Surfaces for Point Clouds using Non-Uniform Grids on the GPU

Daniel Schiffner
Goethe Universität
Robert-Mayer-Strasse 10
60054, Frankfurt, Germany
schiffner@gdv.cs.uni-frankfurt.de

Claudia Stockhausen
Goethe Universität
Robert-Mayer-Strasse 10
60054, Frankfurt, Germany
stockhausen@gdv.cs.uni-frankfurt.de

Marcel Ritter
AHM, Universität Innsbruck
Technikerstrasse 21a
6020, Innsbruck, Austria
marcel.ritter@uibk.ac.at

ABSTRACT

Clustering data is a standard tool to reduce large data sets, such as scans from a LiDAR, enabling real-time rendering. Starting from a uniform grid, we redistribute points from and to neighboring cells. This redistribution is based on the properties of the uniform grid and thus the grid becomes implicitly curvilinear, which produces better matching representatives. Combining these with a polygonal surface reconstruction enables us to create interactive renderings of dense surface scans. Opposed to existing methods, our approach is running solely on the GPU and is able to use arbitrary data fields to influence the curvilinear grid. The surfaces are also generated on the GPU to avoid unnecessary data storage.
For evaluation, different data sets stemming from engineering and scanning applications were used and have been compared against typical CPU based reconstruction methods in terms of performance and quality. The proposed method turned out to reach interactivity for large sized point clouds, while being able to adapt to the point clouds geometry, especially when using non-uniform sampled data.

Keywords
Surface reconstruction, point clouds, clustering, curvilinear grids

1 INTRODUCTION

With the increasing use of laser light detection and ranging (LiDAR), applications easily generate several billions of points measurements [PMOK14] [OGW+13]. If semi-automated algorithms are to be applied, interactive rendering of such large data sets becomes important. However, such large amounts of geometry data do not fit into the graphic hardware's memory as they easily reach hundreds of giga-bytes. While out-of-core mechanisms are used, the generation of the needed information cannot be solely computed on the graphics card.

Our approach addresses one part of the overall problem by generating a representation that allows interactive rendering. We want to avoid as many pre-processing steps as possible, and thus use a clustering algorithm as our base method for data reduction. We leverage the final grid definition to modify the grid using per-cell information, resulting in a curvilinear representation. Following these steps, we are then able to use these curvilinear cells to produce better fitting surface-representations based on cell-only information. In this work, we enhance our previous definition and contribute the following:

- Refined and more precise definition of the curvilinear grid.
- Simple and robust surface normal estimation
- Different, dynamic reconstruction methods, ranging from cells to surfaces

After providing some background information, we gather related and previous work in section 2. The approach is described in-depth in section 3 followed by an evaluation and results in section 4. Here, the main results regarding to timing and visualization are presented and discussed. The article concludes in section 5, and closes with thoughts on future work in section 6.

2 RELATED AND PREVIOUS WORK

The application of vertex clustering recently has grown in interest due to its fast processing capabilities. Linear

methods, such as grid based clustering methods, are especially well suited for large data sets that may contain several million or even billion data points. By reducing the input set, such as presented by DeCoro [DT07] or Willmot [Wil11], the rendering of large data is possible again with a little overhead at the initial clustering phase. In the latter case, individual attributes of an input data set are stored separately to increase detail after reduction.

Promising results have also been achieved by Peng and Cao [PC12], as they are able to provide frame-to-frame coherence when applying their reduction method. Their approach is based on an edge collapse algorithm, which was presented by Garland and Heckbert [GH98]. They apply the computation of the quadric error metric in parallel and then decide where to reduce and restructure the output triangles.

The selection of individual level of details is also a crucial part and often includes offline processing methods. In [SK12], we used a parallel approach to dynamically update the current representation while retaining interactivity. This is done by first computing a raw estimate of the object that is being refined during processing. In our previous work [SRB14], we have used the *cluster move* paradigm to enhance clustering of unstructured point clouds. In Limper et al. [LJBA13], the so-called POP Buffer has been introduced. The authors make use of a simple clustering followed up by a sorting on the CPU. This allows fast LOD exchange, as solely VBOs are created that can also be instanced. None of the presented approaches makes use of the processing capabilities provided by modern GPUs.

A comparison of two clustering algorithms has been presented by Pauly [PGK02]. In this case, a hierarchical and an incremental clustering method are applied to reduce and refine point-set-surfaces[ABCO+01]. Both approaches showed good results regarding reduction quality and run-time performance. Clustering, especially in the context of SPH data sets, has been utilized by [FAW10] with a perspective grid. They include a hierarchy (octree) in the data organization and apply texture based volume rendering in front to back order of the perspective grid for drawing.

[PGK02] use a covariance technique in the point neighborhood to compute a reconstructed 'surface normal' and to measure a distance from a cluster point to the original surface. A similar method based on the same dyadic product, called the point distribution tensor, was introduced in our previous work [RB12]. However, the product also includes distance weighting functions and the analysis based on the tensor's Eigenvalues is different. Three scalar fields are derived from the second order tensor called linearity, planarity, and sphericity. These describe the geometric point neighborhood and are normalized between 0 and 1. If points are distributed on a straight line, linearity is high, and if points are distributed on a plane planarity is high, respectively. We pre-computed the planarity for some of the data sets used in the benchmarks and include it in the clustering process, such that variations in planarity are preserved and homogeneous planar regions are clustered.

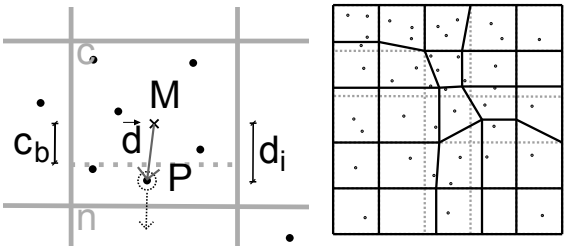


Figure 1: **Left:** Detailed view of the marked cell in Figure 2. This graphic illustrates how a point is "moved" from one cell to its neighbor below. A point P of the cell c is assigned to a different cell if the largest component d_i of the direction vector $\vec{d}$ from the cell center M to P is larger than a certain cell bound c_b which depends on parameters of cell c and the neighboring cell n. **Right:** Curvilinear grid after moving the points. Note that the curvilinear grid is not computed explicitly. It is indirectly defined by the points being assigned to the cells.

Besides the area of workstations the problem has also been addressed for mobile devices. Rodríguez et al. [RGM+12] presented a method for interaction with giga-sample-sized point clouds on mobile devices. The solution is based on a server-client framework, with a previous pre-processing step. In the pre-processing step, the data is partitioned as a kd-tree and reduced to a target point size. The points are reduced by merging points with their closest neighbor with compatible normals. For a further overview of 3D graphics on mobile devices we refer to Koskela et al. [KVA15]

3 OUR APPROACH

Our approach makes use of a simple vertex clustering that can be computed and generated on the graphics card. This initial clustering is used in a second iteration to resize individual cells. The available information hereby ranges from simple density distributions up to more complex scenarios, where planarity is being computed.

We enhance our previous definition [SRB14] by providing a more complex, but more powerful function to compute the cell boundaries. We also pass surface information gathered during the clustering-stages to later stages of the pipeline to allow dynamic reconstruction of surfaces. The resulting surfaces are only created within the graphics card, but can also be retrieved using transform feedback mechanics.

It is noteworthy, that the source data is not altered during the computation of the cell boundaries. The processed vertex is only passed to a different cell within the second iteration.

The presented approach works in multiple steps: *cluster*, *move*, *reduce*, and *reconstruct*. *Cluster* and *move* as well as *reduce* and *reconstruct* are pairwise related. For this reason, they will be introduced together in the following sections.

3.1 Cluster and Move

The *cluster* operation applies a classical vertex clustering, but we also accumulate information required for the *move* operation. The incoming points are mapped to a uniform grid. The grid might be warped via an affine transformation as shown in figure 2. The point position is converted to an cell index that is used in further computations.

The *move* operation then identifies whether a point needs to be placed in a neighboring cell. Based on the accumulated information, e.g the number of points inside a cell and the position, curvilinear cell bounds are derived. If the current point is located outside the curvilinear cell bound, it is emitted as being part of this neighbor.

The given definition of the cell boundaries is curvilinear, as a formula is given that modifies the actual borders. This is also illustrated in figure 1. In the following, we provide the formula given by [SRB14]:

$$\vec{d} = P - M \tag{1}$$

$$\Delta_i = \max_{j=1..3} \{|d_j|\} \tag{2}$$

$$\mathrm{w}(c,n) = \min\left(lb, \left(\frac{dens(c)}{dens(c)+dens(n)}\right)^{\gamma}\right) \tag{3}$$

$$c_b = w(c,n) \tag{4}$$

$$\Delta_i > c_b \begin{cases} true, & \textit{move } P \text{ to } n \\ false, & \text{skip} \end{cases}, \tag{5}$$

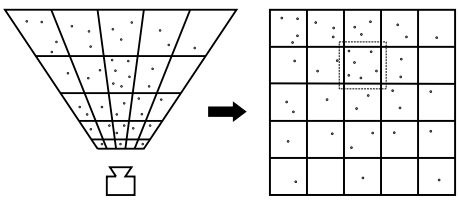


Figure 2: Points are transformed into a local coordinate system of the camera view frustum. Initial cells are defined by a uniform grid. The clustering algorithm operates in this coordinate system. The grid's geometry preserves more detail close to the camera and reduces detail far way.

with M the center point of the current cell c, P a point in c, i the index of the maximal component of vector $\vec{d}$, n the neighbor cell, lb a lower bound of the cell size of c, c_b the cell boundary in direction of the component i, and γ a non-linear scaling factor.

For a point P, its direction vector $\vec{d}$ from the cell's center is computed. The maximal absolute component of this vector is chosen, see Equation 1 and 2. Then, a weight dependent on the current cell c and its neighbor n, a lower bound and a non-linear scaling factor is computed, defining a new theoretical cell bound c_b, see Equation 3 and 4. If the maximal component is larger than this new cell bound, the point is assigned to the neighbor cell, see Equation 5.

The $dens(x)$ function operates on information available inside a cell x and returns a scalar value. For data sets having only point coordinates available, we use the number of points inside a cell as density. Any data available per point can be included in the density function. We also experimented using a pre-computed planarity field [RB12]. This field describes a local geometric property of the point cloud, influencing the *move* operation.

The computational complexities of the *cluster* and *move* operations scale with the size of the input data $\mathcal{O}(N)$. Each cell, identified by the index, is processed and data is accumulated per cell.

To apply this function to 3d objects, we enhance the definition by replacing them by

$$\Delta = P - M \tag{6}$$

$$f(c,n) = 2\frac{dens(c)}{dens(c)+dens(n)} \tag{7}$$

$$w(c,n) = clamp\left((f(c,n))^{\gamma}, lb, 1-lb\right) \tag{8}$$

$$idx_c \mathrel{+}= \sum_{i=0}^{3} (w(c,n_i) < \Delta_i) \cdot idx_{n_i} \tag{9}$$

The function expands the current relative weights to the range of $[0,1]$ and limits it to the resulting range of the lower bounds. The delta is used to relate the new boundaries with the current location of the inspected point. The resulting index of the current point (idx_c) is computed by the current index plus the possible offsets (idx_{n_i}). A Boolean value of false is "0", while a Boolean value of true is "1", like in C.

In either case, a piecewise-linear shift for each cell boundary is created. However, due to the fact, that each cell has its own relative weights, these boundaries are still curvilinear.

3.2 Reduce and Compute Normals

The *reduce* operation emits an representative for each cell that has been previously computed. Thus, the output is a reduced set of points. Any accumulated data can

 ISBN 978-80-86943-66-4

be emitted and visualized as well. As the single cells are iterated, the computational complexity is bound linearly with the number of cells $\mathcal{O}(C)$. After this reduction, the visualization of the reduced data set can be done using classical splatting techniques.

To reconstruct surfaces, we derive a surface normal along with the boundaries of the current cell. We are able to approximate the surface normals by using two different approaches. The first method leverages the point-distribution tensor, whereas the second uses the cell neighborhood. In the former case, the *reconstruction* operation computes the minor eigenvector and uses it as the surface normal.

The surface normal approximation based on the cells uses the density information gathered in the *move* operation. We hereby assume, that the surface normal will be oriented from a cell towards empty cells. Thus we iterate the neighboring cells, and add up directions, where the neighboring cells are empty. This yields a surface normal, which points outwards from the current cell. If all neighboring cells are empty, no normal can be derived (which is correct).

Independent of the used approach, the estimated normal vector has no preferred direction and has to be oriented. We therefore use the positive-z axis in eye space, favoring normal vectors that are facing towards the observer.

3.3 Dynamic Reconstruction

Based on the averaged cell position, the bounding box and the optional estimated normal vector, we create geometric representatives for each cell. We use three different geometric objects: boxes, oriented splats, and cell-filling-quads. One such geometric representative is generated per cell. The boxes can the created from the position and the bounding box, whereas the oriented splats and the cell-filling-quads require the estimated normal vector for correct vertex orientation.

The splats are given the radius based on the distance to the neighboring cells, which can easily be derived to the fixed relationship of the cells. Along with the estimated surface normal, a perspective correct splatting can be achieved.

In the case of the oriented surfaces, we use the surface normal, to map 4 vertices in the bounding box. The resulting positions are thus limited by the bounding box of the current cell.

All methods share, that the created geometry is created on the GPU, and no transfer or storage of this data is required. By leveraging geometry shaders, the generation of new primitives can be achieved efficiently, as the maximal number of primitives is limited by the grid resolution used during the vertex clustering stage. Thus, we can derive a maximal number of vertices and ensure interactive visualizations by limiting the grid size.

4 RESULTS

To create test results, we have implemented our approach with openGL using compute shader capabilities that are available since version 4.3. We did not use an openCL approach, as the data will be rendered directly after the processing. This way, we have direct control on the results of the cluster algorithm when altering the individual parameters. In the core specification, no floating point atomic operations are specified but can be added by using an extension from nVidia. When using other vendors, one could emulate this feature, by converting the float value to an integer. For further details, the reader may be referred to [CCSG12].

As our approach consists of a *cluster* and a *move* step, we can simply omit the latter to allow an evaluation of the overhead generated. Thus, this algorithm applies a basic clustering to the input data set. A top-down octree has been implemented using the CPU. Obviously, the octree will not be able to compete in terms of computation speed, but the reduced cells are used for a visual comparison. We opted to use an octree because the clustering methods presented in the background section either require heavy precomputations ([PGK02], [FAW10]) or use a hierarchical clustering ([DT07],[Wil11]). In the latter case, especially regarding the work of [Wil11], we do not have several attributes for our data, thus we cannot make use of the advantages of this algorithm.

While processing large data sets, one must take special considerations into account. One being the limitations of the used graphics card. The compute shader capabilities have several, graphics card dependent factors, such as maximal work group size or maximal buffer size. The latter is especially important for large data sets, as a streaming of individual data is necessary. The approach is able to compute partial solutions, as the grid can be constructed in a streaming fashion. In the case of the largest data set (refer to 1), the computation times partially reached a Windows specific timeout (TDR), where the driver is shutdown and restarted. We use a swap of the back buffer to circumvent such an timeout after each step. While not being optimal, as the graphics is busy swapping a buffer, it allows to keep the driver alive. Similar problems are known when using expensive shaders of any type, CUDA and OpenCL applications. Additionally, an out-of-core mechanism is required, if the used data exceeds the maximal buffer size of the GPU. However, this is not taken into account yet.

4.1 Time Measurement Results

Based on our application, several benchmarks have been conducted. They vary in terms of input size, grid size and used graphics card. In general, a test has been repeated 10 times and the median time values are given.

Timings are reported in milliseconds. Each test was run with varying input parameters, i.e. the object and the grid size. These benchmarks were executed on 3 different PC's, running on Windows 7 and Linux. The results are listed in table 1. The first system (1) uses an i5-670 and a nVidia GeForce 680 GTX. The second (2) uses an i5-333 with a GTX 780 Ti. The last machine (3) consists of an i5-3450 and a nVidia GeForce GTX 460 with 1GB RAM. All systems operate on a MS-Windows platform.

The results in table 1 are split into two sections, the grid based operations and the visualization. The latter uses a standard view, to be comparable among the different tests. The number of reduced elements is given in the first row of each data set. Note, that the test system 3 is running at its maximum capabilities, due to the available memory, and could not process the last data set trivially. For this reason, we have excluded it from the benchmark, as an out-of-core strategy needs to be used.

The individual timings indicate an overhead due to the additional processing step of our approach. We have an increase of approximately 100% if the *move* operation is used. Note, however, that our compute shader has not been optimized and leaves room for further improvements. Currently, the *move* operation does a complete reclustering of the source data instead of using the results of the first stage.

A visualization of the presented timings using a different grid size can be seen in Figure 3. Interestingly, the computation times reduce, as the grid increases in size. This is mainly due to the fact, that an atomic counter is required, once an element is emitted into a cell. The smaller the overall cell count, the more atomic writes into an individual cell are required. This results in more sequential writes in this case.

As one can easily see in Figure 3, the GeForce 460 GTX is not able to compete with the newer generations. This may be due to the limited memory as well as being the first generation supporting compute shader capabilities.

To further emphasize the influence of each individual processing step, i.e. *cluster*, *move* and *reduce*, we created a visualization of these steps in figure 4. The *move* operation uses approximately the same time as the *clustering*. The influence of the neighborhood size, i.e. zero, one or three, is negligible. The reduction in this case uses the point-tensor reconstruction, which may be opted out for an rougher and faster approximation.

4.2 Visual Results

The visualization technique draws either oriented splats using elliptical weighted average (ewa) splatting [ZPvBG02], a cell representation based on boxes or a surface approximation, as described in section 3.3. In figure 5 some results generated with our surface

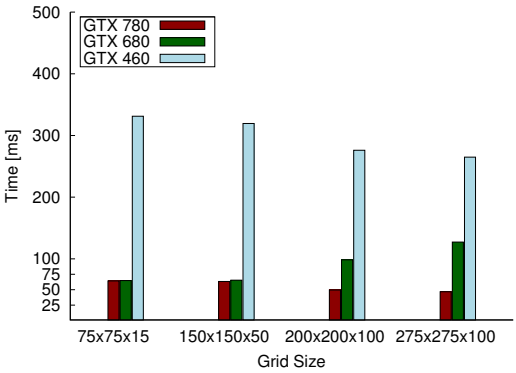


Figure 3: The influence of the grid size on the overall performance of our algorithm. The GeForce 780 GTX outperforms the other graphics cards. The 460 GTX is not able to compete with any of the newer versions. We used the GasTank data set for computation.

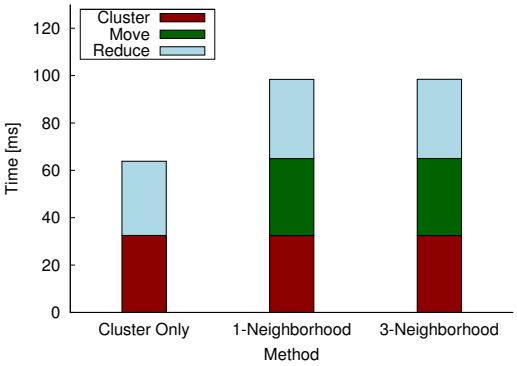


Figure 4: Timing values for each processing step of our algorithm. The reported values are the median of all runs. For this values, the medium sized object with a grid size of 200x200x100 has been used. Measurements were taken on the Test System 1.

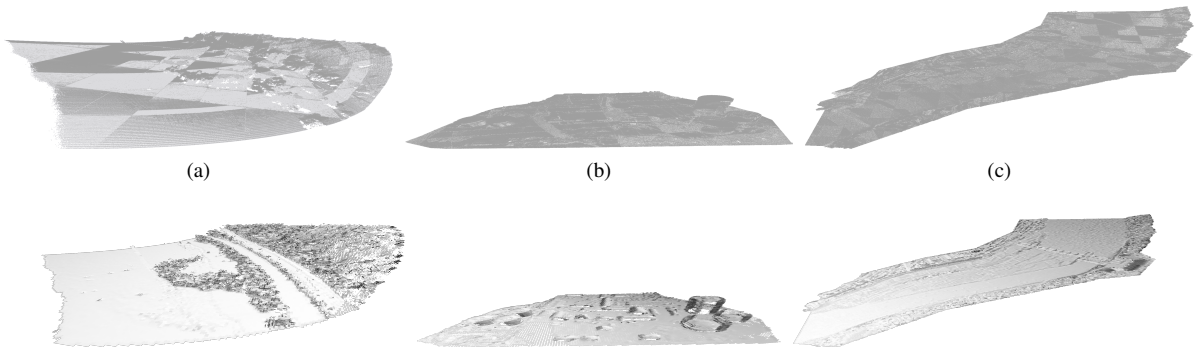
reconstruction method are shown. We used the prior mentioned data sets to apply a clustering. The number of generated primitives is significantly lower than the input count, thus achieving (in general) much higher frame rates (refer to table 1).

The different methods of visualizations are generated using a geometry shader using information stemming from the bounding boxes and the normal estimation. In case of the boxes representation, the normal estimation step can be skipped. The other methods require a surface normal for orientation. An overview of the results using the Small River data set can be seen in figure 6.

As mentioned before, we use an octree implementation to show differences in restruction quality of our approach. We selected an octree level, which approximately generates the same number of primitives as our approach, as seen in figure 7. Our algorithm is able to create similar results, but is much more flexible and can be solely computed on the GPU, which is not trivial to achieve with the octree approach.

Model	Sys	Our	Cluster	CPU	Original	Splat	Boxes	Surfaces
SmallRiver					# 2.075.993	# 75.173	# 902.076	# 150.346
	1	62.8	46.7	873.0	7.6	2.3	10.2	3.8
	2	63.1	46.6	180.0	4.1	2.8	11.0	5.0
	3	68.3	52.0	790.0	7.7	46.6	77.9	48.1
GasTanks					# 11.133.482	# 67.993	# 815.916	# 135.986
	1	65.7	49.3	4753.2	30.3	2.2	9.5	3.8
	2	64.3	43.5	940.0	18.3	3.1	11.0	5.0
	3	318.7	267.5	4110.0	37.0	46.3	79.7	48.4
RiverDam					# 26.212.555	# 79.099	# 949.188	# 158.198
	1	123.9	89.2	11006.6	71.8	2.2	9.3	3.8
	2	97.9	65.1	2234.9	45.6	2.8	13.7	5.1

Table 1: Benchmark results of our GPU algorithm, a basic cluster approach, an CPU and an octree implementation. All shown tests have been performed with a grid size of 150x150x50. Timings are reported in ms, the numbers in the first row of each data set denote the number of elements used for visualization. Note that the test system 3 was not able to perform the clustering due to hardware limitations, which may be the reason for the bad timings in the visualization, despite the low number of vertices used.



(a) (b) (c)

(d) SmallRiver (e) GasTank (f) RiverDam

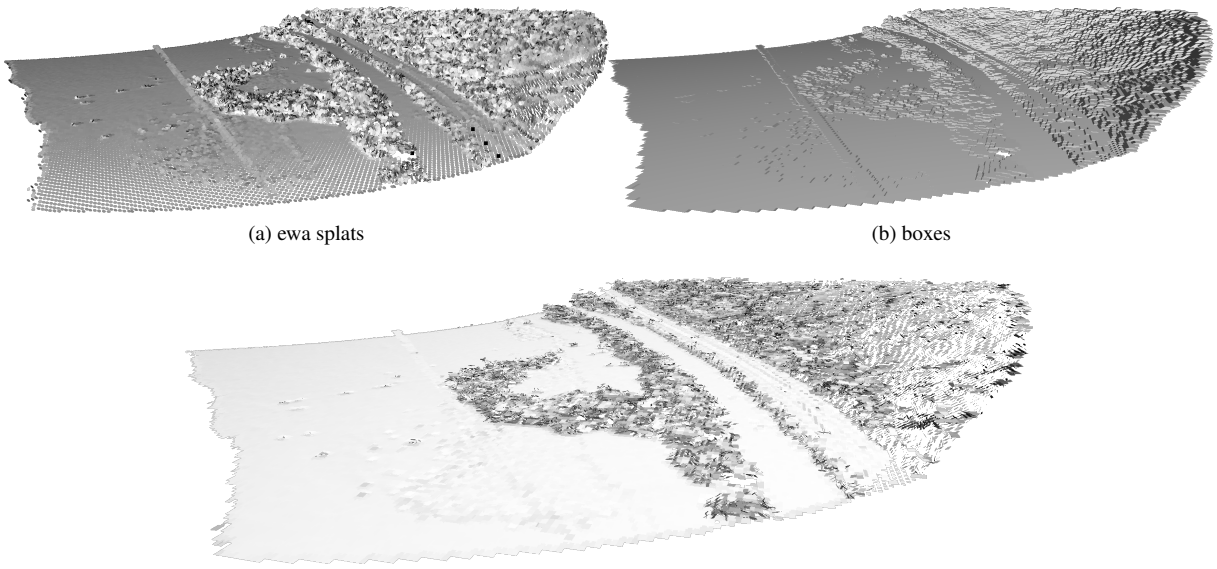
Figure 5: **Top row:** The used three point cloud data sets used for testing. Different sizes and different geometrical distributions are benchmarked. The points in the LiDAR data sets are mainly distributed on surfaces with small volumetric regions in vegetation and water. The point density varies relatively little over the whole data set.
Bottom row: Visual results of the surface reconstruction using clustering for the different data sets illuminated using a headlight. The grid size is set to $150 \times 150 \times 50$.

Both methods used for normal estimation provide stable results. The raw approximation based on the reduced cell set has a significant lower processing time (about 40%), but the tensor method provides much smoother and higher quality normals. A comparison of the achieved results can be seen in figure 8. The holes in both images method appear, as the cell-filling quads are aligned on the surface normal.

We lastly tested the influence of the *move* operator and the number of neighbors used, show in figure 9. As seen in the timing measurements, the number of neighbors does not significantly changes the processing time of the *move* operator. The quality of the surfaces increases with the larger number of neighbors used in the curvilinear grid computation. In the case of 3-neighbors, the discontinuity in the scan can better be represented due to the better matching cells, than in the other two cases. While the gaps between the individual quads disappear, there are still misplaced patches. These arise do to missing neighborhood information in these regions. Thus, the normal computation fails to derive an unambiguous direction, yielding these misplaced patches. By increasing the neighborhood, these cases can be avoided but with the cost of more expensive computations.

5 CONCLUSION

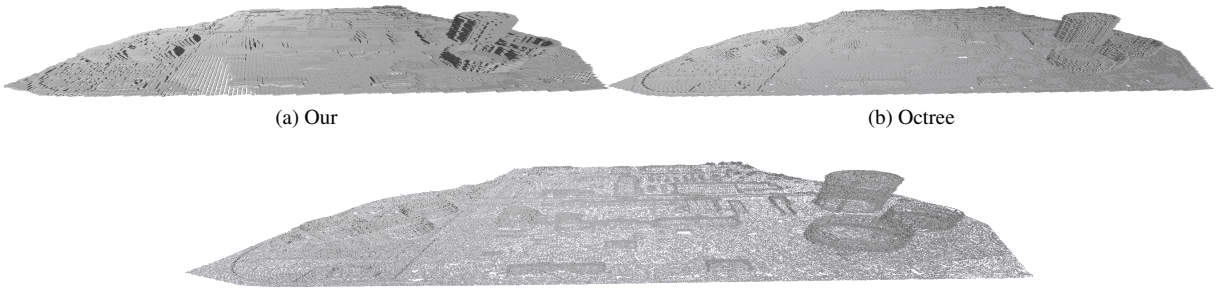
We have presented a new approach to reconstruct surfaces by leveraging a non-linear clustering to arbitrary objects. We are able to use multiple information from the current geometry and are not limited any preprocessing. The applied reduction is made selectively, due to a restructuring of individual cells. Currently, our data sets are point based and do not incorporate connectivity information. However, an extension to triangles or polygons can easily be achieved, as shown by other researchers ([PC12] [Wil11]).



(a) ewa splats (b) boxes

(c) surface

Figure 6: The small river data set geometrically reconstructed on the GPU. A geometric object is created per cell based on its information, i.e. center point, density, cell-size, and normal vector. The top row illustrated ewa splats and boxes. The bottom figure shows result of our surface reconstruction method. The model is illuminated using a headlight.



(a) Our (b) Octree

(c) Octree high detail

Figure 7: Comparison of our approach and an octree reconstruction. The achieved quality is very similar, while our approach is created solely on the GPU and does not require any precomputations. The high detail representation is included to show the desired result. Note that the high detail representation fails to create a closed surface, as the leaf-level of the octree does not fill the resulting gaps.

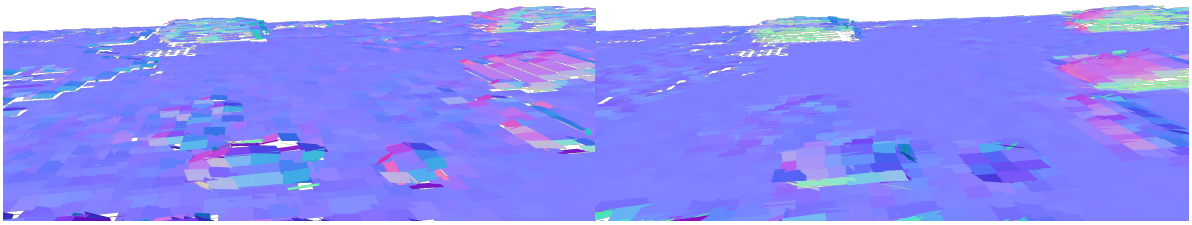
The computation times of the *cluster move* operations are interactive for medium sized point clouds and has a good performance with large data sets. Our implementation has not been optimized and leaves room for further enhancements.

We have shown the differences between classical clustering and our curvilinear implementation. Due to the dynamic cells, details in an object are better preserved. This increases the quality during a rendering and represents the topology of the basic object more accurately, while still reducing the input data set.

The reconstruction of a surface based on the geometric properties of an individual cell allows different visualizations without the need of re-computation. We use accumulated information of the cluster cells to create simple per-cell geometric elements to approximate a surface. We demonstrated three different elements on a LiDAR data set, allowing to reconstruct a polygonal digital surface model in real-time.

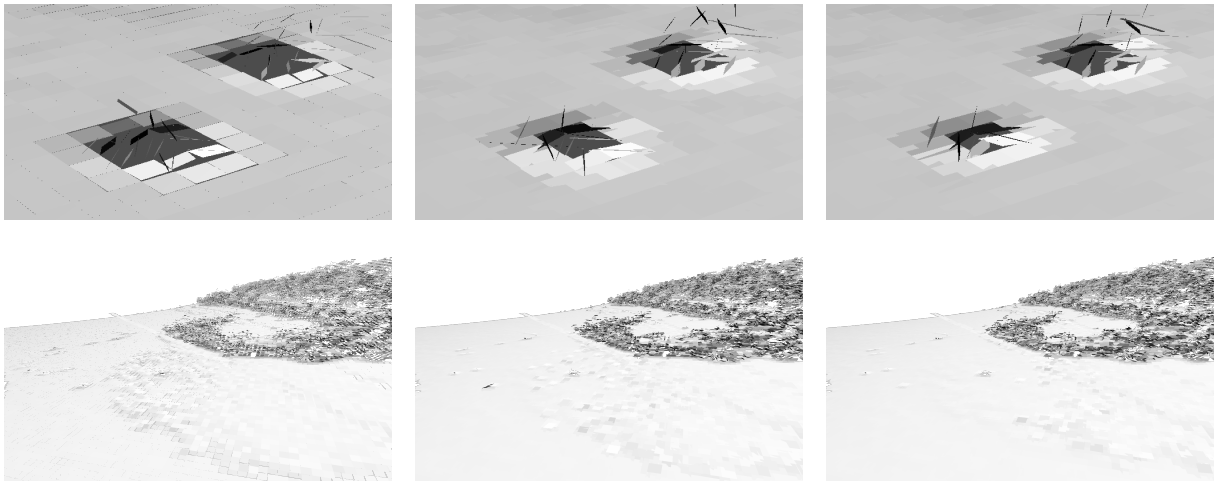
6 FUTURE WORK

The high performance of the compute shader drives us to further investigate streaming of big data. This includes a fast discard of unnecessary data, as well as selective reloading of individual fragments of a rendered object. Especially, the efficiency of the *move* allows repetitive execution (more iterations) or more complex grid modifications. The current restrictions to direct neighbors can be removed with the cost of additional lookups during the *move* operation. We assume that this will further improve the quality of the clustering.



(a) Normals using empty cells (b) Normals using point tensors

Figure 8: Comparison of our normal computations based on the reduced cells. The surface normals based on the tensor are more stable and produce more smooth approximations, but has significantly higher computational complexity. As one can see, the planar regions show less jitter in the tensor case. Both images were created using the cell-filling quads for visualization.



(d) Cluster only (e) Single neighbor (f) Three neighbors

Figure 9: Comparison of the different neighborhood computations. On the left, the *move* operation has been skipped, i.e. cluster only), where as the middle image shows the single neighbor result. In the right picture, 3 neighbors were taken into account, resulting in less gaps in the visualization. The outliers arise due to the surface normal estimation, which is performed in an online manner. The lower row shows an overview of the data set.

As the cell-filling quads are an interesting starting point, to polygonalize a surface, we think, that curved surfaces, such as Bézier or NURBS patches, do better match the underlying geometry. However, further testing needs to be done, how these patches can be utilized without any larger pre-processing of the data set. Furthermore, these patches could be controlled in terms of level of detail by a tessellation-shader, further enhancing the dynamic reconstruction.

An interesting topic is the dynamic construction of reusable information by defining a maximal footprint of GPU memory to use. In this case, LoD algorithms need to be applied, to ensure correct selection and eviction of primitives for display.

We will improve the quality of the tensor computation, especially by investigating better points of reference than the center point of a cluster cell. We intent to represent more information gathered in the tensor in the geometrical reconstruction. We will enable the reconstruction of linear structures besides planar ones. We want to improve the cell-filling-quads generation to better represent partially smooth closed surfaces.

Acknowledgements. This work was supported by the Austrian Ministry of Science BMWF as part of the Konjunkturpaket II of the Focal Point Scientific Computing at the University of Innsbruck and as part of the UniInfrastrukturprogramm of the Research Platform Scientific Computing at the University of Innsbruck and funded by the Austrian Science Fund (FWF) DK+ project Computational Interdisciplinary Modeling, W1227-N16. We like to thank Frank Steinbacher [ahm15] to provide the LiDAR data sets.

7 REFERENCES

[ABCO+01] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Cláudio T. Silva. Point Set Surfaces. In Thomas Ertl, Kenneth I. Joy, and Amitabh Varshney, editors, *IEEE Visualization*. IEEE Computer Society, 2001.

[ahm15] 2015 (accessed March 9, 2015). http://ahm.co.at.

[CCSG12] Cyril Crassin and Simon Green. Octree-Based Sparse Voxelization Using the GPU Hardware Rasterizer. In Patrick Cozzi and Christophe Riccio, editors, *OpenGL Insights*, pages 303–319. CRC Press, July 2012. `http://www.openglinsights.com/`.

[DT07] Christopher DeCoro and Natalya Tatarchuk. Real-time Mesh Simplification Using the GPU. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 161–166, New York, NY, USA, 2007. ACM.

[FAW10] Roland Fraedrich, Stefan Auer, and Rüdiger Westermann. Efficient high-quality volume rendering of sph data. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1533–1540, 2010.

[GH98] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization*, pages 263–269, 1998.

[KVA15] Timo Koskela and Jarkko Vatjus-Anttila. Optimization techniques for 3d graphics deployment on mobile devices. *3D Research*, 6(1):1–27, 2015.

[LJBA13] M. Limper, Y. Jung, J. Behr, and M. Alexa. The pop buffer: Rapid progressive clustering by geometry quantization. *Computer Graphics Forum*, 32(7):197–206, 2013.

[OGW+13] Johannes Otepka, Sajid Ghuffar, Christoph Waldhauser, Ronald Hochreiter, and Norbert Pfeifer. Georeferenced Point Clouds: A Survey of Features and Point Cloud Management. *ISPRS International Journal of Geo-Information*, 2(4):1038–1065, 2013.

[PC12] Chao Peng and Yong Cao. A GPU-based Approach for Massive Model Rendering with Frame-to-Frame Coherence. *Comp. Graph. Forum*, 31(2pt2):393–402, May 2012.

[PGK02] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient Simplification of Point-sampled Surfaces. In *Proceedings of the Conference on Visualization '02*, VIS '02, pages 163–170, Washington, DC, USA, 2002. IEEE Computer Society.

[PMOK14] N. Pfeifer, G. Mandlburger, J. Otepka, and W. Karel. OPALS - A framework for Airborne Laser Scanning data analysis. *Computers, Environment and Urban Systems*, 45(0):125 – 136, 2014.

[RB12] Marcel Ritter and Werner Benger. Reconstructing Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field. *Journal of WSCG*, 20(3):223–230, 2012.

[RGM+12] Marcos Balsa Rodriguez, Enrico Gobbetti, Fabio Marton, Ruggero Pintus, Giovanni Pintore, and Alex Tinti. Interactive exploration of gigantic point clouds on mobile devices. In *VAST*, pages 57–64, 2012.

[SK12] Daniel Schiffner and Detlef Krömker. Parallel treecut-manipulation for interactive level of detail selection. In *20th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, volume 20, 2012.

[SRB14] Daniel Schiffner, Marcel Ritter, and Werner Benger. Using curvilinear grids to redistribute cluster cells for large point clouds. *Proceedings of SIGRAD 2014*, pages 9–16, 2014.

[Wil11] Andrew Willmott. Rapid Simplification of Multi-Attribute Meshes. In *High-Performance Graphics 2011*, August 2011.

[ZPvBG02] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus H. Gross. Ewa splatting. *IEEE Trans. Vis. Comput. Graph.*, 8(3):223–238, 2002.