SIGRAD 2014
M. Obaid, D. Sjölie, E. Sintorn and M. Fjeld (Editors)

Using Curvilinear Grids to Redistribute Cluster Cells for Large Point Clouds

D. Schiffner[1], M. Ritter [2,3], D. Steinhauser [3] and W. Benger [2]

[1]Professur für Graphische Datenverarbeitung, Goethe Universität Frankfurt, Germany
[2]AirborneHydroMapping GmbH, Technikerstr 21a, Innsbruck, Austria
[3]Universität Innsbruck, Technikerstr 13/25, Innsbruck, Austria

Abstract

Clustering data is a standard tool to reduce large data sets enabling real-time rendering. When applying a grid based clustering, one cell of a chosen grid becomes the representative for a cluster cell. Starting from a uniform grid in a projective coordinate system, we investigate a redistribution of points from and to neighboring cells. By utilizing this redistribution, the grid becomes implicitly curvilinear, adapting to the point cloud's inhomogeneous geometry. Additionally to pure point locations, we enabled data fields to influence the clustering behaviour. The algorithm was implemented as a CPU and a GPU code. The GPU implementation uses GLSL compute shaders for fast evaluation and directly manipulates the data on the graphics hardware, which reduces memory transfers. Data sets stemming from engineering and astrophysical applications were used for benchmarking. Different parameters dependent on the geometric properties were investigated and performance was measured. The method turned out to reach interactivity for medium sized point clouds and still good performance for large point clouds. The grid based approach is fast, while being able to adapt to the point cloud geometry.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Hierarchy and geometry transformations

1. Introduction

Large point cloud data sets are produced by observations and simulations. Today, laser light detection and ranging (LiDAR) applications easily generate several billions of points measurements [PMOK14, OGW*13], similar amounts of particle based data are generated by state-of-the-art astrophysical simulations, i.e. by smooth particle hydrodynamic codes [SWJ*05, Spr05]. Interactive rendering of data becomes important, i.e., when semi-automated algorithms are applied. The classification of LiDAR data, e.g., requires a quality check and 'hand-made' corrections done by users. Here, interactive response and rendering is important for an efficient work-flow. Such large amounts of geometry data do not fit into the graphic hardware's (GPU's) memory as they easily reach hundreds of giga-bytes. Thus, data has to be prepared to support out-of-core rendering, for example in spatially sorted data fragments. But, more geometry data can be loaded onto the GPU's memory than the GPU can display at interactive frame rates.

Here, our approach aims at geometry reduction on the GPU to still achieve interactive frame rates per out-of-core data fragment. We want to avoid any additional data preprocessing, but can enhance the reduction when using pregenerated information. We cluster the incoming vertices to reduce the amount of data being displayed by creating an implicit curvilinear grid originating from an affine transformed uniform grid. The cluster process consists of two steps: a grid cluster operation and a move operation. The cluster operation is simple as it operates on an initial uniform grid. The move operation uses accumulated information from the first step and processes indices only. This allows a fine grained control over individual cells and their number of contained vertices enabling to manipulate the details of the rendered data while not needing to preserve it for further processing. In the context of out-of-core rendering and big data sets, this becomes ever increasing in importance.

Our method aims to be:

fast. Utilization of the GPU and preparing the clusters via GLSL compute shaders to reduce memory transfers and unleash parallelism available in standard workstations.

simple. Data organization is linear and no hierarchical data structure is required. It is directly used for rendering.

flexible. Besides the vertices, additional data fields such as scalar, vector, or tensor fields, may be taken into account for clustering. The method can be combined with a coarse Level of Detail technique on out-of-core data fragments.

versatile. Since the grid is being adjusted to the point clouds's geometry, the method is applicable to different kinds of data. The approach makes no assumption on how the point cloud originated or if it represents a specific kind of geometry. The points may or may not describe lines, surfaces, volumes, or other geometrical distributions, available at many time steps or just at one.

We chose application motivated data sets to do benchmarks. We study different parameters of the approach and the influence of data set properties on the performance and the applicability.

After having provided some background information, we gather related and previous work in section 2. The approach is described in-depth in section 3. Data sets stemming from LiDAR and from astrophysical particle simulations, see section 4, are the basis for benchmarks in section 5. Here, the main results regarding to timing and visualization are presented and discussed. The article is concluded in section 6, and closes with thoughts on future work in section 7.

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 method 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 could be done by computing a raw estimate of the object that is being iteratively refined.

An comparison of two clustering algorithms has been presented by [PGK02]. In this case, a hierarchical and an incremental clustering method are applied to reduce point-set-surfaces [ABCO*01], where cells were iteratively refined. 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.

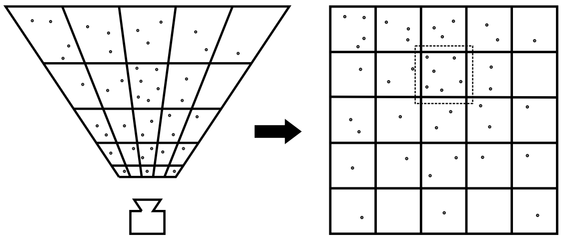


Figure 1: Points are transformed into a local coordinate system of the camera view frustum. Initial cells are defined by a uniform grid. The clustering algorithm operates in this coordinate system. The grid's geometry preserves more detail close to the camera and reduces detail far way.

[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 precomputed 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.

3. Our Approach

The main idea behind our approach is to re-size individual cells based on their internal data. The less points contribute to an individual cell, the better the quality once a reduction is applied. This applies, as long as the representative is being computed using the values taken from a single cell.

The most basic scenario for shrinking a cell is that it contains more points than their neighbors. This can be achieved by reducing the cell extents. Note, that this reshaping does not alter the actual data but is only used internally to derive a new cell. More elaborate results can be achieved, by using

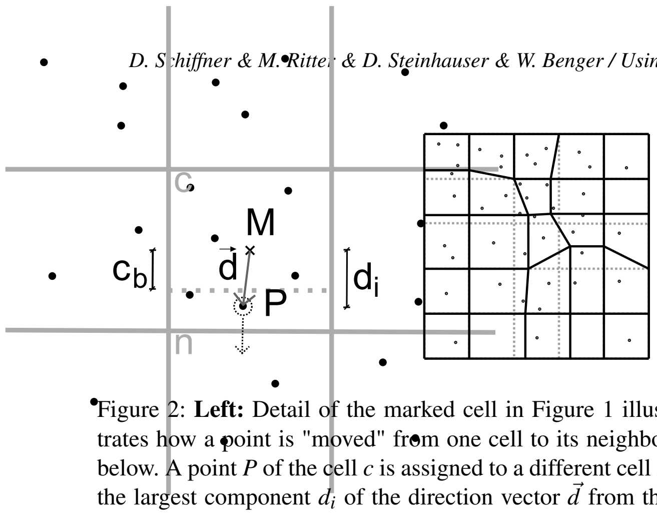


Figure 2: **Left:** Detail of the marked cell in Figure 1 illustrates how a point is "moved" from one cell to its neighbor below. A point P of the cell c is assigned to a different cell if the largest component d_i of the direction vector $\vec{d}$ from the cell center M to P is larger than a certain 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, but indirectly defined by the points assigned to each cell.

geometric properties, such as curvature or tensor data, that may have been computed in advance.

Also, this approach can be combined with classical warping techniques. In these cases, a non-uniform transformation is applied prior to the clustering, see Figure 1.

3.1. Overview

We apply a 3 step method to create the reduced input set: *cluster*, *move*, and *reduce*. The first step applies a classical clustering, but we also accumulate information needed for the second step. The incoming vertices are mapped to a grid that may can be warped via an affine transformation. The resulting position is converted to an index that is used for further computations.

The second, i.e. the *move*, identifies, whether a data point needs to be placed in a neighboring cell. It uses the accumulated information from step 1 and local information of the current data point to compute new, temporary, cell bounds. If the point is located outside the temporary cell bound it is *moved* to its neighbor. This renders the cell bounds curvilinear, as the actual shape is being altered. Figure 2 illustrates the involved geometrical objects of the method, which formally is described by:

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

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

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

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

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

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 assuring its minimal size, 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 from the cell's center is computed. Then, the maximal absolute component of this vector is chosen and compared to the according component of

Both, step 1 and step 2 scale with the size of the input data $\mathcal{O}(N)$. Each cell, identified by the index, is processed and the according data is accumulated to compute a representative data point.

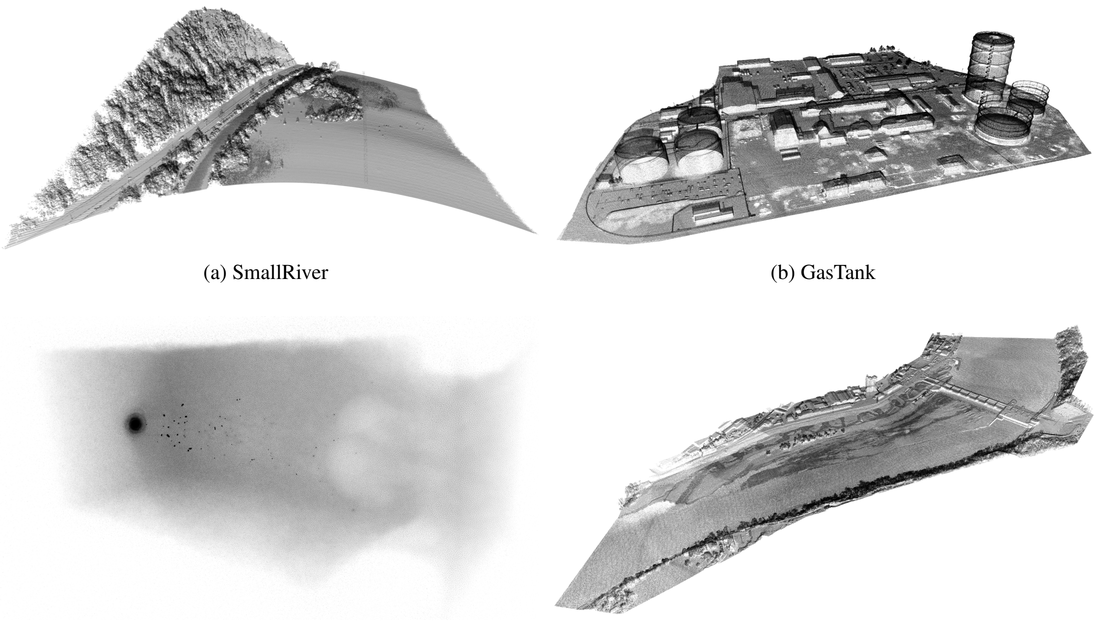
The last step simply reduces the input point set by emitting the previously averaged cell position. More sophisticated methods such as median or a quadratic error minimization could be utilized to derive the representative. As the single cells are iterated in this case, the time complexity is bound linearly with the number of cells $\mathcal{O}(C)$. The final output is a reduced point set, that can be visualized. To allow further displaying of additional data, the accumulated data of the *cluster* or *move* steps can be emitted as well.

3.2. Computation Details

The processing flow of our method can be described as follows. On each call, the input position from the raw data set is being warped by a given projection matrix. This may be identity, if no warping should be applied. The resulting position is in normalized device coordinates and is matched to the underlying grid by multiplying it with the grid size. Finally, a grid index is derived by performing a 3D to 1D mapping. From this point, the individual shaders diverge and different operations are performed.

In the *cluster* operation, a scalar value is read from an additional buffer that is aligned with the input positions. This value represents the individual weight of an input point and is atomically added to an internal counter. We also store a maximum value to allow visualizations regarding the overall weight.

In case of the *move* operation, each cell is compared to their immediate neighbors. The previously summarized counter is used to extract the total weight of a cell. Weights that are less than the derived neighbor's weight are not taken into account and the processing is aborted. Otherwise, the new cell bounds for the current active cell are computed by applying formulas (4) and (3). We use the internal counter from the first step as the density(c) and set used a default parameter set for all tests ($lb = 0.1, \gamma = 1.0$). If the current position exceeds the new cell bound, the current point is emitted to that neighboring cell used in the computations.


(a) SmallRiver (b) GasTank

(c) FormingStars (d) RiverDam

Figure 3: Four point cloud data sets were 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. The SPH simulation of forming stars is fully volumetric and has small regions of much higher point densities.

Name	Nr. of Points	Scalar Field
SmallRiver	2.075.993	Planarity
GasTanks	11.133.482	Planarity
FormingStars	16.250.000	Type / Density
RiverDam	26.212.555	Planarity

Table 1: Data set sizes used for the benchmarks.

4. Data Sets

We use data sets stemming from LiDAR measurements and an astrophysical particle simulation to test our algorithm, see Figure 3. Table 1 lists the data sets, its sizes, and an available scalar field on the points.

LiDAR: For the LiDAR data three airborne scans with increasing complexity were chosen. The data was captured with a green laser system by Riegl, the VQ820g, specialized for bathymetric scanning. The laser system has an especially high pulse rate of up to 520 kHz and a wide footprint optimized for capturing shallow water regions. The *RiverDam* data set was enriched by additional sonar measurements and, thus, includes ground echos of the deeper (>3m) river sections, besides the shallow water regions of the fish ladder (<3m) [DBS*13]. Such high density bathymetric laser scans are used for hydraulic engineering, planing water related building structures, and environmental engineering. Grids for numerical hydraulic computations can be generated, e.g., for flooding simulations or morphological studies. To generate such grids from a point cloud several processing steps are required. Points are filtered and geo-referenced. Then, they are classified into, at least, the two classes: water and non-water points. Next, the water surface is extracted and non-water points are corrected to eliminate the effect of the water's refraction. Especially, the step of classification needs control and corrections by human users to support automatic algorithms. For all the LiDAR data sets the planarity was pre-computed, an attribute given per point, describing a geometrical property of the surrounding neighborhood [RB12]. It was computed via a the point distribution tensor and describes how closely points are distributed towards a fitting plane in the neighborhood. The radius of the neighborhood was set to 2 meters.

Astrophysics: The *FormingStars* data set represents one time step of a combined N-Body/Hydrodynamic simulation of a galaxy undergoing ram-pressure stripping [SHKS12]. Such simulations are performed in order to understand the evolution of galaxies in dense environments in the universe. In galaxy clusters, the largest gravitationally bound structures in the universe, galaxies move in their mutual gravitational field. Besides the galaxies and dark-matter, such clusters consist of a very hot and thin gas, the intra-cluster medium (ICM). The galaxies are encountering this gas and feel its ram pressure, nonetheless it is very thin. This induces enhanced star formation within the galaxy at first, and leads to the stripping of the inter-stellar medium (ISM), the gas within a galaxy, reservoir for forming new stars. As a consequence, star formation in the galaxy ceases, but stars can be formed from stripped gas in the wake of the galaxy. The mass distribution of different components in GADGET-2 [Spr05] (gas(type 0), dark-matter(1), old stars(2), bulge stars(3), newly formed stars(4)) is discretized and sampled using a Monte Carlo method. Except gas, all other types of matter are then modeled as a collision-less fluid, interacting only via gravity. To solve the resulting N-Body problem, a tree code is used (e.g. [BH86]). The hydrodynamic equations for the gaseous component are solved via SPH (smoothed particle hydrodynamics [Mon92]). Initially, the density estimate of each particle is calculated using a kernel interpolation technique. Consequently, the momentum and thermal energy equation can be integrated in time, the continuity equation is implicitly fulfilled.

The points of the LiDAR data sets reside mostly on surfaces, such as measured ground or building structures. Only a few points captured in vegetation and water regions represent volumes. However, in the star forming simulation the points describe a volume. We want our algorithm to perform well in all cases and want to investigate its behaviour. All data sets still fit into 1GB of GPU memory, but only the smaller ones can be displayed at interactive frame rates.

5. 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 is going to be rendered directly after the processing. This way, we can directly control the outcome 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 two steps, we can simply omit the second one (and the additional computations) to allow an evaluation of the overhead generated by our additional *move* operation. Thus, this algorithm applies a basic clustering to the input data set.

A CPU implementation has been realized for sake of completeness. Obviously, the CPU variant will not be able to compete with the GPU implementation.

As stated before, we want to avoid any pre-computations, e.g. computation of tensors or connectivity, on the available data sets. The algorithm is able to perform a reduction without planarity information, but can produce better results with them.

5.1. Timing

Based on our applications, 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 mean 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 2. The first machine (1) consists of an i5-3450 and a nVidia GeForce GTX 460 with 1GB RAM. The second system (2) uses an i5-670 and a nVidia GeForce 680 GTX. The last configuration (3) contains an Intel Xeon-X5650 and a nVidia Quadro 5000. (1) and (2) operate on a MS-Windows platform while (3) runs a Linux system.

Model	Sys	Our[ms]	Cluster[ms]	CPU[ms]
SmallRiver	1	68.9	49.7	700.0
	2	14.6	10.2	831.0
	3	93.9	52.0	879.0
GasTanks	1	298.1	239.5	3780.0
	2	65.4	34.8	4445.4
	3	480.0	256.5	4758.5
FormingStars	1	648.8	479.9	5751.0
	2	129.8	88.6	6858.3
	3	749.0	434.4	7146.2
RiverDam	1	950.3	671.5	8670.0
	2	206.7	146.7	10292.0
	3	1228.6	719.9	11062.7

Table 2: Benchmark results of our GPU algorithm, a basic cluster approach and a CPU implementation. All shown tests have been performed with a grid size of 75x75x15. This grid was chosen due to the planar point distribution.

The individual timings indicate an overhead due to the additional processing step of our approach. Yet, we only have an increase of roughly 50% despite the additional computations performed in the *move* operation. Note that our compute shader has not been optimized and leaves room for further improvements. A visualization of the presented timings using a different grid size can be seen in Figure 4.

The influence of the grid size is in all computation steps very small. This is due to the fact that the individual steps mostly depend on the data input size, while only the last step scales with the size of the grid. As one can see in Figure 5, the GeForce 680 outperforms the older graphics cards.

5.2. Visual Results

The visualization technique in the OpenGL demo simply draws equally sized non-transparent splats. Color is con-

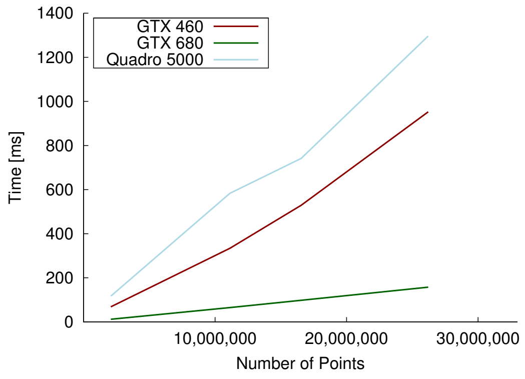


Figure 4: Timing values generated by processing each object repeatedly. The reported values are the mean of all runs. For all objects, a grid size of 200x200x100 has been used.

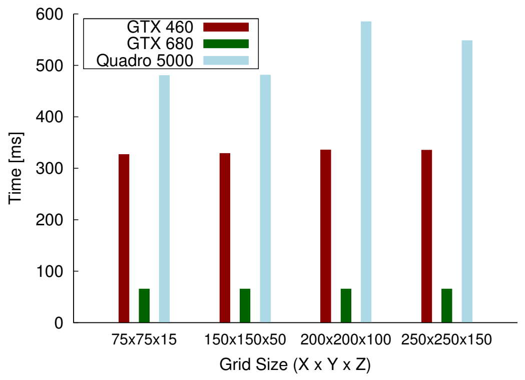


Figure 5: The influence of the grid size on the overall performance of our algorithm. The GeForce 680 GTX outperforms the other graphics cards. The Quadro, despite its larger memory, is not able to compete with the GeForce 460 GTX. We used the GasTank data set for computation.

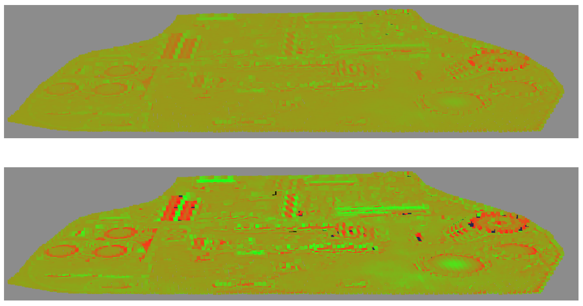


Figure 8: GasTank data set visualized clustered on a perspective grid. **Top:** *move* operation based on cell densities only. **Bottom:** *move* operation including the scalar field planarity which was computed in a pre-processing step. Smaller cells are created in regions of low planarity (e.g. edges) and, thus, preserving more detail. Dense cells are created in regions of homogeneous planar regions, were less detailed information is necessary for a good visual representation. Geometric features of the point cloud are enhanced, when taking the planarity into account.

trolled by a scalar value via a red to green color map. As presented in section 5.1, the impact of the additional *move*-step is acceptable, as the computation times are within interactive response times. The following Figures show several images that were created with both the curvilinear and a classic cluster algorithm with different grid sizes. The color map either illustrates changes based on the relative movement from the cells or the cluster cell density.

In Figure 6 some results generated with our method are shown. We used the prior mentioned data sets to apply a clustering. The colors indicate the density of the represented cell. The more red-ish the color, the more data points have been collected in this cell.

Especially with larger grid sizes, the reduction quality is increasing. In Figure 7, the cell density of each step is used for the color mapping. After application of the *move* operation, the global average is reduced, which results in the red color, as the same maximum is used for the mapping. The lower image visualizes the differences regarding the additional *move* operation. The curvilinear grid matches the underlying source more closely, as can be seen via the cluttered splats at the top right of the image.

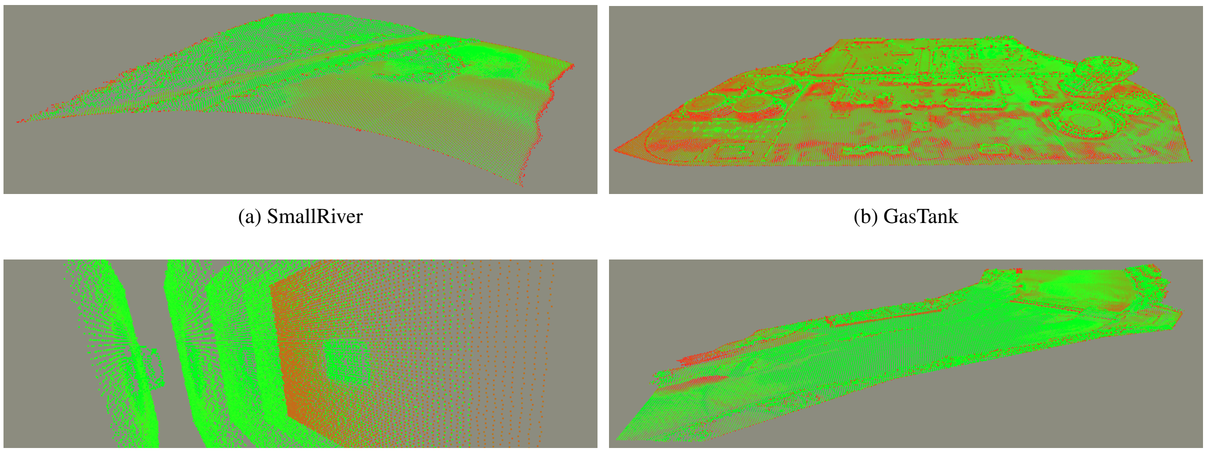
By introducing precomputed information, our algorithm can perform even better. As one can see in Figure 8, regions where edges are present are better fitted as smaller cells are used. This is indicated by the more distinct color values present in the individual cells, e.g. it the lower right of the image.

6. Conclusion

We have presented a new approach to apply a non-linear clustering to arbitrary objects. We are able to use multiple information from the current geometry and are not limited to scalar field properties. 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]).

The computation times of the *move* operation has been shown to be 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. For example, the calculation of cluster indices is performed in both the *cluster* and the *move* operation, which is not necessary.

We have shown the differences between classical cluster-


(a) SmallRiver

(b) GasTank

(c) FormingStars

(d) RiverDam

Figure 6: Visual results of the clustering for the different data sets. Color represents the cell density. The number of points per cell is illustrated by a green to red color map going from many (red) to one (green) point. Grid size varies from $150 \times 150 \times 25$ to $300 \times 300 \times 100$ in (a), (b), and (d), which yield good results for reduced overview visualizations. In (c) the grid resolution in z-direction was reduced to 5 slices allowing to see inside the volume. When inspecting the leftmost slice one can see how the representing points are pulled toward the high point density region of the galaxy, thus emphasizing a region of interest. The simple non-transparent splat rendering prevents better insights into the volume.

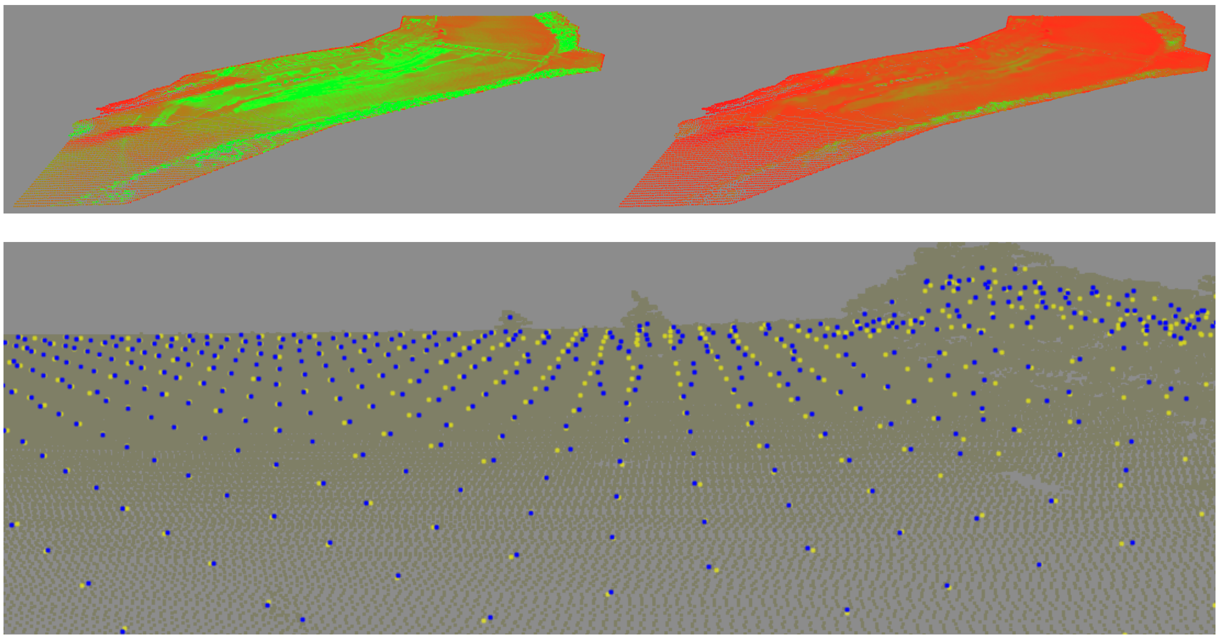


Figure 7: The differences due to the application of the proposed method. In the first picture of the top row, the green-ish regions indicate cells with high density. These are reduced by rescaling the cell sizes, which results in a more even distribution, as seen on the right top. The image below shows a detailed view, where and how the *move* operation modifies the positions of the resulting cells. The yellow cells are created by the clustering while blue ones are the result with the additional *move* operation. Note that the latter produces a splat at the tree in the top of the image.

ing and our curvilinear implementation. Due to the dynamic cells, details in an object are more likely to be preserved. This preservation of features during a rendering increases the quality and topology of the basic object, while still reducing the input data set. Thus, we have made another step towards interactive rendering of large, unprocessed data sets.

7. 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. We intent to use several reconstruction methods to enable the visualization of closed surfaces as well as available geometric properties, such as the point distribution tensor or the planarity. This will allow an identification of interesting regions within the large scale object. Tensor analysis may also be computed on the fly on the GPU.

The visualization can be enhanced by displaying the individual cell sizes. This way, a user could visually control, whether the implicitly generated curvilinear grid matches the expectations. Also, the information within a cluster cell could be visualized showing the influence of the available parameters to the effectively computed grid.

We also want to investigate, whether we could use the fast approximation to create a fingerprinting of these large data sets. To compare large data sets for equality, the accumulated information could be used instead of the raw data. However, it remains to be shown, if the generated data is unique enough for a clear identification.

8. Acknowledgments

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 [ahm] to provide the LiDAR data sets.

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point Set Surfaces. In *IEEE Visualization* (2001), Ertl T., Joy K. I., Varshney A., (Eds.), IEEE Computer Society.

[ahm] http://ahm.co.at.

[BH86] BARNES J., HUT P.: A hierarchical 0 (N log iV) force-calculation algorithm. *Nature* (1986).

[CCSG12] CYRIL CRASSIN, SIMON GREEN: Octree-Based Sparse Voxelization Using the GPU Hardware Rasterizer. In *OpenGL Insights*, Cozzi P., Riccio C., (Eds.). CRC Press, July 2012, pp. 303–319. `http://www.openglinsights.com/`.

[DBS*13] DOBLER W., BARAN R., STEINBACHER F., RITTER M., NIEDERWIESER M., BENGER W., AUFLEGER M.: Die Zukunft der Gewässervermessung: Die Verknüpfung moderner und klassischer Ansätze: Airborne Hydromapping und Fächerecholotvermessung entlang der Rheins bei Rheinfelden. *Wasser-Wirtschaft 9* (2013), 18–25.

[DT07] DECORO C., TATARCHUK N.: Real-time Mesh Simplification Using the GPU. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 161–166.

[FAW10] FRAEDRICH R., AUER S., WESTERMANN R.: Efficient High-Quality Volume Rendering of SPH Data. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2010) 16*, 6 (November-December 2010), to appear.

[GH98] GARLAND M., HECKBERT P. S.: Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization* (1998), pp. 263–269.

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Annual review of astronomy and astrophys. 30* (1992), 543–574.

[OGW*13] OTEPKA J., GHUFFAR S., WALDHAUSER C., HOCHREITER R., PFEIFER N.: Georeferenced Point Clouds: A Survey of Features and Point Cloud Management. *ISPRS International Journal of Geo-Information 2*, 4 (2013), 1038–1065.

[PC12] PENG C., CAO Y.: A GPU-based Approach for Massive Model Rendering with Frame-to-Frame Coherence. *Comp. Graph. Forum 31*, 2pt2 (May 2012), 393–402.

[PGK02] PAULY M., GROSS M., KOBBELT L. P.: Efficient Simplification of Point-sampled Surfaces. In *Proceedings of the Conference on Visualization '02* (Washington, DC, USA, 2002), VIS '02, IEEE Computer Society, pp. 163–170.

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

[RB12] RITTER M., BENGER W.: Reconstructing Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field. *Journal of WSCG 20*, 3 (2012), 223–230.

[SHKS12] STEINHAUSER D., HAIDER M., KAPFERER W., SCHINDLER S.: Galaxies undergoing ram-pressure stripping: the influence of the bulge on morphology and star formation rate. *Astronomy & Astrophysics 544* (July 2012), A54.

[SK12] SCHIFFNER D., KRÖMKER D.: Parallel treecut-manipulation for interactive level of detail selection. In *20th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision* (2012), vol. 20.

[Spr05] SPRINGEL V.: The cosmological simulation code gadget-2. *Monthly Notices of the Royal Astronomical Society 364*, 4 (Dec. 2005), 1105–1134.

[SWJ*05] SPRINGEL V., WHITE S. D. M., JENKINS A., FRENK C. S., YOSHIDA N., GAO L., NAVARRO J., THACKER R., CROTON D., HELLY J., PEACOCK J. A., COLE S., THOMAS P., COUCHMAN H., EVRARD A., COLBERG J., PEARCE F.: Simulating the Joint Evolution of Quasars, Galaxies and their Large-scale Distribution. *Nature* (2005).

[Wil11] WILLMOTT A.: Rapid Simplification of Multi-Attribute Meshes. In *High-Performance Graphics 2011* (August 2011).