The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLVIII-2/W1-2022
7th International Workshop LowCost 3D – Sensors, Algorithms, Applications, 15–16 December 2022, Würzburg, Germany

COMPUTING ARBITRARILY LARGE MESHES WITH LEVEL-OF-DETAIL SUPPORT FOR CESIUM 3D TILES

Malte Hillmann[1], Felix Igelbrink[1], Thomas Wiemann[2*]

[1]Knowledge Based Systems Group, Osnabrück University, 49080 Osnabrück Germany, (mhillmann, felix.igelbrink)@uos.de
[2]Plan-based Robot Control, DFKI Niedersachsen, 49080 Osnabrück Germany, thomas.wiemann@dfki.de

Commission II

KEY WORDS: 3D Surface Reconstruction, Mesh Simplification, Level-of-Detail Rendering, Web-based Visualization

ABSTRACT:

In this paper we present an approach to compute arbitrarily sized meshes of large scale environments. The meshes can be reconstructed from laser-scanned point clouds or existing high-resolution meshes. The algorithm automatically builds a level of detail hierarchy in the Cesium 3D Tiles format using an octree partition. The main contribution of this paper is a method that ensures that the generated meshes for each level-of-detail stage are computed in a consistent manner to minimize visual artifacts between different detail levels during rendering. Furthermore, both the reconstruction and simplification algorithm are designed to constrain the memory consumption, which enables to process even very large data sets on consumer-grade hardware. The export into the Cesium 3D Tiles format allows to render such large meshes efficiently in all web-based viewers that support this format. In our experiments we evaluate the method on different datasets and assess the visual quality during the rendering process and analyze the memory footprint as well as the runtime behaviour.

1. INTRODUCTION

3D point clouds of large scale environments are usually too large to be rendered on consumer hardware. WebGL-based viewers such as Potree (Schütz et al., 2016) or Cesium (Cesium GS Inc., 2022a) enable to interactively stream and visualize such very large datasets on low-cost hardware. They achieve this by converting the raw point cloud data into a multi-resolution spatial hierarchy consisting of many independent chunks using an octree partition scheme. In Cesium, this concept in generalized further in the 3D Tiles format to support arbitrary spatial hierarchies (Cesium GS Inc., 2022b). Generating such level of detail (LOD) representations from point clouds is rather easy, as point clouds do not encode any surface information and thus allow parts covering the same region to be streamed and displayed in a convenient way using an additive subsampling scheme. In LOD rendering, multiple parts of an object with different resolutions are commonly visible at the same time because the higher resolution data is still pending or visible chunks are much further away from the camera as others. Thus, all of these parts have to be displayed alongside each other in a visually appealing way with consistent borders to minimize artifacts. This is difficult to realize for meshes. Hence, they are seldom used in the context of rendering of large scale hierarchical data, even though more recent versions of the 3D Tiles format also include support for 3D meshes. Instead, multi-resolution-meshes are mostly restricted to self-containing objects, such as entire buildings, where an entire low-resolution mesh is just replaced by a higher resolution version, if the object is sufficiently close to the camera. For 3D point clouds captured by high-resolution 3D laser scanners, the data is usually not available as multiple segmented objects. So visualization of meshes based on point clouds, the consistent behaviour at the chunk borders has to be taken into account during surface reconstruction.

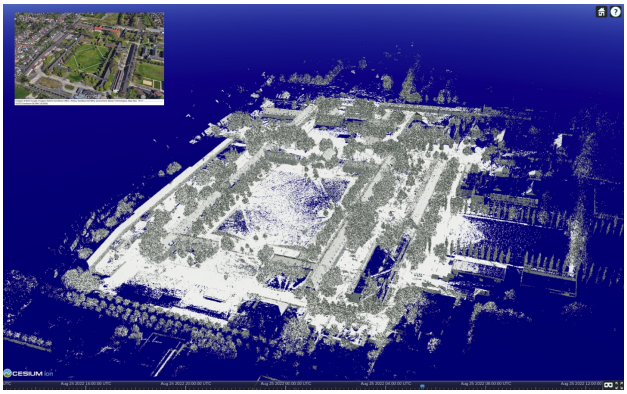

Figure 1. A reconstruction of the Jacobs University Campus in Bremen, Germany with 1.2 billion triangles. The inlaid image shows the corresponding area in Google Maps. With our approach, it can be displayed on any hardware via the Cesium Viewer. The generated LOD structure allows efficient and fast rendering of only the visible sections, which are loaded at the required resolution depending on the virtual camera position.

In this paper, we present an approach to generate large scale multi-resolution meshes suitable for LOD-rendering using Cesium 3D Tiles. Our meshes are generated in a way which allows display multiple LODs side-by-side without any visual artifacts. The tile data required for rendering in Cesium-based Viewers can be computed on consumer-grade hardware due to a constant memory footprint. The resulting representations can be deployed via a web server and displayed in any supported web-based viewer. All algorithms discussed in this paper are fully integrated in our mesh processing library LVR2 (Wiemann et al., 2012) and are freely available under BSD 3-clause licence[1].

* Corresponding author

[1] https://www.las-vegas.uni-osnabrueck.de

2. RELATED WORK

As the amount of points in large 3D scanning projects can grow arbitrarily large, usually the memory available on the GPUs of consumer-grade hardware is not sufficiently large to load or let alone to display the data in full resolution. Over the past years, several approaches have been proposed to render very large point clouds. Examples are, for instance, (Schütz et al., 2016, Hobu Inc., 2022, Seufert et al., 2020, Schütz et al., 2020). They all rely on hierarchical structures like octrees to subdivide the points into spatial chunks with increasing detail in lower levels. The point cloud chunks can be displayed consistently by additively combining all levels from the current one up to the root of the hierarchy. More details can be loaded as required, e.g., using frustum culling and prefetching techniques, keeping the memory requirements low. This simple but efficient rendering technique cannot easily be extended to meshes. Among these formats, the Cesium 3D Tiles (Cesium GS Inc., 2022b) specification does support mesh data in principle. However, automatic subdivision of meshes into the necessary multi-resolution structure is not yet available for large meshes. As meshes, in contrast to point clouds, cannot generally be subsampled due to geometric and topological constraints, more sophisticated simplification schemes are required.

Simplification of self-contained meshes is a well understood topic in computer graphics (Botsch et al., 2010). An existing fine-grained mesh is usually simplified in-core by removing vertices and collapsing edges while trying to preserve as much of the geometric surface information as possible, controlled by an error metric such as quadrics (Garland and Heckbert, 1997) or normal deviations (Melax, 1998). These approaches have also been extended to out-of-core processing, enabling the simplification of very large meshes (Lindstrom, 2003, Cignoni et al., 2004). Typically, multiple versions of the same mesh are generated in applications and exchanged dynamically during rendering depending on their distance to the camera. As our meshes result from an automatic reconstruction of large laser-scanned environments, the required segmentation into separate consistent objects is not available. Instead, the reconstruction typically results in one large mesh possibly with several (small) disconnected clusters in regions with low point density. Thus, it is preferable to reconstruct and simplify such meshes in spatial chunks - similar to the point clouds above. This simplification algorithm must be able to generate adequate representations of the geometry for multiple hierarchy levels with varying detail from the finest at the bottom all the way up to the coarsest LOD at the root of the hierarchy. In order to guarantee a geometrically consistent mesh at any point during rendering, the resulting mesh chunks must fit into all other level of the hierarchy to not result in visual or geometric artifacts, as neighbouring chunks of a mesh of the same resolution may not have been available yet. Examples for such artifacts are shown in Figures 6 and 7.

Due to the excessive amount of memory required to process large meshes as a whole, the subdivision algorithm has to process the chunks out-of-core to not exceed the available main memory, especially on low-cost hardware. In order to partition a mesh into a spatial hierarchy suitable for visualization, many methods have been developed. The simple ones ignore the explicit topology of a mesh entirely and treat it similar to a point cloud. In (Rusinkiewicz and Levoy, 2000), a bounding sphere hierarchy is generated from the initial mesh representation, where each sphere approximates the geometry and color of all vertices it contains. In contrast to that (Baert et al., 2013) use a sparse voxel grid. They build up the hierarchy by recursively combining neighbouring voxel grids in an octree. Using frustum culling, the required LOD is generated in both approaches by traversing the hierarchy down to a node appropriate for the current camera distance during rendering.

Other methods rely on existing high-resolution meshes. In (Cignoni et al., 2004), the hierarchy is built by recursively subdividing the space into a combination of tetrahedra according to the underlying mesh geometry. Each tetrahedron is then simplified independently under the constraint of shared bordering vertices to ensure a consistent geometry. The work of (Yoon et al., 2005) uses a hierarchy of spatially close mesh regions with similar triangle counts. Each leaf-region is then represented as a progressive mesh (Hoppe, 1996) and simplified using half-edge collapses. However, both algorithms assume that the initial mesh is already available. In our use-cases this is not the case, though. Instead, the raw data is only available in form of high-resolution point clouds, from which the meshes are reconstructed. In this process, it is not possible to build an optimized hierarchy directly for the respective mesh geometry. In this paper, we address this problem and provide an efficient reconstruction algorithm.

In this spirit, the work of (Poliarnyi, 2021) is similar to ours, as it also uses an octree and reconstructs the geometry for each node using a combination of total generalized variation minimization (TGV) and Marching Cubes (Lorensen and Cline, 1987). However, whereas our algorithm is able to handle any mesh or point cloud, their approach strongly relies on depth maps from stereo-based methods or generated from 3D terrestrial laser scans.

In (Wiemann et al., 2018) we proposed a mesh reconstruction method to handle arbitrary large input data. In this approach, the data is first partitioned into spatially uniform chunks of fixed size in an out-of-core fashion. Each chunk is then reconstructed independently of the others using the Marching Cubes algorithm. To allow later re-integration of the chunks into a consistent global mesh, every chunk has a small amount of overlap with neighbouring chunks. Using this overlapping information, the borders between neighbouring chunks can be corrected, resulting in an artifact-free final mesh. However, this algorithm only supports a single, global reconstruction resolution and thus cannot generate a multi-resolution hierarchy.

In this work we present an extension of this large-scale mesh reconstruction algorithm to support multiple LODs. It is able to automatically generate the required multi-resolution hierarchy from any existing mesh or point cloud data. Due to out-of-core processing, the amount of main memory required for conversion can be scaled and limited on demand, enabling such computations even on low-cost hardware.

3. METHODS

In this section, we present the algorithm to compute consistent meshes with different LODs. The core of the procedure is based on the large scale reconstruction approach present in (Wiemann et al., 2018). It is extended by a hierarchical LOD approach to support the requirements of the targeted 3D Tiles format. The main goal is to create consistent transitions between the different detail levels to minimize visual artifacts. Finally, we exploit the chunk-based structure of the reconstruction algorithm to limit the required memory, which enables our approach to

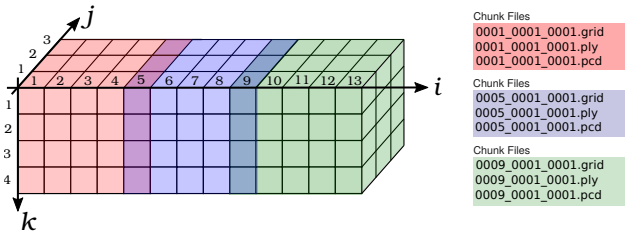

Figure 2. Alignment of chunks in the meta grid. Indices on the axes correspond the to global position of the voxel, the indices encoded in the file names encode the position of the chunks.

run on consumer-grade hardware. The components of our algorithm are detailed in the following sections.

3.1 Large Scale Reconstruction

The basis of our approach is the chunk-based reconstruction method presented in (Wiemann et al., 2018). The main idea is to generate a meta voxel grid that manages smaller sub-grids of fixed size that can be stored in local memory, similar to the voxel hashing strategy presented in (Nießner et al., 2013). These sub-grids (chunks) contain all the necessary data to compute a part of the global mesh, including the point cloud data, surface normals and meta-data, i.e., the position of the chunk within the global grid. While loading the point cloud, the single points are sorted into the corresponding chunks on the hard drive using memory mapped files to maximize performance. Each file encodes the position of the chunk within the global grid via meta indices that are encoded in the file name. With this information it is possible to map local cell indices in the chunks to global indices in the meta grid by adding the respective offsets, which is needed to identify neighbouring cells in the following processing steps.

After parsing the initial point cloud, we get a set of partially overlapping chunks as sketched in Figure 2. The partial overlap is necessary to guarantee consistent vertex configurations at the chunk borders without having to load adjacent cells from the meta grid to interpolate. With this method it is possible to reconstruct the local triangle mesh directly via Marching Cubes. The resulting chunks are directly built up to a consistent global mesh after removing redundant vertices in the overlapping areas in a post-processing step. This fine-grained initial mesh defines the lowest hierarchy level of our reconstruction. The aim of the following processing steps is to generate visually similar meshes with lower resolution that blend over smoothly when rendered.

3.2 Cesium 3D Tiles

Cesium 3D Tiles (Cesium GS Inc., 2022b) is an open specification for a file format that allows efficient rendering and processing of arbitrarily large datasets. This is achieved by storing the data in a multi-resolution hierarchy and loading only the required parts. The hierarchy itself is encoded as a tileset in a JSON file alongside the location, bounding volume, geometric error and potential children of each tile. The location points to the data of the tile via a path or url. The data may be a mesh, point cloud, combination thereof, or another tileset and must be stored in the formats specified by 3D Tiles. The bounding volume always holds a sub-volume that must contain the data and all children entirely. It may be larger than required, but setting the bounding volume too large will negatively impact rendering performance, since viewers use this information to determine visibility and required quality. The geometric error encodes this quality, giving a measurement of how far a tile deviates from the actual geometry that it tries to represent. In the case of LOD, as it is used in this paper, this is a metric of how much the simplification impacts the visual appearance compared to the original mesh. Finally, there are the children of the tile, which are in turn other tiles and their children, forming a tree-like structure to represent the hierarchy. The children are used to refine the parent tile by either adding detail to it or replacing it altogether with a higher quality version. This means that the deepest children, or leaves of the tree, contain the most amount of detail, whereas higher levels are coarser.

A viewer, like the one provided by Cesium (Cesium GS Inc., 2022a), can use the bounding volume to determine the visibility of a tile, its size on the screen, and distance to the camera. Based on these parameters, in combination with the geometric error, it can select an appropriate LOD to render. This selection can also take the current resource usage into account. The selected Tile is then loaded and displayed and later discarded once it is no longer visible. This process allows the viewer to always show as much detail as needed, while using as few resources as possible. Details on the format specification can be found in the official documentation provided by Cesium.

3.3 Hierarchical Level of Detail

To create a multi-resolution hierarchy, it is necessary to partition the mesh into small parts. One possible algorithm to accomplish this is sketched in Figure 3. The mesh is first split into segments based on the connectivity of the vertices and edges, where a segment is a part of the mesh that is not connected to any other parts. This division allows using existing boundaries instead of cutting the mesh. Cuts are problematic, since both sides need to visually fit together across all LODs without any gaps or other artifacts. The segmentation is problematic in two edge cases, namely if there are too many small segments or when some segments are too large. The quantity of small segments is reduced by combining them back together into larger meshes. This is accomplished with a simple chunk grid, as depicted on the left of Figure 3. The right part of Figure 3 shows the large segments being cut into chunks. Cutting cannot be avoided in this context, and chunk-shaped cuts are used to better align the meshes with the cuboid shape of bounding boxes.

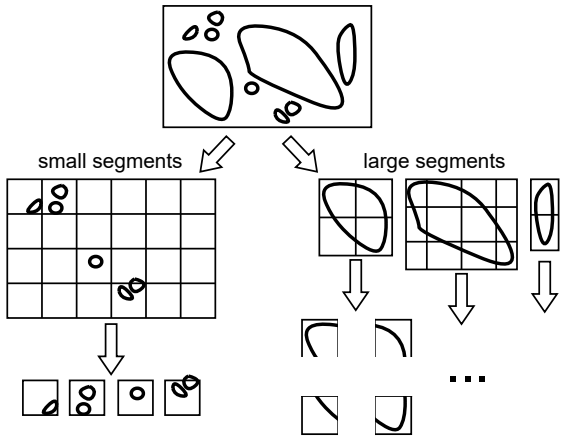

Figure 3. The process for dividing a mesh into parts, visualized on a 2D example.

In both cases the spatial extent of the chunks has to be known. Furthermore, a threshold value for the distinction between small and large segments has to be found. These values depend strongly on the size and density of the processed mesh, so they have to be configured by the user. For simplicity, a single parameter is used here, which serves as both the chunk size and for the distinction of segments: A segment whose longest axis is smaller than a chunk can not be subdivided and is therefore called a small segment. All segments larger than a chunk are called large segments. The generated chunks are then arranged in an octree to form the hierarchy. This is achieved by successively combining 2x2x2 adjacent chunks into a single super-chunk. An example of consistent chunks at the highest LOD within the hierarchy, that is used as basis to build-up this superstructure, is shown in Figure 4.

As mentioned above, the large-scale reconstruction used in this paper generates the mesh from pre-computed chunks. These can also be used as the finest LOD in the hierarchy, allowing for a very efficient integration of the hierarchy-generating algorithm. The merging of the chunks into a single mesh by the reconstruction and subsequent division into segments is skipped as the chunks are directly used in the octree.

The different resolutions in the hierarchy are generated by simplifying the super-chunks every time they are combined. This ensures that they become coarser, the higher they are in the hierarchy, while the bottommost layer contains the original mesh. This process uses incremental simplification to achieve the highest visual quality. The resulting LOD and its application is shown in Figure 5. It demonstrates the usefulness of LOD, since the difference in mesh quality is clearly visible when viewed up close, but unnoticeable at the distance for which they are intended.

Using incremental simplification automatically ensures that no new holes appear within the mesh or that the topology is destroyed. However, this cannot be checked for connections outside the mesh, which is especially relevant for the boundaries of the chunks. These chunks originally belonged to a single, connected mesh and have to be recombined for the hierarchy. However, a normal simplification might perform operations that result in gaps between the chunks, as can be seen in Fig. 6. Therefore, it is necessary to signal to the simplification that no changes may be made to the boundaries. This can be implemented by so-called ”features”. Features are areas that are highlighted by the caller of the simplification as special, like sharp edges or borders, and therefore are not allowed to be edited. In the case of the subdivision of large segments, features are placed along the cuts between the chunks. For the chunks generated directly by the reconstruction, an overlap is present to indicate the connection to the surrounding chunks. This information can be used by marking each vertex that is connected to other vertices outside the chunk boundary as a feature. The overlap itself is removed afterwards to avoid rendering artifacts.

The nature of the cuts and the design of the overlap ensure that every feature vertex in one chunk has a corresponding feature vertex with the same coordinates in an adjacent chunk. These vertices need to be combined when fusing chunks in the higher levels of the hierarchy. Finding such duplicates is implemented by a radius search around the vertices. Since the positions should be exactly identical in theory, the search could also be restricted by testing for equality. However, due to the limited accuracy of floats, especially when considering separately generated chunks, there may be deviations that prevent the detection exact equality due to numerical issues. Therefore, such a radius search is also performed with a very small bound to account for such inaccuracies. However, this may result in very close but not identical vertices being grouped together, potentially breaking neighboring faces. This is less of a problem here, as the face has to be very small for this to happen. For the actual merging, one of the vertices must be deleted and its neighborhood information transferred to the other vertex. This is a very complex operation, since the adjacent half-edges of a vertex always form a closed ring. A join would thus have to break and join two or more half-edges of these rings, for which there are no ready-made methods on classical half-edge meshes due to the complexity and number of edge cases. Instead, this is usually handled by also deleting all faces adjacent to the vertex to be deleted and then re-inserting them with new vertices. The necessary steps for repairing the rings when deleting and inserting faces are well researched and pre-implemented by most half-edge mesh libraries. For our purposes, we use the Polygon Mesh Processing Library (PMP)[2], as it has a simple interface and is therefore easy to integrate into existing software.

Since the chunks generated by the large scale reconstruction are used individually without being merged into a mesh, we discovered that a simple overlap may still result in minor inconsistencies between chunks, as demonstrated in the left image in Figure 7. These were previously handled during a merging step, which is skipped in this context. Replicating the necessary behavior without merging the chunks may not always be correct, so we decided to unify the truncated signed distances (TSDF) that are used for reconstruction instead. For that, the TSDFs of overlapping regions are averaged to ensure that they are identical in both chunks. This fusion relies on a weighted average where the weight is determined by the distance to the center of the corresponding chunk, so that central values receiving a higher weight than those near the edges. The result of this can be seen in the right image of Figure 7. It is worth noting that the removal of such gaps is necessary not just to avoid visual artifacts, but also to ensure that merge algorithm of chunk borders for higher LOD finds correct corresponding vertices.

3.4 Memory Bounding

When using large-scale meshes, the limits of available memory are quickly reached. In particular, when generating the hierarchy, copies of the mesh must be made in order to simplify them. For this simplification, the mesh is also required in half-edge mesh format, which requires a lot of memory. So the meshes are stored on disk and reloaded on demand. Despite this, memory overflow can occur if too many partial meshes are loaded at the same time. Our implementation allows to create and simplify super-chunks in parallel. Hence, memory usage scales with the number of CPU-cores. To restrict this, simple heuristics are used to limit the global memory consumption. The *minimal* setting only uses a single thread to keep the memory usage as low as possible at the cost of a slower runtime. The *moderate* mode only uses a limited, user-defined number of parallel processes (default: 25% of the available CPU cores). Moderate is the preferred option since minimal is too restrictive in most cases. In *unbounded* mode it is assumed that memory usage is not an issue. It allows to use all CPU-cores while also keeping the chunks loaded in memory to speed up the process as much as possible. These options can only be used when processing pre-computed chunks as produced by our large-scale

[2] `https://www.pmp-library.org`

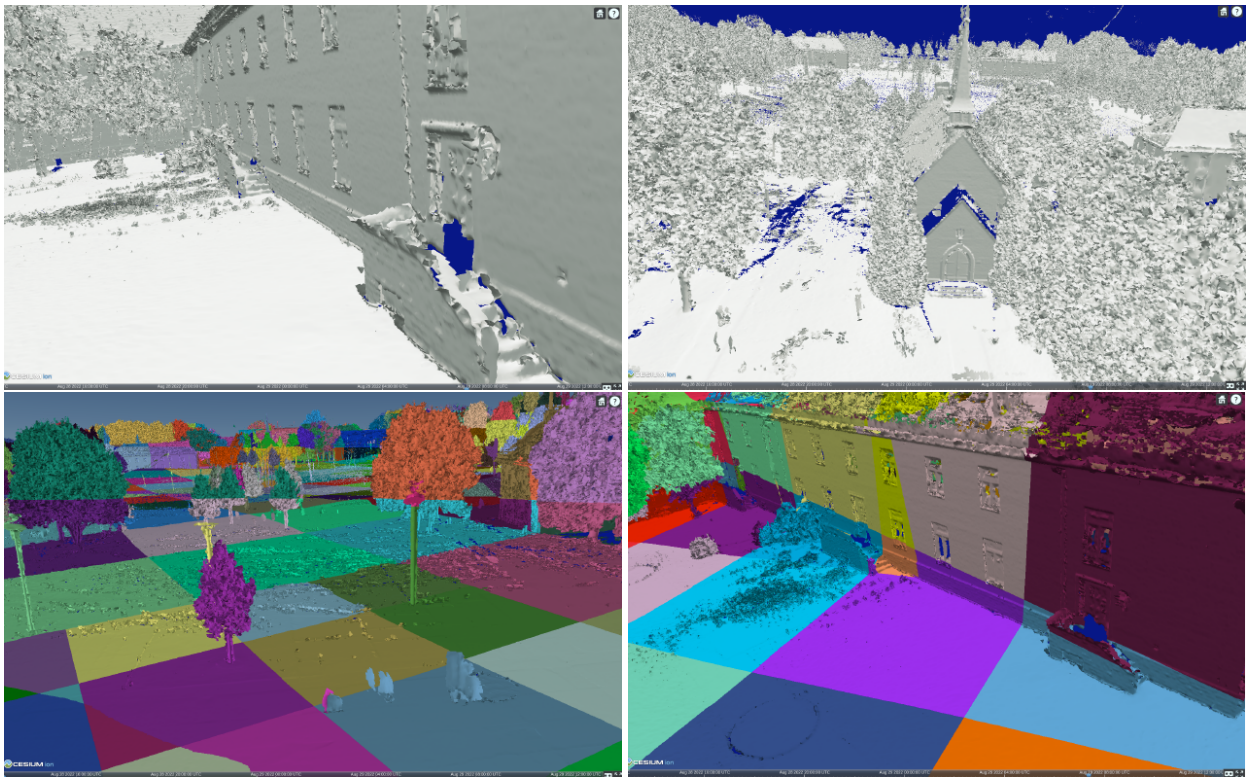

Figure 4. Visualization of chunks in the mesh data.

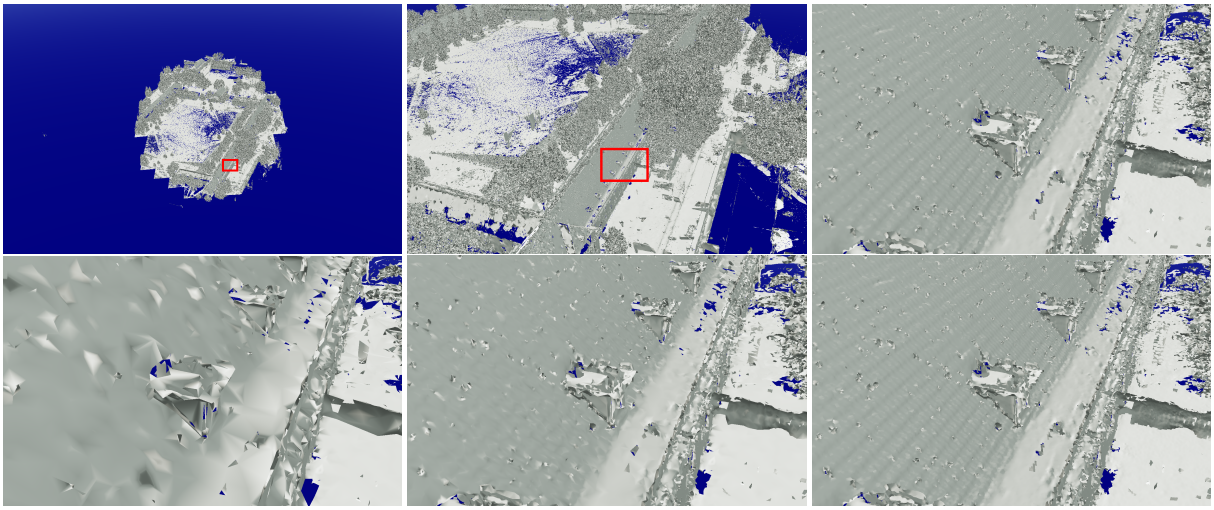

Figure 5. A scene viewed from different distances (top) with a detail view (bottom) of the LOD used for the highlighted area.

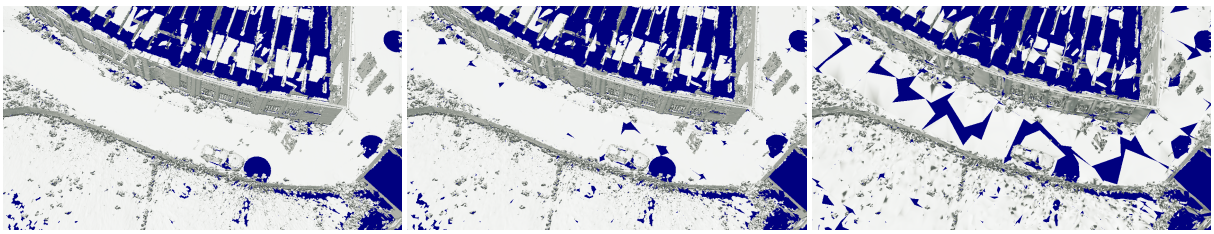

Figure 6. A chunked mesh with three LOD, from finest (left) to coarsest (right). The LOD-generation in this example did not handle chunk boundaries correctly, which results in easily visible gaps forming in the floor of the right image.

https://doi.org/10.5194/isprs-archives-XLVIII-2-W1-2022-109-2022

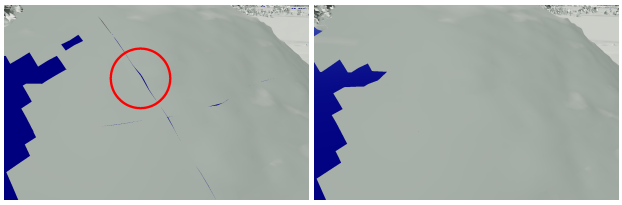

Figure 7. An example of the gaps that can arise between chunks with insufficient overlap during the reconstruction (left). In the image in the right the same overlap was used, but with unified TSDF-values.

reconstruction. If an existing mesh is used, it is always necessary to load it completely.

4. EXPERIMENTAL RESULTS

4.1 Qualitative Results and Visual Inspection

We evaluated our implementation on several freely available 3D data sets from the Robotic 3D Scanning Repository[3] and the RIEGL Pointcloud Samples[4]. The datasets were chosen to cover different practical aspects such as varying point density, reconstruction volume and noise level. To benchmark our approach, we reconstructed them with LVR2 at a voxel resolution of 5 cm, generated the chunking structure and tracked the memory consumption using KDE Heaptrack[5]. The resulting meshes were exported into the 3D Tiles format and deployed on a web server for visual inspection and benchmarking. The reconstructed meshes are shown in Figure 8. They consist of up to 1.2 billion triangles and cover areas of up to 851 760 square meters, cf. Table 1.

To assess the visual quality of the meshes during data download in the Cesium viewer, we used the CIE76 method according to EN ISO 3668. Although this measure is deprecated, we think it is a fast and good measure to assess the similarity of renderings, as we are mainly interested in the number of differing pixels. For that, we computed the number of differing pixels over time and compared it to the highest achievable resolution. The results of this experiment are shown in Figure 9. It can be seen that all data sets except Campus are loaded within a few seconds with good quality. For Campus, the loading time is significantly longer, as the dataset itself is more than ten times as large as the other ones. At that stage, an acceptable visual quality is achieved with a lower level of detail. In most cases this is preferable over missing data or lags in the virtual navigation within the scenes. The steps in the graphs can be explained with the way the data is loaded. Once a new chunk with higher detail is loaded, a complete area of the rendered scene is replaced with a more detailed model. Hence, the visual quality improves and stays constant until the next detail level is loaded. This behaviour properly reflects the way the data is organized and loaded in our approach.

4.2 Influence of Reduction Parameters

The main parameters that influence the appearance of the generated meshes are the *reduction factor*, which determines how many triangles are removed, the number of detail levels that are computed (*combine depth*) and the threshold for *normal deviation*. Fig. 9 shows development of the visual quality of the rendered data sets while loading the data. The left image shows the time for increasing number of detail levels while keeping the reduction factor constant. The central one shows the quality for two detail levels but different reduction settings.

Table 1. Statistics for the evaluated datasets for a voxel size of 5 cm. Numbers in millions, lengths in meters.

	Points	Faces	Vert.	Dimensions
Campus	2 200	1 200	610	819 × 1040 × 195
Roundhouse	1 900	97	52	280 × 255 × 32
Castle	415	82	46	152 × 180 × 25

When varying the number of levels with fixed reduction factor, more levels increase the quality of the initial visual results. Here, more LODs help to load to relevant areas at high quality faster. This only holds for the first few seconds. Later on, the higher number of detail levels requires more data to be transferred, which significantly increases the loading time. Here, fewer levels with more detailed models help to reduce the load time. However, fewer levels reduce the possibilities to simplify the meshes, hence especially for large scenes, more levels with coarser geometry are required to quickly provide visually compelling results. In our implementation we therefore set the default number of detail levels to three.

The other experiment shows that – as expected – with a fixed detail level, more reduced models lead to shorter loading times. High reduction however leads to problems with the incremental simplification, since it can only simplify meshes by a certain amount given its quality constraints. A higher reduction on each level means that this point is reached sooner, preventing the higher levels from further simplifying the mesh. This, in turn, results in identical LODs in the higher levels and therefore duplicate loads and slower loading times the more levels there are. Hence, we chose to start with a medium compression level of 0.2 to complement the previously set number of three levels.

The influence of the accepted normal deviation is shown in the graph on the right in Figure 9. A smaller angle means that the normals may deviate less from their original direction, so that the simplification is finished earlier. This results in a LOD where the operations with the greatest visual impact are not performed, so that the meshes are more accurate but less reduced. For the graph in the figure, the strictest configuration with an allowed deviation of 10 degrees was used as reference. It can be seen that 10° to 30° deliver the best meshes, while the other configurations do not exceed 80% similarity. In return such configurations have much shorter loading times, since the meshes could be reduced more. Incremental simplification usually ensures that the operations with the least visual impact are executed first. The deviation of normals does not influence this prioritization, but only prevents operations that have too much deviation.

Generally, the usage of this approach would be favourable as a good compromise between quality and compression, if it could be limited to higher LODs. However, in our mesh simplification library no such weighting is implemented. Hence, we will include this in future work.

[3] http://kos.informatik.uni-osnabrueck.de/3Dscans/
[4] http://data.riegl.international/riegl-pointclouds-samples-in-potree.html
[5] https://github.com/KDE/heaptrack

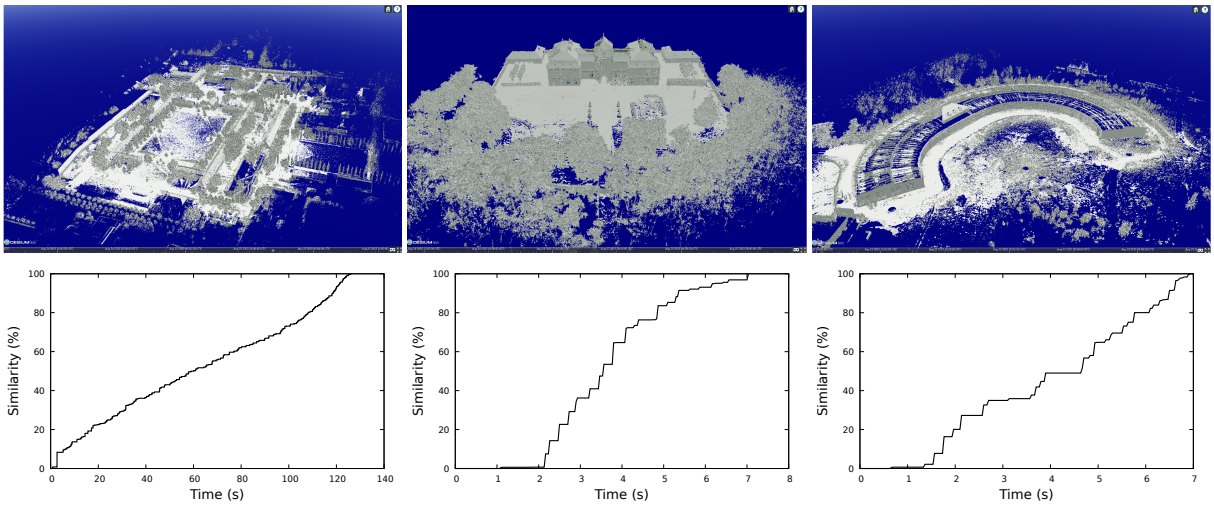

Figure 8. Bremen Campus, Castle and Roundhouse rendered in the Cesium viewer (top) and similarity scores during loading (bottom).

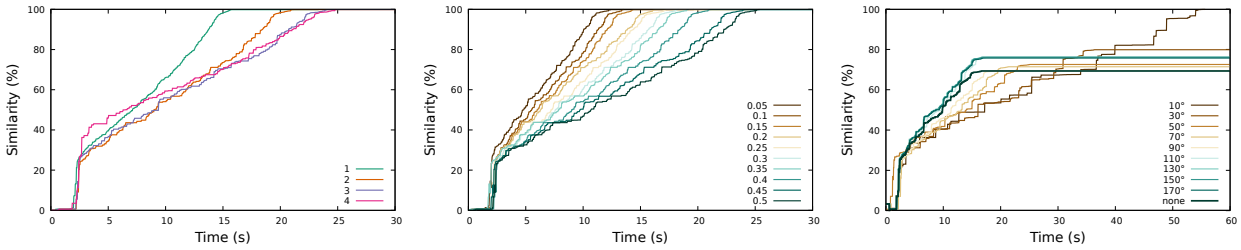

Figure 9. Similarity scores over time with fixed reduction factor of 0.4 and up to four LODs (left), three LODs and varying reduction factors (middle) and fixed LOD and reduction factor (same values as left and middle).

Table 2. Runtimes in minutes for 3D Tiles generation.

	Mesh	Chunk	Cloud
Campus	265	215	1169
Roundhouse	28	17	83
Castle	15	12	95

4.3 Runtime and Memory Consumption

The runtime experiments were performed on an Intel Xeon Workstation with 52 Cores and 192 GiB RAM. This setup was primarily chosen due to the large amount of main memory required to compare our chunk-based approach with direct reconstruction without chunking. To make the results comparable, we performed all experiments in this paper on this computer. Due to the overall low memory requirements, the results should be reproducible on standard desktop hardware as well. The runtimes to convert our data into the 3D Tiles format is summarized in Table 2.

The *Mesh* and *Chunk* columns report the time to build a tileset from pre-computed meshes or chunks. In these cases no surface reconstruction is necessary, hence the times are significantly lower compared to last column, where the complete reconstruction time is included. Overall, the times reported here indicate, that the generation of such data is feasible on consumer grade hardware.

An important feature of our implementation is the limitation of the memory consumption. In contrast to previous approaches, the amount of required memory does not increase with the size of the reconstructed area. The memory usage of the creation process of this mesh can be seen in the diagrams in Fig. 10.

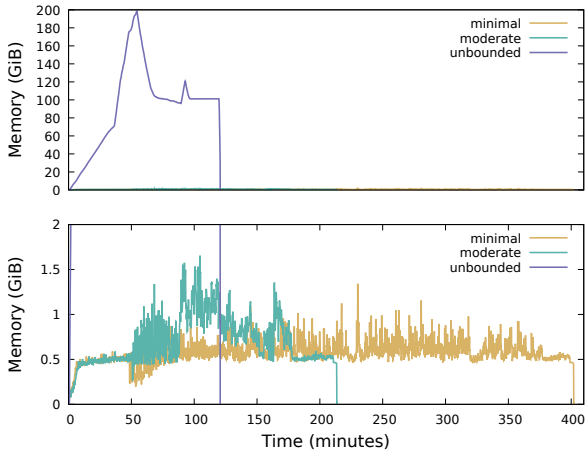

Figure 10. Memory consumption of our approach over time. The bottom graph is a zoomed-in view of the bottom area in the upper graph.

Here, the memory consumption is reported on different scales. The upper part shows the total consumption of the unbounded reconstruction on the Campus dataset, where no memory optimizations are used. Here, the peak is around 200 GiB, which is an order of magnitude more than capacities on consumer hardware (8 to 32 GiB). However, the process finishes early and requires only about 25% of the time required with the minimal and moderate bounded approaches. The lower part details the memory allocation for the latter. The moderate heuristic uses only a little more RAM than the minimal approach but needs only half the time. For the presented datasets in both cases the required

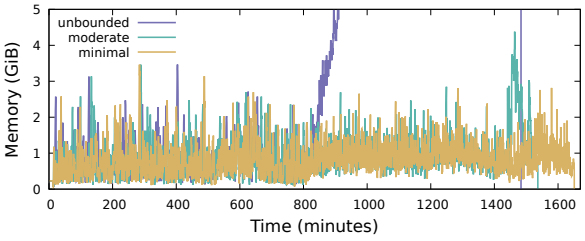

Figure 11. Memory consumption of an entire reconstruction and LOD generation of the Campus dataset from a point cloud.

memory is well below 2 GB. This indicates that the moderate heuristic is suitable for consumer hardware and delivers the results nearly 50% faster than the minimal setting. Hence, this method should be preferred in most application scenarios.

A full measurement of the reconstruction from a point cloud using Large Scale Reconstruction with the newly added extension to generate 3D Tiles can be seen in Fig. 11. The overall process does not exceed 4 GiB when set to minimal, which is still well within the constraints of consumer hardware.

5. DISCUSSION AND OUTLOOK

In this paper we presented an approach to generate Cesium 3D Tiles datasets for fast rendering from point clouds on consumer hardware. The presented approach is able to generate visually consistent meshes of arbitrarily large areas on standard PCs. The main benefit of our method is, that it is able to limit the memory consumption while maintaining reasonable performance in terms of computation time. In contrast to other methods that requires manual pre-segmentation, our algorithm computes consistent meshes for different levels-of-detail that are automatically converted into the 3D Tiles format. In future work, we plan to integrate texture information from camera data into the reconstruction process to generate visually more compelling 3D models. Also, automatic segmentation methods could be included, to replace instances of reoccurring objects with pre-defined models in order to further enhance the visual quality and reduce the amount of data transferred.

ACKNOWLEDGEMENTS

The DFKI Niedersachsen (DFKI NI) is sponsored by the Ministry of Science and Culture of Lower Saxony and the VolkswagenStiftung. Hardware for the experiments was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Grant no. 456666331.

REFERENCES

Baert, J., Lagae, A., Dutré, P., 2013. Out-of-core construction of sparse voxel octrees. *Proceedings of the 5th high-performance graphics conference*, 27–32.

Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., Lévy, B., 2010. *Polygon mesh processing*. CRC press.

Cesium GS Inc., 2022a. Cesium. `https://github.com/CesiumGS/cesium` Last accessed: 2022-11-04.

Cesium GS Inc., 2022b. Cesium 3d tiles specification. `https://github.com/CesiumGS/3d-tiles` Last accessed: 2022-11-04.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., Scopigno, R., 2004. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics (TOG)*, 23(3), 796–803.

Garland, M., Heckbert, P. S., 1997. Surface simplification using quadric error metrics. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 209–216.

Hobu Inc., 2022. Entwine. `https://entwine.io/entwine-point-tile.html` Last accessed: 2022-11-04.

Hoppe, H., 1996. Progressive meshes. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 99–108.

Lindstrom, P., 2003. Out-of-core construction and visualization of multiresolution surfaces. *Proceedings of the 2003 symposium on Interactive 3D graphics*, 93–102.

Lorensen, W. E., Cline, H. E., 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH*.

Melax, S., 1998. A simple, fast, and effective polygon reduction algorithm. *Game Developer*, 11(Nov), 44–49.

Nießner, M., Zollhöfer, M., Izadi, S., Stamminger, M., 2013. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6), 1–11.

Poliarnyi, N., 2021. Out-of-core surface reconstruction via global tgv minimization. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5641–5650.

Rusinkiewicz, S., Levoy, M., 2000. Qsplat: A multiresolution point rendering system for large meshes. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 343–352.

Schütz, M. et al., 2016. Potree: Rendering large point clouds in web browsers. *Technische Universität Wien, Wiedeń*.

Schütz, M., Ohrhallinger, S., Wimmer, M., 2020. Fast out-of-core octree generation for massive point clouds. *Computer Graphics Forum*, 39number 7, Wiley Online Library, 155–167.

Seufert, M., Kargl, J., Schauer, J., Nüchter, A., Hoßfeld, T., 2020. Different points of view: Impact of 3d point cloud reduction on qoe of rendered images. *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, IEEE, 1–6.

Wiemann, T., Mitschke, I., Mock, A., Hertzberg, J., 2018. Surface reconstruction from arbitrarily large point clouds. *2018 Second IEEE International Conference on Robotic Computing (IRC)*, IEEE, 278–281.

Wiemann, T., Nüchter, A., Hertzberg, J., 2012. A toolkit for automatic generation of polygonal maps-las vegas reconstruction. *ROBOTIK 2012; 7th German Conference on Robotics*, VDE, 1–6.

Yoon, S.-E., Salomon, B., Gayle, R., Manocha, D., 2005. Quick-VDR: Out-of-core view-dependent rendering of gigantic models. *IEEE Transactions on Visualization and Computer Graphics*, 11(4), 369–382.