



# Interactive Segmentation of Textured Point Clouds

P. Schmitz , S. Suder, K. Schuster and L. Kobbelt 

Visual Computing Institute, RWTH Aachen University

---

## Abstract

*We present a method for the interactive segmentation of textured 3D point clouds. The problem is formulated as a minimum graph cut on a  $k$ -nearest neighbor graph and leverages the rich information contained in high-resolution photographs as the discriminative feature. We demonstrate that the achievable segmentation accuracy is significantly improved compared to using an average color per point as in prior work. The method is designed to work efficiently on large datasets and yields results at interactive rates. This way, an interactive workflow can be realized in an immersive virtual environment, which supports the segmentation task by improved depth perception and the use of tracked 3D input devices. Our method enables to create high-quality segmentations of textured point clouds fast and conveniently.*

## CCS Concepts

• **Computing methodologies** → *Point-based models; Image processing; Virtual reality;*

---

## 1. Introduction

Automatic 3D scene understanding is vital for an increasing number of important applications such as human-machine interaction, autonomous driving, drone navigation, geographical surveying and surveillance. For such applications, an accurate and reliable segmentation is a crucial building block to distinguish and identify different classes of objects.

Other types of applications capture and process high-fidelity reproductions of real-world scenes. Examples include the entertainment industry and diverse areas such as cultural heritage preservation, archeology, construction site monitoring or crime scene investigation. For many of these, an accurate segmentation of individual objects is essential for further processing and analysis, such as semantic labeling or mesh generation.

Strong fully automatic segmentation methods for 3D point clouds exist due to the increased availability of training data and advances in data-driven models. Yet, they are not always sufficiently reliable and can lack the necessary precision for downstream applications in geometry processing. To improve such approaches further, the availability of manually labeled training data is crucial. Precisely labeling 3D point cloud data is, however, a labor-intensive and expensive process. A critical component for further development of point cloud segmentation methods are thus interactive solutions that enable a human operator to generate precise segmentations fast and efficiently.

Interactivity is also the prerequisite to implement a user interface for the segmentation task in an immersive virtual environment. The ability to move freely, the added depth perception, and the use of

tracked 3D input devices naturally support the task of inspecting three-dimensional structures and directly interacting with them.

In this work, we present a user-assisted point cloud segmentation method based on minimal graph cuts. It is inspired by the work of Rother et al. and extends the *GrabCut* technique into the domain of textured 3D point clouds [RKB04]. Our method is designed to work efficiently on large datasets and yields results at interactive rates. The user can thus be kept “in the loop” and directly interact with the segmentation result to refine it.

Our method improves on previous approaches by leveraging the rich information in the photographs for textured splat rendering. We show that our method significantly improves the initial segmentation result compared to using an average color per splat. Due to our interactive workflow, the result can be refined fast and conveniently, such that high-quality segmentations are achieved with low effort.

## 2. Related Work

In the following, we review prior work in point-based rendering, graph cut-based image segmentation and user-assisted segmentation of 3D point clouds.

The use of points as a rendering primitive has been pioneered by Levoy and Whitted [LW85]. Pfister et al. proposed *surfels* as a rendering primitive that approximates the local surface with oriented discs in 3D space [PZVG00]. Botsch et al. introduced *Phong Splatting*, which uses a two-pass rendering approach to generate a smoothly varying normal field for high-quality lighting and shading [BSK04]. The method was subsequently extended to run on modern GPUs and to include textures for improved surface appearance [BHZK05]. Schmitz et al. proposed virtualized textures for

massively detailed datasets and 3D ellipsoids as a rendering primitive [SBMK20].

Minimal graph cuts are commonly used to segment 2D images, as first introduced by Boykov et al. [BJ01]. Starting from two sets of known foreground and background pixels, a graph is constructed that connects each pixel to a virtual source and sink node. The weights assigned to these edges are based on so-called “regional terms” that model the cost of assigning the pixel to the respective class given its color or intensity. A second set of edges connects each pixel with its direct neighbors. Their weight is based on the “boundary terms” that model the cost of labeling the two adjacent pixels differently. The boundary terms are based on a similarity measure between pixel values, such that labeling two very similar adjacent pixels differently incurs a high cost. The segmentation is then computed as the cut with minimum cost that separates the virtual source from the sink node.

A popular variant is the *GrabCut* technique by Rother et al. [RKB04]. They propose an interactive segmentation method, in which the user first annotates foreground and background pixels. A Gaussian Mixture Model (GMM) is estimated to model the probabilities of pixels to belong to either foreground or background. The algorithm then computes the minimal cut, and refines the GMM parameters based on the predicted labels to compute new edge weights. By iterating this process multiple times, the segmentation result can be improved without requiring additional user input. If the resulting segmentation is still inaccurate in some places, the user can refine it by labeling additional parts of the image and repeat the process. Our presented method is inspired by their work and extends it to the domain of textured 3D point clouds. Other previous work extends the GrabCut technique to single RGB-D images [SD13] or reconstructed polygonal scenes [MD15].

A number of automatic methods for the segmentation of 3D point clouds have been proposed, based on region growing and model fitting, unsupervised clustering, classic supervised or semi-supervised machine learning as well as deep learning. They have thoroughly been covered in the literature and we want to refer the reader to previous surveys on the topic [NL13; GMR17; XTZ20]. The majority of work uses only geometric information, without further attributes such as color, texture or semantics.

For interactive point cloud segmentation, Golovinskiy and Funkhouser present a purely geometrical approach for 3D point clouds [GF09]. They compute a minimum cut on a  $k$ -nearest neighbor graph with inverse distance weighting of the edges and a background radius prior for the expected size of the segmented object. Sedlacek and Zara perform graph cuts on 2.5D terrain data with edge weights based on Euclidean distance and the goal to segment buildings from a ground surface [SZ09]. Sallem and Devy incorporate a single color per splat into their graph cut formulation for the segmentation of single RGB-D images [SD13]. An approach based on model-fitting within a region of interest defined via 2D sketching was presented by Steinlechner et al. [SRS\*19].

### 3. Method

In the following, we present our method for the user-assisted segmentation of 3D point clouds. In contrast to previous methods,

we target *textured* point clouds as used in textured splat rendering [BHZK05]. We leverage the rich information in the splat textures for a more accurate and reliable segmentation. Since each splat is associated with the section of a high-resolution photograph, we can use image similarity metrics to take the detailed surface appearance into account. We show that this significantly improves the segmentation accuracy compared to using just a single RGB color per splat as in previous work [SD13]. Moreover, our goal is to conveniently segment multiple object instances at the same time.

The input to our method is a 3D point cloud with a texture patch associated with each point. Given a position and an estimated normal that defines a tangent plane, the texture information can be obtained by projecting each splat into an image. For high-quality reconstructions based on laser range scanners, these are typically high-resolution photographs taken during range data acquisition. For photogrammetric reconstructions, the input images can directly serve as the source of texture data for each reconstructed point.

It is important to note, that our segmentation algorithm can be performed independently of the input modality, e.g., on a desktop workstation with traditional input devices or even non-interactively. Yet, we propose to perform the task in an immersive virtual environment, where the user can directly interact with the point cloud. We believe that the added depth perception and direct 3D interaction help to achieve an intended segmentation quicker and more easily. Evaluating this in a formal user study is, however, not within the scope of this work and will be investigated in the future.

#### 3.1. Overview

Our interactive segmentation involves the following steps:

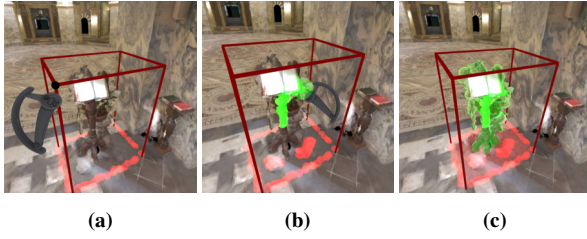
1. Select a volume of interest
2. Label some splats as foreground or background
3. Compute the segmentation
4. Optional: refine the labeling and recompute the segmentation

Inspired by the GrabCut technique, we begin with a broad selection around the desired object (see Figure 1a). The user can then interactively label points as *Foreground* (green) or *Background* (red) (see Figure 1b). The segmentation is then computed as a *minimum s-t cut*. The steps do not have to be performed in strict order and can be repeated whenever needed. To improve the segmentation accuracy without requiring additional user input, we perform multiple graph cut iterations similar to [RKB04]. Figure 1c shows an exemplary segmentation result. In the following sections, we explain the individual steps in more detail.

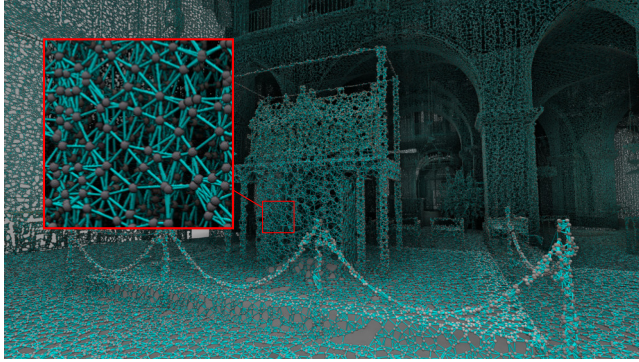
#### 3.2. Graph Cut Formulation

Our goal is to achieve a high segmentation accuracy at interactive computation times for large datasets. Similar to the work by Funkhouser et al. [GF09], we compute the segmentation as a *minimum s-t cut* on a  $k$ -nearest neighbor ( $kNN$ ) graph (see Figure 2).

To measure similarity between splats, we propose to use color and gradient histograms as the discriminative features. While being sufficiently descriptive to capture differences in surface appearance, comparing the resulting feature vectors can be performed



**Figure 1:** Interactive Segmentation: (a) Select the volume of interest, (b) Label splats and (c) Compute the segmentation.



**Figure 2:** A *k*-nearest neighbor (*kNN*) graph defines topological structure in the otherwise unstructured domain of point samples.

efficiently. To determine the class probabilities for *Foreground* or *Background* splats, we propose to train a *Support Vector Machine* (*SVM*) on the respective set of labeled splats. The *SVM* has a high discriminative performance, however can be retrained efficiently. We investigated different feature combinations and comparison measures with regard to pre-computation cost, memory requirements, runtime efficiency and segmentation accuracy. We present the techniques that we used in our experiments in the following.

### 3.2.1. Features

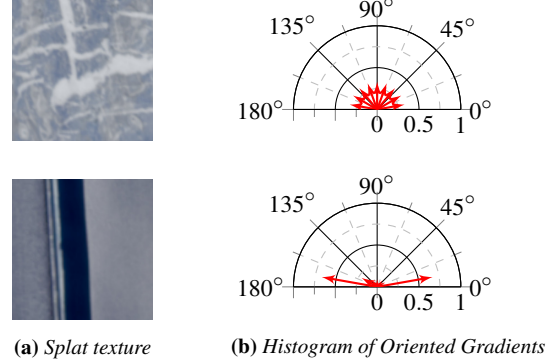
Besides geometric features like position and normal, our targeted point clouds have a splat texture assigned to every point. In this work, we focus on texture information as the discriminative feature. Our method can easily be extended to include geometric information, such as distances or normal deviation, by extending the feature vectors and adding respective penalty terms to the edge weights.

The most basic color feature is the average color per splat, which is readily available from typical RGB-D scanners. Since previous point cloud segmentation methods have used per-point colors, we compare our approach against the average color as a baseline.

To retain more information contained in the splat textures, we investigated different types of *color histograms*: one-dimensional histograms, computed separately for every color channel, and full three-dimensional histograms computed over all color channels. Table 1 shows the respective number of bins that we used. As the RGB color space is device-oriented but the segmentation result

	R	G	B	$\Sigma/\Pi$
1D	16	16	16	48
3D	8	8	8	512

**Table 1:** Bin counts for per-channel 1D and full 3D histograms



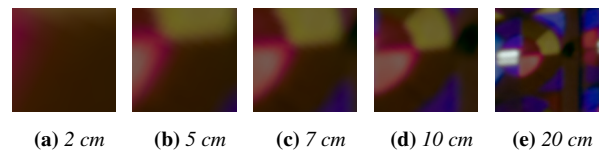
**Figure 3:** Gradient histograms for two splat textures. The feature descriptor captures the statistical distribution of edge orientations.

should focus on human perception, we investigated the CIELAB color space as an alternative.

We use a *Histogram of Oriented Gradients* (*HOG*) [DT05] to capture gradient information. Usually, when computing the *HOG* descriptor, an image is split into overlapping cells and a gradient histogram is computed for every cell. We compute only one such histogram per splat to be invariant to translation. For the gradient direction we use 9 bins. Figure 3 shows a visualization of the *HOG* features of two different splat textures.

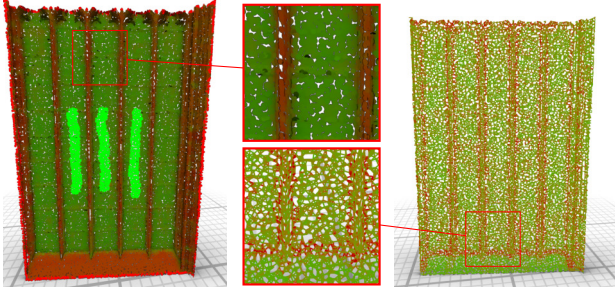
### 3.2.2. Accumulation Radius

For datasets with high geometric resolution, the texture-based features may be too local to capture similarities of the same object. For example, a colorful window may consist of splats that are mostly single-colored (see Figure 4). In this case, neighboring splats that belong to the same object can have very dissimilar feature vectors. However, we still want to encode a strong similarity. We therefore define a radius over which the feature vectors of neighboring splats are accumulated. Note, that this is independent of the radius of a single splat that is used for rendering.



**Figure 4:** A splat texture of a glass window resampled at different radii. If the radius is too small, splats showing the same object can have largely varying color distributions.





**Figure 5:** *Unary potentials (left) encode the probability of a splat to belong to the foreground. Pairwise potentials (right) encode the similarity between neighboring splats.*

### 3.2.3. Unary Potentials

The *Unary Potentials* or “regional terms”  $\varphi(p)$  encode an individual penalty of assigning the splat  $p$  to the *Foreground* or *Background* class [BJ01]. To compute the unary potentials, we train a *Support Vector Machine* (SVM) on the feature vectors of all labeled splats. We use a C-Support Vector Classification (C-SVC) machine with a radial basis function kernel [CL11].

The edge weights  $\varphi_s(p)$  for the virtual source link and  $\varphi_t(p)$  for the sink link of a splat  $p$  are computed as:

$$\varphi_s(p) = \begin{cases} K & \text{“Foreground”} \\ 0 & \text{“Background”} \\ \frac{1}{1 + \exp(-R(p))} & \text{otherwise} \end{cases} \quad (1)$$

and

$$\varphi_t(p) = \begin{cases} 0 & \text{“Foreground”} \\ K & \text{“Background”} \\ 1 - \varphi_s(p) & \text{otherwise} \end{cases} \quad (2)$$

where  $R(p)$  is the raw prediction output of the SVM, which corresponds to the signed distance from the decision boundary. It encodes the confidence for a splat belonging to either class as a potentially unbounded positive or negative number. We use the sigmoid function to obtain bounded values in the  $[0, 1]$  range. Similar to [BJ01] we define  $K$  as:

$$K = 1 + \max_{p \in P} \sum_{\{p, q\} \in N} \psi(p, q),$$

where  $P$  is the set of all splats,  $N$  is the set of all edges in the kNN graph and  $\psi(p, q)$  is the pairwise potential for the edge  $\{p, q\}$ .

Figure 5 shows a visualization of the unary potentials, where the RGB color of every splat  $p$  is defined as  $(\varphi_t(p), \varphi_s(p), 0)$ . Splats with a green color are more likely to belong to the *Foreground* class and splats with a red color to the *Background* class. Bright green or red colored splats are those labeled by the user.

### 3.2.4. Pairwise Potentials

The *Pairwise Potentials* or “boundary properties” encode the similarity of neighboring splats. They act as a geometric regularizer

and favor cuts with short segmentation boundaries between dissimilar splats. To compute similarity, histogram-based features can be compared using *bin-wise* or *cross-bin* distance measures. Using the former, a small shift in color intensities could result in a low similarity score for otherwise perceptually similar histograms [RTG00]. In contrast, cross-bin measures are not susceptible to these differences, but are computationally more expensive. We investigated the *Hellinger distance* and the *symmetric Chi-Square* ( $\chi^2$ ) distance as bin-by-bin measures [Bra00], as well as the *Earth Mover’s Distance* as a cross-bin measure [RTG00]. To convert distances to similarity, we used the exponential function as  $s(d) = \exp(-d)$ . To include a combination of different features in the pairwise potentials, we compute an affine combination of the individual measures.

Figure 5 shows a visualization of the pairwise potentials. Edges with a green color represent a higher similarity between splats and edges with a red color a lower similarity.

### 3.2.5. Iterated Graph Cuts

Similar to [RKB04] we perform iterated graph cuts. In each iteration, the resulting labels from the previous iteration are used to re-train the SVM, in addition to the user-specified labels. This way, the pairwise potentials can repeatedly act as a regularizer, which can significantly improve the segmentation result (see Section 4.1.4). If necessary, the user can perform adjustments by manually labeling additional points and repeat the process.

### 3.3. 3D User Interaction

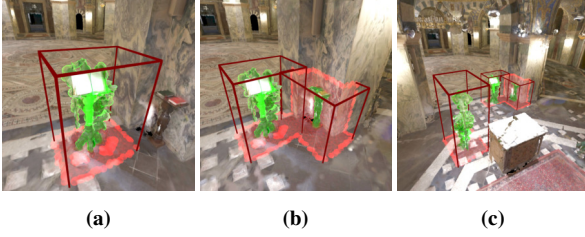
Our main methodical contribution is the graph cut formulation based on image similarity between textured splats. To evaluate our algorithm and to build an interactive workflow that enables us to apply it in practice, we chose to build a user interface in a virtual environment. It enables direct interaction with the 3D point cloud and occurs to us as a natural choice for this task.

Since our method targets large point cloud datasets, both in terms of scale and number of points, the segmentation is computed within a user-defined *Volume of Interest* (VOI) that encloses the target object. This helps to achieve interactive response times, which are required to keep the user in the loop. Using tracked motion controllers, the user can define boxes directly in 3D space by pressing a button at opposing corners. The union of all boxes then defines the VOI. To minimize the required user interaction, all splats at the boundary of the VOI are implicitly labeled as *Background*.

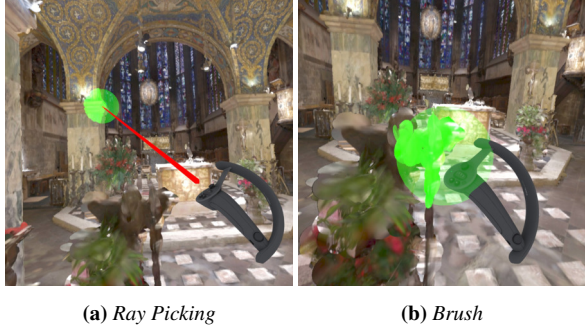
The volumes do not have to be connected and additional boxes can be added after an initial segmentation has been computed (see Figure 6). This enables to focus on one part of the scene first (a) and then extend the VOI, either by adding overlapping (b) or completely separate boxes (c). The segmentation can thus easily be extended to include multiple similar object instances.

We provide two interaction techniques for splat labeling: a picking ray and a proximity-based “brush” (see Figure 7). The picking ray can be used on 2D displays or to mark splats at a distance. The proximity-based brush offers a more natural and precise interaction method to mark nearby points in an immersive environment. Both methods mark splats around a query point within an adjustable radius. The query point of the brush is the controller position. For the





**Figure 6:** The initial volume of interest (a) can be extended by adding overlapping boxes (b) or completely separate boxes (c).



**Figure 7:** Interaction techniques for splat labeling. Ray picking enables to select splats at a distance without traveling. The proximity-based brush enables precise marking of nearby splats.

picking ray, we perform ray-marching from the controller position into the depth buffer to efficiently find the intersection.

To travel in the scene, we use pointing with the controller for hand-directed steering and the thumbstick for “snap-turn” rotation in discrete  $45^\circ$  intervals. Furthermore, the user can shrink the scene to interact at different scales, e.g. to define the VOI or perform brushing on structures that are not within arm’s reach.

### 3.4. Implementation

We implemented our method as a C++ library. An accompanying *Unreal Engine 4 (UE4)* plugin, together with a rendering plugin for textured splats, enables the user to perform the segmentation in a Virtual Reality (VR) environment. For graph-related algorithms we use the *Boost Graph Library (BGL)*. To compute the kNN graph, we use *nanoflann* [BR14] to accelerate the nearest neighbor search. Image processing and computer vision tasks are implemented with the *Open Source Computer Vision Library (OpenCV)* [Bra00].

## 4. Results and Discussion

In the following, we present a series of experiments to evaluate our proposed segmentation method. We assess the design decisions by measuring the segmentation accuracy for different feature and parameter combinations. Finally, we compare the achievable accuracy to a baseline segmentation that uses only the average splat color. The segmentation accuracy is measured as the *Jaccard index* between two sets of splat indices: a manually created ground

	Windows	Shrine	Pulpit	Statue	Chairs
Vertices	35,278	13,188	21,925	7,559	13,265
Edges	380,048	144,224	236,334	81,750	143,262

**Table 2:** Number of graph vertices and edges inside the volume of interest for our test cases, including terminal and reverse edges.

truth segmentation and the segmentation result of our method. It is defined as the intersection of the two sets over their union:

$$J(A, B) = \frac{A \cap B}{A \cup B} \quad (3)$$

Furthermore, we measure the computation times for the different stages of our method. For the interactive segmentation, we aim at around five seconds between starting the computation and displaying the result to the user. Preparation steps, like constructing the kNN graph or pre-computing features and pairwise potentials, are less time-critical, but should also be within reasonable time bounds.

### 4.1. Experiments

As our test data set, we use a 3D scan of the Aachen cathedral interior. The data set consists of roughly 6 million point samples, accompanied by 441 PNG images of different views of the scene, using 8 bits per RGB color channel and a resolution of  $3680 \times 2456$  pixels. From these images, a  $64 \times 64$  pixels splat texture is sampled for every point and used to extract the texture-based features.

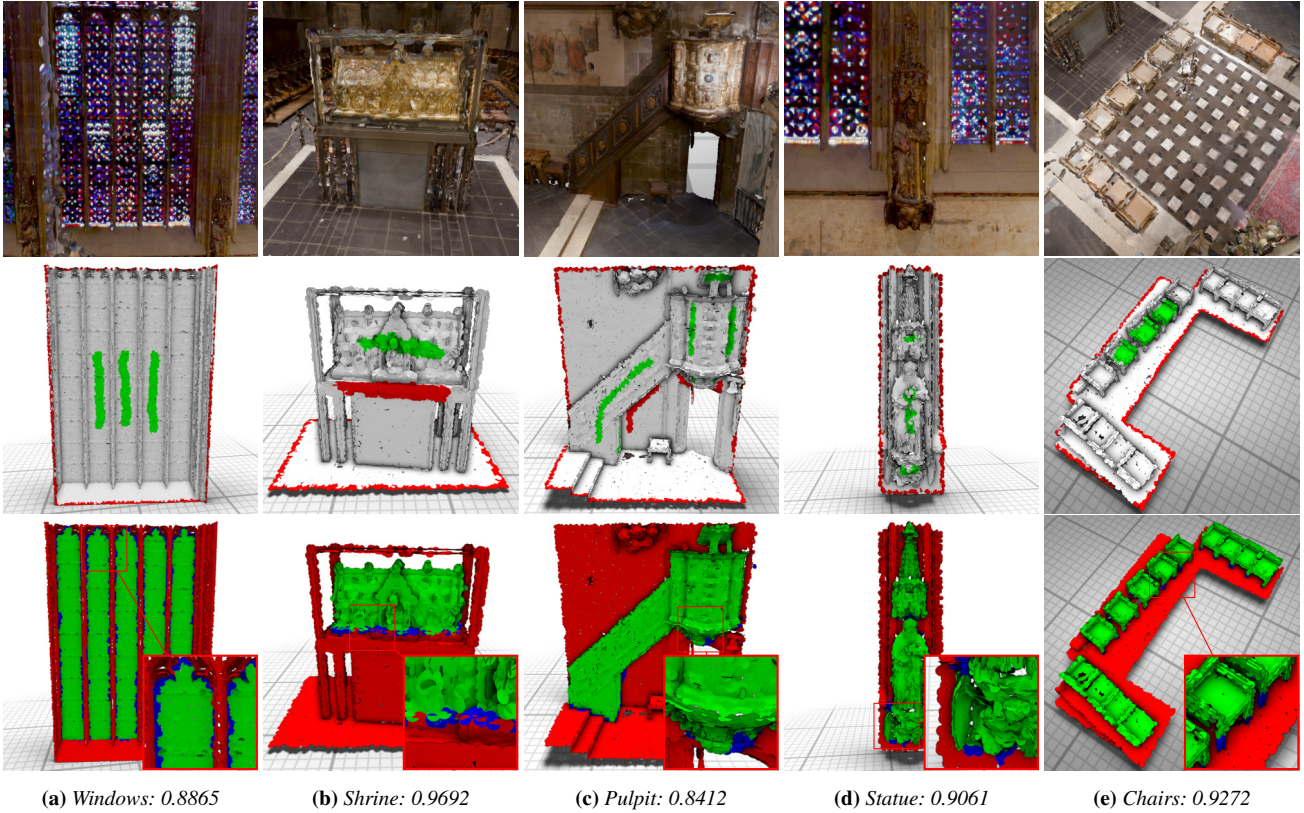
We used  $k = 6$  for the kNN graph and the final graph has around 65 million directed edges, after adding the terminal nodes and reverse edges. The graph construction takes around 15 seconds. All experiments were performed on a Windows 10 system with the following specifications: Intel Xeon E3-1230 v3, 16 GB RAM, Nvidia GeForce GTX 1060 6GB, Samsung 860 EVO 1TB SSD.

#### 4.1.1. Test Cases

For the evaluation, we selected five test cases of varying size and complexity (see Figure 8). Some have a more complex geometry (shrine, pulpit, statue), while others include multiple objects (windows, chairs) or a higher color variance (windows). The number of vertices and edges of the test cases are shown in Table 2. Here, the terminal nodes / links and reverse edges are included.

We manually created input labelings using our interactive VR application, like a user could potentially label the scenes (see Figure 8). In most cases, only the *Foreground* (green) had to be manually labeled as the border of the volume of interest was implicitly labeled as *Background* (red). Note that such a sparse labeling can be created in a matter of seconds.

To measure the segmentation accuracy, we manually created a ground truth segmentation for each test case. Here the depth perception in VR and the use of the 3D motion controllers provided a great advantage over a 2D display with ray picking. Using the latter, it was very difficult and time-consuming to create an accurate ground truth labeling, particularly for test cases with more complex geometry and protruding or occluded parts. Creating the labeling in VR was nevertheless tedious and took up to 30 minutes per test case, which underlines the motivation for this work.



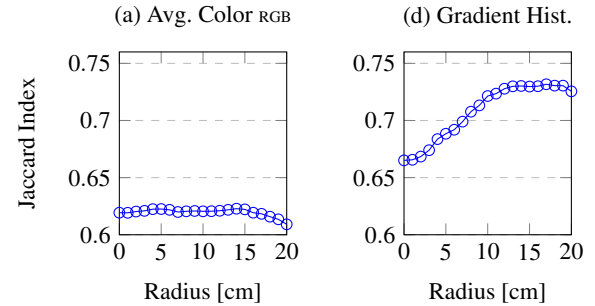
**Figure 8:** Segmentation results on our test cases. Top: textured splat rendering. Center: Labels used for the evaluation. Bottom: Initial segmentation result. The segmentation accuracy is given in the caption. Incorrectly labeled splats are highlighted in blue.

#### 4.1.2. Unary Potentials

We first measured the segmentation accuracy using only unary potentials. To do so, we computed the Jaccard index for the splat labels predicted by the SVM (see Section 3.2.3) and the ground truth labels. We did not perform extensive hyper-parameter optimization for the SVM. Table 3 shows the segmentation accuracy using the different features (see Section 3.2.1). Features in the CIELAB color space generally performed worse on our test cases. Therefore, we used RGB features for unary potentials in all further experiments.

We accumulated feature vectors over a splat's neighborhood with radii ranging from 0 to 20 cm (see Section 3.2.2). The resulting segmentation accuracy for the average color and for gradient histograms is shown in Figure 9. Since a higher radius also impacts computation time, we did not simply pick the radius that yields the highest accuracy. For the following experiments, we chose an accumulation radius of 5 cm for the average color and a radius of 10 cm for all histograms.

By combining color and gradient features, the segmentation accuracy notably improved compared to using single features (see Table 3). Furthermore, accumulating features with the previously determined radii had an even bigger impact on the segmentation accuracy. While color and gradient features can be weighted differently, we found that equal weights were already a good choice.



**Figure 9:** Segmentation accuracy for different accumulation radii.

	Single	Accumulated
Avg. Color RGB	0.619	0.626
Avg. Color RGB + Gradient Hist.	0.712	<b>0.809</b>
Color Hist. RGB	0.711	0.721
Color Hist. RGB + Gradient Hist.	0.724	0.782
Color Hist. 3D, RGB	0.684	0.715
Color Hist. 3D, RGB + Gradient Hist.	0.747	<b>0.822</b>
Gradient Hist.	0.664	0.721

**Table 3:** Segmentation accuracy for different unary feature combinations and feature accumulation within a neighborhood.

ARGB	Avg. Color RGB
H1D, RGB	Color Hist. RGB
H3D, RGB	Color Hist. 3D, RGB
GH	Gradient Hist.

Features	Train (ms)	Predict (ms)	Total (s)
ARGB + GH	[185, 758]	[35, 159]	[2.65, 6.67]
H1D, RGB + GH	[311, 2423]	[64, 535]	[3.47, 17.67]
H3D, RGB + GH	[2597, 19435]	[568, 4381]	[17.86, 129.17]

**Table 4:** Min. and max. times for different feature combinations. SVM training and prediction in milliseconds per iteration. Training times ignore the first cut, because training is much cheaper for the sparse labeling. Total times in seconds for four cut iterations.

	Average
Avg. Color RGB	0.852
Avg. Color CIELAB	0.828
Color Hist. RGB	0.844
Color Hist. CIELAB	<b>0.862</b>
Color Hist. 3D, RGB	0.840
Color Hist. 3D, CIELAB	<b>0.854</b>
Gradient Hist.	<b>0.863</b>

**Table 5:** Segmentation accuracy of pairwise potential features.

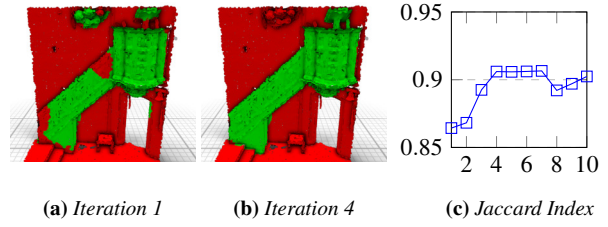
As the unary potentials have to be repeatedly updated before computing a minimum s-t cut, it is desirable that training the SVM and computing the unary potentials is as fast as possible. Table 4 shows the required minimum and maximum times for training the SVM on the combined features and computing the unary potentials for all splats in the volume of interest. While 3D histograms yielded the best results, we accept a slightly lower accuracy in favor of a faster computation time and use the feature combination of average color and gradient histogram in all further experiments.

#### 4.1.3. Pairwise Potentials

With the unary potentials defined, we evaluated the effect of the pairwise potentials. We measured the overall segmentation accuracy using different features and kept all parameters for the unary potentials fixed. We used the same accumulation radii as for the unary potentials. Table 5 shows the resulting segmentation accuracy. Here, we used the earth mover’s distance to compare color histograms and  $\exp(-d)$  to compare the average color and the gradient histograms, where  $d$  is the Euclidean distance.

Using pairwise potentials notably improved the segmentation accuracy. Gradient histograms outperformed the other features, however by a small margin. In further tests we found that a combination of color features and gradient histograms did not significantly improve the segmentation accuracy. We tested the different comparison measures (Hellinger,  $\chi^2$ , Earth Mover’s Distance) for color histograms and found that it had a negligible impact on the resulting accuracy. The required times to compute the pairwise potentials for the test cases ranged from about 10 to 100 seconds.

As the pairwise potentials do not have to be computed at run-time, we decided to use 3D color histograms with EMD in addition



**Figure 10:** Segmentation result over multiple graph cut iterations.

	Windows	Shrine	Pulpit	Statue	Chairs	Average
Base	0.611	0.701	0.627	0.717	<b>0.931</b>	0.717
Ours	<b>0.887</b>	<b>0.969</b>	<b>0.841</b>	<b>0.906</b>	0.927	<b>0.906</b>

**Table 6:** Comparison of the segmentation accuracy for our method and a segmentation with only the average color as a feature (base).

tion to gradient histograms in all further experiments. Although the combination performed neither significantly better nor worse than gradient histograms alone, we believe that this choice generalizes better to new datasets since more information is retained.

#### 4.1.4. Iterated Graph Cuts

By computing multiple graph cut iterations, the segmentation accuracy could be improved further without additional user input. Figure 10 (a) and (b) show how missing or incorrectly labeled regions are corrected after multiple iterations. However, we found that in some cases the iterated segmentation does not converge to a stable solution and, depending on the scene, can deteriorate after many iterations. In our experiments, we found four iterations to be a good choice (see Figure 10 (c)). Using this limit, we achieve computation times of around 5 seconds between starting the segmentation and displaying the result to the user (see Table 4).

#### 4.1.5. Baseline Comparison

Table 6 shows the segmentation accuracy of our method using color and gradient histograms as features, accumulating features over the splat neighborhood and computing multiple graph cut iterations. We compare to the baseline that uses only the average color as a feature, without accumulated features and iterated graph cuts. Our method performed significantly better in almost all cases. Although using iterated graph cuts for the baseline improved the accuracy in some cases, it reduced the overall accuracy. Note, that we measured the segmentation accuracy in our experiments on a fixed set of input labels and without further user interaction. Figure 8 shows the initial segmentation results for our test cases.

## 4.2. Discussion

We achieved high segmentation accuracies on our test cases and were able to compute the segmentation in about five seconds to achieve an interactive workflow. We showed that texture-based features are beneficial and particularly the gradient histograms had a positive impact on the segmentation accuracy. Furthermore, the segmentation of multiple objects at the same time is possible and provided good results.



The size of the used dataset of approximately 6 million points is, however, just at the boundary of interactivity. When we tested the method on another similar but much larger dataset with approximately 71 million points, we were no longer able to stay within our prescribed bounds regarding computation time. While the segmentation algorithm can run non-interactively, we lose the benefit of being able to observe and improve the result while being immersed in the virtual environment. Note that the size of the full dataset is not the limiting factor, but the number of points within the ROI and thus the size of the graph cut problem. An interesting avenue for future work would be to reduce the size of the problem with a heuristic that aggregates similar adjacent splats that are unlikely to be part of the segmentation boundary.

Generating the feature files for large graphs also required high computation times and disk space. For example, generating the three-dimensional RGB color histogram for the larger data set took around 8 hours and required 135 GB of disk space. As the color histograms are very sparse, signatures could be used instead to reduce the required disk space (see [RTG00]). This could also reduce the required time to access the feature vectors at runtime.

Furthermore, finding suitable features and parameter values was challenging and time-consuming. Exploring data-driven approaches such as [NAS\*17] would be beneficial, for which our method can serve to quickly generate real-world training examples.

Using iterated graph cuts did not always reliably improve the segmentation result and in some cases even deteriorated to an unusable result. To counter this, we chose a fixed number of iterations that we found worked well with our data set, but our method still needs work in this regard to become more robust.

We found that the interaction in our VR application worked very well and has many advantages compared to using a mouse and keyboard setup. However, this is only our own experience, as no formal usability study was conducted.

## 5. Conclusion

We presented an interactive method for the segmentation of textured point clouds. It employs the high-resolution photographs used for textured splat rendering as the discriminative feature. We could show that our approach increases the achievable segmentation accuracy significantly compared to using an average color per splat. We implemented an interface to our algorithm in an immersive virtual environment. This supports the user's workflow with improved spatial perception and the ability to interact directly with the 3D point cloud. With the presented technique, users can create accurate segmentations of textured point clouds fast and conveniently.

## References

- [BHZK05] BOTSCH, MARIO, HORNING, ALEXANDER, ZWICKER, MATTHIAS, and KOBBELT, LEIF. "High-quality surface splatting on today's GPUs". *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics*, 2005. IEEE. 2005, 17–141 1, 2.
- [BJ01] BOYKOV, YURI Y and JOLLY, M-P. "Interactive graph cuts for optimal boundary & region segmentation of objects in ND images". *Proceedings eighth IEEE international conference on computer vision. ICCV 2001*. Vol. 1. IEEE. 2001, 105–112 2, 4.
- [BR14] BLANCO, JOSE LUIS and RAI, PRANJAL KUMAR. *nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) with KD-trees*. <https://github.com/jlblancoc/nanoflann>. 2014 5.
- [Bra00] BRADSKI, GARY. "The OpenCV Library". *Dr. Dobbs's Journal of Software Tools* (2000) 4, 5.
- [BSK04] BOTSCH, MARIO, SPERNAT, MICHAEL, and KOBBELT, LEIF. "Phong splatting". *Proceedings of the First Eurographics conference on Point-Based Graphics*. Eurographics Association. 2004, 25–32 1.
- [CL11] CHANG, CHIH-CHUNG and LIN, CHIH-JEN. "LIBSVM: a library for support vector machines". *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011), 1–27 4.
- [DT05] DALAL, NAVNEET and TRIGGS, BILL. "Histograms of oriented gradients for human detection". *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. 2005 3.
- [GF09] GOLOVINSKIY, ALEKSEY and FUNKHOUSER, THOMAS. "Min-Cut Based Segmentation of Point Clouds". *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. 2009 2.
- [GMR17] GRILLI, ELEONORA, MENNA, FABIO, and REMONDINO, FABIO. "A review of point clouds segmentation and classification algorithms". *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 42 (2017), 339 2.
- [LW85] LEVOY, MARC and WHITED, TURNER. *The use of points as a display primitive*. Citeseer, 1985 1.
- [MD15] MEYER, GREGORY P. and DO, MINH N. "3D GrabCut: Interactive Foreground Extraction for Reconstructed 3D Scenes". *Proceedings of the 2015 Eurographics Workshop on 3D Object Retrieval*. 2015 2.
- [NAS\*17] NOH, HYEONWOO, ARAUJO, ANDRE, SIM, JACK, et al. "Large-scale image retrieval with attentive deep local features". *Proceedings of the IEEE international conference on computer vision*. 2017, 3456–3465 8.
- [NL13] NGUYEN, ANH and LE, BAC. "3D point cloud segmentation: A survey". *2013 6th IEEE conference on robotics, automation and mechatronics (RAM)*. IEEE. 2013, 225–230 2.
- [PZVG00] PFISTER, HANSPETER, ZWICKER, MATTHIAS, VAN BAAR, JEROEN, and GROSS, MARKUS. "Surfels: Surface elements as rendering primitives". *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 2000, 335–342 1.
- [RKB04] ROTHER, CARSTEN, KOLMOGOROV, VLADIMIR, and BLAKE, ANDREW. "' GrabCut' interactive foreground extraction using iterated graph cuts". *ACM transactions on graphics (TOG)* 23.3 (2004), 309–314 1, 2, 4.
- [RTG00] RUBNER, YOSHI, TOMASI, CARLO, and GUIBAS, LEONIDAS J. "The earth mover's distance as a metric for image retrieval". *International journal of computer vision* (2000) 4, 8.
- [SBMK20] SCHMITZ, PATRIC, BLUT, TIMOTHY, MATTES, CHRISTIAN, and KOBBELT, LEIF. "High-Fidelity Point-Based Rendering of Large-Scale 3-D Scan Datasets". *IEEE computer graphics and applications* 40.3 (2020), 19–31 2.
- [SD13] SALLEM, NIZAR K. and DEVY, MICHEL. "Extended GrabCut for 3D and RGB-D Point Clouds". *15th International Conference on Advanced Concepts for Intelligent Vision Systems - Volume 8192*. 2013 2.
- [SRS\*19] STEINLECHNER, HARALD, RAINER, BERNHARD, SCHWÄRZLER, MICHAEL, et al. "Adaptive pointcloud segmentation for assisted interactions". *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 2019, 1–9 2.
- [SZ09] SEDLACEK, DAVID and ZARA, JIRI. "Graph Cut Based Point-Cloud Segmentation for Polygonal Reconstruction". *International Symposium on Visual Computing*. 2009 2.
- [XTZ20] XIE, YUXING, TIAN, JIAOJIAO, and ZHU, XIAO XIANG. "Linking points with labels in 3D: A review of point cloud semantic segmentation". *IEEE Geoscience and Remote Sensing Magazine* 8.4 (2020), 38–59 2.