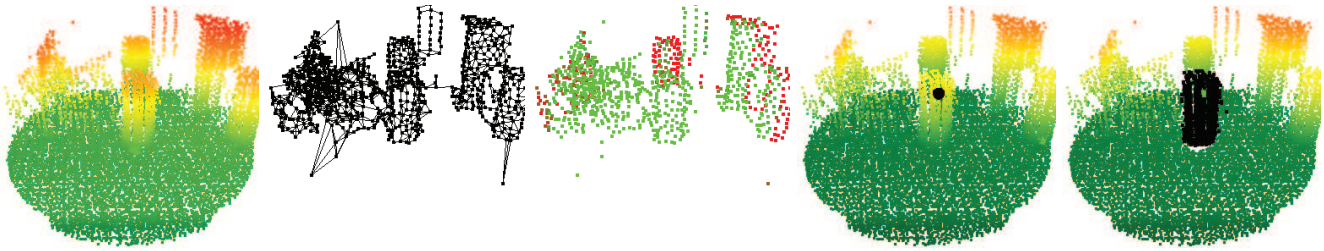


Figure 2. Overview of our system. (a) The system takes as input a point cloud near an object location (in this case, a short post). (b) A k-nearest neighbors graph is constructed. (c) Each node has a background penalty function, increasing from the input location to the background radius (visualized with color turning from green to red as the value increases). (d) In the automatic version of our algorithm, a foreground point is chosen as a hard constraint (in the interactive mode, the user chooses hard foreground and background constraints). The resulting segmentation is created via a min-cut (e).

such as normals and curvature. In outdoor scans, effective estimation of curvatures is difficult for many objects, because the data is noisy and encodes complex objects, so the segmentation cues we use do not rely on curvature estimation. In general, the problem of extracting foreground object points from background in outdoor scenes is relatively new, and is driven by the recent availability of outdoor scans.

Part Decomposition of 3D Objects. Much work has been done in computer graphics and CAD on segmentation of 3D models of single objects, usually represented by meshes (a survey can be found in [14]). A wide variety of algorithms have been proposed for this problem, including ones based on convex decomposition [2], watershed analysis [12], K-means [17], hierarchical clustering [4, 5], region growing [19], and spectral clustering [11]. The objective is to decompose an object into functionally meaningful parts or regions. These methods typically construct a graph from the input mesh, and cluster the graph to produce a segmentation by using cues such as concavity along boundaries and part compactness, often using graph cut techniques, as in [9]. Some methods (such as [15]) create a scalar function over the surface of the mesh, and then decompose the mesh along discontinuities in that function using min-cut based methods. [7] present an interactive method where the user paints on different segments of the mesh, and a region-growing method is used to partition the mesh with these constraints–this is similar to our interactive system. Our goal differs in that rather than decompose an object into parts, we aim to separate the foreground object from the background points. Also, we do not have fine-scale geometrical cues such as curvature, but rather rely on distances between points and point densities.

Image Segmentation. Segmentation of images is a classic problem in computer vision. There are approaches that seek to decompose an image into regions (such as Normalized Cuts [16]). There is also a great deal of work on foreground/background segmentation. Some methods are data-driven and use examples of object instances from a class to segment a new object [10]. More relevant to us are methods that perform foreground segmentation in a non-data-driven manner. One such method is Snakes [8], in which an initial boundary is specified either by the user or automatically, and is then refined to minimize an error functional. The most relevant work to ours is [1], in which the authors take advantage of the min-cut algorithm to create a segmentation that satisfies constraints placed by the user or automatic cues. As the authors note, the min-cut algorithm is particularly adept at such segmentations, since it solves a global minimization in low-order polynomial time, and the terms of the minimization can include neighborhood smoothing constraints as well as hard or soft foreground/background constraints. We adapt this framework to the domain of 3D point clouds, where the main cues are distances and point densities, rather than colors and textures.

3. Overview

Given a suspected location of an object, we need an algorithm that returns the foreground points that belong to the object. In particular, we are interested in objects found in cities that range in size from fire hydrants to traffic lights. Desirable properties of a segmentation algorithm include:

- **Correctness:** it would be best to get the foreground as accurately as possible (with respect to precision/recall)
- **Input parameters:** since over- and under-segmentation is inevitable, it is helpful for an algorithm to accept additional intuitive parameters describing a-priori assumptions about the object (for example, the approximate horizontal radius to the background). This was used in the system of [6], for example, to request multiple segmentations of an object for more robust feature extraction.
- **Speed:** a segmentation algorithm is likely to be executed for many locations in a large point cloud, so running time is important.

The intuition behind our algorithm is that a good foreground segmentation consists of points that are well-connected to each other, but poorly connected to the background. The overview of our method is shown in Figure 2. Given an input scene (Figure 2a), we build a nearest-neighbor graph (Figure 2b) to encourage neighboring points

to be assigned the same label. Then, given as input an expected horizontal distance to the background, we create a background penalty (Figure 2c) that encourages more distant points to be in the background. In an automatic or interactive manner, we add hard constraints for foreground and, optionally, background points (Figure 2d). Our algorithm returns the segmentation generated by the min-cut, which (i) minimizes the cut cost of the nearest neighbor graph, (ii) minimizes the background penalty, and (iii) adheres to the hard foreground or background constraints (Figure 2e).

Section 4 describes the construction of the graph and background penalty. Then, section 5 describes the addition of hard constraints in the fully automatic regime of our algorithm (including a method of automatically choosing the background radius), the addition of hard constraints in the interactive regime, and accelerations for greater efficiency.

4. Basic Setup

The input to the segmentation algorithm is (i) a 2d location and (ii) a suspected background radius (horizontal radius at which we assume the background begins). We estimate the ground plane with iterative plane fitting, and remove points close to the ground (within .2m). We then construct a graph between neighboring points, which ensures smoothness of segmentation, and a soft background penalty, which encourages points far from the object location to be labeled as background.

4.1. Graph

We create a graph representing the structure of the point cloud, where closer points are more strongly connected. Specifically, we construct a k-nearest neighbors graph on the input points (we use k = 4). The edges of this graph have weights that decrease with distance, so that if edge i connects points at a distance d_i, its weight is $w_i = exp(-(d_i/\sigma)^2)$, where we use σ = .1m (a common spacing of points in our data). Because the k-nearest neighbors graph often results in several disconnected components within each foreground object, we connect the closest point pairs of disconnected components.

Note that the cut cost of a potential segmentation of this graph takes into account both distances between points in a foreground/background cut, and the density of points on the boundary via the number of broken link edges. This makes the min-cut algorithm more robust to spurious connections between foreground and background. Note also that the construction of the graph makes it adaptive to the point cloud resolution, without requiring a pre-defined threshold. When the min-cut is computed, this graph ensures that the segmentation is smooth (neighboring points are more likely to be assigned to the same segment) and that a larger separation between foreground and background is encouraged.

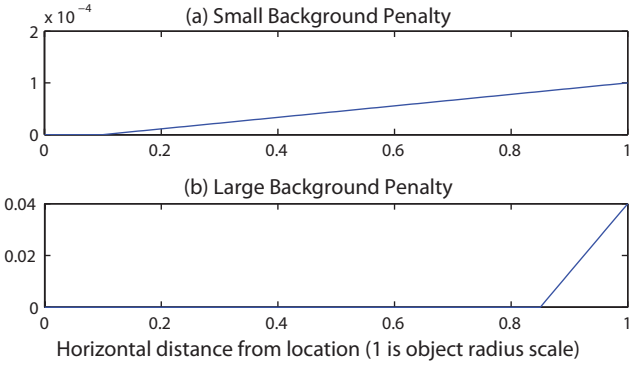


Figure 3. Background penalty, as a function of horizontal radius from the input location (normalized so that 1 is the input background radius). The penalty consists of two components: (a) a penalty starting at a small radius but rising slowly that encourages points disconnected from foreground to be in the background, and (b) a steep penalty close to the background radius that mandates that points outside of the background radius be in the background.

4.2. Background Penalty

Given a background radius, we create a soft background penalty whose goal is two-fold: to strongly encourage points at the background radius to be labeled as background, and to encourage components only loosely connected to the object location to be in the background. We create a pointwise background penalty $B(p)$ that is added to the total cost of the cut for every point p chosen to be in the foreground. This is done by introducing a virtual background node that connects to every real node with edges of cost $B(p)$.

This background penalty $B(p)$ can be set to reflect any background prior. In our system, we make $B(p)$ a function of the horizontal distance r to the object location (relative to the background radius). The penalty consists of two linear components (Figure 3). The first component begins at a small distance, and increases relatively slowly (Figure 3a). This ensures that components that are disconnected or weakly connected to the foreground are encouraged to be in the background. The second component begins at a large distance, closer to the background radius, and increases rapidly (Figure 3b), ensuring that points near the background radius are labeled as background.

5. Performing Segmentation

The previous section described the how the graph is set up to encourage two properties with soft constraints: a smoothness error that encourages nearby points to have the same label, and a background penalty that encourages points close to the background radius to be in the background. It remains to specify constraints that encourage points to be in the foreground. Our algorithm can be run in two regimes: automatically, and interactively, both adding hard constraints. In both cases, the final segmentation is found with a min-cut. Below, we describe the automatic regime (including automatically choosing the background

radius), the interactive regime, and some accelerations.

5.1. Automatic Regime

In the automatic version of our algorithm, some assumption for the foreground needs to be made. In our system, we include as a hard constraint the point closest to the (horizontal) object location at a predefined height (we use 1.2m) and its M closest neighbors in the foreground (we use $M = 3$).

The algorithm given so far produces an automatic segmentation of objects with a radial scale given as input. Some applications (such as extracting features from multiple segmentations, as in [6]) benefit from this parameter, but for other applications, a fully automatic selection of this radial scale is useful. To select the background radius (which ranges from 1m to 5m for our objects of interest), we run the min-cut algorithm for several iterations to automatically determine the best background radius for the segmentation. Starting from the smallest radius in the range, we run the above algorithm, and increase the radius to the maximum of the range (by 1m increments) until (i) the number of foreground points exceeds a threshold (we use 35) segmentation and (ii) the resulting cut is below a threshold (we use .4).

Two examples of choosing the background radius automatically are shown in in Figure 4. For these objects (a car and a sign), segmentations are shown for $R = 2m$ and $R = 4m$, and the chosen radius is outlined. These examples illustrate that it is difficult to choose a static background radius that works for a range of objects, but that it is possible to automatically determine (even without additional prior information) the appropriate radius for a particular object.

5.2. Interactive Regime

While no automatic algorithm will be completely successful, in some scenarios it may be practical to use an interactive segmentation tool. Such a tool should follow the user-constraints at interactive rates, while automatically making a reasonable guess in unconstrained regions, and allowing any segmentation to be reached with sufficiently many constraints. Similar to the ideas of [1] our min-cut algorithm is easily set up for such a tool.

The interactive algorithm starts with the graph and background weights given in previous section. Instead of assuming a foreground constraint, as in the automatic algorithm, we allow the user to iteratively add (and remove) points as hard background or foreground constraints. The segmentation is re-calculated as the min-cut under these constraints.

The interactive tool is shown in Figure 5. To create a segmentation, the user looks at the input scene (Figure 5a), and selects a radius that includes the object to segment (Figure 5b). Note that for an object such as the shown newspaper box, automatic segmentation is very difficult since the box is connected to adjacent newspaper boxes. To perform manual segmentation, the user adds foreground and background constraints as necessary, responding to the interactively generated segmentation until the result is satisfactory

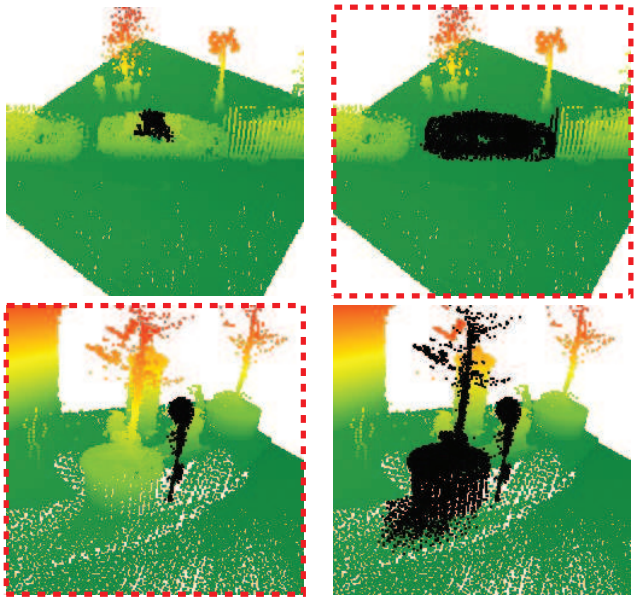


Figure 4. A constant choice of background radius cannot segment both examples above (R = 2m and R = 4m are shown). To automatically choose the background radius, we run the min-cut algorithm for several choices, and select the smallest radius that results in a segmentation with a small enough cut cost and large enough foreground size, choosing correctly the segmentations outlined by dotted squares in the above examples.

(Figure 5c, d). Note that the nearest neighbors graph ensures that each successive segmentation is a smooth extrapolation of the constraints.

5.3. Accelerations

Segmentation code is likely to be used at least once for each object of interest. A scan of even a moderately sized city or town will have tens of thousands (if not more) of objects of interest, so it is important for the algorithm to run quickly. A typical object will have from 10 to 200 thousand points in its radial support, so the basic algorithm described above will run slowly (the $O(n^3)$ cost of the min-cut algorithm is the initial bottleneck). We describe several accelerations to the basic algorithm that reduce the running time from about 10s to about .1s per object.

To reduce the number of nodes on which the min-cut is performed, we contract the graph by hierarchically merging nodes. A number of clustering errors may be used; we order the nodes to merge by the distance between their centroids, and merge nodes until the graph reaches a pre-set size (we use 1000 nodes). Note that arc weights are added to form outgoing arcs from a newly merged node, so the min-cut is altered only if two nodes on different sides of the correct min-cut are merged during this stage.

After introducing the above contraction, finding K-nearest neighbors (using a KD-tree) and the contraction itself become the bottleneck. To reduce the number of nodes that serve as input to finding nearest neighbors and contrac-

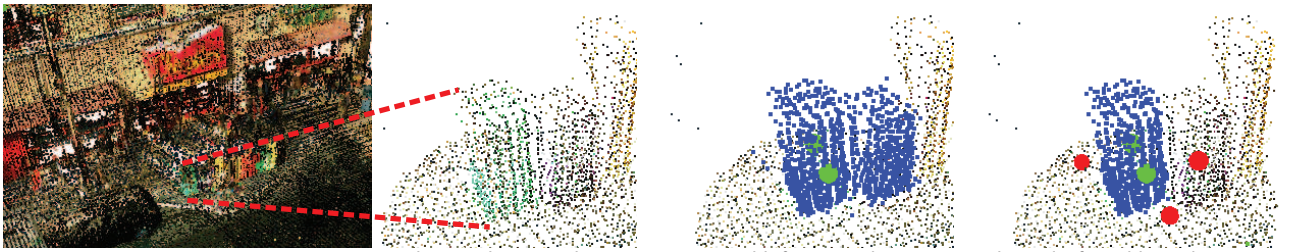


Figure 5. The user surveys the input scene (a), and chooses a radius that includes the object to segment (b). The user creates several foreground constraints (green circles), and a segmentation is interactively performed, with foreground point shown in blue (c). If necessary, the user adds additional constraints (background constraints in red), until the segmentation is satisfactory (d). The user has the option of toggling between views of all points (as shown here), or only background or foreground points, to make sure the object is not over or under segmented.

tion, we add a pre-process that creates an axis-aligned grid and creates a graph node at each occupied grid cell, at the location of the centroid of points in that cell. We use a grid spacing of .2m. Note that this stage adjusts the cut costs of those grid cells that have multiple nodes.

6. Results

In this section, we first describe the data used for testing our prototype system. We then describe two alternative segmentation algorithms, and show some example results and comparisons. Finally, we perform a quantitative evaluation of our algorithm.

6.1. Data

We tested our segmentation algorithm on the LIDAR scan and dataset described in [6], which covers about 6 square kilometers of Ottawa, Canada. The truthed part of that scan covers about 300,000 square meters with about 100 million points, and contains about 1000 objects of interest placed by BAE Systems. These object of interest form the basis of our quantitative evaluation.

The scans were collected by Neptec. They were collected from four car-mounted TITAN scanners facing left, right, forward-up, and forward-down, and from an airborne scanner. The scans were merged and provided to us as a single point cloud, with a position, intensity, and color per point. The colors from car-mounted scanners were not very accurate, so we focused on geometry as the only cue for segmentations in this paper. The reported alignment error between ground-based and air-based scans is 0.05 meters, with a reported vertical accuracy is 0.04 meters.

6.2. Alternative Algorithms

In this section we describe two simple alternative to perform segmentation of point clouds.

All Points. A simple way to extract the foreground points is to assume that all points within some distance of the input location are foreground. The algorithm is then to return all (above-ground) points within a preset horizontal radius.

While this is a trivial technique, it works well for isolated objects, and it is consistent. It is also effective for feature extraction for applications that do not require the points in the object to be explicitly identified. Extracting all points allows the maximum radius of an object as input, which is intuitive. Of course, this method is limited, and fails when there are background points within the specified radius.

Connected Components. Noting that foreground objects often consist of a dense group of points far from the background, a slightly more sophisticated way to perform segmentation is to assume that the foreground consists of a connected component. That is, the algorithm starts with a seed point assumed to be in the object (in our case, the closest point to the input 2d location at a preset height of 1.2m), and labels as foreground the set of points connected to this seed point via distances smaller than some threshold length.

This method works for simple cases, when objects are relatively dense and well-separated from the background. It is a version of the clustering algorithm known as single-link clustering, which suffers from two well-known problems. First, it is not easy to choose a global distance threshold that works for a large range of objects. Second, since this is a greedy algorithm, it is not robust to noise or points that are found between the foreground and background, and for many objects, there does not exist a distance threshold that captures all of the foreground in the connected component without capturing any of the background. Because a choice of distance threshold may result in an unreasonably large connected component, a practical implementation of this method requires two input parameters: the connected component distance threshold, and a cut-off horizontal radius beyond which points are assumed to be in the background.

6.3. Examples

In this section, we show several example segmentations created with our method as well as with alternatives. While a more complete, quantitative comparison is performed in the next section, these examples provide useful intuition.

Figure 6 contains segmentations of several objects, including a car, several lamp posts, a sign, and a trash can. The first column has the ground truth segmentation created with our interactive tool. The next column has the all

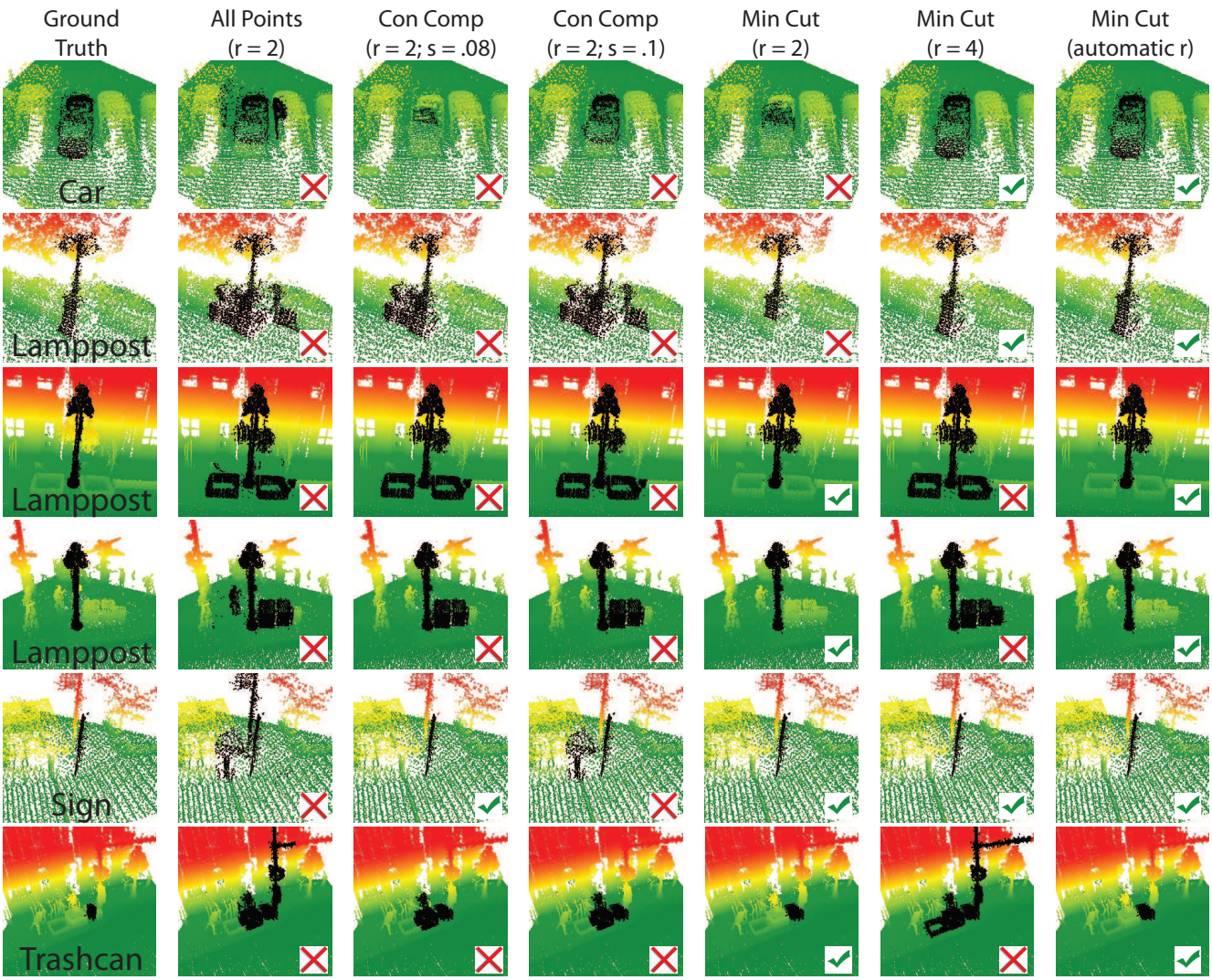


Figure 6. Example segmentations. Each row has an object with ground truth segmentation, followed by an all-points segmentation, two connected component segmentations with different spacings, two min-cut segmentations with different static background radii, and a min-cut segmentation with automatically chosen background radius. While connected components and min-cut with static background radii are sometimes successful, the min-cut segmentation with automatic background radius is more robust to clutter and varying object sizes.

points segmentation, with a radius $r = 2$. The next two columns have connected components segmentations, with radius $r = 2$ and spacing $s = .08$ and .1. The next two columns have our min-cut segmentation, with a static background radius $r = 2$ and $r = 4$. The last column has results of our method with an automatic background radius.

Because none of the example objects are isolated, the all points segmentation returns many background points. The connected components segmentation is more successful: the result for the car example is close to correct with $s = .1$, and the sign example is correct for $s = .08$. However, the spacing is difficult to adjust: both connected component segmentations under-segment the car while over-segmenting most of the other examples.

The min-cut segmentations with a static background have better results. For the sign example, both radii return the correct segmentation, and the first lamp post example is close to correct with both radii. For each of the example objects, one of the two settings of r returns the correct segmentation with the min-cut. But, for most examples, the wrong choice of r returns a drastic over- or under-segmentation. The min-cut method with automatic radius, on the other hand, is able to choose the correct background radius for these examples, returning correct segmentations.

Of course, no automatic segmentation algorithm is perfect. Some example failure cases of our automatic algorithm are shown in Figure 7. Our algorithm can fail in several scenarios. In (a), many noise points lie between a control box (on the right) and a light standard (on the left). The resulting high cut cost of the correct segmentation makes it difficult for our algorithm to automatically choose the correct (smaller) background radius, so both ob-

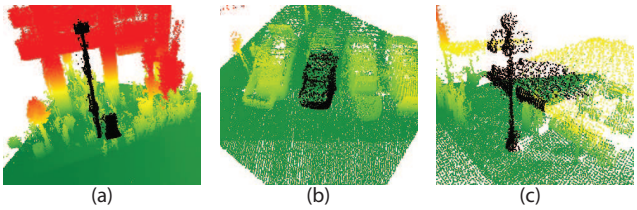


Figure 7. Example failures of the automatic segmentation algorithm. In (a) the control box (on the right) cannot be separated from a close light standard (on the left). In (b), only a part of a car is returned. In (c), a lamp post is not separated from a close roof.

jects are returned. In (b), a car has a very weakly connected component, which is returned. The small cost of the cut of this component (relative to the correct segmentation of the entire car, which has noisy connections to the adjacent car), again makes it difficult to choose the correct background radius. In (c), a lamp post is very strongly connected to an adjacent roof. While an appropriate choice of background may return a better segmentation, more useful cues for this case include normal continuity and concavity.

6.4. Evaluation

Using the ground truth segmentations we created with our interactive segmentation tool, we are able to quantitatively evaluate the performance of our segmentation algorithm, and compare to the alternatives.

We gather statistics as follows. For each object of interest, we run a segmentation algorithm, and record its precision (ratio of correctly predicted foreground points to the total number of predicted foreground points) and recall (ratio of correctly predicted foreground points to the number of ground truth foreground points). A high precision indicates that most of the predicted foreground points are in the object, and a high recall indicates that most of the object points have been predicted to be in the foreground.

Table 1 contains the results, averaged first by object class and then overall, for the segmentation algorithms shown in Figure 6: all points, two settings of connected components, two settings of min-cut with static radius, and min-cut with an automatically chosen background radius. Some objects are easier to segment: parking meters are often isolated, so both connected component and min-cut algorithms perform well. Other objects, such as trash cans, are often close to background clutter, so the precision is lower. Min-cut algorithms are able to raise both precision and recall for trash cans relative to connected components. Likewise, min-cut algorithms improve performance significantly for cars and signs. Other objects, such as newspaper boxes (an example of one is shown in Figure 5) are very close to each other, so while our algorithm improves the precision, it remains low.

Overall, as expected, the all points algorithm has a relatively high recall at the cost of low precision. The two connected component algorithms have a higher precision, and the min-cut algorithms improve on this performance.

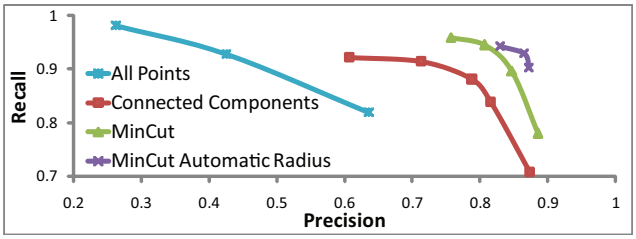


Figure 8. Precision-recall plots of our algorithm compared to several alternatives. The all points algorithm is shown in blue at varying radii. The connected components algorithm is shown in red with varying spacing. The min-cut algorithm is shown in green with varying statically chosen background radii. Finally, the min-cut algorithm with automatically chosen background radius is shown in purple with varying cut cost thresholds. The performance improves in the order of the algorithms presented.

The min-cut algorithm with automatic radius has better performance that the two shown settings of the static radius version. This last point is more apparent in Figure 8 (replicated from [6]), which shows the precision-recall curves resulting from running the above segmentation algorithms at several settings. Specifically, it shows the all-points algorithm at varying radii (blue), connected components at varying spacing (red), min-cut with varying static background (green), and min-cut with automatically chosen radius at varying thresholds (purple). Comparing the last two curves shows the improvement in performance made by automatically choosing the background radius.

7. Conclusion

In this paper, we presented a graph-cut based method for segmenting objects in point clouds. We showed how our method can be adapted for both automatic and interactive segmentation. We used the interactive version of our algorithm to generate a truth set of about 1000 segmentations, and used this truth set to quantitatively evaluate the automatic algorithm, comparing to two alternatives.

There are two immediate directions for future work. First, we can augment our algorithm with more cues to make it more effective. One can fit geometric primitives, such as planes and cylinders to the data, and augment our algorithm with the observation that since many urban objects are man-made, points belonging to the same primitive are likely to belong to the same object. Similarly, one can add cues such as convexity (object parts are likely to be convex) and symmetry (many objects exhibit strong symmetry, so segmentations ought to be symmetric as well).

Second, our segmentation algorithm is likely to be a step in recognition system, as shown in [6]. While our segmentations can inform recognition, feedback is likely to be useful – once an object type has been proposed, our algorithm is simple to adjust to account for the prior shape of this type, and a more accurate segmentation can be generated.

Class	# in Truth Area	All Points (r = 2) Pr	All Points (r = 2) Re	Con Comp (r = 2; s = .08) Pr	Con Comp (r = 2; s = .08) Re	Con Comp (r = 2; s = .1) Pr	Con Comp (r = 2; s = .1) Re	Min-Cut (r = 2m) Pr	Min-Cut (r = 2m) Re	Min-Cut (r = 4m) Pr	Min-Cut (r = 4m) Re	Min-Cut (auto r) Pr	Min-Cut (auto r) Re
Short Post	338	13	99	89	99	86	99	93	98	82	99	92	99
Car	238	77	75	93	47	91	59	93	20	92	82	92	77
Lamp Post	146	60	99	82	95	79	97	89	96	86	99	89	98
Sign	96	36	100	73	74	68	97	84	98	73	100	83	100
Light Standard	58	68	93	84	92	83	92	92	86	91	92	91	92
Traffic Light	42	58	75	75	75	72	75	92	72	84	87	84	86
Newspaper Box	37	13	100	15	96	14	100	40	86	21	100	38	93
Tall Post	34	35	100	42	89	42	96	79	84	46	100	58	96
Fire Hydrant	20	36	100	81	89	81	95	89	100	82	100	88	100
Trash Can	19	17	100	48	93	43	94	57	100	54	100	60	100
Parking Meters	10	14	100	100	98	100	99	100	100	100	100	100	100
Traffic Control Box	7	19	100	82	96	79	99	79	100	68	100	80	100
Recycle Bins	7	46	100	71	94	64	99	92	99	80	100	92	100
Advertising Cylinder	6	70	100	79	83	79	83	97	100	89	100	96	100
Mailing Box	3	48	100	86	100	86	100	98	100	98	100	98	100
“A” - frame	2	59	100	70	50	69	100	87	100	69	100	86	100
All	1063	43	93	82	84	79	88	89	78	81	95	86	93

Table 1. Per-class precision/recall results of the segmentation algorithms.

8. Acknowledgments

We thank Neptec, John Gilmore, and Wright State University for providing the 3D LIDAR data set. This work started as part of the DARPA's URGENT program, and we thank BAE Systems for including us in the project, especially Erik Sobel, Matt Antone, and Joel Douglas. Aleksey Boyko, Xiaobai Chen, Forrester Cole, Vladimir Kim, and Yaron Lipman provided valuable ideas, and Kristin and Kelly Hageman helped with ground truthing. Finally, we thank NSF (CNFS-0406415, IIS-0612231, and CCF-0702672) and Google for providing funding.

References

[1] Y. Boykov and G. Funka-Lea. Graph cuts and efficient n-d image segmentation. *IJCV*, 70(2):109–131, 2006.

[2] B. Chazelle, D. Dobkin, N. Shourhura, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry: Theory and Applications*, 7(4-5):327–342, 1997.

[3] J. Fransens and F. Van Reeth. Hierarchical pca decomposition of point clouds. In *Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 591–598, 2006.

[4] M. Garland, A. Willmott, and P. Heckbert. Hierarchical face clustering on polygonal surfaces. In *ACM Symposium on Interactive 3D Graphics*, pages 49–58, 2001.

[5] N. Gelfand and L. Guibas. Shape segmentation using local slippage analysis. In *Symposium on Geometry Processing*, pages 214–223, 2004.

[6] A. Golovinskiy, V. G. Kim, and T. Funkhouser. Shape-based recognition of 3d point clouds in urban environments. *ICCV*, September 2009.

[7] Z. Ji, L. Liuy, Z. Chen, and G. Wang. Easy mesh cutting. In *Eurographics*, volume 25, 2006.

[8] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, January 1988.

[9] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics (TOG)*, 22(3):954–961, 2003.

[10] B. Leibe, A. Leonardis, and B. Schiele. Combined object categorization and segmentation with an implicit shape model. In *In ECCV workshop on statistical learning in computer vision*, pages 17–32, 2004.

[11] R. Liu and H. Zhang. Segmentation of 3d meshes through spectral clustering. In *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*, 2004.

[12] A. Mangan and R. Whitaker. Partitioning 3D surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, 1999.

[13] T. Rabbani, F. van den Heuvel, and G. Vosselmann. Segmentation of point clouds using smoothness constraint. In *IEVM06*, 2006.

[14] A. Shamir. Segmentation and shape extraction of 3d boundary meshes (state-of-the-art report). In *Eurographics*, pages 137–149, 2006.

[15] L. Shapira, A. Shamir, and D. Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *Vis. Comput.*, 24(4):249–259, 2008.

[16] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE PAMI*, 22(8):888–905, 2000.

[17] S. Shlafman, A. Tal, and S. Katz. Metamorphosis of polyhedral surfaces using decomposition. In *Eurographics 2002*, pages 219–228, September 2002.

[18] R. Unnikrishnan and M. Hebert. Robust extraction of multiple structures from non-uniformly sampled data. In *IROS*, volume 2, pages 1322–29, October 2003.

[19] E. Zuckerberger, A. Tal, and S. Shlafman. Polyhedral surface decomposition with applications. *Computers & Graphics*, 26(5):733–743, 2002.