Point Cloud Manager: Applications of a Middleware for Managing Huge Point Clouds

Omar A. Mures *University of A Coruña, Spain*
Alberto Jaspe *CRS4, Italy*
Emilio J. Padrón *University of A Coruña, Spain*
Juan R. Rabuñal *University of A Coruña, Spain*

INTRODUCTION

Presently, acquisition technologies such as LIDAR (Laser Imaging Detection and Ranging) (Van Genderen, 2011) have seen an unprecedented amount of advancements in terms of the quality and precision of the acquisition hardware. These devices measure distance by using a laser to illuminate a target and then analyzing the reflected light. This distance information is combined with different data obtained from other techniques such as photogrammetry, radiometry, etc. These measurements are repeated for all surfaces of the target reachable by the laser scanner, resulting in a set of points with information about their position, color, reflectivity, etc. This acquisition procedure leads to high precision georeferenced 3D scans of the real world with an exceptional amount of data, sometimes exceeding billions of points. The processing and visualization of these datasets on commodity systems present several challenges that can be addressed from a Big Data perspective by applying High Performance Computing and Computer Graphics techniques (Yuan, 2012).

In order to manage these huge point clouds and perform operations seamlessly on them, we have developed a middleware we have named Point Cloud Manager (PCM) (Jaspe, 2012) (Mures, 2014a). This software package (Mures, Jaspe, Padrón, & Rabuñal, 2013) comprises a multiplatform library and a set of tools around it that allows the management of massive point clouds with arbitrary attached data on commodity hardware. The library provides an abstraction for an arbitrarily large point cloud stored in secondary memory (HDD, SSD, NFS...), exposing a simple and clear API to get access to the dataset in RAM or VRAM and perform out-of-core operations on CPU or GPU. The two main pillars behind PCM are a multiresolution spatial structure and a hierarchy of software caches as can be seen in Figure 1.

Given the spatial nature of the point clouds datasets, the multiresolution structure used in PCM is strongly inspired in the space subdivision techniques usually applied in 3D computer graphics. This structure is exploited by PCM to provide interactive access to the dataset when needed, for example for visualization, or an iterative computation based on the multiresolution levels, that is, converging towards a solution by traversing the multiresolution structure with a certain threshold.

A hierarchy of two software caches is used for the transparent and efficient out-of-core access to the dataset, a synchronous software cache in VRAM to exploit GPGPU capabilities or simply perform advanced point-based rendering, and an asynchronous one in RAM to provide multi-thread support for CPU(s). Thus, chunks of 3D point data are transferred when needed, as a response to the high level API calls.

Hence, from low-level memory management to conversion and visualization of the point clouds, PCM makes out-of-core point cloud operations easier and more efficient for the programmer. The usage of this framework allows us to use datasets with unprecedented precision, since it is not necessary to decimate clouds before processing them. This means that we can perform operations taking advantage of the high precision of the scanners, sometimes even reaching micrometer precision. This can be especially relevant in fields such as civil engineering, topography or architecture, where applications are usually forced to decimate huge point clouds to be able to manage them and apply certain algorithms.

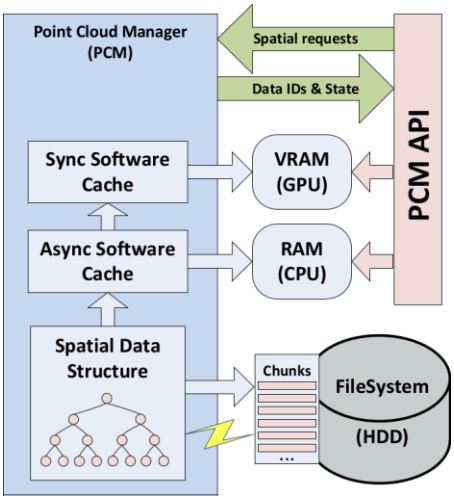

Figure 1. System overview of PCM

This article shows in a didactic manner the application of the aforementioned techniques to real world case studies.

BACKGROUND AND RELATED WORK

The first point-based rendering techniques appeared long ago (Levoy & Whitted, 1985), when datasets were small and the main research effort was devoted to displaying points with the highest quality possible. As years passed, the acquisition hardware improved greatly, leading to datasets with millions of points, and causing research shifted towards achieving not only the best possible rendering results, but also dealing with huge amounts of point data and the new issues it brings along. An example of this is (Gobbetti & Marton, 2004), a simple point-based multiresolution structure that deals with the problems associated with big point datasets. The multiresolution structure used in our system was strongly inspired by this important work.

Novel approaches for point-based renderers such as (Elseberg, Borrmann, & Nüchter, 2013) focus on memory efficiency and out-of-core rendering of big point sets. They use a memory efficient octree that uses fixed depth and a minimum number of points in its construction. The acceleration structure is employed for frustum culling, ray casting, nearest neighbor search and RANSAC (for plane detection). Another recent work (Wenzel, Rothermel, Fritsch, & Haala, 2014b) is also based on an octree, in this case allowing the creation of a dynamic spatial structure with on-demand management of memory for loading and writing of points. This proposal is tested with a photogrammetric filtering algorithm in (Wenzel, Rothermel, Fritsch, & Haala, 2014a).

Although the octree is maybe the most popular acceleration structure when working with point sets, other spatial structures have also been used over the years. In (Kuder, Šterk, & Žalik, 2013) a quadtree is used to render hybrid point-polygon models. This work does not address out-of-core rendering, but performs a point cloud simplification that is eventually rendered by using triangle meshes and textures. Other approaches use multi-way kd-trees as an acceleration structure such as (Goswami, Erol, Mukhi, Pajarola, & Gobbetti, 2013). This type of structure is built with an out-of-core approach and applying a multiresolution approach. It is based on a fast high quality point simplification method, which leads to a balanced tree with uniformly sized nodes. Since memory efficiency is key, LZO compression is used to minimize the memory footprint and an efficient visualization method based on a rendering budget is used to display the points. This system is also capable of performing occlusion culling and back-face culling, which will aid when dealing with huge datasets.

None of the above approaches propose a general-purpose out-of-core multiresolution middleware like PCM. PCM's out-of-core kd-tree has the main advantage of having dynamic logarithmic depth and a multiresolution approach that will yield benefits when either visualizing the clouds or computing on it. PCM pushes the boundaries of out-of-core point rendering even in low-end hardware, being able to work with really huge datasets (more than 10 K Million points), increasing in an order of magnitude the number of points tested in other approaches. All this exposing an interface oriented to the creation of new out-of-core point cloud algorithms in an almost transparent way for the user/programmer.

MAIN FOCUS OF THE ARTICLE

This article focuses on showcasing how the above mentioned framework can be used to perform arbitrary operations on massive point clouds, showing different real world applications built using PCM, in both real-time and offline, render-oriented and computation-related operations as well. An insightful description about PCM, the design decisions, how the implemented software caches work and the API exposed to the programmers are the objectives of this chapter (Jaspe, Mures, Padrón, & Rabuñal, 2014). How to perform common types of point cloud filtering, processing, visualization and object detection using PCM will be shown. Employing PCM when working with huge point clouds will offer several advantages that will be outlined in the following sections.

Point cloud filtering and processing

The first filter we have chosen is a statistical outlier filter, which is typically used to remove noise or registration errors. Removing these outliers is good to facilitate further calculations, such as normal or radius estimation. This example performs a statistical analysis of each point neighborhood, eliminating those points that meet a certain criteria. Using PCM to implement such a simple filter as this one is quite fast, since we do not have to worry about point cloud formats or implementing an acceleration structure for neighborhood queries. In PCM the clouds are divided in *chunks* that are subsets of points, which make dealing with these datasets easier for the programmer. The use of these artifacts not only helps the programmer, but will allow the writing of pseudocode very similar to the actual implementation of these algorithms using PCM in C++. For this purpose, we will start by computing the mean distance $\bar{D}$ to each point neighbor:

$$\bar{D} = \frac{\sum_j \|p_j - p\|}{k}, \forall p_j \in \mathcal{N}_k(p) \quad (1)$$

Being $\mathcal{N}_k(p)$ the neighbors of point p. Assuming that the resulting distribution will be Gaussian in nature, it will have a mean and a standard deviation. Points that have a mean distance that does not fall in an interval can be removed with confidence. This interval is given by the mean and standard deviation of all the global distances. The mean is calculated with the following equation:

$$\mu = \frac{\sum_j \bar{D}_j}{N}, \forall \bar{D}_j \in \mathcal{D}_N \quad (2)$$

Being $\mathcal{D}_N$ the mean distances to the neighbors of each point. The next equation is used in the calculation of the standard deviation:

$$\sigma = \sqrt{\frac{\sum_j (\bar{D}_j - \mu)^2}{N}}, \forall \bar{D}_j \in \mathcal{D}_N \quad (3)$$

Once these two values are known, we could use the following pseudocode to perform the filtration:

```
Initialize distances to 0
for each node of point cloud do
    Request point cloud chunk
    for each point of chunk do
        distances ← K mean distance to neighbors
    end for
    Free point cloud chunk
end for
for each distance of distances do
    sum ← sum + distance
    sumsquared ← sumsquared + distance * distance
end for
mean ← sum / N of points
variance ← (sumsquared - sum * sum / N of points) / (N of points - 1)
stddev ← sqrt(variance)
threshold ← mean + factor * stddev
for each node of point cloud do
    Request point cloud chunk
    for each point of chunk do
        if distance <= threshold then
            Store point
        else
            Discard point
        end if
    end for
    Free point cloud chunk
end for
```

Algorithm 1. The statistical outlier filtering algorithm using PCM

The second filter we have chosen as an example of application is a voxel grid filter. Even though PCM can manage arbitrarily dense point clouds, as multiresolution layers are inherently being applied, we may need to obtain less precise clouds, generating new lightweight datasets from our huge clouds by down sampling them. This can be useful, if the datasets want to be used in other point cloud software that is not able to deal with these vast amounts of points. We have implemented a voxelized grid approach in conjunction with PCM to achieve this. This filter creates a 3D voxel grid from the cloud data, approximating all the points within each voxel by its centroid. This results in a less dense cloud depending on the filter parameters, and also in a more constant density in the point cloud, which is really important for certain visualization applications. PCM makes it easy to obtain an efficient implementation of this kind of filters. The first step to apply this filter is dividing the space in a set of 3D boxes (voxel grid). Once the points $\mathcal{V}_N$ that belong to a voxel are isolated, we can use the following equation to compute the centroid of the voxel:

$$C = \frac{\sum_j p_j}{N}, \forall p_j \in \mathcal{V}_N \quad (4)$$

Being p the position of the corresponding point. In the above equation, we could substitute position for color, normal or any other point attribute; to obtain the rest of the centroid data. The calculated centroid will be the down sampled point corresponding to the voxel in the filtered cloud. The following pseudocode shows how to implement this filter with PCM:

```
Initialize indexes to 0
minbb ← minimum point coordinate * inverse voxel size
for each node of point cloud do
    Request point cloud chunk
    for each point of chunk do
        coord ← point coordinate
        indexes ← coord * inverse voxel size - minbb
    end for
    Free point cloud chunk
end for
Sort indexes
total ← number of different values in indexes
for each index of indexes do
    Request point cloud chunk
    Request point
    centroid ← point coordinate
    Free point cloud chunk
    points ← 0
    while index is equal do
        Request point cloud chunk
        Request point
        centroid ← centroid + point coordinate
        Free point cloud chunk
        points ← points + 1
    end while
    centroid ← centroid / points
    Store centroid
end for
```

Algorithm 2. The voxel grid filtering algorithm using PCM

Since a point does not possess volume or area, points are not directly useable in normal 3D applications. Because of this, another common necessity when trying to obtain a high quality visualization of a point cloud, is the estimation of the point radius. This is another typical preprocessing step performed in a point cloud. This can be used by the most advanced point-rendering techniques, to obtain a higher quality visualization. This computation could be carried out applying a naive approach, but the massive size of the clouds makes it difficult to perform this computation in a reasonable amount of time. By performing this calculation using PCM, a dramatically reduction in both the computation time and the amount of work needed to implement this operation is achieved. With this example, we also showcase the parallel opportunities when using PCM, as we use a GPU kernel to achieve the desired result. In order to estimate the point radii, its k-nearest neighbors $\mathcal{N}_k(p)$ must be obtained first, this operation can be performed using PCM. Next, a watertight surface is desirable, that is a surface bounding a closed solid; or even better a closed manifold. This should be kept in mind when the point radii are estimated. When the point normal is available, the following equation will be used:

$$r = \max_j \left\| (p_j - p) - n^T (p_j - p) n \right\|, \forall p_j \in \mathcal{N}_k(p) \quad (5)$$

Being $p_j$the neighbor position and n the point normal. If the point normal is not available, we could use the next equation:

$$r = \max_j \left\| p_j - p \right\|, \forall p_j \in \mathcal{N}_k(p) \quad (6)$$

Having these two equations in mind, the following algorithm can be used in conjunction with PCM to perform this calculation using a GPU:

```
for each node of point cloud do
    Request point cloud chunk
    for each point of chunk do
        coord ← point coordinate
        Request point cloud neighboring chunks to GPU cache (VBOs)
        for each VBO of VBOs do
            Make VBO available in OpenCL
            Make coord available in OpenCL
            Launch kernel sorting by distances to neighbors in VBO
            k-neighbors ← read buffer with results
            Radius ← biggest distance to k-neighbors
        end for
    end for
    Set chunk write flag
    Free point cloud chunk
end for
```

Algorithm 3. The radii estimation algorithm using PCM

Point cloud rendering

The first approaches to point rendering like (Zwicker, Pfister, Van Baar, & Gross, 2001) were CPU based, this was clearly not the right approach for real-time point rendering. To increase performance, the massively parallel computing power of GPUs should be taken advantage of. In a similar fashion to how triangle meshes are rendered nowadays, we will offload rendering to the graphics hardware freeing the CPU to perform other tasks. This will provide much higher performance than a CPU based solution.

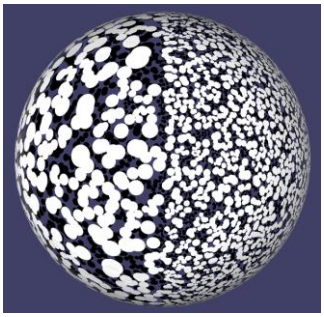

Figure 2. Screenshot of a visualizer that uses PCM

Obviously, having at one's disposal a visualization tool to manage even the hugest point clouds in real-time is ideal in these areas, so an out-of-core visualizer implementing some of the most advanced point-

based rendering techniques has been built using PCM (Mures, 2014). The software caches transparently load the points in VRAM when needed; exploiting again the inherent multiresolution features in PCM, and OpenGL is used to get a quality render with interactive frame rates. This application is also useful as a GUI for applying other operations on the point clouds being managed. The next pseudocode will show how a basic visualizer can be implemented using the software caches available in PCM as can be seen in Figure 1.

```
while render do
    mvp ← model view projection matrix
    Bind shader program
    for each cloud of point clouds do
        Pass shader arguments
        chunks ← Query multiresolution structure
        list ← Request chunks to GPU cache
        for each node of list do
            for each VBO of node do
                Bind VBO
                Set attribute buffer data
                Enable attribute buffer
                Draw points in VBO
                Disable attribute buffer
            end for
        end for
    end for
    Release shader program
    Swap buffers
end while
```

Algorithm 4. The visualization algorithm using PCM

Architecture and civil engineering applications

Civil engineering, architecture and related areas are nowadays an important target for point cloud applications and models. An example can be found in the building preservation and conservation field; where measuring and testing the correctness of a built structure might be needed. This could be easily achieved by coding a kernel on PCM to estimate the closest point-to-point or point-to-triangle distance, just disposing a 3D model of the structure obtained from the CAD plans and a point cloud dataset of the real structure, obtained from a LIDAR scanner, for example. When the minimum distance exceeds a given threshold there is a high possibility that we have encountered a defect, so these points are marked to alert the user about it. This technique could also be applied in industrial design for CAD based inspection, obtaining a point cloud of a manufactured part and then use a similar process to compare the CAD design with the real part. The out-of-core features in PCM allow us to take advantage of the full precision available in the dataset, avoiding the need to decimate even the biggest point clouds.

In order to achieve the aforementioned objective, it will be necessary to be able to calculate a point-to-point distance. This is easily achieved for points p_1 and p_2 with the following formula:

$$D_p = \|p_1 - p_2\| \quad (7)$$

The point-to-triangle distance is a little harder to obtain. In this case, the minimum distance will be computed using the squared-distance function for any point in a triangle t to the point p:

$$D_t(u,v) = |t(u,v) - p|^2 \quad (8)$$

For the following u, v values:

$$(u,v) \in S = \{(u,v) : u \in [0,1], v \in [0,1], u + v \leq 1\} \quad (9)$$

The goal will be to minimize $D_t(u,v)$ over S. Knowing that D_t is a differentiable function, the minimum will occur either in the boundary of S or where the gradient $\Delta S = (0,0)$ (Eberly, 1999).

Taking all of this into account, PCM can now be used to compute the minimum point-to-point or point-to-triangle distances for two 3D models, a point cloud against a triangle mesh or another point cloud. The CPU version will be showcased since it is easier to understand, but it could also be implemented using a GPU.

```
Initialize distances to 0
for each node of point cloud do
    Request point cloud chunk
    for each point of chunk do
        for each triangle of mesh do
            dist ← point-triangle distance
            if dist < distances then
                distances ← dist
            end if
        end for
    end for
    Free point cloud chunk
end for
```

Algorithm 5. The minimum distances algorithm using PCM

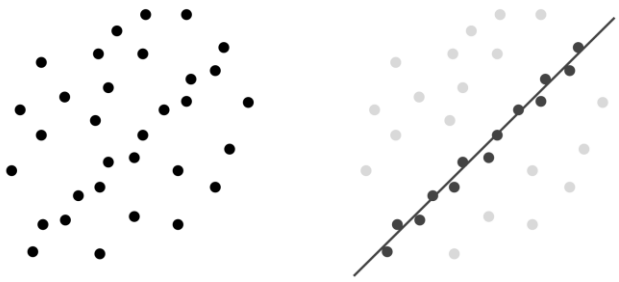

Figure 3. Result of RANSAC fitting of a line in a set of points that contain outliers and inliers

Another useful application of PCM in the fields of architecture and civil engineering could be object detection in point clouds. By applying a RANSAC-like algorithm on top of PCM we can find planes, cylinders and spheres easily and quickly in massive point clouds. RANdom Sample Consensus is an

iterative method to estimate parameters of a mathematical model from a set of observed points which can contain outliers. This is a non-deterministic algorithm, since we are not able to guarantee that it will produce a correct result. The probability of reaching a reasonable result will improve as the number of iterations increases. We showcase how this algorithm can be interactively used to detect the aforementioned primitives in point clouds. This can aid architects in the creation of CAD floor plans from point clouds, civil engineers in the documentation of existing structures, etc. For this purpose we will also demonstrate how to export the estimated primitives in a CAD friendly format. This might seem trivial at first, but since the estimated primitives are of parametric nature and some of them are not bounded, this is not an easy task. It is also useful for interactively removing objects in the point clouds, since we are able to perform primitive estimation in real-time.

To illustrate how this can be achieved using PCM, the use case of estimating a plane in a point cloud in real-time is chosen. The first step, is selecting a subset of points in the dataset, this can be done by the end user or randomly. Next, RANSAC is used to estimate the mathematical parameters of the plane. The basic assumption is that the data contains *inliers*, though it may contain noise and *outliers* (Figure 3). Outliers can appear because of errors or imprecise measurements, extreme values of noise, etc. RANSAC assumes that for a given set of inliers, there should exist a procedure that can estimate the unknown parameters of the plane model that will fit this data.

The essence of the basic RANSAC algorithm can be outlined in the following steps:

Step 1: Select randomly the minimum set of points to compute the model parameters.

Step 2: Solve for the parameters of the model (i.e. with a least squares method).

Step 3: Test how many points in the dataset fit the estimated model with a predefined tolerance ϵ.

Step 4: If the fraction of inliers over the total number of points is enough, then the mathematical model is re-estimated (since it was only estimated using the initial inliers).

Step 5: Finally, the model is evaluated by estimating the error against the inliers.

This procedure is repeated for a certain number of times, each time obtaining a new model which can be rejected if too few points are classified as inliers or a better model with its corresponding error measure.

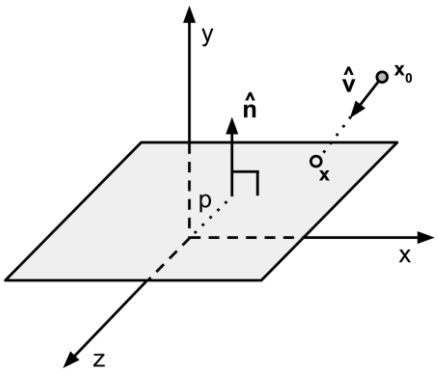

Figure 4. Hessian normal form of a plane

Once the parameters of the plane are obtained, since it is infinite, it will need to be bounded. This is achieved calculating its intersection with the axis aligned bounding box (AABB) of the point cloud. This means that a common ray-plane intersection test has to be performed for each edge. The result of this

process will be a polyline directly exportable to a CAD format. The point of intersection in the plane can be written as:

$$\hat{n} \cdot x = -p \quad (10)$$

This is called the Hessian normal form of a plane (Figure 4). For the ray, it can be written as:

$$x = x_0 + t\hat{v} \quad (11)$$

Being x_0 the origin of the ray, $\hat{v}$ the direction and t:

$$t = \frac{-(x_0 \cdot \hat{n} + p)}{\hat{v} \cdot \hat{n}} \quad (12)$$

Once all the intersection points have been calculated, they are tested to check if they fall inside the AABB. After this the vertices of the bounded plane are ready to be exported as a polyline. The pseudocode necessary to implement this use case in PCM is showcased in the following code block.

```
selectedpoint ← get user selected intersection point
for each node of neighborhood do
    Request point cloud chunk
    for each point of chunk do
        dist ← distance between point and selectedpoint
        if dist <= sampledistance then
            samples ← point
        end if
    end for
    Free point cloud chunk
end for
coefficients ← segment plane from samples
Color plane coefficients in GPU point data
intpoints ← Compute intersection of the plane with cloud AABB
polyline ← Sort intpoints
Write polyline to CAD compatible file
```

Algorithm 6. The plane estimation algorithm using PCM

FUTURE RESEARCH DIRECTIONS

The current version of the software was designed for PCs with Linux/GNU, Windows or MacOS. Future research could focus on making PCM compatible with mobile operating systems (Android, iOS, etc.), or even better modern internet browsers using WebGL and decoupling it in a client-server approach. In order to be able to utilize PCM in mobile platforms or browsers, it will need point streaming capabilities, this will mean transferring clouds over a network.

Other future improvements could be improving the construction algorithm of the acceleration structure, so points are distributed in a better manner between levels. Alternative acceleration structures could also be explored, since the multiresolution structure is not the most optimal acceleration system for tasks not related to visualization.

But certainly, one of the most innovative lines of research in point based rendering, is using virtual reality headsets to visualize these datasets. It has incredible possibilities in the fields of architecture and civil engineering, since it provides professionals with new tools to interact with clients or among them with an unprecedented amount of detail and immersion.

CONCLUSION

In this article, several applications of the out-of-core point management framework PCM were shown. It was designed to deal with arbitrary size datasets; simplifying greatly programming tasks that involve point clouds. From point cloud processing to real-time visualization, PCM is of great help and frees the programmer from dealing with problems like point cloud formats, memory management, point cloud manipulation, acceleration structures, etc. Additionally, the library also exploits the full potential of both modern CPUs and GPUs, facilitating the use of parallel processing to reduce the computation time as much as possible.

Furthermore, having a system that is able to manage such a huge amount of data, will let the user take advantage of the full potential of datasets generated by cutting-edge 3D capture devices. These datasets present great opportunities for a wide range of fields, but these new types of 3D models present a great challenge because of the vast amounts of data they generate. Thanks to the techniques explained, researchers or professionals will be able to manage these massive point clouds transparently using the already mentioned framework.

ACKNOWLEDMENT

This work is financed by the CDTI, Enmacosa and partially supported with FEDER funds. The project in which this work was developed is ToVIAS: Topographic Visualization, Interaction and Analysis System (IDI-20120575). This research has also received funding from the DIVA Marie Curie Action of the People programme of the EU FP7/2007- 2013/ Program under REA grant agreement 290227.

REFERENCES

Eberly, D. (1999). Distance between point and triangle in 3D. *Magic Software, http://www. magic-software. com/Documentation/pt3tri3. pdf*.

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

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

Goswami, P., Erol, F., Mukhi, R., Pajarola, R., & Gobbetti, E. (2013). An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *The Visual Computer, 29*(1), 69-83.

Jaspe, A., Mures, O. A., Padrón, E. J., & Rabuñal, J. R. (2014). *A Multiresolution System for Managing Massive Point Cloud Data Sets*. Technical Report. University of A Coruña.

Kuder, M., Šterk, M., & Žalik, B. (2013). Point-based rendering optimization with textured meshes for fast LiDAR visualization. *Computers & Geosciences, 59*(0), 181-190. doi: http://dx.doi.org/10.1016/j.cageo.2013.05.012

Levoy, M., & Whitted, T. (1985). *The use of points as a display primitive*: University of North Carolina, Department of Computer Science.

Mures, O. A. (2014). ToView point cloud visualizer. from https://youtu.be/cyeOUs0PyNw

Van Genderen, J. (2011). Airborne and terrestrial laser scanning. *International Journal of Digital Earth, 4*(2), 183-184.

Wenzel, K., Rothermel, M., Fritsch, D., & Haala, N. (2014a). FILTERING OF POINT CLOUDS FROM PHOTOGRAMMETRIC SURFACE RECONSTRUCTION. *ISPRS-International*

Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 1, 615-620.

Wenzel, K., Rothermel, M., Fritsch, D., & Haala, N. (2014b). *An out-of-core octree for massive point cloud processing.* Paper presented at the PROCEEDINGS, IQMULUS 1ST WORKSHOP ON PROCESSING LARGE GEOSPATIAL DATA.

Yuan, C. (2012). *High performance computing for massive LiDAR data processing with optimized GPU parallel programming*: THE UNIVERSITY OF TEXAS AT DALLAS.

Zwicker, M., Pfister, H., Van Baar, J., & Gross, M. (2001). *Surface splatting.* Paper presented at the Proceedings of the 28th annual conference on Computer graphics and interactive techniques.

KEY TERMS AND DEFINITIONS

Point cloud: Set of vertices or points in a three-dimensional coordinate system. These vertices are usually positioned in 3D space and have a set of coordinates (x, y, z). These sets of points normally are representative of the external surface of an object.

LiDAR: Remote sensing technology that measures distance to a target by illuminating it with a laser and analyzing the reflected light.

Out-of-core: Type of algorithm that is designed to process data that is too large to fit into a computer's main memory at one time. These algorithms must be optimized to efficiently fetch and access data stored in slow secondary memory such as hard drives.

GPGPU: General-purpose computing on graphics processing units is the utilization of graphics processing units (GPU), which typically handles computer graphics workloads, to perform computation in applications traditionally handled by the central processing unit (CPU).

Multiresolution: A technique that allows you to break an image down into multiple levels (or layers) so that every zoom level has good resolution.

Rendering: The process of generating an image or set of images from a 2D or 3D model, by means of computer software.

Photogrammetry: The science of making measurements from photographs, especially for recovering the exact positions of surface points.