IMPLICIT LOD FOR PROCESSING, VISUALISATION AND CLASSIFICATION IN POINT CLOUD SERVERS

Rémi Cura [AB], Julien Perret [A], Nicolas Paparoditis [A]

[A] Universite Paris-Est, IGN, SRIG, COGIT & MATIS, 73 avenue de Paris, 94160 Saint Mande, France
first_name.last_name@ign.fr
[B] Thales Training & Simulation SAS, 1 rue du Général de Gaulle 95523 Cergy-Pontoise, France

KEY WORDS: RDBMS, point cloud, point cloud management, point cloud processing, filtering, indexing, compression, point cloud storage, point cloud I/O, point cloud generalisation, patch, point grouping, classification, multiscale dimensionality descriptor, preprocessing, LOD, Level Of Detail

1. ABSTRACT

We propose a new paradigm to effortlessly get a portable geometric Level Of Details (LOD) for a point cloud inside a Point Cloud Server. The point cloud is divided into groups of points (patch), then each patch is reordered (MidOc ordering) so that reading points following this order provides more and more details on the patch. This LOD have then multiple applications: point cloud size reduction for visualisation (point cloud streaming) or speeding of slow algorithm, fast density peak detection and correction as well as safeguard for methods that may be sensible to density variations. The LOD method also embeds information about the sensed object geometric nature, and thus can be used as a crude multi-scale dimensionality descriptor, enabling fast classification and on-the-fly filtering for basic classes.

2. INTRODUCTION

2.1 Problem

Point cloud data is becoming more and more common. Following the same trend, the acquisition frequency and precision of the Lidar device are also increasing. Thus point cloud processing is entering in the Big Data realm. Point cloud data usage is also spreading and going out of their traditional user communities. Lidar data are now commonly used by non-specialized users. The now common datasets in the multi billion point range can not fit in computer memory. Furthermore, having the complete and fully detailed point cloud is impracticable, unnecessary, or even damageable for most applications.

The ability to reduce the number of points is then vital for practical point cloud management and usage.

There are basically two levers to reduce the amount of data considered (See Figure 2). The first would be **filtering** data based on its characteristics (position, time, semantic, etc.), thus keeping the original data, but only a portion of it. The second lever would be to **generalise** the data, that is replace many points with few objects that represent well those points. Those objects can be more abstract (geometric primitives like plane for example), or of the same type, i.e. simply some well chosen points (subset).

For instance, to visualize a part of a massive point cloud, we might fetch only the points that are visible (filtering), and not display the totality of selected points, but only points better representing the scene (generalisation).

The Point Cloud Server (PCS) we propose un **?** is compatible with filtering and generalization. It extensively covers the filtering part, with many possibilities (spatial, semantic, attributes, using vector and raster data, using metadata). The PCS also use generalisation of points in the form of more abstract types (bounding box, planes, stats, etc.). In this article, we explore the last option: how to generalise groups of points by choosing a representative subset of points (See Fig. 2).

This kind of problem is common in Geographical Information System (GIS): how to generalize information contained in huge datasets, that is reduce the details of data while retaining its principal organisation.

This problem is also much broader and very commons across research field. It could be seen as compression, clustering, dimensionality reduction, or Level Of Detail (LOD) if the goal is visualisation.

In this article, we consider successive reducion of the amount of points in several levels while preserving the geometric characteristics of the underlying sensed object, in an efficient manner, and while being robust point density variation. The goal is not only visualization of large point cloud, but mainly processing of large point clouds.

Robustness to variation of density is necessary because the sensing may be structured for the sensing device (for instance a Lidar may sense point using a constant angle), but not necessary for the sensed object (see Fig. 3). Furthermore,fusing multiple point clouds also produce non regular density. .

2.2 Related Work

Sophisticated methods have been proposed to generalise 2D points for cartographic applications. Sester (2001) uses Self Organizing Maps, Schwartges et al. (2013) uses Active Range and (mixed) integer linear programming. In both case, the goal is a very specific type of visualisation (cartography), and it obviously relies on having a 2D plan on which perform the methods. Applying directly such methods to point clouds would thus require to have access to surfaces of sensed objects.

Yet, getting this surface (reconstruction) is a very hard challenge, sometime not even possible, and thus we can not rely on it.

Because the goal si to produce hierarchical levels of points, it seems natural to use a hierarchical structure to compute those levels. Rusinkiewicz and Levoy (2000) use a Bounding Sphere Hierarchy for a visualisation application. On the other hand, Octree (Meagher (1982)) have become the de-facto choice. Indeed, they can be build efficiently ordering points by Morton order (Feng and Watanabe (2014)), which is similar in principle to GeoHash

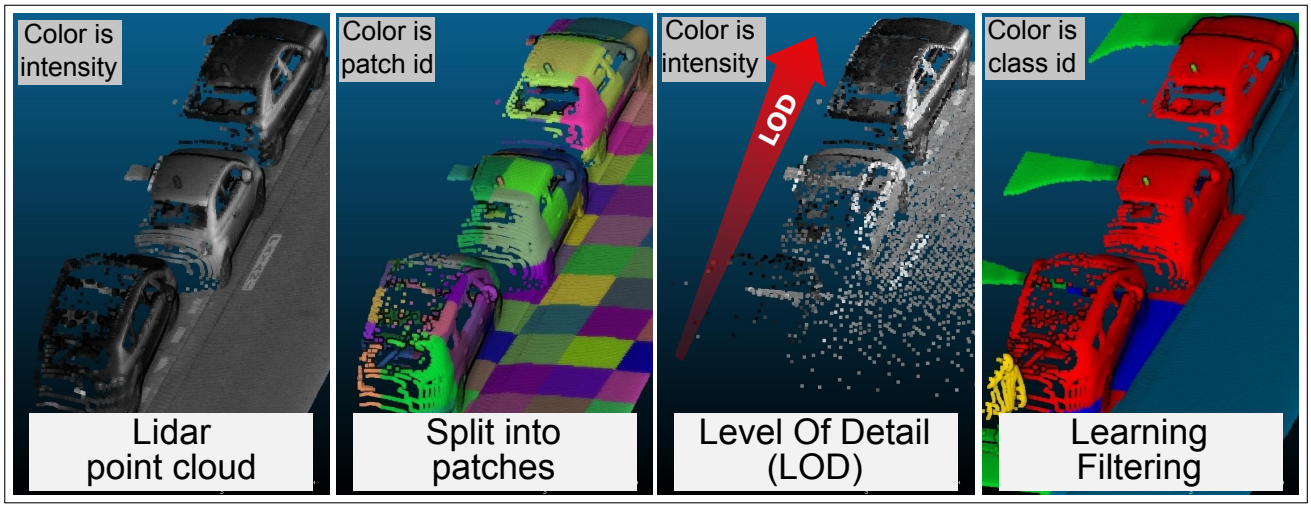


Figure 1: Graphical Abstract : a Lidar point cloud (1), is split it into patches (2) and stored in the PCS (**?**), patches are re-ordered to obtain free LOD (3) (a gradient of LOD here), lastly the ordering by-product is a multiscale dimensionality descriptor used as a feature for learning and efficient filtering (4).

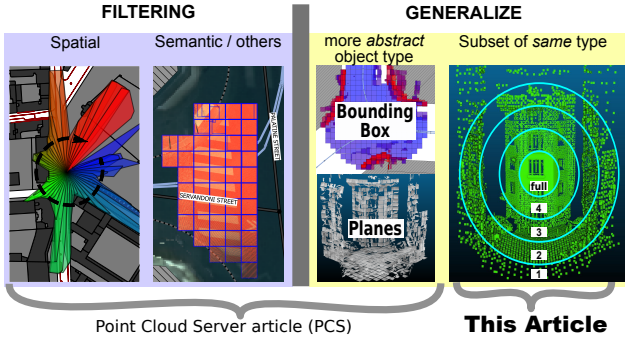


Figure 2: Two strategies to limit the amount of points to work on.

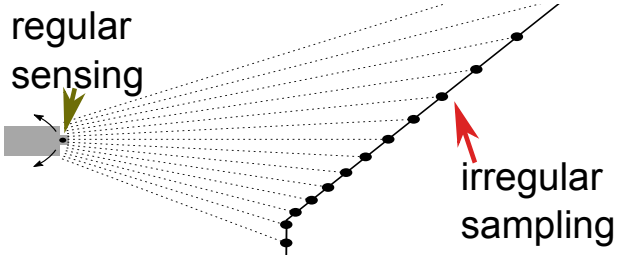


Figure 3: Regular sensing does not imply regular sampling.

(Sabo et al. (2014)). Octree can even be created out of memory for extremely large point clouds (Baert et al. (2014)).

Moreover, their regularity allows efficient representation and compression (Schnabel and Klein (2006); Huang et al. (2006)), as well as fast geospatial access to data (Elseberg et al. (2013)). Octree are also natural condidates to nesting (i.e. create a hierarchy of octrees with various resolution and occupancy, as in Hornung et al. (2013)).

Bereuter (2015) recently gave an overview of how quad tree can be used for point generalisation. The steps are first to compute a tree for the point cloud. Then, the point generalisation at a given level is obtained for each cell of the same tree level, by having one point represent all the points of this cell.

There are two methods to choose a point representing the others. The first one is to select on points among all ('select'). The second method is to create a new point that will represent well the others ('aggregate'). Both these methods can use geometry of points, but also other attributes.

In theory, choosing an optimal point would also depend on application. For instance lets consider a point cloud containing a classification, and suppose the application is to visually identify the presence of a very rarely present class C. In this case a purely geometrical LOD would probably hide C until the very detailed levels. On the opposite, prefering a point classified in C whenever possible would be optimal for this application.

However, a LOD method has to be agnostic regarding point clouds, and point clouds may have many attributes of various type and meaning, as long as many applications. Therefore, most methods use only the minimal common factor of possible attributes, that is spatial coordinates. For visualisation applications, aggregating points seems to be the most popular choice Schütz and Wimmer (2015); Hornung et al. (2013); Elseberg et al. (2013). with aggregating functions like centroids of the points or centroid of the cell.

All of this methods also use an aggregative function (barycentre of the points, centroid of the cell) to represent the points of a cell. Using the barycentre seems intuitive, as it is also the point that minimize the squared distance to other points in the cell, and thus a measure of geometric error.

However, using the 'aggregate' rather than 'select' strategy necessary introduces aggregating errors (as opposed to potential aliasing error), and is less agnostic. Indeed, aggregating means fabricating new points, and also necessitate a way to aggregate for each attributes, which might be complex (for instance semantic aggregating; a point of trash can and a point of bollard could be aggregated into a point of street furniture). This might not be a problem for visualization application. Yet our goal is to provide LOD for other processing methods, which might be influenced by aggregating errors. Furthermore, the barycentre is very sensible to density variations.

Therefore, we prefer to use a 'select' strategy. The point to be selected is the closest to the centroid of the octree cell. If the point cloud density is sufficient this strategy produces a nearly regularly

sampled point cloud, which might be a statistical advantage for processing methods. To establish a parallel with statistics, picking one point per cell is akin to a Latin Hypercube (see McKay et al. (1979)). Avoiding the averaging strategy might also increase the quantity of information than can be retrieved (similar to compressed sensing, see Fornasier and Rauhut (2010)).

We note that most of the LOD systems seems to have been created to first provide a fast access to point (spatial indexing), and then adapted to provide LOD. Using the PCS, we can separate the indexing part, and the LOD scheme. From this stems less design constraints, more possibilities, and a method than is not dedicated to only one application (like visualisation).

2.3 Contribution

This paper re-uses and combines existing and well established methods with a focus on simplicity and efficiency. As such, all the methods are tested on billions scale point cloud, and are Open Source for sake of reproducibility test and improvements.

Our first contribution is to store the LOD implicitly in the ordering of the points rather than externally, avoiding any data duplication. Thus, we don't duplicate information, and the more we read points, the more precise of an approximation of the point cloud we get. If we read all the points, we have the original point cloud.

The second contribution (MidOc) is a simple way to order points in order to have an increasingly better geometric approximation of the point cloud when following this order.

The third contribution is to show that this ordering embed information about the dimensionality of the sensed object, to the point of being a simple multi-scale dimensionality descriptor. We demonstrate the interest of this descriptor by comparing it to a state of the art dimensionality descriptor, then by assessing it potential by performing a Random Forest classification that can then be used for very fast pre-filtering of points, and other applications.

2.4 Plan of the article

The rest of this article is organized as follows: in the next section 3. we present the methods. In the result section 4. we give the results. We discuss it and the possible limitations in section 5..

Following the IMRAD format (Wu, 2011), the remainder of this article is divided into three sections. Section 3. presents the LOD solution, how it produces a dimensionality descriptor, and how this can leveraged for classification. Section 4. reports on the experiments validating the methods. Finally, the details, the limitations, and potential applications are discussed in Section 5..

3. METHOD

In this section, we first present the Point Cloud Server (section 3.1)(PCS **?**) upon which this article is based. Then we introduce the LOD solution that we propose , which consists of reordering groups of points from less to more details (3.2), and then choose which LOD is needed. Although any ordering can be used, we propose a simple geometric one (3.3) which is fast and robust to density variation. Furthermore, constructing this ordering produces a rough dimensionality descriptor (3.4). This descriptor can be used in the PCS to perform density correction (3.5) and classification at the patch level (3.6). This classification can be directly transferred to points, or indirectly exploited in a pre-filtering step.

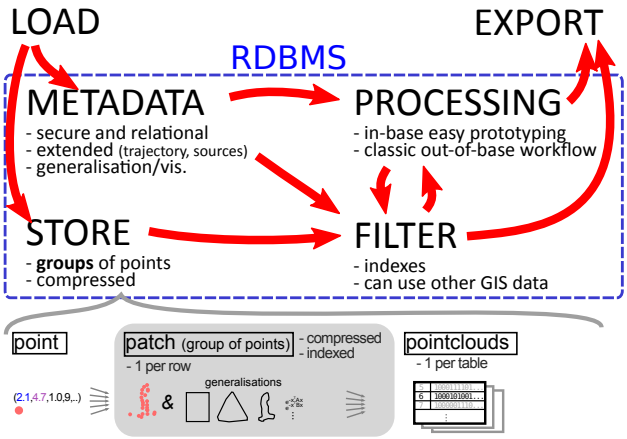


Figure 4: Overall and storage organisations of the Point Cloud Server.

3.1 The Point Cloud Server

This article strongly depends on using the Point Cloud Server described in **?**, therefore we introduce its principle and key relevant features (see figure 4).

The PCS is a complete and efficient point cloud management system based on a database server that works on groups of points rather than individual points. This system is specifically designed to solve all the needs of point cloud users: fast loading, compressed storage, powerful filtering, easy data access and exporting, and integrated processing.

The core of the PCS is to store groups of points (called patches) that are multi-indexed (spatially, on attributes, etc.), and represented with different generalisation depending on the applications. Points can be grouped with any rules. In this article, the points are regrouped spatially by cubes 1m (Paris) or 50m (Vosges) wide.

All the methods described in this article are applied on patches. We propose is to reorder each patch following the MidOc ordering, allowing LOD and producing a dimensionality descriptor per patch. It can then be used to classify patches.

We stress that our method used on any point cloud will provide LOD, but that using it with the PCS is much more interesting, and adds key feature such as spatial indexing, fast filtering, etc.

3.2 Exploiting the order of points

We propose to exploit the ordering of points to indirectly store LOD information. Indeed, whatever the format, be it file or database based, points ends up as a list, which is ordered.

The idea is then to exploit the order of this list, so that when reading the points from beginning to end, we get gradually a more accurate geometrical approximation of the point cloud (see figure 5).

For instance, given list $L[P_1, .., P_N]$ of ordered points. Reading P_1 to P_5 gives a rough approximation of the point cloud, and reading another 16 points (P_1 to P_{21}) is going to give a slightly better approximation. Reading points 1 to N is going to get the exact point cloud, so there is no data loss, nor data duplication.

Using the point ordering as LOD results in three main advantages.

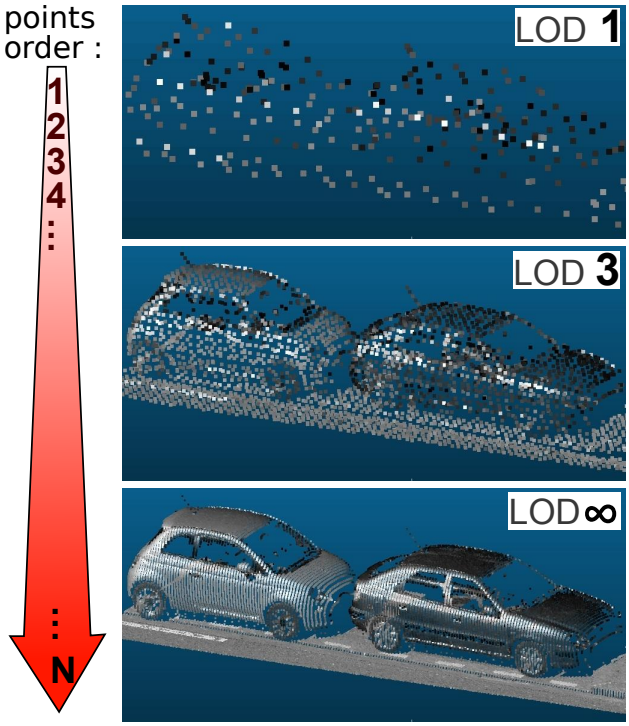


Figure 5: 3 Geometrical Level Of Detail (LOD) for the same point cloud. Reading points from 1 to N gradually increases the details, because of the specific order of points (MidOc).

Implicit Except a pre-processing step to write the point cloud following a given ordering, each time the user wants to get a Level Of Detail version of the point cloud, there is no computing at all (only data reading). This may not make a big difference for non-frequent reading, but in a context where the same point cloud is going to get used several times at several levels and by several users simultaneously (for instance Point Cloud as a Service), no processing time makes a big difference.

No Duplication Another big advantage is that exploiting point ordering does not necessitate additional storage. This is an advantage on low level. It saves disk space (no data duplication, no index file). Because the LOD information is embedded right within the point cloud, it is perfectly concurrent-proof, i.e. the point cloud and the LOD can not become out of sync. (Even in heavy concurrent Read/Write, a user would always get a coherent LOD). Lastly because the LOD only relies on ordering the original points, and does not introduces any other points or data, it avoids all precision-related issues that may come from aggregating.

Portable The last advantage comes from the simplicity of using the ordering. Because it is already something that all point cloud tools can deal with (a list of points!), it is portable. Most softwares do not change the points order inside a cloud (See Section 4.3). Even if a tool were to change the order, it would be easy to add the ordering number as an attribut (though slightly increasing the storage requirement). This simplicity also implies that adapting tools to exploit this ordering is very easy.

3.3 MidOc : an ordering for gradual geometrical approximation

3.3.1 Requirements and hypothesis The method exploits the order of points to store LOD information, so that the more points are read, the more detailed the result becomes. Obviously an ordering method that class the points from less details (LOD_0) to full details(LOD_∞) is needed. This ordering is in fact a geometric measure of point relevance, that is how well a point represents the point cloud (in a neighbourhood depending of the LOD).

This ordering will be used by on different point clouds and for many applications, and so can not be tailored to one. As such, we can only consider the geometry (the minimal constituent of a point). Because of the possible varying-density point clouds, the ordering method also have to recreate a regular-ish sampling.

Although many ordering could be used (for example, a simple uniform-random ordering), a suitable one would have low-discrepancy (that is be well homogeneous in space, see Rainville et al. (2012)), not be sensitive to density variations, be regular, be fast to compute and be deterministic (which simplify the multiuser use of the point cloud).

We make two hypothesis that are mostly verified on Lidar point cloud. The first hypothesis ('disposable density') is that the density does not gives information about the nature of the object being sensed. That is, depending on the sensing situation, some parts of the cloud are more or less dense, but this has nothing to do with the nature of the object sensed, thus can be discarded. The second hypothesis (low noise) is that the geometrical noise is low. We need this hypothesis because 'disposable density' forbids to use density to lessen the influence of outliers.

A common method in LOD is to recursively divide a point cloud into groups and use the barycentre of the group as the point representing this group. The ground of this method is that the barycentre minimise the sum of squarred distance to the points.

However such method is extremely sensible to density variation, and artificially creates new points.

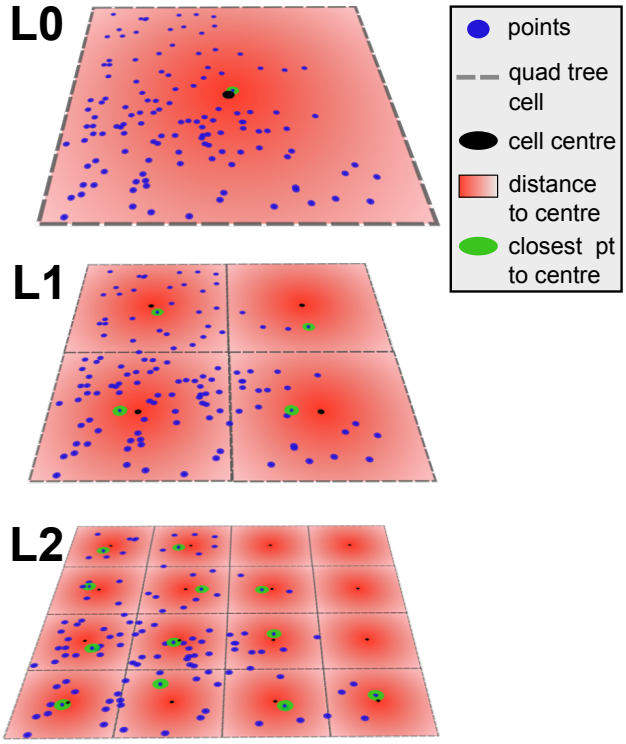


Figure 6: MidOc explained in 2D. Given a point cloud (Blue) and quad tree cells (dashed grey), the chosen point (green ellipse) is the one closest to the centre (black point) of the cell.

3.3.2 Introducing the MidOc ordering We propose the reuse of well known and well proven existing methods that is the

octree subsampling (for instance, the octree subsampling is used in Girardeau-Montaut (2014)). An octree is built over a point cloud, then for each cell of the octree the LOD point is the barycentre of the points in the cell. With this, browsing the octree breadth-first provides the points of the different levels.

We adapt this to cope with density variation, and to avoid creating new point because of aggregation. We name this ordering MidOc (Middle of Octree subsampling) for clarity of this article, nonetheless we are probably not the first to use it.

The principle is very simple, and necessitate an octree over the point cloud (octree can be implicit though). We illustrate it on Figure 6 (in 2D for graphical comfort). We walk the octree breadth-first. For each non-empty cell, the point closest to the cell centre is chosen and assigned the cell level, and removed from the available point to pick. The process can be stopped before having chosen all possible points, in which case the remaining points are added to the list, with the level L_∞.

The result is a set of points with level (P, L_i). Inside one level L_i, points can be ordered following various strategies (see Section 3.3.4).

Because each point is assigned a level, we can store the total number of points per level, which is a multi-scale dimensionality descriptor, see Section 3.4).

3.3.3 Implementation MidOc ordering is similar to octree building. Because Octree building has been widely researched, we test only two basic solutions among many possibilities.

The first kind of implementation uses SQL queries. For each level, we compute the centres of the occupied cells using bit shifts and the closest point to these. Picked points are removed, and the process is repeated on the next level. It relies on the fact that knowing each point octree cell occupancy does not require to compute the octree (see Figure 21).

The second implementation uses python with a recursive strategy. it only necessitates a function that given a cell and a list of points chose the point closest to the centre of the cell, then split the cell and the list of points for the next level, and recursively calls itself on this subcells with the sublists.

A more efficient and simpler implementation is possible by first ordering the points following the Morton (Hypothesis : or Hilbert) curve, as in Feng and Watanabe (2014) (Section 2.5.1, page 37), in the spirit of linear octree.

3.3.4 Intra-level ordering Inside one LOD points can be ordered with various methods. The intra-level ordering will have an impact if the LOD is used in a continuous way, and moreover may influence methods that relies on low-discrepancy. More precisely, if only a part of the points in a level are going to be used, it may be essential that they cover well the spatial layout of the totality of points. Several methods give this kind of coverage (see Rainville et al. (2012))

Lets take the example where the goal is to find the plan that best fits a set of points and the method to do so in online (for instance it could be based on online robust PCA like in (Feng et al. (2013))). The plan fitting method reads one by one the points of a designated level L_i, and successively computes a better plan estimation.

The Figure 7 presents some possible ordering. If the plan detection method was applied on the Y ordering, it would necessitate a great number of points to compute a stable plan. For instance the first 16

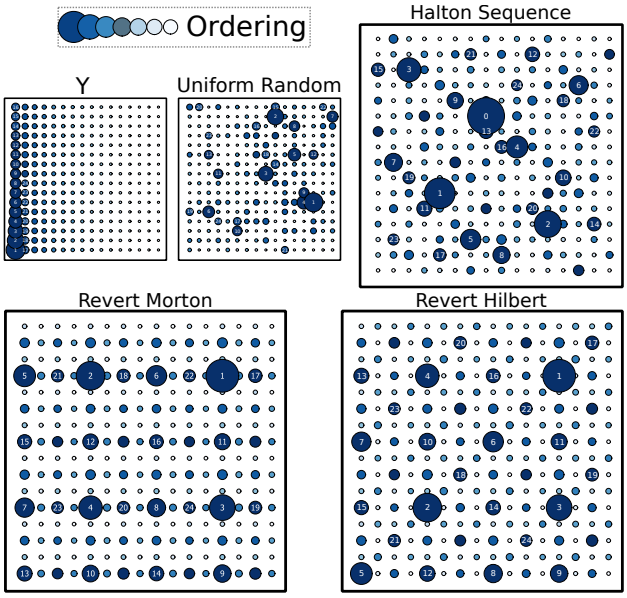


Figure 7: Several possible intra-level orders with various coverage from bad to good. Revert Morton and Revert Hilbert have offset for illustration.

points (1 column) would not permit to compute a plan. Similarly, if the point were ordered randomly, estimating a plan would still require lots of points, because uniform randomly chosen points are not well spread out (on the figure, the first 25 points are over represented in the upper left part).

On the opposite, using a low discrepancy ordering like the Halton sequence makes the points well spread, while being quasi-random. Inverted space filling curves like the Morton or Hilbert curves also cover well space, at the price of being much more regulars.

The Halton sequence ordering is obtained by generating a Halton sequence (nD points) and successively pick points closest to the Halton generated points. The revert Morton ordering and revert Hilbert ordering are the distance along Morton or Hilbert curve expressed in bit and read backward (with a possible offset).

3.4 Crude multi-scale dimensionality descriptor (MidOc by-product)

3.4.1 Principle During MidOc building process, the number of chosen points per level can be stored. Each ordered patch is then associated with a vector of number of points per level $ppl = (N_{L_1}, .., N_{L_{\max}})$. The number of picked point for L_i is almost the voxel occupancy for the level i of the corresponding octree. Almost, because in MidOc points picked at a level do not count for the next Levels. Occupancy over a voxel grid has already been used as a descriptor (See Bustos et al. (2005)). However we can go a step further. For the following we consider that patches contain enough points and levels are low enough so that the removing of picked points has no influence.

In theory for a level L_i, a centred line would occupy 2^{L_i} cells, a centred plan 4^{L_i} cells, and a volume all of the cells (8^{L_i} cells). Thus, by comparing $ppl[L_i]$ to theoretical $2^{L_i}, 4^{L_i}, 8^{L_i}$ we retrieve a dimensionality indice $Dim_{LOD}[i]$ about the dimensionality of the patch at the level L (See Figure 8). This occupancy is only correctly estimated when the patch is fully filled and homogeneous. However, we can also characterize the dimensionality $Dim_{LODDiff}$ by the way the occupancy evolves (difference

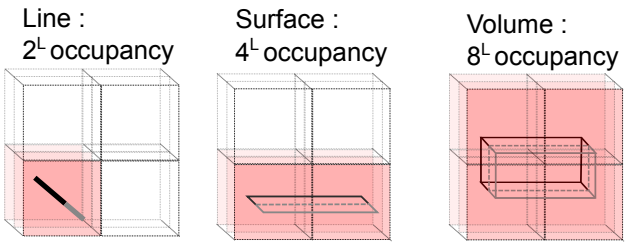


Figure 8: Voxel occupancy is a crude dimensionality descriptor: 3D line, surface or volume occupy a different amount of voxels.

mode). Indeed, a line occupying k cells of an octree at level L_i will occupy $2 * k$ cells at the level L_{i+1}, if enough points.

We stress that the information contained in *ppl* is akin to a multi-scale dimensionality indice, with the scale being the level of the octree. For the rest of this article we consider that the dimensionality is roughly the same across level (which is entirely false in some case, see 4.6.1).

The Figure 9 illustrate this. Typical parts of a street in the Paris dataset were segmented: a car, a wall with window, a 2 wheelers, a public light, a tree, a person, poles and piece of ground including curbs.
Due to the geometry of acquisition and sampling, the public light is almost a 3D line, resulting in the occupation of very few octree cells. A typical number of points chosen per level for a public light patch would then be $(1, 2, 4, 8, ...)$, which looks like a 2^L function. A piece of ground is often extremely flat and very similar to a planar surface, which means that the points chosen per level could be $(1, 4, 16, 64...)$, a 4^L function. Lastly a piece of tree foliage is going to be very volumetric in nature, due to the fact that a leaf is about the same size as point spacing and is partly transparent to laser (leading to several echo). Then a foliage patch would typically be $(1, 8, 64...)$ (if enough points), so a 8^L function. (Tree patches are in fact a special case, see 4.6.1).

3.4.2 Comparing crude dimensionality descriptor with covariance - based descriptors A sophisticated per-point dimensionality descriptor is introduced in Demantké (2014); Weinmann et al. (2015), then used to find optimal neighbourhood size. A main difference is that this feature is computed for each point (thus is extremely costly to compute), and that dimensionality is influenced by density variation.

At the patch level, we do not need to find the scale at which compute dimensionality, the descriptor is computed on the whole patch.

This dimensionality descriptors (Dim_{cov}) relies on computing covariance of points centred to the barycentre (3D structure tensor), then a normalisation of covariance eigen values. As such, the method is similar, and has the same advantages and limitation, as the Principal Component Analysis (See Shlens (2014) for a reader friendly introduction). It can be seen as fitting an ellipsoid to the points.

First this method is sensible to density variations because all the points are considered for the fitting. As opposite to our hypothesis (See Section 3.3.1), this method considers implicitly that density holds information about the nature of sensed objects. Second, this methods only fits one ellipse, which is insufficient to capture complex geometric forms. Last, this method is very local and does not allow to explore different scale for a point cloud as a whole. Indeed this method is classically used on points within a growing sphere to extend the scale.

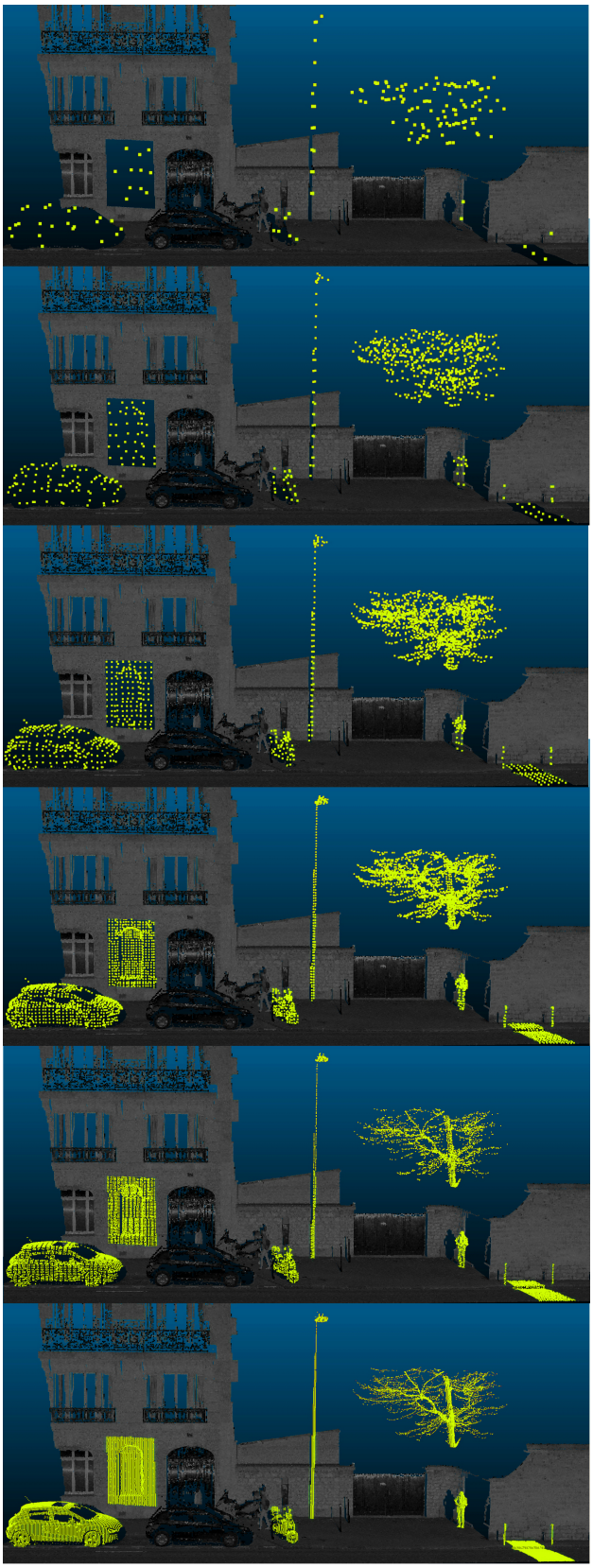


Figure 9: All successive levels for common objects (car, window, 2 wheelers, light, tree, people, pole, ground), color is intensity for other points.

However scale should be defined as the size of the features being analysed in sensed objects, and ot the scale of the neighbourhood

of the centroid.

We compute both dimensionality descriptor and then compare them for the Paris dataset.

3.4.3 crude dimensionality descriptor as a feature Using the result of the MidOc ordering has the advantage of not necessitate extra computing, the patch being ordered with MidOc for LOD anyway.

Moreover, because $x_1 \rightarrow (2^1)^x$, $x_2 \rightarrow (2^2)^x$, $x_3 \rightarrow (2^3)^x$ diverge very fast, we only need to use few levels to have a quite good descriptor. For instance, using $L = 2$, we have $x_i = [4, 16, 64]$, which are very distinguishable values, and don't require a total density above 70 points per patch. As long as the patch contains a minimal number of points, the descriptors is density and scale invariant. Lastly a mixed result (following neither of the $x_i \rightarrow (2^i)^x$ function) can be used as an indicator that the patch contains mixed geometry, either due to nature of the objects in the patch, or due to the way the patch is defined (sampling).

Although it might be possible to go a step further and decompose a patch ppl vector on the base of $x_i \rightarrow (2^i)^x, i \in [1..3]$, the direct and exact decomposition can't be used because the decomposition might depends on L_i. For instance a plane with a small line could appear as a plan for L_1 and L_2, and starts to appear differently over L_3 and higher level. In this case, an Expectation-Maximization scheme might be able to decompose robustly.

3.5 Excessive Density detection and correction

Lidar point cloud do not have a constant density, even if the acquisition is performed at a constant sensing rate, because the sensed object geometry (See Fig. 3).

Important variation of density can be a serious issue for some processing methods. For instance if millions of points are concentrated in a small volume, a processing method operating on fixed size volume may exceed the maximum memory of the system. Large density variation are also bad for performances in parallel environment. Indeed, efficient parallel computing may require that all the workers have about the same amount of work. One worker stumbling upon a very dense part of the point cloud would have much more points to process than the other workers. The figure 12 shows a place in the Paris dataset where the density is 5 times over the average value of this data set. In this context of terrestrial Lidar, this density peak is simply due to the fact that the acquisition vehicle stopped at this place , while continuing to sense data.

The PCS coupled with LOD patches allows to quickly find abnormally high density. The PCS filters in few milliseconds the patch containing lots of points. This suffice for most applications. For a finer density estimation, we compute the approximate volume of the patch. For a level L, the $ppl[L]$ number of points multiplied by the theoretical cell size for this level gives an approximate volume (or surface) of the patch. The total number of points divided by this volume (surface) gives a finer volumetric (surface) density estimation.

Then, correcting density consists of taking into account only the first K points, where K is computed to attain the approximate patch volume (surface).

3.6 Classification with the Point Cloud Server

3.6.1 Principle We propose to perform patch classification using the Point Cloud Server and the previously introduced crude multi-scale dimensionality descriptor, along with other basic descriptors, using a Random Forest classifier. Following the position of the PCS towards abstraction, the classification is performed at the patch level and not at the point level. This induces a massive scaling and speeding effect, at the cost of introducing quantization error. Indeed, compared to a point classification, a patch may contain points belonging to several classes (due to generalisation), yet it will only be classified in one class, thus the "quantization" error.

Because patch classification is so fast and scales so well, the end goal can be however slightly different than for usual point classification.

Patch classification can be used as a fast preprocess to another slower point classification, be it to speed it (an artificial recall increase for patch classification may be needed, see Figure 20), or to better a point classification. The patch classification can provide a rough classification. Based on that the most adapted point classifier is chosen (similarly to Cascaded classifiers), thus improving the result of the final point classification. For instance a patch classified as urban object would lead to chose a classifier specialized in urban object, and not the general classifier. This is especially precious for classes that are statistically under-represented.

Patch classification may also be used as a filtering preprocess for applications that only require one class. Many applications only need one class, and do not require all the points in it, but only a subset with good confidence. For this it is possible to artificially boost the precision (by accepting only high confidence prediction). For instance computing a Digital Terrain Model (DTM) only requires ground points. Moreover, the ground will have many parts missing due to objects, so using only a part of all the points will suffice anyway. The patch classifier allow to find most of the ground patch extremely fast. Another example is registration. A registration process typically require reliable points to perform mapping and registration. In this case there is no need to use all points, and the patch classification can provide patches from ground and façade with high accuracy (for point cloud to point cloud or point cloud to 3D model registration), or patches of objects and trees (for points cloud to landmark registration). In other applications, finding only a part of the points may be sufficient, for instance when computing a building map from façade patches.

Random Forest method started with Amit and Geman (1997), theorized by Breiman (2001) and has been very popular since then. They are for instance used by Golovinskiy et al. (2009) who perform object detection, segmentation and classification. They analyse separately each task on an urban data set, thus providing valuable comparison. Their method is uniquely dedicated to this task, like Serna and Marcotegui (2014) who provide another method and a state of the art of the segmentation/classification subject. Both of this methods are in fact 2D methods, working on an elevation image obtained by projecting the point cloud. However we observe that street point clouds are dominated by vertical surfaces, like building (about 70% in Paris data set). Our method is fully 3D and can then easily be used to detect vertical object details, like windows or doors on buildings.

3.6.2 Classification details

Features The first descriptor is ppl, the crude multi-scale dimensionality descriptor produced by the MidOc ordering (see Section 3.4). We use the number of points for the level $[1..4]$. For each level L, the number of points is normalized by the maximum number of points possible (8^L), so that every feature is in $[0, 1]$.

We also use other simple features that require very limited computing (avoiding complex features like contextual features). Due

to the PCS patch compression mechanism, min, max, and average of any attributes of the points are directly available. Using the LOD allows to quickly compute other simple feature, like the 2D area of points of a patch (points being considered with a given diameter).

Dealing with data set particularities The Paris data set classes are organized in a hierarchy (100 classes in theory, 22 populated). The rough patch classifier is not designed to deal with so many classes, and so a way to determine what level of hierarchy will be used is needed. We propose to perform this choice with the help of a graph of similarity between classes (See Fig. 15 and 16

We first determinate how similar the classes are for the simple dimensionality descriptors, classifying with all the classes, and computing a class to class confusion matrix. This confusion matrix can be interpreted as an affinity between class matrix, and thus as a graph. We draw the graph using a spectral layout (Networkx (2014)), which amounts to draw the graph following the first two eigen vector of the matrix (Similar to Principal Component Analysis). Those two vectors maximize the variance of the data (while being orthogonal), and thus best explain the data. This graph visually helps to choose the appropriate number of classes to use. A fully automatic method may be used via unsupervised clustering approach on the matrix (like The Affinity Propagation of Frey and Dueck (2007)).

Even when reducing the number of classes, the Paris dataset if unbalanced (some class have far less observations than some others). We tried two classical strategies to balance the data set regarding the number of observation per class. The first is under-sampling big classes : we randomly under-sample the observations to get roughly the same number of observation in every class.

The second strategy is to compute a statistical weight for every observation based on the class prevalence. This weight is then used in the learning process when building the Random Forest.

3.6.3 Using the confidence from the classifier

Contrary to classical classification method, we are not only interested in precision and recall per class, but also by the evolution of precision when prediction confidence varies.

In fact, for a filtering application, we can leverage the confidence information provided by the Random Forest method to artificially boost precision (at the cost of recall diminution). We can do this by limiting the minimal confidence allowed for every prediction. Orthogonaly, it is possible for some classes to increase recall at the cost of precision by using the result of a first patch classification and then incorporate in the result the other neighbour patches.

We stress that if the goal is to detect objects (and not classify each point), this strategy can be extremely efficient. For instance if we are looking for objects that are big enough to be in several patches (e.g. a car). In this case we can perform the classification (which is very fast and efficient), then keep only highly confident predictions, and then use the position of predictions to perform a local search for car limits. The classical alternative solution would be to perform a per point classification on each point, which would be extremely slow.

4. RESULT

4.1 Introduction to results

We design and execute several experiments in order to validate all points introduced in Section 3.. First we prove that is it effectively possible to leverage points order, even using canonical open sources software out of the box. Second we perform MidOc ordering on very large point cloud and analyse the efficiency, quality and applications of the results. Third we use the number of points chosen in the MidOc ordering as a descriptors for a random forest classifier on two large data sets, proving their usefulness. Last we analyse the potential of this free descriptors, and what it brings when used in conjunction to other simple descriptors.

The base DBMS is PostgreSQL (2014). The spatial layer PostGIS (2014) is added to benefits from generic geometric types and multidimensional indexes. The specific point cloud storage and function come from pgPointCloud (2014). The MidOc is either plpgsql or made in python with SciPy (2014). The classification is done with Scikit (2014), and the network clustering with Networkx (2014). Timings are only orders of magnitude due to the influence of database caching.

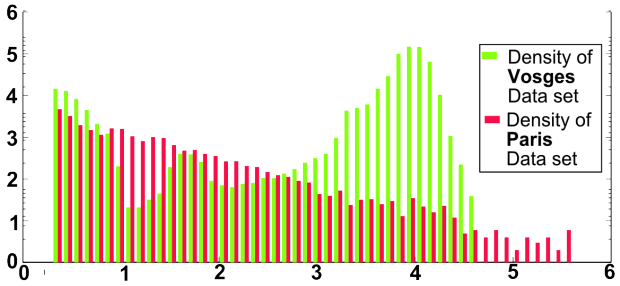


Figure 10: Histogram of number of points per patch, with a logarithmic scale for X and Y axis

We use two data sets. There were chosen as different as possible to further evaluate how proposed methods can generalise on different data (See Figure fig:hist-density-dataset for histogram of patch density). The first data set is IQmulus (2014) (Paris data set), an open source urban data set with varying density, singularities, and very challenging point cloud geometry. Every point is labelled with a hierarchy of 100 classes. The training set is only 12 millions points. Only 22 classes are represented. We group points in $1\mathrm{m}^3$ cubes. The histogram of density seems to follow an exponential law (See figure 10), the effect being that many patches with few points exist.

We also use the Vosges data set, which is a very wide spread, aerial Lidar, 5.5 Billions point cloud. Density is much more constant at 10k pts/patch . A vector ground truth about surface occupation nature (type of forest) is produced by the French Forest Agency. Again the classes are hierarchical, with 28 classes. We group points in 50 $\times 50\mathrm{m}$ squares.

4.2 Using the Point Cloud Server for experiments

All the experiments are performed using a Point Cloud Server (cf Cura (2014)). The key idea are that point clouds are stored inside a DBMS (postgres), as patch. Patch are compressed groups of points along with some basic statistics about points in the group. We hypothesize that in typical point cloud processing workflow, a point is never needed alone, but almost always with its surrounding points.

Each patch of points is then indexed in an R tree for most interesting attributes (obviously X,Y,Z but also time of acquisition, meta data, number of points, distance to source, etc.)

Having such a meta-type with powerful indexes allows use to find points based on various criteria extremely fast. (order of magnitude : ms). As an example, for a 2 Billion points dataset, we

can find all patches in few milliseconds having : - between -1 and 3 meters high in reference to vehicle wheels - in a given 2D area defined by any polygon - acquired between 8h and 8h10 - etc.

The PCS offers an easy mean to perform data-partition based parallelism. We extensively use it in our experiments.

4.3 Exploiting the order of points

We proposed to implicitly store LOD in the order of the points (Section 3.2). In this first experiment we check that point cloud ordering is correctly preserved by common open source point cloud processing software. For this, we use a real point cloud, which we order by MidOc ordering. We export it as a text file as the reference file. For each software, we read the reference file and convert it into another format, then check that the conversion did not change the order of points. The tree common open source software tested are CloudCompare[1] , LasTools[2] and Meshlab[3]. All pass the test.

4.4 MidOc: an ordering for gradual geometrical approximation

We first test the visual fitness of MidOc ordering. Then we compute MidOc for our two datasets and evaluate the trade-off between point cloud size and point cloud LOD. As a proof of concept we stream a 3D point cloud with LOD to a browser.

The figure 11 illustrates LOD on a typical street of Paris dataset The figure 9 shows LOD on common street objects of various dimensionality.

We compute the size and canonical transfer time associated for a representative street point cloud. For this order of magnitude, the size is estimated at 5*4 Byte (5 floats) per point, and the (internet) transfer rate at 1 $\mathrm{M}byte/\mathrm{s}$.

Table 1: Number of points per LOD for the point cloud in the Figure 11 , plus estimated transfer time at 1 $\mathrm{M}byte/\mathrm{s}$.

Level	Typical spacing (cm)	Points number (k)	Percent of total size	Estimated time (s)
All	0.2 to 5	1600	100	60
0	100	3	0.2	0.1
1	50	11.6	0.7	0.4
2	25	41	2.6	1.5
3	12	134	8.4	5
4	6	372	23	14

We use 3 implementations of MidOc, two being pure plpgsql (postgreSQL script langage), and one Python (See Section 3.3.3). We successively order all the Paris and Vosges data sets with MidOc, using 20 parallel workers, with a plpgsql implementation. The ordering is successful on all patches, even in very challenging areas where there are big singularities in density, and many outliers. The total speed is about 100 millions points/hour using in-base processing. We prototyped an out-of-base processing where the extraction of points from patch is done on the client, and reached a 180 Mpts /h. The same method, without any ordering (only converting patch to point then point to patch) reach a 2.3 B pts/h. We consider it to be at least 10 times too slow for practical use. We briefly analyse performances, and conclude that only 10 workers are efficient.

As a proof of concept we stream points from the data base to a browser IGN (2014). Because patch may contain a large number of points and because the browser WebGL technology is limited in how much points it can display, we limit the number of points per patch sent to the browser using LOD. Patch are ordered with MidOc, so the visual artifact is greatly reduced, and the data loads more quickly, as expected.

4.5 Excessive Density detection and correction

We detect the abnormal density (explained in Section 3.5) in the Paris data set in $\sim 100\mathrm{ms}$ (See Figure 12). In comparison, computing the density per point with neighbourhood is extremely slow (only for this 1.5 Million extract, 1 minute with CloudCompare,4x2.5GHz, 10cm rad) (top right illustration), and after the density is computed for each points, all the point cloud still need to be filtered to find abnormal density spot.

If the patch are ordered following MidOc, unneeded points are removed by simply putting a threshold on points per patch (bottom left, 1 to 5k points $/\mathrm{m}^3$, bottom right , 5k to 24 k pts $/\mathrm{m}^3$). It considerably reduces the number of points (-33%).

This strategy can be automated by stating than no patch should return points over Level L_i. Then when getting points from the PCS, so that only points in those levels are sent.

4.6 Rough Dimensionality descriptor

We test the dimensionality descriptor (ppl) in two ways. First we compare the extracted (Dim_{LOD}) to the classical structure tensor based descriptor (Dim_{cov}). Second we assess how useful it is for classification, by analysing how well it separates classes, and how much it is used when several other features are available.

4.6.1 Comparing LOD-based descriptor with Structure tensor-based descriptor

We compute Dim_{cov} following the indications of Weinmann et al. (2015) to get $p_{dim} -> [0..1]^3$, i.e. the probability to belong to [1D,2D,3D]. We convert this to Dim_{cov} with $Dim_{cov} = \sum_{i=1}^{3} i * p_{dim}[i]$.

Optionally, we test a filtering option so that the maximum distance in biggest two dimensions is more equivalent. However this approach fails to significantly improve results.

We test several method to extract Dim_{LOD} from ppl. The first method is to compute $Dim_{LODs}[i] = log2(ppl[i])/i$, which gives the simple dimensionality indice for each level. The second method is the same but work on occupancy evolution, with $Dim_{LODd}[i] = log2(ppl[i]/ppl[i-1])$ (discarding L_0). In both case the result is a dimensionality indice between 0 and 3 for each Level. We use both indices to fusion the dimensionality across Levels (working on $Dim_{LODA} = Dim_{LODs} \bigcup Dim_{LODd}$). The first method uses a RANSAC (SciPy (2014) implementation of Choi et al. (2009)) to find the best linear regression. The slope gives an idea of confidence (ideally, should be 0), and the value of the line at the middle of abscissa is an estimate of Dim_{LOD}. The second method robustly filters Dim_{LODA} based on median distance to median value and average the inliers to estimate Dim_{LOD}.

Dim_{cov} and Dim_{LOD} are computed with in-base and out-of-base processing, the latter being executed in parallel (8 workers). For 10k patches, 12 Mpts, retrieving data and writing result accounts for 48s, computing Dim_{LOD} to 8s, Dim_{cov} to 64s. Computing ppl (which is multiscale) using a linear octree takes between 58 L_6 and 85s L_8. (

Comparing Dim_{cov} and Dim_{LOD} is not straightforward because the implicit definition of dimension is very different in the two methods. We analyse the patches where $|Dim_{LOD} -$

[1] www.danielgm.net/cc
[2] www.cs.unc.edu/~isenburg/lastools
[3] http://meshlab.sourceforge.net/

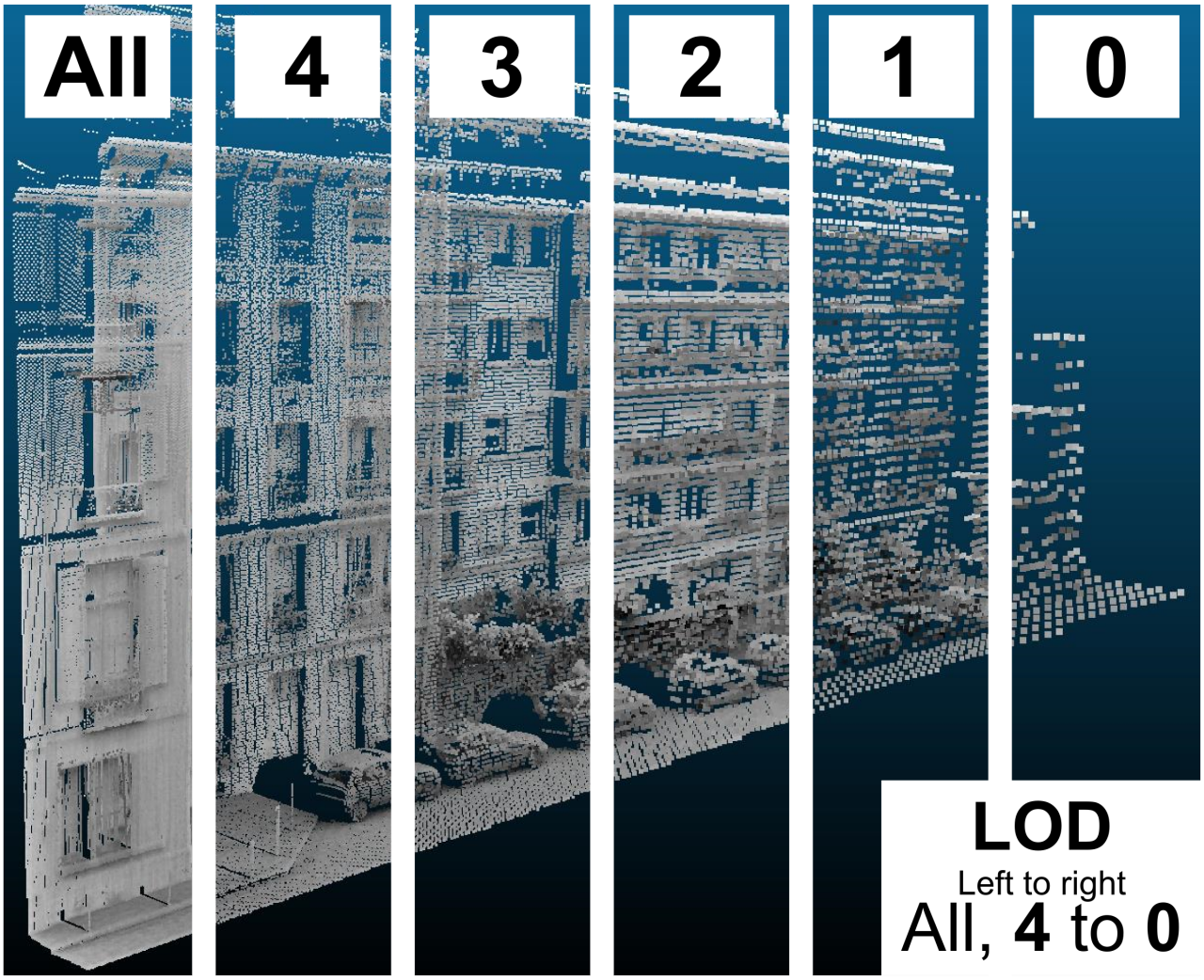


Figure 11: Schematic illustration of different LOD. Left to right, all points, then LOD 4 to 0. Visualized in cloud compare with ambient occlusion. Point size varies for better visual result.

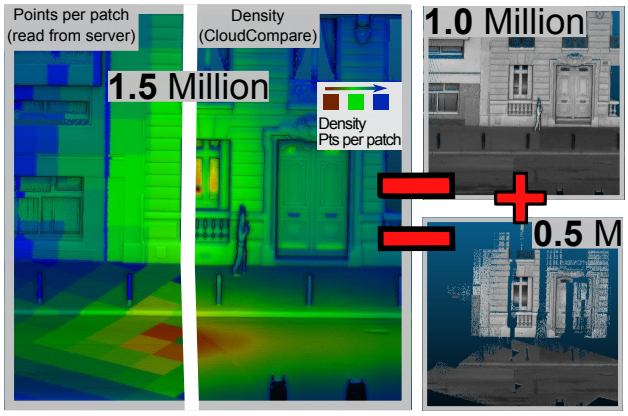


Figure 12: Abnormal density detection and correction. Top points per patch (left) or density (right), green-yellow-red. Bottom reflectance in grey.

$Dim_{cov}| <= 0.5$. 0.5 is an arbitrary threshold, but we feel that it represents the point above which descriptors will predict unreconcilable dimensions. Those patches represent 93% of the data set (0.96 % of points), with a correlation of 0.80. Overall the proposed

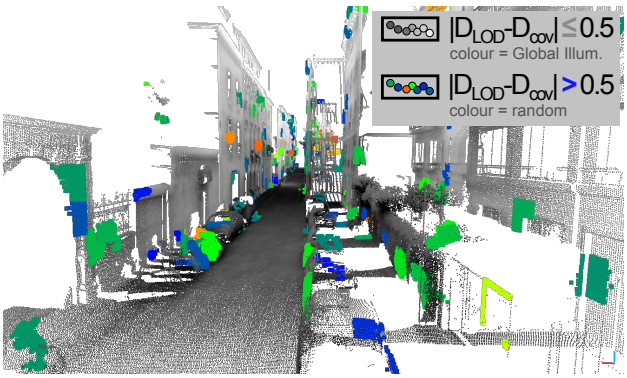


Figure 13: Dim_{LOD} and Dim_{cov} are mostly comparable, except for few patches (5%, coloured)

dimensions are similar for the majority of patch, especially for well filled 1D and 2D patches (See Fig. 13).

We analyse the 684 remaining patches to look for possible explanations of the difference in dimension (See Fig. 14).

We consider the following four main sources of limitations from Dim_{cov}.

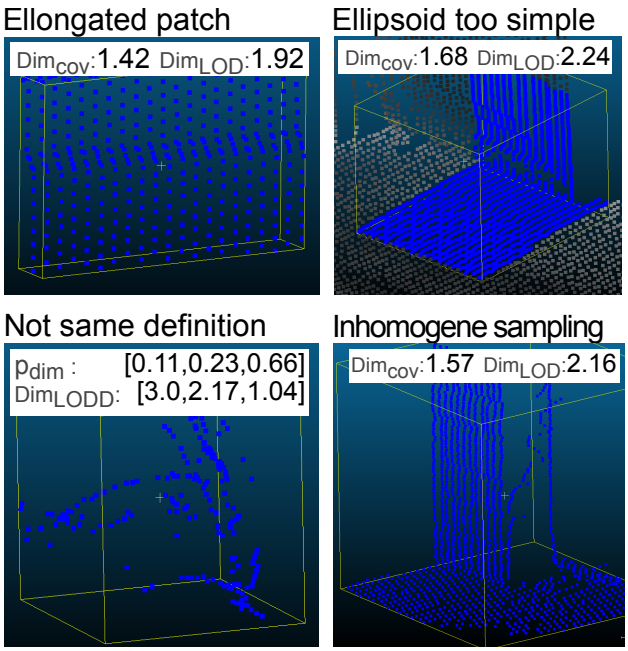


Figure 14: Representative patches for $|Dim_{LOD}\text{-}Dim_{cov} > 0.5|$. Most differences are explained by Dim_{cov} limitations (See 4.6.1.)

- Elongated patch.
  Dim_{cov}=1.42 ,Dim_{LOD}=1.92. If the patch is not roughly a square,Dim_{cov} gives a bad estimation as it is biased by the un-symmetry of point distribution.
- Ellipsoid too simple.
  Dim_{cov}=1.68, Dim_{LOD}=2.24. Dim_{cov} fits an ellipsoid, which can not cope with complex objects, especially when the barycentre does not happen to be at a favourable place.
- Coping with heterogeneous sampling.
  p_{dim}=[0.56,0.32,0.12], Dim_{cov}=1.57, Dim_{LOD}=2.16. Dim_{cov} is sensitive to difference in point density. The points on the bottom plan are much 3 times less dense than in the vertical plan, leading to a wrong estimate.
- Definition of dimension different.
  General:$Dim_{cov} \in [1.2, 2.6]$, $Dim_{LOD} \in [1.7..2.7]$
  This patch: p_{dim}=[0.11, 0.23, 0.66]
  ppl=[1,8,36,74..], Dim_{LODD}=[3.0,2.17,1.04]. Trees are a good example of how the two descriptors rely on a different dimension definition. For Dim_{cov} points may be well spread out, so usually p_{3D} is high. Yet, tree patches are also subject to density variation, and may also be elongate, which renders Dim_{cov} very variable. On the opposite, Dim_{LOD} considers the dimensionality at different scale (See Fig. 24). From afar a tree-patch is volumetric, at lower scal, it seems planar (leaf and small sticks form rough plans together). Lastly at small scale, the tree looks linear (sticks).

4.6.2 Usefulness of rough descriptor for classification Using only the *ppl* descriptor, a classification is performed on Paris data set, then a confusion matrix is computed. We use the spectral layout (see Section 3.6.2) to automatically produce the graph in Figure 15. We manually ad 1D,2D and 3D arrows. On this graph, classes that are most similar according to the classification with *ppl* are close. The graph clearly present an organisation following 3 axis. Those axis coincide with the dimensionality of the classes. For instance the "tree" classe as a strong 3D dimensionality. The "Punctual object" class, defined by "Objects which representation

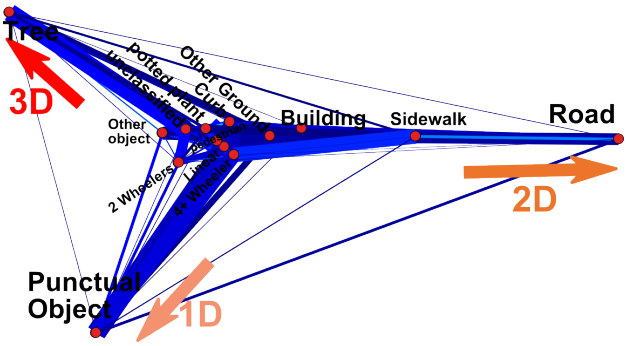


Figure 15: Spectral clustering of confusion matrix of Paris data set classification using only *ppl* descriptor. Edge width and colour are proportional to affinity. Node position is determined fully automatically. Red-ish arrows are manually placed as visual clues.

on a map should be a point", is strongly 1D (lines), with object like bollard and public light. The "Road" class is strongly 2D, the road being locally roughly a plan. The center of the graph is occupied by classes that may have mixed dimensionality, for instance "4+ wheeler" (i.e. car) may be a plan or more volumetric, because of the $1\mathrm{m}^3$ sampling. "Building" and "sidewalk" are not as clearly 2D as the "road" class. Indeed, the patch of "sidewalk" class are strongly mixed (containing 22 %of non-sidewalk points, See figure 18). The building class is also not pure 2D because building facade in Paris contains balcony, building decoration and floors, whcih introduce lots of object-like patches, which explain that building is closer to the object cluster. (See Figure 11, in the Level 3 for instance). The dimensionality descriptor clearly separates classes following their dimensionality, but can't separate classes with mixed dimensionality.

To further evaluate the dimensionality descriptor, we introduce other classification features (see 4.7), perform classification and compute each feature importance. The overall precision and recall result of these classification is correct, and the *ppl* descriptor is of significant use (See Figure 18 and 17), especially in the Vosges data set. The *ppl* descriptor is less used in Paris data set, maybe because lots of classes can not really be defined geometrically, but more with the context.

4.7 Patch Classification

4.7.1 Introducing other features The dimensionality descriptor alone cannot be used to perform sophisticated classification, because many semantically different objects have similar dimension (for instance, a piece of wall and of ground are dimensionally very similar , yet semantically very different). We introduce additional simple features for classification (See Section 3.6.2). All use already stored patch statistics, and thus are extremely fast to compute. (P : for Paris , V : for Vosges: - average of altitude regarding sensing device origin(P) - area of $patch\_bounding\_box$ (P) : - patch height (P) - $points\_per\_level$ (*ppl*), level 1 to 4 (P+V) - average of intensity (P+V) - average of $number\_of\_echo$ (P+V) - average Z (V)

For Vosges data set, we reach a speed of 1 Mpoints/s/worker to extract those features.

4.7.2 Classification Setting Undersampling and weighting are used on the paris dataset. First Undersampling to reduce the over dominant building classe to a 100 factor of the smallest class support. Then weighting is used to compensate for differences in support. For the Vosges data set only the weighting strategy is used. The weighting approach is favoured over undersampling

because it lessen variability of results when classes are very heterogeneous.

To ensure significant results we follow a K-fold cross-validation method. We again compute a confusion matrix (i.e. affinity between classes) on the Paris data set to choose which level of class hierarchy should be used. fig:class-clustering-all-features

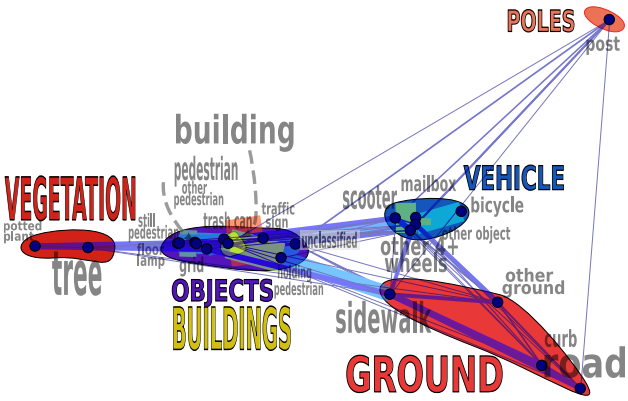


Figure 16: Result of automatic spectral clustering over confusion matrix for patch classification of Paris data set with all simple features. Edges width and colour are proportional to confusion. Manually drawn clusters for easier understanding.

4.7.3 Analysing class hierarchy Choosing which level of the class hierarchy to use depends on data set and applications. In a canonical classification perspective, we have to strongly reduce the number of classes if we want to have significant results. However reducing the number of class (i.e use a higher level in the classes hierarchy) also means that classes are more heterogeneous.

Both data set are extremely unbalanced (factor 100 or more). Thus our simple and direct Random Forest approach is ill suited for dealing with extremely small classes. (Cascading or one versus all framework would be needed).

For Vosges data set a short analysis convince us to use 3 classes: Forest, Land, and other, on this three classes, the Land class is statistically overweighted by the 2 others.

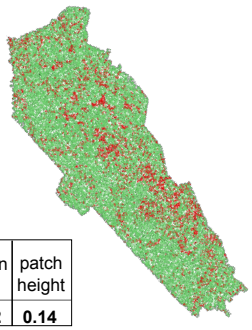
For the Paris data set, we again use a spectral layout to represent the affinity graph (See Figure 16). Introducing other features clearly helps to separate more classes. The graph also shows the limit of the classification, because some class cannot be properly defined without context (e.g. the side-walk, which is by definition the space between building and road, hence is defined contextually).

4.7.4 Classification results We perform a analysis of error on Vosges dataset and we note that errors seem to be significantly correlated to distance to borders.

The learning time is less than a minute, the predicting time is less than a second.

For both dataset, patches main contain points from several classes. We measure how much of the patch points pertain to the dominant class. The result is given in the columns "mix". For instance the patch of the class "building" contains an average of 98.6 %points of the class "building", whereas the patch from the class "forest" contains 88.3 %points of the class "forest". Therefore, to provide a comparison with point based classification, we can compute the precision of the classification per point as $Precision_{point} = Precision_{patch} * Mix.$. (Same for recall).

	prec.	rec.	supp.	mix.
Closed Forest	**0.99**	**0.91**	**390k**	**0.883**
Moor	0.18	0.68	8.7k	0.741
Not forest	**0.86**	**0.89**	**128k**	**1**
avg/total	0.94	0.90	526k	0.901



Feature usage

ppl_1	ppl_2	ppl_3	ppl_4	Intensity	nber of echo	mean Z	patch height
0.57 (0.07+0.13+0.18+0.19)				0.09	0.06	**0.12**	**0.14**

Figure 17: Vosges dataset. (table 2) Precision(prec.), recall (rec.), support (supp.), and average percent of points of the class in the patches, for comparison with point based method (mix.). (table 1)Feature usage

4.7.5 Precision or Recall increase As explained in Section 3.6.3, we can leverage the random forest confidence score to artificially increase the precision.

We focus on the building class. As seen in the Figure 19, initial classification results (blue) are mostly correct. Yet, only keeping patches with high confidence may greatly increase precision (to 100 %). Further filtering on confidence can not increase precision, but will reduce the variability of the found building patches. This result (red) would provides a much better base for building reconstruction for instance.

The patch classifier can also be used as a filtering preprocess. In this case, the goal is not to have a great precision, but to be fast and with a good recall. Such recall may be increased artificially for class of objects bigger than the sampling size ($1\mathrm{m}^3$ for Paris).

We take the example of ground classification (See Figure 20). The goal is to find all ground patches very fast. We focus on a small area for illustration purpose. This area contains 3086 patches, including 439 ground patches. Patch classification finds 421 ground patch, with a recall of 92.7%. Using the found patch, all the surrounding patches (X,Y : 2 m, Z : 0.5 m) are added to the result (few seconds). There are now 652 patches in the result, and the recall is 100%. This means that from a filtering point of view, a complex classifier that would try to find ground points can be used on $652/3086 = 21\%$ of the data set, at the price of few seconds of computing, without any loss of information.

5. DISCUSSION

5.1 Point cloud server

We refer the reader to Cura (2014) for an exhaustive analyse of the Point Cloud Server. Briefly, the PCS has demonstrated all the required capacitites to manage point clouds and scale well. To the best of our knowledge the fastest and easiest way to filter very big point cloud using complex spatial and temporal criteria, as well as natively integrate point cloud with other GIS data (raster, vector). The main limitation is based on the hypothesis than points can be regrouped into meaningful (regarding further point utilisation) patches. If this hypothesis is false, the PCS lose most of its interest.

5.2 Exploiting the order of points

From a practical point of view, implicitly storing the LOD using the point ordering seems to be estremely portable. Most softwares would not change the order of points. For those who might change

	prec.	rec.	supp.	mix.
unclassified	0.13	0.12	104	*0.858*
other ground	0.21	0.64	73	*0.704*
road	**0.92**	**0.82**	**1278**	***0.940***
sidewalk	0.75	0.71	815	*0.774*
curb	0.23	0.59	37	*0.717*
building	**0.9**	**0.75**	**1771**	***0.986***
other object	0.34	0.78	102	*0.759*
post	0.71	0.95	91	*0.968*
floor lamp	0.19	0.7	20	*0.982*
traffic sign	0.07	0.22	9	*0.789*
mailbox	0	0	2	*1.000*
trash can	0.1	0.36	11	*0.782*
grid	0.17	0.59	41	*0.861*
pedestrian	0	0	3	*0.961*
other pedestrian	0.23	0.52	21	*0.787*
still pedestrian	0.04	0.11	9	*0.963*
holding pedestrian	0	0	4	*0.721*
bicycle	0.11	0.33	3	*1.000*
scooter	0.28	0.58	76	*0.925*
other 4+ wheels	0.73	0.43	462	*0.938*
tree	**0.93**	**0.92**	**2803**	***0.991***
potted plant	0.32	0.9	10	*0.973*
avg/total	0.84	0.78	7745	*0.953*

	prec.	rec.	supp.
unclassified	0.13	0.13	104
other ground	0.21	0.58	73
road	**0.91**	**0.82**	**1278**
sidewalk	0.74	0.71	815
curb	0.24	0.62	37
building	**0.89**	**0.74**	**1836**
other object	0.31	0.82	102
punctual object	0.64	0.76	133
linear	0.15	0.59	41
pedestrian	0.23	0.76	37
2 wheelers	0.3	0.65	79
4+ wheelers	0.69	0.45	462
tree	**0.93**	**0.92**	**2803**
potted plant	0.33	0.8	10
avg/total	0.84	0.78	7810

	prec.	rec.	supp.
unclassified	0.11	0.13	104
ground	**0.95**	**0.94**	**2203**
building	**0.91**	**0.78**	**1907**
other object	0.3	0.84	102
static	0.49	0.72	174
Dynamic	0.65	0.71	578
natural	**0.94**	**0.92**	**2813**
avg/total	0.88	0.86	7881

feature usage

ppl_1	ppl_2	ppl_3	ppl_4	intensity	nber of echo	height above ground	patch height	patch area
0.17 (0.03+0.05+0.05+0.04)				**0.21**	0.11	**0.29**	0.1	0.12
0.16 (0.02+0.04+0.05+0.05)				**0.2**	0.1	**0.3**	0.11	0.12
0.21 (0.03+0.05+0.06+0.07)				0.11	0.08	**0.37**	0.09	**0.15**

Figure 18: Results for Paris data set: at various level of class hierarchy. Precision(prec.), recall (rec.), support (sup.) and average percent of points of the class in the patches of the class, for comparison with point based method (mix.). Classes of the same type are in the same continuous tone. Feature usage is evaluated for each level in the class hierarchy.

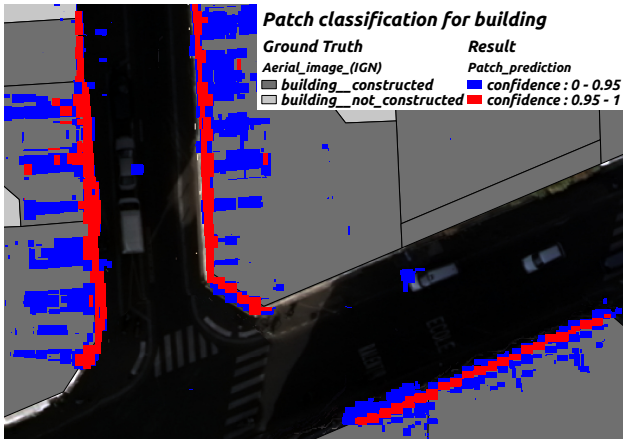


Figure 19: Plotting of patches classified as building, using confidence to increase precision. Ground truth from IGN and Open Data Paris

the order of points, it is still very easy to add the order as an attribute, thus making it fully portable. However, this approach has two limitations. The first limitation is that the order of point might already contains precious information. For instance with a static Lidar device, the acquisition order allows to reconstruct neighbourhood information. The second limitation is that an LOD ordering might conflict with compression. Indeed ordering the points to form LOD will create a list of points were successive points are very different. Yet compressing works by exploiting similarities. A software like LasTool using delta compressing might suffer heavily from this.

5.3 MidOc : an ordering for gradual geometrical approximation

We stress that the LOD are in fact almost continuous (as in the third illustrations of Fig. 1).

MidOc is a way to order points based on their importance. In MidOc, the importance is defined geometrically. Yet specific applications may benefit from using other measure of importance, possibly using other attributes than geometrical one, and possibly using more perceptual-oriented measures.

MidOc relies on two hypothesis which might be false in some case. Indeed, variation of density may be a wanted feature (e.g. stereovision, with more image on more important parts of the object being sense). Again the low geometrical noise hypothesis might be true for Lidar, but not for Stereo vision or medical imaging point cloud. However in those case denoising methods may be applied before computing MidOc.

5.3.1 Applications MidOc ordering might be of use in 3 types of applications. First it can be used for graphical LOD, as a service for point cloud visualisation. Second the ordering allows to correct density to be much more constant. Complex processing methods

Figure 20: Map of patch clustering result for ground. The classical result finds few extra patches that are not ground (blue), and misses some ground patches (red). Recall is increased by adding to the ground class all patches that are less than 2 meters in X,Y and 0.5 meter in Z around the found patches. Extra patches are much more numerous, but all the ground patches are found.

may benefits from an almost constant density, or for the absence of strong density variation. Third the ordering can be used for point cloud generalisation, as a service for processing methods that may only be able to deal with a fraction of the points.

The illustration 11 gives visual example of LOD result and how it could be used to vary density depending on the distance to camera. Figure 9 also gives visual examples for common objects of different dimensionality. It is visually clear that the rate of increase of points from LOD 0 to 4 for floor lamp (1D) window (2D) and tree (3D) is very different. Small details are also preserved like the poles or the antenna of the car. preserving those detail with random or distance based subsampling would be difficult.

5.3.2 Implementation Octree construction may be avoided by simply reading coordinates bitwise in a correctly centred/scaled point cloud. We centre a point cloud so that the lowest point of all dimension is $(0, 0, 0)$, and scale it so that the biggest dimension is in $[0, 1[$. The point cloud is then quantized into $[0..2 * *L - 1]$ for each coordinate. The coordinates are now integers, and for each point, reading its coordinates bitwise left to right gives the position of the point in the octree for level of the bit read. This means performing this centring/scaling/quantization directly gives the octree. Moreover, further operations can be performed using bit arithmetic, which is extremely fast.

On this illustration the point P has coordinates $(5, 2)$ in a $[0, 2^3 - 1]^2$ system. Reading the coordinates as binary gives $(b'101', b'010')$.

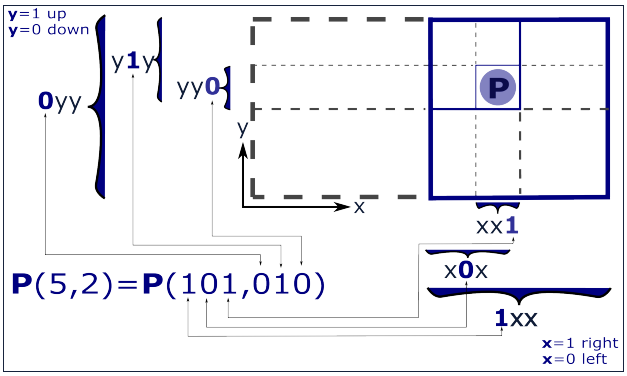


Figure 21: Principle of binary coordinates for a centered, scaled and quantized point cloud.

Thus we know that on the first level of a quad tree, P will be in the right (x=$b'1xx'$) bottom (y=$b'0yy'$) cell. For the next level, we divide the previous cell in 2, and read the next binary coordinate. P will be in the left (x=$b'x0x'$) up (y=$b'y1y'$) cell. There is no computing required, only bit mask and data reading.

Regarding implementation, the three we propose are much too slow, by an order of magnitude to be easily used in real situation. We stress however that the slowness comes from ineficient data manipulation, rather than from the complexity of the ordering. It may also be possible to use the revert Hilbert ordering to directly compute MidOc. Furthermore, octree construction has been commonly done on GPU for more than a decade.

5.3.3 Size versus LOD trade-off The table 1 shows that using the level 3 may speed the transfer time by a 10 factor. The point cloud server throughput is about 2-3 $Mbyte$ /s(monoprocess), sufficient for an internet troughput, but not fast enough for a LAN 10 $Mbyte$ /s. This relatively slow troughput is due to current point cloud server limitation (cf 5.1).

5.3.4 Large scale computing The relatively slow computing (180 Millions points /h) is a very strong limitation. This could be avoided. A C implementation which can access raw patch would also be faster for ordering points.

5.3.5 LOD stream Streaming low level of detail patches greatly accelerate visualisation, which is very useful when the full point cloud is not needed. To further accelerate transmission, patch LOD can be determined according to the distance to camera (frustrum culling). (See Figure 22 for a naive visual explanation.)
As seen before (See Section 5.3.3), the point cloud server is fast enough for an internet connection, but is currently slower than a file-based points streaming. Thus for the moment LOD stream is interesting only when bandwidth is limited.

5.4 Excessive Density detection and correction

5.4.1 Fast detection Density abnormality detection at the patch level offer the great advantage of avoiding to read points. This is the key to the speed of this method. We don't know any method that is as fast and simple.
The limitations stems from the aggregated nature of patch. the number of points per patch doesn't give the density per point, but a quantized version of this per patch. So it is not possible to have a fine per point density.

5.4.2 Simple correction The correction of density peak we propose has the advantage of being instantaneous and not induce

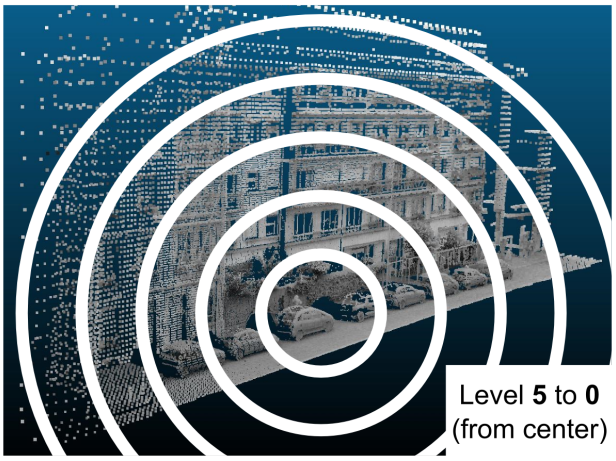


Figure 22: Schematique example of LOD depending on distance to camera

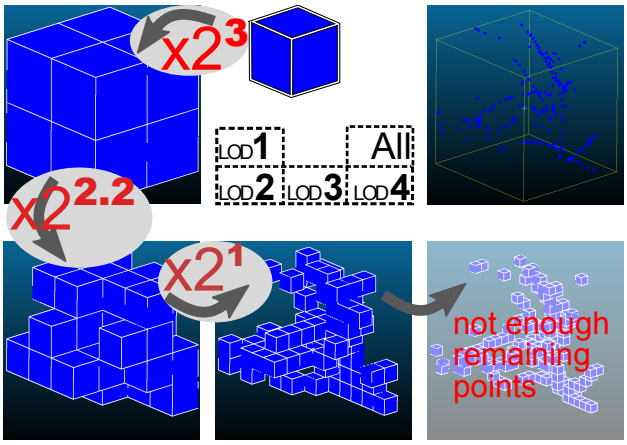


Figure 24: Evolution of tree patch octree cells occupancy, illustrating the various dimensions of trees depending on the scale. (Dimension is embeded it the power of 2).

any data loss. It is also easy to use as safeguard for an application that may be sensible to density peak : the application simply defines the highest number of points $/\mathrm{m}^3$it can handle, and the Point cloud server will always output less than that.
The most important limitation this method doesn't guarantee homogeneous density, only a maximum density. For instance if an application requires 1000 points $/\mathrm{m}^3$for ground patches, all the patches must have more than 1000 points, and patch must have been ordered with MidOc for level 0 to at least 5 ($4^5 = 1024$). The homogeneous density may also be compromised when the patch are not only split spatially, but with other logics (in our case, points in patch can not be separated by more than 30 seconds, and all points in a patch must come from the same original acquisition file).

5.5 Crude dimensionality descriptor (MidOc by-product)

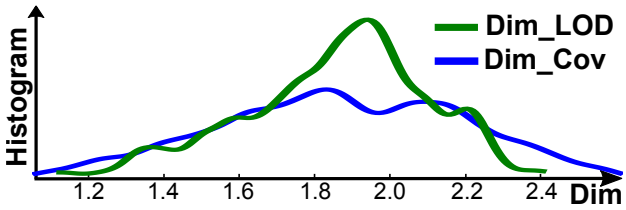


Figure 23: Histogram of Dim_{LOD} and Dim_{cov} for patch in trees (500 kpts). Tree dimension could be from 1.2 to 2.6, yet Dim_{LOD} is less ambiguous than Dim_{cov}

Tree patches are challenging for both dimensionality descriptor. There possible dimension changes a lot (See Fig. 23), although Dim_{LOD} is more concentrated. Yet, *ppl* is extremely useful to classify trees. Indeed, *ppl* contains the dimensionality at various scale, and potentially the variation of it, which is quite specific for trees (See Fig. 24).

We stress that a true octree cell occupancy (i.e. without picking points as in the *ppl*) can be obtained without computing the octree, simply by using the binary coordinates (See 21). We implement it in python as a proof of concept. Computing it is about as fast as computing Dim_{cov}.

Overall, *ppl* offers a good alternative to the classical dimensionality descriptor (Dim_{cov}), being more robust and multiscale. However the *ppl* also has limitations. First the quality of the dimensionality description may be affected by a low number of points in the patch. Second in some case it is hard to reduce it to a meaningful Dim_{LOD}]. Last because of assumption on density, it is sensible to geometric noise.

5.6 Patch Classification

The *ppl* descriptor contains lots of information about the patch. This information is leveraged by the Random Forest method and permit a rough classification based on geometric differences. As expected, *ppl* descriptor are not sufficient to correctly separate complex objects, which is the main limitation for a classification application.

The additional features are extremely simple, and far from the one used in state of the art. Notably, we don't use any contextual feature. We choose to classify directly in N classes, whereas due to the large unbalance indataset, cacade or 1 versus all approaches would be more adapted.

5.6.1 Analysing class hierarchy The figure 16 shows the limit of a classification without contextual information. For instance the class grid and buildings are similar because in Paris buildings balcony are typically made of grids.

To better identify confusion between classes, we use a spectral layout on the affinity matrix. Graphing this matrix in 2D ammount to a problem of dimensionality reduction. It could use more advanced method than simply using the first two eigen vector, in perticular the two vector wouldn't need to be orthogonal (for instance, like in Hyvärinen and Oja (2000)).

5.6.2 Classification results First the feature usage for vosges data set clearly shows that amongst all the simple descriptor, the *ppl* descriptor is largely favored. This may be explained by the fact that forest and bare land have very different dimensionality, which is conveyed by the *ppl* descriptor.

Second the patch classifier appears to have very good result to predict if a patch is forest or not. The precision is almost perfect for forest. We reach the limit of precision of ground truth. Because most of the errors are on border area, the recall for forest can also be easily artificially increased. The percent of points in patch that are in the patch class allow to compare result with a point based method. For instance the average precision per point for closed forest would be $0.99 * 0.883 = 0.874$. We stress that this is averaged results, and better precision per point could be

achieved because we may use random forest confidence to guess border area (with a separate learning for instance). For comparison with point methods, the patch classifier predict with very good precision and support over 6 billions points in few seconds (few minutes for training). We don't know other method that have similar result while being as fast and natively 3D. The Moor class can't be separated without more specialised descriptor, because Moor and no forest classes are geometrically very similar.

The principal limitation is that for this kind of aerial Lidar data set the 2.5D approximation may be sufficient, which enables many raster based methods that may perform better or faster.

The figure 18 gives full results for paris data set, at various class hierarchy level. Because the goal is filtering and not pure classification, we only comment the 7 classes result. The proposed methods appears very effective to find building, ground and trees. Even taking into consideration the fact that patch may contains mixed classes (column mix.), the result are in the range of state of the art point classifier, while being extremely fast. This result are sufficient to increase recall or precision to 1 if necessary. We stress that even results appearing less good (4+wheelers , 0.69 precision, 0.45 recall) are in fact sufficient to increase recall to 1 (by spatial dilatation of the result), which enables then to use more subtle methods on filtered patches.

ppl descriptor is less used than for the Vosges data set, but is still useful, particularly when there are few classes. It is interesting to note that the mean intensity descriptor seems to be used to distinguish between objects, which makes it less useful in the 7 classes case. The patch classifier for Paris data set is clearly limited to separate simple classes. In particular, the performances for objects are clearly lower than the state of the art. A dedicated approach should be used (cascaded or one versus all classifier).

5.6.3 Estimating the speed and performance of patch based classification compared to point based classification The Point Cloud Server is designed to work on patches, which in turns enable massive scaling.

Timing a server is difficult because of different layer of caches, and background workers. Therefore, timing should be considered as order of magnitude. For Paris data set,extracting extra classification features requires $\sim \frac{400\text{s}}{n_{workers}}$(1 to 8 workers), learning $\sim$ 210s, and classification $\sim$ few s. We refer to Weinmann et al. (2015)(Table 5) for point classification timing on the same dataset (4.28h, 2s, 90s) (please note that the training set is much reduced by data balancing). As expected the speed gain is high for complex feature computing (not required) and point classification (done on patch and not points in our case).

For Vosges data set, features are extracted at $1\text{M}pts/\text{s}/worker$, learning $\sim few\text{min}$, classification $\sim$ 10s. The Vosges data set has not been used in other articles, therefore we propose to compare the timings to Shapovalov et al. (2010) (Table 3). Keeping only feature computation and random forest training (again on a reduced data set), they process 2 Mpoints in 2 min, whereas our method process the equivalent of 5.5 B points in few minutes.

Learning and classification are monothreaded (3 folds validation), although the latter is easy to parallelise. Overall, the proposed method is one to three orders of magnitude faster.

For Paris data set (Fig. 18), we compare to Weinmann et al. (2015)(Table 5). As expected there results are better, particularly in terms of precision (except for the class of vegetation). This trend is aggravated by taking into account the "mix." factor. Indeed we perform patch classification, and patch may not pertain to only one class, which is measured by the mix factor (amount of points in the main class divided by the total number of point). However, including the mix factor the results are still within the 85 to 90 % precision for the main classes (ground, building, natural).

For Vosges data set (Fig 17), we refer to Shapovalov et al. (2010) (Table 2). There random forest classifier get trees with 93% precision and 89% recall. Including the mix factor we get trees with a precision of 87% and 80% recall. As a comparison to image based classification, an informal experiment of classification with satellite image reaches between 85 % and 93 % of precision for forest class depending on the pixel size (between 5 and 0.5 m).

Overall, the proposed method get reasonably good results compared to more sophisticated methods, while being much faster. It so makes a good candidate as a preprocessing filtering step.

5.6.4 Precision or Recall increase Because the propose methods are preprocess of filtering step, it can be advantageous to increase precision or recall.

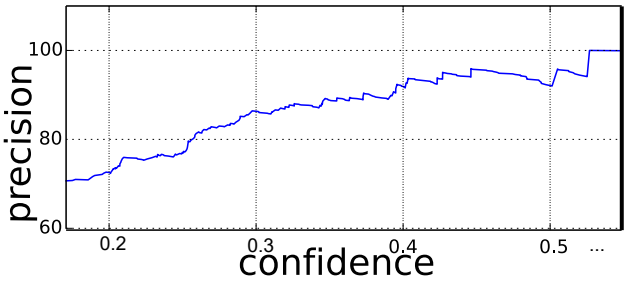


Figure 25: Precision of 4+wheelers class is a roughly rising function of random forest confidence scores.

In the Figure 19 gives a visual example where increasing precision and reducing class heterogeneity is advantageous. This illustrates that having a 1 precision or recall is not necessary the ultimate goal. In this case it would be much easier to perform line detection starting from red patches rather than blue patches.
The limitation of artificial precision increase is that it is only possible when precision is roughly a rising function of random forest confidence, as seen on the illustration 25. For this class, by accepting only prediction of random forest that have a confidence over 0.3 the precision goes from 0.68 to 0.86, at the cost of ignoring half the predictions for this class. This method necessitates that the precision is roughly a rising function of the confidence, as for the 4+wheeler class for instance (See Figure 25). This strategy is not possible for incoherent class, like unclassified class.

The method we present for artificial recall increase is only possible if at least one patch of each object is retrieved, and objects are spatially dense. This is because a spatial dilatation operation is used. This is the case for "4+wheelers" objects in the paris data set for instance. The whole method is possible because spatial dilatation is very fast in point cloud server (because of index). Moreover, because the global goal is to find all patches of a class while leaving out some patches, it would be senseless to dilate with a very big distance. In such case recall would be 1, but all patches would be in the result, thus there would be no filtering, and no speeding.
The limitation is that this recall increase method is more like a deformation of already correct results rather than a magical method that will work with all classes.

6. CONCLUSION

Using the Point Cloud Server, we propose a new paradigm by separating the spatial indexing and LOD scheme. Subdivision of

point clouds into groups of points (patches) allows us to implicitly store LOD into the order of points rather than externally. After an ordering step, exploiting this LOD does not require any further computation. We propose an geometrical ordering (MidOc) based on the closest point to octree cell centre that produces reliable LOD, successfully used for visualization or as a service for other processing methods (density correction/reduction). By also performing intra-level dedicated ordering, we create LOD that can be used partially and still provide good coverage. Furthermore, by collecting the number of points per octree level, an information available during MidOc ordering, we create a multi-scale dimensionality descriptor. We show the interest of this descriptor, both by comparison to the state of the art and by proof of its usefulness in real Lidar dataset classification. Classification is extremely fast, sometime at the price of performance (precision / recall). However we prove that those results can be used as a pre-processing step for more complex methods, using if necessary precision-increase or recall-increase strategies.

References

Amit, Y. and Geman, D., 1997. Shape quantization and recognition with randomized trees. Neural Comput. 9, pp. 1545–1588. 7

Baert, J., Lagae, A. and Dutré, P., 2014. Out-of-Core Construction of Sparse Voxel Octrees. In: Computer Graphics Forum, Vol. 33, Wiley Online Library, pp. 220–227. 2

Bereuter, P., 2015. Quadtree-based Real-time Point Generalisation for Web and Mobile Mapping. PhD thesis, Mathematisch-naturwissenschaftlichen Fakultät der Universität Zürich, Zurich. 2

Breiman, L., 2001. Random Forests. In: Machine Learning, pp. 5–32. 7

Bustos, B., Keim, D. A., Saupe, D., Schreck, T. and Vranić, D. V., 2005. Feature-based similarity search in 3D object databases. ACM Computing Surveys 37(4), pp. 345–387. 5

Choi, S., Kim, T. and Yu, W., 2009. Performance evaluation of ransac family. In: Proceedings of the British Machine Vision Conference. 9

Cura, R., 2014. A PostgreSQL Server for Point Cloud Storage and Processing. 8, 12

Demantké, J., 2014. Reconstruction of photorealistic 3D models of facades from terrestrial images and laser data. PhD thesis, Paris Est, Paris. 6

Elseberg, J., Borrmann, D. and 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, pp. 76–88. 2

Feng, J. and Watanabe, T., 2014. Index and Query Methods in Road Networks. Springer. 1, 5

Feng, J., Xu, H. and Yan, S., 2013. Online robust pca via stochastic optimization. In: Advances in Neural Information Processing Systems, pp. 404–412. 5

Fornasier, M. and Rauhut, H., 2010. Compressive sensing. Handbook of Mathematical Methods in Imaging 1, pp. 187–229. 3

Frey, B. J. and Dueck, D., 2007. Clustering by passing messages between data points. science 315(5814), pp. 972–976. 8

Girardeau-Montaut, D., 2014. CloudCompare. 5

Golovinskiy, A., Kim, V. G. and Funkhouser, T., 2009. Shape-based recognition of 3d point clouds in urban environments. In: Computer Vision, 2009 IEEE 12th International Conference on, pp. 2154–2161. 7

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. and Burgard, W., 2013. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots 34(3), pp. 189–206. 2

Huang, Y., Peng, J., Kuo, C.-C. J. and Gopi, M., 2006. Octree-based progressive geometry coding of point clouds. In: Proceedings of the 3rd Eurographics/IEEE VGTC conference on Point-Based Graphics, Eurographics Association, pp. 103–110. 2

Hyvärinen, A. and Oja, E., 2000. Independent component analysis: algorithms and applications. Neural networks 13(4), pp. 411–430. 15

IGN, 2014. ITowns. 9

IQmulus, 2014. IQmulus & TerraMobilita Contest. 8

McKay, M. D., Beckman, R. J. and Conover, W. J., 1979. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. Technometrics 21(2), pp. 239. 3

Meagher, D., 1982. Geometric modeling using octree encoding. Comput. Graph. Image Process. 19(2), pp. 129–147. 1

Networkx, d. t., 2014. Networkx. 8

pgPointCloud, R., 2014. pgPointCloud. 8

PostGIS, d. t., 2014. PostGIS. 8

PostgreSQL, d. t., 2014. PostgreSQL. 8

Rainville, D., Gagné, C., Teytaud, O., Laurendeau, D. and others, 2012. Evolutionary optimization of low-discrepancy sequences. ACM Transactions on Modeling and Computer Simulation (TOMACS) 22(2), pp. 9. 4, 5

Rusinkiewicz, S. and Levoy, M., 2000. QSplat: A multiresolution point rendering system for large meshes. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., pp. 343–352. 1

Sabo, N., Beaulieu, A., Bélanger, D., Belzile, Y. and Piché, B., 2014. The GeoHashTree: a multi-resolution data structure for the management of point clouds. Technical notes 4, canada. 2

Schnabel, R. and Klein, R., 2006. Octree-based Point-Cloud Compression. In: SPBG, pp. 111–120. 2

Schütz, M. and Wimmer, M., 2015. High-Quality Point Based Rendering Using Fast Single Pass Interpolation. IEEE. 2

Schwartges, N., Allerkamp, D., Haunert, J.-H. and Wolff, A., 2013. Optimizing Active Ranges for Point Selection in Dynamic Maps. In: Proceedings of the 16th ICA Generalisation Workshop (ICA'13), Dresden. 1

Scikit, d. t., 2014. scikit-image: image processing in Python. PeerJ 2, pp. e453. 8

SciPy, d. t., 2014. SciPy: Open source scientific tools for Python. 8, 9

Serna, A. and Marcotegui, B., 2014. Detection, segmentation and classification of 3D urban objects using mathematical morphology and supervised learning. ISPRS J. Photogramm. Remote Sens. p. 34. 7

Sester, M., 2001. Kohonen Feature Nets for Typification. 1

Shapovalov, R., Velizhev, A. and Barinova, O., 2010. Non-associative markov networks for 3D point cloud classification. In: Photogrammetric Computer Vision and Image Analysis (PCV 2010), Vol. 38, pp. 103–108. 16

Shlens, J., 2014. A tutorial on principal component analysis. arXiv preprint arXiv:1404.1100. 6

Weinmann, M., Urban, S., Hinz, S., Jutzi, B. and Mallet, C., 2015. Distinctive 2D and 3D features for automated large-scale scene analysis in urban areas. Computers & Graphics 49, pp. 47–57. 6, 9, 16

Wu, J., 2011. Improving the writing of research papers: IMRAD and beyond. Landscape Ecology 26(10), pp. 1345 – 1349. 3