Smooth, Interactive Rendering Techniques on Large-Scale, Geospatial Data in Flood Visualisations

Christian Kehl, Tim Tutenel and Elmar Eisemann
Computer Graphics and Visualisation Group
Delft University of Technology
Email: [c.kehl, t.tutenel, e.eisemann]@tudelft.nl

***Abstract*—Visualising large-scale geospatial data is a demanding challenge that finds applications in many fields, including climatology and hydrology. Due to the enormous data size, it is currently not possible to render full datasets interactively without significantly compromising quality (especially not when information changes over time).**

In this paper, we present new approaches to render and interact with detail-varying Light Detection and Range (LiDAR) point sets. Furthermore, our approach allows the attachment of large-scale geospatial meta information and the modification of point attributes on the fly.

The core of our algorithm is a dynamic GPU-based hierarchical tree data structure that is used in conjunction with an out-of-core, Level-of-Detail (LoD)-Point-based Rendering (PBR) algorithm to modify data on the fly. This combination makes it possible to augment the original data with dynamic context information that can be used to highlight features (e.g., routes, marked areas) or to reshape the entire data set in real-time.

We showcase the usefulness of our algorithm in the context of disaster management and illustrate how decision makers can discuss a flood scenario covering a large area (spanning 300 km^2) and discuss hazards, as well as related protection measures, interactively. One of our presented reference point sets includes parts of the AHN2 data set (14 TB of LiDAR data in total). Previous rendering algorithms relied on a long offline preprocessing (several hours) to ensure a quick data display. This step made any changes to the data impossible. With our new approach, we can modify point sets without requiring a new preprocessing run.

Introduction

An increasing amount of 3D geographic information has recently become available as Digital Elevation Model (DEM), height maps and LiDAR point scans. These data are used in many domains, such as climatology [12], [6] and hydrology [7], [17], [10], to help in the decision making processes. Topographic measurements, geospatial meta data (i.e. area selections) and domain-specific data (e.g. fluid simulation results, water currents and cloud simulation data and measurements) are combined in an illustrative manner.

Recent geographic point data, often captured with terrestrial and aerial LiDAR technology, are too large to be rendered directly. The data neither fit in arendering-dedicated graphics memory, nor in a workstation's main memory. We refer to such data sets as *massive point sets*.

Many tree- and graph-based solutions for rendering large data sets [15], [4], [1], [11], [5], and LiDAR data in particular [9], [8], [3], are already available. Graph-based solutions continuously request new data nodes for the areas in view. The data is often structured in trees in which nodes correspond to a set of points in the corresponding scene area. By limiting the descent in the tree, the point density can be adjusted. We similarly rely on a tree structure that allows us to ensure a *rendering-on-budget* paradigm, such as explored by Goswami et al.[5], to address the rendering of massive, coloured LiDAR datasets. Further, we avoid common popping artifacts that result from adding/removing the points in a node and rather add/remove them progressively, hereby ensuring a smooth appearance.

Besides the rendering, interaction is another key component for using massive point sets in a decision making process. In this context, we refer to "interaction" as the possibility to modify data attributes (e.g. position, colour, visibility) on-the-fly to highlight or adapt information and to get instantaneous visual feedback of the performed operation. Massive point sets are usually modified in an offline process. This procedure is time consuming and not suitable for collaborative decision making. The interactive modification of massive point sets is currently not possible without visual quality loss. Available point set editors [18], [14] can only facilitate interactive modifications for data as far as they fit in the computer's main memory. Wand et al. [15] presented an improved system with the capability to process massive data set in a batch-process manner using a tree hier-

archy. A main assumption of current approaches is that modifying operations are permanently valid. Regarding decision making processes and the goal to provide techniques for collaborative, on-the-fly information exchange, we assume that modifications are only temporarily valid.

The combination of the two presented techniques makes it possible to augment the original data with dynamic context information that can be used to highlight features (e.g., routes, marked areas) or to reshape the entire data set in real-time. We showcase the applicability of our approach in the domain of flood hazard management. Three use cases are presented that demand such large-scale modifications. Our approach is able perform operations interactively that formerly demanded substantial preprocessing. We also briefly present first performance assessment results.

Methods

Rendering-on-Budget

Our initial approach spatially subdivides the point set into tiles of 1000 metre by 1250 metre. Each tiles' points are distributed uniformly along the levels of details [2], [7]. Using this technique, lots of points need to be loaded as a bucket immediately if the user approaches higher detail levels. Our method always loads at least the roughest detail level for each bucket in the dataset. This results in spatial limitations on the presentable extent of LiDAR datasets in practice. This could be solved by employing additional, view-dependent criteria. As formerly discussed [7], this bucket-loading behaviour leads to visual discontinuities that irritate the user.

In our new approach, we apply the rendering-on-budget paradigm to provide a constant rendering speed for large point sets [16]. Hence, The currently available budget depends on the data size and the technical capabilities of the rendering system. The budget is adapted using a Proportional-Integral-Differential (PID) controller, which keeps track of the number of rendered points in relation to total rendering time.

During the rendering, we traverse the LoD tree structure in breadth-first order. For each node in the tree, we render its points as long as rendering budget is still available. We use the local density as a measure to decide which particular should be rendered next. Because the rendering decision is taken on per-point-basis rather than the per-node point bucket of the initial approach, the new technique offers a smooth LoD transition while preserving rendering speed.

On-line Modification of Large-Scale, Geospatial LiDAR point sets

We adapt the point attributes via GPU shaders in an indirect manner. In order to describe the modifications, we use interactively-places, textured 2D polygons. One can imagine that these are defined on a map of the point-cloud region and the changes will affect all points that are covered. They include line-like structures for route descriptions and polygons definitions for areas. These geometric shapes are stored as structured vector lists. In contrast to Scheiblauer and Wimmer [13] the vector format is independent of the point cloud. Further, it is not limited to a uniform grid size (such as geometry, given as Volume Images). Our approach allows us to achieve an accurate rendering at varying scales. Nevertheless, we want to avoid preprocessing the data set, which would result in hours of computation. Instead, we resolve the issue at runtime. For each rendered point, we test whether it falls in one of the polygons and then modify its attributes according to the regions definition.

During the rendering stage, we can not evaluate the bare polygon geometry because the point vertices and the polygon vertices are part of the same global vertex set. Hence, to overcome this drawback, we store the polygon geometry in a texture that can subsequently be evaluated for each point cloud vertex.

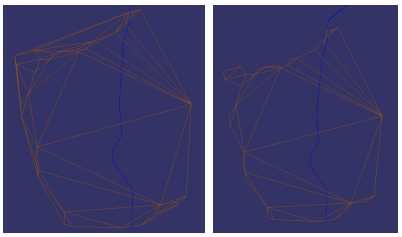
As a first step, we need to find the polygon each point belongs to. In this case, each point needs to be tested with each polygon for an intersection. We need to do this because there is no other reference information available (apart from the point's 3D position) to establish a point-polygon relation. If this localisation succeeds, we can apply our attribute modification. Our initial algorithm presented in Alg. 1.

Calculating the intersection of a point and a polygon is computationally expensive and existing implementations of this procedure do not map well on GPU architectures. It is thus not practical to use a point-in-polygon check for the number ofLiDAR points and annotations we intend to handle. Hence, we simplify Alg. 1 by transforming the polygons into 2D triangular meshes. This exchanges the *PointPolygonIntersection*-test with a *PointTriangleIntersection*-test, which is much simpler to compute and implement on a GPU.

Algorithm 1 Apply point attribute modification

```
function ModifyAttribute(pointlist, polygons)
    for all point ∈ pointlist do
        for all polygon ∈ polygons do
            if PointPolygonIntersect(point, polygon)
is true then
                attribute ← GetAttribute(polygon)
            else
                attribute ← 0
            end if
            ApplyAttribute(point, attribute)
        end for
    end for
end function
```

On the other hand, transforming the 2D polygon to a valid triangular mesh that includes exclusively triangles inside the (possibly concave) polygonal border is another challenge. We approach this problem by first computing a constrained 2D Delaunay Triangulation. This guarantees that the polygonal borders are part of the triangulation. The triangulation generates a convex mesh representation, which means it may also include exterior triangles outside of the polygon constraint. In a second step, we consequently eliminate the unnecessary triangles. We extract the triangles for each polygon separately. Thus, we know which polygon each triangle belongs to. The resulting meshes for an example region are shown in Fig. 1(a)- 1(b).



(a) (b)

Figure 1. polygon-conform constrained 2D Delaunay Triangulation: A straight implementation of the constrained 2D Delaunay Triangulation creates exterior triangles (a), which need to be eliminated. We do this by testing if each triangle is inside the polygon. The result is a polygon-conform 2D mesh (b).

Due to the massive point sets and the numerous triangles, it is still not possible to test each point with each triangle for intersections while maintaining interactive frame rates. We incorporate the triangles in a quadtree data structure in order to reduce the number of triangles that needs to be checked. In the quadtree construction, we start off with a single cell and sequentially add triangles to the quadtree. If the number of triangles in a cell exceeds a limit X ($X = 4$ in our experiments), we split the cell into 4 equal parts. We continue the subdivision up until a certain maximum depth Y ($Y = 5$ in our experiments). Applying this maximum depth limits the tree traversal processed and proved efficient in our implementation.

During runtime, we traverse the tree to determine for each point its respective quadtree node. After extracting a reduced list of candidate triangles (resp. leafs of the final node) for each point, we perform the aforementioned *PointTriangleIntersection*-test to accurately establish a point-triangle relation while maintaining interactive frame rates.

At this point, we have access to the triangle attributes (such as colour and texture coordinates) and their textures. By applying the stored triangle colour or evaluating its texture coordinate inside the texture, we can recover the modification operation for the given point.

Currently, we support the following operations:

- colour a point according to the triangle colour
- discard a point if it is inside the triangle
- colour a point according to the triangle's texture
- displace a point according to a displacement map

The final algorithm is shown in Alg. 2 and 3.

Results

In a first use case, we apply our algorithm together with the discard-operation to create a historic visualisation of the 1953 North Sea flood in the Dutch provinces of South Holland, North Brabant and Zeeland. This visualisation spans an area of 45 km by 60 km, containing around 2 TB of coloured LiDAR data. The problem for a historic visualisation is the amount of landscape- and water protection changes that occurred since 1953, in comparison to the acquired data we currently have. In order to recreate the scenario, we have to adapt the landscape and cut out structures that were added since then. Without the presented approach, this would mean to re-process the 2 TB dataset and discard unavailable points in the process. This process takes several days of computing time. With our current method, we can define the areas we want to exclude from the visualisation, and all point vertices that are covered by the defined areas are discarded by the GPU during rendering. The result is shown on parts of the data set in Fig. 2(a) and 2(b).

Algorithm 2 triangle texture generation

```
function PolygonToTriangleMesh(polygon)
    ConstrainedDelaunay2D(polygon)
    for all triangles ∈ trianglelist do
        if triangle not inside polygon then
            Delete(triangle,trianglelist)
        end if
    end for
    return trianglelist
end function

function MeshToTexture(trianglelist)
    ▷ converts mesh into a quadtree, then encodes as texture
    create quadtree with root node
    for all triangles ∈ trianglelist do
        locate node in quadtree for triangle
        if depth(node) < Y then
            if length(siblings(node)) > X then
                sibling ← SplitNode(node)
                node ← sibling
            end if
        end if
        AddTriangleLeaf(node,triangle)
    end for
    texture ← Texture1D()
    SerialiseQuadTree(quadtree, texture)
    return texture
end function
```

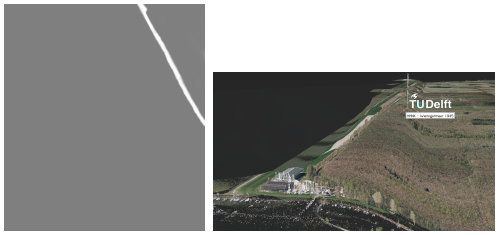
Algorithm 3 point attribute modification on the GPU

```
for all point ∈ pointlist do
    function GetAttribute(pointlist, texture)
        quadtree ← Texture1D(texture)
        node ← root
        while (depth > 5) ∧ (point ∈ node) do
            node ← Sibling(point, node)
            if node = NULL then
                attribute ← 0
                return attribute
            end if
        end while
        for all triangle ∈ node do
            if PointTriangleIntersect(point, triangle) is true then
                attribute ← GetAttribute(triangle)
                return attribute
            end if
        end for
        attribute ← 0
        return attribute
    end function
    ApplyAttribute(point, attribute)
end for
```

(a) (b)

Figure 2. Visualisation of the 1953 North Sea Flood: originally acquired data (a) and historically adapted data (b)

In a second use case, flood hazard managers can use the system to interactively discuss dyke adaptation strategies. For this case, we generate a displacement map together with the area marking. The displacement can be modified during the runtime to update the point vertex displacement at the dyke. This allows interactive discussions on future adaptation strategies. In combination with the flood simulation visualisation, it allows first estimates of the discussed measures' applicability. The adapted point cloud can be seen in Fig. 3(b), the related displacement map in Fig. 3(a).

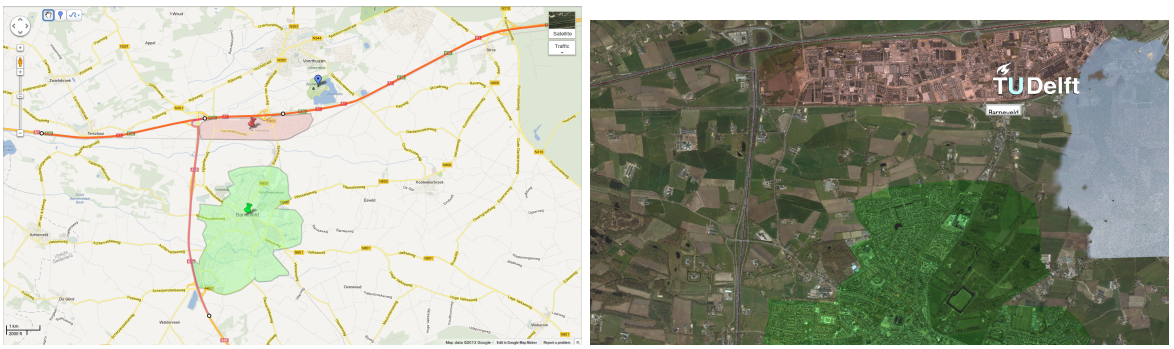


(a) (b)

Figure 3. Dyke adaptation to discuss water protection policy measures, with applied displacement (a) and the resulting point cloud of the dyke (b).

In the final use case, we show an area of 15 km by 20 km near the Dutch city of Barneveld. In this use case, policy managers can discuss protection strategies by marking different areas of importance interactively. For this use case, we colour the point cloud according to polygon colours that are given by a GoogleMaps KML-file. This allows policy managers to use familiar interfaces via mobile devices in order to interact with the point cloud data. In the example shown in Fig. 4(a) and 4(b), we distinguish between an industrial area and the residential parts of the city (marked in red respectively green).

Our initial performance assessment on the interactive point cloud modification reveals a tight correlation between the rendering speed and the geometric complexity of the polygon (i.e. the num-



(a) (b)

Figure 4. Interactive colouring of big LiDAR data (b) using GoogleMaps interfaces (a).

ber of triangles resulting from the triangulation). For the assessment, we have chosen data sets of 3, 10 and 20 areas, including 20, 42, and 298 triangles. Renderings of these data sets are shown in Fig. 5. The speed measurements are presented in Fig. 6. The test system was a workstation with a 6 x 3.2GHz Intel Xeon processors (HT), 16GB of main memory and an NVIDIA GeForce GTX 680 graphics adapter.

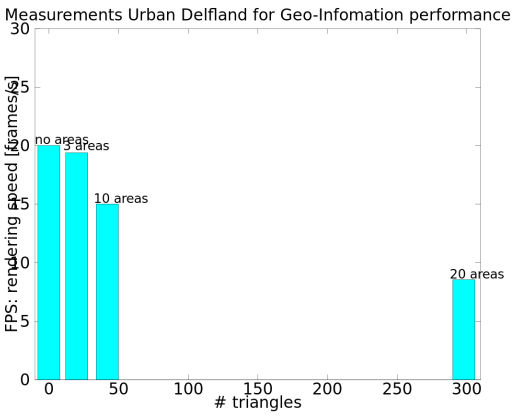


Figure 6. rendering speed measurements for geo-information integration with 3, 10 and 20 areas.

Conclusions

We presented an approach to render and interact with massive aerial LiDAR point sets.

Our rendering-on-budget approach results in a smooth level-of-detail transition at real-time frame rates.

With our on-line modification algorithm and its implementation, we have shown that even out-of-core datasets allow on-the-fly modification without tedious preprocessing procedures. The algorithm is a first step towards more elaborate procedures that allow on-the-fly marker placement and interaction as well as full 3D-modification via volume images or 3D polygons.

Our system, combining all discussed methods, enables decision makers in flood hazard management to collaboratively discuss protection scenarios and devise new protection measures.

Acknowledgements

This project was funded by the Dutch Research Program Knowledge for Climate (KfC) and partially supported by the Intel VCI, and the European Project Harvest4D.

References

[1] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. GigaVoxels: ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 15–22, New York, NY, USA, 2009. ACM. Cited by 0115.

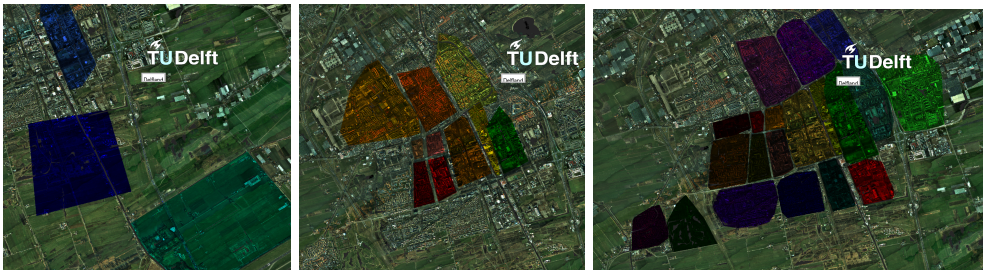
[2] G. de Haan. Scalable visualization of massive point clouds. *Nederlandse Commissie voor Geodesie KNAW*, 49:59–67, 2009.

[3] Zhenzhen Gao, Luciano Nocera, and Ulrich Neumann. Fusing oblique imagery with augmented aerial LiDAR. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 426–429, New York, NY, USA, 2012. ACM.

[4] Enrico Gobbetti, Fabio Marton, and JosÃĭ Antonio Iglesias GuitiÃąn. A single-pass gpu ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24(7–9):797–806, 2008.

[5] Prashant Goswami, Fatih Erol, Rahul Mukhi, Renato Pajarola, and Enrico Gobbetti. An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *The Visual Computer*, February 2012.

[6] H. Jänicke, M. Bottinger, and G. Scheuermann. Brushing of attribute clouds for the visualization of multivariate data. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1459–1466, 2008.



(a) (b) (c)

Figure 5. Geo-information integration use cases: 3 (a), 10 (b) and 20 (c) area data set

[7] Christian Kehl and Gerwin de Haan. Interactive simulation and visualisation of realistic flooding scenarios. In *Intelligent Systems for Crisis Management*, 2012.

[8] Bostjan Kovac and Borut Zalik. Visualization of LIDAR datasets using point-based rendering technique. *Computers & Geosciences*, 36(11):1443–1450, November 2010.

[9] Oliver Kreylos, Gerald W. Bawden, and Louise H. Kellogg. Immersive visualization and analysis of LiDAR data. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Paolo Remagnino, Fatih Porikli, Jörg Peters, James Klosowski, Laura Arns, Yu Ka Chun, Theresa-Marie Rhyne, and Laura Monroe, editors, *Advances in Visual Computing*, number 5358 in Lecture Notes in Computer Science, pages 846–855. Springer Berlin Heidelberg, January 2008.

[10] Wenqing Li, Ge Chen, Qianqian Kong, Zhenzhen Wang, and Chengcheng Qian. A VR-Ocean system for interactive geospatial analysis and 4Dvisualization of the marine environment around antarctica. *Computers & Geosciences*, 37(11):1743–1751, November 2011.

[11] Ruggero Pintus, Enrico Gobbetti, and Marco Agus. Real-time rendering of massive unstructured raw point clouds using screen-space operators. In *Proceedings of the 12th International conference on Virtual Reality, Archaeology and Cultural Heritage*, VAST'11, pages 105–112, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.

[12] W. Ribarsky, N. L. Faust, Z. J. Wartell, C. D. Shaw, and J. Jang. Visual query of time-dependent 3D weather in a global geospatial environment. 2002.

[13] Claus Scheiblauer and Michael Wimmer. Out-of-core selection and editing of huge point clouds. *Computers & Graphics*, 35(2):342–351, 2011. <ce:title>Virtual Reality in Brazil</ce:title> <ce:title>Visual Computing in Biology and Medicine</ce:title> <ce:title>Semantic 3D media and content</ce:title> <ce:title>Cultural Heritage</ce:title>.

[14] Michael Wand, Alexander Berner, Martin Bokeloh, Arno Fleck, Mark Hoffmann, Philipp Jenke, Benjamin Maier, Dirk Staneker, and Andreas Schilling. Interactive editing of large point clouds. In *Proceedings of Symposium on Point-Based Graphics (PBG 07)*, pages 37–46, 2007.

[15] Michael Wand, Alexander Berner, Martin Bokeloh, Philipp Jenke, Arno Fleck, Mark Hoffmann, Benjamin Maier, Dirk Staneker, Andreas Schilling, and Hans-Peter Seidel. Processing and interactive editing of huge point clouds from 3d scanners. *Computers & Graphics*, 32(2):204 – 220, 2008.

[16] Berend Wouda. Visualization on a budget for massive lidar point clouds. Master's thesis, Delft University of Technology, 2011.

[17] Izham Mohamad Yusoff, Muhamad Uznir Ujang, and Alias Abdul Rahman. 3D dynamic simulation and visualization for GIS-based infiltration excess overland flow modelling. In Jiyeong Lee and Sisi Zlatanova, editors, *3D Geo-Information Sciences*, Lecture Notes in Geoinformation and Cartography, pages 413–430. Springer Berlin Heidelberg, 2009.

[18] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3D: an interactive system for point-based surface editing. *ACM Trans. Graph.*, 21(3):322–329, July 2002.