Adaptive Pointcloud Segmentation for Assisted Interactions

Harald Steinlechner
VRVis Research Center
hs@vrvis.at

Bernhard Rainer
Austrian Institute of Technology

Michael Schwärzler*
Delft University of Technology

Georg Haaser
VRVis Research Center

Attila Szabo
VRVis Research Center

Stefan Maierhofer
VRVis Research Center

Michael Wimmer
TU Wien

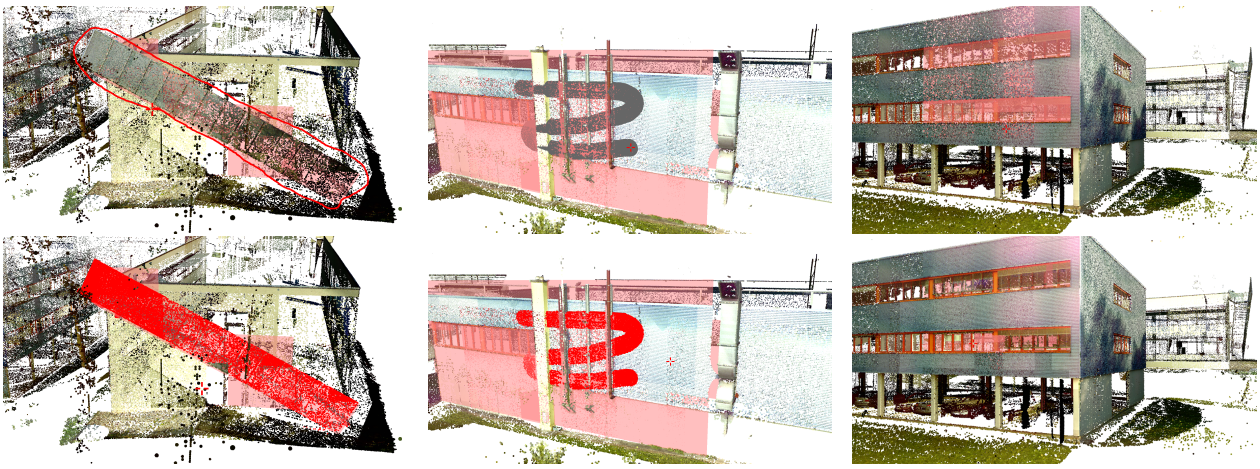

Figure 1: Our novel interactive approach for shape detection in point clouds allows for sophisticated interactions: *Left*: A Lasso selection selects only points that lie on the support shape as shown in the top image. Points in front and back of the support shape are not selected (bottom). *Middle*: A volumetric brush selection is performed on the selected support shape (top). Points are only selected if they belong to the support shape and intersect the brush (bottom). *Right*: Interactive LoD increment interaction along the selected support shape (drawn in red). The top image shows the original rendering model of the point cloud; the bottom image shows the point cloud with the additional points.

ABSTRACT

In this work, we propose an interaction-driven approach streamlined to support and improve a wide range of real-time 2D interaction metaphors for arbitrarily large pointclouds based on detected primitive shapes. Rather than performing shape detection as a costly pre-processing step on the entire point cloud at once, a user-controlled interaction determines the region that is to be segmented next. By keeping the size of the region and the number of points small, the algorithm produces meaningful results and therefore feedback on the local geometry within a fraction of a second. We can apply these finding for improved picking and selection metaphors in large point clouds, and propose further novel shape-assisted interactions that utilize this local semantic information to improve the user's workflow.

CCS CONCEPTS

• **Computing methodologies** → **Point-based models**; **Shape detection**.

KEYWORDS

Pointcloud Segmentation, Shape Detection, Interactive Editing

ACM Reference Format:
Harald Steinlechner, Bernhard Rainer, Michael Schwärzler, Georg Haaser, Attila Szabo, Stefan Maierhofer, and Michael Wimmer. 2019. Adaptive Pointcloud Segmentation for Assisted Interactions. In *Symposium on Interactive 3D Graphics and Games (I3D '19), May 21–23, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3306131.3317023

1 INTRODUCTION

Acquisition of point clouds from real environments becomes ever simpler and cheaper. Laser scanners, depth cameras, and photogrammetric reconstructions are routinely used to produce highly detailed point clouds that contain hundreds of millions or even billions of points. This sheer size presents challenges to data processing as well as user interaction. As data is too large to fit into system and GPU memory, various out-of-core solutions have emerged [Elseberg et al. 2013; Rusu and Cousins 2011; Scheiblauer 2014]. Most approaches create structured files in a preprocessing step, then

*Also with VRVis Research Center.

I3D '19, May 21–23, 2019, Montreal, QC, Canada

transfer chunks of points to memory as needed based on different culling and caching heuristics.

It is usually necessary to extract regions of interest or to remove imperfect, noisy or nonessential data, depending on the application domain. However, achieving this goal using standard 2D manual interaction metaphors can be tedious due to the lack of topological structure in point clouds. Selecting or picking with a mouse or a touch-enabled device often results in inadvertently *picking through* a desired object.

A way of introducing topological structure into unstructured point sets is shape detection. The objective is to find regions with similar characteristics, such as local curvature and neighborhoods, to help the system better understand local and global structures. For example, Schnabel et al. [2007a; 2007b] demonstrate high quality segmentation of point clouds into sets of (partial) geometric primitives. Unfortunately, performance of such algorithms is far from real-time, requiring minutes for a few million points, and being infeasible for billions of points. It also does not make sense to perform the segmentation as a preprocessing step, because the point cloud changes due to interactive editing. On the other hand, most user interactions, at a single point in time, only require a very specific subset of all the possible topological information.

We therefore propose an *interaction-driven* approach streamlined to support and improve a wide range of real-time 2D interaction metaphors for arbitrarily large (out-of-core) point clouds. By repeatedly performing shape detection on a very small subsets of points relevant to the current interaction, we create a *continuous stream of partial shape approximations*, which in turn is used to *incrementally* build up a bigger picture of the environments topological structure.

Our main contributions are:

- A background process generating a continuous stream of partial shape matches on massive out-of-core point clouds.
- Real-time clustering of aforementioned partial shapes into coherent larger shapes.
- Improved *point picking* and *region selection* user interactions using local geometry as support shapes, with the further benefit of shorter computation times.
- A novel interaction technique allowing the user to interactively *increase the level of detail locally along a shape* to focus on detail structures.

2 RELATED WORK

The scope of this paper covers data management, shape detection and interactive editing of massive point clouds.

2.1 Out-of-Core Point Clouds

The QSplat system [Rusinkiewicz and Levoy 2000] was one of the earliest systems capable of handling datasets with well over hundred million points. A hierarchy of bounding spheres makes it easy to perform visibility culling and LoD control. Gobbetti and Marton [2004] proposed Layered Point Clouds (LPC), a multi-resolution approach for rendering massive point clouds. Points are stored in a binary tree of chunks of approximately the same size, including LoD representations. Compared to the QSplat system, LPC hides out-of-core latency by speculatively fetching data, reduces the cost of traversal on the CPU-side significantly, and benefits from the parallel GPU architecture. Wimmer and Scheiblauer [2006] introduced nested octrees to create a multi-resolution model similar to LPCs. A nested octree consists of an outer octree used for visibility culling and a memory-optimized SPT as inner octree for efficient point rendering. Dachsbauer et al. [2003] introduced Sequential Point Trees (SPT), a data structure that arranges and sorts points (by depth) in a list, allowing for adaptive rendering of point clouds completely on the GPU (but also bound by GPU memory).

Besides rendering and storing, modifications are a key task for out-of-core point cloud systems. Wand et al. [2007] describe a multi-resolution out-of-core octree, that stores additional points per LoD to enable efficient removal and insertion operations. The Modifiable Nested Octree (MNO) by Scheiblauer and Wimmer [2011] replaces SPTs from inner nodes with a grid and introduce a so-called *Selection Octree* storing selected points separately, allowing to interactively change the visualization model without actually having to modify original data permanently. Wenzel et al. [2014] propose a system tailored towards quick data updates by maintaining an additional cycle stack as a node history of in-memory data.

2.2 Shape Detection and Processing

The objective of detecting structures in point clouds is defined very loosely, since structures can relate to geometric primitives, but also to more complex components representing distinct man-made or natural formations. In our work, we are interested in detecting basic geometric shapes. The 2D Hough transform [VC 1962] is used in image processing to detect lines and curves. It was extended to 3D by Maas et al. [1999] and refined by Rabbani and Van Den Heuvel [2005].

Schnabel et al. [2007a] propose Random Sampling Consensus (RANSAC) [Fischler and Bolles 1981] to extract a minimal set of primitive shapes approximating the global structure of the point cloud. The algorithm randomly selects a set of points that roughly follow the curvature of a shape. If a defined number of points are approximated by this shape, the shape is considered valid.Our system is based on an extension to out-of-core datasets [Schnabel et al. 2007b].

GlobFit by Li et al. [2011] uses RANSAC to detect not only primitive shapes, but also their global mutual relations, such as orientation, placement, and equality among shapes. Oesau et al. [2016] propose an alternative approach to detect planar shapes and their relations using region growing, and Attene and Patanè [2010] use a hierarchical method construct a multiresolution representation for region selection.

2.3 Interactions

Interactions (such as removing unwanted points, selecting regions of interest, or creating new geometry) are crucial to improve the data quality of the point cloud. The Lasso tool is a common interaction metaphor to select regions in 2D screen space. It was applied in 3D space by Lucas et al. [2005] by drawing the lasso on a tracked 2D canvas that shows a desired view of the scene. Yu et al. [2012] present two new methods of interaction that turn a 2D lasso into a three-dimensional volume that is fitted to the spatial structure of the point cloud: Similar to sketch-based modeling

[Igarashi et al. 2007], TeddySelection inflates a user-drawn lasso using a heuristic accounting for local point density. CloudLasso uses Marching Cubes [Lorensen and Cline 1987] to identify and select regions within the lasso where the density is beyond a threshold. Both techniques only use two degrees of freedom, thus can be used in traditional mouse-based interactions, as well as in direct-touch environments. Another example of a 3D interaction technique is the Volumetric Brush by Weyrich et al. [2004]. A brush follows the local curvature by tracking the current depth of the cursor in the z-Buffer. Scheiblauer and Wimmer [2011] utilize the volumetric brush for selections in point clouds.

Huang et al. [2014] propose GPU-driven point picking. The idea is to perform picking in screen space and choose the point that is closest to the mouse position. Potree [Schütz 2016] applies a similar technique without using compute shaders. Instead, a unique per-point id is rendered into a texture from which a small window around the cursor is downloaded to the CPU to perform the final picking decision.

3 OUT-OF-CORE OCTREE

Our approach relies on an out-of-core octree with a structure similar to that of Wand et al [2007], where the entire point set is distributed among the node's children. A random subset of points is duplicated (in contrast to Instant Points [Wimmer and Scheiblauer 2006]) to create a LoD representation, which also makes each node self-contained as no points from parent nodes are needed. This results in a simpler system and we save memory, as we only need to keep nodes representing the current rendering horizon alive.

Our octree split criterion is a fixed point count threshold (5000), which keeps variations in loading time and shape detection low, and allows for immediate feedback in user interactions. We apply view-frustum culling to consider only those nodes that contribute to the currently viewed scene. If the octree is culled purely for rendering purposes, it would be sufficient to collect visible nodes in a set and use this for rendering. However, since interaction and processing tasks are performed on the viewed data, the hierarchical structure is kept intact.

We also compute and store additional properties used in the *Shape Detection and Clustering* steps (Section 4). Each node contains an *rkd-tree* [Tobler 2011] for efficient ray-queries and k-nearest neighbor searches, per-point normals, and the points' centroid and average point distance which we use as approximate for the spatial *density*. For the average point distance we use nearest neighbor distance queries for each point and use their average value.

4 SHAPE DETECTION AND CLUSTERING

Our proposed system is not tied to any specific shape detection algorithm. Any algorithm will do, as long as it is capable of extracting (partial) shapes from single octree cells and for which a clustering metric for merging partial shapes can be defined.

Currently our implementation uses the algorithm by Schnabel et al. [Schnabel et al. 2007a] for proof of concept, as it provides different types of primitive shapes like planes, spheres, cylinders, cones, and tori. As we found, the algorithm is particularly suitable for the detection of primitives in urban environments. We use the average point distance in a node as the algorithm's ϵ parameter. Parameters for the minimum number of support points n per shape (250) or the normal deviation α (0.95) are constant. As we perform shape detection only on the points of a single octree node at a time (5000 points or less) its performance is adequate.

Since planes, cylinders, and cones are of infinite size, they need to be refitted to encapsulate the corresponding support points and create a minimal boundary. This is especially true for rendering and interaction. Since our approach produces a continuous stream of new primitive shapes and accurate reconstruction is not a goal, the refitting process only uses the set of support points to refit the shape and create a finite representation. For planes, we calculate a bounding quad used as the representation of the plane. The quad is obtained by using the minimum-bounding-rectangle algorithm by Freeman [1975]. Cylinders and Cones are adjusted to have minimal volume but still contain all their support points.

4.1 Shape Matching

Performing shape detection on single octree nodes naturally results in (partial) shapes limited to the extents of the node. Of course these shapes are presumably part of a larger structure. We therefore continuously match newly found (partial) shapes to previously matched shapes to form larger coherent shape clusters. To this end, we propose a set of matching functions to determine if two shapes originate from the same geometry. Only shapes of the same type can be matched to form a cluster.

4.1.1 Elementary Matching Functions. As primitive shapes are represented by only a handful of parameters, we use a simple similarity measure (with threshold λ) based on the following elementary matching functions:

- **Scalars** f_1, f_2: $\frac{f_1}{f_2} \geq \lambda$, where $f_1 \leq f_2$
- **Directions** v_1, v_2: $\frac{v_1}{|v_1|} \cdot \frac{v_2}{|v_2|} \geq \lambda$
- **Positions** p_1, p_2:

$$\sqrt{(p_1.x - p_2.x)^2 + (p_1.y - p2_y)^2 + (p_1.z - p_2.z)^2} \leq \lambda$$

- **Axes** a, b:
  An axis is defined by a start and end point (a_{start}, a_{end}, b_{start}, b_{end}). Furthermore, let $v_a = a_{end} - a_{start}$, and $v_b = b_{end} - b_{start}$, and let d be the largest distance of a start- or end point to its complementary axis. Similarity is then defined as follows:

$$\frac{v_1}{|v_2|} \cdot \frac{v_2}{|v_2|} \geq \lambda_1 \wedge d \leq \lambda_2$$

4.1.2 Primitive Shape Matching Functions. Based on the elementary matching functions, we define the following pairwise matching functions for shapes:

- A **plane shape** consists of a quad enclosing all support points. Let d be the largest normal distance of a corner of a quad to the plane of the other quad. Two plane shapes match, if the planes' normal vectors are similar within a threshold λ_1 and if $d \leq \lambda_2$.
- Two **cylinder shapes** are matching if radii and axes are matching.
- Two **cone shapes** are matching if axes, apexes and opening angles are matching.

- Two **sphere shapes** are matching if centers and radii are matching.
- Two **torus shapes** are matching if center, axes, major radii, and minor radii are matching.

Table 1 shows all threshold values used for our implementation.

Table 1: Threshold values for matching.

	threshold values
Scalars	$\lambda = 0.99$
Vectors	$\lambda = 0.95$
Positions	$\lambda = 0.05$
Axes	$\lambda_1 = 0.95, \lambda_2 = 0.05$
Planes	$\lambda_1 = 0.99, \lambda_2 = 0.05$

4.2 Shape Clustering

Our graph-based shape clustering algorithm is based on Oesau et al. [2016], but not limited to planes. We search the octree for partial shapes that were detected previously and that match a base shape. From this set of partial shapes, a complete graph is created, and a region-growing algorithm reduces the number of shapes to those that create a coherent shape cluster.

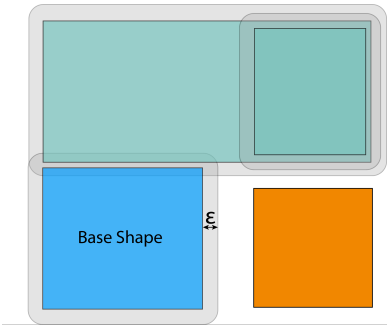

Figure 2: A cluster of plane shapes created by computing the ϵ-connected component. The base shape is colored in blue. Planes that belong to the cluster are colored in turquoise. Shapes that do not belong to the cluster are colored in orange. The cluster's ϵ distance is drawn in grey. Each shape that intersects this area belongs to the cluster.

4.2.1 Building a set of matching primitive shapes. Clusters are constructed from currently visible primitive shapes. Therefore, rather than searching the full octree, its current rendering horizon is queried for shapes matching the base shape. The base shape, selected by the user, has a particular level of detail. For the shape cluster to mimic a structure of a similar level of detail, only shapes with a level of detail within a user-specific threshold l are used. Using a threshold value of 0 creates small clusters, as larger overlapping shapes cannot form bridges between smaller shapes. For our test data, a threshold $l = 2$ creates clusters large enough to include meaningful parts of a structure, while still having little to no unwanted shapes.

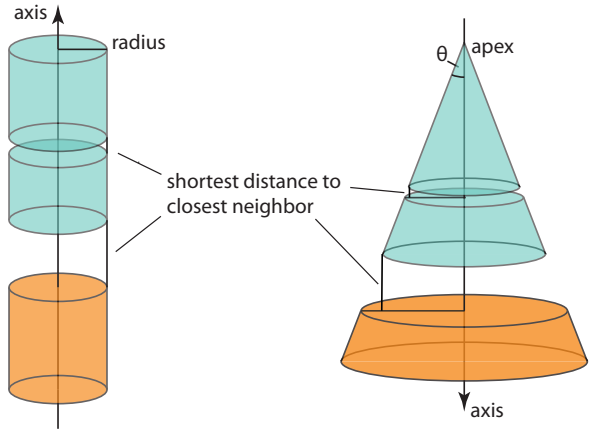

Figure 3: This figure shows a cylinder cluster and a cone cluster build from matching shapes. Shapes that belong to the cluster are colored in turquoise.

In such a cluster each primitive shape can already be seen as a part of a larger shape. However, distances between shapes are not yet taken into account. Thus, gaps may still exist. Hence, for infinite shapes, an additional step is performed using a graph-based region-growing algorithm.

4.2.2 Graph-based Region Growing. A cluster of shapes can be seen as an ϵ-connected component from a larger graph. A complete graph is created from the vertices of all matching shapes, including the base shape. In a complete graph, an edge exists between any pair of vertices. The weight of an edge is determined by a distance function for each primitive shape:

- As a plane shape is bounded by a quad, the **distance between two plane shapes** is computed as the shortest distance between the two bounding quads.
- The **distance between two cylinder shapes** is determined by the shortest distance between all pairs of start/end points of both cylinders.
- The **distance between two cone shapes** is determined by the shortest distance between all pairs of start/end points from both cones.

Overlapping shapes can occur if shapes from multiple levels of detail are clustered together. However, this causes no problems, as the distance between two overlapping shapes is set to 0. A cluster is created by growing a region in a graph, only adding vertices that connect to the current region via an edge with weight smaller than ϵ. This creates a cluster of shapes, ensuring that the distance to the closest neighboring shape is at maximum ϵ.

Figure 2 shows an exemplary illustration of region growing for plane shapes. The distance between two plane shapes is measured as the closest distance between the two bounding quads. Figure 3 illustrates region growing for both cylinder shapes and cone shapes. The matching heuristic already confirms that the shapes lie on the same axis and share a similar radius. Therefore, instead of a 3D world distance, a one-dimensional distance between two points is sufficient to build a shape cluster. The region-growing component

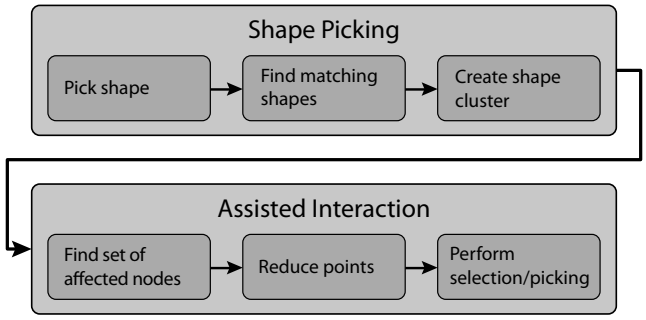

Figure 4: The workflow for shape-assisted interactions is divided into two major steps. First, the user picks a primitive shape, from which a larger cluster is created and used as support shape for the following interaction. Upon starting an assisted interaction, the system finds all nodes that are affected by the interaction, reduces the set of points to those that are approximated by this shape and, finally, performs the interaction (e.g., selection, picking).

of the clustering heavily depends on the ϵ distance threshold. For this task, we chose the density of the node of the base shape, more specific: $\epsilon = 2 \cdot density$.

5 INTERACTIVE APPLICATION

Next, we present an interactive application which finally exposes the potential enabled by our approach. To this end, we implemented an interactive heuristic for user-guided shape detection and show how picking and selection can be improved using shape-assisted interactions.

5.1 User-Guided Shape Detection

Typically, our previously described shape-detection and clustering algorithm provides results in less than 250 milliseconds for a single octree node. Based on the user's current view and mouse position, *user-guided shape detection* continuously updates the active shape in real-time. A background process continuously processes nodes that

(1) are currently rendered,
(2) intersect the pick ray,
(3) contain at least n points (= minimal support for shape detection), and
(4) have not been processed before

and favors nodes with higher LoD, such that the user receives geometric information for the most detailed parts of the currently explored region first. For nodes with the same LoD, the node closest to the camera is chosen.

The selection of a suitable candidate node depends heavily on the camera position. When zooming out, the camera moves away from the scene, thus reducing the render horizon and therefore reducing the maximum LoD. If the node with the highest LoD is processed already, the heuristic chooses a node from a lower LoD. Thus, a multi-scale representation of the local geometry is constructed over time.

5.2 Shape Picking

The use of a support shape raises the need for an initial interaction to pick a primitive shape. A ray-cast from the current mouse position is performed to create an initial filtering of octree nodes. Then all primitive shapes from these nodes are sorted using a custom key, $key = level + 1 - depth$, with $depth$ ranging from 0 (near plane) to 1 (far plane). The primitive shape with the largest key (highest LoD and closest to the camera) is chosen as support shape. From this, shape clustering as described in Section 4.2 is applied. This step is important, since user interactions are usually performed on larger areas. A cluster can be seen as a single, multi-level shape and is referred to as *support shape*.

5.3 Shape-Assisted Interactions

In 3D applications, standard 2D interaction metaphors, such as *lasso selection* or *point picking*, are limited by the lack of information on the desired depth (see Section 2). By using a primitive shape as support, the user can easily interact with the point cloud in a way that only points are considered that belong to the support shape. Figure 4 demonstrated the basic workflow for shape-assisted user interaction, comprising two major steps. The first interaction is to pick a primitive shape and build a larger cluster from it. This cluster acts as a support shape for the following user interaction. For this interaction, when initialized, all affected nodes are collected, and their point sets are reduced to only those points that are approximated by this shape. A point is filtered if it fulfills the same ϵ-distance and normal-deviation criteria with regard to the support shape as used for the shape detection in Section 4. After the filtering step, the actual interaction is performed on the remaining points.

5.3.1 Shape-Assisted Point Picking. *Point picking* is used to select a single point from the scene. A pick radius r denotes the maximum distance of a point to the pick ray for the point to be considered. It can either be constant in world space, constant in screen space, or dependent on the depth value (see Figure 5). Each case has certain drawbacks, e.g. preferring points in the foreground, or introducing depth disambiguities. Furthermore, a single ray-cast is not sufficient to find all octree nodes that are affected by this interaction, since it is possible that the cylinder (in world space) or the cone (in screen space) also intersects neighboring nodes.

Shape-Assisted Point Picking utilizes primitive shapes to perform picking only on points that are part of a structure. The user selects a cluster to restrict candidate points to the support shape. Rather than using a cone- or ray-cast to collect all candidate nodes, only nodes intersecting the support shape are considered. This leaves only a handful of nodes containing points that mostly follow the support shape on which the interaction is performed. Points on the back of a shape are projected near the mouse position as well and might get favored over points facing the user. Therefore, picking is restricted to points inside a sphere centered at the pick ray's intersection with the support shape, which also reduces computation time. The sphere's radius is calculated by unprojecting the desired screen space pick radius to the sphere's world space center.

Usually, a shape cluster intersects far fewer nodes than a ray-cast, as the cluster's extent is limited, while a ray traverses the full

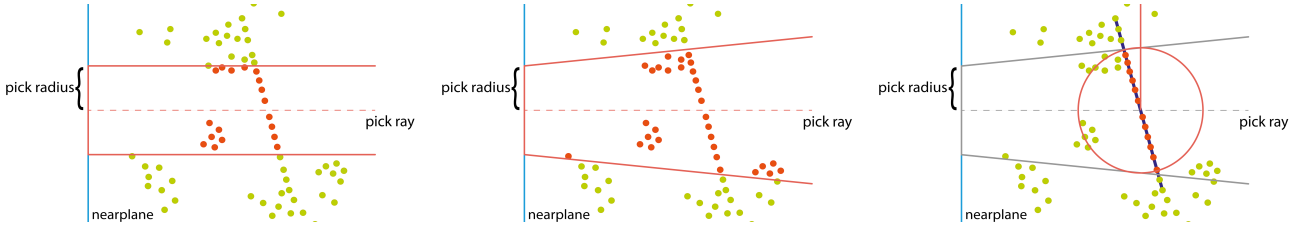

Figure 5: Comparison of picking methods. Selected points are marked red. *Left:* Ray-cast with radius using a cylinder in world space. *Middle:* Ray-case using a cone instead. *Right:* A support shape (dark blue) is used to further constrain selection.

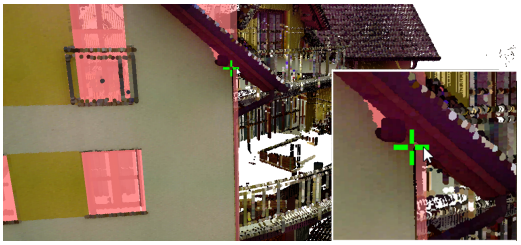

Figure 6: Shape-assisted point picking performed on a shape cluster representing the foreground wall. Note that the cross hair does not jump to a point in the background even if it is closer to the cursor in screen space.

Figure 8: *Shape-assisted Volumetric Brush* using a shape cluster (transparent red) representing the front wall. Roof points intersecting the brush path (left) are not selected (right).

extent of the point cloud. This again reduces computation time. An example of shape-assisted point picking can be seen in Figure 6.

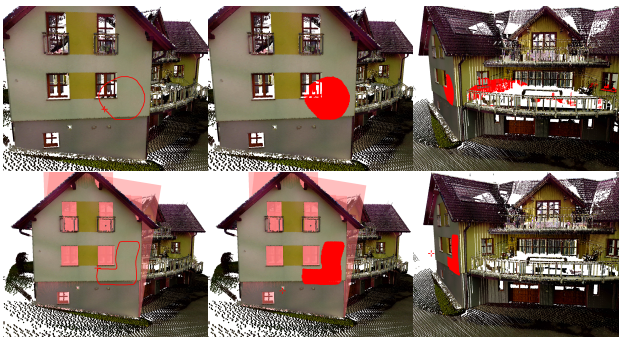

Figure 7: Top row: *Classic Lasso Selection.* The user draws a polygon in screen space (left) to select all points that are projected to the area of the polygon (middle). Same selection from a different point of view (right). Bottom Row: *Shape-Assisted Lasso Selection* avoids unintentional selection of points obscured by foreground objects.

5.3.2 Shape-Assisted Lasso Selection. *Lasso Selection* is commonly used to select regions in 2D. However, in 3D, a screen-space lasso polygon combined with the camera view creates a 3D volume (see Figure 7, top). In order to select exactly the desired point set, usually multiple select/deselect interactions from differing views are required.

We propose *Shape-Assisted Lasso Selection* to provide smaller sets of points on which a lasso selection is performed (see Figure 7, bottom and Figure 1, top). To construct these smaller sets, the octree is consulted for nodes that intersect the support shape. The number of candidate points per node is reduced by filtering points that are approximated by the support shape. On this reduced set, a normal lasso selection is performed. The result is a selection that mimics a lasso selection, with the benefit of not selecting 'through' the point cloud. The depth ambiguities of the lasso selection are circumvented by introducing continuous depth boundaries defined by the local curvature of the shape cluster.

5.3.3 Shape-Assisted Volumetric Brush. The *Volumetric Brush* is designed to project a selection volume onto the foremost geometry [Scheiblauer and Wimmer 2011; Weyrich et al. 2004]. Since this technique follows the foremost geometry only, sudden depth changes occur if different geometry occludes the area of interest. Thus, view changes are still required. In regions close to intersections with other structures, the user must control the volume size to avoid selecting points on neighboring structures. We propose *Shape-Assisted Volumetric Brush* (see Figure 8 and Figure 1, middle), in which, instead of consulting the depth buffer to reconstruct the cursor's world position, the pick ray is intersected with the selected support shape. The octree is consulted for nodes that intersect the support shape, and the sets of points are reduced to only those that are approximated by the support shape.

5.3.4 Shape-Assisted Local LoD Increment. Section 3 describes the culling heuristic to create an octree that only contains nodes that are rendered for the current frame. This heuristic uses view-frustum culling paired with LoD culling. If a user wants to investigate local structures in more detail, the currently rendered LoD might not suffice. We propose *Shape-assisted local LoD increment* to fetch

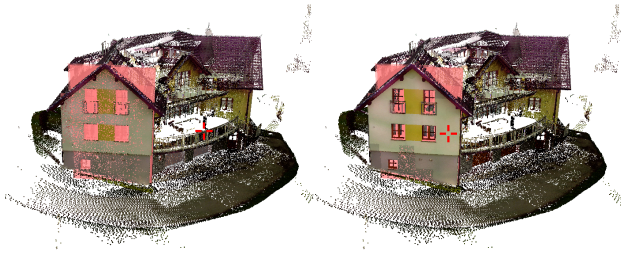

Figure 9: *Shape-assisted LoD Increment.* Default LoD (left) is increased (right) along the active shape cluster (transparent red).

Table 2: Datasets with total point count, number of octree nodes, and maximum octree depth (LoD).

	#Points	#Nodes	max. Depth
JB_House.pts	620.722	440	15
TechCenter.pts	11.762.924	8863	15
Synthetic.pts	472.000	315	5

additional data from the octree: By adding nodes without additional point filtering, noise and unwanted structures are amplified as well. Therefore, this interaction utilizes a shape cluster, selected by the user, to amplify detail only on structures of interest. The candidate set is filtered further, such that only those nodes remain that intersect the support shape. For each node from this final set, only points approximated by the support shape are added to the scene (see Figure 9 and Figure 1, bottom).

6 RESULTS

All results were obtained on a PC with an AMD FX-9590 CPU, 16 GB RAM, and an AMD Radeon HD 7970 GPU. The proposed techniques were tested on three different datasets (see Table 2). The synthetic point cloud (Figure 11) is constructed from a set of primitives discretized as point sets and includes approximations for all types of detectable primitives. In the supplemental video, we demonstrate both the interactive performance of our approach, as well as the successful application of our novel interaction metaphors for point cloud selection and picking.

6.1 Interactive Shape-Detection Performance

The goal of *user-guided shape detection* is to provide meaningful results within interactive time, such that detected shapes can be used to support interactions immediately. The capabilities of the implementation by Schnabel et al. are benchmarked on three different datasets in Table 2 using the interactive approach, as well as on the point cloud at once.

The tests are performed without user interaction. Instead, all octree nodes are collected and fed into the shape-detection pipeline sequentially to retrieve results for each node of the point cloud independently. Therefore, shape detection is performed on the complete point cloud for each LoD. Each octree node contains at most 5000 points. The results are averaged over five runs. Table 3 shows the per-node performance of the user-guided shape detection. Each node from the datasets is benchmarked five times and the results are averaged. It can be seen that detecting only planes is significantly faster than detecting all types of primitive shapes. Detecting all types of primitives still produces results within the desired time.

Table 3: Interactive shape detection performance measures for 3 datasets. The last column shows the number of shapes found in each dataset on all LoDs (without clustering). Datasets were benchmarked using plane detection only (*), and by detecting all types of primitives (results averaged over five runs).

	#Shapes per Node (avg)	Time (ms) per Node (avg)	#Shapes total
JB_House*	1.31	22.396	576.40
JB_House	1.40	93.264	616.00
TechCenter*	1.18	18.781	10458.34
TechCenter	1.19	83.180	10546.97
Synthetic*	1.89	29.779	595.35
Synthetic	1.24	136.474	390.60

Table 4: Performance measures using the original shape detection algorithm by Schnabel et al. [2007a], executed on the entire point cloud at once. Again, datasets were benchmarked using plane detection only (*), as well by detecting all types of primitives (results in seconds and averaged over five runs).

	#Shapes (avg)	Time (avg, seconds)
JB_House*	64	1.79
JB_House	66	5.60
TechCenter*	689	285.84
TechCenter	773	514.94
Synthetic*	13	4.01
Synthetic	12	8.39

The results of the original shape detection on the whole point cloud at once are shown in Table 4. The interactive approach processes the point cloud in chunks, resulting in the detected primitives being chunks of larger primitives as well. This explains the higher number of detected primitives using the interactive approach. While on small datasets, such as the JB_House, the original shape-detection approach produces results within seconds, larger datasets require significantly more computation time (up to ~9 minutes).

6.2 Shape-Detection Results

Figure 10 compares the results of the RANSAC shape detection on the entire dataset with our interactive approach. Our interactive method finds more shapes that can be grouped into one larger consecutive shape. However, the overall structure of the building is approximated similarly in both techniques. The interactive shape detection was unable to produce results on the synthetic point cloud (which can be seen in Figure 11). On coarse LoDs, implausible shapes are detected that occluded the entire scene. Hence, a visual comparison of the classic shape detection with our approach did not yield any satisfying results.

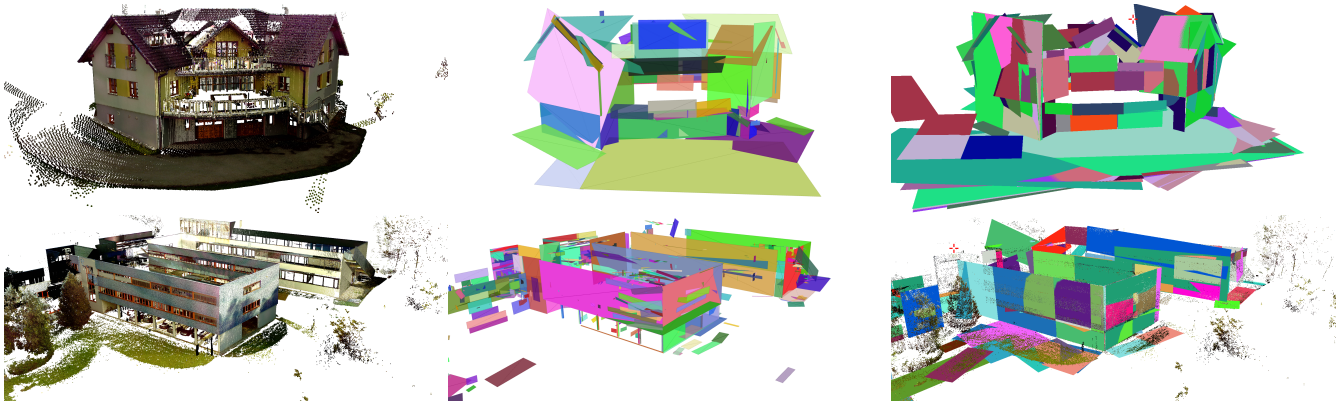

Figure 10: Top row: the JB_House dataset, bottom row: the TechCenter dataset. From left to right: Rendering, the resulting shapes of the RANSAC shape detection, and the detected shapes using the interactive shape-detection method.

6.3 Interaction Performance

Shape-assisted point picking drastically reduces computation time compared to standard point picking. Randomly picking points with and without support shape on all three datasets shows a tenfold performance increase on average (3.28ms vs. 37.96ms). Although per-point processing time is higher in shape-assisted point picking, far fewer points need to be checked due to the additional support shape constraint.

We also compared *shape-assisted lasso selection* to a classic lasso selection, sharing the same lasso polygon. The averaged computation time ratio between shape-assisted (437ms) and classic (748ms) lasso selection is ~1.81.

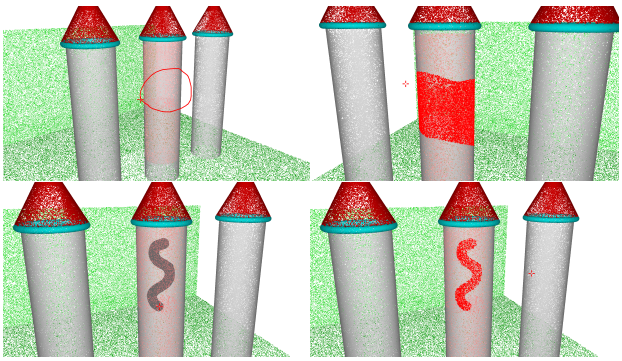

Figure 11: Top row: Lasso selection supported by a cylinder shape. Points on the back of the cylinder are selected. Bottom row: In contrast, the volumetric brush selection sticks to the side of the cylinder that is facing the camera.

7 CONCLUSION AND FUTURE WORK

In this work we proposed an interactive, user-assisted segmentation method enabling enriched interaction techniques for out-of-core pointclouds. In contrast to previous work we fluently integrate processing and interaction with a level-of-detail approach. The continuous stream of shape approximations can be used to guide interactions such as improved *point picking*. Based on *clustering of shapes*, we allow interaction techniques such as *lasso selections* to directly operate on geometric *multi-scale* approximations. In future work, we plan to further investigate on interaction techniques as well as incremental segmentation methods.

ACKNOWLEDGMENTS

VRVis is funded by BMVIT, BMDW, Styria, SFG and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (854174) which is managed by FFG.

REFERENCES

Marco Attene and Giuseppe Patanè. 2010. Hierarchical Structure Recovery of Point-Sampled Surfaces. *Computer Graphics Forum* 29, 6 (2010), 1905–1920.

Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. 2003. Sequential point trees. *ACM Transactions on Graphics (TOG)* 22, 3 (2003), 657–662.

Jan Elseberg, Dorit Borrmann, and Andreas Nüchter. 2013. One billion points in the cloud–an octree for efficient processing of 3D laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing* 76 (2013), 76–88.

Martin A Fischler and Robert C Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (1981), 381–395.

Herbert Freeman and Ruth Shapira. 1975. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Commun. ACM* 18, 7 (1975), 409–413.

Enrico Gobbetti and Fabio Marton. 2004. Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics* 28, 6 (2004), 815–826.

Ming Huang, Yanmin Wang, Yong Zhang, and Xinle Fu. 2014. Pickup of large scale point cloud based on GPU. *BioTechnology: An Indian Journal* 10, 23 (2014), 6.

Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. 2007. Teddy: a sketching interface for 3D freeform design. In *Acm siggraph 2007 courses*. ACM, New York, USA, 21.

Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J Mitra. 2011. Globfit: Consistently fitting primitives by discovering global relations. *ACM Transactions on Graphics (TOG)* 30, 4 (2011), 52.

William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics* 21, 4 (1987), 163–169.

John Lucas, Doug Bowman, Jian Chen, and Chad Wingrave. 2005. *Design and evaluation of 3D multiple object selection techniques*. Technical Report. Virginia Polytechnic Institute and State University.

Hans-Gerd Maas and George Vosselman. 1999. Two algorithms for extracting building models from raw laser altimetry data. *ISPRS Journal of photogrammetry and remote sensing* 54, 2 (1999), 153–163.

Sven Oesau, Florent Lafarge, and Pierre Alliez. 2016. Planar shape detection and regularization in tandem. *Computer Graphics Forum* 35, 1 (2016), 203–215.

Tahir Rabbani and Frank Van Den Heuvel. 2005. Efficient hough transform for automatic detection of cylinders in point clouds. *Isprs Wg Iii/3, Iii/4* 3 (2005), 60–65.

Szymon Rusinkiewicz and Marc Levoy. 2000. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, New York, USA, 343–352.

Radu Rusu and Steve Cousins. 2011. 3D is here: Point cloud library (PCL). In *IEEE International Conference on Robotics and Automation 2011 (ICRA 2011)*. IEEE, New York, 1–4.

Claus Scheiblauer. 2014. *Interactions with Gigantic Point Clouds*. Ph.D. Dissertation. Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria.

Claus Scheiblauer and Michael Wimmer. 2011. Out-of-core selection and editing of huge point clouds. *Computers & Graphics* 35, 2 (2011), 342–351.

Ruwen Schnabel, Roland Wahl, and Reinhard Klein. 2007a. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum* 26, 2 (June 2007), 214–226.

Ruwen Schnabel, Roland Wahl, and Reinhard Klein. 2007b. RANSAC Based Out-of-Core Point-Cloud Shape Detection for City-Modeling. *Schriftenreihe des DVW, Terrestrisches Laser-Scanning (TLS 2007)* 53 (2007), 8.

Markus Schütz. 2016. *Potree: Rendering Large Point Clouds in Web Browsers*. Master's thesis. Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria.

Robert F Tobler. 2011. The rkd-tree: An improved kd-tree for fast n-closest point queries in large point sets. In *Proceedings of Computer Graphics International*, Vol. 2011. 4.

Hough Paul VC. 1962. Method and means for recognizing complex patterns. US Patent 3,069,654.

Michael Wand, Alexander Berner, Martin Bokeloh, Arno Fleck, Mark Hoffmann, Philipp Jenke, Benjamin Maier, Dirk Staneker, and Andreas Schilling. 2007. Interactive Editing of Large Point Clouds. In *Eurographics Symposium on Point-Based Graphics*, M. Botsch, R. Pajarola, B. Chen, and M. Zwicker (Eds.). The Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 9.

K Wenzel, M Rothermel, D Fritsch, and N Haala. 2014. An out-of-core octree for massive point cloud processing. In *Proceedings of 1st Workshop on Processing Large Geospatial Data 2014*. 53.

Tim Weyrich, Mark Pauly, Richard Keiser, Simon Heinzle, Sascha Scandella, and Markus H Gross. 2004. Post-processing of Scanned 3D Surface Data. *SPBG* 4 (2004), 85–94.

Michael Wimmer and Claus Scheiblauer. 2006. Instant Points: Fast Rendering of Unprocessed Point Clouds. In *Symposium on Point-Based Graphics*, Mario Botsch, Baoquan Chen, Mark Pauly, and Matthias Zwicker (Eds.). The Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 9.

Lingyun Yu, Konstantinos Efstathiou, Petra Isenberg, and Tobias Isenberg. 2012. Efficient structure-aware selection techniques for 3D point cloud visualizations with 2DOF input. *IEEE TVCG* 18, 12 (2012), 2245–2254.